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 fail-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

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