

VENDOR'S GUIDE TO DATA REQUIREMENTS FOR WEAPONIZED UAS SOFTWARE SYSTEM SAFETY PROGRAM

PREPARED BY: U.S. Army Aviation and Missile Command, Safety Office

PREPARED FOR: Drone Dominance Program

Version 1.0

Distribution Statement A (26-108): Approved for public release; distribution unlimited. July 2026.

Vendor's Guide to Data Requirements for Weaponized UAS Software System Safety Program
Version 1.0

FOREWORD

This guide has been developed by the U.S. Army Aviation and Missile Command Safety Office. It complements the Vendor's Guide to Data Requirements for Weaponized UAS Safety Concurrence developed by the Naval Surface Warfare Center Indian Head Division's Research & Engineering Department's System Safety Division. This guide refines the minimum software safety data and documentation requirements for industry partners to support the U.S. Government's review of the software safety program for weaponized Unmanned Aircraft Systems (UAS). Adherence to this guide is critical for ensuring the best evaluation of the software system safety program and determination of courses of actions for reassessing the software contribution to system risks.

This guide will be updated with lessons learned and evaluation of software applications using Artificial Intelligence (AI) technology.

1. Introduction

The information identified in this guide will be the primary source of information for the Functional Hazard Analysis. From the system level analysis, allocation to software for implementation (in part or whole) will determine the expected rigor in the software development lifecycle. The maturity of the software system safety program and associated activities will be based on the ability to articulate the expected software lifecycle rigor, its traceability to system safety and obtaining credit for safety verification activities.

The overarching requirements to assess the risk for software are found in MIL-STD-882E Change 1, section 4.4.

2. Foundational Data

2.1. Evaluation Criteria

Software will be classified in accordance with the criteria of MIL-STD-882E Change 1 Tables I, IV, V and VI.

- Table I, severity categories. This table is used for the system level Hazard Analysis (HA).

SEVERITY CATEGORIES		
Description	Severity Category	Mishap Result Criteria
Catastrophic	1	Could result in one or more of the following: death, permanent total disability, irreversible significant environmental impact, or monetary loss equal to or exceeding \$10M.
Critical	2	Could result in one or more of the following: permanent partial disability, injuries or occupational illness that may result in hospitalization of at least three personnel, reversible significant environmental impact, or monetary loss equal to or exceeding \$1M but less than \$10M.
Marginal	3	Could result in one or more of the following: injury or occupational illness resulting in one or more lost work day(s), reversible moderate environmental impact, or monetary loss equal to or exceeding \$100K but less than \$1M.
Negligible	4	Could result in one or more of the following: injury or occupational illness not resulting in a lost work day, minimal environmental impact, or monetary loss less than \$100K.

- Table IV, software control categories (SCC). This table is unique to software. For each software Configuration Item (CI), determine the best suited SCC based on the functionality provided by that software CI.

TIP: Table IV identifies functional areas – software that controls and software that carries information. Do not discard SCC because your software CI only transmits information.

TIP: Redundant Fault Tolerant (RFT) SCC requires the identification of interlocks and/or redundant pathways. Use the architecture of the system to substantiate RFT; it is the most challenging SCC to justify.

SOFTWARE CONTROL CATEGORIES		
Level	Name	Description
1	Autonomous (AT)	Software functionality that exercises autonomous control authority over potentially safety-significant hardware systems, subsystems, or components without the possibility of predetermined safe detection and intervention by a control entity to preclude the occurrence of a mishap or hazard. <i>(This definition includes complex system/software functionality with multiple subsystems, interacting parallel processors, multiple interfaces, and safety-critical functions that are time critical.)</i>
2	Semi-Autonomous (SAT)	- Software functionality that exercises control authority over potentially safety-significant hardware systems, subsystems, or components, allowing time for predetermined safe detection and intervention by independent safety mechanisms to mitigate or control the mishap or hazard. <i>(This definition includes the control of moderately complex system/software functionality, no parallel processing, or few interfaces, but other safety systems/mechanisms can partially mitigate. System and software fault detection and annunciation notifies the control entity of the need for required safety actions.)</i> - Software item that displays safety-significant information requiring immediate operator entity to execute a predetermined action for mitigation or control over a mishap or hazard. Software exception, failure, fault, or delay will allow, or fail to prevent, mishap occurrence. <i>(This definition assumes that the safety-critical display information may be time-critical, but the time available does not exceed the time required for adequate control entity response and hazard control.)</i>
3	Redundant Fault Tolerant (RFT)	- Software functionality that issues commands over safety-significant hardware systems, subsystems, or components requiring a control entity to complete the command function. The system detection and functional reaction includes redundant, independent fault tolerant mechanisms for each defined hazardous condition. <i>(This definition assumes that there is adequate fault detection, annunciation, tolerance, and system recovery to prevent the hazard occurrence if software fails, malfunctions, or degrades. There are redundant sources of safety-significant information, and mitigating functionality can respond within any time-critical period.)</i> - Software that generates information of a safety-critical nature used to make critical decisions. The system includes several redundant, independent fault tolerant mechanisms for each hazardous condition, detection and display.
4	Influential	Software generates information of a safety-related nature used to make decisions by the operator, but does not require operator action to avoid a mishap.
5	No Safety Impact (NSI)	Software functionality that does not possess command or control authority over safety-significant hardware systems, subsystems, or components and does not provide safety-significant information. Software does not provide safety-significant or time sensitive data or information that requires control entity interaction. Software does not transport or resolve communication of safety-significant or time sensitive data.

- Table V, software safety criticality matrix. For each function allocated to a software CI and for the SCC previously determined, identify the Software Criticality Index (SwCI).

SOFTWARE SAFETY CRITICALITY MATRIX				
	SEVERITY CATEGORY			
SOFTWARE CONTROL CATEGORY	Catastrophic (1)	Critical (2)	Marginal (3)	Negligible (4)
1	SwCI 1	SwCI 1	SwCI 3	SwCI 4
2	SwCI 1	SwCI 2	SwCI 3	SwCI 4
3	SwCI 2	SwCI 3	SwCI 4	SwCI 4
4	SwCI 3	SwCI 4	SwCI 4	SwCI 4
5	SwCI 5	SwCI 5	SwCI 5	SwCI 5

- Table V, Level of Rigor (LOR) tasks. The high level LOR tasks at the bottom of Table V provide the minimum expectations for data or documentation of the software development lifecycle.

SwCI	Level of Rigor Tasks
SwCI 1	Program shall perform analysis of requirements, architecture, design, and code; and conduct in-depth safety-specific testing.
SwCI 2	Program shall perform analysis of requirements, architecture, and design; and conduct in-depth safety-specific testing.
SwCI 3	Program shall perform analysis of requirements and architecture; and conduct in-depth safety-specific testing.
SwCI 4	Program shall conduct safety-specific testing.
SwCI 5	Once assessed by safety engineering as Not Safety, then no safety specific analysis or verification is required.

- Table VI, relationship between SwCI, risk level and LOR tasks. If the data or documentation provided is insufficient to meet the intent of the LOR tasks in Table V, this table is used to assign the software contribution to system-level risk.

RELATIONSHIP BETWEEN SwCI, RISK LEVEL, LOR Tasks, AND RISK		
Software Criticality Index (SwCI)	Risk Level	Software LOR Tasks and Risk Assessment/Acceptance
SwCI 1	High	If SwCI 1 LOR tasks are unspecified or incomplete, the contributions to system risk will be documented as HIGH and provided to the PM for decision. The PM shall document the decision of whether to expend the resources required to implement SwCI 1 LOR tasks or prepare a formal risk assessment for acceptance of a HIGH risk.
SwCI 2	Serious	If SwCI 2 LOR tasks are unspecified or incomplete, the contributions to system risk will be documented as SERIOUS and provided to the PM for decision. The PM shall document the decision of whether to expend the resources required to implement SwCI 2 LOR tasks or prepare a formal risk assessment for acceptance of a SERIOUS risk.
SwCI 3	Medium	If SwCI 3 LOR tasks are unspecified or incomplete, the contributions to system risk will be documented as MEDIUM and provided to the PM for decision. The PM shall document the decision of whether to expend the resources required to implement SwCI 3 LOR tasks or prepare a formal risk assessment for acceptance of a MEDIUM risk.
SwCI 4	Low	If SwCI 4 LOR tasks are unspecified or incomplete, the contributions to system risk will be documented as LOW and provided to the PM for decision. The PM shall document the decision of whether to expend the resources required to implement SwCI 4 LOR tasks or prepare a formal risk assessment for acceptance of a LOW risk.
SwCI 5	Not Safety	No safety-specific analyses or testing is required.

2.2. Documentation to Support Software Safety Program

To demonstrate rigor in a safety program, you must document your process: what will be done, who will do it, when it will be done and what the output will be. Having a plan for the execution of the LOR tasks is strongly recommended.

a. Foundation.

For software safety, the processes and techniques to execute the software system safety program are usually documented in:

- A software safety section in the system safety plan (WHISPR); and/or
- A Software Development Plan (SDP); and/or
- A Software Test Plan (STD); and/or
- A Software Configuration Management Plan (SCMP); and/or
- WIKI page(s); and/or

Commented [LM1]: Add AI/ML discussion (spiral) - disclosure including whether it's locked down.

- Standard Operating Procedures, Instructions, or similar.

b. Additional Considerations for Artificial Intelligence Technologies.

For the initial phase (triage), the vendor needs to disclose information on the presence and approach to learning the AI technology present in the system.

- Is AI used in the development of software applications?
- Is AI used in the operation of software applications?
- Is AI used in a safety-critical function identified in the Functional Hazard Analysis?
- Is AI learning locked-out during the operation of the system (i.e., the model is unable to update its weights or to absorb collected information; the model cannot accept new inputs such as prompts)

For the subsequent phases (spirals), the capability provided via AI during development and/or operation will be added to the Functional Hazard Analysis. Be prepared to present information on AI system lifecycle, including the datasets used for development, training, pruning and validation of the model.

3. Evidence for Level of Rigor Tasks

3.1. High-level Required Tasks

As stated in section 2, the high-level required tasks are defined in MIL-STD-882E Change 1, Table V and recalled below.

SwCI	Level of Rigor Tasks
1	Program shall perform analysis of requirements, architecture, design, and code; and conduct in-depth safety-specific testing.
2	Program shall perform analysis of requirements, architecture, and design; and conduct in-depth safety-specific testing.
3	Program shall perform analysis of requirements and architecture; and conduct in-depth safety-specific testing.
4	Program shall conduct safety-specific testing.
5	Once assessed by safety engineering as Not Safety, then no safety-specific analysis or verification is required.

3.2. Assessment of LOR Tasks

a. Compliance with Analysis of Requirements.

Because most programs implement requirements-based testing, it is key that requirements cover the safety design and the desired behaviors for abnormal and emergency/contingency conditions. Fast-paced programs understandably focus on function and performance. For readiness, effectiveness and personnel safety, it is as important to include safety-significant requirements.

Safety-significant requirements need to be identified ("tagged"). For example, look for requirements that:

- Require the introduction of software-enabled capability organically associated with one or more hazards (e.g., laser, emitter)
- Provide hazard control implemented in software (e.g., monitor, gating)
- Provide fault management (detection, isolation, identification, annunciation, recovery)
- Provide personnel alerting of abnormal, emergency, contingency conditions
- Dictates redundancy, independence, dissimilarity, partitioning
- Provide status or health for safety significant command/control functions

Documents supporting the analysis of requirements include but are not limited to:

- Software Requirements Specification (SRS)
- Software Interface Requirements (SIR)

If the program does not use requirements (e.g., Agile methodology), representation of behaviors can be substituted as long as they cover abnormal, emergency and contingency conditions:

- Activity Diagrams (MBSE)
- User Stories (Agile)
- WIKIs

b. Compliance with Analysis of Architecture and Design.

The objective is to identify the architectural features that support hazard control and software control category determination for RFT. Identify the following features in the design (not an exhaustive list):

- Partitioning between safety-significant and non-safety significant implementations; or between safety-significant implementations of different

LOR (e.g., source code partitioning, different processors, different processor cores, different cards, or different programmable logic devices)

- Paths (logical or physical) that support fault management

Documents supporting the analysis of architecture and design include but are not limited to:

- Software Architecture Document (SAD)
- Software Design Document (SDD)
- Human Factors Engineering (HFE) report
- Engineering drawings of software hosting electronic components
- Fault Tree(s)
- State Machine Diagram(s), States and Modes Diagram
- Sequence Diagram(s)

c. Compliance with Code Analysis.

The objective is to demonstrate that all behaviors that are (in part or whole) implemented in software are understood. It is also to ensure “bugs” are caught as early as possible (i.e., least number of bugs is discovered during developmental or operational testing). Finally, the analysis needs to address any additional functionality that is not necessary to meet the requirements or required behaviors (e.g. extra code added by the compiler, dead or unused code). The following are the most often used code analyses:

- Static analysis examines the code without executing it (e.g., compliance with coding standard)
- Dynamic analysis develops specific code coverage metrics relevant to code execution
- Worst-case Execution Time (WCET) analysis

If the code base is simple enough, static analysis as a code walkthrough can be done, but most of the time, the code is too complex, and static analysis will likely rely on tools.

Code coverage or structural coverage analysis is a key measure to show that sufficient testing has been performed and it can correlate to an increased likelihood that problems have been uncovered. The following structural (code) coverage metrics are identified as LOR tasks:

- Statement coverage: number of statements exercised during the execution of a program as a percentage of number of statements in the program (objective: 100%)
- Branch (decision) coverage: number of branches exercised during the execution of a program as a percentage of number of statements in the program. Ensures both outcomes of any decision branching are exercised (e.g., go/no-go paths, non-happy paths, error handler cases)
- Modified condition/decision coverage: testing criteria that ensures each condition within a decision statement independently affects the decision's outcome.

WCET analysis should be considered when execution time of the executable code on the target hardware is a safety characteristic (e.g., time critical application, low robustness to latency).

Documents supporting code analysis typically include:

- Code walkthrough/review checklists
- Report from code coverage analysis tools (with the explanation of any deviation from full compliant result)
- WCET analysis report

d. Considerations for Safety-specific Testing.

Safety-specific testing will cover all safety-significant requirements (per the coverage analysis) and regression testing. Tests will be conducted at the appropriate level to verify the desired behavior. Typical types of tests include, but are not limited to:

- Unit level testing
- Regression testing
- Functional thread testing
- Failure modes and effects testing, fault insertion
- Robustness testing
- Smoke testing (or go/no-go path testing)
- Stress testing
- Interface testing (including user interface verification)
- Input/output testing (typically for off the shelf items)

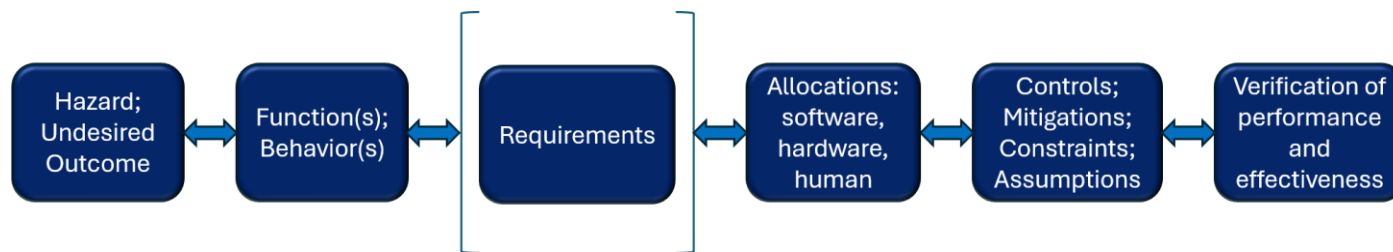
For systems where safety is interlaced with security, consider penetration testing.

Documents supporting safety-specific testing typically include:

- Software Test Plan
- Software Test Cases and Procedures
- Quick-look reports
- Software Test Reports (STRs)
- User evaluation test reports
- Warnings, Cautions and Notes (to identify deficiencies in safety-significant software implementation)
- Human-System Interface (HSI) report

3.3. Demonstrating End-to-end Traceability

Using the Hazard Analysis or the Functional Hazard Analysis as a canvas, the objective is to demonstrate bidirectional end-to-end traceability.



4. Glossary

Unless otherwise specified, the definitions are from MIL-STD-882E.

Firmware	The combination of a hardware device and computer instructions or computer data that reside as read-only software on the hardware device. The software cannot be readily modified under program control.
Go/No-go path testing	Also known as smoke testing. A binary (pass/fail) gate checks that determine if a software build is stable enough ("go") for the next phase or should be blocked ("no-go"). This testing is typically carried out via automated tests. [Wikipedia]
Happy path testing	A subset of positive testing that focuses on the most common and expected scenarios. A Happy path test is a well-defined test case using known input, which executes without exception and produces an expected output. Happy path testing can show that a system meets its functional requirements, but it does not guarantee a graceful handling of error conditions or aid in finding hidden bugs. [Wikipedia] Note: there is defined term for the opposite of happy paths, since lots of things can go wrong.
Hazard	A real or potential condition that could lead to an unplanned event or series of events (i.e. mishap) resulting in death, injury, occupational illness, damage to or loss of equipment or property, or damage to the environment.
Level of Rigor	A specification of the depth and breadth of software analysis and verification activities necessary to provide a sufficient level of confidence that a safety-critical or safety-related software function will perform as required.
Mode	A designated system condition or status (e.g., maintenance, test, operation, storage, transport, and demilitarization).
Safety-critical	A term applied to a condition, event, operation, process, or item whose mishap severity consequence is either Catastrophic or Critical (e.g., safety-critical function, safety-critical path, and safety-critical component).
Safety-critical Function (SCF)	A function whose failure to operate or incorrect operation will directly result in a mishap of either Catastrophic or Critical severity.
Safety-critical Item (SCI)	A hardware or software item that has been determined through analysis to potentially contribute to a hazard with Catastrophic or

Vendor's Guide to Data Requirements for Weaponized UAS Software System Safety Program
Version 1.0

	Critical mishap potential, or that may be implemented to mitigate a hazard with Catastrophic or Critical mishap potential.
Safety-related	A term applied to a condition, event, operation, process, or item whose mishap severity consequence is either Marginal or Negligible.
Safety-significant	A term applied to a condition, event, operation, process, or item that is identified as either safety-critical or safety-related.
Software	A combination of associated computer instructions and computer data that enable a computer to perform computational or control functions. Software includes computer programs, procedures, rules, and any associated documentation pertaining to the operation of a computer system. Software includes new development, complex programmable logic devices (firmware), NDI, COTS, GOTS, re-used, GFE, and Government-developed software used in the system.

5. Resources

Available from Assist (<https://quicksearch.dla.mil/qsSearch.aspx>)

- MIL-STD-882E with Change 1, Standard Practice System Safety
- AOP-52, Guidance on Software Safety Design and Assessment of Munition Related Computing Systems

Available upon request

- JS-SSA, Software System Safety Implementation Process and Tasks Supporting MIL-STD-882E (revision B, 14 March 2018)
- U.S. Army Aviation and Missile Command Regulation 385-17, Software System Safety Policy, 11 July 2020
- [Distro B, Direct request to NOSSA] Guidance to Perform Functional Hazard Analysis for Weapon Systems with Artificial Intelligence Capabilities, 19 May 2022