Historical Aerospace Software Errors Categorized to Influence Fault Tolerance

Spacecraft Anomalies and Failures Workshop
March 2024

Lorraine Prokop, Ph.D.

Technical Fellow for Software
NASA Engineering and Safety Center
lorraine.e.prokop@nasa.gov

Flight Software Error Visualization

Flight Computer *without* Software Errors
(Credit NASA)

Flight Computer *with* Software Errors
(Credit NASA)

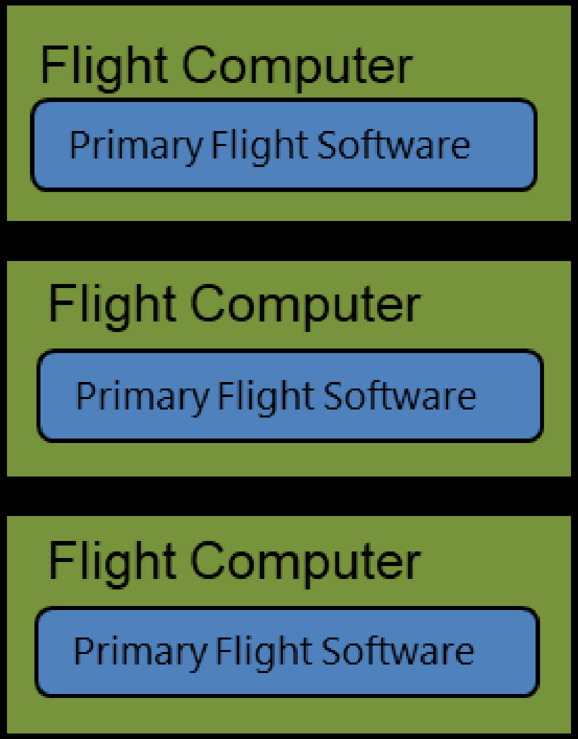
Introduction

- Motivation
  - Very little literature exists characterizing software errors in real-time avionic systems
    - *How, where, and why is software* most likely to fail?
- Purpose
  - Raise awareness of how software fails through historical study
  - Recommend improvements to software fault tolerant design based on historical study
- Outline
  - Discuss Software Failures - Common Cause, Failure Classes, Mitigation strategies
  - Review NASA Human Rating Requirements regarding software/automation
  - Review Historical Software Failures
  - Analyze failures and provide statistics
    - Erroneous vs. fail-Silent
    - Reboot recoverability likelihood
    - Code location
    - Missing code?
    - Unknown unknowns
    - Computer science related?

Software Common-Cause Failure

- What is Software "Common Cause" or "Common Mode" Failure?
  - In many avionic architectures, hardware replication into multiple "strings" is done for hardware fault tolerance
  - However, the *same software load* often runs on these multiple processors
    - In this case, a *single* software failure normally would affect all strings in the same way at the same time
    - If only one processor is used, then *any* software failure could be considered "common mode" or "common cause"

Sample Avionic Architecture



NASA Requirements for Software Fault Tolerance

- NPR 8705.2C: HUMAN-RATING REQUIREMENTS FOR SPACE SYSTEMS
  - 3.2.7 The space system shall provide the capability to mitigate the hazardous behavior of critical software where the hazardous behavior would result in a catastrophic event. The software system will be designed, developed, and tested to:

Pre-flight
1) Prevent hazardous software behavior.
2) Reduce the likelihood of hazardous software behavior.

In-flight
3) Mitigate the negative effects of hazardous software behavior. However, for complex software systems, it is very difficult to definitively prove the absence of hazardous behavior. Therefore, the crewed system has the capability to mitigate this hazardous behavior if it occurs. The mitigation strategy will depend on the phase of flight and the time to effect of the potential hazard. Hazardous behavior includes erroneous software outputs or performance.

  - 3.2.3 The space system shall provide at least single failure tolerance to catastrophic events, with specific levels of failure tolerance and implementation (similar or dissimilar redundancy) derived via an integration of the design and safety analysis.
  - 3.2.4 The space system shall provide the failure tolerance capability in 3.2.3 without the use of emergency equipment and systems.
  - 3.3.2 The crewed space system shall provide the capability for the crew to manually override higher level software control and automation (such as automated abort initiation, configuration change, and mode change) when the transition to manual control of the system will not cause a catastrophic event.
- NPR 7150.2D: NASA SOFTWARE ENGINEERING REQUIREMENTS
  - 3.7.3 If a project has safety-critical software or mission-critical software, the project manager shall implement the following items in the software: [SWE-134] …
    - No single software event or action is allowed to initiate an identified hazard. …
- Software Assurance Standards to Assure these Requirements are Met:
  - NPR 8739.8A: SOFTWARE ASSURANCE AND SOFTWARE SAFETY STANDARD
  - NASA-STD-8719.13B: SOFTWARE SAFETY STANDARD

Software Failure Classes & Categories

- Consider Two classes of software common cause:
  - Fail Silent – Computers stop outputting, Ex: simultaneous "crash"
  - Erroneous output – *Software behaves unexpectedly / does the wrong thing – Broader class*
    - *Both should be considered when designing for fault-tolerance*
- Why distinguish?
  - *Detection and response is different* -- Easier to know if software "crashed" – watchdog timer
    - How to determine if automation/software is doing something *wrong*? – ex. Independent monitoring
    - Space systems approach these manifestations in different ways – mainly human-in-the-loop
- Fail-Silent Cause Examples (*loss of output*)
  - Operating System Halt, memory access violation, infinite loop / process Starvation
- Erroneous Output Causes Examples (*wrong output*)
  - Coding/Logic Error - Missing/Wrong Requirements, Insufficient modeling of real-world, unanticipated situations
  - Data Parameter Misconfigured - Wrong data input, database, Units, precision, sign
  - Unanticipated / Erroneous Sensor Input
  - Erroneous Command Input - Operator / Procedural Error

55 Significant Historical Software Incidents (1962 – 2023)

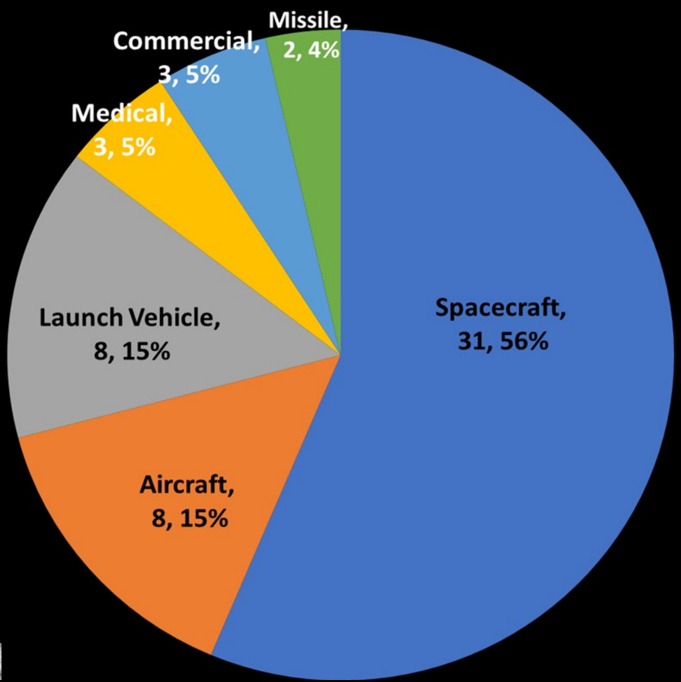
1962 Mariner 1 – Atlas-Agena	1965 Gemini 3	1965 Gemini 5	1968 Apollo 8	1969 Apollo 10
1981 STS-1	1982 Viking-1	1985-87 Therac-25	1988 Phobos-1	1988 Soyuz TM-5
1991 Aries - Red Tigress I	1991 Patriot Missile	1992 F-22 Raptor	1994 Clementine Lunar Mission	1994 Pegasus XL STEP-1
1994 Pegasus HAPS	1995 SOHO	1996 Ariane 5	1997 Pathfinder	1998 Delta III
1999 Mars Polar Lander	1999 Mars Climate Orbiter	1999 Titan IV B Centaur	2000 Zenit 3SL	2001 Pegasus XL/HyperX / X-43A
2001 STS-108 through 110	2003 Multidata Systems Radiation Machine	2003 Soyuz - TMA-1	2003 North American Power Grid	2004 Spirit Mars Exploration Rover
2005 CryoSat-1	2005 DART	2006 Mars Global Surveyor	2007 F22 First Deployment	2008 STS-124
2008 Quantas Flight 72, Airbus A330-303	2008 B-2 Spirit -Guam crash	2012 Red Wings Flight 9268 TU-204 crash	2015 Airbus A400M test flight	2015 SpaceX CRS-7
2016 Hitomi X-ray space telescope	2017 SpaceX CRS-10	2018, 2019 Boeing 737 MAX	2019 Boeing Orbital Flight Test (OFT)	2019 Beresheet
2019 Chandrayaan-2 Vicram Lunar Lander	2020 Amazon Web Service (AWS) Kinesis	2020 BD Alaris™ Infusion Pump	2021 Global Facebook Outage	2021 ISS Attitude Spin
2022 CAPSTONE	2023 NOTAM – Notice To Air Mission	2023 ispace Hakuto-R	2023 Launcher Orbiter SN3 space tug	2023 Voyager-2

Significant Software Failure –

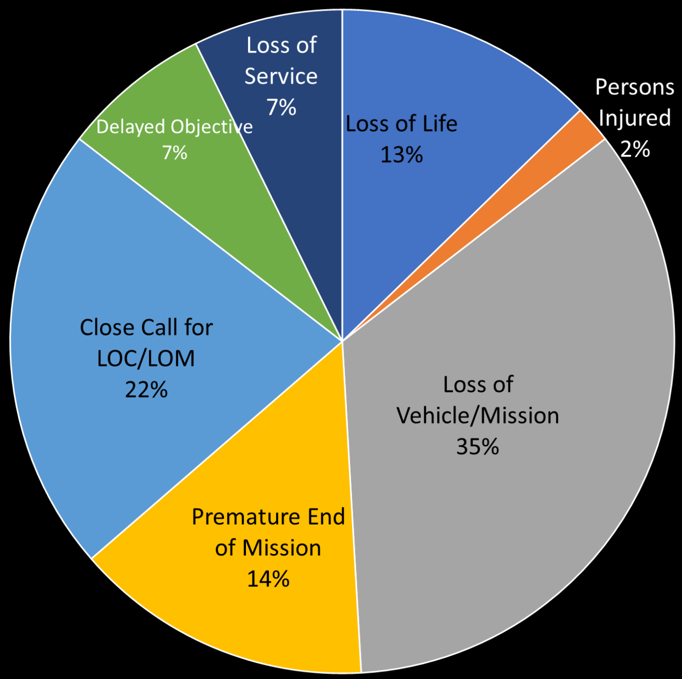
- Software/automation *did not behave as expected* causing loss of life, injury, loss/end of mission, or significant close-call
- **NOTE**: The *root cause* of these failures may not all be software (***why*** it was programmed like that), but how the incident *initially behaved* during operations is characterized

- Categorization:
  - Fail silent or erroneous?
  - Correctable by reboot?
  - Absence of Code?
  - Unknown/unknown?
  - Error Location?
  - Computer Science Discipline?
  - Unknown-unknown?

Industries in Data Set

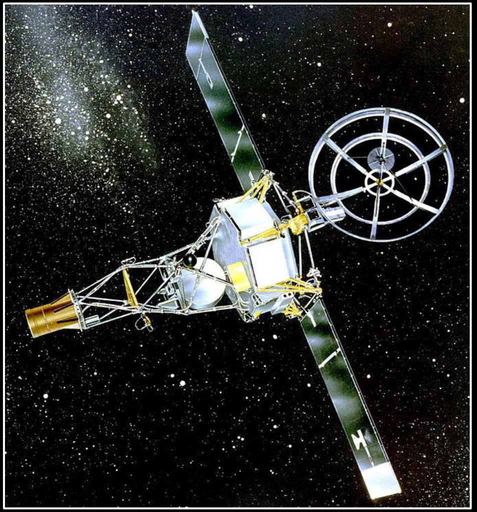


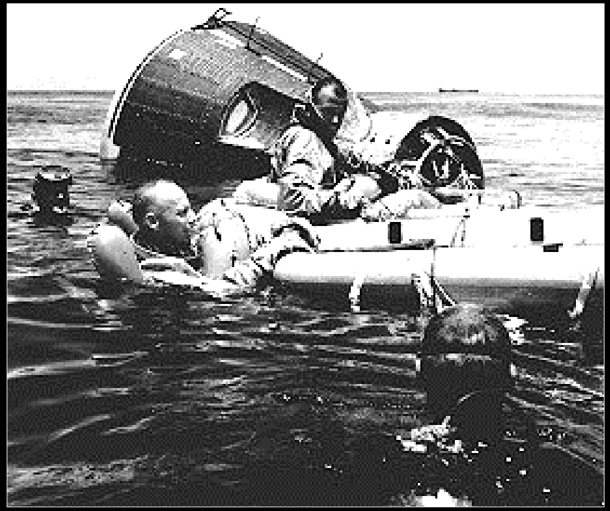
Impact/Result of Failure



Historical Software Incidents (1962-1981)

Year	Flight or System	Title	Result / Outcome	Erroneous or Silent?	Reboot Fix?	Missing Code?	Error Location	Unknown-unknown
1962	**Mariner 1 Mission – Atlas-Agena**	Programmer error in ground guidance veered launch vehicle off course	Loss of vehicle	Erroneous Output	No	No	Code/Logic	No
1965	**Gemini 3**	Incorrect lift estimate causes short landing	Landed 84 km short, crew manually compensated, decreasing short landing error	Erroneous Output	No	Yes	Code/Logic	Yes
1965	**Gemini 5**	Data error of earth rotation lands Gemini 5 short	Landed 130 km short	Erroneous Output	No	No	Data	No
1968	**Apollo 8**	Memory Inadvertently Erased	Close Call fixed manually	Erroneous Output	No	No	Command Input	No
1969	**Apollo 10**	Switch Misconfigured as bad input data to abort guidance	Vehicle tumbled, close call, recovered manually	Erroneous Output	No	No	Data	No
1981	**STS-1**	Failure of computers to sync	Launch Scrub of First Shuttle flight	Fail Silent	Yes	Yes	Code/Logic	No





(Photo Credits: NASA)

Historical Software Incidents (1982-1994)

Year	Flight or System	Title	Result / Outcome	Erroneous or Silent?	Reboot Recoverable	Missing Code?	Error Location	Unknown-unknown?
1982	**Viking-1**	Erroneous Command caused loss of comm	End of mission	Erroneous Output	No	No	Command Input	No
1985-87	**Therac-25**	Radiation Therapy machine output lethal doses, user input speed	Four deaths, two chronic injured	Erroneous Output	No	No	Code/Logic	No
1988	**Phobos-1**	Erroneous unchecked uplinked command lost vehicle	Loss of vehicle/Mission	Erroneous Output	No	No	Command Input	No
1988	**Soyuz TM-5**	Wrong code executed to perform de-orbit burn	Extra day in orbit, New code uplinked	Erroneous Output	No	No	Code/Logic	No
1991	**Aries - Red Tigress I**	Bad command causes guidance error	Loss of Vehicle	Erroneous Output	No	No	Sensor Input	No
1991	**Patriot Missile**	Patriot failed target intercept due to 24-bit rounding error growth in time over time	Failed to intercept scud missile, resulting in American barracks being struck, 28 soldiers killed, 100 injured	Erroneous Output	Yes	No	Code/Logic	No
1992	**F-22 Raptor**	Software failed to compensate for pilot-induced oscillation in presence of lag	Loss of test vehicle	Erroneous Output	No	Yes	Sensor Input	Yes
1994	**Clementine Lunar Mission**	Erroneous thruster firing exhausted propellant, cancelling asteroid flyby	Failed mission objective	Erroneous Output	No	No	Code/Logic	No

Photo Credits: The National Archives, NAID: 6361754 (top), NAID: 6424495 (bottom)

Historical Software Incidents (1994-1999)

Year	Flight or System	Title	Result / Outcome	Erroneous or Silent?	Reboot Fix?	Missing Code?	Error Location	Unknown - unknown
1994	**Pegasus XL STEP-1**	Booster loss of control due to lateral instability	Loss of vehicle/Mission	Erroneous Output	No	Yes	Code/Logic	Yes
1994	**Pegasus HAPS**	Navigation software error prematurely shut down upper stage	Unintended/low orbit	Erroneous Output	No	Yes	Code/Logic	No
1995	**Solar and Heliospheric Observatory (SOHO)**	Gyro Data used from unpowered sensor spins vehicle out of communication	Loss of mission during extended use	Erroneous Output	No	Yes	Code/Logic	No
1996	**Ariane 5 Maiden Flight**	Unprotected overflow in floating-point to integer conversion disrupted inertial navigation system	Loss of Vehicle	Erroneous Output	No	No	Code/Logic	No
1997	**Pathfinder**	Software priority inversion caused images to stall	Close Call for Mission Loss	Erroneous Output	No	No	Code/Logic	No
1998	**Delta III**	Unanticipated 4Hz Oscillation in control system led to vehicle loss	Loss of vehicle	Erroneous Output	No	Yes	Code/Logic	Yes
1999	**Mars Polar Lander**	Premature shut down of landing engine due to misinterpretation of landing signature	Loss of Vehicle/mission	Erroneous Output	No	Yes	Sensor Input	No
1999	**Mars Climate Orbiter**	Metric vs. imperial units error	Loss of vehicle/mission	Erroneous Output	No	No	Data	No

Mars Polar Lander (Credit: NASA)

Historical Software Incidents (1999-2003)

Year	Flight or System	Title	Result / Outcome	Erroneous or Silent?	Reboot Fix?	Missing Code?	Error Location	Unknown-unknown
1999	**Titan IV B Centaur**	Programming error omitting decimal in data file caused loss of control	Unintended orbit, Milstar Satellite lost 10 days after launch	Erroneous Output	No	No	Data	No
2000	**Zenit 3SL**	Ground software error failed to close valve.	Loss of Vehicle	Erroneous Output	No	No	Code/Logic	No
2001	**Pegasus XL/HyperX Launch Vehicle / X-43A**	Airframe failure due to inaccurate analytical models	Loss of vehicle/mission	Erroneous Output	No	Yes	Code/Logic	Yes
2001	**STS-108 through 110**	Shuttle main engine controller mix-ratio software coefficient sign-flip error	Significant close call, SME underperformance, though not extreme enough to not reach orbit.	Erroneous Output	No	No	Data	No
2003	**Multidata Systems Radiation Machine**	Radiation Therapy machine output lethal doses, counterclockwise user input	Many injured, 15 people dead.	Erroneous Output	No	No	Code/Logic	No
2003	**Soyuz - TMA-1**	Undefined yaw value triggered Ballistic reentry	landed 400 km short	Erroneous Output	No	No	Code/Logic	No
2003	**North American Electric Power Grid**	Real-time software errors contribute to Widespread power outage	Widespread Loss of Power Service (2 hr - 4 days)	Fail Silent	No	No	Code/Logic	No

STS-108 Crew (Credit: NASA)

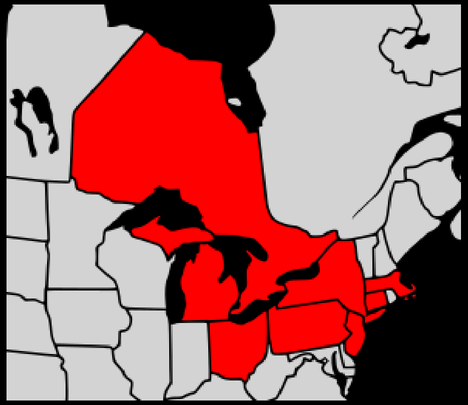


Photo Attribution: Lokal_Profil, under [Creative Commons Attribution-Share Alike 2.5 Generic license]

Historical Software Incidents (2005-2008)

Year	Flight or System	Title	Result / Outcome	Erroneous or Silent?	Reboot Fix?	Missing Code?	Error Location	Unknown-unknown
2005	**CryoSat-1**	Missing command causes loss of vehicle	Loss of Vehicle	Erroneous Output	No	Yes	Code/Logic	No
2005	**DART (Demonstration of Autonomous Rendezvous Technology)**	Navigation software errors fail mission objectives.	Loss of mission objectives	Erroneous Output	No	No	Code/Logic	No
2006	**Mars Global Surveyor (MGS)**	Erroneous command led to pointing error and power/vehicle loss	Premature Loss of vehicle	Erroneous Output	No	No	Code/Logic	No
2007	**F22 First Deployment**	International Date Line crossing crashed computer systems	Loss of navigation & communication	Fail Silent	No	Yes	Code/Logic	No
2008	**STS-124**	All 4 shuttle computers fail / disagree during fueling	Fueling stopped	Erroneous Output	No	Yes	Sensor Input	No
2008	**Quantas Flight 72, Airbus A330-303**	Sensor Input spikes caused autopilot to pitch-down, resulting in crew and passenger injuries	One crew member and 11 passengers suffered serious injuries	Erroneous Output	No	Yes	Sensor Input	Yes

Quantas Flight 72 (Credit: Masakatsu Ukon, CC BY-SA 2.0, via Wikimedia Commons).

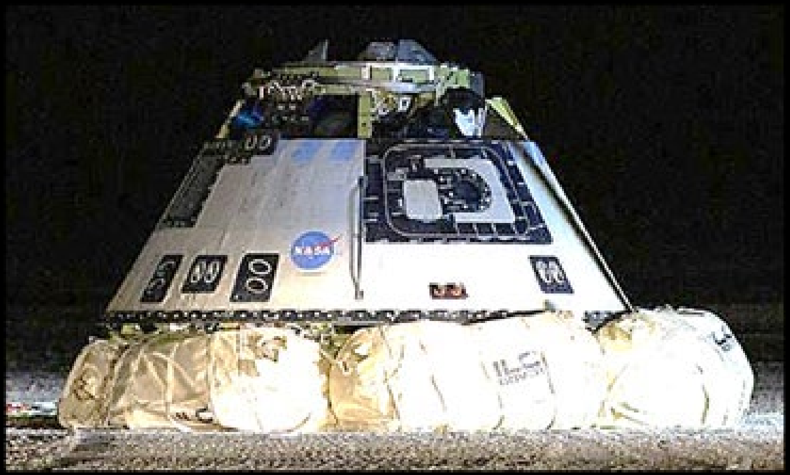
Historical Software Incidents (2008-2017)

Year	Flight or System	Title	Result / Outcome	Erroneous or Silent?	Reboot Fix?	Missing Code?	Error Location	Unknown-unknown
2008	**B-2 Spirit - Guam crash**	Miscalculation in flight computers with missing input data calculated uncommanded pitch up	Crew members successfully ejected.	Erroneous Output	No	Yes	Sensor Input	Yes
2012	**Red Wings Flight 9268 TU-204 crash**	Unanticipated landing circumstances coupled with design features resulted in crash landing	5 of 8 crewmembers killed	Erroneous Output	No	Yes	Code/Logic	Yes
2015	**Airbus A400M test flight**	Missing software parameters during installation cause crash	Four fatalities	Erroneous Output	No	No	Data	No
2015	**SpaceX CRS-7**	"Open Chute" command invalidated after launch vehicle failure	Possibly could have saved Dragon capsule from crash landing.	Erroneous Output	No	Yes	Code/Logic	No
2016	**Hitomi X-ray space telescope**	Error in computing spacecraft orientation led to spacecraft loss	Lost of vehicle	Erroneous Output	No	No	Code/Logic	No
2017	**SpaceX CRS-10**	Erroneous relative state vector transmitted to Dragon	ISS rendezvous delay	Erroneous Output	No	No	Data	No
2018, 2019	**737 Max crash**	Unanticipated software response to faulty sensor input	346 people died on two flights	Erroneou s Output	No	Yes	Sensor Input	Yes

CRS-7 Mishap (Credit: credit: Nathan Koga for NSF/L2)

Historical Software Incidents (2018-2021)

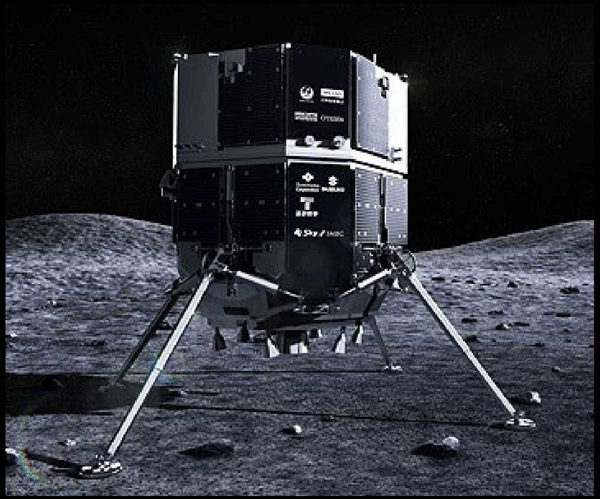
Year	Flight or System	Title	Result / Outcome	Erroneous or Silent?	Reboot Fix?	Missing Code?	Error Location	Unknown-unknown
2019	**Boeing Orbital Flight Test (OFT)**	Incorrect MET causes no ISS rendezvous and short mission, and uncovers other latent LOM software errors.	Failed ISS rendezvous, multi-year program delay	Erroneous Output	No	No	Code/Logic	No
2019	**Beresheet**	Reboots cause engine shutdown on lunar descent	Loss of vehicle	Fail Silent	No	No	Code/Logic	No
2019	**Chandrayaan-2 Vicram Lunar Lander**	Unexpected velocity behavior during descent caused crash landing	Loss of vehicle	Erroneous Output	No	Yes	Code/Logic	No
2020	**Amazon Web Service (AWS) Kinesis**	Maximum threads reached caused cascading server outage	Loss of service, revenues.	Fail Silent	No	Yes	Code/Logic	No
2020	**BD Alaris™ Infusion Pump**	Infusion delivery system software causes injury/death	55 injuries, 1 death	Erroneous Output	No	No	Code/Logic	No
2021	**Global Facebook Outage**	Bad command causes global Facebook and cascading communication outages.	Disrupted communication, loss of revenues	Fail Silent	No	No	Command Input	No
2021	**ISS**	Uncontrolled ISS attitude spin from erroneous thruster firing software	Close Call	Erroneous Output	No	No	Code/Logic	No



Boeing OFT Landing (Photo Credit: NASA)

Historical Software Incidents (2022-present)

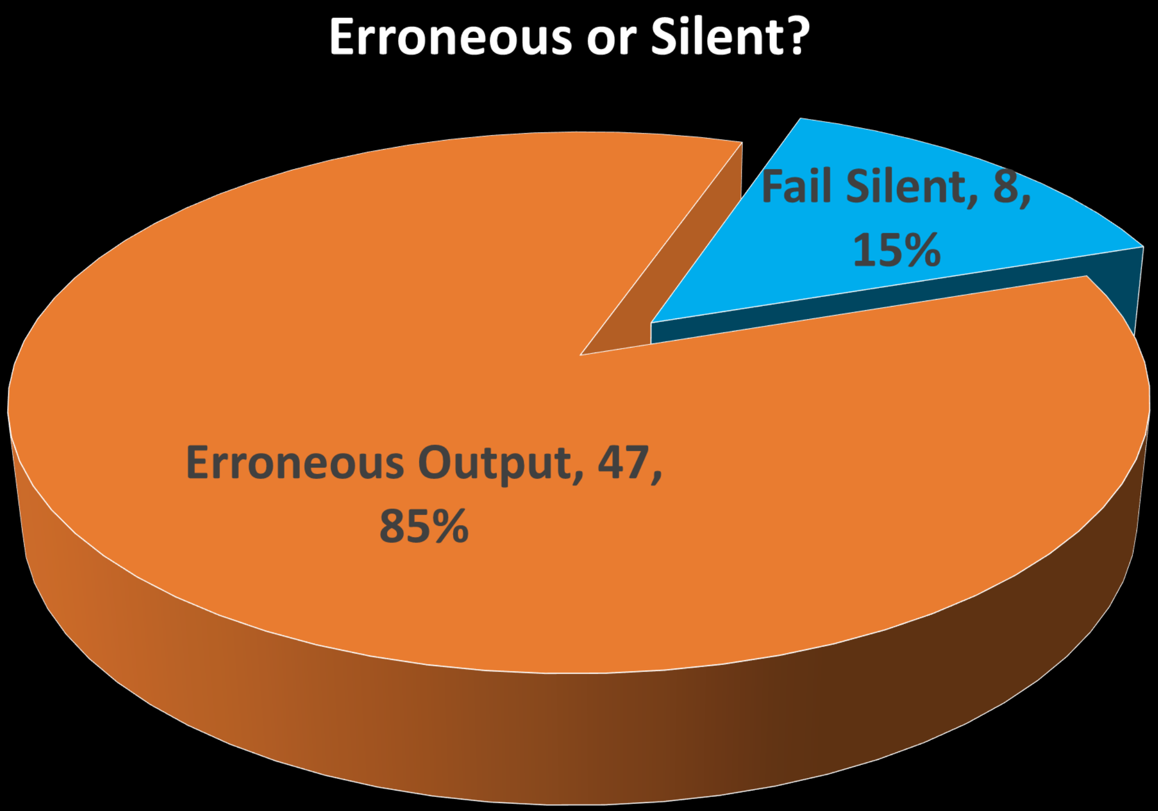
Year	Flight or System	Title	Result / Outcome	Erroneous or Silent?	Reboot Fix?	Missing Code?	Error Location	Unknown-unknown
2022	**CAPSTONE**	Bad Command causes Temporary Comm Loss	Delayed Trajectory Course Maneuver Objective, Close Call for LOM	Erroneous Output	No	No	Command Input	No
2023	**NOTAM – Notice To Air Mission**	Corrupted database file causes flight cancellations	Loss of Service	Fail Silent	No	Yes	Data	No
2023	**ispace Hakuto-R**	Invalidated Altitude data during Lunar descent loses Lander	Loss of Mission	Erroneous Output	No	Yes	Sensor Input	No
2023	**Launcher Orbiter SN3 space tug**	Uncontrolled attitude spin lost power and spacecraft	Loss of Mission	Erroneous Output	No	Yes	Code/Logic	No
2023	**Voyager-2**	Bad command causes 2° antenna shift	Temporary Loss of Communications (Close Call)	Erroneous Output	No	No	Command Input	No



Hakuto-R (Photo Credit: ispace)

Voyager-2 Rendition (Photo Credit: NASA)

Erroneous vs. Fail Silent



Takeaway:

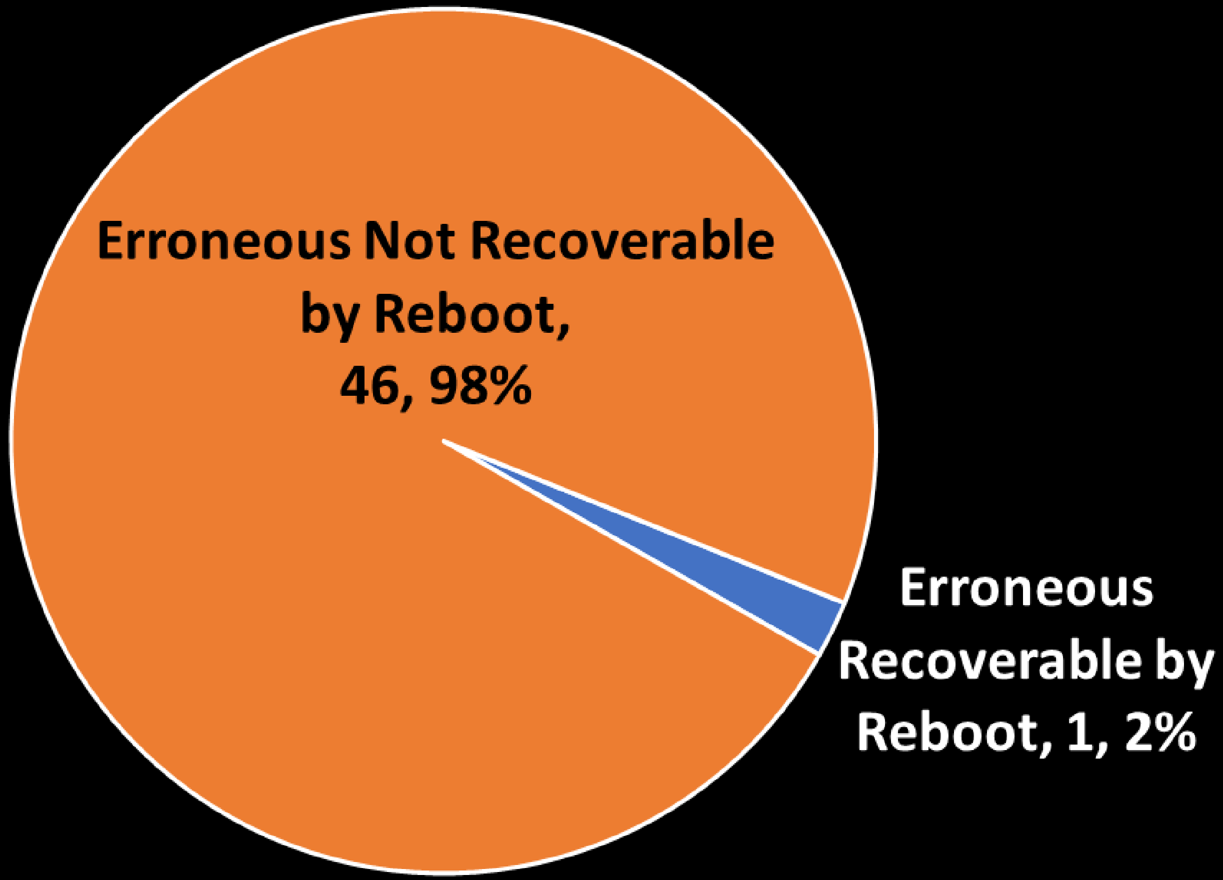
- ***Historically, erroneous output situations were much more prevalent than fail-silent cases***
- ***85-15%, over 5 times as likely***

Fault-tolerant Design Tip:

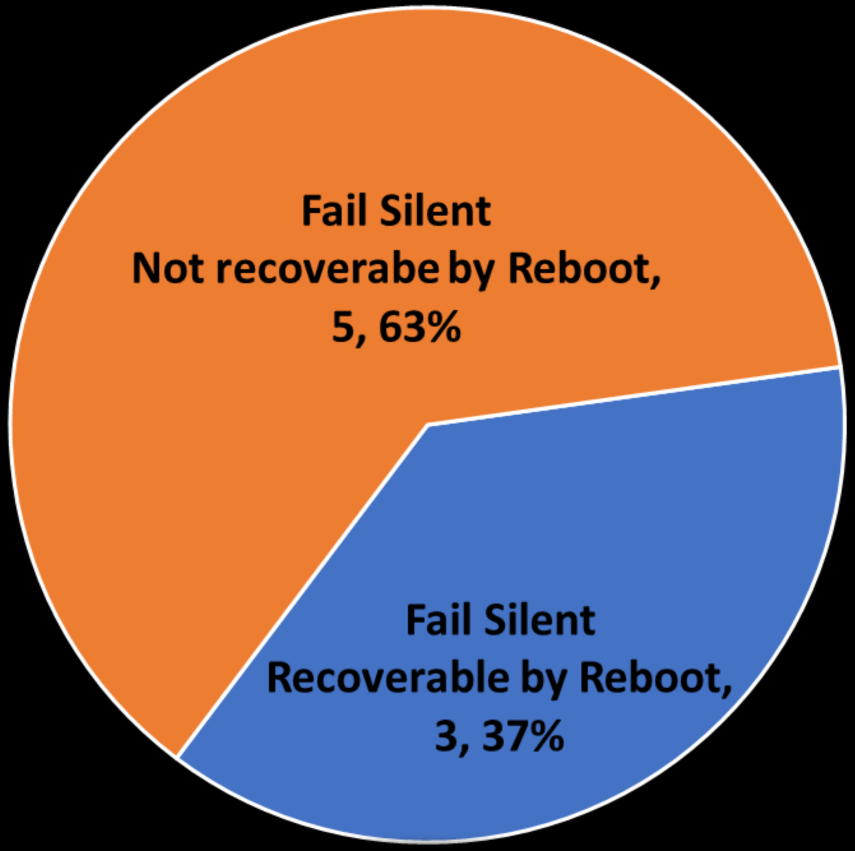
- ***Design should consider relative likelihoods of these manifestations***
- ***Systems should consider the question, "What if the software does something wrong?" at critical moments***

Reboot Recoverability Likelihood
Erroneous vs. Fail Silent

Erroneous Recoverable with Reboot?



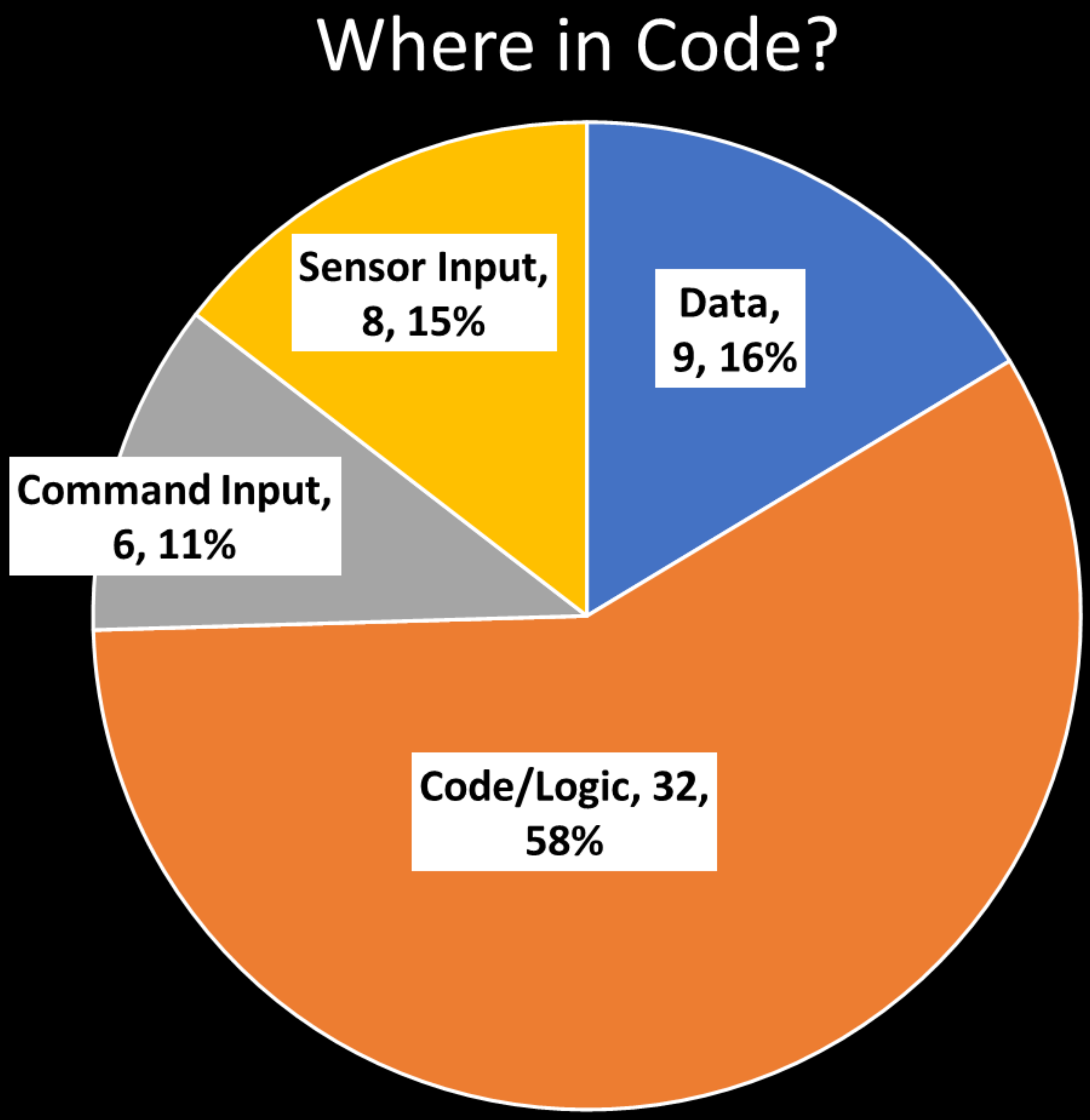
Fail Silent Recoverable with Reboot?



Takeaways:
- ***Rebooting is predominantly ineffective to clear/recover from erroneous output situations***
- ***Rebooting is a partial solution to clear fail-silent errors***

Fault-tolerant Design Tip:
- ***Do not rely on reboot to clear all software faults***

Error Location



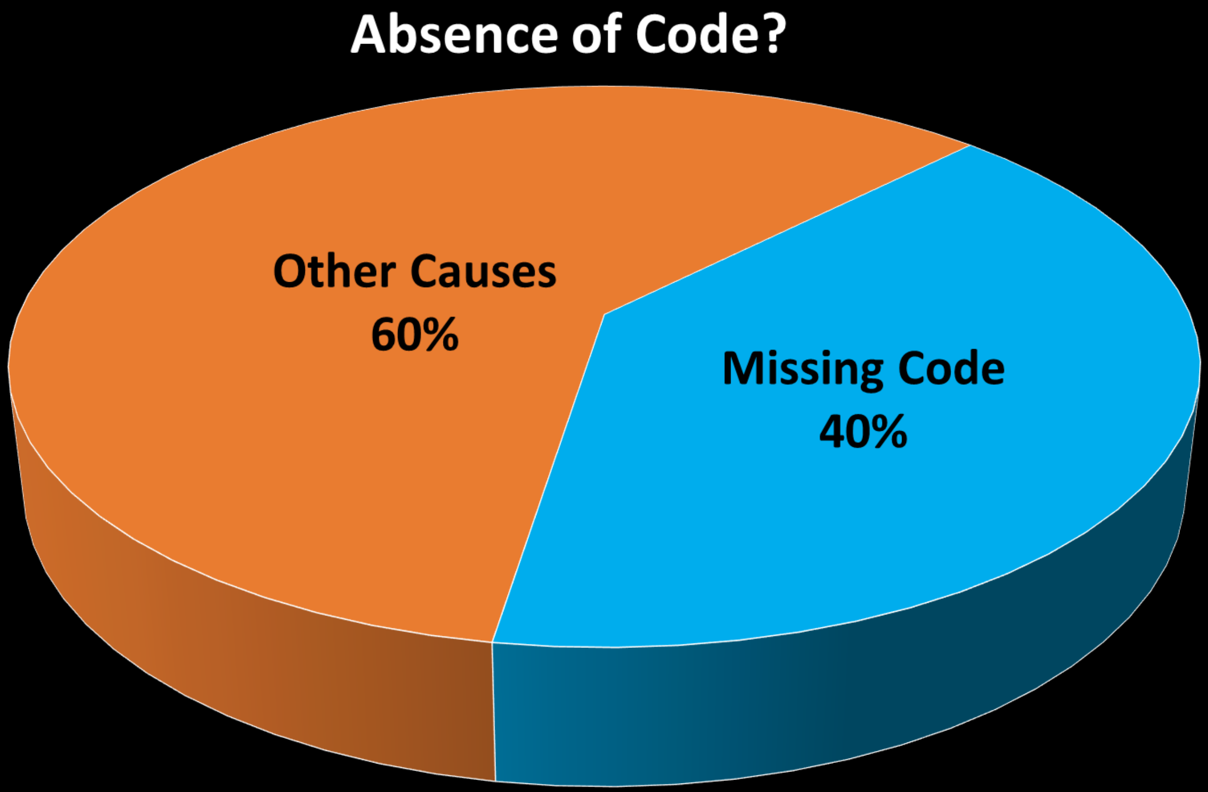
Takeaway(s):

- ***Coding/logic errors account for most software incidents, but very few are "mistakes"***
  - ***This category includes missing requirements, unknowns, unanticipated situations, misunderstanding or incomplete modeling of real-world***
- ***Input Errors – Command or Sensor Input Accounted for 26% of errors***
  - ***Sensor Input are mainly unexpected code/logic errors as well***

Fault-tolerant Code Tips:

- ***Project should test according to likelihoods***
- ***Code/Logic – off-nominal testing, peer review, unit testing, increased simulation/modeling***
- ***Data Misconfiguration – data validation prior to use, system expert review***
- ***Input Errors – Off-nominal or random input test generation***
  - ***Sensor input –hardware-in-the-loop testing***
  - ***Command input – validation, processes/procedures***

Absence of Code ?



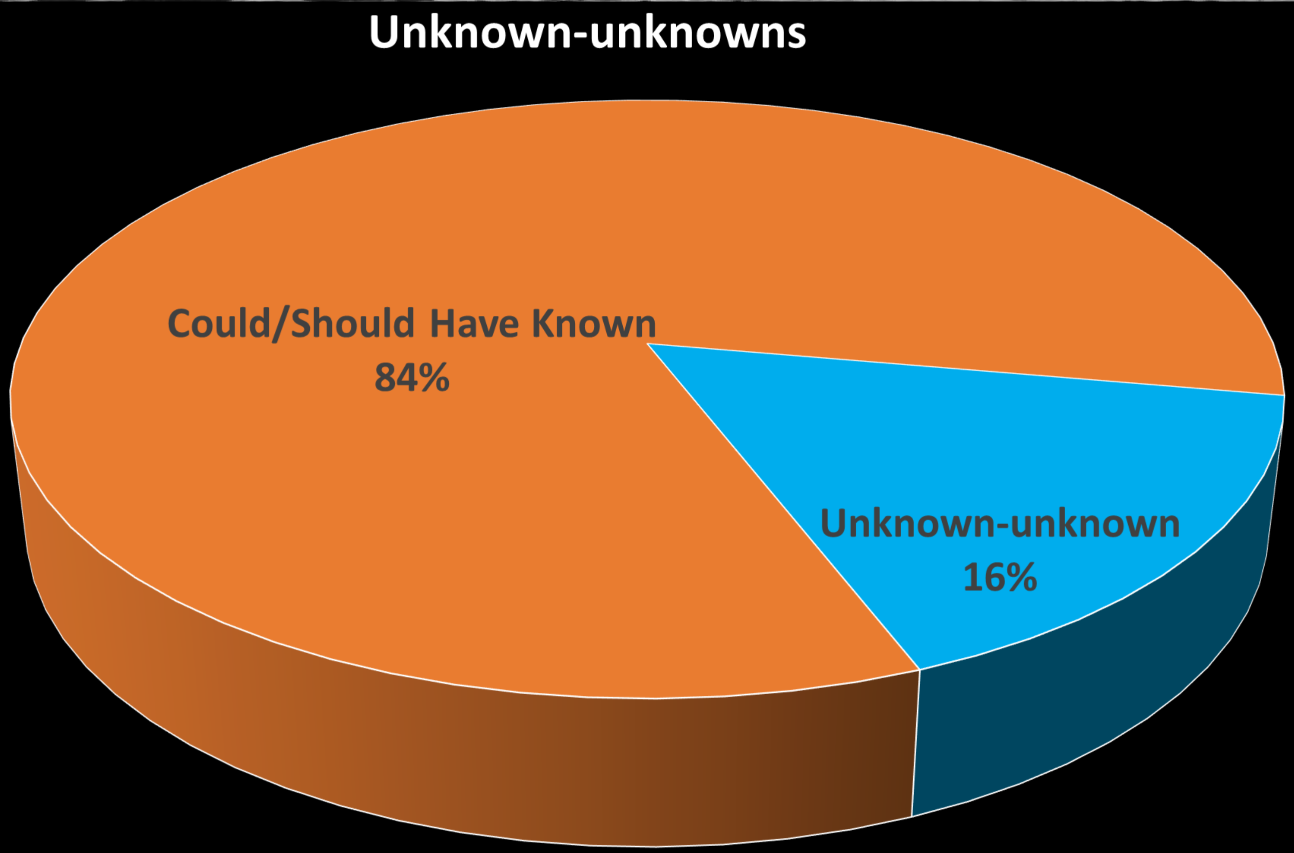
Takeaways:

- ***Many of the studied incidents (40%) could have been averted with the addition of code (in hindsight)***
  - ***Missing Code arises from missing requirements, unanticipated situations, insufficient understanding or modeling of real-world***
- ***Even fully tested code does not uncover errors that arise from missing code/unanticipated situations***
  - ***Hard to test code that is not there – off nominal testing may hep to uncover***

Fault-tolerant Design Tip:

- ***Projects should reserve test time to create off-nominal or unexpected conditions to expose absent code***

"Unknown Unknowns"



Takeaways:

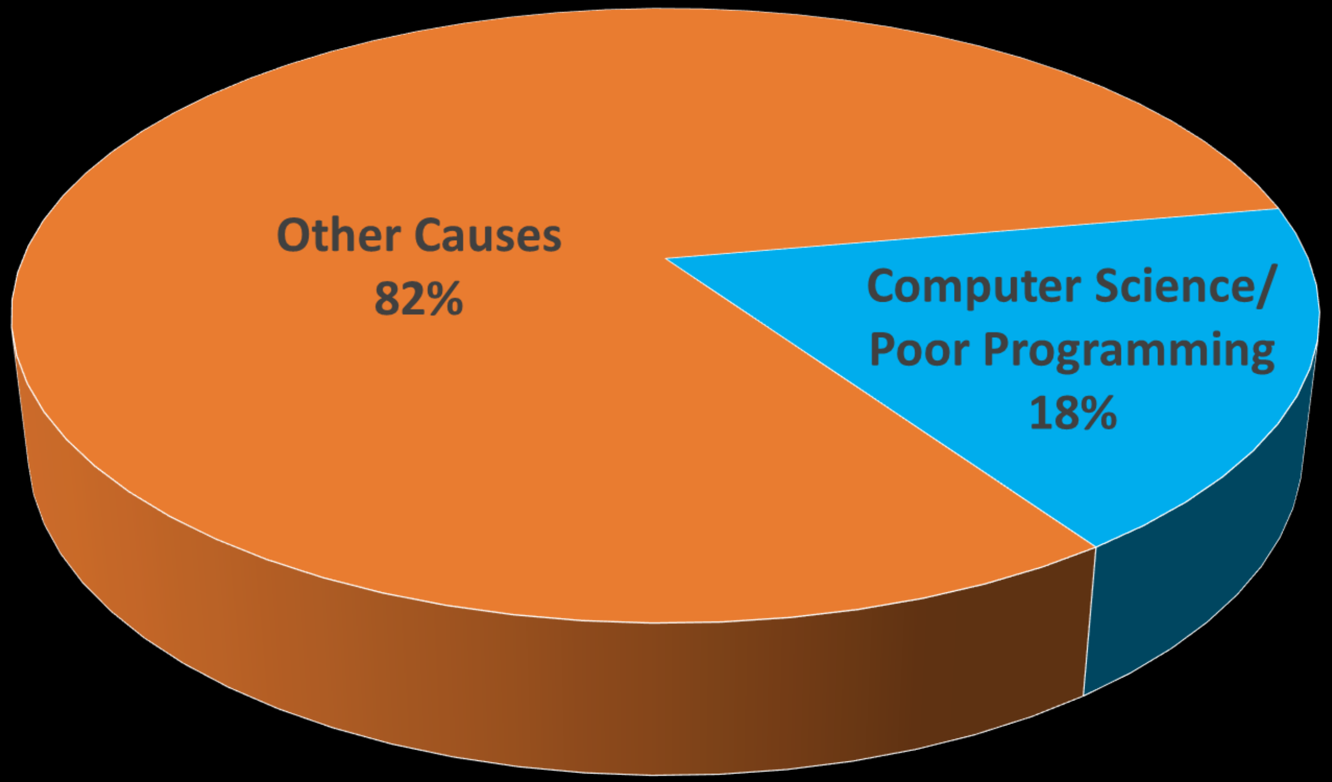
- ***Categorizing "unknown-unknowns" is highly subjective***
- ***Included here:***
  - ***unknown aero/handling, physics***
  - ***Insufficient modeling***
  - ***highly unusual input***
  - ***unexpected behavior in the presence of faults or multiple failures***

Fault-tolerant Design Tip:

- ***Backup strategies should be considered to protect for "unknown- unknowns" and other software error causes***
- ***Projects should actively work to balance risk between "knowing everything" and project constraints (budget/schedule)***

Computer Science Discipline?

Computer Science in Nature



Takeaways:
- ***Most software failures are not a result of something normally considered "computer science" or "software" discipline in nature***
- ***No incidents studied resulted from operating system, programming language, tool chain, or development environment failure***

Fault-tolerant Design Tip:
- ***Projects should consider requiring software "ownership" across multiple disciplines***

Software Errors - Preflight Prevention and In-Flight Mitigation

Captured in a NASA Engineering and Safety Center (NESC) Technical Bulletin "Considerations for Software Fault Prevention and Tolerance"

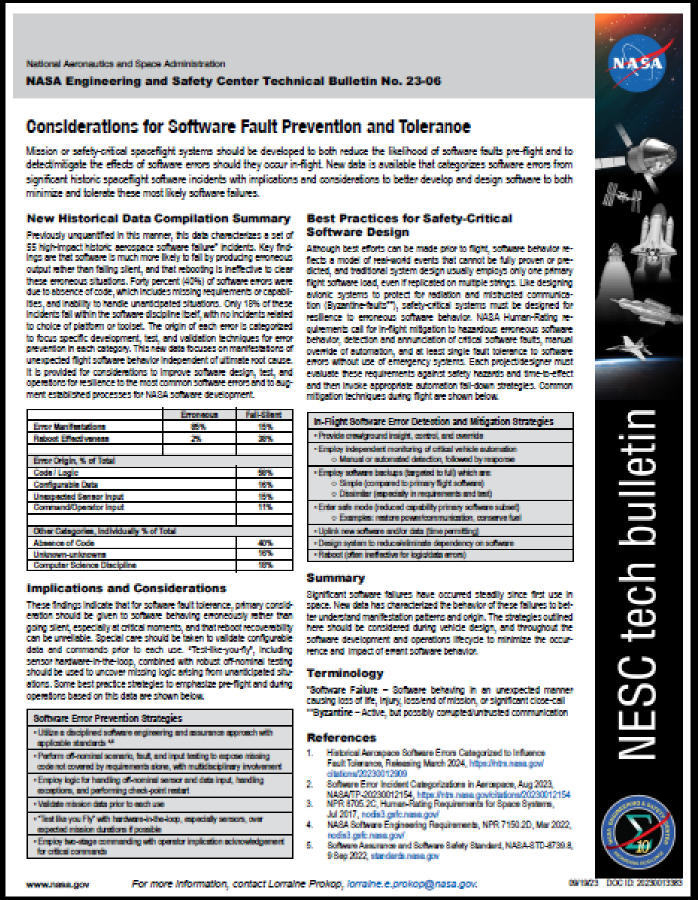
Pre-flight Software Error *Prevention* Strategies

- Utilize a **disciplined software engineering approach**
- Perform **off-nominal** scenario, fault, and sensor input testing to expose missing code
- **Validate mission data** prior to each use
- "**Test like you Fly**" with hardware-in-the-loop over expected mission durations
- **Employ two-stage commanding** with operator implication acknowledgement for critical commands

In-Flight Software Error Detection and *Mitigation* Strategies

- Provide **crew/ground insight, control**, and override
- Employ **independent monitoring** of critical vehicle automation
  - Manual or automated detection, followed by response
- Employ **software backups** (targeted to full) which are:
  - Simple (compared to primary flight software)
  - Dissimilar (especially in requirements and test)
- Enter **safe mode** (reduced capability primary software subset)
  - Examples: restore power/communication, conserve fuel
- **Uplink new software** and/or data (time permitting)
- Design system to reduce/**eliminate dependency on software**
- **Reboot** (limited effectiveness)

Mitigation strategies should be evaluated considering criticality, phase dynamics, and time-to-effect.



National Aeronautics and Space Administration

NASA Engineering and Safety Center Technical Bulletin No. 23-06

Considerations for Software Fault Prevention and Tolerance

Mission or safety-critical spaceflight systems should be developed to both reduce the likelihood of software faults pre-flight and to detect/mitigate the effects of software errors should they occur in-flight. New data is available that categorizes software errors from significant historic spaceflight software incidents with implications and considerations to better develop and design software to both minimize and tolerate these most likely software failures.

New Historical Data Compilation Summary

Previously unquantified in this manner, this data characterizes a set of 55 high-impact historic aerospace software failure* incidents. Key findings are that software is much more likely to fail by producing erroneous output rather than failing silent, and that rebooting is ineffective to clear these erroneous situations. Forty percent (40%) of software errors were due to absence of code, which includes missing requirements or capabilities, and inability to handle unanticipated situations. Only 18% of these incidents fall within the software discipline itself, with no incidents related to choice of platform or toolset. The origin of each error is categorized to focus specific development, test, and validation techniques for error prevention in each category. This new data focuses on manifestations of unexpected flight software behavior independent of ultimate root cause. It is provided for considerations to improve software design, test, and operations for resilience to the most common software errors and to augment established processes for NASA software development.

	Erroneous	Fail-Silent
Error Manifestations	85%	15%
Reboot Effectiveness	2%	38%
Error Origin, % of Total		
Code / Logic		58%
Configurable Data		16%
Unexpected Sensor Input		15%
Command/Operator Input		11%
Other Categories, Individually % of Total		
Absence of Code		40%
Unknown-unknowns		16%
Computer Science Discipline		18%

Implications and Considerations

These findings indicate that for software fault tolerance, primary consideration should be given to software behaving erroneously rather than going silent, especially at critical moments, and that reboot recoverability can be unreliable. Special care should be taken to validate configurable data and commands prior to each use. "Test-like-you-fly", including sensor hardware-in-the-loop, combined with robust off-nominal testing should be used to uncover missing logic arising from unanticipated situations. Some best practice strategies to emphasize pre-flight and during operations based on this data are shown below.

Software Error Prevention Strategies
• Utilize a disciplined software engineering and assurance approach with applicable standards [4,5]
• Perform off-nominal scenario, fault, and input testing to expose missing code not covered by requirements alone, with multidisciplinary involvement
• Employ logic for handling off-nominal sensor and data input, handling exceptions, and performing check-point restart
• Validate mission data prior to each use
• "Test like you Fly" with hardware-in-the-loop, especially sensors, over expected mission durations if possible
• Employ two-stage commanding with operator implication acknowledgement for critical commands

Best Practices for Safety-Critical Software Design

Although best efforts can be made prior to flight, software behavior reflects a model of real-world events that cannot be fully proven or predicted, and traditional system design usually employs only one primary flight software load, even if replicated on multiple strings. Like designing avionic systems to protect for radiation and mistrusted communication (Byzantine-faults**), safety-critical systems must be designed for resilience to erroneous software behavior. NASA Human-Rating requirements call for in-flight mitigation to hazardous erroneous software behavior, detection and annunciation of critical software faults, manual override of automation, and at least single fault tolerance to software errors without use of emergency systems. Each project/designer must evaluate these requirements against safety hazards and time-to-effect and then invoke appropriate automation fall-down strategies. Common mitigation techniques during flight are shown below.

In-Flight Software Error Detection and Mitigation Strategies
• Provide crew/ground insight, control, and override
• Employ independent monitoring of critical vehicle automation ○ Manual or automated detection, followed by response
• Employ software backups (targeted to full) which are: ○ Simple (compared to primary flight software) ○ Dissimilar (especially in requirements and test)
• Enter safe mode (reduced capability primary software subset) ○ Examples: restore power/communication, conserve fuel
• Uplink new software and/or data (time permitting)
• Design system to reduce/eliminate dependency on software
• Reboot (often ineffective for logic/data errors)

Summary

Significant software failures have occurred steadily since first use in space. New data has characterized the behavior of these failures to better understand manifestation patterns and origin. The strategies outlined here should be considered during vehicle design, and throughout the software development and operations lifecycle to minimize the occurrence and impact of errant software behavior.

Terminology

*Software Failure – Software behaving in an unexpected manner causing loss of life, injury, loss/end of mission, or significant close-call

**Byzantine – Active, but possibly corrupted/untrusted communication

References

1. Historical Aerospace Software Errors Categorized to Influence Fault Tolerance, Releasing March 2024, https://ntrs.nasa.gov/citations/20230012909
2. Software Error Incident Categorizations in Aerospace, Aug 2023, NASA/TP-20230012154, https://ntrs.nasa.gov/citations/20230012154
3. NPR 8705.2C, Human-Rating Requirements for Space Systems, Jul 2017, nodis3.gsfc.nasa.gov/
4. NASA Software Engineering Requirements, NPR 7150.2D, Mar 2022, nodis3.gsfc.nasa.gov/
5. Software Assurance and Software Safety Standard, NASA-STD-8739.8, 9 Sep 2022, standards.nasa.gov

NESC tech bulletin

www.nasa.gov For more information, contact Lorraine Prokop, lorraine.e.prokop@nasa.gov. 09/19/23 DOC ID: 20230013383

2023, Lorraine Prokop, Ph.D, NASA Technical Fellow for Software

Software Common Cause Failure Summary

- Software “common cause” or “common mode” errors occur when a single software error results in unexpected behavior, even if running on multiple strings
- Software in NASA Space Systems should be architected for redundancy based on criticality and time-to-effect, with requirements driven primarily by NPR 8705.2C and NPR 7150.2D
- Software Errors manifest in two ways: Silent or Erroneous
- Study of historical software incidents indicates the following
  - Erroneous output is much more prevalent – 85% of the incidents
  - Rebooting is largely ineffective to recover from erroneous situations, and not reliable for silent software
  - Software logic errors are the most common form, then data config, and 26% of errors arise from input
  - Missing Code accounted for 36% (including requirements, unknowns) of historic software errors
  - “Unknown-unknowns” account for over 15% of software error incidents, subjectively
- Fault-tolerant systems should be designed with these statistics in mind – overall recommendations
  - Consider the Erroneous Case more than failing silent
  - Don’t always rely on reboot
  - Employ hardware-in-the-loop, test-like-you-fly, and off-nominal testing
  - Validate configuration and command data prior to use
  - Consider use of backup strategies for critical events

References & Follow-on Work

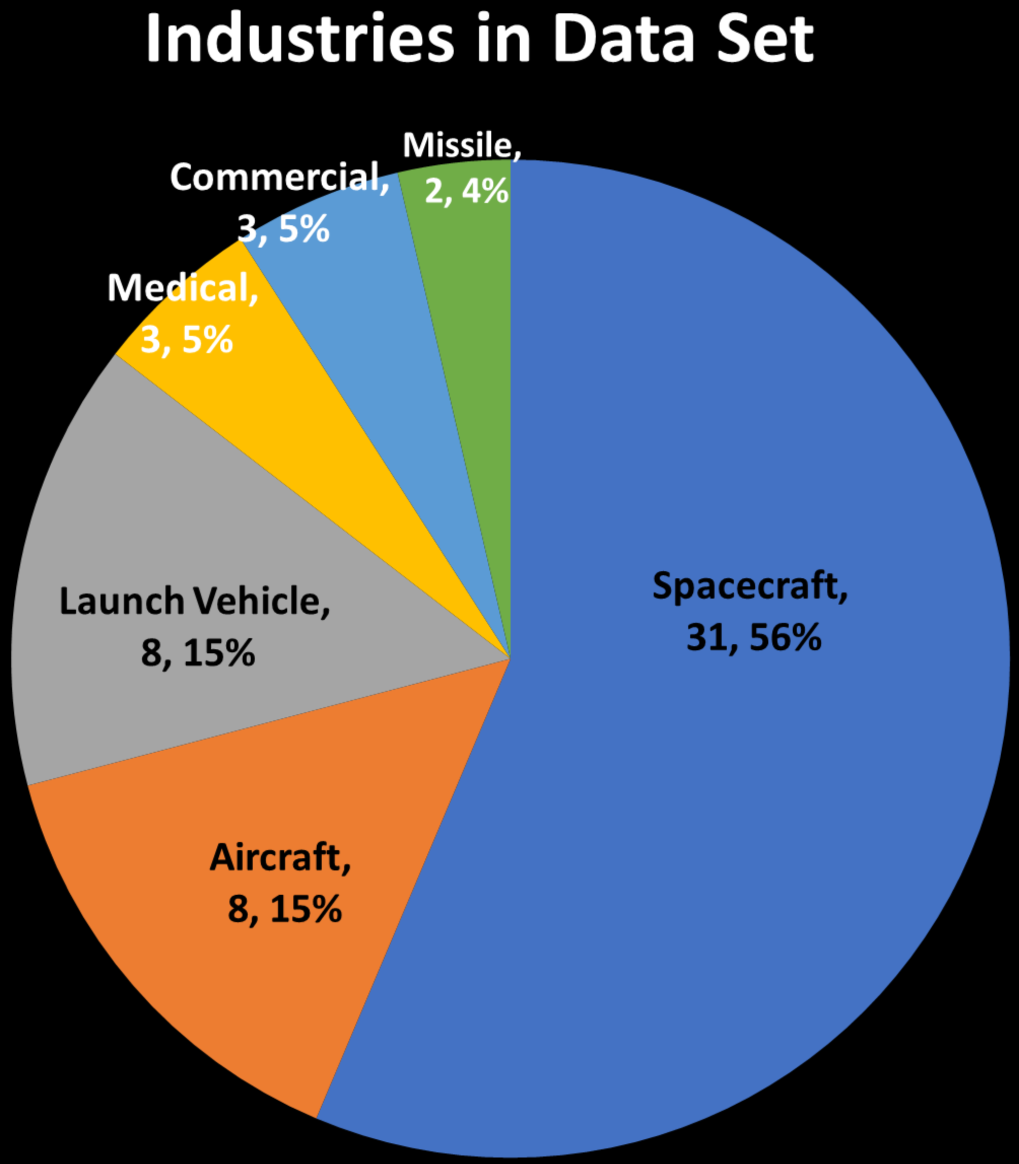
- NESC Technical Bulletin 23-06: Considerations for Software Fault Prevention and Tolerance, September 2023.
- Prokop, Lorraine, E., "Software Error Incident Categorizations in Aerospace", NASA Technical Publication, NASA/TP−20230012154. August 2023.
- "Historical Aerospace Software Errors Categorized to Influence Fault Tolerance", March 2024, AIAA Aerospace Conference 2024, https://ntrs.nasa.gov/citations/20230012909
- Prokop, Lorraine, E., "Software Error Incident Categorizations in Aerospace", [Manuscript in publication], Journal of Aerospace Information Systems.

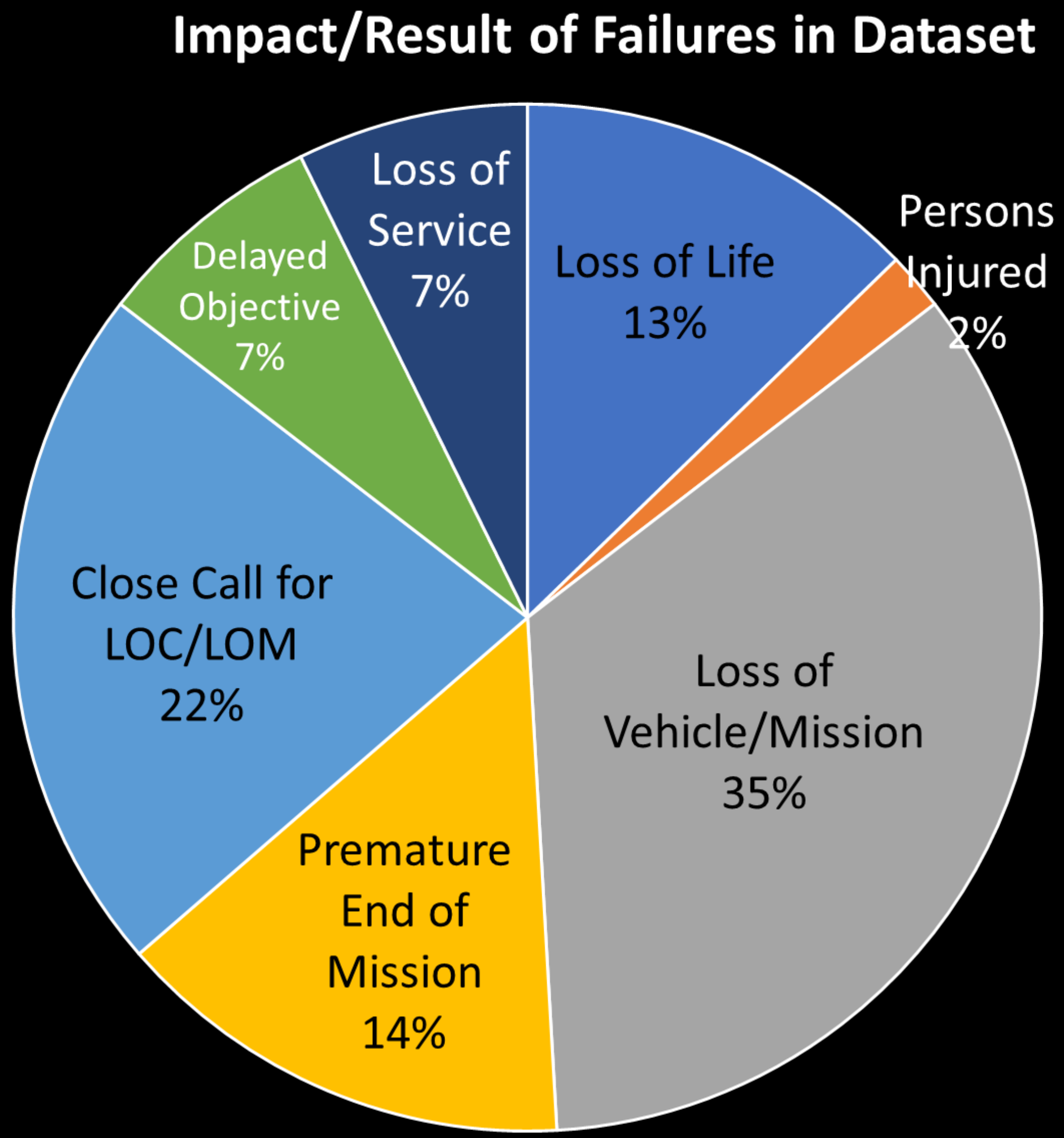
- The dataset used for this study, with more description and references, is available upon request

- Follow-on work:
  - This dataset can be used for further study, for example, to answer the following
    - What was the root cause of this error? (Why was the software programmed the way it was?)
    - Would a backup system have helped?
      - What kind of a backup system could have helped?"
      - Would a human-in-the-loop, a dissimilar backup, a monitor system, or no backup at all be best?
    - Was this a multi-string common-cause failure?
    - Was a manual or automated backup system used?
    - What phase of the project could/should this incident been averted?
    - How much and what type of testing may have uncovered these errors?

Backup

Dataset Industry & Impact Breakdown





Historical Analysis

- Historic Failure Incidents Involving Software
  - We studied software significant failure incidents primarily within NASA and aerospace – when automation did not behave as expected
    - Software Failure – Software/automation did not behave as expected causing loss of life, injury, loss/end of mission, or significant close-call
    - 55 incidents were characterized – since beginning of computers
      - Aerospace (49) – loss of life, mission, close-call
      - Non-Aerospace (6) – 3 Medical (loss of life) , 3 Commercial (3) (loss of service)
    - We categorized software errors to determine:
      - Which is more prevalent – fail silent or erroneous?
      - Could the failure have been corrected by reboot?
      - Was this an unanticipated situation – missing code, wrong code, or unknown unknown?
      - *Where* in the code was the failure introduced?
  - **NOTE**: The *root cause* of these failures may not all be attributable to software (**why** it was programmed like that), but how the incident *initially manifested* during operations (**how** it behaved) is characterized

Dataset Sample: Historical Software Incidents (1982-1994)

Year	Flight or System	Title	Result / Outcome	Erroneous or Silent?	Reboot Recoverable	Missing Code?	Error Location	Unknown-unknown?
1982	**Viking-1**	Erroneous Command caused loss of comm	End of mission	Erroneous Output	No	No	Command Input	No
1985-87	**Therac-25**	Radiation Therapy machine output lethal doses, user input speed	Four deaths, two chronic injured	Erroneous Output	No	No	Code/Logic	No
1988	**Phobos-1**	Erroneous unchecked uplinked command lost vehicle	Loss of vehicle/Mission	Erroneous Output	No	No	Command Input	No
1988	**Soyuz TM-5**	Wrong code executed to perform de-orbit burn	Extra day in orbit, New code uplinked	Erroneous Output	No	No	Code/Logic	No
1991	**Aries - Red Tigress I**	Bad command causes guidance error	Loss of Vehicle	Erroneous Output	No	No	Sensor Input	No
1991	**Patriot Missile**	Patriot failed target intercept due to 24-bit rounding error growth in time over time	Failed to intercept scud missile, resulting in American barracks being struck, 28 soldiers killed, 100 injured	Erroneous Output	Yes	No	Code/Logic	No
1992	**F-22 Raptor**	Software failed to compensate for pilot-induced oscillation in presence of lag	Loss of test vehicle	Erroneous Output	No	Yes	Sensor Input	Yes
1994	**Clementine Lunar Mission**	Erroneous thruster firing exhausted propellant, cancelling asteroid flyby	Failed mission objective	Erroneous Output	No	No	Code/Logic	No

Photo Credits: The National Archives,
NAID: 6361754 (top), NAID: 6424495 (bottom)