International System Safety Training Symposium

Software Safety Tutorial
NSWCDD-PN-14-00295

5 August 2014

Ms. Peggy Rogers
NOSSA, 301-744-6002
peggy.rogers@navy.mil

Mr. Stuart Whitford
NOSSA, 301-744-6058
stuart.whitford@navy.mil

Mr. Mike Zemore
NSWCDD, 540-653-7881
michael.zemore@navy.mil

Ms. Carolyn Tilghman
NSWCDD, 540-653-6764
carolyn.tilghman@navy.mil

Ms. Rebecca Funkhouser
NSWCDD, 540-653-9224
rebecca.funkhouser@navy.mil
Outline

1. **Course Overview**

2. Mock System Overview - Very Important Person Protection System (VIPPS)

3. Software Safety Process
   1. Perform Software Criticality Index and Level of Rigor Assessment
   2. Perform Architecture and Design Analysis
   3. Perform Code Analysis (No coding experience necessary)

4. Assess Technology Insertion Proposal

5. Software (SW) Safety Analysis Summary

6. Conclusion
Learning Objectives

- Recognize the importance of SW Safety and how it fits into the Integrated Defense Acquisition System
- Assess SW contributions to the overall System Risk
- Understand how to identify software functions that implement Safety-Significant Functionality
- Determine Software Criticality Index (SwCI) based on the control category of the software and level of severity of potential mishaps with software contributors
- Become familiar with specific analyses, tests, and verifications that are recommended based on the criticality index
- Recognize software design and failure modes that could result in hazards and causal factors
- Understand the actions and mitigations required to eliminate hazards and causal factors, or control their associated risk
SW Safety Analysis

What is SW Safety Analysis and why is it important?

- SW Safety Analysis is typically conducted during the Subsystem Hazard Analysis (SSHA) efforts.
- SW can cause, influence, contribute to, or mitigate hazards.
- SW Analysis results in improved designs and mitigations, and reduces the likelihood that SW will initiate a hazardous condition or mishap [Military Standard (MIL-STD)-882E B.2.2]
  - SW System Safety Analysis (the focus of this tutorial) increases the confidence that the software will perform as specified to software system safety and performance requirements while reducing the number of contributors to hazards that may exist in the system.
- The SW Safety Analysis process is not defined directly in MIL-STD-882E, but the Level of Rigor (LOR) tasks include:
  - SW Requirements Hazard Analysis
  - SW Architectural Hazard Analysis
  - SW Design Hazard Analysis
  - Code Level Hazard Analysis
  - SW Safety Testing and Verification
SSHA in the Integrated Defense Acquisition System

- SW Safety Analysis is primarily conducted in the Engineering and Manufacturing Development Phase, but may begin in the Technology Development Phase.

ACRONYMS

<table>
<thead>
<tr>
<th>ACRONYMS</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>CDR</td>
<td>Critical Design Review</td>
</tr>
<tr>
<td>FOC</td>
<td>Full Operational Capability</td>
</tr>
<tr>
<td>FRP</td>
<td>Full Rate Production</td>
</tr>
<tr>
<td>HHA</td>
<td>Health Hazard Assessment</td>
</tr>
<tr>
<td>IOC</td>
<td>Initial Operational Capability</td>
</tr>
<tr>
<td>O&amp;SHA</td>
<td>Operating and Support Capability</td>
</tr>
<tr>
<td>PDR</td>
<td>Preliminary Design Review</td>
</tr>
<tr>
<td>PHA</td>
<td>Preliminary Hazard Analysis</td>
</tr>
<tr>
<td>PHL</td>
<td>Preliminary Hazard List</td>
</tr>
<tr>
<td>SRHA</td>
<td>System Requirements Hazard Analysis</td>
</tr>
<tr>
<td>SRVM</td>
<td>System Requirement Verification Matrix</td>
</tr>
<tr>
<td>SHA</td>
<td>System Hazard Analysis</td>
</tr>
<tr>
<td>SSHA</td>
<td>Subsystem Hazard Analysis</td>
</tr>
<tr>
<td>WSESRRB</td>
<td>Weapon System Explosives Safety Review Board</td>
</tr>
</tbody>
</table>

SYSTEM SAFETY MANAGEMENT PLAN

- Guidelines for coordination/conduct of large system safety programs

SYSTEM SAFETY PROGRAM PLAN

- Specifics detailing system safety methods, efforts, tasks and milestones

SAFETY TESTING

- PHIL
- FHA
- PHA
- PESHE
- SRHA
- SRVM
- SSHA
- SHA
- O&SHA
- HHA

CHANGE REVIEW & ANALYSIS

- Hazard Tracking Database

Safety Assessment Reports and Formal WSESRRB Reviews

- Introductory
- Pre-CDR
- Pre-Low Rate Initial Production (LRIP)
- Pre-Full Rate Production (FRP)
- Pre-Deployment
- As-occurring, prior to at-sea testing

NSWCDD-PN-14-00295 is approved for Distribution Statement A: Approved for Public Release; distribution is unlimited.
Outline

1. Course Overview

2. Mock System Overview - Very Important Person Protection System (VIPPS)

3. Software Safety Process
   1. Perform Software Criticality Index and Level of Rigor Assessment
   2. Perform Architecture and Design Analysis
   3. Perform Code Analysis (No coding experience necessary)

4. Assess Technology Insertion Proposal

5. Software (SW) Safety Analysis Summary

6. Conclusion
System Overview and Documentation

VIPPS Concept of Operations (CONOPS)

VIPPS System / Subsystem Specification (SSS)

VIPPS Subsystem Design Document (SSDD)

Gun Critical Item Development Specification (CIDS)

Gun Subsystem Design Document (SSDD)

VIPPS Gun Software Description Document

VIPPS is a Mock System used for training purposes only.
The Very Important Person Protection System (VIPPS) should detect and track threats approaching the VIP’s house. It should characterize the movements of those threats and aid the operator in recognizing hostile actions and intent. The intrusion would cause an alert at the operators console and provide the operator an approach bearing of the threat. The VIPPS will track the threat via video and allow Identification (ID) assignments. Based on hostile intent and distance from the home, the operator will engage within the warning zone with an Unambiguous Warning Device (UWD), which is a loudspeaker and/or high intensity spotlight. If the intruder continues toward the home with determined hostile intent, the operator will employ lethal effectors to neutralize the intruder before they enter the house.
Assessment Zone
Warning Zone
Engagement Zone

VIPPS is a Mock System used for training purposes only.
Camera (Sensor) and Illuminator Locations

VIPPS is a Mock System used for training purposes only.
VIPPS is a Mock System used for training purposes only.
VIPPS Block Diagram

Battery Backup

Gun Subsystem
- AMMO Supply
- Fire Control Interface
- Optics Suite

High Intensity Spotlight

Hailer

Data Recording Device

Command and Control Computer (C2)

Operator Console

Power

Data

VIPPS is a Mock System used for training purposes only

Assumptions
- Interior Equipment
- Exterior Equipment

Battery Backup

VIPPS Block Diagram

NSWCDD-PN-14-00295 is approved for Distribution Statement A:
Approved for Public Release; distribution is unlimited.
VIPPS is a Mock System used for training purposes only.
Gun Subsystem Block Diagram

VIPPS is a Mock System used for training purposes only.
### VIPPS Safety Schedule

<table>
<thead>
<tr>
<th>Tasks</th>
<th>FY 0001</th>
<th>FY 0002</th>
<th>FY 0003</th>
<th>FY 0004</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Program Milestones</strong></td>
<td>M/S B</td>
<td>M/S C</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ASR</td>
<td>Q1</td>
<td>Q1</td>
<td>Q1</td>
<td>Q1</td>
</tr>
<tr>
<td>SRR</td>
<td>Q2</td>
<td>Q2</td>
<td>Q2</td>
<td>Q2</td>
</tr>
<tr>
<td>PDR</td>
<td>Q3</td>
<td>Q3</td>
<td>Q3</td>
<td>Q3</td>
</tr>
<tr>
<td>CDR/HW</td>
<td></td>
<td>CDR/SW</td>
<td>CDR/SYS</td>
<td></td>
</tr>
<tr>
<td>CDR/SW</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CDR/SYS</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IPR</td>
<td></td>
<td></td>
<td>IP/PR</td>
<td></td>
</tr>
<tr>
<td>PRR</td>
<td></td>
<td></td>
<td></td>
<td>PRR</td>
</tr>
<tr>
<td>Cert</td>
<td></td>
<td></td>
<td></td>
<td>Cert</td>
</tr>
<tr>
<td><strong>Test Events</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TEMP</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Vib</td>
<td></td>
<td></td>
<td>LBIT</td>
<td></td>
</tr>
<tr>
<td>Toxicity</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Noise</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DT B1B</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DT B2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DT B3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DT B4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>WSESRB</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pre-PDR/MSB</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SSSTRP</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pre-CDR</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SSSTRP/WSESRB Live Fire DT</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LF</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LF</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SSSTRP/WSESRB FRP</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>SAR/TDP</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PHA w/FHA</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SSHAB</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(Include FMECA, SW Safety, Audit)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HHA</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SRHA (SSS, SRS)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SHA (including FTA)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>O&amp;SHA</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ESOH</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Haz Track</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SSWG</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

NSWCDD-PN-14-00295 is approved for Distribution Statement A: Approved for Public Release; distribution is unlimited.
Outline

1. Course Overview

2. Mock System Overview - Very Important Person Protection System (VIPPS)

3. **Software Safety Process**
   1. Perform Software Criticality Index and Level of Rigor Assessment
   2. Perform Architecture and Design Analysis
   3. Perform Code Analysis (No coding experience necessary)

4. Assess Technology Insertion Proposal

5. Software (SW) Safety Analysis Summary

6. Conclusion
Key Definitions

- Safety Significant
  - A term applied to a condition, event, operation, process, or item that is identified as either safety critical or safety related

- 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 Related
  - A term applied to a condition, event, operation, process, or item whose mishap severity consequence is either Marginal or Negligible
Software Safety Analysis and Verification Process

Top-Level Process

Start → System Definition and Software Safety Planning → Determine Software Criticality Index (SwCI) → Software Requirements Hazard Analysis (SwCI 1-3) → Software Architectural Hazard Analysis (SwCI 1-3) → Software Design Hazard Analysis (SwCI 1-2) → Code Level Hazard Analysis (SwCI 1) → Operator Documentation Safety Review → Software Testing and Verification (SwCI 1-4) → Formal Review → Fleet Release

Sub-Process

Defect Resolution → Regression Testing → Fleet Anomaly Reporting

Software Criticality Matrix
• System Safety Management Plan (SSMP) to include software consideration
• Develop the software safety plan to define the scope of the software safety program and add or include within the overall System Safety Program Plan (SSPP)
  ➢ Add a Software Safety Program Plan (SwSPP) as an appendix, or include the software safety plan within the context of the SSPP without separating as an appendix
• SSMP and SSPP are provided to the Safety Review Boards during an Introductory Review
Software Safety Analysis and Verification Process

Top-Level Process

Start → System Definition and Software Safety Planning

Software Design Hazard Analysis (SwCl 1-2) → Code Level Hazard Analysis (SwCl 1) → Operator Documentation Safety Review → Software Testing and Verification (SwCl 1-4) → Formal Review → Fleet Release

Determine Software Criticality Index (SwCl)

Software Requirements Hazard Analysis (SwCl 1-3)

Software Architectural Hazard Analysis (SwCl 1-3)

In-depth Safety - Specific Testing (SwCl 1-3)

Sub-Process

Defect Resolution → Regression Testing → Fleet Anomaly Reporting

NSWCDD-PN-14-00295 is approved for Distribution Statement A: Approved for Public Release; distribution is unlimited.
Outline

1. Course Overview

2. Mock System Overview - Very Important Person Protection System (VIPPS)

3. **Software Safety Process**
   1. Perform Software Criticality Index and Level of Rigor Assessment
   2. Perform Architecture and Design Analysis
   3. Perform Code Analysis (No coding experience necessary)

4. Assess Technology Insertion Proposal

5. Software (SW) Safety Analysis Summary

6. Conclusion
Determine SW Criticality Index

- SW Safety-Significant Functions (SSFs) are identified by flowing the system level SSFs to the implementing SW function
  - Utilizing PHA and Functional Hazard Analysis (FHA)
- The potential mishap severity is defined by the hazard or mishap associated to the software function
- The degree of software control is assessed and defined for each SW SSF
  - Using Software Control Categories (SCCs) in MIL-STD-882E
- The MIL-STD-882E Software Safety Criticality Matrix (SSCM) establishes the SwCI based on mishap severity categories associated with the SW SSF and SCCs
- The SwCIs are used to define the required LOR tasks based on MIL-STD-882E definitions
## Software Safety Criticality Matrix

<table>
<thead>
<tr>
<th>Software Control Category</th>
<th>Catastrophic (1)</th>
<th>Critical (2)</th>
<th>Marginal (3)</th>
<th>Negligible (4)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>SwCI 1</td>
<td>SwCI 1</td>
<td>SwCI 3</td>
<td>SwCI 4</td>
</tr>
<tr>
<td>2</td>
<td>SwCI 1</td>
<td>SwCI 2</td>
<td>SwCI 3</td>
<td>SwCI 4</td>
</tr>
<tr>
<td>3</td>
<td>SwCI 2</td>
<td>SwCI 3</td>
<td>SwCI 4</td>
<td>SwCI 4</td>
</tr>
<tr>
<td>4</td>
<td>SwCI 3</td>
<td>SwCI 4</td>
<td>SwCI 4</td>
<td>SwCI 4</td>
</tr>
<tr>
<td>5</td>
<td>SwCI 5</td>
<td>SwCI 5</td>
<td>SwCI 5</td>
<td>SwCI 5</td>
</tr>
</tbody>
</table>

### Level of Rigor

<table>
<thead>
<tr>
<th>SwCI</th>
<th>Level of Rigor</th>
</tr>
</thead>
<tbody>
<tr>
<td>SwCI 1</td>
<td>Program shall perform analysis of requirements, architecture, design, and code; and conduct in-depth safety-specific testing.</td>
</tr>
<tr>
<td>SwCI 2</td>
<td>Program shall perform analysis of requirements, architecture, and design; and conduct in-depth safety-specific testing.</td>
</tr>
<tr>
<td>SwCI 3</td>
<td>Program shall perform analysis of requirements and architecture, and conduct in-depth safety-specific testing.</td>
</tr>
<tr>
<td>SwCI 4</td>
<td>Program shall conduct safety-specific testing.</td>
</tr>
<tr>
<td>SwCI 5</td>
<td>Once assessed by safety engineering as Not Safety, then no safety specific analysis or verification is required.</td>
</tr>
</tbody>
</table>
MIL-STD-882E describes the software safety in the context of “software contribution to system risk”

- To avoid the misconception that software analyses are performed without system context
- To ensure all software safety issues have clearly defined system mishap context

What is the process for defining software contribution to system risk?
To define software contribution to system risk:

1. Flow the system level SSFs to the software function
   - Determines which software functions contribute to the SSF
   - Performed during the FHA (typically) or PHA
2. Evaluate each software-safety function for mishap severity potential (i.e., Catastrophic, Critical, Marginal, or Negligible)
   - Utilizing the hazards from previous analyses (e.g., PHA, SSHA)
3. Evaluate each software safety function for level of autonomy (i.e., SCC 1-5)
4. Derive the SwCI and associated LOR tasks using MIL-STD-882E Table V
5. Execute the LOR:
   - Safety in software design, development, and verification processes
   - Software safety analytical and verification tasks
6. All identified risk is defined and associated to hazard and system level mishaps
To define software contribution to system risk:

1. Flow the system level SSFs to the software function
   - Determines which software functions contribute to the SSF
   - Performed during the FHA (typically) or PHA

2. Evaluate each software-safety function for mishap severity potential (i.e., Catastrophic, Critical, Marginal, or Negligible)
   - Utilizing the hazards from previous analyses (e.g., PHA, SSHA)

3. Evaluate each software safety function for level of autonomy (i.e., SCC 1-5)

4. Derive the SwCI and associated LOR tasks using MIL-STD-882E Table V

5. Execute the Level of Rigor:
   - Safety in software design, development, and verification processes
   - Software safety analytical and verification tasks

6. All identified risk is defined and associated to hazard and system level mishaps
VIPPS is a Mock System used for training purposes only.

Gun Subsystem Block Diagram

- Shoot-It Custom Gun Control Computer
- Gun Assembly Station (GAS)
- Sensor Suite
- Laser Pointing Device
- Gun Motor
- Clutch
- Ammunition Booster Motor
- Ammunition Supply
Flow SSF to Software Function

To flow Safety Significant Functionality to the software functions, you must first understand how the software works (functionally).

- The gun receives digital messages from the C2 called External Functions (EFs), the messages are processed by the Gun Control Computer (GCC).
- Each EF is intended for a specific gun as defined in the Interface Design Specification (IDS):
  - EF 04  Illegal Message
  - EF 21  Gun Subsystem Select (apply power to internal components)
  - EF 23  Safe/Deselect (power off internal components)
  - EF 25  Master Reset
  - EF 27  BIT On
  - EF 33  Activate Laser Aiming Device
  - EF 35  De-Activate Laser Aiming Device
  - EF 37  Command Gun Azimuth/Elevation
  - EF 39  Fire Command
The gun executes as appropriate (e.g., if valid) and responds to the C2 with digital information as appropriate. The messages from the Gun to the C2 are called IDs.

Each ID provides status or data as defined in the IDS:

- ID 04 Illegal Message
- ID 06 Gun Subsystem Select Acknowledge
- ID 12 Status Message (periodic)
- ID 14 Deselect Response
- ID 16 GSS Initialized
- ID 20 BITE Response
- ID 26 Video Tracking/Positional Data
- ID 30 Round Fired
Flow SSF to Software Function

Software Functions

- Software Functions within each Computer Software Configuration Item (CSCI) is typically defined during the software development process. For the VIPPS Gun the software functions are:

  - Main Gun
  - Select Gun
  - Monitor Hazard
  - Process EF_ID
  - CRC
  - Validate MSG
  - Verify GunID
  - GetGunIDfrmMSG
  - Move Gun
  - Deactivate Gun
  - Fire Command
  - Master Reset
  - Bite On
  - Activate Laser Aiming Device
  - De-Activate Laser Aiming Device

All within the “Gun Controller” CSCI
Flow SSF to Software Function

Basic Functionality of the Gun Software

Shoot-It Custom Gun Control Computer (Unit Level)

Operating System

Main Gun

Monitor Hazard

Process EF_ID

CRC

Validate MSG

GetGunID frmMsg

Deactivate Gun

Select Gun

Master Reset

BITE On

Activate Laser Aiming Device

Fire Command

Move Gun

De Activate Laser Aiming Device

Gun Controller CSCI (CSCI Level)

Gun Subsystem (Subsystem Level)

Power Control

Other Components

(System Level)

C2 Subsystem

24v

VIPPS is a Mock System used for training purposes only

LEGEND

Software Function

Computer Based Functionality

Analogue Interface

Software message interface

Electrical

NSWCDD-PN-14-00295 is approved for Distribution Statement A:
Approved for Public Release; distribution is unlimited.
Overview of VIPPS Safety-Significant Functions

SSF-1: System Power On / Initialize/ Monitor Config

SSF-2: State Transitions

SSF-3: Exclusion Zones

SSF-4: Operator Situational Awareness

SSF-5: Threat Database

SSF-6: Verification of Gun Activation Deactivation and Movement Commands

SSF-7: Calibration Verifications

SSF-8: Track ID

SSF-9: Track Correlation

SSF-10: Sensor Activation, Deactivation, and Movement

SSF-11: Threat Assessment

SSF-12: Hailer Calibration

SSF-13: Hailer Arm/Disarm

SSF-14: Hailer Activation

SSF-15: High Intensity Light Calibration

SSF-16: High Intensity Light Arm / Disarm

SSF-17: High Intensity Light Activation, Deactivation, and Movement

SSF-18: Gun Calibration

SSF-19: Gun Arm/Disarm

SSF-20: Gun Activate, Deactivate, and Movement

VIPPS is a Mock System used for training purposes only
### Software Criticality Analysis Worksheet

**STUDENT HANDOUT - POPULATE 2.d - 2.i**

<table>
<thead>
<tr>
<th>2.a Safety Significant Function</th>
<th>2.b Safety Significant Function (Portion)</th>
<th>2.c CSCI</th>
<th>2.d Software Function</th>
<th>2.e Related Hazard Number from VIPPS Worksheet</th>
<th>2.f Level of Mishap Severity for the Identified Hazard</th>
<th>2.g Software Control Category</th>
<th>2.h SwCI</th>
<th>2.i Level of Rigor Tasks Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSF #6: Verification of Gun activation / deactivate and movement Commands</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

NSWCDD-PN-14-00295 is approved for Distribution Statement A: Approved for Public Release; distribution is unlimited.
Overview of VIPPS Safety - Significant Functions

- SSF-6: Verification of Gun Activation Deactivation and Movement Commands
  - Verify Gun Activation
  - Verify Gun Deactivation
  - Verify Gun Movement

VIPPS is a Mock System used for training purposes only
<table>
<thead>
<tr>
<th>2.a</th>
<th>2.b</th>
<th>2.c</th>
<th>2.d</th>
<th>2.e</th>
<th>2.f</th>
<th>2.g</th>
<th>2.h</th>
<th>2.i</th>
</tr>
</thead>
<tbody>
<tr>
<td>Safety- Significant Function</td>
<td>Safety- Significant Function (Portion)</td>
<td>CSCI</td>
<td>Software Function</td>
<td>Related Hazard Number from VIPPS Worksheet</td>
<td>Level of Mishap Severity for the Identified Hazard</td>
<td>Software Control Category</td>
<td>SwCI</td>
<td>Level of Rigor Tasks Required</td>
</tr>
</tbody>
</table>

SSF #6: Verification of Gun activation / deactivate and movement Commands
Verify Activate
Identify Software Functions Associated with Gun Activate

Activate
(e.g., Fire on cmd only if GSS address in cmd matches)

SSF-6: Verification of activation / deactivate and movement Commands

OPERATOR / C2 LEVEL
Operate Gun

Operator
Designate Gun (x) to position

C2
Command Gun (x) to desired position via EF37

Operator
Command Fire Gun (x)

C2
Command Gun (x) to Fire

GUN SUBSYSTEM LEVEL

GCC
Cmd GCU move gun to required azimuth and elevation

GCC
Execute Fire Command (GSS ADRS (x))

GUN SOFTWARE FUNCTIONAL LEVEL

Process EF_ID
Receive and process EF commands

Validate MSG
Validate command in sequence and intended for ADRS Gun

VIPPS is a Mock System used for training purposes only

NSWCDD-PN-14-00295 is approved for Distribution Statement A:
Approved for Public Release; distribution is unlimited.
Flow SSF to Software Function

Basic Functionality of the Gun Software

Operating System
Shoot-It Custom Gun Control Computer (Unit Level)

Monitor Hazard
Process EF_ID
Main Gun
CRC
Validate MSG
GetGunID frmMsg
Deactivate Gun
Master Reset
Select Gun
BITE On
Move Gun
Activate Laser Aiming Device
Fire Command
De Activate Laser Aiming Device

Gun Controller CSCI (CSCI Level)

Gun Subsystem (Subsystem Level)
Power Control

Other Components

24v
C2 Subsystem

LEGEND
Software Function
Computer Based Functionality
Analog Interface
Software message interface
Electrical

VIPPS is a Mock System used for training purposes only

NSWCDD-PN-14-00295 is approved for Distribution Statement A:
Approved for Public Release; distribution is unlimited.

37
<table>
<thead>
<tr>
<th>2.a Safety-Significant Function</th>
<th>2.b Safety-Significant Function (Portion)</th>
<th>2.c CSCI</th>
<th>2.d Software Function</th>
<th>2.e Related Hazard Number from VIPPS Worksheet</th>
<th>2.f Level of Mishap Severity for the Identified Hazard</th>
<th>2.g Software Control Category</th>
<th>2.h SwCI</th>
<th>2.i Level of Rigor Tasks Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSF #6: Verification of Gun activation / deactivate, and movement Commands</td>
<td>Verify Activate (Fire on fire cmd only if GSS ADDRESS in fire cmd matches)</td>
<td>Gun Controller</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**SSF #6:**
- **Verification of Gun activation / deactivate, and movement Commands**
- **Verify Activate** (Fire on fire cmd only if GSS ADDRESS in fire cmd matches)
- **Gun Controller**
Identify Software Functions Associated with Gun Activate

**Activate**
*(e.g., Fire on cmd only if GSS address in cmd matches)*

**SSF-6:** Verification of Gun activation/deactivate and movement Commands

**OPERATOR / C2 LEVEL**

**Operate Gun**

- **Operator**
  - Designate Gun (x) to position

- **C2**
  - Command Gun (x) to desired position via EF37

**GUN SUBSYSTEM LEVEL**

- **GCC**
  - Cmd GCU move gun to required azimuth and elevation

- **GCC**
  - Periodic - convert video signal to digital and send via ID 26

**GUN SOFTWARE FUNCTIONAL LEVEL**

- **Process EF_ID**
  - Receive and process EF commands

- **Validate MSG**
  - Validate command in sequence and intended for ADRS Gun

**EXECUTE FIRE COMMAND**

- **GSS ADRS (x)**

**VIPPS is a Mock System used for training purposes only**
<table>
<thead>
<tr>
<th>2.a Safety-Significant Function</th>
<th>2.b Safety-Significant Function (Portion)</th>
<th>2.c CSCI</th>
<th>2.d Software Function</th>
<th>2.e Related Hazard Number from VIPPS Worksheet</th>
<th>2.f Level of Mishap Severity for the Identified Hazard</th>
<th>2.g Software Control Category</th>
<th>2.h SwCI</th>
<th>2.i Level of Rigor Tasks Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSF #6: Verification of Gun activation / deactivate, and movement Commands</td>
<td>Verify Activate (Fire on fire cmd only if GSS ADDRESS in fire cmd matches)</td>
<td>Gun Controller</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

NSWCDD-PN-14-00295 is approved for Distribution Statement A: Approved for Public Release; distribution is unlimited.
<table>
<thead>
<tr>
<th>2.a Safety-Significant Function</th>
<th>2.b Safety-Significant Function (Portion)</th>
<th>2.c CSCI</th>
<th>2.d Software Function</th>
<th>2.e Related Hazard Number from VIPPS Worksheet</th>
<th>2.f Level of Mishap Severity for the Identified Hazard</th>
<th>2.g Software Control Category</th>
<th>2.h SwCI</th>
<th>2.i Level of Rigor Tasks Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSF #6: Verification of Gun activation / deactivate, and movement Commands</td>
<td>Verify Activate (Fire on fire cmd only if GSS ADDRESS in fire cmd matches)</td>
<td>Gun Controller</td>
<td>Process EF_ID</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

SSF #6: Verification of Gun activation / deactivate, and movement Commands

- Gun Controller
- Process EF_ID
- Validate MSG
To define software contribution to system risk:

1. Flow the system level SSFs to the software function
   - Determines which software functions contribute to the SSF
   - Performed during the FHA (typically) or PHA

2. Evaluate each software-safety function for mishap severity potential (i.e., Catastrophic, Critical, Marginal, or Negligible)
   - Utilizing the hazards from previous analyses (e.g., PHA, SSHA)

3. Evaluate each software-safety function for level of autonomy (i.e., SCC 1-5)

4. Derive the SwCI and associated LOR tasks using MIL-STD-882E Table V

5. Execute the Level of Rigor:
   - Safety in software design, development, and verification processes
   - Software safety analytical and verification tasks

6. All identified risk is defined and associated to hazard and system level mishaps
## Evaluate SW Function for Severity

<table>
<thead>
<tr>
<th>Haz ID #</th>
<th>Phase of Operation</th>
<th>State/Mode</th>
<th>Hazard Description</th>
<th>Causal Factors</th>
<th>Mishap</th>
<th>Effects</th>
<th>Hazard Controls</th>
<th>MRI</th>
<th>System</th>
<th>Subsystem</th>
<th>Component(s)</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>132</td>
<td>Operational</td>
<td>Tactical</td>
<td>Upon receipt detection of GSS Electronics overtemperature, the GCC fails to halt processing after 120 seconds (if C2 does not safe/deselect)</td>
<td>GCC fails to activate 120 second timer or terminate processing in response to 120 second timeout</td>
<td>Electronics overheating, causing circuit board damage</td>
<td>Equipment damage</td>
<td>The GSS shall continuously monitor for a GSS over-temperature condition [electronics]. If a GSS over-temperature condition is detected and continues to exist 1.0 + 0.025 seconds after initial detection, the GSS shall update the status register indicating GSS Over-temperature for transfer to the C2 within the ID 12, and continue operation until discontinued by the C2.</td>
<td>3D</td>
<td>VIPPS</td>
<td>Gun Subsystem</td>
<td>Gun Control Computer</td>
<td>SSF 20</td>
</tr>
<tr>
<td>188</td>
<td>Operational</td>
<td>Tactical</td>
<td>Upon receipt of EF23 (Safe/ Deselect), the GSS fails to remove power to the gun motor causing premature motor failure</td>
<td>Software Failure in the Gun Control Computer</td>
<td>Premature motor failure</td>
<td>Equipment damage</td>
<td>None</td>
<td>3D</td>
<td>VIPPS</td>
<td>Gun Subsystem</td>
<td>Gun Control Computer</td>
<td>SSF 20</td>
</tr>
<tr>
<td>299</td>
<td>Operation</td>
<td>Tactical</td>
<td>Fire command processed by the wrong gun results in accidental firing by the (wrong) gun</td>
<td>The GCC software fails to validate firing commands intended for the gun, and processes commands intended for a different gun</td>
<td>Engage Friendly/non-hostile target</td>
<td>Personnel injury or death</td>
<td>[4.6.5.h] VIPPS [GSS] software shall prevent the lethal effector from being fired without a proper command and C2 shall provide an alarm to the operator when an attempt is made to fire the lethal effector without a proper command. If the GSS receives an EF command with the GSS Address signal set to an illegal value, the GSS shall discard the command and transmit an Illegal Message (GSS-C2-ID04) to the C2 with the Malfunction Type signal set to 5 (Wrong GSS Address).</td>
<td>1D</td>
<td>VIPPS</td>
<td>Gun Subsystem</td>
<td>Gun Control Computer</td>
<td>SSF 6</td>
</tr>
<tr>
<td>310</td>
<td>Operation</td>
<td>Test</td>
<td>GSS fails to move at reduced rate of movement, moving the gun at &quot;normal&quot; rates when &quot;reduced speed&quot; is ordered, leading to accidental impact of gun with personnel</td>
<td>The GCC software fails to validate 'rate of gun movement' before commanding gun to move at normal speed.</td>
<td>Inadvertent physical contact of gun with personnel</td>
<td>Personnel injury or death</td>
<td>Upon receipt of gun pointing data (EF 37), the GSS shall slew the gun barrel to the pointing location (relative azimuth and elevation) as designated in the message. The GSS shall move the gun at normal speed (5 ms per degree) or reduced speed (60 ms per degree) as specified in the message.</td>
<td>1D</td>
<td>VIPPS</td>
<td>Gun Subsystem</td>
<td>Gun Control Computer</td>
<td>SSF 6</td>
</tr>
</tbody>
</table>

**VIPPS is a Mock System used for training purposes only**
## Software Criticality Analysis Worksheet

### STUDENT HANDOUT - POPULATE 2.d - 2.i

<table>
<thead>
<tr>
<th>2.a Safety- Significant Function</th>
<th>2.b Safety- Significant Function (Portion)</th>
<th>2.c CSCI</th>
<th>2.d Software Function</th>
<th>2.e Related Hazard Number from VIPPS Worksheet</th>
<th>2.f Level of Mishap Severity for the Identified Hazard</th>
<th>2.g Software Control Category</th>
<th>2.h SwCl</th>
<th>2.i Level of Rigor Tasks Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>Safety Function</td>
<td>Safety Function (Portion)</td>
<td>CSCI</td>
<td>Software Function</td>
<td>Related Hazard Number from VIPPS Worksheet</td>
<td>Level of Mishap Severity for the Identified Hazard</td>
<td>Software Control Category</td>
<td>SwCl</td>
<td>Level of Rigor Tasks Required</td>
</tr>
</tbody>
</table>

SSF #6: Verification of Gun activation / deactivate, and movement Commands

- **Verify Activate**
  - (Fire on fire cmd only if GSS ADDRESS in fire cmd matches)
- **Gun Controller**
  - Process EF_ID
  - Validate MSG
Gun Controller software fails to validate firing commands intended for the gun and processes commands intended for different gun.

**Hazard ID 299**

<table>
<thead>
<tr>
<th>Hazard ID</th>
<th>Phase of Test</th>
<th>State</th>
<th>Hazard</th>
<th>Causal</th>
<th>Mishap</th>
<th>Effects</th>
<th>MRI</th>
<th>System</th>
<th>Subsystem</th>
<th>Component(s)</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>299</td>
<td>Operation</td>
<td>Tactical</td>
<td>Fire command processed by the wrong gun results in accidental firing by the (wrong) gun</td>
<td>The GCC software fails to validate firing commands intended for the gun, and processes commands intended for a different gun</td>
<td>Engage Friendly/Non-hostile target</td>
<td>Personnel injury or death</td>
<td>4.6.6.6</td>
<td>VIPPS</td>
<td>Gun Subsystem</td>
<td>SSF 5</td>
<td>VIPPS is a Mock System used for training purposes only</td>
</tr>
<tr>
<td>310</td>
<td>Operation</td>
<td>Test</td>
<td>GSS fails to move at reduced rate of movement, moving the gun at &quot;normal&quot; rates when &quot;reduced speed&quot; is ordered, leading to accidental impact of gun with personnel</td>
<td>The GCC software fails to validate ‘rate of gun movement’ before commanding gun to move at normal speed.</td>
<td>Indecent physical contact of gun with personnel.</td>
<td>Personnel injury or death</td>
<td>Upon receipt of gun pointing data (EF 37), the GSS shall slew the gun barrel to the pointing location (relative azimuth and elevation) as designated in the message. The GSS shall move the gun at normal speed (5 ms per degree) or reduced speed (60 ms per degree) as specified in the message.</td>
<td>1D</td>
<td>VIPPS</td>
<td>Gun Subsystem</td>
<td>SSF 5</td>
</tr>
</tbody>
</table>

VIPPS is a Mock System used for training purposes only.
## Software Criticality Analysis Worksheet

### STUDENT HANDOUT - POPULATE 2.d - 2.i

#### 2.e and 2.f Answer

<table>
<thead>
<tr>
<th>2.a Safety Significant Function</th>
<th>2.b Safety Significant Function (Portion)</th>
<th>2.c CSCI</th>
<th>2.d Software Function</th>
<th>2.e Related Hazard Number from VIPPS Worksheet</th>
<th>2.f Level of Mishap Severity for the Identified Hazard</th>
<th>2.g Software Control Category</th>
<th>2.h SwCI</th>
<th>2.i Level of Rigor Tasks Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSF #6: Verification of Gun activation / deactivate, and movement Commands</td>
<td>Verify Activate (Fire on fire cmd only if GSS ADDRESS in fire cmd matches)</td>
<td>Process EF_ID</td>
<td>299</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Gun Controller</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Validate _ MSG</td>
<td>299</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

SSF #6: Verification of Gun activation / deactivate, and movement Commands

Verify Activate (Fire on fire cmd only if GSS ADDRESS in fire cmd matches)

Process EF_ID

Gun Controller

Validate _ MSG

Related Hazard Number from VIPPS Worksheet: 299

Level of Mishap Severity for the Identified Hazard: 1
Software Contribution to System Risk

- To define software contribution to system risk:
  - Flow the system level SSFs to the software function
    - Determines which software functions contribute to the SSF
    - Performed during the FHA (typically) or PHA
  - Evaluate each software-safety function for mishap severity potential (i.e., Catastrophic, Critical, Marginal, or Negligible)
    - Utilizing the hazards from previous analyses (e.g., PHA, SSHA)
  3. Evaluate each software-safety function for level of autonomy (i.e., SCC 1-5)
  4. Derive the SwCI and associated LOR tasks using MIL-STD-882E Table V
  5. Execute the LOR:
    - Safety in software design, development, and verification processes
    - Software safety analytical and verification tasks
  6. All identified risk is defined and associated to hazard and system level mishaps
BREAK
Software Contribution to System Risk

To define software contribution to system risk:

- Flow the system level SSFs to the software function
  - Determines which software functions contribute to the SSF
  - Performed during the FHA (typically) or PHA
- Evaluate each software-safety function for mishap severity potential (i.e., Catastrophic, Critical, Marginal, or Negligible)
  - Utilizing the hazards from previous analyses (e.g., PHA, SSHA)

3. Evaluate each software-safety function for level of autonomy (i.e., SCC 1-5)

4. Derive the SwCI and associated LOR tasks using MIL-STD-882E Table V

5. Execute the LOR:
   - Safety in software design, development, and verification processes
   - Software safety analytical and verification tasks

6. All identified risk is defined and associated to hazard and system level mishaps
# Software Control Categories

<table>
<thead>
<tr>
<th>Level</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
</table>
| 1     | Autonomous (AT)               | - Autonomous CONTROL over safety hardware systems, and  
                  - No possibility of detection and intervention by control entity                                                                                                                                 |
| 2     | Semi-Autonomous (SAT)         | - CONTROL over safety hardware systems, and  
                  - Time for detection and intervention by INDEPENDENT safety mechanism  
                  - SW that displays safety-significant data for immediate predetermined operator action to prevent mishap, or  
                  - The SW fault or delay will allow, or fail to prevent, the mishap |
| 3     | Redundant Fault Tolerant (RFT)| - Issues commands over safety hardware system, and  
                  - Requires control entity to complete command function, and  
                  - Includes REDUNDANT, INDEPENDENT fault tolerant mechanisms  
                  - Generates information of safety-critical nature to make decisions, and  
                  - System includes SEVERAL REDUNDANT, INDEPENDENT fault tolerant mechanisms |
| 4     | Influential                   | - Generates information of safety-related nature to make decisions by the operator, and  
                  - Does not require operator action to avoid mishap |
| 5     | No Safety Impact (NSI)        | Software functionality that does not possess command or control authority over safety-significant hardware systems, and does not provide safety-significant information 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. |
VIPPS Interlocks

VIPPS is a Mock System used for training purposes only

NSWCDD-PN-14-00295 is approved for Distribution Statement A:
Approved for Public Release; distribution is unlimited.
Identify Hazards Associated Identified Software Functions

SSF-6: Verification of Gun activation/deactivate and movement Commands

Verify Activate
(Fire on fire cmd only if GSS ADDRESS” in fire cmd matches)

PROCESS EF_ID FUNCTIONAL DESCRIPTION
Process EF_ID function performs initial processing of gun command messages by verifying message transfer using a Cyclic Redundancy Check (CRC) then sending the message to the Validate Msg Function.

OPERATOR / C2 LEVEL
Operator
Command Gun (x) to Fire

GUN subsystem LEVEL
GSS
Cmd GCU move gun to required azimuth and elevation

GUN SOFTWARE FUNCTIONAL LEVEL
GCC
Periodic - convert video signal to digital and send via ID 26

VIPPS is a Mock System used for training purposes only

Operator
Command Fire Gun (x)

C2
Command Gun (x) to Fire

GCC
Execute Fire Command (GSS ADRS (x))

Process EF_ID
Receive and process EF commands

Validate MSG
Validate command in sequence and intended for ADRS Gun

NSWCDD-PN-14-00295 is approved for Distribution Statement A:
Approved for Public Release; distribution is unlimited.
Software Control Associated with System Hazard

- Process EF_ID verifies message content including a valid Gun ID
- Fire gun with incorrect Gun Address
  - AND
  - Wrong Gun Address passes verification checks (as valid Fire Command EF)
    - OR
    - Gun ID corrupted, valid and undetected (C2 or GCC)
      - CRC Failure (C2 or GCC)
    - Gun ID set correctly for different gun
    - Gun ID set incorrectly for intended gun
      - OR
      - Failure of operator
      - Failure of C2
    - Failure of gun software to validate EF gun address against intended gun address for match
      - OR
      - Coding Error
      - Incorrect designation in gun specific datafile (e.g., wrong gun number in adaptation data)

Validated MSG ensures Gun ID correct for intended Gun

VIPPS is a Mock System used for training purposes only
# Software Control Categories

<table>
<thead>
<tr>
<th>Level</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Autonomous (AT)</td>
<td>Software functionality that exercises autonomous control authority over potentially safety significant hardware systems, and no possibility of detection and intervention by control entity</td>
</tr>
<tr>
<td>2</td>
<td>Semi-Autonomous (SAT)</td>
<td>Software functionality that exercises control authority over potentially safety significant hardware systems, and time for detection and intervention by independent safety mechanism. SW that displays safety-significant data for immediate predetermined operator action to prevent mishap, or The SW fault or delay will allow, or fail to prevent, the mishap</td>
</tr>
<tr>
<td>3</td>
<td>Redundant Fault Tolerant (RFT)</td>
<td>Issues commands over safety hardware system, and requires control entity to complete command function, and includes redundant, independent fault tolerant mechanisms. Generates information of safety-critical nature to make decisions, and system includes several redundant, independent fault tolerant mechanisms</td>
</tr>
<tr>
<td>4</td>
<td>Influential</td>
<td>Software generates information of safety-related nature to make decisions by the operator, and does not require operator action to avoid mishap</td>
</tr>
<tr>
<td>5</td>
<td>No Safety Impact (NSI)</td>
<td>Software functionality that does not possess command or control authority over safety significant hardware systems, and does not provide safety significant information. Software does not transport or resolve communication of safety-significant or time sensitive data.</td>
</tr>
</tbody>
</table>

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. 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.

Sw that displays safety-significant data for immediate predetermined operator action to prevent mishap, or The SW fault or delay will allow, or fail to prevent, the mishap

Issues commands over safety hardware system, and requires control entity to complete command function, and includes redundant, independent fault tolerant mechanisms. Generates information of safety-critical nature to make decisions, and system includes several redundant, independent fault tolerant mechanisms

Generates information of safety-related nature to make decisions by the operator, and does not require operator action to avoid mishap

Software functionality that does not possess command or control authority over safety significant hardware systems, and 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>
<thead>
<tr>
<th>2.a Safety-Function</th>
<th>2.b Safety-Function (Portion)</th>
<th>2.c CSCI</th>
<th>2.d Software Function</th>
<th>2.e Related Hazard Number from VIPPS Worksheet</th>
<th>2.f Level of Mishap Severity for the Identified Hazard</th>
<th>2.g Software Control Category</th>
<th>2.h SwCI</th>
<th>2.i Level of Rigor Tasks Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSF #6: Verification of Gun activation / deactivate, and movement Commands</td>
<td>Activate (Fire on fire cmd only if GSS ADDRESS&quot; in fire cmd matches)</td>
<td>Gun Controller</td>
<td>Process ID</td>
<td>299</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Validate MSG</td>
<td>299</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

SSF #6: Verification of Gun activation / deactivate, and movement Commands
Activate (Fire on fire cmd only if GSS ADDRESS" in fire cmd matches)
<table>
<thead>
<tr>
<th>Level</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Autonomous (AT)</td>
<td>• Autonomous CONTROL over safety hardware system, and No possibility of detection and intervention by control entity.</td>
</tr>
<tr>
<td>2</td>
<td>Semi-Autonomous (SAT)</td>
<td>• CONTROL over safety hardware system, and Time for detection and intervention by INDEPENDENT safety mechanisms.</td>
</tr>
<tr>
<td>3</td>
<td>Redundant Fault Tolerant (RFT)</td>
<td>• Issues commands over safety hardware system, and Requires control entity to complete command function, and Includes REDUNDANT, INDEPENDENT fault tolerant mechanisms.</td>
</tr>
<tr>
<td>4</td>
<td>Influential</td>
<td>• Generates information of safety-critical nature to make decisions, and System includes SEVERAL REDUNDANT, INDEPENDENT fault tolerant mechanisms.</td>
</tr>
<tr>
<td>5</td>
<td>No Safety Impact (NSI)</td>
<td>• Generates information of safety-related nature to make decisions BY THE OPERATOR, and Does NOT require operator action to avoid mishap.</td>
</tr>
</tbody>
</table>

**Software Control Categories**

Software that generates information of a safety-critical nature used to make decisions. The system includes several redundant, independent fault tolerant mechanisms for each hazardous condition, detection, and display.

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.

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.

Software items that display safety-critical 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. 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.

Software items that issue commands over safety-significant hardware systems, or transport or resolve communication of safety-significant or time sensitive data. Software does not display safety-critical information.

Software items that display safety-critical information to operators. This includes separate fault tolerant mechanisms for each hazardous condition and displayed information. Software does not generate information to make decisions.

Software that displays safety-significant information requiring immediate operator entity to execute a predetermined action to prevent a mishap. Software exception, failure, fault, or delay will allow, or fail to prevent, the mishap occurrence. 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.

Approved for Public Release; distribution is unlimited.
## Software Control Categories

<table>
<thead>
<tr>
<th>Level</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Autonomous (AT)</td>
<td>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. (This definition includes complex system/software functionality with multiple subsystems, interacting parallel processors, multiple interfaces, and safety-critical functions that are time critical.)</td>
</tr>
<tr>
<td>2</td>
<td>Semi-Autonomous (SAT)</td>
<td>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. (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.) 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. (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.)</td>
</tr>
<tr>
<td>3</td>
<td>Redundant Fault Tolerant (RFT)</td>
<td>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. (This definition assumes that there is adequate fault detection, annunciation, 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.) 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. • <strong>Verifies or validates</strong> information of safety-critical nature to make decisions • System includes SEVERAL REDUNDANT, INDEPENDENT fault tolerant mechanisms • Generates information of safety-critical nature to make decisions • System includes SEVERAL REDUNDANT, INDEPENDENT fault tolerant mechanisms</td>
</tr>
<tr>
<td>4</td>
<td>Influential</td>
<td>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.</td>
</tr>
<tr>
<td>5</td>
<td>No Safety Impact (NSI)</td>
<td>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.</td>
</tr>
</tbody>
</table>
# Software Control Categories

<table>
<thead>
<tr>
<th>Level</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Autonomous (AT)</td>
<td>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. (This definition includes complex system/software functionality with multiple subsystems, interacting parallel processors, multiple interfaces, and safety-critical functions that are time critical.)</td>
</tr>
<tr>
<td>2</td>
<td>Semi-Autonomous (SAT)</td>
<td>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. (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.)</td>
</tr>
<tr>
<td>3</td>
<td>Redundant Fault Tolerant (RFT)</td>
<td>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. (This definition assumes that there is adequate fault detection, annunciation, 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.)</td>
</tr>
<tr>
<td>4</td>
<td>Influential</td>
<td>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.</td>
</tr>
<tr>
<td>5</td>
<td>No Safety Impact (NSI)</td>
<td>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.</td>
</tr>
<tr>
<td>2.a Safety- Significant Function</td>
<td>2.b Safety- Significant Function (Portion)</td>
<td>2.c CSCI</td>
</tr>
<tr>
<td>---------------------------------</td>
<td>------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>SSF #6: Verification of Gun activation / deactivate, and movement Commands</td>
<td>Activate (Fire on fire cmd only if GSS ADDRESS&quot; in fire cmd matches)</td>
<td>Gun Controller</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

SSF #6: Verification of Gun activation / deactivate, and movement Commands (Fire on fire cmd only if GSS ADDRESS" in fire cmd matches)
Identify Hazards Associated Identified Software Functions

Verify Activate
(Fire on fire cmd only if GSS ADDRESS in fire cmd matches)

SSF-6: Verification of Gun activation/deactivate and movement Commands

VALIDATE MSG FUNCTIONAL DESCRIPTION
Validate MSG function checks that each message is valid by evaluating content for assigned Gun ID, boundary conditions, and message sequence based on last message received. If the message is validated, the appropriate action function is called based on message content (e.g., Select Gun, Deactivate Gun, Fire Command, Master Reset, BIT On, Activate Laser Aiming Device, De-Activate Laser Aiming Device or Move Gun).

VIPPS is a Mock System used for training purposes only.

NSWCDD-PN-14-00295 is approved for Distribution Statement A: Approved for Public Release; distribution is unlimited.
Software Control Associated with System Hazard

Process EF_ID verifies message content including a valid Gun ID

- **Fire gun with incorrect Gun Address**

  **AND**

  - **Wrong Gun Address passes verification checks (as valid Fire Command EF)**
    - **OR**
      - Gun ID corrupted, valid and undetected (C2 or GCC)
      - Gun ID set correctly for different gun
      - Gun ID set incorrectly for intended gun
    - OR
      - CRC Failure (C2 or GCC)
      - Failure of operator
      - Failure of C2
  - **OR**
    - **Failure of gun software to validate EF gun address against intended gun address for match**
      - Coding Error
      - Incorrect designation in gun specific datafile (e.g., wrong gun number in adaptation data)

Validate MSG ensures Gun ID correct for intended Gun

VIPPS is a Mock System used for training purposes only
<table>
<thead>
<tr>
<th>2.a Safety-Significant Function</th>
<th>2.b Safety-Significant Function (Portion)</th>
<th>2.c CSCI</th>
<th>2.d Software Function</th>
<th>2.e Related Hazard Number from VIPPS Worksheet</th>
<th>2.f Level of Mishap Severity for the Identified Hazard</th>
<th>2.g Software Control Category</th>
<th>2.h SwCI</th>
<th>2.i Level of Rigor Tasks Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSF #6: Verification of Gun activation / deactivate, and movement Commands</td>
<td>Activate (Fire on fire cmd only if GSS ADDRESS&quot; in fire cmd matches)</td>
<td>Gun Controller</td>
<td>Process ID</td>
<td>299</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Validate MSG</td>
<td>299</td>
<td></td>
<td></td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>
### Software Control Categories

<table>
<thead>
<tr>
<th>Level</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Autonomous (AT)</td>
<td>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. (This definition includes complex system/software functionality with multiple subsystems, interacting parallel processors, multiple interfaces, and safety-critical functions that are time critical.)</td>
</tr>
<tr>
<td>2</td>
<td>Semi-Autonomous (SAT)</td>
<td>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. (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.)</td>
</tr>
<tr>
<td>3</td>
<td>Redundant Fault Tolerant (RFT)</td>
<td>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. (This definition assumes that there is adequate fault detection, annunciation, and system recovery to prevent the hazard occurrence if software fails, malfunctions, or degrades. There are redundant and independent sources of safety-critical information, and mitigating functionality can respond within any time-critical period.)</td>
</tr>
<tr>
<td>4</td>
<td>Influential</td>
<td>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.</td>
</tr>
<tr>
<td>5</td>
<td>No Safety Impact (NSI)</td>
<td>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.</td>
</tr>
</tbody>
</table>

- **Verifies or validates** information of safety-critical nature to make decisions
- System **does not** include SEVERAL REDUNDANT, INDEPENDENT fault tolerant mechanisms

---

NSWCDD-PN-14-00295 is approved for Distribution Statement A: Approved for Public Release; distribution is unlimited.
## 2. Student Handout - Populate 2.d - 2.i

### 2.g Answer

<table>
<thead>
<tr>
<th>2.a</th>
<th>2.b</th>
<th>2.c</th>
<th>2.d</th>
<th>2.e</th>
<th>2.f</th>
<th>2.g</th>
<th>2.h</th>
<th>2.i</th>
</tr>
</thead>
<tbody>
<tr>
<td>Safety-</td>
<td>Safety-</td>
<td>CSCI</td>
<td>Software</td>
<td>Related Hazard</td>
<td>Level of Mishap</td>
<td>Software</td>
<td>SwCI</td>
<td>Level of Rigor Tasks</td>
</tr>
<tr>
<td>Significant</td>
<td>Significant</td>
<td>Function</td>
<td>Function</td>
<td>Number from</td>
<td>Severity for the</td>
<td>Control</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Function (Portion)</td>
<td></td>
<td></td>
<td></td>
<td>VIPPS Worksheet</td>
<td>Identified Hazard</td>
<td>Category</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| SSF #6: Verification of Gun activation / deactivate, and movement Commands | Activate (Fire on fire cmd only if GSS ADDRESS" in fire cmd matches) | Gun Controller | Process ID | 299 | 1 | 2 | | |
| Validate MSG | 299 | 1 | 2 | | | | | |
## Software Control Categories

<table>
<thead>
<tr>
<th>Level</th>
<th>Name</th>
<th>Description</th>
<th>Tailoring</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Autonomous (AT)</td>
<td>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. (This definition includes complex system/software functionality with multiple subsystems, interacting parallel processors, multiple interfaces, and safety-critical functions that are time critical.)</td>
<td>Software that generates, verifies or validates information of a safety-critical nature used to make critical decisions. The system does not include several redundant, independent fault tolerant mechanisms for each hazardous condition, detection, and display.</td>
</tr>
<tr>
<td>2</td>
<td>Semi-Autonomous (SAT)</td>
<td>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. (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.) 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. (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.)</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Redundant Fault Tolerant (RFT)</td>
<td>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. (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.) 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.</td>
<td>Software that generates, verifies or validates 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.</td>
</tr>
<tr>
<td>4</td>
<td>Influential</td>
<td>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.</td>
<td>Software that generates, verifies or validates information of a safety-related nature used to make decisions.</td>
</tr>
<tr>
<td>5</td>
<td>No Safety Impact (NSI)</td>
<td>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.</td>
<td></td>
</tr>
</tbody>
</table>
Software Contribution to System Risk

- To define software contribution to system risk:
  - Flow the system level SSFs to the software function
    - Determines which software functions contribute to the SSF
    - Performed during the FHA (typically) or PHA
  - Evaluate each software-safety function for mishap severity potential (i.e., Catastrophic, Critical, Marginal, or Negligible)
    - Utilizing the hazards from previous analyses (e.g., PHA, SSHA)
  - Evaluate each software-safety function for level of autonomy (i.e., SCC 1-5)

4. Derive the SwCI and associated LOR tasks using MIL-STD-882E Table V
5. Execute the LOR:
   - Safety in software design, development, and verification processes
   - Software safety analytical and verification tasks
6. All identified risk is defined and associated to hazard and system level mishaps
### Software Safety Criticality Matrix

<table>
<thead>
<tr>
<th>Software Control Category</th>
<th>Severity Category</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Catastrophic (1)</td>
</tr>
<tr>
<td></td>
<td>Critical (2)</td>
</tr>
<tr>
<td></td>
<td>Marginal (3)</td>
</tr>
<tr>
<td></td>
<td>Negligible (4)</td>
</tr>
<tr>
<td>1</td>
<td>SwCl 1</td>
</tr>
<tr>
<td>2</td>
<td>SwCl 1</td>
</tr>
<tr>
<td>3</td>
<td>SwCl 2</td>
</tr>
<tr>
<td>4</td>
<td>SwCl 3</td>
</tr>
<tr>
<td>5</td>
<td>SwCl 5</td>
</tr>
</tbody>
</table>

Determine Software Criticality Index using MIL-STD-882E Table V.
Determine Software Criticality Index using MIL-STD-882E Table V

**Software Safety Criticality Matrix**

<table>
<thead>
<tr>
<th>Software Control Category</th>
<th>Severity Category</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Catastrophic (1)</td>
</tr>
<tr>
<td></td>
<td>Critical (2)</td>
</tr>
<tr>
<td></td>
<td>Marginal (3)</td>
</tr>
<tr>
<td></td>
<td>Negligible (4)</td>
</tr>
<tr>
<td>1</td>
<td>SwCl 1</td>
</tr>
<tr>
<td>2</td>
<td>SwCl 1</td>
</tr>
<tr>
<td>3</td>
<td>SwCl 2</td>
</tr>
<tr>
<td>4</td>
<td>SwCl 3</td>
</tr>
<tr>
<td>5</td>
<td>SwCl 5</td>
</tr>
</tbody>
</table>

**For Gun Software Function:**

- **Process EF_ID**

- **Validate.MSG**

VIPPS is a Mock System used for training purposes only.
## Software Criticality Analysis Worksheet

<table>
<thead>
<tr>
<th>2.a Safety Significant Function</th>
<th>2.b Safety Significant Function (Portion)</th>
<th>2.c CSCI</th>
<th>2.d Software Function</th>
<th>2.e Related Hazard Number from VIPPS Worksheet</th>
<th>2.f Level of Mishap Severity for the Identified Hazard</th>
<th>2.g Software Control Category</th>
<th>2.h SwCI</th>
<th>2.i Level of Rigor Tasks Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSF #6: Verification of Gun activation / deactivate, and movement Commands</td>
<td>Activate (Fire on fire cmd only if GSS ADDRESS” in fire cmd matches)</td>
<td>Gun Controller</td>
<td>Process EF_ID</td>
<td>299</td>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Validate MSG</td>
<td></td>
<td></td>
<td>299</td>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Determine Software Criticality Index using MIL-STD-882E Table V

<table>
<thead>
<tr>
<th>Software Safety Criticality Matrix</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Software Control Category</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>5</td>
</tr>
</tbody>
</table>

For Gun Software Function:

**Process EF_ID**

VIPPS is a Mock System used for training purposes only.

For Gun Software Function:

**Validate_MSG**

Determine Software Criticality Index using MIL-STD-882E Table V

<table>
<thead>
<tr>
<th>Software Safety Criticality Matrix</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Software Control Category</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>5</td>
</tr>
</tbody>
</table>
### Software Criticality Analysis Worksheet

#### STUDENT HANDOUT - POPULATE 2.d - 2.i

**2.h Answer**

<table>
<thead>
<tr>
<th>2.a Safety-Significant Function</th>
<th>2.b Safety-Significant Function (Portion)</th>
<th>2.c CSCI</th>
<th>2.d Software Function</th>
<th>2.e Related Hazard Number from VIPPS Worksheet</th>
<th>2.f Level of Mishap Severity for the Identified Hazard</th>
<th>2.g Software Control Category</th>
<th>2.h SwCI</th>
<th>2.i Level of Rigor Tasks Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSF #6: Verification of Gun activation / deactivate, and movement Commands</td>
<td>Activate (Fire on fire cmd only if GSS ADDRESS” in fire cmd matches)</td>
<td></td>
<td>Process EF_ID</td>
<td>299</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Gun Controller</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Validate MSG</td>
<td>299</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>
## Level of Rigor

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

- The SwCIs are used to define the level of safety analytical and verification rigor the software must undergo to increase the confidence that the software will perform as specified while reducing the number of contributors to hazards that may exist in the system.
## Software Criticality Analysis Worksheet

<table>
<thead>
<tr>
<th>2.a Safety-Significant Function</th>
<th>2.b Safety-Significant Function (Portion)</th>
<th>2.c CSCI</th>
<th>2.d Software Function</th>
<th>2.e Related Hazard Number from VIPPS Worksheet</th>
<th>2.f Level of Mishap Severity for the Identified Hazard</th>
<th>2.g Software Control Category</th>
<th>2.h SwCI</th>
<th>2.i Level of Rigor Tasks Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSF #6: Verification of Gun activation / deactivate, and movement Commands</td>
<td>Activate (Fire on fire cmd only if GSS ADDRESS&quot; in fire cmd matches)</td>
<td>Gun Controller</td>
<td>Process EF_ID</td>
<td>299</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Validate MSG</td>
<td>299</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>
### Level of Rigor

<table>
<thead>
<tr>
<th>SwCl</th>
<th>Level of Rigor</th>
</tr>
</thead>
<tbody>
<tr>
<td>SwCl 1</td>
<td>Program shall perform analysis of requirements, architecture, design, and code; and conduct in-depth safety-specific testing.</td>
</tr>
<tr>
<td>SwCl 2</td>
<td>Program shall perform analysis of requirements, architecture, and design; and conduct in-depth safety-specific testing.</td>
</tr>
<tr>
<td>SwCl 3</td>
<td>Program shall perform analysis of requirements and architecture, and conduct in-depth safety-specific testing.</td>
</tr>
<tr>
<td>SwCl 4</td>
<td>Program shall conduct safety-specific testing.</td>
</tr>
<tr>
<td>SwCl 5</td>
<td>Once assessed by safety engineering as Not Safety, then no safety specific analysis or verification is required.</td>
</tr>
</tbody>
</table>

- The SwCIs are used to define the level of safety analytical and verification rigor the software must undergo to increase the confidence that the software will perform as specified while reducing the number of contributors to hazards that may exist in the system.
## 2.i Answer

<table>
<thead>
<tr>
<th>2.a</th>
<th>2.b</th>
<th>2.c</th>
<th>2.d</th>
<th>2.e</th>
<th>2.f</th>
<th>2.g</th>
<th>2.h</th>
<th>2.i</th>
</tr>
</thead>
<tbody>
<tr>
<td>Safety Significant Function</td>
<td>Safety Significant Function (Portion)</td>
<td>CSCI</td>
<td>Software Function</td>
<td>Related Hazard Number from VIPPS Worksheet</td>
<td>Level of Mishap Severity for the Identified Hazard</td>
<td>Software Control Category</td>
<td>SwCI</td>
<td>Level of Rigor Tasks Required</td>
</tr>
</tbody>
</table>

| SSF #6: Verification of Gun activation / deactivate, and movement Commands | Activate (Fire on fire cmd only if GSS ADDRESS" in fire cmd matches) | Gun Controller | Validate MSG | 299 | 1 | 2 | 1 |

Program shall perform analysis of requirements, architecture, design, and code; and conduct in-depth safety-specific testing.

Program shall perform analysis of requirements, architecture, design, and code; and conduct in-depth safety-specific testing.
### Software Risk

#### RELATIONSHIP BETWEEN SwCI, RISK LEVEL, LOR TASKS, AND RISK

<table>
<thead>
<tr>
<th>Software Criticality Index (SwCI)</th>
<th>Mishap Risk Level</th>
<th>Software LOR Tasks and Risk Assessment/Acceptance</th>
</tr>
</thead>
<tbody>
<tr>
<td>SwCI 1</td>
<td>HIGH</td>
<td>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 <strong>prepare a formal risk assessment for acceptance of a HIGH risk.</strong></td>
</tr>
<tr>
<td>SwCI 2</td>
<td>SERIOUS</td>
<td>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 <strong>prepare a formal risk assessment for acceptance of a SERIOUS risk.</strong></td>
</tr>
<tr>
<td>SwCI 3</td>
<td>MEDIUM</td>
<td>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 <strong>prepare a formal risk assessment for acceptance of a MEDIUM risk.</strong></td>
</tr>
<tr>
<td>SwCI 4</td>
<td>LOW</td>
<td>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 <strong>prepare a formal risk assessment for acceptance of a LOW risk.</strong></td>
</tr>
<tr>
<td>SwCI 5</td>
<td>Not Safety</td>
<td>No safety-specific analyses or testing is required.</td>
</tr>
</tbody>
</table>
Software Contribution to System Risk

- To define software contribution to system risk:
  1. Flow the system level SSFs to the software function
     - Determines which software functions contribute to the SSF
     - Performed during the FHA (typically) or PHA
  2. Evaluate each software-safety function for mishap severity potential
     (i.e., Catastrophic, Critical, Marginal, or Negligible)
     - Utilizing the hazards from previous analyses (e.g., PHA, SSHA)
  3. Evaluate each software-safety function for level of autonomy
     (i.e., SCC 1-5)
  4. Derive the SwCI and associated LOR tasks using MIL-STD-882E Table V
  5. Execute the LOR:
     - Safety in software design, development, and verification processes
     - Software safety analytical and verification tasks
  6. All identified risk is defined and associated to hazard and system level mishaps

Address Later in the day
Software Safety Analysis and Verification Process

Top-Level Process

Start → System Definition and Software Safety Planning → Determine Software Criticality Index (SwCI) → Software Requirements Hazard Analysis (SwCI 1-3) → Software Architectural Hazard Analysis (SwCI 1-3) → Software Testing and Verification (SwCI 1-4) → Formal Review → Fleet Release

Sub-Process

- Defect Resolution
- Regression Testing
- Fleet Anomaly Reporting

Software Criticality Matrix
SwCI (1,2, and 3) LOR Task: SW Requirements Hazard Analysis

- Safety Requirements Hazard Analysis (SRHA) is performed on SW as part of the Low-Level SRHA to ensure that there are adequate safety requirements associated with safety-significant SW functionality
  - SW Safety Requirements Trace from System-Level > Sub-System Level > Software Requirements. New safety requirements are derived per SRHA process

- Three Categories of SW Safety Requirements are:
  - Initiating Software Safety Requirements (ISSRs)
    - SW Requirements related to the SSFs that may initiate hazards if not defined and implemented appropriately
  - Generic Software Safety Requirements (GSSRs)
    - Are designed features, constraints, development processes and coding standards that are generally used with SW
  - Mitigating Software Safety Requirements (MSSRs)
    - These requirements mitigate or control mishap or hazard causes to acceptable levels of safety risk with regards to the system’s SW

- Safety Requirements Verification Matrix (SRVM)
  - SRVM documents SW Safety Requirements analysis/test results
Software Safety Analysis and Verification Process

Top-Level Process

Start → System Definition and Software Safety Planning

Determining Software Criticality Index (SwCl)

Software Requirements Hazard Analysis (SwCl 1–3)

Software Architectural Hazard Analysis (SwCl 1–3)

Software Design Hazard Analysis (SwCl 1-2)

Code Level Hazard Analysis (SwCl 1)

Operator Documentation Safety Review

Software Testing and Verification (SwCl 1-4)

Safety - Specific Testing (SwCl 4)

In-depth Safety - Specific Testing (SwCl 1-3)

Formal Review

Fleet Release

Sub-Process

Defect Resolution

Regression Testing

Fleet Anomaly Reporting

Software Criticality Matrix

NSWCDD-PN-14-00295 is approved for Distribution Statement A: Approved for Public Release; distribution is unlimited.
Software Safety Analysis and Verification Process

Top-Level Process

Start → System Definition and Software Safety Planning → Determine Software Criticality Index (SwCI) → Software Requirements Hazard Analysis (SwCI 1-3) → Software Architectural Hazard Analysis (SwCI 1-3)

Sub-Process

- Software Design Hazard Analysis (SwCI 1-2)
- Code Level Hazard Analysis (SwCI 1)
- Operator Documentation Safety Review
- Software Testing and Verification (SwCI 1-4)
  - Safety - Specific Testing (SwCI 4)
  - In-depth Safety - Specific Testing (SwCI 1-3)
- Formal Review → Fleet Release

Sub-Processes:
- Defect Resolution
- Regression Testing
- Fleet Anomaly Reporting

Software Criticality Matrix
Outline

1. Course Overview

2. Mock System Overview - Very Important Person Protection System (VIPPS)

3. Software Safety Process
   1. Perform Software Criticality Index and Level of Rigor Assessment
   2. *Perform Architecture and Design Analysis*
   3. Perform Code Analysis (No coding experience necessary)

4. Assess Technology Insertion Proposal

5. Software (SW) Safety Analysis Summary

6. Conclusion
SwCl (1, 2, and 3) LOR Task: Architectural Hazard Analysis

• Architectural Analysis
  ➢ SW Architecture - The organizational structure of a system or Computer Software Configuration Item (CSCI), identifying its components, their interfaces, and concept of execution among them [Reference Allied Ordnance Publication (AOP)-52]
  ➢ Conducting computing system and software architectural hazard analysis:
    • Identify/Define the allocation of System Functions to Architecture
    • Identify/Define the Software and Interface Architecture Requirements
    • Reviewing architecture against software safety-significant requirements (SSRs) and Hazard Tracking Record (HTR) software mitigations to determine which cannot be supported by the current architectures
    • Identify new architecture hazards and define supported mitigations
SW Architectural Diagrams

• Various architectural views per International Organization for Standardization (ISO)/International Electrotechnical Commission (IEC)/Institute of Electrical and Electronics Engineers (IEEE) 42010 (IEEE 1471):
  ➢ Functional/logical viewpoint
  ➢ Code/module viewpoint
  ➢ Development/structural viewpoint
  ➢ Concurrency/process/runtime/thread viewpoint
  ➢ Physical/deployment/installation viewpoint
  ➢ User action/feedback viewpoint
  ➢ Data view/data model

Architectural Analysis is conducted on available documents (requirements and views) and generic requirements
SwCI (1 and 2) LOR Task: SW Design Hazard Analysis

- SW Design Analysis
  - **SW Design** - The characteristics of a system or CSCI that are selected by the developer in response to the requirements. Some will match the requirements; others will be elaborations of requirements, such as definitions of all error messages; others will be implementation related, such as decisions, about what software units and logic to use to satisfy the requirements. [Reference AOP-52]

- Conducting SW Design Hazard Analysis:
  - Identify/Define allocation of System Functions to SW Design
  - Identify the correlating SW Interface Design Requirements
  - Review design against software SSRs and HTR software mitigations to determine which cannot be supported by the current design
  - Identify new design hazards and define supported mitigations
• Software architecture defines the design constraints so it will be available for use in detailed design
• Analysis of architecture can detect hazards early, when they can be economically mitigated
• Software architecture drives software design, but actual design may exceed architectural intent or fall short
  ➢ Shortfalls with safety impact require risk assessment and (likely) new mitigations
• Safety analysis approach [architecture and design] similar but at different levels of abstraction
  ➢ SSRs are evaluated on 2-pass approach, architectural then design
E.8.5 Data Transfer Messages
Data transfer messages shall be of a predetermined format and content. Each transfer shall contain a word or character string indicating the message length (if variable), the type of data, and the content of the message. At a minimum, parity checks and checksums shall be used for verification of correct data transfer. CRCs shall be used where practical. *No information from data transfer messages shall be used prior to verification of correct data transfer.*

E.3.13 Positive Feedback Mechanisms
Software control of critical functions *shall have feedback mechanisms* that give positive indications of the function’s occurrence.

- Use each generic requirement to assess the architectural intent
  - If generic requirements are not supported in architecture, may represent risk
  - If generic requirements are supported in the architecture, record as planned mitigations for hazards
JSSSEH Generic Requirements

- **E.8.5 Data Transfer Messages**
  - Data transfer messages shall be of a predetermined format and content. Each transfer shall contain a word or character string indicating the message length (if variable), the type of data, and the content of the message. At a minimum, *parity checks* and *checksums* shall be used for verification of correct data transfer. *CRCs* shall be used where practical. No information from data transfer messages shall be used prior to verification of correct data transfer.

- **E.3.13 Positive Feedback Mechanisms**
  - Software control of *critical functions* shall have feedback mechanisms that give positive indications of the function’s occurrence.

- Use each generic requirement to assess the architectural intent
  - If generic requirements *are not* supported in architecture, may represent risk
  - If generic requirements *are* supported in the architecture, record as planned mitigations for hazards

- Use each generic requirement to confirm the design (when design is available) implements as a mitigation
Instructions for Exercise

1. Identify causal factors using SW generic requirements at the architecture level
   - Review JSSSEH Generic Requirements (*E.8.5, E.3.13*)
   - Review architectural data against current hazards and functional hazard analysis
   - Use the SSF-6 Interface and Function Architecture Diagram to identify and record causal factors
### SW Causal Factors
Identified by Architectural Analysis

**STUDENT HANDOUT - ARCHITECTURE**
Populate Causal Factor Description Column

<table>
<thead>
<tr>
<th>Mishap</th>
<th>Hazard</th>
<th>Causal Factor Description Based on JSSSEH Violations</th>
<th>Analysis Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>Engage Friendly / non-hostile target</td>
<td>Fire command processed by the wrong gun results in accidental firing by the (wrong) gun</td>
<td></td>
<td>SSF-6 Interface and Function Architecture Diagram</td>
</tr>
</tbody>
</table>

VIPPS is a Mock System used for training purposes only.
**E.8.5 Data Transfer Messages**

Data transfer messages shall be of a predetermined format and content. Each transfer shall contain a word or character string indicating the message length (if variable), the type of data, and the content of the message. At a minimum, parity checks and checksums shall be used for verification of correct data transfer. CRCs shall be used where practical. *No information from data transfer messages shall be used prior to verification of correct data transfer.*

**E.3.13 Positive Feedback Mechanisms**

Software control of critical functions *shall have feedback mechanisms* that give positive indications of the function’s occurrence.

---

**Exercise #1**

Analyze architecture to ensure it supports verification of safety data.

Analyze architecture then design to ensure they support positive feedback for safety functions.
5 Minute Exercise

Task 1: Architectural Analysis
SW Causal Factors
Identified by Architectural Analysis

SSF-6 Interface and Function Architecture Diagram

Gun Subsystem

- Could violate E.8.5
- Validate command is in sequence
- Validate command is intended for ADRS Gun
- One way violates E.3.13
- Validate Msg
- Redundant No Fire Zone Controller

C2 Subsystem

- Receive EF commands
- Process EF commands
- Process EF_ID

VIPPS is a Mock System used for training purposes only
## SW Causal Factors Identified by Architectural Analysis

<table>
<thead>
<tr>
<th>Mishap</th>
<th>Hazard</th>
<th>Causal Factor Description</th>
<th>Analysis Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>Engage Friendly / non-hostile</td>
<td>Fire command processed by the wrong gun results in accidental firing by the (wrong) gun</td>
<td>1) Likely violates JSSSEH Generic Requirement E.8.5, because information from the data transfer message is available to EF_ID Processing for use prior to verification of correct data transfer in Message Validation</td>
<td>SSF-6 Interface and Function Architecture Diagram</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2) One way interface does not allow JSSSEH Generic Requirement E.3.13 for positive feedback mechanism, “Ack” message may be paired with wrong EF Command</td>
<td></td>
</tr>
</tbody>
</table>

VIPPS is a Mock System used for training purposes only
1. Identify causal factors using SW generic requirements at the architecture level
   - Review AOP-52 definitions and JSSSEH Generic Requirements (E.8.5, E.3.13)
   - Review architectural data against current hazards and functional hazard analysis
   - Use the SSF-6 Interface and Function Architecture Diagram to identify and record causal factors

2. Identify causal factors using previous architecture hazard analysis at the design level
   - Compare SSF-6 Control Flow Analysis of Interrupt Design and Principal for Safety (PFS) approved Control Flow Analysis of Interrupt (Concept of Execution) Architecture Diagram
   - Review design data against current hazards and previous architectural analysis and determine if additional risk is created and record causal factors identified
Software Interrupts

• **in·ter·rupt**  *n. Computer Science*
  ➢ 1. A signal to a computer that stops the execution of a running program so that another action can be performed.
  ➢ 2. A circuit that conveys a signal stopping the execution of a running program.

• **interrupt**
  ➢ *n* (Electronics and Computer Science / Computer Science) the signal to initiate the stopping of the running of one computer program in order to run another, after which the running of the original program is usually continued [from Latin *interrumpere*, from INTER- + rumpere to break]
**SW Causal Factors Identified by Design Analysis**

**STUDENT HANDOUT - DESIGN ANALYSIS**

Populate Causal Factor Description

<table>
<thead>
<tr>
<th>Mishap</th>
<th>Hazard</th>
<th>Causal Factor Description Based on Design Evaluation</th>
<th>Analysis Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>Engage Friendly / non-hostile target</td>
<td>Fire command processed by the wrong gun results in accidental firing by the (wrong) gun</td>
<td></td>
<td>Comparison of SSF-6 Control Flow Analysis of Interrupt Design and SSF-6 Control Flow Analysis of Interrupt Architecture</td>
</tr>
</tbody>
</table>

VIPPSS is a Mock System used for training purposes only

NSWCDD-PN-14-00295 is approved for Distribution Statement A: Approved for Public Release; distribution is unlimited.
VIPPS PFS Approved Control Flow Analysis of Interrupt Architecture

Concept of Execution

- Interrupt is valid

Interrupt allowed in current code segment?  N

- Store Safety Significant Data
- Perform Interrupt Processing
- Restore Safety Significant Data

- Stop main program task execution, save new messages as they come in
- Pause main program processing
- Resume main program processing

- Process saved messages in the order received

VIPPS is a Mock System used for training purposes only
5 Minute Exercise

Task 2: Design Analysis
Interrupt Processing

- Step through both slides and confirm the PFS approved steps are conducted

**Concept of Execution**

- Stop main program task execution, save new messages as they come in
- Store Safety Significant Data
- Perform Interrupt Processing
- Resume main program processing
- Restore Safety Significant Data

- Interrupt allowed in current code segment?
  - Interrupt is valid
  - Missing step

- Process saved messages in the order received

- Store SS Data Items: Deactivate_CMD_flag, GunID, EFmessage
- Perform Interrupt Processing
- Resume main program processing
- Restore Safety Significant Data

- Interrupt allowed in current code segment?
  - Interrupt is valid
  - Missing data

- Stop main program task execution
- Pause main program processing
- Resume main program Processing
- Resume main program execution

**VIPPS is a Mock System used for training purposes only**
<table>
<thead>
<tr>
<th>Mishap</th>
<th>Hazard</th>
<th>Causal Factor Description Based on Design Evaluation</th>
<th>Analysis Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>Engage Friendly / non-hostile target</td>
<td>Fire command processed by the wrong gun results in accidental firing by the (wrong) gun</td>
<td>1) Interrupt processing does not restore safety data before resuming execution</td>
<td>Comparison of SSF-6 Control Flow Analysis of Interrupt Design and SSF-6 Control Flow Analysis of Interrupt Architecture</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2) Interrupt processing does not save incoming messages and process them in the order received after main program resumed - lost safety messages</td>
<td>Comparison of SSF-6 Control Flow Analysis of Interrupt Design and SSF-6 Control Flow Analysis of Interrupt Architecture</td>
</tr>
</tbody>
</table>
Outline

1. Course Overview

2. Mock System Overview - Very Important Person Protection System (VIPPS)

3. Software Safety Process
   1. Perform Software Criticality Index and Level of Rigor Assessment
   2. Perform Architecture and Design Analysis
   3. *Perform Code Analysis (No coding experience necessary)*

4. Assess Technology Insertion Proposal

5. Software (SW) Safety Analysis Summary

6. Conclusion
BREAK
Outline

1. Course Overview

2. Mock System Overview - Very Important Person Protection System (VIPPS)

3. Software Safety Process
   1. Perform Software Criticality Index and Level of Rigor Assessment
   2. Perform Architecture and Design Analysis
   3. Perform Code Analysis *(No coding experience necessary)*

4. Assess Technology Insertion Proposal

5. Software (SW) Safety Analysis Summary

6. Conclusion
Software Safety Analysis and Verification Process

Top-Level Process

Start → System Definition and Software Safety Planning → Determine Software Criticality Index (SwCI) → Software Requirements Hazard Analysis (SwCI 1–3) → Software Architectural Hazard Analysis (SwCI 1–3)

Software Design Hazard Analysis (SwCI 1–2) → Code Level Hazard Analysis (SwCI 1) → Operator Documentation Safety Review → Software Testing and Verification (SwCI 1-4) → Formal Review → Fleet Release

Sub-Process

Defect Resolution → Regression Testing → Fleet Anomaly Reporting

NSWCDD-PN-14-00295 is approved for Distribution Statement A: Approved for Public Release; distribution is unlimited.
SwCI (1) LOR Task: SW Code Level Hazard Analysis

• Code Analysis
  - **SW Code** - is a set of instructions intended for execution by a computing system. Code-level safety analysis is done for code that implements safety-critical functionality; focuses on how the safety-significant functions have been implemented in the SW

• Conducting SW Code Level Hazard Analysis:
  - After identifying system functions to a particular area of the SW Design and using the Software Requirements Specification (SRS) and Interface Design Requirements, analyze SW code to ensure compliance with safety requirements
  - Identify new hazards and mitigations
  - Recommend SW Safety Test Cases and/or provide updated safety input to test planning and procedures
• Code analysis is also performed to find software coding errors, including but not limited to:

  - Fault detections .............................................. [Out of bounds, so what?]
  - Race conditions .............................................. [Fire gun or point first?]
  - Logic errors ................................................... [Incorrect solutions]
  - Incorrect algorithms e.g., Division by zero .... [Illogical may crash program]
  - Infinite loops .................................................. [Stops all other processing]
  - Unused code .................................................... [What if it executes?]
  - Memory leaks .................................................... [Slow to no available resources]
  - Inadequate error handling ................................. [e.g., when others null]
  - Undeveloped segment paths ......................... [Fail to implement critical logic]
  - Inaccurate use of variables ......................... [Local, global, data type, etc.]
A Sample Format of Source Code

<table>
<thead>
<tr>
<th>Documentations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pre-process or statements e.g., # include &lt;stdio.h&gt;</td>
</tr>
<tr>
<td>Global Declarations</td>
</tr>
<tr>
<td>Main ()</td>
</tr>
<tr>
<td>{</td>
</tr>
<tr>
<td>Local Declarations</td>
</tr>
<tr>
<td>Program Statements</td>
</tr>
<tr>
<td>…</td>
</tr>
<tr>
<td>Calling User defined functions (option to user)</td>
</tr>
<tr>
<td>}</td>
</tr>
<tr>
<td>User defined Function 1</td>
</tr>
<tr>
<td>{</td>
</tr>
<tr>
<td>Local Declarations</td>
</tr>
<tr>
<td>Function 1 Statements</td>
</tr>
<tr>
<td>}</td>
</tr>
<tr>
<td>User defined Function 2</td>
</tr>
<tr>
<td>{</td>
</tr>
<tr>
<td>Local Declarations</td>
</tr>
<tr>
<td>Function 2 Statements</td>
</tr>
<tr>
<td>}</td>
</tr>
</tbody>
</table>

Body of Main Function

Block of code for a user defined Function

Block of code for a user defined Function
Global Variable Declarations

Global Variable - It is declared outside of a function and can be accessed everywhere in the code

Documentations

Pre-process or statements e.g., # include <stdio.h>

Global Declarations

Main ()
{
    Local Declarations
    Program Statements
    ...
    Calling User defined functions (option to user)
}

User defined Function 1
{
    Local Declarations
    Function 1 Statements
}

User defined Function 2
{
    Local Declarations
    Function 2 Statements
}

Body of Main Function

Block of code for a user defined Function

Block of code for a user defined Function
A Sample Format of Source Code

<table>
<thead>
<tr>
<th>Global Variable Declarations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pre-process or statements e.g., # include &lt;stdio.h&gt;</td>
</tr>
<tr>
<td>Global Declarations</td>
</tr>
</tbody>
</table>

**Main**

```c
Main ()
{
    Local Declarations
    Program Statements
    ...
    Calling User defined functions (option to user)
}
```

**Body of Main Function**

**Block of code for a user defined Function**

**Local Variable - It is declared within a function and is not accessible to code outside that function**

**Local Variable Declarations**

**Global Variable Declarations**

**User defined Function 1**

```c
User defined Function 1
{
    Local Declarations
    Function 1 Statements
}
```

**Block of code for a user defined Function**

**User defined Function 2**

```c
User defined Function 2
{
    Local Declarations
    Function 2 Statements
}
```
A Sample Format of Source Code

- **Global Variable Declarations**
- **Local Variable Declarations**

**Pre-process or statements e.g., # include <stdio.h>**

**Global Declarations**

**Main ()**

{  
Local Declarations  
Program Statements  
...  
Calling User defined functions (option to user)  
}

**User defined Function 1**

{  
Local Declarations  
Function 1 Statements  
}

**User defined Function 2**

{  
Local Declarations  
Function 2 Statements  
}

**Function**: Collection of SW code that perform a number of pre-defined commands to accomplish something

**Function names typically describe the function**

Body of Main Function

Block of code for a user defined Function

Block of code for a user defined Function
A Sample Format of Source Code

Code Structured Like

- Documentations
- Pre-process or statements i.e. # include <stdio.h>
- Global Declarations

Main()
{
    Local Declarations
    Program Statements
    ...Calling User defined functions (option to user)
}

User defined Function 1
{
    Local Declarations
    Function 1 Statements
}

User defined Function 2
{
    Local Declarations
    Function 1 Statements
}

But Looks Like

```c
// The following is the processEF_ID Function which is called by main() Function to initiate a CRC check and EF message
// verification after an EF command is received from C2

// Start processEF_IDFunction

// declare local variables
Boolean CRC_pass;

Call CRCFunction;

If CRC_pass is equal to true, then
    Send an ID02 back to C2,
    Call validateMsg Function,
Else
    Send an ID04 back to C2;
// EndIf
```

VIPPS is a Mock System used for training purposes only
# Software Safety Code Analysis

<table>
<thead>
<tr>
<th>SwCl</th>
<th>Level of Rigor</th>
</tr>
</thead>
<tbody>
<tr>
<td>SwCl 1</td>
<td>Program shall perform analysis of requirements, architecture, design, and code; and conduct in-depth safety-specific testing.</td>
</tr>
<tr>
<td>SwCl 2</td>
<td>Program shall perform analysis of requirements, architecture, and design; and conduct in-depth safety-specific testing.</td>
</tr>
<tr>
<td>SwCl 3</td>
<td>Program shall perform analysis of requirements and architecture, and conduct in-depth safety-specific testing.</td>
</tr>
<tr>
<td>SwCl 4</td>
<td>Program shall conduct safety-specific testing.</td>
</tr>
<tr>
<td>SwCl 5</td>
<td>Once assessed by safety engineering as Not Safety, then no safety specific analysis or verification is required.</td>
</tr>
</tbody>
</table>
Overview of VIPPS
Safety Significant Functions

SSF-1: System Power On / Initialize/ Monitor Config

SSF-2: State Transitions

SSF-3: Exclusion Zones

SSF-4: Operator Situational Awareness

SSF-5: Threat Database

SSF-6: Verification of Gun Activation Deactivation and Movement Commands

SSF-7: Calibration Verifications

SSF-8: Track ID

SSF-9: Track Correlation

SSF-10: Sensor Activation, Deactivation, and Movement

SSF-11: Threat Assessment

SSF-12: Hailer Calibration

SSF-13: Hailer Arm/Disarm

SSF-14: Hailer Activation

SSF-15: High Intensity Light Calibration

SSF-16: High Intensity Light Arm / Disarm

SSF-17: High Intensity Light Activation, Deactivation, and Movement

SSF-18: Gun Calibration

SSF-19: Gun Arm/Disarm

SSF-20: Gun Activate, Deactivate, and Movement

VIPPS is a Mock System used for training purposes only
A variety of techniques exist for code analysis, including but not limited to:

- **Process Flow Diagram (PFD)**
  - Diagram used to depict process flow in a step-by-step logical sequence
- **Data Structure Analysis**
- **Data Flow Analysis**
- **Forward Flow**
  - Forward trace of the flow of execution of the software in response to the occurrence of an event
    - E.g., Process gun fire command
- **Backward Flow**
  - Backward trace from unwanted hazardous events to see if software can cause an event to happen under credible circumstances
    - E.g., Inadvertent gun firing

All performed to verify design against requirements and to identify causal factors that could lead to hazards and mishaps.
VIPPS is a Mock System used for training purposes only

EFs from C2

Main Gun Function

Process EF_ID Function

CRC Function

Verify GunID Function

GetGunID frmMsg Function

Verify CRC Pass?

Verify integrity of data in message

Validate Msg Function

Does ID Match?

Yes

No

Is it EF 21?

Yes

No

Is it EF 23?

Yes

No

Is it EF 37?

Yes

No

Verify command appropriate (e.g., in sequence)

Select Gun Function

Deactivate Gun Function

Move Gun Function

Illegal Message

A

B

C

Verify command for the correct Gun

ID04 To C2

NSWCDD-PN-14-00295 is approved for Distribution Statement A: Approved for Public Release; distribution is unlimited.
Each variable (local or global) must be declared as a valid data type, assigned a name, and should be initialized to a starting value

```
int sum = 0;
```

Variable name: sum

Data type: integer

Initialized value: 0
Common Data Types

- **Character (char):**
  - This is a single character, like X, £, 4, or *

- **String:**
  - This is a “string” of characters of any length

- **Integer (int):**
  - A whole number – whole meaning there are no digits after a decimal point. So 65 would be a valid integer; 65.78 would not.

- **Floating-point number (float):**
  - A number that may have digits after the decimal place. 65.00 is technically a floating point number, even though it could be represented just as easily as an integer as 65. It takes more memory to store a float, which is why there is a distinction instead of just creating a “number” datatype.

- **Boolean (bool):**
  - A variable to represent true or false (or it could also mean 0 or 1)
The purpose of data structure analysis is to verify how the data items are defined and that the definition is used consistently throughout the code. The best way to analyze data is to construct a table consisting of all of the data items utilized.

### Example

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Data Type</th>
<th>Global/Local</th>
<th>Where is Variable used/referenced</th>
<th>Variable used/referenced consistently? If not, explain and provide safety implications</th>
</tr>
</thead>
<tbody>
<tr>
<td>GunID</td>
<td></td>
<td></td>
<td>1. mainGun Function 2. verifyGunID Function</td>
<td></td>
</tr>
</tbody>
</table>
Data Structure Analysis - GunID

**Declare Global Variables** – a global variable is a variable declared outside all functions.

```cpp
Integer GunID;
```

//This is an integer type variable; used to store the value of Gun ID

**MainGun Function** – where GCC program starts execution.

```
//Start the mainGun Function

Get the value of the gun address from Config File and store it in the GunID variable; //Gun address is equal to 2 in the Configuration File

//VerifyGunID Function – is called by validateMsg Function to check the value of Gun ID specified in the received EF match with the value of Gun ID specified in the config file.

//Start verifyGunID Function

//Declare Local variable

Integer gunIDfrmMsg;
```

//A local variable is one declared within the body of a function.

//An integer type variable used to store the value of gun address from the received EF

Call getGunIDfrmMsgFunction and store the return value into gunIDfrmMsg variable;

If gunIDfrmMsg is equal to GunID, then

**EXAMPLE**

VIPPS is a Mock System used for training purposes only

**************************************************************************

//verifyGunID Function – is called by validateMsg Function to check the value of Gun ID specified in the received EF match with the value of Gun ID specified in the config file.

**************************************************************************

Call getGunIDfrmMsgFunction and store the return value into gunIDfrmMsg variable;

If gunIDfrmMsg is equal to GunID, then
## Data Structure Analysis - GunID

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Data Type</th>
<th>Global/Local</th>
<th>Where is Variable used/referenced</th>
<th>Variable used/referenced consistently? If not, explain and provide safety implications</th>
</tr>
</thead>
<tbody>
<tr>
<td>GunID</td>
<td>Integer</td>
<td>Global</td>
<td>1. mainGun Function 2. verifyGunID Function</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**EXAMPLE**
Task 1: Data Structure Analysis - For the given safety significant data items define the data types and usages using highlighted yellow code. Define any identified issues.
## Data Structure Analysis Exercise

### STUDENT HANDOUT - Populate Table for the given safety significant data items using highlighted yellow code

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Data Type</th>
<th>Global/Local</th>
<th>Where is Variable used/referenced</th>
<th>Variable used/referenced consistently? If not, explain and provide any safety implications</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRC_pass</td>
<td></td>
<td></td>
<td>1. processEF_ID Function 2. CRC_Function</td>
<td></td>
</tr>
<tr>
<td>1. currentAZ</td>
<td>1.</td>
<td>1.</td>
<td>1. mainGun Function</td>
<td></td>
</tr>
<tr>
<td>2. defaultAZ</td>
<td>2.</td>
<td>2.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

VIPPS is a Mock System used for training purposes only

NSWCDD-PN-14-00295 is approved for Distribution Statement A: Approved for Public Release; distribution is unlimited.
10 Minute Exercise

Task 1: Data Structure
processEF_ID Function

*******************************************************************************
***************************
//processEF_ID Function – is called by mainGun Function to initiate a CRC check and EF message verification after an
EF command is received from C2
*******************************************************************************
//declare Local variables
Boolean CRC_pass,

Call CRC Function and store the return value in CRC_pass;

If CRC_pass is equal to true, then
    Send an ID02 back to C2,
    Call validateMsgFunction,
Else
    Send an ID04 back to C2;
//EndIf
//End of processEF_ID Function
//CRC Function – is called by processEF_ID Function to check the received EF command is a valid EF message or not.

//declare Local variables
Boolean CRC_pass,

Integer calculated_CRC, saved_CRC,

Get the values from the fields in the EFmessage to perform CRC calculation;

Calculate the CRC based on those values and store the result in calculate_CRC;
Get the pre-calculated CRC value from EFmessage and store it in the saved_CRC variable;

If calculate_CRC is equal to saved_CRC, then

Call verifyGunIDFunction and store the return value in verifyGunID;

If verifyGunID is equal to true, then

Set CRC_pass to true;
//EndIf

Else

Set CRC_pass to false;
//EndIf

return the CRC_pass value to the called function;
//End of CRC Function
## Data Structure Analysis Exercise Answers

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Data Type</th>
<th>Global/Local</th>
<th>Where is Variable used/referenced</th>
<th>Variable used/referenced consistently? If not, explain and provide any safety implications</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRC_pass</td>
<td>Boolean</td>
<td>Local</td>
<td>1. processEF_ID Function 2. CRC Function</td>
<td>Yes</td>
</tr>
<tr>
<td>1. currentAZ</td>
<td></td>
<td>1.</td>
<td>1. mainGun Function</td>
<td></td>
</tr>
<tr>
<td>2. defaultAZ</td>
<td></td>
<td>2.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

VIPPS is a Mock System used for training purposes only
mainGun Function

//declare Global Variables – a global variable is a variable declared outside all functions.

Long EFmessage,
Integer GunID,
Float defaultAZ,
Integer currentAZ,
Integer defaultEL, currentEL,
Integer defaultGunSpeed, currentGunSpeed,
Boolean de_activate_CMD_flag,
Boolean verifyGunID,

//mainGun Function – where GCC program starts execution.

Get the value of the gun address from Config File and store it in the GunID variable; //Gun address is equal to 2 in the Configuration File
Get the value of the AZ position from Config File and store it in the defaultAZ variable;
Set the value of the currentAZ variable equal to the value of the defaultAZ variable;
//The value after decimal point will be truncated if a float
//number is going to be saved in an integer type variable.

. .
Call processEF_IDFunction;

//End of mainGun Function
## Data Structure Analysis Exercise Answers

| Variable Name | Data Type | Global/Local | Where is Variable used/referenced | Variable used/referenced consistently?  
|---------------|-----------|--------------|-----------------------------------|-----------------------------------------|
| CRC_pass      | Boolean   | Local        | 1. processEF_ID Function  
|               |           |              | 2. CRC Function                   | Yes                                     |
| 1. currentAZ  | 1. Integer| 1. Global    | 1. mainGun Function               | No, the data type of the defaultAZ and currentAZ variables are different. The value after the decimal point is truncated when a float number is saved in an integer variable. In this case the gun would lose azimuth precision and point in an unintended area leading to personnel injury, equipment or environmental damage |
| 2. defaultAZ  | 2. Float  | 2. Global    |                                   |                                         |

VIPPS is a Mock System used for training purposes only

NSWCDD-PN-14-00295 is approved for Distribution Statement A: Approved for Public Release; distribution is unlimited.
Outline

1. Course Overview

2. Mock System Overview - Very Important Person Protection System (VIPPS)

3. Software Safety Process
   1. Perform Software Criticality Index and Level of Rigor Assessment
   2. Perform Architecture and Design Analysis
   3. Perform Code Analysis (No coding experience necessary)
      - Data Structure Analysis
      - Data Flow Analysis
      - AOP-52 Compliance
      - Software Safety Analysis and Verification Process Flow

4. Assess Technology Insertion Proposal

5. Software (SW) Safety Analysis Summary

6. Conclusion
BREAK
Outline

1. Course Overview

2. Mock System Overview - Very Important Person Protection System (VIPPS)

3. Software Safety Process
   1. Perform Software Criticality Index and Level of Rigor Assessment
   2. Perform Architecture and Design Analysis
   3. Perform Code Analysis (No coding experience necessary)
      - Data Structure Analysis
      - Data Flow Analysis
      - AOP-52 Compliance
      - Software Safety Analysis and Verification Process Flow

4. Assess Technology Insertion Proposal

5. Software (SW) Safety Analysis Summary

6. Conclusion
Data Flow Analysis

• The purpose of data flow analysis is to identify errors in the use of data that is accessed by multiple routines

• The following are some examples of errors that can be found via data flow analysis:
  ➢ Data which is utilized by a system prior to being initialized
  ➢ Unused data items
  ➢ Unintended data item modification
  ➢ Failure to accurately update or modify data items
Data Flow Analysis Exercise

- Task 2: Data Flow Analysis - Conduct data flow analysis to identify errors in the use of data that is accessed by multiple routines
  - Find “GunID” in each of the functions listed
  - Document the value of the data item from each function
  - After all values captured, determine if a safety concern exists and document rationale
### Data Flow Analysis Exercise

**STUDENT HANDOUT - Populate Table for GunID using highlighted yellow code**

<table>
<thead>
<tr>
<th>Data Item</th>
<th>Data type</th>
<th>Global/Local</th>
<th>Where is Variable used/referenced</th>
<th>Value stored in Variable</th>
<th>Any safety concerns? [If yes, explain]</th>
</tr>
</thead>
<tbody>
<tr>
<td>GunID</td>
<td>Integer</td>
<td>Global</td>
<td>1. declare Global variable</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2. mainGun Function</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>3. verifyGunID Function</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

VIPPS is a Mock System used for training purposes only.
5 Minute Exercise

Task 2: Data Flow
//declare Global Variables – a global variable is a variable declared outside all functions.

Long EFmessage, //This is a 64-bit variable; used to store two 32-bit word EF data.
Integer GunID, //This is an integer type variable; used to store the value of Gun ID
Float defaultAZ, //This is a float type variable; used to store the default azimuth position.
Integer currentAZ, //This is an integer type variable used to store the current azimuth position
Integer defaultEL, currentEL, //Both are integer type variables; used to store the elevation position
Integer defaultGunSpeed, currentGunSpeed, //Both are integer type variables; used to store the gun speed
Boolean de_activate_CMD_flag, //This is a Boolean type variable which has a either True (1) or False (0) value; used to indicate whether or
//not the deactivate command has been received
Boolean verifyGunID,

//mainGun Function – where GCC program starts execution.

//Start the mainGun Function

Get the value of the gun address from Config File and store it in the GunID variable; //Gun address is equal to 2 in the Configuration File
Get the value of the AZ position from Config File and store it in the defaultAZ variable;
Set the value of the currentAZ variable equal to the value of the defaultAZ variable;

//End of mainGun Function
**verifyGunID Function**

//verifyGunID Function – is called to check the value of Gun ID specified in the received EF match with the value of Gun ID specified in the config file.

//Start verifyGunID Function

//declare Local variable

Integer gunIDfrmMsg,

Boolean verifyGunID_flag,

Set GunID equal to 0;

Call Call getGunIDfrmMsgFunction and store the return value into gunIDfrmMsg variable;

If gunIDfrmMsg is equal to GunID, then

Set verifyGunID_flag to True,

Call FireGunFunction,

Call validateMsgFunction;

Else

Set verifyGunID_flag to False;

//EndIf

return the verifyGunID_flag value to the called function;

//End of verifyGunID Function
### Data Flow Analysis Exercise Answers

<table>
<thead>
<tr>
<th>Data Item</th>
<th>Data type</th>
<th>Global/Local</th>
<th>Where is Variable used/referenced</th>
<th>Value stored in Variable</th>
<th>Any safety concerns? [If yes, explain]</th>
</tr>
</thead>
<tbody>
<tr>
<td>GunID</td>
<td>Integer</td>
<td>Global</td>
<td>1. declare Global variable</td>
<td>Variable no initialized at declaration</td>
<td>Modify the value of GunID could lead to unintentionally activate wrong gun.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2. mainGun Function</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>3. verifyGunID Function</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

VIPPS is a Mock System used for training purposes only.
The purpose of conducting a compliance assessment is to ensure the code follows a set of coding standards. Non-compliance could result in errors that could lead to potential safety impact.

Compliance requirements can come from a multitude of sources, with AOP-52 and JSSSEH being two of them.

Three generic requirements from AOP-52 will be the discussed in this last exercise.
### Definition of Terms

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flags and Variables</td>
<td>Flags and variable names shall be unique. Flags and variables shall have a single purpose and shall be defined and initialized prior to use.</td>
</tr>
<tr>
<td>Execution Path</td>
<td>Safety Critical Computing System Functions (SCCSFs) shall have one and only one possible path leading to their execution.</td>
</tr>
<tr>
<td>Conditional Statements</td>
<td>Conditional statements shall have all possible conditions satisfied and be under full software control (i.e., there shall be no potential unresolved input to the conditional statement). Conditional statements shall be analyzed to ensure that the conditions are reasonable for the task and that all potential conditions are satisfied and not left to a default condition. All condition statements shall be annotated with their purpose and expected outcome for given conditions.</td>
</tr>
</tbody>
</table>
### AOP-52 Compliance Example

<table>
<thead>
<tr>
<th>AOP-52 Requirement</th>
<th>Compliant</th>
<th>Function where code error is located</th>
<th>Rationale for non-compliance</th>
<th>Potential Safety Impact(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conditional Statements</td>
<td>No</td>
<td>validateMsgFunction</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**If** `current_EF` is equal to 11 then
Call `selectGunFunction`;  
//Check the received EF command is equal to EF11
Else
  **If** `current_EF` is not equal to 21 then
  Call `selectGunFunction`;  
  //Check the received EF command is equal to EF21-activate gun
Else
  **If** `current_EF` is equal to 23 then
  Call `deselectGunFunction`;  
  //Check the received EF command is equal to EF23-deselect gun
Else
  **If** `current_EF` is equal to 37 then
  Call `moveGunFunction`;  
  //Check the received EF command is equal to EF37-move gun

VIPPS is a Mock System used for training purposes only
If current_EF is equal to 11 then
   Call selectGunFunction;
Else
   If current_EF is not equal to 21
      then
         Call selectGunFunction;
   Else
      If current_EF is equal to 23
         then
            Call deselectGunFunction;
      Else
         If current_EF is equal to 37
            then
               Call moveGunFunction;
• Task 3: Use AOP-52 requirement, compliance assessment, and function to define compliance rationale. Using highlighted yellow code. Define any safety issues.
**AOP-52 Code Compliance**

**Task 3 Exercise**

**STUDENT HANDOUT**

<table>
<thead>
<tr>
<th>AOP-52 Requirement</th>
<th>Compliant</th>
<th>Function where code error is located</th>
<th>Rationale for non-compliance</th>
<th>Potential Safety Impact(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conditional Statements</td>
<td>No</td>
<td>validateMsgFunction</td>
<td>Select and power on the gun for ANY EF except for EF 21 - GSS Select</td>
<td>Will result in the Gun remaining active (selected and powered on) no matter what EF is received from C2, including an EF 23 - Safe Deselect (Power Off)</td>
</tr>
<tr>
<td>Flags and Variables</td>
<td>No</td>
<td>moveGunFunction</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Execution Path</td>
<td>No</td>
<td>verifyGunIDFunction</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
10 Minute Exercise

Task 3: Code Compliance
moveGun Function

DOB

************

/\moveGun Function – which is called by validateMsg Function to command the gun motor to move gun to the specified AZ and EL positions with specified speed.

//Start moveGun Function which is passing three parameter variables i.e. AZ, EL, and gun speed

//declare Local variables

Integer commandedAZ,
Integer commandedEL,
Integer commandedGunSpeed,
Boolean moveGunStatus_flag = False,

Get the value of AZ position from the parameter (defaultAZ) and store it in the commandedAZ variable;
Get the value of EL position from the parameter (defaultEL) and store it in the commandedEL variable;

If the value of the commandedAZ variable is “not” equal to the value of the currentAZ variable, then

   Call moveAZ_MotorFunction (commandedAZ, commandedGunSpeed);

//EndIf
.
.
.
//End of moveGun Function
### AOP-52 Code Compliance Exercise Answer

<table>
<thead>
<tr>
<th>AOP-52 Requirement</th>
<th>Compliant</th>
<th>Function where code error is located</th>
<th>Rationale for non-compliance</th>
<th>Potential Safety Impact(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conditional Statements</td>
<td>No</td>
<td>validateMsgFunction</td>
<td>Select and power on the gun for ANY EF except of EF 21 - GSS Select</td>
<td>Will result in the Gun remaining active (selected and powered on) no matter what EF is received from C2, including an EF 23 - Safe Deselect (Power Off)</td>
</tr>
<tr>
<td>Flags and Variables</td>
<td>No</td>
<td>moveGunFunction</td>
<td>Variable is declared but not initialized prior to use; undetermined value for the variable commandedGunSpeed</td>
<td>Undetermined speed of the gun could lead to equipment damage</td>
</tr>
<tr>
<td>Execution Path</td>
<td>No</td>
<td>verifyGunIDFunction</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**VIPPS is a Mock System used for training purposes only**

NSWCDD-PN-14-00295 is approved for Distribution Statement A: Approved for Public Release; distribution is unlimited.
**verifyGunID Function**

.InterfaceGunID – is called to check the value of Gun ID specified in the received EF match with the value of Gun ID specified in the config file.

//Start verifyGunID Function

//declare Local variable

Integer gunIDfrmMsg,

Boolean verifyGunID_flag,

Call Call getGunIDfrmMsgFunction and store the return value into gunIDfrmMsg variable;

If gunIDfrmMsg is equal to GunID, then

Set verifyGunID_flag to True,

Call FireGunFunction,

Call validateMsgFunction;

Else

Set verifyGunID_flag to False;

//EndIf

return the verifyGunID_flag value to the called function;

//End of verifyGunID Function
**AOP-52 Code Compliance Exercise Answer**

<table>
<thead>
<tr>
<th>AOP-52 Requirement</th>
<th>Compliant</th>
<th>Function where code error is located</th>
<th>Rationale for non-compliance</th>
<th>Potential Safety Impact(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conditional Statements</td>
<td>No</td>
<td>validateMsgFunction</td>
<td>Select and power on the gun for ANY EF except of EF 21 - GSS Select</td>
<td>Will result in the Gun remaining active (selected and powered on) no matter what EF is received from C2, including an EF 23 - Safe Deselect (Power Off)</td>
</tr>
<tr>
<td>Flags and Variables</td>
<td>No</td>
<td>moveGunFunction</td>
<td>Variable is declared but not initialized prior to use; undetermined value for the variable commandedGunSpeed</td>
<td>Undetermined speed of the gun could lead to equipment damage</td>
</tr>
<tr>
<td>Execution Path</td>
<td>No</td>
<td>verifyGunIDFunction</td>
<td>The FireGunFunction is called as part of the VerifyGunIDFunction, violating the “. . . One and only one execution path . . .” for SCCSFs</td>
<td>Gun would be commanded to fire as part of VerifyGunID Function even before the EF were validated, with potential catastrophic consequences.</td>
</tr>
</tbody>
</table>

VIPPS is a Mock System used for training purposes only
<table>
<thead>
<tr>
<th>Risk Levels</th>
<th>Description of Risk Criteria</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A software implementation or software design defect that upon occurring during normal or credible off-nominal operations or tests:</td>
</tr>
</tbody>
</table>
| High        | - Can lead directly to a catastrophic or critical mishap, or  
              - Places the system in a condition where no independent functioning interlocks preclude the potential occurrence of a catastrophic or critical mishap. |
| Serious     | - Can lead directly to a marginal or negligible mishap, or  
              - Places the system in a condition where only one independent functioning interlock or human action remains to preclude the potential occurrence of a catastrophic or critical hazard. |
| Medium      | - Influences a marginal or negligible mishap, reducing the system to a single point of failure, or  
              - Places the system in a condition where two independent functioning interlocks or human actions remain to preclude the potential occurrence of a catastrophic or critical hazard. |
| Low         | - Influences a catastrophic or critical mishap, but where three independent functioning interlocks or human actions remain, or  
              - Would be a causal factor for a marginal or negligible mishap, but two independent functioning interlocks or human actions remain.  
              - A software degradation of a safety critical function that is not categorized as high, serious, or medium safety risk.  
              - A requirement that, if implemented, would negatively impact safety; however code is implemented safely. |
Software Safety Analysis and Verification Process

Top-Level Process

Start

System Definition and Software Safety Planning

Determine Software Criticality Index (SwCI)

Software Requirements Hazard Analysis (SwCI 1-3)

Software Architectural Hazard Analysis (SwCI 1-3)

Software Design Hazard Analysis (SwCI 1-2)

Code Level Hazard Analysis (SwCI 1)

Operator Documentation Safety Review

Software Testing and Verification (SwCI 1-4)

Safety – Specific Testing (SwCI 4)

In-depth Safety - Specific Testing (SwCI 1-3)

Formal Review

Fleet Release

Sub-Process

Defect Resolution

Regression Testing

Fleet Anomaly Reporting

NSWCDD-PN-14-00295 is approved for Distribution Statement A: Approved for Public Release; distribution is unlimited.
Operational Documentation Review

- Review of operational document to identify potential hazards/causal factors and verify mitigations
  - Operator Manuals (consoles or units driven by software)
  - Capabilities and Limitations Documents (associated with computing systems)
  - Software Installation manuals (documents software build capability/fixes and code install/startup)
  - Ship Messages (updates to operations, usages or considerations for safety)
Software Safety Analysis and Verification Process

Top-Level Process

Start → System Definition and Software Safety Planning

Software Design Hazard Analysis (SwCI 1–2) → Code Level Hazard Analysis (SwCI 1)

Determine Software Criticality Index (SwCI) → Software Requirements Hazard Analysis (SwCI 1–3)

Software Testing and Verification (SwCI 1-4) → Formal Review

Safety – Specific Testing (SwCI 4)

In-depth Safety - Specific Testing (SwCI 1-3)

Fleet Release

Sub-Process

Defect Resolution → Regression Testing → Fleet Anomaly Reporting

Software Criticality Matrix

NSWCDD-PN-14-00295 is approved for Distribution Statement A:
Approved for Public Release; distribution is unlimited.
Software Safety Testing

- Testing should address not only performance-related SW Requirements, but the SW Safety Significant Requirements as well
- The minimum level of software safety testing depends upon the LOR performed on the associated SW SSF:
  - Two types of Software Testing - MIL-STD-882E Table V defines the LOR where In-Depth Safety-Specific Testing (LOR SwCI 1-3) and Safety-Specific Testing (LOR SwCI 4) are required
  - In all cases:
    - Safety input required to the test plan to ensure test and verification of safety significant software (i.e., Participate in or Witness Testing)
    - Safety Testing should be conducted at Unit, CSCI, (Sub-)System Level, and Test Coverage Analysis
- Leads to Verification and Validation of SW SSR implementation
  - Results recorded in SRVM for LOR SwCI 1-3 only
- Test Plans and Test Reports provide documentation for safety engineer to cite during final risk assessment
Examples of SW Safety Specific Testing (SwCl 4)

- **Endurance Testing** - Demonstrate the ability of the system to run for a defined period of time without failing (Defined in JSSSEH 4.4.2.6.)
- **User Interface Tests** - Verify the functionality of the user interface (Defined in JSSSEH 4.4.2.7.)
- **Fault Insertion and Failure Testing** - Provide assurance that the software will safely respond to various faults or failures in the hardware and software (Defined in JSSSEH 4.4.2.8.)
- **Safety injected** into: Functional Testing, Physical Testing
- **Go/No-Go Path Testing** - Verify required functionality works in the go-path scenario and with failures incurred
- **Human Integration Testing** - Ensure the operator can safety manage the equipment and workload
- **Regression Testing** - Assures modifications to the SW do not adversely affect the functionality
In addition to Safety Specific Testing, In-Depth Safety testing may include:

- **Path Coverage Testing** - Ensure every possible path in the code is executed at least once (Defined in JSSSEH 4.4.2.3.)
- **Requirements-Based Testing** - Verify software implements the high-level requirements (Defined in JSSSEH 4.4.1.2. and 4.4.2.1.)
- **Statement Coverage Testing** - Verify the “success” path of an IF statement in code is exercised (Defined in JSSSEH 4.4.2.4.)
- **Mutation Testing** - Modify code to achieve a specific testing objective
- **Perturbation Testing** - Variation of mutation testing in which test team “perturbs” the execution environment to determine the reaction of software
- Safety injected into Exception Handling, Boundary Handling, and Data Rates Testing
- **Stress Testing** - Verify the ability of the system to function under high stress conditions
- **Stability/Endurance Testing** - Demonstrate the ability of the system to run for a defined period of time without failing (Defined in JSSSEH 4.4.2.6.)
Software Safety Analysis and Verification Process

Top-Level Process

Start → System Definition and Software Safety Planning → Determine Software Criticality Index (SwCI) → Software Requirements Hazard Analysis (SwCI 1–3) → Software Architectural Hazard Analysis (SwCI 1–3)

Software Design Hazard Analysis (SwCI 1–2) → Code Level Hazard Analysis (SwCI 1) → Operate Documentation Safety Review → Software Testing and Verification (SwCI 1–4)

Safety – Specific Testing (SwCI 4)

In-depth Safety – Specific Testing (SwCI 1–3)

Formal Review → Fleet Release

Sub-Process

Defect Resolution → Regression Testing → Fleet Anomaly Reporting
SW Safety Formal Review

- Formal Review - is done to provide documented evidence that the software contribution to system risk is defined and all remaining risks are accepted
  - This review is done after the LOR tasks have been completed and the System risk is updated with SW contribution and documented within the context of a Safety Assessment Report (SAR) and Mishap Assessment Report (MAR)
  - The system risk, including SW’s contribution, is presented within a Technical Data Package for review by the appropriate Safety Authority
Software Safety Analysis and Verification Process

Top-Level Process

- Start
  - System Definition and Software Safety Planning
  - Determine Software Criticality Index (SwCI)
  - Software Requirements Hazard Analysis (SwCI 1–3)
  - Software Architectural Hazard Analysis (SwCI 1–3)

Sub-Process

- Software Design Hazard Analysis (SwCI 1–2)
- Code Level Hazard Analysis (SwCI 1)
- Operating Document Safety Review
- Software Testing and Verification (SwCI 1–4)
  - Safety – Specific Testing (SwCI 1–4)
  - In-depth Safety – Specific Testing (SwCI 1–3)

- Defect Resolution
- Regression Testing
- Fleet Anomaly Reporting

Fleet Release
Post-Fleet Release SW Safety Analysis

- **Even after the System has been released for Fleet use, SW Safety Analysis continues throughout the System’s life cycle**

- **Fleet Anomaly Reporting** - the user identifies any anomalies during its use of the system and reports it (i.e., Software Trouble Reports (STRs), reporting to the In-Service Engineering Agents). Then the anomalies are assessed to determine its impact to Safety functions and system risk.

- **Defect Resolution** - defects can be resolved within the design at any time in the system life cycle. Safety maintains involvement to ensure safety issues are resolved and implementations do not increase risk.

- **Regression Testing** - defect resolution forces regression testing to assess system and safety performance. “Regression” is considered a sub-set of previously performed testing. The safety challenge is to ensure adequate safety-specific testing is performed based on the nature of the change.
Software Safety Analysis and Verification Process

Top-Level Process

1. Start
2. System Definition and Software Safety Planning
3. Determine Software Criticality Index (SwCI)
4. Software Requirements Hazard Analysis (SwCI 1–3)
5. Software Architectural Hazard Analysis (SwCI 1–3)

Sub-Process

1. Software Design Hazard Analysis (SwCI 1–2)
2. Code Level Hazard Analysis (SwCI 1)
3. Operate Documentation Safety Review
4. Software Testing and Verification (SwCI 1-4)
   - Safety – Specific Testing (SwCI)
   - In-depth Safety – Specific Testing (SwCI 1–3)

Defect Resolution
Regression Testing
Fleet Anomaly Reporting

Fleet Release
Outline

1. Course Overview

2. Mock System Overview - Very Important Person Protection System (VIPPS)

3. Software Safety Process
   1. Perform Software Criticality Index and Level of Rigor Assessment
   2. Perform Architecture and Design Analysis
   3. Perform Code Analysis (No coding experience necessary)

4. Assess Technology Insertion Proposal

5. Software (SW) Safety Analysis Summary

6. Conclusion
BREAK
Outline

1. Course Overview
2. Mock System Overview - Very Important Person Protection System (VIPPS)
3. Software Safety Process
   1. Perform Software Criticality Index and Level of Rigor Assessment
   2. Perform Architecture and Design Analysis
   3. Perform Code Analysis (No coding experience necessary)
4. **Assess Technology Insertion Proposal**
5. Software (SW) Safety Analysis Summary
6. Conclusion
Software Safety Analysis and Verification Process

Top-Level Process

Start → System Definition and Software Safety Planning → Determine Software Criticality Index (SwCl) → Software Requirements Hazard Analysis (SwCl 1-3) → Software Architectural Hazard Analysis (SwCl 1-3) → Software Design Hazard Analysis (SwCl 1-2) → Code Level Hazard Analysis (SwCl 1) → Operate Documentation Safety Review → Software Testing and Verification (SwCl 1-4) → Formal Review → Fleet Release

Sub-Process

Defect Resolution → Regression Testing → Fleet Anomaly Reporting
Technology Insertion

- Technology insertion is the process of incorporating and exploiting new or improved technology into existing platforms, systems, and equipment.
- Software safety process verifies implementation of new technologies:
  - Identify and mitigate hazards/causal factors.
- Software technologies introduced may impact safety:
  - Interface protocols.
  - Processor architecture.
  - Operating system.
  - Coding language.
  - Compiler.
  - Other.
Technology Insertion (Cont’d)

• Technology insertion can be early in the program or later via technology refresh
• Choice of technology presents potential risk with respect to safety requirements and design
  ➢ Cost to verify/validate
  ➢ Inability to implement
  ➢ Safety and program risk associated with maturity level
• Safety Engineer must be able to identify the potential safety risk for design selection and mitigation decisions
  ➢ Conduct literature searches to identify technology strengths / weaknesses
  ➢ Consider appropriate LOR verification against the technologies
Example

- New correlation processor added to an old combat system
- New correlation processor
  - When combat system sends a correlation request on two tracks (T1 and T2)
    - Bit of 0, could not complete the request
    - Bit of 1 and with both track numbers (T1 and T2), request completed and the tracks do not correlate
    - Bit of 1 and with one track number (T1 and T2 = 0), request completed and the tracks correlate
- Combat System
  - When the combat system receives a correlation request
    - Bit of 1, the tracks correlate
    - Bit of 0, the tracks do not correlate
• Shoot-It Gun Control Computer has submitted an Engineering Change Proposal (ECP) to upgrade the gun subsystem
  ➢ Port GCC software to Java language
  ➢ Replace analog Gun Subsystem interfaces with the following:
    ◆ Data Link Layer: Bus-Based Ethernet
    ◆ Transport Layer: Transmission Control Protocol (TCP)
Concerns

- Java (Whitford, Loutzenhiser, and Mears)
  - Automatic garbage collection preempt application processing for significant amounts of time at unpredictable processing points
  - Byte-code interpretation renders execution time unpredictable
  - Priority inversion among sets of Java threads cause high priority threads to be unnecessarily delayed by lower priority threads

Recommendations

- Java (Whitford, Loutzenhisier, and Mears)
  - Utilize new Java technologies like Real Time Garbage Collection (RTGC) or Real Time Specification for Java (RTSJ) and Ravenscar Profile for Java to ensure real-time response of code
    - RTGC - supports soft real-time applications with provisions for controlling automatic garbage collection
    - RTSJ - supports hard real-time applications with provisions for periodic and aperiodic task, support for deadlines and Central Processing Unit (CPU) time budgets
    - Ravenscar - high-integrity subset of RTSJ which provides predictability of memory utilization, timing, and data flow
  - Perform LOR analyses (to include thorough analysis of the timing of the system under development)
### ECP Safety Considerations

<table>
<thead>
<tr>
<th>Technology of Concern</th>
<th>Concerns(s)</th>
<th>Recommendation(s)</th>
</tr>
</thead>
</table>
| Java language         | - Automatic garbage collection  
- Unpredictable execution time  
- Priority inversion   | - Utilize RTGC (for soft real-time) or RTSJ and Ravenscar profile (for hard real-time) to ensure real-time response of code  
- Perform LOR analyses|
Student Exercise

- Review reference material on Ethernet Protocol (Section 5.2.2.1)
- Based on your review of the technology, define:
  - Any concerns about using the technology
  - Any recommendations concerning design selections or mitigations

<table>
<thead>
<tr>
<th>Technology of Concern</th>
<th>Concern(s)</th>
<th>Recommendation(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ethernet Protocol</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Time to Complete: Ten Minutes
10 Minute Exercise

Task 1: Ethernet Protocol
Identify concerns, impact to safety analyses, and possible recommendations related to ECP technologies

<table>
<thead>
<tr>
<th>Technology of Concern</th>
<th>Concerns(s)</th>
<th>Recommendation(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ethernet Protocol</td>
<td>-Bus-based configuration may result in lost data and introduce message timing uncertainty</td>
<td>-Use switch-based Ethernet system to preclude message collisions</td>
</tr>
</tbody>
</table>
Student Exercise

- Review reference material on TCP Protocol (Section 5.2.4.1)
- Based on your review of the technology, define:
  - Any concerns about using the technology
  - Any recommendations concerning design selections or mitigations

<table>
<thead>
<tr>
<th>Technology of Concern</th>
<th>Concern(s)</th>
<th>Recommendation(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCP Protocol</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Time to Complete: Ten Minutes
10 Minute Exercise

Task 2: TCP Protocol
Identify concerns, impact to safety analyses, and possible recommendations related to ECP technologies

<table>
<thead>
<tr>
<th>Technology of Concern</th>
<th>Concerns(s)</th>
<th>Recommendation(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCP Protocol</td>
<td>-Data overhead of TCP protocol will potentially impact timeliness of messages</td>
<td>-Consider utilizing UDP transport protocol and implementing reliability checks in application layer</td>
</tr>
</tbody>
</table>
Outline

1. Course Overview

2. Mock System Overview - Very Important Person Protection System (VIPPS)

3. Software Safety Process
   1. Perform Software Criticality Index and Level of Rigor Assessment
   2. Perform Architecture and Design Analysis
   3. Perform Code Analysis (No coding experience necessary)

4. Assess Technology Insertion Proposal

5. Software (SW) Safety Analysis Summary

6. Conclusion
Software Safety Analysis and Verification Process

Top-Level Process

- Start
  - System Definition and Software Safety Planning
  - Determine Software Criticality Index (SwCI)
  - Software Requirements Hazard Analysis (SwCI 1-3)
  - Software Architectural Hazard Analysis (SwCI 1-3)
  - Software Design Hazard Analysis (SwCI 1-2)
  - Code Level Hazard Analysis (SwCI 1)
  - Operator Documentation Safety Review
  - Software Testing and Verification (SwCI 1-4)
  - Formal Review

Sub-Process

- Defect Resolution
- Regression Testing
- Fleet Anomaly Reporting

NSWCDD-PN-14-00295 is approved for Distribution Statement A: Approved for Public Release; distribution is unlimited.
Exercises

- Performed SwCI and LOR Assessment
  - Completed SW criticality analysis worksheet which included: software functions, mishap severity, related hazards, SW control category, SwCI, and the LOR tasks required
- Performed Architecture and Design Analysis
  - Used guidance from software safety standards and guidance documents
- Performed Code Analysis
  - Data structure analysis exercise
  - Data flow analysis exercise
  - AOP-52 compliance assessment
- Technology Insertion Analysis
  - Analyzed an ECP
SW Safety Summary

- SW Safety Analysis is typically conducted during the SSHA efforts
- SW can cause, influence, contribute to, or mitigate hazards
- SW Analysis results in improved designs and mitigations, and reduces the likelihood that SW will initiate a hazardous condition or mishap. [MIL-STD-882E B.2.2]
  - SW System Safety Analysis (the focus of this tutorial) increases the confidence that the software will perform as specified to software system safety and performance requirements while reducing the number of contributors to hazards that may exist in the system

- SW Safety Analysis process tasks include:
  - SW Requirements Hazard Analysis
  - SW Architectural Hazard Analysis
  - SW Design Hazard Analysis
  - Code Level Hazard Analysis
  - SW Safety Testing and Verification

- After the completion of SW Safety Analysis Tasks, assess and incorporate Software’s contribution to overall System Risk
Outline

1. Course Overview

2. Mock System Overview - Very Important Person Protection System (VIPPS)

3. Software Safety Process
   1. Perform Software Criticality Index and Level of Rigor Assessment
   2. Perform Architecture and Design Analysis
   3. Perform Code Analysis (No coding experience necessary)

4. Assess Technology Insertion Proposal

5. Software (SW) Safety Analysis Summary

6. Conclusion
Conclusion

- This concludes the Software Safety Tutorial. Thank you for your participation.
- Questions?