Gaining Early Insight into Software Safety: Measures of Potential Problems and Risks

Victor Basili
Kathleen Dangle
Linda Esker
Fraunhofer Center Maryland

Frank Marotta
U. S. Army Aberdeen Test Center

Systems & Software Technology Conference
June 2007

Contents

Topics Covered

- System Safety Introduction
- Visibility into the System Safety Process
- Software Safety Concepts
- Safety Measurement Areas and Levels
- Metrics
- Experiences and Summary

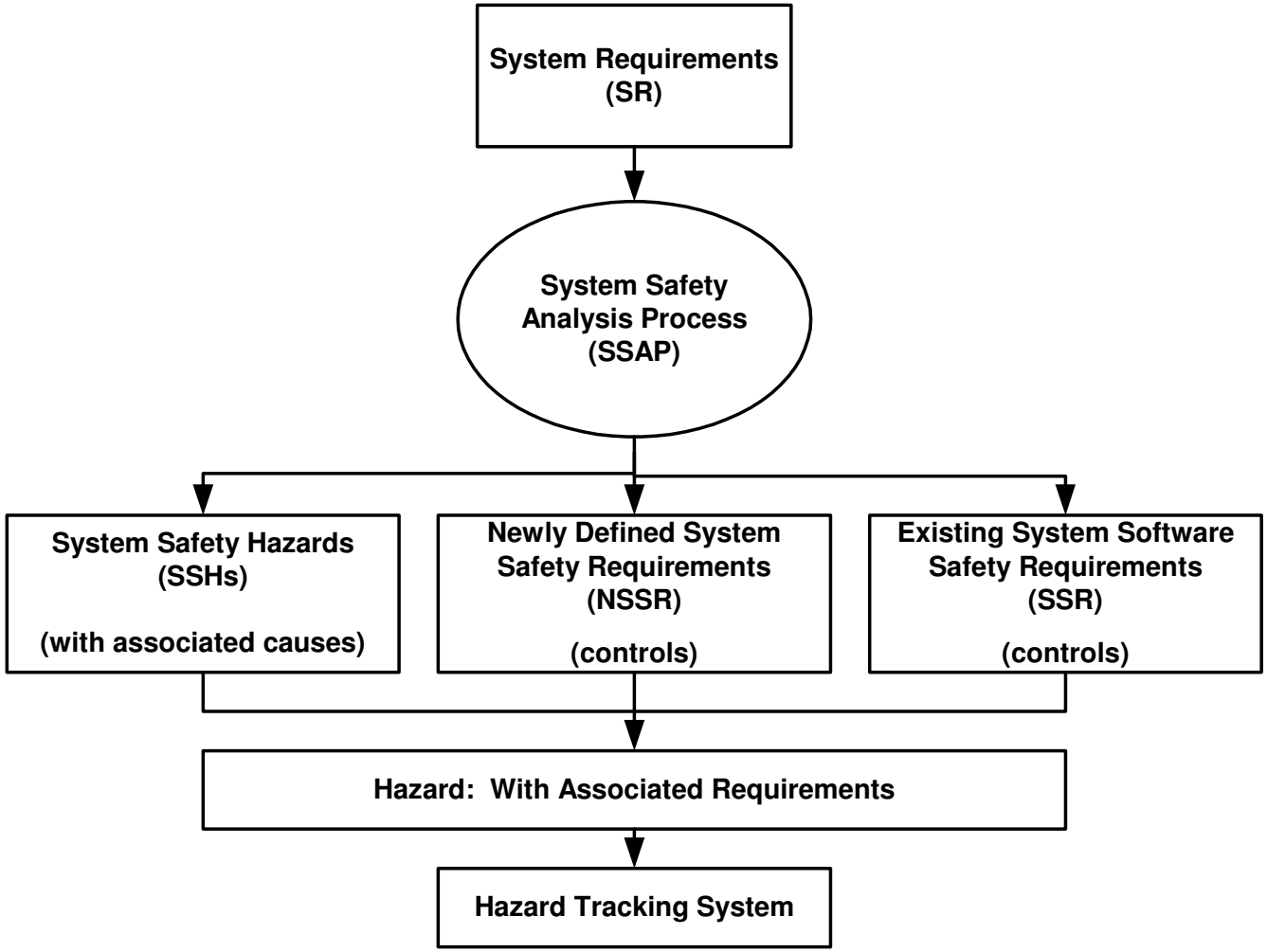
© 2007 Fraunhofer Center Maryland

System Safety

- **The purpose of System Safety is to**
  - Identify and mitigate hazards associated with the operation and maintenance of the system
  - Define the residual risk, based on success of the controls implemented
- **System Safety is implemented through an approach that**
  - Identifies hazards and assigns appropriate controls across multiple integrated platforms in a system of systems network which includes platform specific hazards and hazards related to the interaction of the various platforms
  - Establishes process checks to ensure hazard controls are not compromised at all levels of the system of system

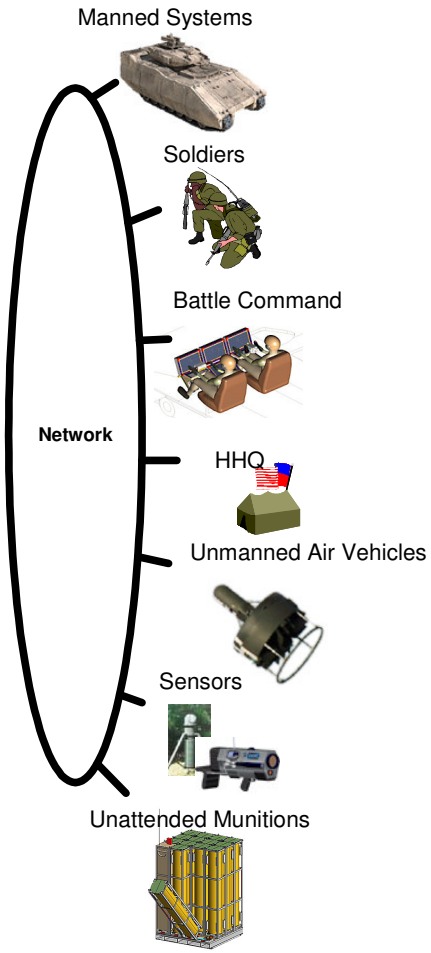
© 2007 Fraunhofer Center Maryland

Hazard Analysis



© 2007 Fraunhofer Center Maryland

Elements of a Hazard



Hazard: Check fire command fails to execute.

- **Network**
  - **Cause 1: Command routed to incorrect destination**
    - **Control/mitigation 1.1** — Verification 1
    - ⋮
    - **Control/mitigation 1.x$_1$** — Verification 1
  - **Cause X: . . .**
    - **Control/mitigation X.1** — Verification 1
    - ⋮
    - **Control/mitigation 1.x$_2$** — Verification 1
- **Manned Systems**
  - **Cause 1: . . .**
    - **Control/mitigation 1.1** — Verification 1
    - ⋮
    - **Control/mitigation 1.y$_1$** — Verification 1
  - **Cause Y: . . .**
    - **Control/mitigation Y.1** — Verification 1
    - ⋮
    - **Control/mitigation 1.y$_2$** — Verification 1
- ⋮

Fraunhofer USA, Inc
Center for Experimental Software Engineering Maryland

© 2007 Fraunhofer Center Maryland

So, What's The Problem?

- Evaluation of the safety of a system is usually done during test at the end of the system's development life cycle
  - Preventable problems not identified soon enough
- Today, many hazardous systems are
  - Bigger
  - Distributed
  - Network-connected
  - Composed of hardware and software elements
- Systems of systems provide more complex problems for safety engineers—not stand-alone and manually controlled
- A large network centric system of systems will have a **large** number of safety hazards (with multiple causes and controls) that safety engineers must track and verify before the system is deployed—a predominately manual process won't work
- Software is an ever-increasing part of the system

© 2007 Fraunhofer Center Maryland

Visibility Into System Safety

- Important to gain visibility into problems early to mitigate risks and impacts to the Program
- Cannot measure the safety of the system during its development
  - Possible to gain ***visibility*** into safety considerations, thereby reducing safety risks
- Visibility can be achieved by the identification of safety-related requirements, their traceability, the identification and quantification of safety-related defects, and the extent to which safety-related failures are detectable by software

© 2007 Fraunhofer Center Maryland

Safety Visibility Measurement Goal

- **Goal** is to develop and implement a set of metrics that provide management ***visibility*** into system (and software) safety
  - The hazard management process (identification, analysis, tracking, and resolution)
  - The progress towards achieving safety
  - Hazard controls handled by software
  - The development process of safety-related hardware and software items
- For the **purpose** of assisting management in
  - Asking the right questions
  - Identifying safety risks
  - Monitoring the quality of the safety process
  - Making decisions that mitigate risks and prevent safety issues

Metrics will not tell us whether the system is safe but they can provide indicators of potential problems and risks

© 2007 Fraunhofer Center Maryland

Software Safety Concepts

- Software Safety is a subset of System Safety
- Software Safety Hazards are a subset of system safety hazards where at least one cause is due to software or one control is handled by software
- Existing system safety processes (e.g., safety analysis processes) are leveraged in defining the software safety process
- Our work focused on software, but the measurement areas and metrics are applicable to the entire system—hardware and software

© 2007 Fraunhofer Center Maryland

Safety Measurement Areas

- We have identified five areas that provide **insight** into the implementation of the safety process for software:

  1. **Software Safety Analysis Process**
  2. **Hazard Identification and Mitigation**
  3. **Hazard Management**
  4. **Appropriate Level of Rigor for Software Safety**
  5. **Safety Defects**

- Used Goal, Question, Metric (GQM*) methodology to derive effective metrics

*GQM method originated by V.Basili and D.Weis

© 2007 Fraunhofer Center Maryland

Measurement Levels

- **High Level: Readiness Assessment**
  - For each area, ascertain the **readiness** of the data that is used in visibility measures
  - The set of readiness **questions** provides a preliminary view into the state of the safety process for software
  - These questions are asked **before** any measurement takes place and are answered "*yes*" or "*no*"
  - There are two different **intentions** for the readiness questions:
    - Determine whether basic processes are being followed, they are meant to be asked early
    - Discover whether the data is/will be available so that the deeper metrics can begin—they are meant to be asked at appropriate life-cycle phases
- **Deeper Level: Software Safety Risk Metrics**
  - For each area, define more specific measurement questions
  - Derive effective metrics to provide visibility into the inquiry area using a measurement template

© 2007 Fraunhofer Center Maryland

The Readiness Assessment — Data Readiness Questions

1. **Software Safety Analysis Process:** Safety engineers must certify that the system is in compliance with safety processes; the core to a safety analysis process is knowledge of the safety requirements and their relationship to hazards
   - Is there a documented software safety process that identifies requirements as safety-related?
   - Are requirements identified as safety related in the requirements repository (e.g., DOORS)?

2. **Hazard Identification:** Documenting the appropriate information about a hazard (i.e., causes, controls/mitigations) is critical to ensuring that the program is adequately executing the safety process
   - Is there an (automated) HTS where software-related hazards, causes, and controls are recorded (and can be counted)?

© 2007 Fraunhofer Center Maryland

The Readiness Assessment — Data Readiness Questions

3. **Hazard Management:** Safety engineers must analyze and measure hazards to ensure the safety process is being followed and sufficient actions are being taken to make the system safe
   - Are hazards mapped back to their source (requirements) and controls mapped to requirements?
   - Do all the appropriate HTS sub-fields exist?

4. **Appropriate Level of Rigor for Software Safety:** Level of rigor defines the level of response needed from the software as determined by the importance and complexity of the hazard; identifying the right level of rigor is critical to balancing the cost of safety with the risk
   - Are the various levels of rigor identified and distributed among the levels?

© 2007 Fraunhofer Center Maryland

The Readiness Assessment — Data Readiness Questions

5. **Safety Defects:** Safety engineers need to know whether all safety controls/requirements are tested and if any safety problems remain in the system for the Safety Assessment Reports (SARs)

   - Are failures/faults identified as software safety-related in the Software Problem Reporting System (SPR)?
   - Are test cases identified as safety related?
   - Are defect closures recorded?

© 2007 Fraunhofer Center Maryland

Deeper Level: Software Safety Risk Metrics

- Data for software safety risk metrics is available when initial readiness is achieved
- **Software Safety Risk Metrics** are intended to support detection and analysis of software-related safety issues in a timely manner and **make potential risks visible**
- **Data sources** that commonly exist (e.g., hazard tracking database, requirements management tools, etc.) **are leveraged** in defining these software safety metrics
- Analysis includes
  - Metrics goals, questions, measures, interpretation models, and potential responses

© 2007 Fraunhofer Center Maryland

Measurement Template

- For each metric we supply the
  - **Question** being addressed with the aid of the metric
  - The **metric** definition
  - Suggested **interpretation/responses**
- By "metric" we mean a basic measure or derived metric
- To interpret the values of the metrics we have chosen initial estimates of the bounds because we do not have historical data.
  - These bounds can evolve as we start to see the real data from the particular project
  - Need to create an Experience Base to capture history on this project and others
- For a more detailed Measurement Template, see the back-up slides

© 2007 Fraunhofer Center Maryland

Software Safety Risk Questions and Metrics

1. **Software Safety Analysis Process:** Check how well each organization, platform and the integrator is addressing software safety in the system hazard analysis process.
   - Are there a *reasonable* number of software safety requirements being identified?

Metric:

Percent Software Safety Requirements (PSSR) = The percentage of software safety requirements relative to the total number of software requirements

Interpretation:

If the number of identified software safety requirements is not "reasonable" relative to the platform family or in line with system safety in general, then

- **too few → safety risk**
- **too many →cost and schedule risk**

© 2007 Fraunhofer Center Maryland

Software Safety Risk Questions and Metrics

2. **Hazard identification:** Check if a reasonable number of software-related hazards, causes, controls, and verifications identified
   - ❖ Have a reasonable number of software safety hazards been identified?
   - ❖ Are causes, controls and verifications being generated over time?
   - ❖ Does every cause have at least one control?
   - ❖ Does every control have at least one verification?

Metrics:

Percent Software Hazards (PSH) = #software safety hazards / #system safety hazards
Controls with causes (CwC) = # causes for which there is a control / # of causes for all hazards in the HTS for the current build
Verifications with controls (VwC) = # controls for which there is a verification / # of controls for all causes in the HTS for the current build

Interpretation: If number of identified hazards not "reasonable" ***and*** there are causes without controls ***and*** there are controls without verification
   - → **Problems with the hazard management process for software**

© 2007 Fraunhofer Center Maryland

Software Safety Risk Questions and Metrics

3. Hazard Management: Check if software-related hazards (and hazard software components, i.e., causes, controls, and verifications) are identified and closed at an appropriate rate

- Is the number of open software causes/controls for hazards shrinking over time?

Metric: Hazard cause/control closure evolution (HCCE)

This is a three point moving average of the set of *open causes/controls* OC_i at three consecutive time intervals

$HCCE_{i,3} = MA_{i+1,3} / MA_{i,3}$ where $MA_{i,3} = (OC_{i-2} + OC_{i-1} + OC_i)/ 3$

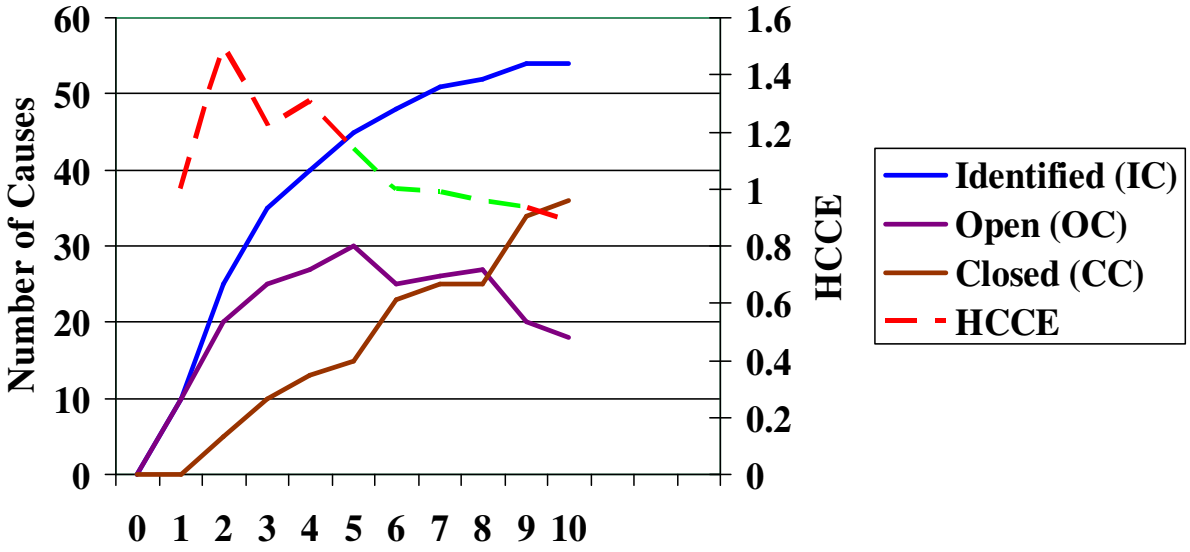
Interpretation: If the moving average of open causes or controls > 1 → hazard causes or controls are opening faster than they are closing

Note: Interpretation of the value depends on where the product is in the life-cycle. Values greater than 1 are ok early on but should remain < 1 after some point in time as fielding milestones are reached.

© 2007 Fraunhofer Center Maryland

Graphing Cause Identification & Closure

Graphing the cumulative identified (IC), open (OC), and closed causes (CC) provides good insight into the variables' trends



Visibility Questions:

For the number of identified causes: Is there an *increasing evolution* in the earlier phases, and *stability* (small increase, if at all) in the later phases?

For the number of closed causes: Is there an *increasing evolution* (HCCE <1)?

For the number of open causes: Is there a *decreasing evolution* (shrinking) and *convergence* to zero?

© 2007 Fraunhofer Center Maryland

Software Safety Risk Questions and Metrics

4. **Appropriate Levels of Rigor for Software Safety:** Check if the various software development groups are assigning reasonable levels of rigor to safety-related software requirements
   - Are the appropriate levels of rigor being applied to the safety-related software items for developed software?

Metric:

Percent requirements (PRLOR) and code (PCLOR) for each level of rigor

- If data cannot be supplied for these two metrics, it is questionable that the level of rigor concept is applied

Interpretation:

If the percents for each level of rigor are not within "reasonable" values, then safety related software might not be developed according to the appropriate level of rigor

- Disproportionately high LOR percentages → cost and schedule risk

© 2007 Fraunhofer Center Maryland

Software Safety Risk Questions and Metrics

5. **Safety Defects:** Check if software safety-related defects are being dealt with

   a) How many software safety requirements do not have a test case associated with them?

Metric:

- **Unlinked safety related software requirements to Test Cases (UTC)** = the number of safety requirements (controls) that are **not** linked to test cases

Interpretation:

If UTC is not approaching 0 as testing begins → the safety risks are not being tested

© 2007 Fraunhofer Center Maryland

Software Safety Risk Questions and Metrics

5. Safety Defects: Check if software safety-related defects are being dealt with

b) Are software safety-related defects being closed?

Metric:

- Count by priority of open safety-related software trouble reports at time i (COSRTR)
- List by priority of open safety-related software trouble reports at time i (LOSRTR)

Interpretation:

- If **COSRTR ≠ 0** then show list and do further analysis
- This metric should be taken periodically starting at the beginning of test up until SAR delivery. At SAR delivery ensure all LOSRTRs are addressed in SAR

© 2007 Fraunhofer Center Maryland

Experiences/Lessons Learned (1/3)

1. **The HTS must allow identification of software hazards, controls, etc. and have traceability to the system requirements**
   - On large, complex systems, Software and Engineering organizations usually work separately
   - Each organizations must be able to manage identifying and controlling hazards for their products
   - Traceability in the HTS is needed to ensure that hazard controls/mitigations are formally part of the system requirements and implementation can be tracked
2. **Safety engineers need to consider both software and system safety to be effective**
3. **It is important to design your Hazard Tracking System (HTS) to support metrics collection to take full advantage of automation**
   - The information in the hazard tracking system needs to be accurate, clear, and specific in order to understand and track hazards

© 2007 Fraunhofer Center Maryland

Experiences/Lessons Learned (2/3)

4. **Need to recognize common causes and controls in the HTS**
   - Many hazards can have their roots in the same cause and many causes can be mitigated by the same control
   - Need to coordinate and link the causes and controls in the HTS
5. **Software-related *hazards, causes, and controls* need to be marked as such**
   - A software-related hazard indicates that something must be done specifically to the software to handle the hazard
   - The process of analyzing hazards and determining controls to mitigate the risks can identify additional requirements and verification tests to ensure that the system is safe
6. **Identifying the right level of rigor is critical to balancing the cost of safety with the risk**
   - Level of rigor defines the level of response needed from the software as determined by the importance and complexity of the hazard

© 2007 Fraunhofer Center Maryland

Experiences/Lessons Learned (3/3)

7. Internal processes will affect metrics and interpretation of metrics

- If hazards are not closed until verification is complete, no progress metric will be available until too late
- Need to track progress on other steps in the process

8. The safety metrics are the beginning of an Experience Base

- Creates an historical base across a current project and for future projects
  - Models, metrics, estimated values, and ranges
  - Sets of potential hazards, causes and controls for different families
  - Bounds can evolve as we start to see the real data from the particular project
- The Experience Base provides a basis to re-calibrate the models when needed

© 2007 Fraunhofer Center Maryland

Summary

- Software metrics and approach were developed from experience on a large DoD system of systems program, working with the safety engineering group
- Software is an ever-increasing component of system safety
- Developing software safety metrics helps to
  - Make software safety visible
  - Monitor the quality of the safety process
  - Make decisions that mitigate risks and prevent safety issues
- Software Safety Metrics collection does not have to be an new activity
  - Can leverage existing data, processes, tools, etc.

© 2007 Fraunhofer Center Maryland

Questions

© 2007 Fraunhofer Center Maryland

Contact Information

Victor Basili
University of Maryland
Fraunhofer Center Maryland
vbasili@fc-md.umd.edu

Kathleen Dangle
Fraunhofer Center Maryland
kdangle@fc-md.umd.edu

Linda Esker
Fraunhofer Center Maryland
lesker@fc-md.umd.edu

Frank Marotta
U. S. Army Aberdeen Test Center
frank.marotta@atc.army.mil

© 2007 Fraunhofer Center Maryland

References

- Joint Software System Safety Committee, *Software System Safety Handbook*, December 1999.
- MIL-STD-882, *DoD Standard Practice for System Safety*, 10 February 2000.

© 2007 Fraunhofer Center Maryland

Acronyms

CC	Closed Cause
COPSRTR	Count by Priority of open Safety-related Trouble Reports
CPC	Controls Per Cause
GQM	Goal, Question, Metric
HCCE	Hazard Cause/Control Closure Evolution
HTS	Hazard Tracking System
IC	Identified Cause
LOR	Level of Rigor
LOSRTR	List by Priority of open Safety-related Trouble Reports
MA	Moving Average
NSSR	Newly Defined System Safety Requirements
OC	Open Cause
PRLOR	Percent Requirements for LOR
PCLOR	Percent of Code at LOR
PSH	Percent Software Hazards
PSSR	Percent Software Safety Requirements
SAR	Safety Assessment Report
SSH	System Safety Hazards
SSR	Existing System Safety Requirements
SR	System Requirements
UTC	Unlinked Test Cases
VPC	Verifications Per Control

© 2007 Fraunhofer Center Maryland

Additional Information

The following slides will not be presented because of time limitations.

© 2007 Fraunhofer Center Maryland

Key Definitions

- **Safety**: Freedom from those conditions that could result in death, injury, occupational illness, damage to or loss of equipment or property, or damage to the environment.
- **System Safety**: The application of engineering and management principles, criteria, and techniques to achieve acceptable mishap risk, within the constraints of operational effectiveness and suitability, time, and cost, throughout all phases of the system life cycle.
- **Hazard**: Any real or potential condition that can cause injury, illness, or death to personnel; damage to or loss of a system, equipment or property; or damage to the environment.
- **Level or Rigor**: The amount of requirements analysis, development discipline, testing, and configuration control required to mitigate the potential safety risks of the software component.

© 2007 Fraunhofer Center Maryland

Example Software Level of Rigor Matrix

Hazard Severity	Software Level of Autonomy					
	(I) **Software exercises autonomous control over potentially hazardous hardware systems, subsystems or components without the possibility of intervention to preclude the occurrence of the hazardous event.**	*(IIa)* **Software exercises control over potentially hazardous hardware systems, subsystems or components allowing time for intervention by independent safety systems to mitigate the hazard. However, these systems by themselves are not considered adequate.**	*(IIb)* **Software item displays information requiring immediate operator action to mitigate a hazard. Software failures will allow or fail to prevent the hazard's occurrence**	*(IIIa)* **Software item issues commands over potentially hazardous hardware systems, subsystems or components requiring operator action to complete the control function. There are several, redundant, independent safety measures for each hazardous event.**	*(IIIb)* **Software generates information of a safety critical nature used to make safety critical decisions. There are several, redundant, independent safety measures for each hazardous event.**	*(IV)* **Software does not control safety critical hardware systems, subsystems or components and does not provide safety critical information.**
Catastrophic (I)	LOR3	LOR3	LOR3	LOR2	LOR2	N/A
Critical (II)	LOR3	LOR2	LOR2	LOR2	LOR2	N/A
Marginal (III)	LOR2	LOR2	LOR2	LOR1	LOR1	N/A
Negligible (IV)	LOR1	LOR1	LOR1	LOR1	LOR1	N/A

© 2007 Fraunhofer Center Maryland

Example Software Development Products for Each LOR

- LOR 1- Lowest
  - Software Safety Requirements
  - Hazard Mitigation Traceability Matrix
  - Functional or System Hazard Analysis
  - Hazard control Records
  - Computer Program Change Requests
  - System or Functional Testing
- LOR 2 –
  - Above plus
  - Design analysis with updates to requirements hazard analysis products
  - Functional hazard analysis
  - Functional testing
  - Stress & stability testing
- LOR 3 – Highest
  - All the above plus
  - Code walkthroughs
  - Condition/decision structural test with safety mitigation records

© 2007 Fraunhofer Center Maryland

Additional Details on Metrics

The following slides provide an example of additional Measurement Template details for the software safety risk metrics developed for measurement area

1. Software Safety Analysis Process

© 2007 Fraunhofer Center Maryland

Measurement Template

- For each metric[1] we supply the
  - **Question** being addressed with the aid of the metric
  - The **metric** definition
  - The **model**[2] recommended[3]
  - The **scope**[4] of application of the metric
  - Suggested **response**s to the application of the model

1 By "metric" we mean basic measures and derived metrics, as well as evidence of some sort (e.g., existence of documents or processes)

2 By "model" we mean the interpretation of the values of the metric(s), in order to answer the target question

3 We have chosen initial estimates of the bounds in the models because we do not have historical data. However these bounds can evolve as we start to see the real data.

4 Scope: We assume these metrics are taken for each software supplier, platform or combination of platforms, unless otherwise specified.

© 2007 Fraunhofer Center Maryland

1. Software Safety Analysis Process

Check how well each organization, platform and the integrator is addressing software safety in the system hazard analysis process.

Measurement Question: Are there a reasonable number of software safety requirements being identified?

Metric: **Percent Software Safety Requirements (PSSR)** **=** the percentage of software safety requirements relative to the total number of software requirements.

Interpretation: If the number of identified software safety requirements is not "reasonable" relative to the platform family or in line with system safety in general, then

- **too few → safety risk**
- **too many →cost and schedule risk**

© 2007 Fraunhofer Center Maryland

1. Metric: Percent Software Safety Requirements (PSSR)

Question: Are there a reasonable number of software safety requirements being identified?

Metric: **PSSR =** # software safety requirements / # software requirements

Model: ***if*** **|PSSR - EPSSR| < e** ***then*** a reasonable number of software safety requirements have been identified

where

EPSSR = the average of the PSSRs for all platforms in the family, (*in line with other systems*) and **e** = σ(EPSSR) or

EPSSR = #system safety requirements / #system requirements, (*in line with system safety in general*) and **e** = 20% of EPSSR.

Response: PSSR not being within the range of EPSSR should indicate the need for a management action, e.g., check into the safety analysis process and whether it is being applied right; come up with a "get well" plan or investigate the reason why the platform under consideration has such a small (or large) number of safety requirements. If too large, what are the cost and schedule implications?

© 2007 Fraunhofer Center Maryland

1. Metric: Percent Software Safety Requirements (PSSR)

Metric: PSSR = # software safety requirements / # software requirements

Data Needed:

software safety requirements for the platform being considered
#software requirements for the platform being considered

AND

#software safety requirements for other platforms in the system
#software requirements for other platforms in the system

Or

#system safety requirements
#system requirements

© 2007 Fraunhofer Center Maryland