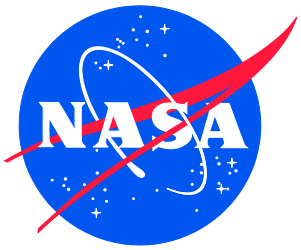
NASA/TP–20230012154
NESC-NPP-22-01775



Software Error Incident Categorizations in Aerospace

Lorraine E. Prokop/NESC
Langley Research Center, Hampton, Virginia

August 2023

NASA STI Program Report Series

Since its founding, NASA has been dedicated to the advancement of aeronautics and space science. The NASA scientific and technical information (STI) program plays a key part in helping NASA maintain this important role.

The NASA STI program operates under the auspices of the Agency Chief Information Officer. It collects, organizes, provides for archiving, and disseminates NASA's STI. The NASA STI program provides access to the NTRS Registered and its public interface, the NASA Technical Reports Server, thus providing one of the largest collections of aeronautical and space science STI in the world. Results are published in both non-NASA channels and by NASA in the NASA STI Report Series, which includes the following report types:

1) TECHNICAL PUBLICATION. Reports of completed research or a major significant phase of research that present the results of NASA Programs and include extensive data or theoretical analysis. Includes compilations of significant scientific and technical data and information deemed to be of continuing reference value. NASA counterpart of peer-reviewed formal professional papers but has less stringent limitations on manuscript length and extent of graphic presentations.

2) TECHNICAL MEMORANDUM. Scientific and technical findings that are preliminary or of specialized interest, e.g., quick release reports, working papers, and bibliographies that contain minimal annotation. Does not contain extensive analysis.

3) CONTRACTOR REPORT. Scientific and technical findings by NASA-sponsored contractors and grantees.

4) CONFERENCE PUBLICATION. Collected papers from scientific and technical conferences, symposia, seminars, or other meetings sponsored or co-sponsored by NASA.

5) SPECIAL PUBLICATION. Scientific, technical, or historical information from NASA programs, projects, and missions, often concerned with subjects having substantial public interest.

6) TECHNICAL TRANSLATION. English-language translations of foreign scientific and technical material pertinent to NASA's mission.

Specialized services also include organizing and publishing research results, distributing specialized research announcements and feeds, providing information desk and personal search support, and enabling data exchange services.

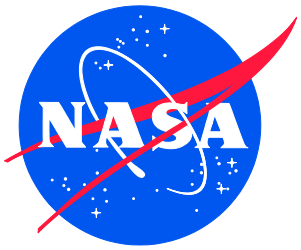
For more information about the NASA STI program, see the following:

7) Access the NASA STI program home page at http://www.sti.nasa.gov

8) Help desk contact information:

https://www.sti.nasa.gov/sti-contact-form/ and select the "General" help request type.

NASA/TP–20230012154
NESC-NPP-22-01775



Software Error Incident Categorizations in Aerospace

Lorraine E. Prokop/NESC
Langley Research Center, Hampton, Virginia

National Aeronautics and
Space Administration

Langley Research Center
Hampton, Virginia 23681-2199

August 2023

Acknowledgments

This work was a culmination of two NESC assessment teams working tirelessly to improve human spaceflight safety by reducing software errors. I would like to sincerely thank each of those members who helped put this dataset together, identify and study incidents, and find credible references. I would like to thank each team member in alphabetical order: Jon Berndt, Tim Brady, Linda Burgess, Jesse Couch, Tim Crumbley, Neil Dennehy, Jenny DeVasher, Captain Victor "Ike" Glover, Bruce Jackson, Dr. Mary Kaiser, Kylene Kramer, John LaNeave, Laura Maynard-Nelson, Dr. Paul Miner, Mike Peacock, David Root, Manuel Rosso-Llopart, Jeremy Shidner, Scott Tashakkor, John West, and the late Aron Wolf. I would also like to thank my son and most critical computer scientist peer reviewer, Daniel Williams, for his valuable insight and comments.

Available from:

NASA STI Program / Mail Stop 148
NASA Langley Research Center
Hampton, VA 23681-2199
Fax: 757-864-6500

Table of Contents

List of Figures

List of Tables

Abstract

Since the first use of computers in space and aircraft, software errors have occurred. These errors can manifest as loss-of-life or less catastrophically. As the demand for automation increases, software in safety-critical systems should be designed to be tolerant to the most likely software faults. This paper categorizes historic aerospace software errors to determine trends of how and where automation is most likely to fail. A distinction between software producing wrong (erroneous) output versus no output (fail-silent) is introduced. Of the historical incidents analyzed, 87% were from software acting unexpectedly rather than simply stopping. Rebooting was found to be ineffective to clear erroneous behavior, and only partially effective for silent software. Errors were traced back to the software logic itself in 62% of cases, 13% within configurable data, and 25% introduced through input. Thirty percent (30%) of unexpected software behavior was caused by the absence of software and 20% was due to "unknown-unknowns". These findings indicate that to achieve fault tolerance in safety-critical systems, backup strategies must be employed to detect and respond to erroneous software behavior beyond only fail-silent cases, and robust off-nominal testing should be performed to uncover unanticipated situations.

1.0 Introduction

This paper explores incidents of unexpected automation/software behavior and software errors primarily in aerospace (and a few other safety-critical and representative systems) to identify and raise awareness of erroneous software manifestation trends. It introduces a dataset of 47 incidents to expose trends in software behavior. It explores if software more often behaves unexpectedly, producing erroneous output, versus simply stopping/crashing. It identifies where within the software architecture the error happened—within code itself, within configurable data, or from sensor or command input. It quantifies cases of missing code, including missing requirements, and quantifies "unknown-unknowns" as causes for unexpected software behavior. By understanding how and where software is most likely to fail, systems may be better designed and proportionately tested for robustness against the most probable failures, and backup strategies may be better architected to minimize software risk.

1.1 Motivation

Before digging in, let's provide a few words of motivation. Software risk seems to be a difficult topic for "non-software people" to grasp as well as an inherently difficult topic to visualize and communicate. Software is intangible and normally concealed beneath layers of abstraction and physical shielding. For example, Figure 1 depicts a flight computer—a Space Shuttle General Purpose Computer—with no software errors. Conversely, Figure 2 shows a flight computer riddled with software errors. With no discernable visual difference between them, the differences only become apparent through use. Software representation is therefore left to the imagination, but undeniably serves as the means to bring hardware to life (or death), providing behavior and personality. It integrates systems together end-to-end, plays a role in virtually all subsystems, and

is ever increasing in complexity. All disciplines should learn to "own" their software and understand its vulnerabilities and inherent risks.

Figure 1. Flight computer without software errors.

(Credit: NASA)

Figure 2. Flight computer with software errors.

(Credit: NASA)

1.2 Incident Dataset

A dataset of 47 historical incidents starting in 1962 is introduced and analyzed. It includes all incidents found since the beginning of employing computers in aerospace to present day such that the software/automation behaved unexpectedly and possibly could or should have been written differently in hindsight to affect a different outcome. In each incident, the automation controlling the system either acted unexpectedly or failed to act (for whatever reason) leading to loss of life, loss of mission, loss of time/revenue, or presented a significant close call. It is important to note that the ultimate root cause of these incidents is not necessarily "software". In fact, it could be argued that in all of these cases the software performed exactly as programmed. Determining root-cause of these failures—identifying *why* the software was programmed that way—is left for further study but may include examples such as lack of system understanding, unknown physics, lack of time or resources, lack of skills, or procedural/process errors.

1.3 Software Common-Cause Errors

Although not specifically studied here, the notion of software errors being "common-cause" should be considered because many, if not all of these incidents could be considered common-cause. Software "common-cause" or "common-mode" failures arise when software failed, either erroneously or silently, but because the software may be duplicated running on multiple redundant computers at the same time, a single software error can affect all redundant computers in the same way simultaneously. This is a software common-cause error. System architecture determines the vulnerability to software common-mode failures. In systems where there is only one copy of flight software, a single software error could be considered a common-cause error. Mitigating software common-cause errors should be assessed based on system criticality and time-to-effect. Some common mitigation strategies include employing a dissimilar software backup system, providing manual control, installing monitoring systems, failing into a safe mode, patching the software, and

of course, rebooting. For example, some crewed space and aircraft allow for manual piloting, which was successfully used in several incidents. Determining which of these strategies, if any, were either employed or may have mitigated these incidents is left for further study.

1.4 Paper Outline

Section 2.0 of this paper identifies the categories and statistics explored with this dataset. It discusses category relevance to system design and in establishing/influencing test strategy. Section 3.0 lists, in chronological order, each historic software incident in the dataset. A description of software's role in the incident is the primary focus, with references left to readers for more in-depth information. Section 4.0 presents statistical results of the categorizations of these incidents along with an interpretive discussion.

2.0 Software Incident Failure Categories

2.1 Erroneous vs. Fail-Silent

First, we must make a distinction between software failing "erroneously," which includes the automation producing wrong or unexpected output, and software failing "silent," providing no output at all (i.e., crashing). This is an important distinction because detecting the "fail-silent" case is usually more straightforward. A watchdog timer can detect the fail-silent case. Rebooting is typically used to recover from a silent computer, but the effectiveness of this strategy is discussed in the next category, "Reboot Recoverability," with associated results provided in Section 4.0.

Detecting and responding to the "erroneous output" case, however, may not be as straightforward. How do you know if the software is doing the wrong thing? If a human is onboard, or a ground team is actively monitoring, they may be able to recognize software performing unexpectedly and override the automation to take appropriate action. But if there is no human in the loop, or if time-critical, software/automatic backup systems may be employed to recognize and respond to the primary software behaving unexpectedly. Fail-down strategies should be employed in safety-critical systems for transitioning to backup strategies or systems to mitigate the erroneous-output case.

2.2 Reboot Recoverability

A common strategy to recover from faulty software is to reboot. Unfortunately, reboots do not fix all software problems. The incident dataset was reviewed subjectively considering the following question, "Would reboot have cleared this problem?" A yes/no answer is tabulated and presented in the Section 4.0. This is important to know because depending on the problem, it is often assumed that performing a simple reboot may correct the problem. But given the effectiveness presented here, a better risk determination can be made for the particular system, and alternate design approaches can be considered.

2.3 Absence of Code

An interesting statistic studied against this dataset is whether the incident could have been avoided by adding code (in hindsight). The incident dataset was reviewed subjectively considering the following question, "Could the problem have been averted by *adding* some code?" A yes/no answer to this question is tabulated in Section 4.0. It is well understood that it is much easier to know what code to add after a mishap rather than predicting the failure in advance. Considering

whether or not the code *could or should have* been there is a more difficult question addressed in the categorization of "unknown-unknowns" below. But simply determining if an incident was the result of the absence of software has large testing implications. If software is only tested against requirements, or tested against code that *exists*, then how can errors caused by the *absence* of software be discovered? This poses a testing challenge. Performing off-nominal testing for random input sets may help to uncover missing code. Test campaigns should consider testing both the existing code as well as for the absence of code proportionately to how errors usually manifest. Results of this investigation and further discussion are provided in Section 4.0.

2.4 Error Location

A categorization as to where in the software the error initially manifested is performed by distinguishing between the following four groups: code/logic, data, sensor input, and command input. The reason for these distinctions is because assuring integrity in each of these areas both pre-flight and operationally have different testing characteristics and procedural validation methods.

First, "coding/logic" includes errors that are in the code itself, encoded into logic or algorithms. This category includes the absence of code as discussed above, and missing requirements, where code could have been written to avert the error. Next, "data" includes those errors due to misconfigured data, or erroneous stored parameters. This is separated from "code/logic" to distinguish between the fact that software is becoming more data-driven, and that data are more likely to change than the code itself. The third category, "sensor input," addresses errors stemming from unexpected or erroneous sensor input. This distinction is made because generating off-nominal tests specifically targeting random sensor input may help to avert this error. The final category, "command input," includes erroneous command input due to operator or procedural error. These errors should normally be averted through command verification during operations prior to their issuance and by process assurance. The overall prevalence of each of these categories is given in Section 4.0.

2.5 Unknown-Unknowns

The last category, "unknown-unknowns," a term popularized by Donald Rumsfeld referring to "the ones we don't know we don't know" [1]", is a highly subjective category, but attempts to conservatively quantify how many of these incidents arose from knowledge only realized or conceived in hindsight that could not have been discovered ahead of time with reasonable effort. It primarily includes cases where aerodynamics or physics were studied but not fully understood and cases of highly unusual sensor input, unanticipated situations, or scenarios created by fault situations. It could be argued that with infinite resources, all of these *could have* been known, such as by performing more wind tunnel testing, more simulation, more analysis, deeper fault level scenario study, or longer and more robust sensor characterization. A subjective evaluation of the question "Could/should it have been reasonably known?" within reasonable project constraints is provided here. This may be used as a rough level-of-risk measure for the unplanned and unexpected and should be assessed in relation to software criticality and backup options.

3.0 Historic Incidents

A dataset consisting of 47 software failure incidents primarily within aerospace along with a representative few in the medical and commercial industries is presented. Note again these

incidents are discussed through a software perspective only as software failures, although the root-cause of the incidents may be attributable otherwise. Here, a software failure is considered to be where the software or automation behaved unexpectedly and could have been corrected within the software to achieve a different outcome. Table 1 shows a breakdown of studied incidents by industry. Eighty-seven percent (87%) of these incidents are in aerospace (spacecraft, aircraft, launch vehicle, missile), with others included as well-known representative software incidents in medical, commercial, or utility systems. As shown in Table 1, over half of the dataset consists of spacecraft. Spacecraft and launch vehicles combined comprise two-thirds of the incidents. Table 2, in Section 3.2, shows the classifications of each incident against each of the categories discussed in Section 2.0.

Table 1. Industry of Incidents Studied

Industry	Percent	Quantity
Spacecraft	51 %	24
Launch Vehicle	17 %	8
Aircraft	15 %	7
Missile	4 %	2
Medical	6 %	3
Commercial	6 %	3

3.1 Incident Descriptions by Year

This section lists and provides brief descriptions of software incidents organized by year. The incident number corresponds to the same number as shown in Table 2 (Section 3.2), where more information on error categorizations is provided.

1) Year: 1962
   System: Mariner 1 – Atlas-Agena Rocket
   Title: Programmer error in ground guidance veered launch vehicle off course
   Result: Loss of vehicle
   Description: Mariner 1 was launched by an Atlas-Agena rocket from Cape Canaveral's Pad 12 on 22 July 1962. Shortly after liftoff, errors in communication between the rocket and its ground-based guidance system caused the rocket to veer off course, and was destroyed by range safety. The errors were traced to two factors: (1) improper operation of beacon equipment resulting in periods of silence, and (2) a programming error (omission of a hyphen) which incorrectly accepted sweep frequency guidance signals into the program during inoperable beacon periods. This caused the computer to produce swinging steering commands sending the vehicle off course. Further documentation can be found in [2,3].
2) Year: 1965
   System: Gemini 3
   Title: Incorrect lift estimate causes short landing
   Result: Landed 84 km short, crew manually compensated to decrease short landing error
   Description: During the first manned entry on March 23, 1965, the “Molly Brown” capsule was off course, landing short due to capsule lift falling short of what was calculated in wind tunnel tests [4, p. 236]. The capsule landed 84 km short. Although wind-tunnel testing was performed and the software did not contain a “bug,” this is an example of how

a lack of understanding of the real-world environment can result in software behaving unexpectedly due to the *absence* of code.

3) Year: 1965
System: Gemini 5
Title: Data error of earth rotation lands Gemini 5 short
Result: Landed 130 km short, crew manually compensated
Description: Although the computer was operating properly, a programmer had entered the rate of the Earth's rotation as 360° per 24 hours instead of 360.98° [4, p. 262]. The crew compensated for the computing error, landing 80 miles (130 kilometers) short of the planned landing point in the Atlantic Ocean. The astronauts controlled the reentry, creating drag and lift by rotating the capsule. This is an example of erroneous data causing software misbehavior. A depiction of the crew for the Gemini 5 space flight, astronauts Charles Conrad Jr., (in water) and L. Gordon Cooper Jr. (in raft), is shown in Figure 3.

Figure 3. Gemini 5 Astronauts training for landing recovery.

(Credit: NASA)

4) Year: 1968
System: Apollo 8
Title: Memory Inadvertently Erased
Result: Close call fixed manually
Description: An inadvertent astronaut command erased computer memory causing the computer to believe the IMU was in an incorrect vehicle orientation. The crew manually corrected the orientation and computer data according to a pre-established procedure [5].

5) Year: 1969
System: Apollo 10
Title: Switch Misconfigured as bad input data to abort guidance
Result: Vehicle tumbled, recovered manually
Description: The Abort Guidance System (AGS) was inadvertently switched from HOLD ATTITUDE to AUTO, which caused the Lunar Module to look for the Command/Service Module (CSM) and tumble. The computer behaved correctly based on the erroneous data switch configuration. The Commander was able to switch the vehicle

into all manual control mode to stabilize the vehicle before losing the energy required for complete lunar ascent. Further documentation can be found in [6,7].

6) Year: 1981
System: STS-1
Title: Failure of computers to sync
Result: Launch Scrub of First Shuttle flight
Description: During the STS-1 countdown, twenty minutes prior to the first Space Shuttle flight, all computer clocks were desynchronized. This was due to programming changes 1 and 2 years prior that caused a General Purpose Computer (GPC) mismatch of time among the computers with a 1 in 67 chance of occurring, though happening that day. When asked to initiate with an incorrect start time in the past, the system set the start cycles in the future which was seen as noise by the backup computer [8].

7) Year: 1982
System: Viking-1
Title: Erroneous Command caused loss of communication
Result: Loss of Vehicle
Description: An erroneous command intended to improve battery charging inadvertently overwrote data used by the antenna pointing software and caused permanent loss of communication [9]. This is an example of an erroneous command.

8) Year: 1985-87
System: Therac-25
Title: Radiation therapy machine output lethal doses caused by user input speed
Result: Four deaths, two chronic injured
Description: Six accidents between 1985 and 1987 provided patients with massive overdoses of radiation. Because of concurrent programming errors, it sometimes gave patients radiation doses that were hundreds of times greater than normal, resulting in death or serious injury. These accidents highlighted the dangers of software control of safety-critical systems, and they have become a standard case study in health informatics, software engineering, and computer ethics. An image of the Therac-25 is shown in Figure 4, and further documentation can be found in [10].

Figure 4. Therac Radiation Therapy Machine.

(Photo Credit: The National Archives, catalog.archives.gov, NAID: 6361754).

9) Year: 1988
System: Phobos-1
Title: Erroneous unchecked uplinked command lost vehicle
Result: Loss of vehicle/Mission
Description: On 2 September 1988, an expected transmission from Phobos 1 was not received. This was traced to a faulty key-command that was sent on 28 August from ground control in Yevpatoria. A technician unintentionally left out a single hyphen in one of the keyed commands. All commands were supposed to be proofread by a computer before being transmitted, but the computer that checked commands was malfunctioning. The technician violated procedure and transmitted the command before the computer could be fixed to proofread it. This minor alteration in the command code activated unused test code and deactivated the attitude thrusters, losing sun tracking and thus depleting its batteries. Further documentation can be found in [11,12].

10) Year: 1988
System: Soyuz TM-5
Title: Wrong code executed to perform de-orbit burn
Result: Extra day in orbit, new code uplinked
Description: After undocking from Mir EP-3, the Soyuz TM-5 spacecraft deorbit engine software shut down prematurely, not completing the burn. A second attempt after engine restart behaved the same.
Ground teams discovered the computer was executing a program that was used to dock with Mir several months earlier. New software was uplinked, and the crew landed safely. Further documentation can be found in [13].

11) Year: 1991
System: Aries – Red Tigress I
Title: Bad command causes guidance error
Result: Loss of Vehicle
Description: The vehicle suffered a guidance error and was destroyed approximately 20 seconds after liftoff. Root cause was an erroneous command automatically issued by a failing/crashing VAX computer. Further documentation can be found in [14].

12) Year: 1991
System: Patriot Missile
Title: Failed target intercept due to 24-bit rounding error growth over time
Result: Failed to intercept incoming scud missile, resulting in American barracks being struck, 28 soldiers killed, 100 injured
Description: A 1970s 24-bit legacy code rounding error in time conversion led to time inaccuracies in predicting incoming missile range prediction. The truncation error grew larger the longer software was run and led to loss of precision. Time was calculated since boot, and in this case, the Patriot battery had been up approximately 100 hours which resulted in a time error of about 0.34 seconds. In trying to intercept a scud moving at 1,676 meters per second, this error placed the scud outside of the Patriot's tracking ability. An image of a patriot missile is shown in Figure 5, and further documentation can be found in [15].

Figure 5. Patriot Missile.

(Photo Credit: The National Archives, catalog.archives.gov, NAID: 6424495)

13) Year: 1992
System: F-22 Raptor

Title: Software failed to compensate for pilot-induced oscillation in presence of feedback lag
Result: Pilot killed, loss of test vehicle
Description: In April 1992 the first F-22 Raptor crashed while landing at Edwards Air Force Base, California. The cause of the crash was found to be a flight control software error that failed to prevent a pilot-induced oscillation [16].

14) Year: 1994
System: Clementine Lunar Mission
Title: Erroneous thruster firing exhausted propellant, cancelling asteroid flyby
Result: Failed mission objective
Description: An erroneous thruster firing exhausted propellant and left the spacecraft rotating at ~80 revolutions per minute (rpm), causing the cancelation of the planned asteroid flyby. The Clementine mission did successfully transmit lunar images and was able to complete its study of radiation impact on sensors and components with an alternative trajectory (passing through the Van Allen belts). Thus, two of the three main mission objectives were completed, albeit with an alternative radiation exposure profile. The mission suffered from a minimal budget as well as schedule pressure; Clementine was launched without the software being complete or tested [17,18,19].

15) Year: 1994
System: Pegasus XL STEP-1
Title: Booster loss of control due to lateral instability
Result: Loss of vehicle/Mission
Description: The control program code grossly underestimated the aerodynamic dihedral effect of high wing. Further, there was insufficient testing to reveal a faulty sideslip estimation algorithm that neglected gravitational acceleration (B. Jackson, personal communication, November 15, 2022). “Several seconds after first-stage ignition, Pegasus veered off course and lost speed, prompting the Range Safety Officer (RSO) to destroy it. The investigation revealed that the vehicle experienced an anomalous roll due to a ‘phantom yaw’ caused by an improper aerodynamics model used in the control system autopilot design” [20, p. 53].

16) Year: 1994
System: Pegasus HAPS
Title: Navigation software error prematurely shut down upper stage
Result: Unintended/low orbit
Description: The Pegasus HAPS liquid upper stage shut down about 25 seconds early due to a software navigation error, resulting in a lower-than-specified orbit. The payload was still able to provide useful data, but its lifespan was reduced by 2.5 years [20].

17) Year: 1996
System: Ariane 5 Maiden Flight
Title: Unprotected overflow in floating-point to integer conversion disrupted inertial navigation system
Result: Loss of Vehicle
Description: During launch of Ariane 5, horizontal velocity was larger than in the legacy Ariane 4 (A4). Conversion from a 64-bit floating point to scaled 16-bit integer caused overflow in reused A4 inertial navigation system alignment routine. Further, the

alignment routine continued for 40 seconds after launch per A4, but this was not required for A5 after liftoff. Identical hardware and software in redundant inertial systems both failed, leaving no other source of data. Further documentation can be found in [21,22].

18) Year: 1997
System: Pathfinder
Title: Software priority inversion caused images to stall
Result: Close Call for Mission Loss
Description: A programming error in real-time priority inheritance on mutex semaphores caused downlink of imaging to be stalled, and the computer watchdog kept resetting the computer. The error was identified and corrected using debugging features of operating system not originally planned to be used in-flight. Further documentation can be found in [23,24,25].

19) Year: 1998
System: Delta III
Title: Unanticipated 4Hz oscillation in control system led to vehicle loss
Result: Loss of vehicle
Description: In August 1998, the Delta III rocket veered off course and was destroyed by range safety 70 seconds into flight. The control software failed to recognize and correct an unanticipated 4Hz oscillating roll that developed during the first minute of flight, depleting the gimbal hydraulic fluid. Further documentation can be found in [20,26].

20) Year: 1999
System: Mars Polar Lander
Title: Premature shut down of landing engine due to misinterpretation of landing signature
Result: Loss of Vehicle/mission
Description: A “jolt” of landing micro-switches during landing gear deployment was misinterpreted as an actual landing, and engines were prematurely shut down. The software was intended to include logic that would discount touchdown indications prior to the enabling of the touchdown sensing logic, but this code was not correctly implemented. Thus, the software accepted this spurious touchdown indication as valid. A rendition of the Mars Polar Lander is shown in Figure 6, and further information can be found in [27,28].

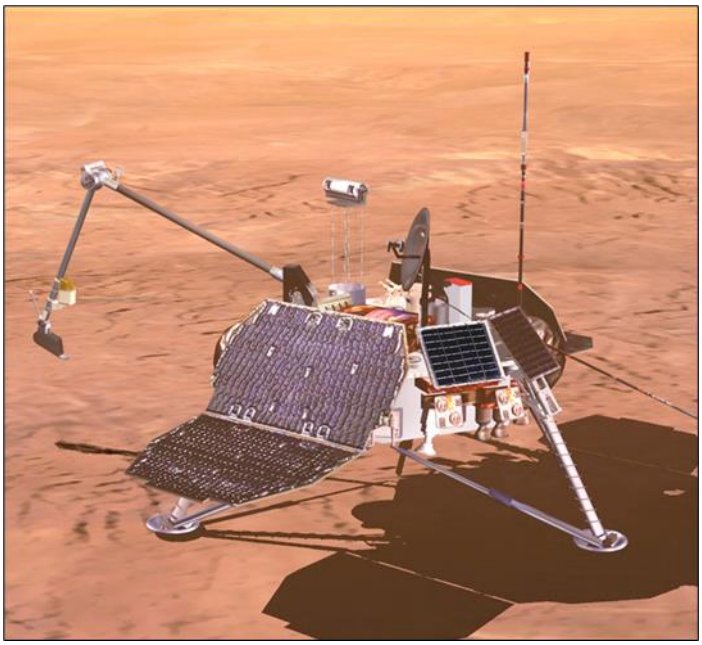


Figure 6. Mars Polar Lander.

(Credit: NASA)

21) Year: 1999
System: Mars Climate Orbiter
Title: Metric vs. imperial units error
Result: Loss of vehicle/mission
Description: Mars Climate Orbiter was lost in September 1999 because of a mismatch between measurement units in the navigation program. The spacecraft encountered Mars on a trajectory that brought it too close to the planet, and it was either destroyed in the atmosphere or escaped the planet's vicinity and entered an orbit around the Sun. An investigation attributed the failure to a measurement mismatch between two software systems: metric units by NASA and US Customary (imperial or "English") units by spacecraft builder Lockheed Martin. Further documentation can be found in [29,30].

22) Year: 1999
System: Titan IV B Centaur
Title: Programming error omitting decimal in data file caused loss of control
Result: Unintended low orbit, Milstar Satellite lost 10 days after launch
Description: This Titan IV B launch vehicle was equipped with a Centaur upper stage intended to deliver a Milstar satellite into geosynchronous orbit. After the Centaur separated from the Titan IV B, the vehicle began to experience anomalous rolls. The vehicle did not reach its intended velocity or orbit. The Milstar satellite was permanently shut down 10 days later and declared dead in orbit. During development of the Centaur computer software, a decimal point was misplaced while manually entering the roll rate filter constant in the Inertial Measurement System flight software configuration file. [20].

23) Year: 2000
System: Zenit 3SL
Title: Ground software error failed to close valve.

Result: Loss of Vehicle

Description: The Zenit-3SL's second stage shut down 80 seconds early into its planned 6.5-minute burn, and vehicle landed in ocean after 450 seconds. The launch failed due to faulty ground software not closing a valve in the rocket's second stage pneumatic system [20].

24) Year: 2001

System: Pegasus XL/HyperX Launch Vehicle / X-43A

Title: Airframe failure due to inaccurate analytical models

Result: Loss of vehicle/mission

Description: The error was caused by combination of misestimated aerodynamic characteristics and aliased solid motor organ tone appearing as significant lateral acceleration at low frequency due to improper signal filtering (B. Jackson, personal communication, November 15, 2022). "The X-43A HXLV failed because the vehicle control system design was deficient for the trajectory flown due to inaccurate analytical models (Pegasus heritage and HXLV specific), which overestimated the system margins" [31].

25) Year: 2001

System: STS-108 through 110

Title: Shuttle main engine controller mix-ratio coefficient sign-flip error

Result: Significant close call, SSME underperformance

Description: Prior to STS-108 a change had been made to the controller software coefficient for the Space Shuttle Main Engine (SSME) to compensate for an observed measurement bias in the SSME main combustion chamber pressure sensor, which controls the SSME fuel/oxidizer mixture ratio. The pressure chamber sensor was biased high causing the flight software to lower the chamber pressure by decreasing the liquid oxygen flow rate. Because of communication errors between ground systems engineers and deficiencies in the flight software verification and validation processes, the software coefficient was adjusted in the wrong direction, resulting in even larger dispersions in the mixture ratio and SSME performance. The error in the coefficient was discovered during post-flight reconstruction of the data from STS-108. The cause of the error remained unknown until after STS-110. The erroneous coefficient was flown on three consecutive flights (STS-108, STS-109, and STS-110) resulting in a slight SSME underperformance on each flight and was fixed with the proper coefficient and independent verification prior to STS-111. The error in software and resulting mixture ratio wasn't severe enough to cause any significant impacts to SSME performance, and all three flights achieved proper orbits. However, if the software error had been larger, more severe impacts to the missions and crew safety could have occurred, including a premature engine shutdown/failure resulting in on-pad or ascent abort and loss of mission. Further documentation may be found in [32,33].

26) Year: 2003

System: Multidata Systems Radiation Machine

Title: Radiation therapy machine output lethal doses caused by counterclockwise user input

Result: Many injured, 15 deaths.

Description: In a series of accidents, therapy planning software created by Multidata Systems International miscalculated the proper dosage of radiation for patients

undergoing radiation therapy leading to lethal doses. Miscalculations in dosage resulted from unexpected operator input, including graphically drawing the treatment region counterclockwise. Further documentation can be found in [34,35,36].

27) Year: 2003
System: Soyuz - TMA-1
Title: Undefined yaw value triggered ballistic reentry
Result: Landed 400 km short
Description: The problem, which caused Soyuz TMA-1 to fail-down to a re-entry in ballistic mode and land 400 km short of the intended landing site, was due to a failure in the BUSP-M guidance system, necessary to carry out a controlled re-entry. This guidance system reads gyroscopes and accelerometers and sends appropriate commands to attitude control thrusters. The yaw control channel, a sub-unit of the BUSP-M produced 'undefined' readings, indicating a malfunction. This caused higher control functions to take the BUSP-M system out of the control loop and engage ballistic re-entry mode. The radio antennae burned off so contact was only established with the crew once on the ground via hand-held radios. This was US Astronauts Don Petit's and Ken Bowersox's ISS return flight. The problem was not duplicated on ground but believed to be small timing issue. Further documentation may be found in [37,38,39].

28) Year: 2003
System: North American Electric Power Grid
Title: Software errors contribute to widespread power outage
Result: Widespread loss of power service (2 hours - 4 days)
Description: The Northeast blackout of 2003 was triggered by a local outage that went undetected and cascaded due to real-time priority inversions in monitoring software and inadequate system modeling in planning tools. The processes that annunciated alarms and provided logs to operators were "stalled", hindering situational awareness, while letting input data "pile up" until overflowing buffers, ultimately crashing the processor. Additionally, their planning tools did not accurately assess the impact of power losses. Several key lines went off-line undetected, and there was a lack of sufficient reactive power reserves contributing to cascading power loss [40].

29) Year: 2005
System: CryoSat-1
Title: Missing command causes loss of vehicle
Result: Loss of Vehicle
Description: The European Space Agency's CryoSat-1 satellite was lost in a launch failure in 2005 due to a missing shutdown command in the flight control system of its carrier rocket. The main engines in the second section of the three-stage rocket continued to burn until they had completely run out of fuel, landing the craft in the Arctic Ocean [41].

30) Year: 2005
System: Demonstration of Autonomous Rendezvous Technology (DART)
Title: Navigation software errors fail mission objectives.
Result: Loss of mission
Description: Disagreement between measured and estimated navigation positions caused repeated resets, and by using the same GPS sensor data carried over between repeated resets coupled with a guidance algorithm that performed continual course

correction, thruster firings depleted propellant and lost the mission. The algorithm was deemed overly-sensitive to divergent navigation data and contained a design flaw favoring the estimated value over the measured value, thus they would never have converged and repeatedly reset as a result. Further documentation may be found in [42,43].

31) Year: 2006
System: Mars Global Surveyor (MGS)
Title: Erroneous command led to pointing error and power/vehicle loss
Result: Premature loss of vehicle
Description: A maintenance update command sent data to the wrong location in memory, over writing communication and solar array pointing, which ultimately caused the craft to deplete power. Further documentation may be found in [44,45,46].

32) Year: 2007
System: F22 First Deployment
Title: International Date Line crossing crashed computer systems
Result: Loss of navigation & communication
Description: Multiple software-related system failures occurred when the crossing the 180th meridian, resulting in loss of navigation and communication. Clear weather permitted the squadron to pilot the aircraft manually and visually by following tanker ships back to Hawaii. Further documentation may be found in [47,48].

33) Year: 2008
System: STS-124
Title: All 4 shuttle computers fail / disagree during fueling
Result: Fueling stopped, flight delayed
Description: A cracked diode in an external sensor effectively sent each of the four primary Shuttle General Purpose Computers (GPC) a different input signal from the same sensor, causing a 1-1-1-1 disagreement among the 4 redundant computers which halted fueling and delayed the flight. Further documentation can be found in [49,50].

34) Year: 2008
System: Quantas Flight 72, Airbus A330-303
Title: Sensor input spikes caused autopilot to pitch-down, resulting in crew and passenger injuries
Result: One crew member and 11 passengers suffered serious injuries
Description: One of the aircraft’s three air data inertial reference units (ADIRU 1) exhibited a data-spike failure mode, during which it transmitted a significant amount of incorrect data to the autopilot without it being flagged invalid. The design never considered these spiked data and resulted in systems warning irregularity, including contradictory stall and overspeed warnings, and issued an uncommanded pitch down. The Australian Transport Safety Bureau (ATSB) investigation found this to be a previously unknown software design limitation of the Airbus A330's fly-by-wire flight control system software. A depiction of the stages in the flight is shown in Figure 7, and further documentation can be found in [51].

Figure 7. Quantas Flight 72.

(Credit: Masakatsu Ukon, CC BY-SA 2.0, via Wikimedia Commons https://creativecommons.org/licenses/by-sa/2.0/ https://commons.wikimedia.org/wiki/File:Qantas_Airways,_Airbus_A330-300_VH-QPA_NRT_(34167383486).jpg)

35) Year: 2008
System: B-2 Spirit - Guam crash
Title: Miscalculation with missing input data caused uncommanded pitch up
Result: Crew members successfully ejected.
Description: After three pressure transducers failed to function due to condensation inside the devices and heavy rain, the flight-control software was without all necessary information and calculated inaccurate aircraft angle-of-attack and airspeed. Once airborne and with a higher indicated speed than actual, a negative angle-of-attack was calculated, causing an uncommanded pitch up. Further documentation can be found in [52].

36) Year: 2012
System: Red Wings Flight 9268 TU-204
Title: Unanticipated landing circumstances coupled with design features resulted in crash landing
Result: 5 of 8 crewmembers killed
Description: When attempting a crosswind landing in snow, the weight-on-wheels switch failed to engage, the aircraft hydroplaned, and the reverse thruster did not deploy. As a safety feature, both sets of main landing gear were required to be compressed simultaneously before the thrust reversers could deploy. Because there was no compression of the right landing gear, the reversers were never deployed, the pilots were unaware that reverse thrusters didn't deploy, and when they moved the controls to the maximum reverse position, it caused an increase of forward thrust in both engines. In addition to the lack of reverse thrust, the airbrakes and spoilers failed to activate automatically, and the crew did not attempt to activate them manually. Further documentation can be found in [53].

37) Year: 2015
System: Airbus A400M test flight
Title: Missing software parameters during installation caused crash
Result: Four fatalities

Description: The absence of a configuration file defining critical engine parameters caused the loss of three of four engines, leading to a crash. During the final assembly process, software was incorrectly installed. Further documentation can be found in [54,55].

38) Year: 2015
System: SpaceX CRS-7
Title: Opening chutes unavailable after launch vehicle failure
Result: Possibly could have saved Dragon capsule from crash landing
Description: After launch vehicle failure, an attempt to save the Dragon vehicle by opening the chutes failed because software for handling this situation was absent from the program. Code to open the nose cone and command chutes to open were disallowed in the current state. Consequently, the vehicle was destroyed by a crash-landing into the ocean. The code was changed after the flight to handle this contingency situation [56]. A figure of the launch explosion is shown in Figure 8.

Figure 8. CRS-7 Mishap.

(Credit: NASA).

39) Year: 2016
System: Hitomi X-ray space telescope
Title: Error in computing spacecraft orientation led to spacecraft loss
Result: Loss of vehicle
Description: An error computing spacecraft orientation from gyros against a failed star-tracker led to cascading failures, including firing thrusters in the wrong direction to increase, rather than arrest, spacecraft spin. The fail-safe for the spin was also confused about orientation so was ineffective, and an erroneous command uplinked for initiating safe mode further accelerated the spin. Further documentation may be found in [57,58].

40) Year: 2017
System: SpaceX CRS-10
Title: Erroneous relative state vector transmitted to Dragon
Result: ISS rendezvous delay
Description: The Dragon spacecraft rendezvoused with the International Space Station on 22 February, but its approach was automatically aborted by an on-board

computer when a data error was reported in its navigation system. This is the first rendezvous abort by a Dragon spacecraft. The problem was traced to an incorrect data value in the spacecraft's Global Positioning System used to determine relative position to the space station. The abort resulted in a 24-hour hold on its approach. Further documentation may be found in [59,60].

41) Year: 2018, 2019
System: 737 Max crash
Title: Unanticipated software response to faulty sensor input
Result: 346 people died on two flights
Description: Erroneous input from a non-redundant faulty angle-of-attack sensor showed higher than actual angle-of-attack, causing the software to respond with a nose-down trim of the horizontal stabilizer. Handling of this erroneous sensor input was not in the software design, information about this software behavior was not generally communicated, and pilots were not trained to respond. Further documentation may be found in [61].

42) Year: 2019
System: Boeing Orbital Flight Test (OFT)
Title: Incorrect MET caused no ISS rendezvous and uncovered other latent software errors
Result: Failed ISS rendezvous, multi-year program delay
Description: An error with the Mission Elapsed Timer (MET) 31 minutes into flight, which was polled from the Atlas V booster nearly 11 hours prior to launch caused the spacecraft to burn into an incorrect orbit and use excess fuel, preventing ISS rendezvous. Investigation into the MET problem uncovered other errors which would likely have led to spacecraft loss upon return but were prevented by ground commanding enabling a safe landing (see Figure 9). NASA stated: “Breakdowns in the design and code phase inserted the original defects. Additionally, breakdowns in the test and verification phase failed to identify the defects preflight despite their detectability. While both errors could have led to risk of spacecraft loss, the actions of the NASA-Boeing team were able to correct the issues and return the Starliner spacecraft safely to Earth.” Further documentation may be found in [62,63].

Figure 9. Boeing OFT Landing.

(Credit: NASA)

43) Year: 2019
System: Beresheet
Title: Repeated reboots cause engine shutdown during lunar descent
Result: Loss of vehicle
Description: Israel's first attempt to land an unmanned spacecraft on the moon with the Beresheet was rendered unsuccessful on April 11, 2019, due to a software bug which caused repeated reboots and engine shut down, preventing it from slowing down during its final descent on the moon's surface [64].

44) Year: 2020
System: Amazon Web Service (AWS) Kinesis
Title: Maximum threads exceeded caused cascading server outage
Result: Loss of service, revenues.
Description: An upper limit on number of threads allowed by the operating system was exceeded when Amazon tried to scale up service. The exceedance caused servers to shed load, cascading to other servers. Further documentation may be found in [65,66].

45) Year: 2020
System: BD Alaris™ Infusion Pump
Title: Infusion delivery system software causes injury/death
Result: 55 injuries, 1 death
Description: Software synchronization errors led to over/under infusion, infusion delay, or infusion interruption. If a user selected two functions from the user interface within a one second interval, a system error was generated that triggered a non-silenceable high priority alarm, program operation continued, and further edits to the unit operation or programing were disallowed. The FDA issued a Class I recall on this device. Further documentation may be found in [67,68].

46) Year: 2021
System: Global Facebook Outage
Title: Bad command causes global Facebook and cascading communication outages
Result: Disrupted communication, loss of revenues
Description: During maintenance, an erroneous inquiry command accidentally disconnected Facebook data centers, leading to the deletion of routing information that disconnected Facebook and subsidiary data centers for several hours. The failure cascaded, locking out internet access to customers as well as secure access by Facebook employees. Further documentation can be found in [69,70].

47) Year: 2021
System: International Space Station (ISS)
Title: Uncontrolled ISS attitude spin from erroneous thruster firing software
Result: Close call
Description: The ISS experienced an uncontrolled spin event caused by erroneous Nauka module thruster firing. “Due to a short-term software failure, a direct command was mistakenly implemented to turn on the module's engines for withdrawal” [71]. Thrusters on the ISS Service and Progress modules were used to compensate, and once propellant was exhausted, control was restored.

3.2 Categorizations of Incidents

Table 2 shows a categorization of the 47 historical incidents enumerated in the previous section. The ID of each incident corresponds with its number in the previous section. Each incident was subjectively evaluated against the categories discussed in Section 2.0 – Erroneous vs Fail-Silent, likelihood of recovering from reboot, whether there was missing code, where the source of the error was within the software architecture, and if this could be considered an "unknown-unknown."

Table 2. Incident Categorization

ID	System	Erroneous Output or Fail-Silent?	Reboot-Recover-able?	Missing Code?	Error Location	Unknown-Unknown?
1	Mariner 1 Mission – Atlas-Agena	Erroneous Output	No	No	Code/Logic	No
2	Gemini 3	Erroneous Output	No	Yes	Code/Logic	Yes
3	Gemini 5	Erroneous Output	No	No	Data	No
4	Apollo 8	Erroneous Output	No	No	Command Input	No
5	Apollo 10	Erroneous Output	No	No	Data	No
6	STS-1	Fail Silent	Yes	Yes	Code/Logic	No
7	Viking-1	Erroneous Output	No	No	Command Input	No
8	Therac-25	Erroneous Output	No	No	Code/Logic	No
9	Phobos-1	Erroneous Output	No	No	Command Input	No
10	Soyuz TM-5	Erroneous Output	No	No	Code/Logic	No
11	Aries - Red Tigress I	Erroneous Output	No	No	Sensor Input	No
12	Patriot Missile	Erroneous Output	Yes	No	Code/Logic	No
13	F-22 Raptor	Erroneous Output	No	Yes	Sensor Input	Yes
14	Clementine Lunar Mission	Erroneous Output	No	No	Code/Logic	No
15	Pegasus XL STEP-1	Erroneous Output	No	Yes	Code/Logic	Yes
16	Pegasus HAPS	Erroneous Output	No	Yes	Code/Logic	No
17	Ariane 5 Maiden Flight	Erroneous Output	No	No	Code/Logic	No
18	Pathfinder	Erroneous Output	No	No	Code/Logic	No
19	Delta III	Erroneous Output	No	Yes	Code/Logic	Yes
20	Mars Polar Lander	Erroneous Output	No	Yes	Sensor Input	No
21	Mars Climate Orbiter	Erroneous Output	No	No	Data	No
22	Titan IV B Centaur	Erroneous Output	No	No	Data	No
23	Zenit 3SL	Erroneous Output	No	No	Code/Logic	No
24	Pegasus XL/HyperX Launch Vehicle / X-43A	Erroneous Output	No	Yes	Code/Logic	Yes
25	STS-108 through 110	Erroneous Output	No	No	Data	No
26	Multidata Systems Radiation Machine	Erroneous Output	No	No	Code/Logic	No
27	Soyuz - TMA-1	Erroneous Output	No	No	Code/Logic	No
28	North American Electric Power Grid	Fail Silent	Yes	No	Code/Logic	No
29	CryoSat-1	Erroneous Output	No	Yes	Code/Logic	No
30	DART (Demonstration of Autonomous Rendezvous Technology)	Erroneous Output	No	No	Code/Logic	No
31	Mars Global Surveyor (MGS)	Erroneous Output	No	No	Code/Logic	No
32	F22 First Deployment	Fail Silent	No	Yes	Code/Logic	No
33	STS-124	Erroneous Output	No	Yes	Sensor Input	No

34	Quantas Flight 72, Airbus A330-303	Erroneous Output	No	Yes	Sensor Input	Yes
35	B-2 Spirit -Guam crash	Erroneous Output	No	Yes	Sensor Input	Yes
36	Red Wings Flight 9268 TU-204 crash	Erroneous Output	No	Yes	Code/Logic	Yes
37	Airbus A400M test flight	Erroneous Output	No	No	Data	No
38	SpaceX CRS-7	Erroneous Output	No	Yes	Code/Logic	No
39	Hitomi X-ray space telescope	Erroneous Output	No	No	Code/Logic	No
40	SpaceX CRS-10	Erroneous Output	No	No	Data	No
41	737 Max crash	Erroneous Output	No	Yes	Sensor Input	Yes
42	Boeing Orbital Flight Test (OFT)	Erroneous Output	No	No	Code/Logic	No
43	Beresheet	Fail Silent	No	No	Code/Logic	No
44	Amazon Web Service (AWS) Kinesis	Fail Silent	No	Yes	Code/Logic	No
45	BD Alaris™ Infusion Pump	Erroneous Output	No	No	Code/Logic	No
46	Global Facebook Outage	Fail Silent	Yes	No	Command Input	No
47	ISS	Erroneous Output	No	No	Code/Logic	No

4.0 Results

4.1 Erroneous vs. Fail-Silent

Using the data from Table 2, Table 3 shows the tabulated results in terms of number of incidents and percent over the dataset of software errors manifesting as unexpected/erroneous behavior versus failing silent, producing no output. Erroneous output was over seven times as likely, 87% of the cases. Critical systems should take the substantially greater likelihood of erroneous behavior into account when considering and designing for fault tolerance. Based on this, the system's operation should be evaluated with the following questions in mind, "What would the impact be if the software behaved unexpectedly at this moment?," "What is the risk of that happening?," "Should/could the erroneous output risk be mitigated" and if so, "How?"

Table 3. Manifestation – Fail-Silent or Erroneous?

Manifestation: Erroneous Output or Fail-Silent?	Number of Incidents	Percent
Erroneous Output	41	87 %
Fail-Silent	6	13 %

4.2 Reboot Recoverability

Table 4 summarizes the subjective reboot recoverability likelihood comparing erroneous output cases and fail-silent cases. Shown here, 98% of the erroneous output cases were deemed not correctable by reboot, with only 2% (the single erroneous output case for the Patriot Missile) recoverable by reboot. Reboot recoverability is ineffective for almost all erroneous output cases. Fail-silent cases showed a greater chance of reboot recoverability over a small dataset of 6 cases with half of those deemed recoverable. This implies that reboot is also not always an effective strategy to clear fail-silent situations. Perhaps depending upon criticality, an alternate backup

mitigation approach besides rebooting should be considered. Overall, reboot only was deemed effective for 4 out of 47 incidents, independent of manifestation, or less than 9% of the cases.

Table 4. Reboot Recoverability

Manifestation Recoverability with Reboot	Recoverable with Reboot	Not Recoverable with Reboot
Erroneous Output (41 incidents)	2 %	98%
Fail Silent (6 incidents)	50 %	50%

4.3 Absence of Code

Table 5 indicates that an interestingly large 36% percent of these incidents were the result of the absence of code, as opposed to wrong code, albeit in hindsight. The absence of code satisfies the question, "Could/should software have been added to correct this incident?," and is subjective, but includes causes such as missing requirements, incomplete understanding or modeling of the real world, and unexpected inputs. This result should influence software test planning. For example, a proportionate amount of requirements verification and unit testing should be performed on the code that exists, but a percentage of the testing should also be reserved for off-nominal cases and unexpected input scenarios, possibly exposing some of the code that is lacking.

Table 5. Absence of Code

Could incident have been corrected by *adding* code?	Yes, Percent	No, Percent
Missing Code?	36 %	64%

4.4 Error Location

Table 6 categorizes the location of the error within the software architecture. A discussion of why these particular categories were chosen is given in Section 4.4, but they were mainly differentiated because mitigating errors between these categories is normally done with different methods, testing, processes, and procedures. Unsurprisingly, the majority of these errors were found to be within the code and logic itself since this category includes not only faulty code, but also missing requirements and "unknown-unknowns." Uncovering missing code during earlier phases such as unit testing or requirements verification may be a challenge, but missing code could possibly be exposed during integration testing, hardware-in-the-loop-testing, and especially with off-nominal scenario testing. Aside from missing code, code/logic errors could be exposed through focused peer reviews and comprehensive unit testing. Misconfigured data alone caused 15% of these errors. To combat data misconfiguration errors, special testing should be performed to assure that configurable data are validated prior to flight and reviewed by system experts, even if the software itself does not change. Input data, sensor, and command input combined accounted for approximately 25% of all errors. While handling input could also be considered part of coding/logic, it is useful to break this out knowing that comprehensive and off-nominal input testing could be employed to uncover errors in this part of the code. Randomized input could be computer-generated to assure robustness to unexpected input. For sensor data, actual sensor hardware should be used to "test like you fly" rather than simulating sensor input. For command input errors, operational procedures should be put in place and safeguards followed to validate commands prior to issue.

Table 6. Software Architectural Error Location

Error location	Percent
Code - Code/Logic/Algorithm	62 %
Data - Data Misconfiguration	15 %
Sensor Input– Unanticipated/Erroneous Sensor Input	15 %
Command Input - Operator/Procedure Error	9 %

4.5 Unknown-unknowns

Characterizing unknown-unknowns is highly subjective and can be a controversial topic. However, considering and designing for "unknown-unknowns" has been common aerospace practice. It could be argued that given enough time and resources, each of these incidents could have been known a priori, so a subjective reasonability test was considered against each incident to distinguish "should or could this have been known within reasonable project constraints" versus, "the project did everything they should have, yet an unknown situation led to unexpected software behavior." Unknown-unknowns include cases of unknown aerodynamics after modeling, highly unusual sensor behavior, or behavior in the presence of unlikely fault situations. Given this subjectivity, the percent of these incidents that could be considered "unknown-unknowns" is arguably and conservatively 20% (see Table 7). If one-fifth of software errors are due to things reasonably unknowable, this alone could give credence to the consideration for erroneous software backup strategies in safety-critical applications. Overall strategies to mitigate the risk of software failing during operations due to unknown-unknowns or other software failures are usually time-criticality dependent, but generally include manual human-in-the-loop control, employing dissimilar backup systems, run-time monitoring and response systems, computer reboot, entering a safe mode, or time-permitting, software reload.

Table 7. Unknown-unknowns

Predictability of Manifestation?	Percent
Unknown unknowns	20 %
Reasonably Could have known	80 %

5.0 Conclusions

This paper introduced a dataset of aerospace incidents involving software since the advent of computerized automation. It analyzed aerospace failures through the eyes of the software and automation discipline in an effort to characterize and predict trends in software behavior (and misbehavior) as a design and test aid to current and future aerospace systems. It characterized how software is most likely to fail—erroneously or silent—and determined that automation fails erroneously much more often than simply "crashing" or ceasing to output. Systems should recognize the relative risk and be designed accordingly. Rebooting the software, though used prevalently, was found to be largely ineffective to clear software failures, effective in less than 9% of the total cases, and less than 2% for the erroneous-output case. This paper explored software errors relating to the absence of code as well as the prevalence of unknown-unknowns, both of which were substantial constituents in the dataset, 36% and 20% respectively. Software testing should be planned to uncover missing code through off-nominal input and integrated testing, and backup systems should be considered to mitigate the risk of "unknown-unknowns" in safety-critical systems. Finally, a categorization determining what location within the software

architecture (code, data, sensor input, or command input) was provided to better influence processes and testing related to those areas both during development and during operations.

The dataset presented here is rich for further study, especially in the areas of backup systems, relationship to common-cause, and manual control for safety-critical systems. Some key questions such as, "Was this a multi-string common-cause failure?," "Was a manual or automated backup system used?," "Would a backup system have helped?," "If so, what kind of a backup system could have helped?" could be explored. Would a human-in-the-loop, a dissimilar backup, a monitor system, or no backup at all be the best option for each situation?

Another key question could be, "What was the root cause of this error?" Looking at how these errors might have been avoided altogether has great merit and projects may better focus on areas most likely to catch root cause software errors. Since the software performed exactly as programmed in these cases, exploring "why" it was programmed the way it was in terms of root-cause may be a lesson to organizations producing software.

"In what phase of the project could/should have this incident been discovered and averted?" is another interesting question. How much testing and what type of testing would have provided the most "bang for the buck" in averting these errors? All of these questions would be useful follow-on work against this, hopefully stagnant, yet continually growing, dataset.

6.0 References

[1] Zak, D., "'Nothing ever ends': Sorting through Rumsfeld's knowns and unknowns," The Washington Post, July 1, 2021.

[2] NASA, "Mariner 1," *NASA Space Science Data Coordinated Archive* [online database], URL: https://nssdc.gsfc.nasa.gov/nmc/spacecraft/display.action?id=MARIN1 [retrieved 11 Oct. 2022].

[3] NASA Jet Propulsion Laboratory, "Mariner-Venus 1962 Final Project Report," NASA SP-59, July 1965, URL: https://history.nasa.gov/SP-59.pdf [retrieved 11 Oct. 2022].

[4] Hacker, B., and Grimwood, J., *On the Shoulders of Titans*, NASA SP-4203, Washington D.C., 1977, URL: https://history.nasa.gov/SP-4203.pdf [retrieved 11 Oct. 2022].

[5] NASA, "Apollo 8 Mission Report," MSC-PA-R-69-1, Manned Spacecraft Center, Houston, TX, February 1969, URL: https://history.nasa.gov/afj/ap08fj/pdf/a08-missionreport.pdf.

[6] NASA, Apollo 10 Mission Report, MSC-00126, Manned Spacecraft Center, Houston, TX, August 1969.

[7] Cernan, G., "Interview on Apollo 10 - (December 23, 2009),", URL: https://www.youtube.com/watch?v=fsObsxU08ys [retrieved 12 Oct. 2022].

[8] Garman, J.R., "The "BUG" heard 'round the world: discussion of the software problem which delayed the first shuttle orbital flight," ACM SIGSOFT Software Engineering Notes, Volume 6, Issue 5, October 1981, pp. 3-10, URL: https://doi.org/10.1145/1005928.1005929 [retrieved 14 Oct. 2022].

[9] Mudgway, D. J., "Telecommunications and Data Acquisition Systems Support for the Viking 1975 Mission to Mars", NASA JPL Publication 82-107, May 15, 1983.

[10] Levenson, N.G. and Turner, C.S., "An Investigation of the Therac-25 Accidents," UCI Technical Report 92-108, Information and Computer Science Department, University of

California, Irvine, CA, URL: https://escholarship.org/uc/item/5dr206s3 [retrieved 14 Oct. 2022].
[11] Waldrop, M., "Phobos at Mars: A Dramatic View - and Then Failure," Science, Vol 245, 8 September 1989, p. 1045.
[12] Katell, A., "Soviet Mars Probe Lost in Space Because of Controller's Error," Associated Press, 9 Sept. 1988.
[13] Harland, D.M., "The Story of Space Station Mir," Springer-Verlag, ISBN 978-0-387-23011-5, 2005.
[14] Applied Technology Associates, Inc., and Dillow, M., "Red Tigress Mission Report," Albuquerque, New Mexico, 1991, URL: https://apps.dtic.mil/sti/pdfs/ADA338873.pdf [retrieved on 12 Oct 2022].
[15] United States General Accounting Office, Report to the Chairman, Subcommittee on Investigations and Oversight, Committee on Science, Space, and Technology, House of Representatives, "Patriot Missile Defense, Software Problem Led to System Failure at Dhahran, Saudi Arabia," Feb. 1992, URL: https://www-users.cse.umn.edu/~arnold/disasters/GAO-IMTEC-92-96.pdf [retrieved on 14 Oct. 2022].
[16] Dornheim, M.A., "Report Pinpoints Factors Leading to YF-22 Crash," Aviation Week and Space Technology, November 9, 1992.
[17] Regeon, P. A., Chapman, R. J., Baugh, R., "Clementine, The Deep Space Program Science Experiment," Acta Astronautica Vol. 35, Suppl., pp. 307-321, 1995.
[18] Horan, Donald M. and Berkowitz, Bruce D., "Clementine," in Reducing Space Mission Cost, edited by Wertz, James R. and Larson, Wiley J., Space Technology Library, Microcosm Press, Torrance, CA, 1996.
[19] Committee on Planetary and Lunar Exploration (COMPLEX) Space Studies Board Commission on Physical Sciences, Mathematics, and Applications, National Research Council, "Lessons Learned from the Clementine Mission," 1997.
[20] NASA, "Mission Success of U.S. Launch Vehicle Flights from a Propulsion Stage-based Perspective 1980-2015," NASA TM-2017-219497, 2017.
[21] Le Lann, Gérard, "An Analysis of the Ariane 5 Flight 501 Failure – A System Engineering Perspective," Proceedings of the 1997 international conference on Engineering of computer-based systems (ECBS'97). IEEE Computer Society, pp. 339–346, 1997.
[22] Dowson, M., "The Ariane 5 Software Failure," ACM SIGSOFT Software Engineering Notes. 22 (2): 84, 1997.
[23] Reeves, G.E., Re: What Really Happened on Mars? Risks Forum 19(54) (1998)
[24] McHale, J., "NASA tackles Pathfinder software glitch," Military & Aerospace Electronics Magazine, August 31, 1997.
[25] Durkin, T., "What the Media Couldn't Tell You About Mars Pathfinder," Robot Science and Technology Magazine, Issue 1, 1998.
[26] Boeing, URL: https://boeing.mediaroom.com/1998-10-15-Boeing-Changes-Delta-III-Control-Software [retrieved 12 Oct 2022].
[27] JPL Special Review Board, Report on the Loss of the Mars Polar Lander and Deep Space 2 Missions, JPL D-18709, 2000.
[28] Reichhardt, T. Software error 'caused Mars lander crash', Note in Nature Vol.404, 423 (2000).

[29] Stephenson, A.G.; LaPiana, L.S.; Mulville, D. R.; Rutledge, P.J.; Bauer, F.H.; Folta, D.; Dukeman, G.A.; Sackheim, R.t; Norvig, P., Mars Climate Orbiter Mishap Investigation Board Phase I Report, NASA, 1999.
[30] Oberg, James, "Why the Mars Probe went off course," IEEE Spectrum. IEEE, 1999.
[31] X-43A Mishap Investigation Board, "Report of Findings X-43A Mishap," Accepted Draft 9/6/02, URL: https://www.nasa.gov/pdf/47414main_x43A_mishap.pdf [retrieved 15 Nov 2022].
[32] Space Shuttle Program Close Call Awareness Papers, "STS-108: MCC Pc Calibration Error," STS 1 -108_CC_paper_(MCC_Pc_Calibration)_07_20_06, URL: https://smasp.jsc.nasa.gov/na/na13/SII-2015-001/Document%20Library/1/SigIncProto%20Drop%204%20FY15%20Final/Interactive%20Sig%20Inc%20Prototype/Space%20Shuttle/STS-108/STS-108_CC_paper_(MCC_Pc_Calibration)_07_20_06.pdf [retrieved 15 Nov 2022].
[33] JSC SMA Flight Safety Office, "Significant Incidents & Close Calls in Human Spaceflight," URL: https://sma.nasa.gov/SignificantIncidents/ [retrieved 15 Nov 2022].
[34] International Atomic Energy Agency, A Panel of Experts (2001), "Investigation of an Accidental Exposure of Radiotherapy Patients in Panama," /Report of a Team of Experts, 26 May – 1 June, 2001, Vienna, Austria: International Atomic Energy Agency/, URL: https://www-pub.iaea.org/MTCD/publications/PDF/Pub1114_scr.pdf [retrieved 15 Nov 2022].
[35] Garfinkel, Simson, "History's Worst Software Bugs," Wired.com, November 8, 2005. URL: https://lessons-from-history.com/Presentations/Articles/History's%20Worst%20Software%20Bugs.pdf [retrieved 15 Nov 2022].
[36] Schmid, Randolph E., "Decree Blocks Firm on Radiation Devices," AP News, May 7, 2003, URL: https://apnews.com/article/0e24f8dd5444a93fcc6c8d2a7dbf010b [retrieved 15 Nov 2022].
[37] The European Space Agency, "New Soyuz TMA spacecraft cleared for next mission with ESA," URL: https://www.esa.int/Science_Exploration/Human_and_Robotic_Exploration/Cervantes_Mission/New_Soyuz_TMA_spacecraft_cleared_for_next_mission_with_ESA_astronaut [retrieved 15 Nov 2022].
[38] Spaceref Editor, "RSC Energia Report on Cause of Soyuz TMA-1 Reentry Problems," 28 May 2003, URL: https://spaceref.com/press-release/rsc-energia-report-on-cause-of-soyuz-tma-1-reentry-problems/ [retrieved 15 Nov 2022].
[39] Spaceref Editor, "RSC Energia: Technical Board reviewing the causes of ballistic mode reentry of Soyuz TMA-1," 16 May 2003, URL: https://spaceref.com/status-report/rsc-energia-technical-board-reviewing-the-causes-of-ballistic-mode-reentry-of-soyuz-tma-1/ [retrieved 15 Nov 2022].
[40] U.S.-Canada Power System Outage Task Force, "Final Report on the August 14, 2003 Blackout in the United States and Canada: Causes and Recommendations," April 2004.
[41] Peplow, M., "CryoSat mission lost", Nature, 10 October 2005.
[42] NASA, "Overview of the DART Mishap Investigation Results," URL: https://www.nasa.gov/pdf/148072main_DART_mishap_overview.pdf, [retrieved 15 Nov 2022].

[43] SpaceNewsEditor, “Multiple Errors Caused DART Rendezvous Mission Mishap,” SpaceNews.com, June 29, 2004. URL: https://spacenews.com/multiple-errors-caused-dart-rendezvous-mission-mishap/ [retrieved 15 Nov 2022].
[44] Minkel, J.R., “Human Error Caused Mars Global Surveyor Failure,” Scientific American, 13 Apr 2007., URL: https://www.scientificamerican.com/article/human-error-caused-mars-g/ [retrieved on 15 Nov 2022].
[45] NASA, “Mars Global Surveyor (MGS) Spacecraft Loss of Contact,” 13 Apr 2007, URL: https://www.nasa.gov/sites/default/files/174244main_mgs_white_paper_20070413.pdf [retrieved on 15 Nov 2022].
[46] NASA Public Lessons Learned System, Lesson Number 1805, Lesson Date 2007-09-03, “Mars Global Surveyor (MGS) Spacecraft Loss of Contact,” URL: https://llis.nasa.gov/lesson/1805 [retrieved on 15 Nov 2022].
[47] Thompson, M., “A $330 Million Case of Jet Lag,” TIME, 07 Mar 2007, URL: http://content.time.com/time/nation/article/0,8599,1597043,00.html [retrieved 18 Nov 2022].
[48] Staff, “F-22 Squadron Shot Down by the International Date Line,” Defense Industry Daily, 01 Mar 2007, URL: https://www.defenseindustrydaily.com/f22-squadron-shot-down-by-the-international-date-line-03087/ [retrieved 18 Nov 2022].
[49] Bergin, C., “STS-124: FRR debate outstanding issues – faulty MDM removed,” NASASpaceflight.com, 15 May 2008, URL: https://www.nasaspaceflight.com/2008/05/sts-124-frr-debate-outstanding-issues-faulty-mdm-removed/ [retrieved 18 Nov 2022].
[50] Bergin, C., “STS-126: Super smooth Endeavour easing through the countdown to L-1,” NASASpaceflight.com, 13 Nov 2008, URL: https://www.nasaspaceflight.com/2008/11/sts-126-endeavour-easing-through-countdown/ [retrieved 18 Nov 2022].
[51] Australian Transport Safety Bureau, “In-flight upset - Airbus A330-303, VH-QPA, 154 km west of Learmonth, WA, 7 October 2008,” Investigation Number AO-2008-070, Released 19 Dec 2011.
[52] USAF Accident Investigation Board, B-2A, S/N 89-0127, 20080223 KSZL501A, URL: https://web.archive.org/web/20160304002933if_/http://www.glennpew.com/Special/B2Facts.pdf [retrieved 18 Nov 2022].
[53] Hradecky, S., “Accident: Red Wings T204 at Moscow on Dec 29th 2012, overran runway on landing,” The Aviation Herald, 29 Dec 2012, URL: http://www.avherald.com/h?article=45b4b3cb [retrieved on 18 Nov 2022].
[54] Osborne, T., “Incorrectly Installed Engine Software Caused A400M Crash, Airbus Official Says,” Aviation Week, 27 May 2015.
[55] Chirgwin, R., “Airbus confirms software brought down A400M transport plane,” The Register, 31 May 2015, URL: https://www.theregister.com/2015/05/31/airbus_software_config_brought_down_a400m/
[56] Bergin, C., “Saving Spaceship Dragon - Software to provide contingency chute deploy,” NASA Spaceflight.com, URL: https://www.nasaspaceflight.com/2015/07/saving-spaceship-dragon-contingency-chute/ [retrieved on 22 Nov 2022].
[57] Klein, A., “Japanese satellite's death spiral linked to software malfunction,” NewScientist, 29 Apr 2016, URL: https://www.newscientist.com/article/2086422-japanese-satellites-death-spiral-linked-to-software-malfunction/ [retrieved 18 Nov 2022].

[58] Witze, A., “Software Error Doomed Japanese Hitomi Spacecraft,” Nature, 29 Apr 2016. URL: https://www.scientificamerican.com/article/software-error-doomed-japanese-hitomi-spacecraft/ [retrieved 18 Nov 2022].
[59] Richardson, D., “Dragon Rendezvous Aborted, Next Attempt in 24 Hours,” Spaceflight Insider, 22 Feb 2017, URL: https://www.spaceflightinsider.com/missions/iss/dragon-rendezvous-aborted-next-attempt-in-24-hours/ [retrieved 22 Nov 2022]
[60] Harwood, W., “SpaceX cargo ship aborts approach to station,” CBS News, 22 Feb 2017, URL: https://www.cbsnews.com/news/spacex-russian-cargo-ships-en-route-to-station/ [retrieved 22 Nov 2022].
[61] Federal Aviation Administration, “Preliminary Summary of the FAA’s Review of the Boeing 737 MAX,” Version 1, 3 Aug 2020, URL: https://www.faa.gov/news/media/attachments/737-MAX-RTS-Preliminary-Summary-v-1.pdf [retrieved 13 Nov 2022].
[62] Chang, K., “Boeing Starliner Flight’s Flaws Show ‘Fundamental Problem,’ NASA Says,” New York Times, 7 Feb 2020, URL: https://www.nytimes.com/2020/02/07/science/boeing-starliner-nasa.html [retrieved 13 Nov 2022].
[63] NASA Blogs, “NASA Shares Initial Findings from Boeing Starliner Orbital Flight Test Investigation,” 7 Feb 2020, URL: https://blogs.nasa.gov/commercialcrew/2020/02/07/nasa-shares-initial-findings-from-boeing-starliner-orbital-flight-test-investigation/ [retrieved 18 Nov 2022].
[64] Nevo, E., “What Happened to Beresheet?,” Weizmann Institute of Science Online Journal, 20 Feb 2020, [retrieved 12 Oct. 2020], URL: https://davidson.weizmann.ac.il/en/online/sciencepanorama/what-happened-beresheet
[65] Whitney, L., “Amazon reveals reason for last week’s major AWS outage,” TechRepublic, 30 Nov 2020, URL: https://www.techrepublic.com/article/amazon-reveals-reason-for-last-weeks-major-aws-outage/ [retrieved on 18 Nov 2022].
[66] Tung, L., “Amazon: Here's what caused the major AWS outage last week,” ZDNet Business, 30 Nov 2020, URL: https://www.zdnet.com/article/amazon-heres-what-caused-major-aws-outage-last-week-apologies/ [retrieved on 18 Nov 2022].
[67] Becton, Dickenson and Company (BD), “Urgent: Medical Device Recall Notification, Affected Device: BD Alaris System,” 10020 Pacific Mesa Blvd, San Diego, CA, 92121, 4 Feb 2020, URL: https://www.bd.com/documents/alerts/AlarisSystem9.x_CustomerRecallPackage.pdf [retrieved 16 Nov 2022].
[68] United States Food and Drug Administration, “Becton Dickinson (BD) CareFusion 303 Inc. Recalls Alaris System Infusion Pumps Due to Software and System Errors,” 2020 Medical Device Recalls, URL: https://www.fda.gov/medical-devices/medical-device-recalls/becton-dickinson-bd-carefusion-303-inc-recalls-alaris-system-infusion-pumps-due-software-and-system [retrieved 16 Nov 2022].
[69] Feuer, W., “Facebook and its apps start to come back online after stocks tank,” New York Post, 4 Oct 2021, URL: https://nypost.com/2021/10/04/facebook-instagram-and-whatsapp-hit-by-global-outage/ [retrieved 16 Nov 2022].
[70] Shead, S., “Facebook says sorry for mass outage and reveals why it happened,” CNBC, 5 Oct 2021, URL: https://www.cnbc.com/2021/10/05/facebook-says-sorry-for-mass-outage-and-reveals-why-it-happened.html [retrieved 16 Nov 2022].

[71] Zastrow, M., “International Space Station saved from out-of-control spin,” Astronomy, 30 Jul 2021, URL: https://astronomy.com/news/2021/07/international-space-station-saved-from-out-of-control-spin [retrieved 16 Nov 2022]

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
08/16/2023	Technical Publication	

4. TITLE AND SUBTITLE
Software Error Incident Categorizations in Aerospace

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
Prokop, Lorraine E.

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER
869021.01.23.01.01

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
NASA Langley Research Center
Hampton, VA 23681-2199

8. PERFORMING ORGANIZATION REPORT NUMBER
NESC-NPP-22-01775

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
National Aeronautics and Space Administration
Washington, DC 20546-0001

10. SPONSOR/MONITOR'S ACRONYM(S)
NASA

11. SPONSOR/MONITOR'S REPORT NUMBER(S)
NASA/TP-20230012154

12. DISTRIBUTION/AVAILABILITY STATEMENT
Unclassified - Unlimited
Subject Category Computer Programming and Software
Availability: NASA STI Program (757) 864-9658

13. SUPPLEMENTARY NOTES

14. ABSTRACT
Since the first use of computers in space and aircraft, software errors have occurred. These errors can manifest as loss-of-life or less catastrophically. As the demand for automation increases, software in safety-critical systems should be designed to be tolerant to the most likely software faults. This paper categorizes historic aerospace software errors to determine trends of how and where automation is most likely to fail.

15. SUBJECT TERMS
Software; Error; Failure; Fault-tolerance

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			STI Help Desk (email: help@sti.nasa.gov)
					19b. TELEPHONE NUMBER *(Include area code)*
U	U	U	UU	35	(443) 757-5802