Introduction

- Welcome to another “software short” video
- Todays topic is called “what is software common cause failure and “how common is it” -- that’s just what I’m going to go over
- By way of a few powerpoint slides I’ll talk about
  - What is software common cause and why even care?
  - Through a historical study statistics
    - When and How software has failed
    - Why it as a failed, and where the code it as failed (surprisingly little literature on the subjedt!)
- Why care? Software controls everything – especially when it’s safety critical
  - Really this is a talk about software/automation failures in general, especially in aerospace, a lessons learned summary on how we can do things better

Software Common Cause Failures

- What is Software "Common Cause" or "Common Mode" Failure?
  - In Hardware – multiple copies of hardware exist for fault tolerance (i.e. 3 "strings")
  - In Software, 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 – this is a "common cause failure"
    - If only one processor is used, *any* software failure is "common cause"
- Why Care?
    - A single software load is a single point of failure (zero fault tolerant)
    - In safety-critical systems, lives depend on software behaving as expected
- Consider two classes of software failures:
  - "Fail-Silent" – Computers all stop outputting, i.e "crash"
  - "Erroneous-Output" – Software/automation does the *wrong thing*
    - Both should be considered when designing for fault tolerance
- Why differentiate?
  - It is easier to determine if the software has stopped (watchdog timer)
  - How to determine if the automation is doing something *wrong*? (Human, backup/monitor software)

Software Failure Categories & Common Mitigations

- Fail-Silent Cause Examples (*no output*)
  - Operating System Halt - Memory access violations, arithmetic errors, coding/logic errors
  - CPU Hog – Process Starvation Infinite loop, priority inversion, extreme latencies
- Erroneous Output Causes Examples (*wrong output*)
  - Coding/Logic Error - Missing Requirements, unexpected situations, insufficient modeling
  - Data Parameter Error - misconfigured data, units errors, lack of precision, sign flip
  - Unanticipated / Erroneous Sensor Input
  - Erroneous Command Input - Operator / Procedural Error

- In Spaceflight, dynamic phases are riskiest due to short time-to-criticality
  - Ascent, Rendezvous, Entry, Loss/Delayed Communication
  - Common mitigations during these times
    - Dissimilar or reduced size backup software, manual piloting / human-in-the-loop
  - More options during less dynamic times - software upload, reboot, crew/ground diagnosis
  - Mitigations should be considered in relation to time and safety criticality

Key NASA Requirements for Software Fault Tolerance

- NPR 8705.2C: HUMAN-RATING REQUIREMENTS FOR SPACE SYSTEMS
  - 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.
  - 3.2.7 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.
  - 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. …

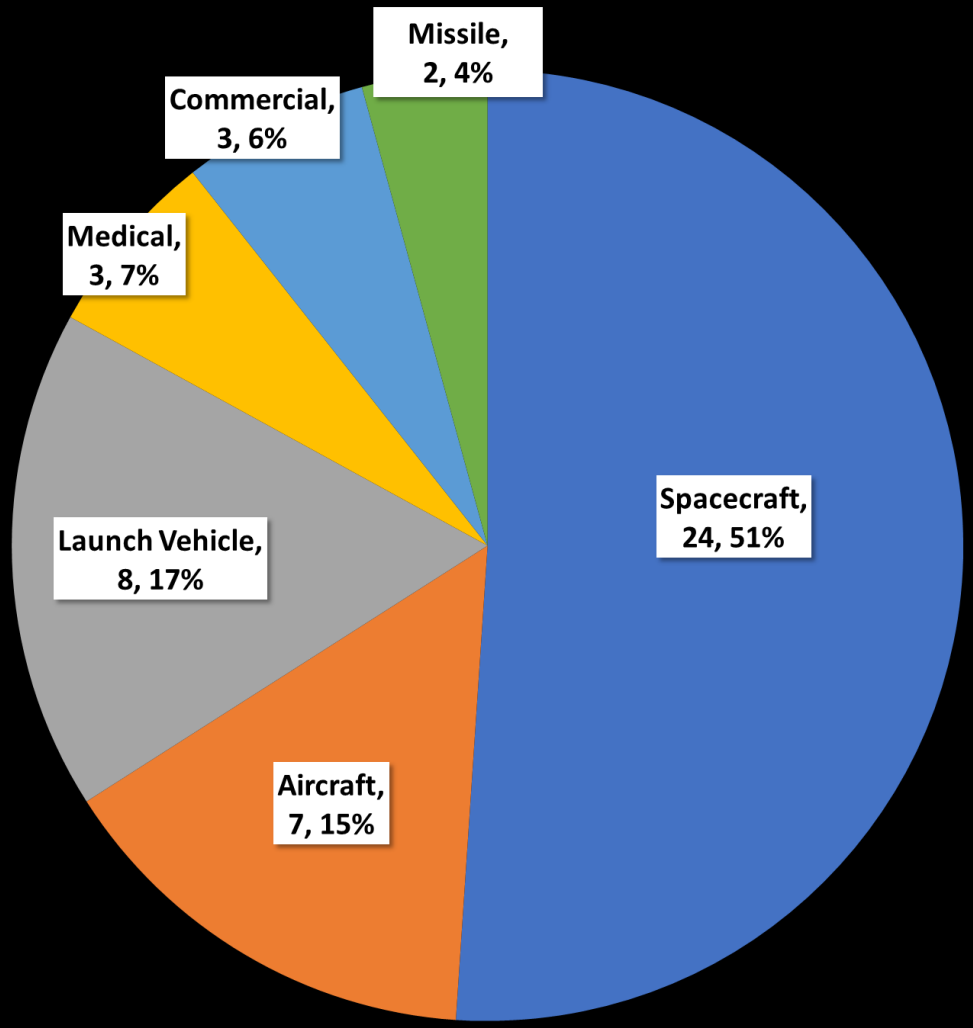
How Common Are Software Failures?

47 Historic Incidents Studied & Characterized – mainly Aerospace/DoD

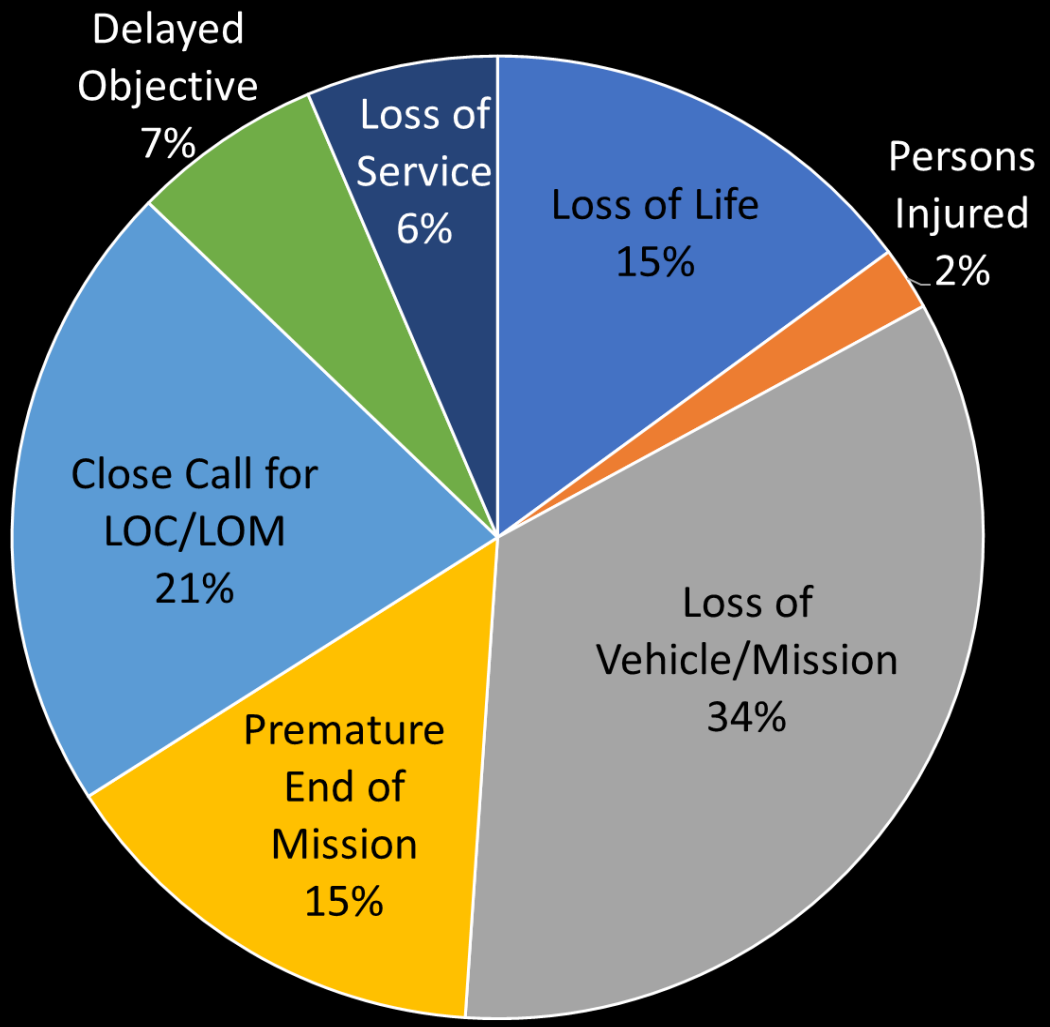
1962 Mariner 1 Mission – Atlas-Agena	1985-87 Therac-25	1994 Pegasus XL STEP-1	1999 Titan IV B Centaur	2005 CryoSat-1	2012 Red Wings Flight 9268 TU-204 crash	2019 Beresheet
1965 Gemini 3	1988 Phobos-1	1994 Pegasus HAPS	2000 Zenit 3SL	2005 DART	2015 Airbus A400M test flight	2020 Amazon Web Service (AWS)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

Dataset Industry & Impact Breakdown

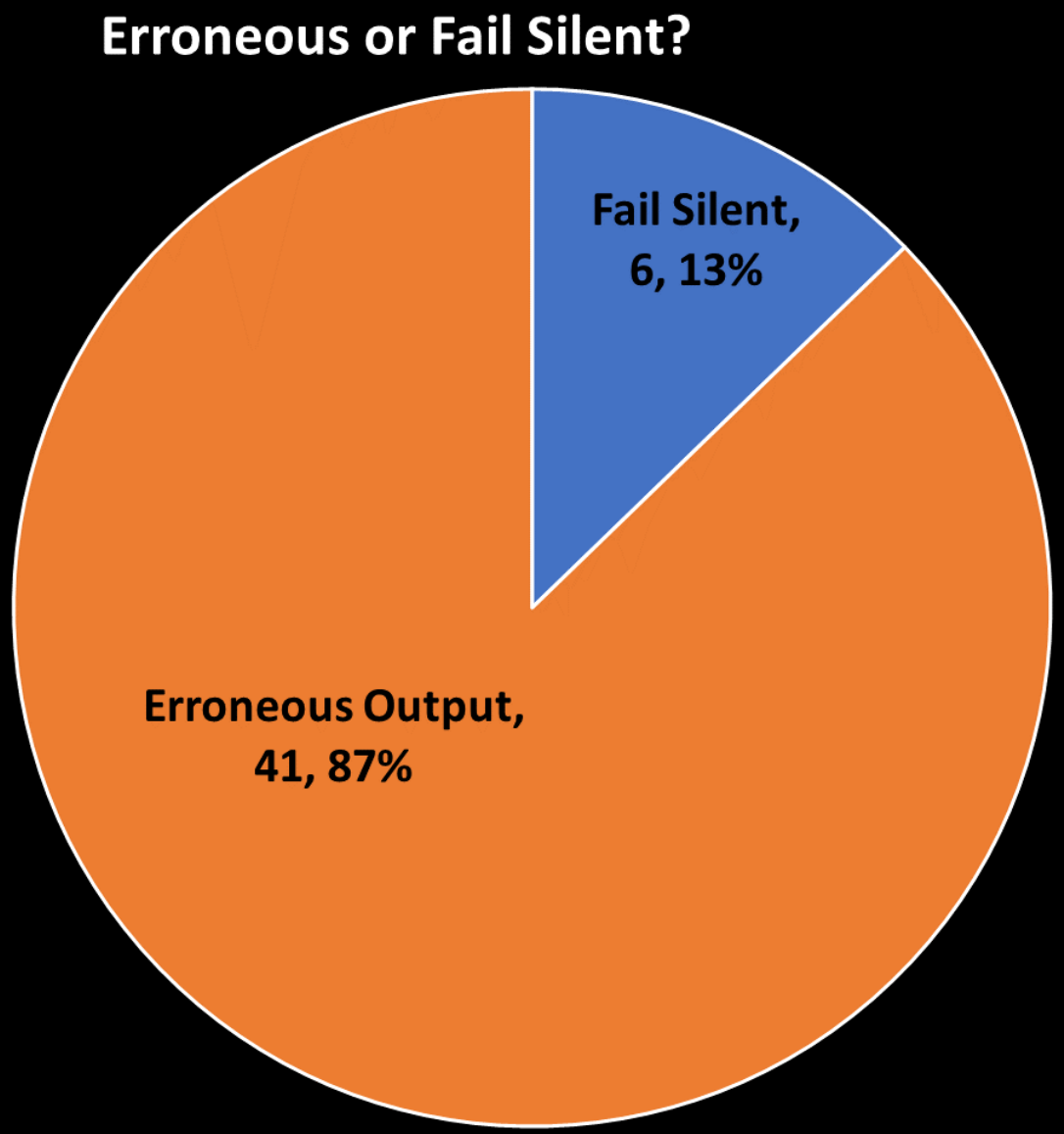
Industries in Data Set



Impact/Result of Failures in Dataset



Incident Statistics – Erroneous vs. Fail Silent



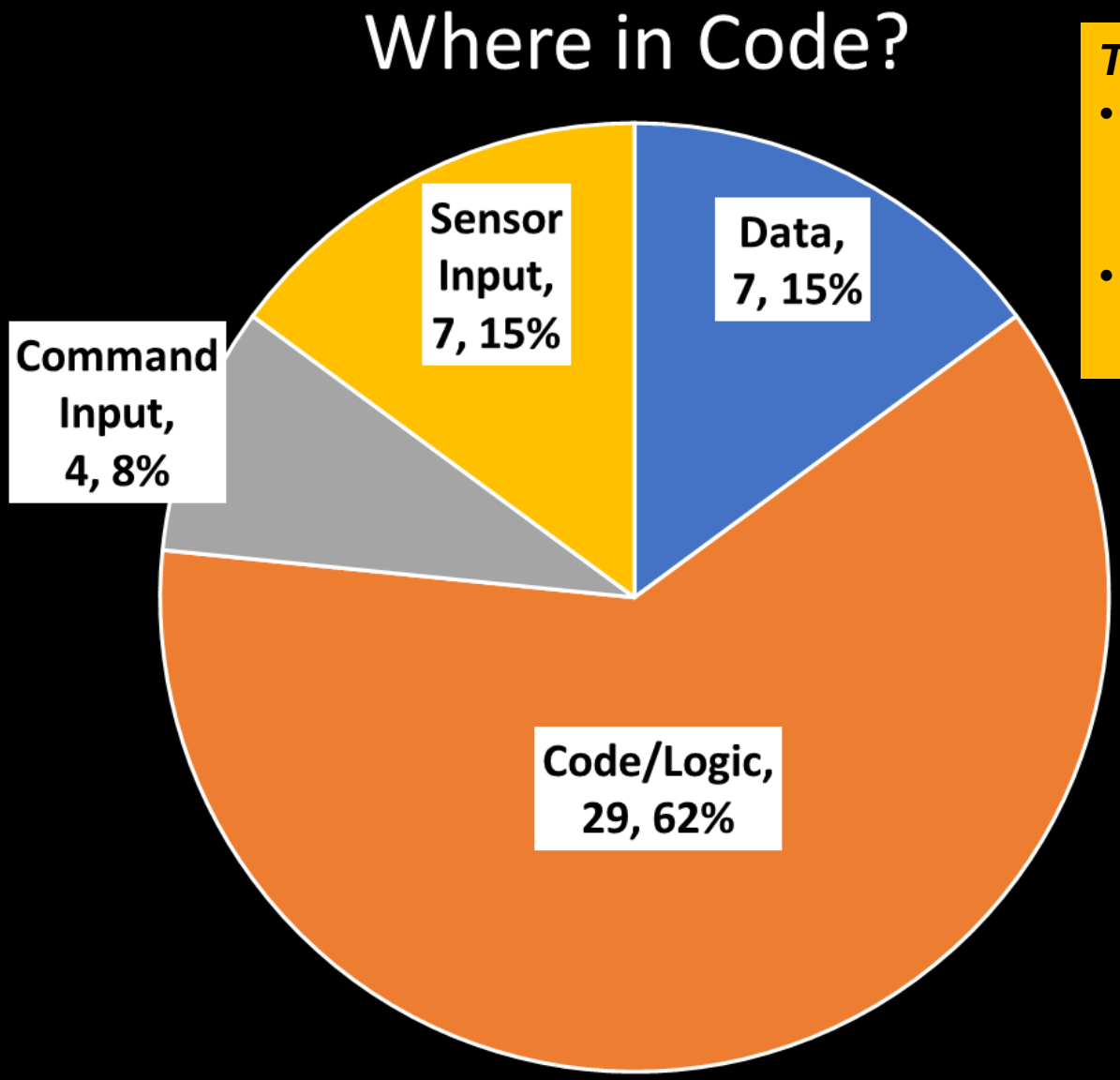
Takeaway:

- ***Historically, erroneous output situations were much more prevalent than fail-silent cases***
- ***Roughly 90-10% rule of thumb***

Fault-tolerant Design Tip:

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

Incident Statistics – Software Architecture Error Location



Takeaway(s):

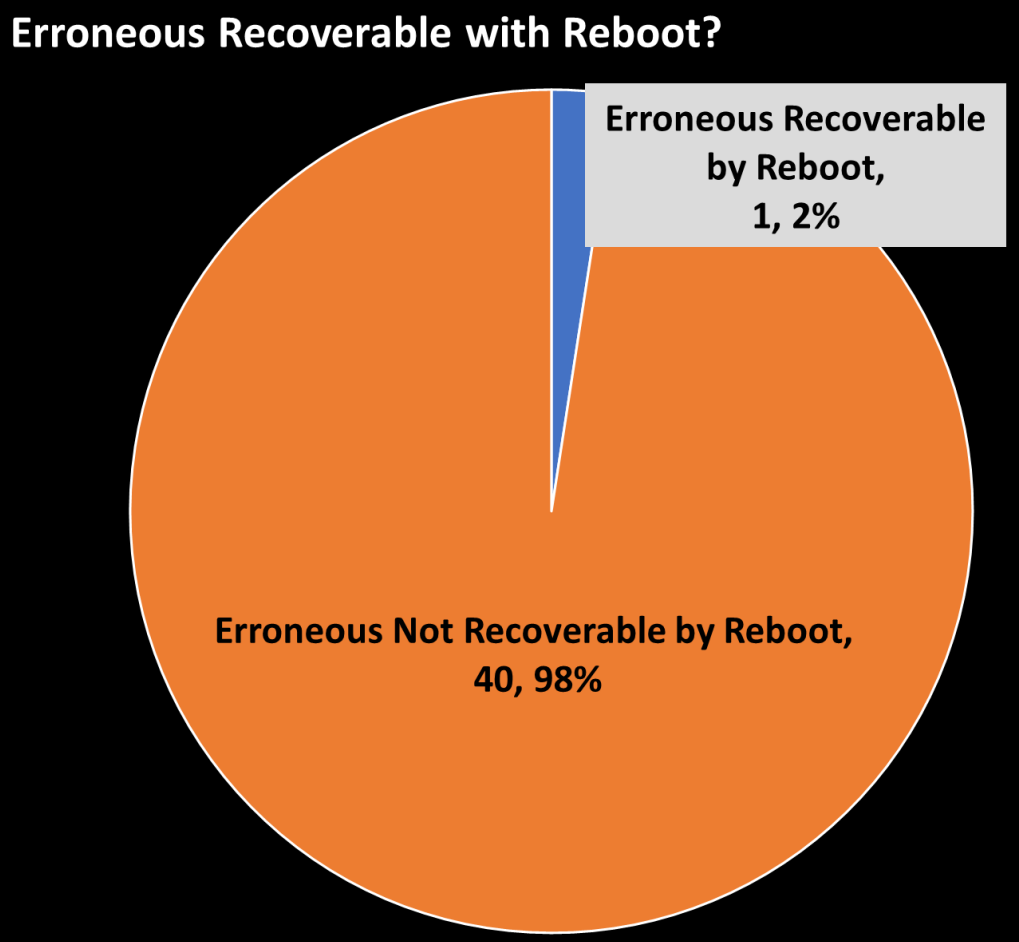
- ***Coding/logic (includes missing requirements and unknowns) and data configuration errors account for most software incidents***
- ***Input Errors – Command or Sensor Input Accounted for 23% of errors***

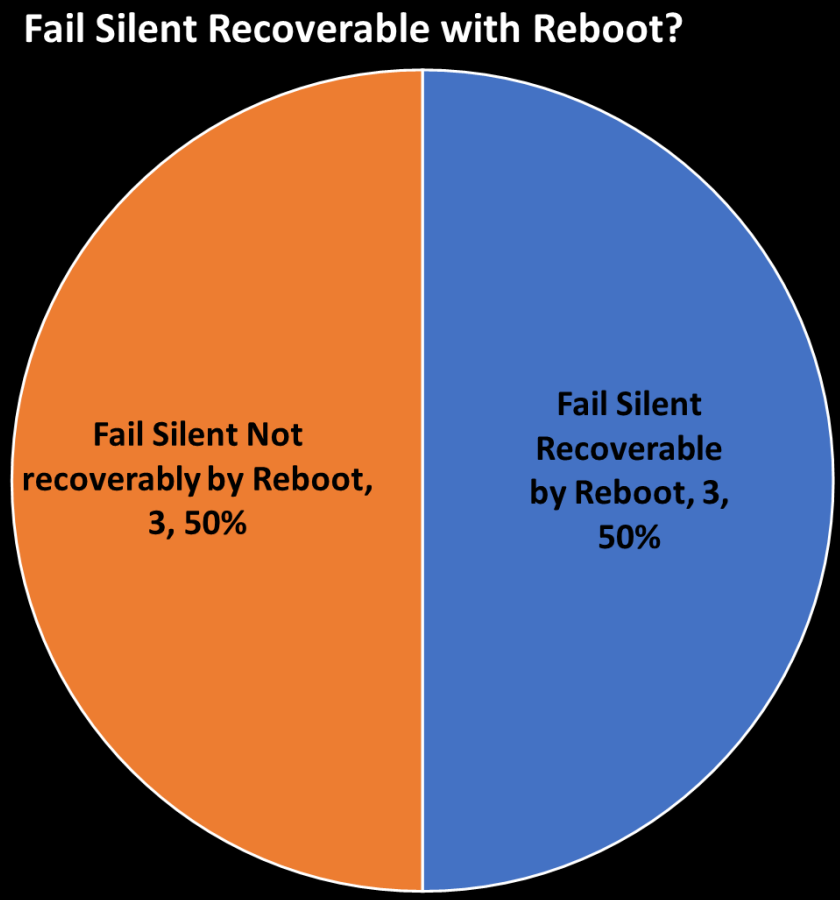
Fault-tolerant Code Tips:

- ***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***

Reboot Recoverability Likelihood
Erroneous vs. Fail Silent





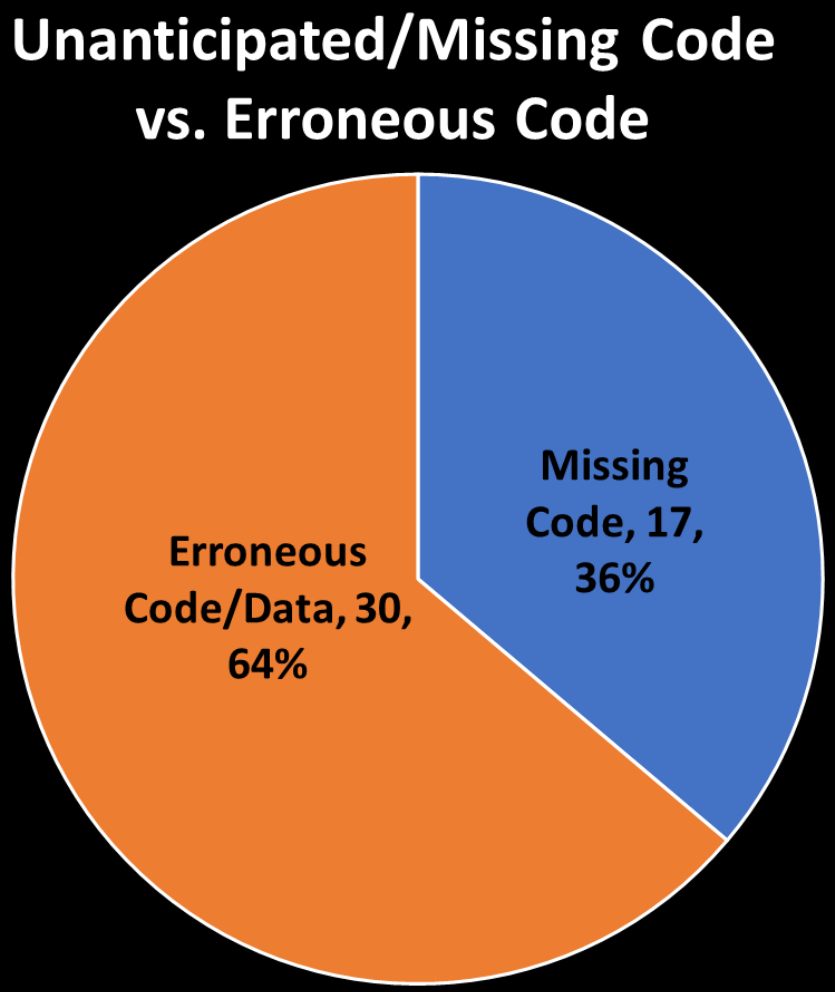
Takeaways:

- ***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***

Incident Statistics – Absence of Code



Missing Code may arise from:

- Missing requirements
- Unanticipated situation
- Insufficient understanding or modeling of real world

- Adding code could have corrected the "missing code" incidents – *in hindsight*

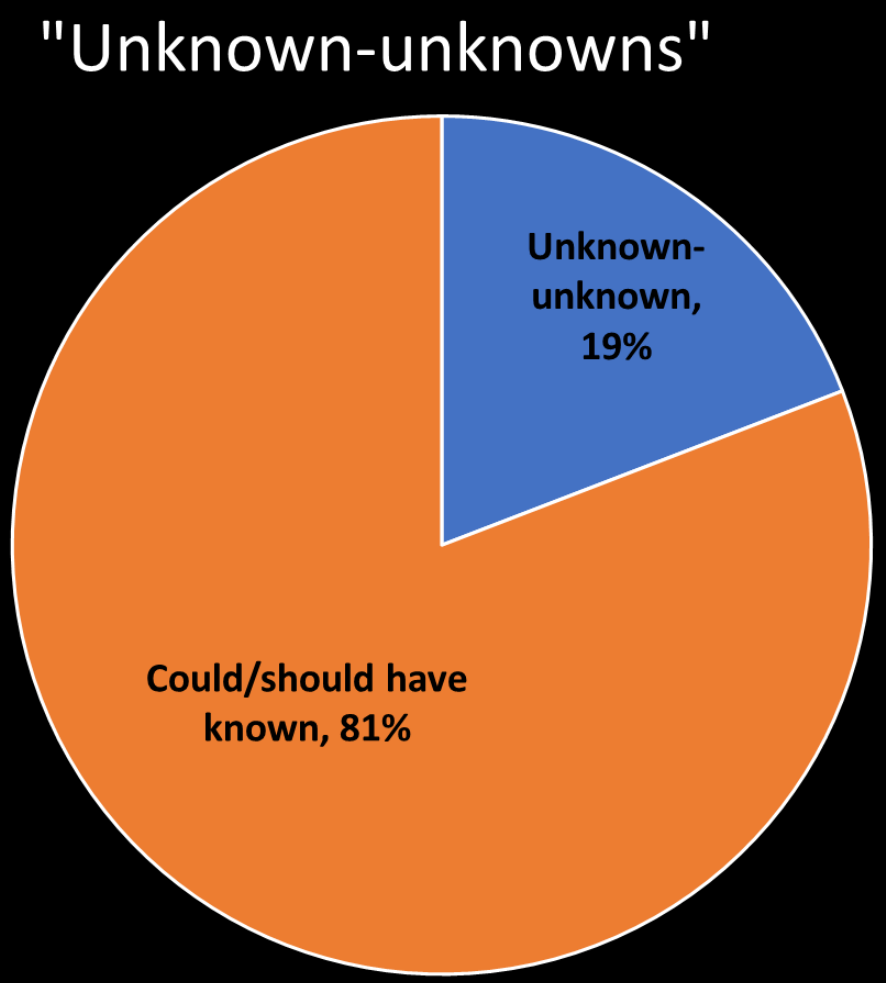
Fault-tolerant Design Tip:

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

Takeaways:

- ***Even fully tested code does not uncover errors that arise from missing code/unanticipated situations***
- ***Hard to test code that is not there(!)***

Incident Statistics – “Unknown Unknowns”



Takeaways:

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

Fault-tolerant Design Tip:

- ***Backup systems can be considered to protect for “unknown- unknowns”***
- ***Projects should actively work to balance risk between “knowing everything” and project constraints (budget/schedule)***

References

- Prokop, Lorraine, E., “Software Error Incident Categorizations in Aerospace” [Manuscript under consideration]. NASA Engineering and Safety Center. 2023.

Conclusion

- Software “common cause” or “common mode” errors occur when a single software error results in unexpected behavior, even if running on multiple strings
- Software Errors manifest in two ways: Silent or Erroneous
- Study of historical software incidents indicates the following
  - Erroneous output is much more prevalent – roughly 90% of the incidents
  - Rebooting is largely ineffective to recover from erroneous situations, and partially effective for silent software
  - Software logic errors are most common, then data config, and 23% of errors arise from input
  - Missing Code accounted for 36% of historic software errors
  - “Unknown-unknowns” account for roughly 20% of software error incidents
- Software should be architected for redundancy based on safety-criticality and time-to-effect with these statistics in mind – lessons learned:
  - Consider the Erroneous Case much more than failing silent
  - Don’t rely on reboot to recover
  - “test like you fly” – use real hardware as much as possible in real-life test cases
  - Leave time for off-nominal testing to expose unanticipated things
  - Validate commands and data prior to use
  - Consider using backup software for critical events