National Aeronautics and Space Administration








Critical Software:

Good Design Built Right

Leadership ViTS Meeting

February 2012

Terry Wilcutt

Chief, Safety and Mission Assurance

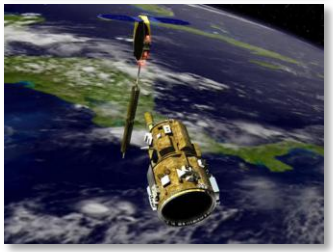
Wilson B. Harkins

Deputy Chief, Safety and Mission Assurance

This and previous presentations are archived at:
nsc.nasa.gov/articles/SFCS

NASA Mishaps Involving Software

STS-126 Software Anomaly: After the Space Shuttle Endeavour reached orbit on 2008, shuttle-to-ground S-band communications failed to automatically switch to the more powerful Ka-band antenna required on-orbit. Then, the Payload Signal Processor (PSP) failed to automatically switch from its wireless RF link to its hardwired umbilical cable. Mission control was able to manually command both switchovers without mission impact. Investigation found that a software change inadvertently shifted data in the shuttle's flight software code, causing several automated functions to fail.

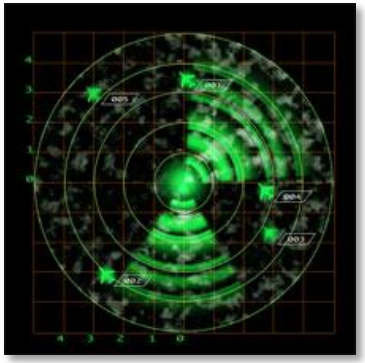


DART Autonomous Rendezvous Failure: In 2005, the DART (Demonstration for Autonomous Rendezvous Technology) spacecraft was to autonomously rendezvous with the MUBLCOM (MUtiple-path Beyond Line-of-sight COMmunications) satellite. Approximately 11 hours into the mission, DART collided with MUBLCOM, ran out of fuel, and prematurely transitioned to retirement sequence without accomplishing its goals. The collision occurred because DART's navigational control failed to transition to the Advanced Video Guidance Sensor (AVGS) to calculate its velocity and position relative to MUBLCOM. The premature retirement of DART occurred due to recurring computational resets of its estimated position. DART's thrusters fired more often than planned following the resets. An incorrect velocity measurement was introduced into the software during each reset. This algorithm defect had been discovered prior to the mission but no software change was implemented.

SOHO Communication Loss: The SOHO (Solar Heliospheric Observatory) spacecraft was launched in 1995 to study the Sun from its deep core to the outer corona and the solar wind. In 1997, software modifications were uploaded to conserve gyroscope operation. Then in 1998, another software modification caused SOHO to lose lock on the Sun and sent the spacecraft into an emergency attitude control mode. During recalibration, the spacecraft spun out of control and all communications were lost. Ground operators found that critical errors in the code modified to conserve gyro usage configured the gyros incorrectly, and caused inaccurate thruster firings which progressively destabilized the spacecraft. Five months of project team work produced a software workaround that recovered control of SOHO.

Mars Pathfinder's Unexpected System Resets: In 1997, the Mars Pathfinder returned an unprecedented amount of data about Mars. But a few days into the mission, the lander's onboard computer repeatedly reset itself, slowing the flow of research data to Earth. A timing defect in software application code caused the computers to continually reboot. A small low-priority task was unable to complete its function during data flow to Earth when all the high-priority tasks were moving at high rates. A fail-safe mechanism in the software, which resets the system automatically when any performance is interrupted, rebooted the system when the glitch occurred. NASA solved the problem by raising the priority of the task involved and adjusting the priority of other tasking in the code.

Non-NASA Software Mishaps



LA Center Air Traffic Control Radio Loss: On 2004, the Los Angeles Air Route Traffic Control Center lost all radio communication with over 400 aircraft over the southwestern United States when their main voice communications system shut down without warning. The incident disrupted almost 800 flights across the country. A function defect was found in software upgrade to a subsystem for the Voice Switching and Control System (VSCS). The upgrade used a timer to send built-in test queries to the VSCS. The upgrade's test counter counted tests down from the highest number that the system's server and software could handle, counting to zero or until reset. Procedures required a technician to reset the voice switching system every 30 days. The upgrade had a design defect—it was programmed to shut down the VSCS without warning if the reset had not been done, which it did when the counter reached zero after 49 days. The FAA later implemented a software patch that periodically reset the counter without human intervention.

The Great Northeast Blackout: On 2003, high-voltage power lines in northern Ohio brushed against some overgrown trees and shorted out. This triggered a distribution system load imbalance and cascading power outage throughout parts of the Northeastern and Midwestern United States and Ontario, Canada, affecting over 55 million people for two days. Due to programming errors, the alarms failed to notify operators of the faults occurring in the system. Without the knowledge of line failure, operators could not act to contain the failure. A "race condition" or software timing defect was found to be the primary cause of the grid event alarm failure. After the alarm system failed silently, the unprocessed events started to queue up and crashed the primary server within 30 minutes. This triggered an automatic transfer of all applications, including the stalled alarm system, from the primary to the backup server, which likewise became overloaded and failed. Hardware, software and procedural improvements followed to prevent recurrence.

MV-22 Osprey Crash: On a routine training mission on December 2000, an MV-22 Osprey carrying four Marines crashed in a wooded area north of Jacksonville, killing all four on-board. It was found that the mishap was caused by a burst hydraulic line in one of the Osprey's two engine casings, coupled with a software defect. The software flaw caused the aircraft to accelerate and decelerate unpredictably and violently when the pilots tried to compensate for the hydraulic failure. The Marine Corps report called for a redesign of both the hydraulics and software systems involved.

Mishaps in Other Industries (cont.)

Korean Air Flight 801 Crash: On August 6, 1997, Korean Air Flight 801 approached for landing at the Guam International Airport with heavy rain and reduced visibility. Due to disabled glideslope equipment for the available runway, the crew was required to monitor altitude via altimeter when the runway was in sight. The aircraft struck a hill about 3 miles short of the runway, at an altitude of 660 feet. 228 were killed. A ground warning system could have alerted controllers to the unsafe descent, but its software had been changed by the FAA to limit spurious alerts. The build defect rendered the system "almost completely useless," preventing approach controllers from warning Flight 801 of its premature descent into the hill.

Ariane 5 Launch Failure: Ariane 5 was an expandable launch system used to deliver payloads into geostationary transfer orbit or low Earth orbit. On 1996, its maiden flight ended in failure, with the rocket veering off its flight path and self-destructing at about 40 seconds after initiation of the flight sequence. The failure was caused by complete loss of trajectory guidance due to malfunction in the control software. Software to align the Inertial Reference System (IRS) had been reused from the Ariane-4 system, but behaved differently in the Ariane-5. Failure in converting 64-bit floating-point number to a 16-bit signed integer caused an overflow condition. The assignment defect shut down the primary inertial reference system; when control was passed to the identical secondary inertial, it predictably suffered the same fate. Ironically, the code containing the error had been designed for an Ariane-4 launch requirement not shared by Ariane-5 and could have been eliminated. The onboard computer misinterpreted diagnostic data as proper flight data and commanded an abrupt maneuver that ripped the boosters from the launch vehicle and activated the rocket's self-destruct mechanism.

Patriot Missile Tracking Failure: On February 25, 1991, during the Gulf War, an American Patriot Missile battery in Dharan, Saudi Arabia, failed to track and intercept an incoming Iraqi Scud missile, killing 28 soldiers. The system contained a software flaw in the control computer. This function defect led to inaccurate tracking calculations that worsened over time of operation. At the time of the fatal Scud attack, over 100 continuous hours of operation increased the error to the degree that the Patriot system could not track or intercept the threat. Two weeks before the incident, Army officials received Israeli data reporting loss in accuracy after 8 consecutive hours of operation. While the Army modified the software to correct the defect, the software patch did not reach Dhahran until the day after the Scud attack.

FOR FUTURE NASA MISSIONS

Does the system software **behave as expected**?
Does the system software **not** do what it's not supposed to do?
Does the system software behave as expected **under adverse conditions**?

- A system driven validation and verification approach effectively stresses the system, testing an integrated system under various anomalous conditions as early in development as possible to understand and generate evidence per the three questions above (applies to all examples).

- Dedicate resources toward validating software requirements—ensure the design is right; for changes made after technical baselines are achieved, then a revalidation from a systems perspective is warranted (all examples).

- Engineering designs should handle faults intelligently---so that persistent faults of the same type cannot bring down a system (MV-22 Osprey Crash).

- Workarounds stemming from software design shortfalls should be unacceptable to designers. The ability to identify when a workaround impacts critical system aspects demands a systems perspective. (LA Center, SOHO, Mars Pathfinder)

FOR FUTURE NASA MISSIONS (cont.)

To ensure good design and implementation, a proven, successful option available to engineers is to run a software verification and validation (V&V) project in parallel with software development.

• How much and what kind of testing is enough? Easy to say that extensive (expensive) testing should be performed, but it is realistic to put forth a robust testing strategy to adequately stress the system under adverse conditions, at every level, from unit through system test, using authentic operational and exception scenarios. (DART)

• When a software design is reused in a different system, revalidate design and test operation. A systems understanding and a systems verification and validation effort can mitigate risks of reusing (or salvaging) previously built designs (Ariane 5)

• Any changes, big or small, made to the software must be properly evaluated, assessed and documented, especially changes to safety-critical software affecting overall system performance or controlling life safety. Treat change with the same attention as original development. Good decisions about change are informed decisions; show project managers the negative impacts on the system, and they can make well-informed risk decisions. Decision makers need the "system impact" perspective but do not always receive it. (STS-126)