HAZARDS ACROSS NASA PROGRAMS: ENHANCING SAFETY THROUGH SOFTWARE ASSURANCE AND IV&V

Robert T. Crumbley
NASA Marshall Space Flight Center,
Huntsville, AL, 35812 USA.
Email: tim.crumbley@nasa.gov

ABSTRACT

NASA missions are increasingly software-intensive and autonomous, making software behavior a major contributor to system-level safety risk. A recent cross-program review of the NASA Cross-Program Hazard Tool—supported by generative-artificial-intelligence (AI) analysis—found that critical software-related hazards recur across major programs including Orion, the Space Launch System (SLS), Ares I, and Gateway. These hazards include erroneous flight-computer outputs, corrupted Guidance, Navigation, and Control (GN&C) data, software-commanding faults, timing and processor-partition failures, configuration-load defects, and common-cause failures affecting redundant avionics. Their recurrence across diverse systems highlights the systemic nature of software vulnerabilities and the need for unified, risk-informed assurance practices across NASA.

This paper summarizes cross-program hazard trends and describes how Software Assurance (SA) and Independent Verification and Validation (IV&V), augmented by generative-AI techniques, improve hazard identification, pattern recognition, configuration verification, and detection of common-cause weaknesses. These insights support a scalable, agency-wide approach for strengthening assurance of safety-critical software throughout the mission lifecycle.

1. INTRODUCTION

NASA programs are becoming increasingly software-intensive, resulting in software behavior emerging as a dominant contributor to mission-level risk. Modern flight systems rely on complex, highly integrated software architectures that support guidance, navigation, control, autonomy, sequencing, fault management, and critical event execution. As these systems grow in scale and interdependence, even subtle defects can propagate across subsystems in ways that are difficult to detect and mitigate early in the lifecycle.

A recent cross-program assessment using the NASA Cross-Program Hazard Tool revealed that many software-related hazard types recur across multiple major missions, including Orion, the Space Launch System (SLS), Ares I, and Gateway. The repetition of these hazards—ranging from data corruption and timing anomalies to configuration-load defects and common-cause failures—indicates systemic patterns rather than isolated issues. This shared hazard landscape provides a significant opportunity to strengthen mission safety through coordinated assurance activities, improved cross-program insight, and more unified approaches to identifying, analyzing, and mitigating software-driven risks across the Agency.

2. CROSS-PROGRAM SOFTWARE HAZARD TRENDS

The assessment revealed consistent categories of software-driven hazards. These include erroneous commanding, corrupted or invalid data inputs, timing anomalies, configuration errors, and common-cause failures across redundant systems. These patterns show that similar types of software vulnerabilities reappear across different spacecraft, mission profiles, and architectures.

3. TOP SOFTWARE HAZARDS IDENTIFIED

This section summarizes the most critical software-related hazards observed across flight systems, highlighting failure modes that pose significant risks to command integrity, navigation accuracy, data processing, and overall mission reliability.

- Erroneous flight computer outputs – Occur when software or firmware produces incorrect commands, timing signals, or effector outputs. These often stem from flawed sequencing logic, arithmetic faults, race conditions, or mismatched interface assumptions.

- Invalid navigation unit outputs – Navigation systems such as Redundant Inertial Navigation Unit (RINUs) and Inertial Measurement Units (IMUs) rely on complex filtering and data fusion algorithms. Hazards emerge when faulty sensor inputs, calibration errors, or algorithmic defects lead to invalid attitude or position estimates.

- Incorrect rate-gyro data processing – Gyro signals may be misinterpreted due to timing jitter, stale samples, incorrect conversion coefficients, or misconfigured filters.

- Loss or corruption of flight-computer outputs – Timing faults, memory errors, thread-priority failures, or partition faults may suppress or distort critical command streams, impacting real-time actuation.

- Loss or corruption of GN&C data – Routing defects, filtering errors, or interface mismatches corrupt state vectors, potentially destabilizing control loops.

- Software commanding and routing errors – Commands may be misrouted due to bus-routing faults, interface configuration mistakes, or message-ID collisions.

- Requirements, design, or coding defects – Incomplete requirements or ambiguous logic lead to flawed designs and incorrect implementation of algorithms.

- Memory partition or isolation failures – Buffer overruns, incorrect memory maps, or shared-resource conflicts cause unintended cross-partition interference.

- Corrupted configuration loads – Version mismatches, incorrect units, sign errors, or parameter drift introduce unstable behavior into flight software.

- Common-cause failures across redundant avionics – Identical software or toolchains across redundant channels create systemic vulnerabilities that undermine fault tolerance.

4. WHY SOFTWARE HAZARDS MATTER

Software interacts with and coordinates nearly every safety-critical subsystem, meaning small defects can scale into mission-level hazards. Timing irregularities, incorrect data, or command routing errors may propagate through guidance, navigation, control, autonomy, and sequencing functions.

4.1 Examples of Software Hazards

This section highlights representative examples of software-related hazards that can compromise the safety, reliability, and correct operation of mission-critical systems.

- Incorrect mode-transition logic – GN&C or propulsion may enter incorrect modes due to flawed state evaluations.
- Misconfigured abort thresholds – Incorrect thresholds may trigger false aborts or prevent legitimate ones.
- Sensor data contamination – Filters may accept corrupted or stale data when validation checks are insufficient.
- Sequence timing defects – Race conditions may lead to incorrect pyrotechnic or separation timing.
- Fault-detection false positives – Incorrect Fault Detection, Isolation, and Recovery (FDIR) rules may isolate healthy subsystems.
- Partition-boundary violations – Memory corruption across partitions disrupts safety-critical processes.
- Invalid control-law coefficients – Incorrect I-loads destabilize guidance or attitude control.
- Firmware version mismatch – Redundant processors behave inconsistently when firmware versions differ.

5. SOFTWARE ASSURANCE AND IV&V OPPORTUNITIES

There are a number of key opportunities for strengthening NASA's software assurance and IV&V capabilities across critical technical and programmatic domains.

- Mitigating common-cause failures – Promote diversity in architectures and toolchains.
- Hazard-focused software assurance and IV&V – Increase targeted testing for high-risk sequences.
- Data integrity and timing assurance – Improve timestamp checks and timing diagnostics.
- Configuration and load assurance – Strengthen validation of configuration files and parameters.
- Algorithm and model validation – Apply formal verification and simulation credibility checks.
- Oversight of Commercial Off-The-Shelf (COTS)/ Real-Time Operating System (RTOS)/ Field-Programmable Gate Array (FPGA) components – Increase vendor process audits.

- Cross-program hazard traceability – Maintain centralized hazard taxonomies.
- Increased independent testing
- Use of AI and static analysis tools in software assurance and IV&V

6. FUTURE TRENDS IN SOFTWARE SAFETY

This section outlines emerging trends poised to shape the future of software safety, reflecting increasing autonomy, advanced analytical methods, and evolving assurance practices across modern mission systems.

- Increased autonomy – Greater reliance on onboard decision-making.
- Digital-twin integration – More real-time synchronization between flight and ground simulations.
- Growth in formal methods – Expanded use of theorem-checking tools.
- AI-assisted hazard detection – Automated reasoning to identify defect patterns.
- Zero-trust software architecture – Design approaches ensuring rigorous verification of all data flows.
- Dynamic configuration safety – Verification approaches for safe in-flight updates.
- Expanded supply-chain assurance – More robust scrutiny of external libraries and firmware.

7. ABOUT THE CROSS-PROGRAM HAZARD (CP HAZARD) SYSTEM

The Cross Program Hazard (CP Hazard) System is NASA's centralized, agency-wide environment for creating, reviewing, approving, and managing hazard reports. Supporting more than 30 program components and maintaining over 6,000 individual hazards, the system ensures consistency, traceability, and high-integrity hazard documentation across missions and directorates.

7.1 System Purpose

The CP Hazard System provides a common platform that improves the quality and uniformity of hazard reporting. It enables programs to share data, maintain consistent formats, and manage complex approval workflows while preserving the integrity of safety information.

7.2 Key System Goals

- Provide a centralized hazard repository with uniform reporting and analysis standards.
- Maintain a common, structured workflow, including electronic signatures, notifications, and approvals.
- Support interoperability with related systems such as Failure Modes and Effects Analysis (FMEA)/ Critical Items List (CIL), Problem Reporting and Corrective Action (PRACA), and Government Mandatory Inspection Points (GMIP) to enable integrated risk management.
- Ensure data integrity through validation against trusted internal and external sources.
- Preserve and provide access to archived Constellation Program hazard data for historical comparison and insight.

7.3 Core Capabilities

- Administrative control over interface fields, configuration, and data entry.
- Robust advanced search tools, including user-defined, savable, and shareable queries.
- Comprehensive data linking, supporting:
    - Relationships within a record (e.g., linking causes to controls).
    - Transfers and cross-references between records.
- Connections to external systems such as FMEA/CIL.
- Tools to edit multiple Hazard Reports simultaneously, improving efficiency for large programs.
- Electronic signature capability for formal approvals.
- Automated email notifications for assignments, updates, and approval actions.

7.4 CP Hazard System Architecture

The CP Hazard System is built on a customized implementation of the Bugzilla open-source defect-tracking platform.

Its architecture includes:

- Backend: Practical Extraction and Reporting Language (PERL) and MySQL
- Front-end: HyperText Markup Language (HTML), Cascading Style Sheets (CSS), and JavaScript for a responsive web-based interface

7.5 Open-Source Libraries

The CP Hazard System and related Mission Assurance Systems (MAS) rely on a variety of open-source

libraries to support user interface, workflow, and data management functionality. These include: @babel, CKEditor, Emotion, Material UI (MUI), PDFTron, lodash, moment, jQuery, react, redux, xlsx, and numerous other frameworks that enhance validation, dynamic content handling, and overall performance.

8. HOW CP-HAZARD DATA WAS EXTRACTED AND USED IN THE ANALYSIS

The cross-program software hazard analysis relied on automated extraction techniques applied to the CP-Hazard System's centralized dataset. Generative-AI-based parsing tools were used to:

- Identify all Hazard Reports containing software-related causes, controls, or failure modes.
- Normalize terminology across programs by mapping hazard attributes to a unified taxonomy.
- Compare hazard categories across Orion, SLS, Ares I, Gateway, and other systems to identify shared patterns.
- Quantify the frequency of specific hazard types and examine clustering around critical subsystems such as GN&C, avionics, mode-transition logic, sequencing, and FDIR.
- Detect systemic vulnerabilities where multiple programs shared identical failure modes, indicating opportunities for agency-wide mitigation.

The CP-Hazard database's linking functionality was used to trace relationships between causes, controls, and transferred hazards across programs. Cross-system links (e.g., to FMEA/CIL) helped validate root-cause consistency and ensured that hazards were reviewed within the broader risk-management context. Archived Constellation Program data provided historical parallels that strengthened confidence in observed trends.

9. CONCLUSION

Cross-program analysis continues to reveal that NASA's software hazards follow repeatable patterns across missions, architectures, and development environments—clear evidence that these issues are systemic rather than isolated. This consistency underscores the importance of the enterprise-level Software Assurance and IV&V opportunities identified earlier: diversity in architectures, hazard-focused testing, deeper configuration and timing assurance, stronger model validation, and unified oversight of COTS/RTOS/FPGA components.

As the Agency transitions toward increasingly autonomous, software-intensive, and AI-enabled systems, the scale and complexity of these hazards will only grow. Modern AI techniques now give NASA the ability to mine and correlate thousands of hazard records, requirements, defect logs, and test artifacts across more than 30 program components and over 6,000 hazards in the CP Hazard System—identifying trends, gaps, and emerging risk signatures that human teams could not detect at mission cadence.

By combining these AI-driven insights with rigorous Software Assurance and IV&V practices, NASA can move from reactive hazard discovery to proactive hazard prediction, ensuring hazards are fully developed, validated, and mitigated in human-rated programs. The path forward is clear: an agency-wide, data-driven assurance ecosystem that leverages advanced analytics, shared evidence, and coordinated SA+IV&V processes will be essential to safeguarding crew, protecting mission integrity, and sustaining reliability in the next era of exploration.

10. REFERENCES

[1] NASA (n.d.). *Cross-Program Hazard Database*. NASA. Retrieved July 17, 2026, from https://cxhazard.nasa.gov/home