# Introduction to Software Safety

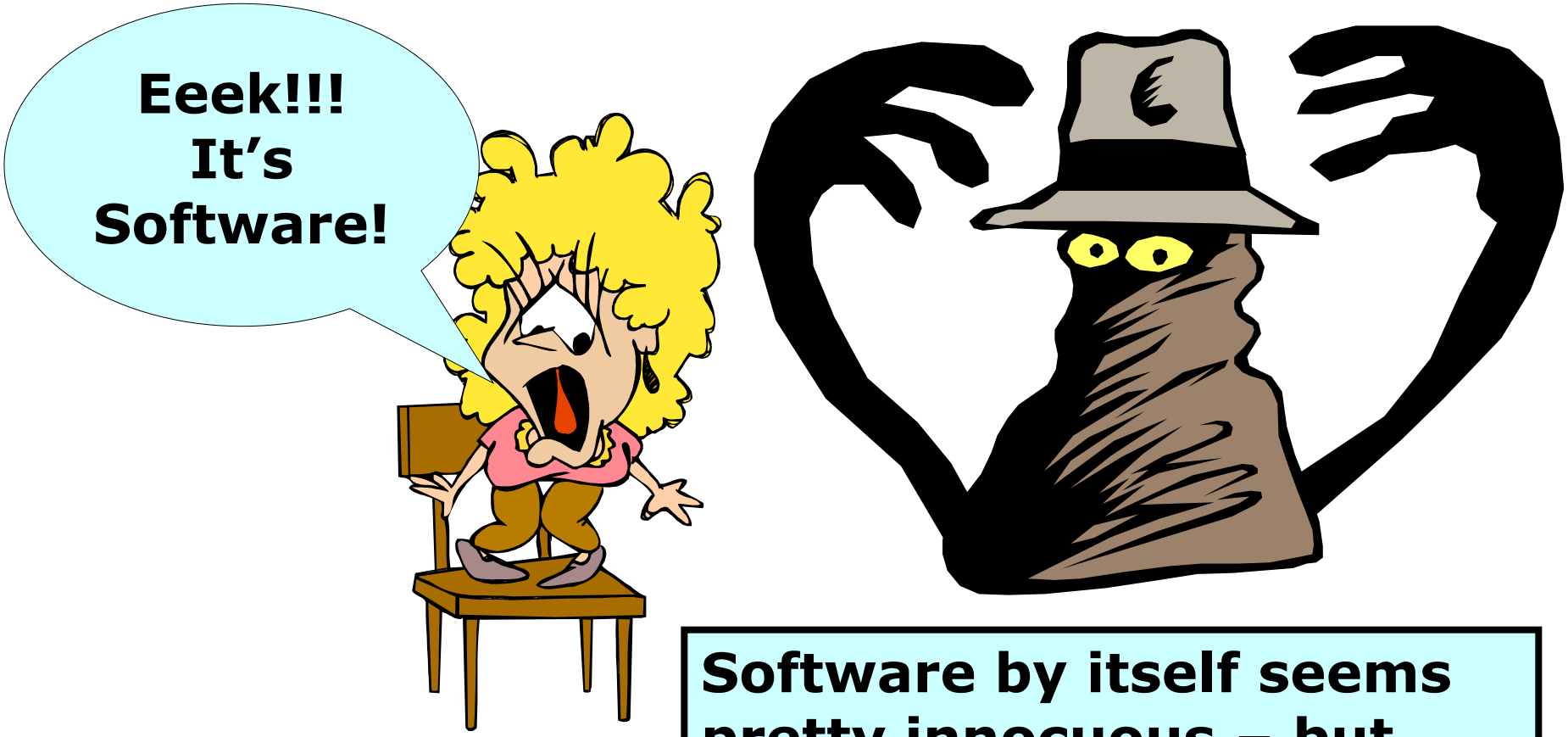
**Dennis J. Frailey**
**Frailey@Lyle.SMU.EDU**

# What This Talk is All About

- This is an introduction to the topic of software safety
- It is based on existing government and commercial standards
- It is intended to explain what software safety means, how software can contribute to safety problems, and what techniques are used to deal with safety-critical software

# Why Be Concerned about Software Safety

- **Can Software Harm Anyone?**




**Software by itself seems pretty innocuous – but …**

# Ways that Software can Harm Someone

- **It can Control the Behavior of Dangerous Devices**
  - Robots
  - Weapons
  - Security Doors at Building entry
  - Medical Devices
  - Chemical Experiments
  - Factory Manufacturing Lines
  - …
- **It can Send Information to People who do Potentially Dangerous Things**
  - Location of Airplanes for ATC
  - Identification of Intruders
  - …
- **It can Deceive**
  - Internet scams
- **And on and on …**

# Software is often used for tasks that once called for human judgment

# Software Safety Starts with System Safety

- **Software is always part of a system**
  - A data base
  - A network
  - A vehicle
  - …
- **If the system can harm someone, then the software may be a factor in whether the system harms someone**
- **So we have to start by analyzing the safety issues of the system**

# Basic Terminology

**Mil-STD 882D**

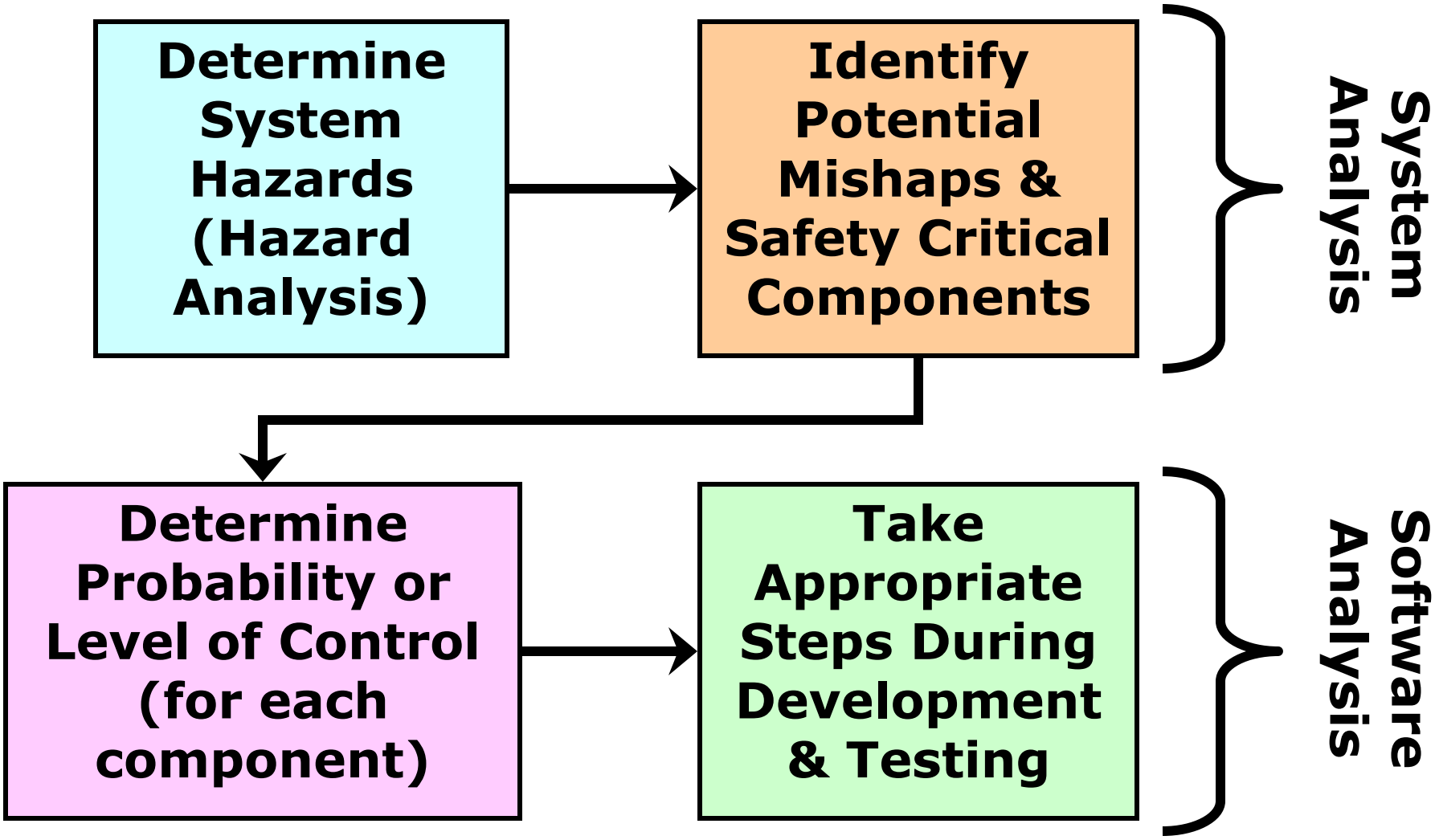
- **Hazard – Any real or potential condition that can cause injury, illness, or death [to a person] or damage to … or loss of … [property or the environment]**
- **Mishap – An unplanned event or series of events resulting in death, injury, …**
- **Safety – Freedom from … conditions that can cause injury, illness, …**

# Examples

- **Hazard: the power plant might catch fire**

- **Mishap:**
  - an incorrect temperature reading may cause
  - a heater to go on, resulting in
  - overheating of a chemical mixture, leading to
  - a chemical reaction, producing
  - excessive pressure, causing
  - an explosion in the fuel storage area, producing
  - a major fire in the power plant

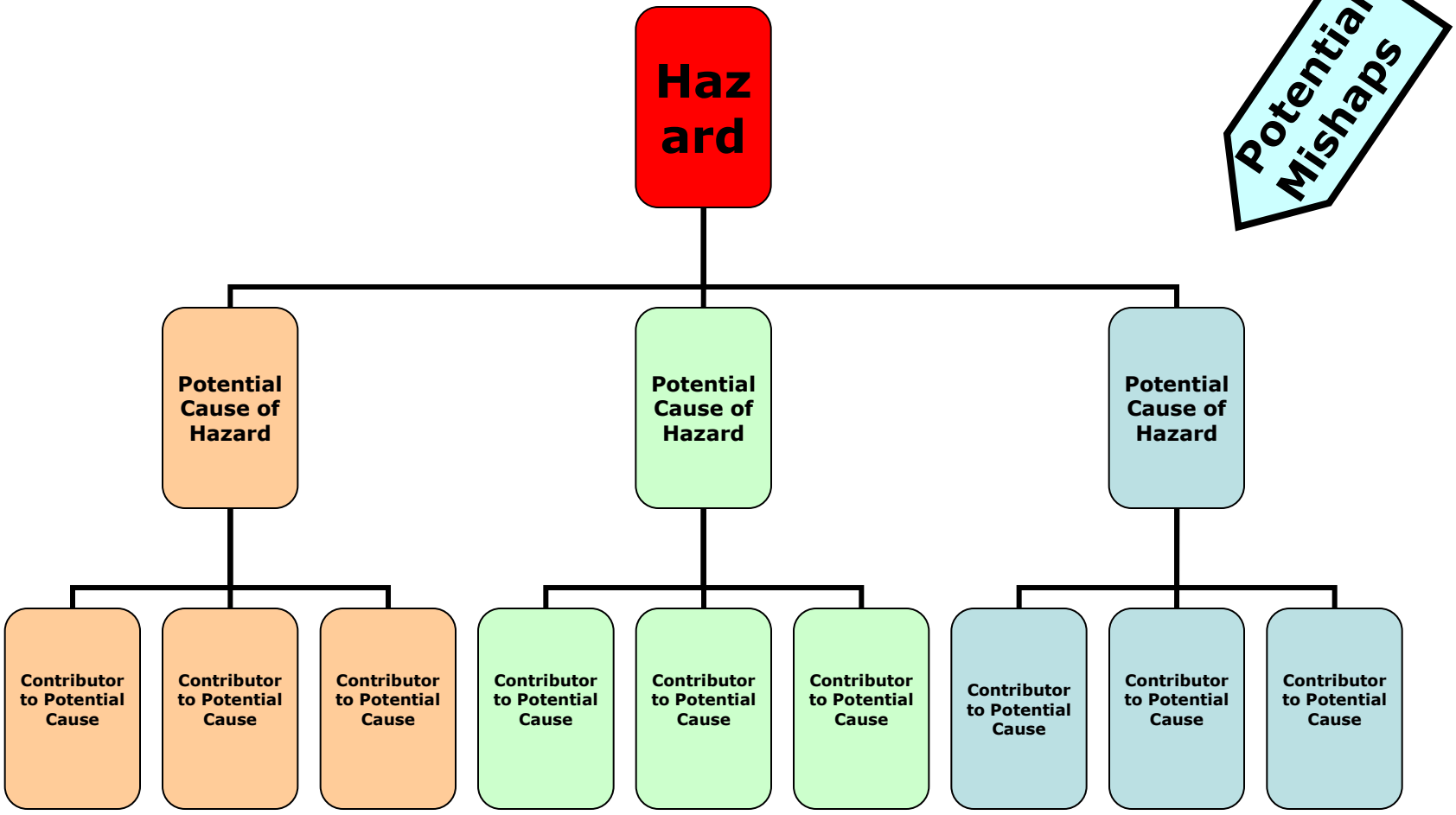
# The Safety Process



# Basic System Safety Process in More Detail

1. Identify the potential hazards
2. Decompose hazard threads (potential mishaps) through subsystem components, including software
3. Link/trace to requirements
4. Generate appropriate mitigation strategy
5. Implement the mitigation
6. Verify that the mitigation is implemented and that it functions as expected
7. Document safety artifacts

# Fault Tree Concept




**Keep going down until you reach a point where you can do something about it.**

# An Example of a System Level Hazard List


- Uncontrolled explosion
- Uncontrolled fire
- Injury and/or illness
- Blockage of ingress/egress paths
- Structural failure
- Collision
- Uncontrolled activation of ordinance
- Electromagnetic interference
- Hazardous/reactive materials
- Electrical energy
- Improper engagement control (Fratricide)
- Surface/air contamination
- Corrosion resulting in loss of strength or integrity of exposed surfaces
- Batteries (exposure to toxic material, explosion)
- Radiation (ionizing and non-ionizing)
- Uncontrolled/unsupervised robotic operations

# A Safety Critical Component is …

… a component that might contribute to a hazard if it fails

- Note that in most hazardous systems there are many components that can contribute to a safety hazard
- Some components may be software

**Identify Safety Critical Components**

**Software is often used to replace mechanical or human controls.**

**When software controls something it is often potentially safety critical.**

**But that's not the only way software can be safety critical.**

# Some Software Functions that May be Safety Critical

Potential Cause of Hazard

- Assessment of overall system health (e.g, power-up and run-time monitors, heartbeat, program memory CRCs, range checks, CPU health)
- Enforcement of critical timing
- Exception trapping and handling including, failure /malfunction detection, isolation, and containment
- Functions that execute the system's response to detected failures/malfunctions
- Functions that enhance the system's survivability (preservation of core functionality)
- Data quarantine/sanitization
- Range and reasonableness (sanity) checking
- Tamper and cyber-attack proofing
- Authentication for lethal actions
- Inhibiting functions and interlocks

# Example Software Contribution to a Mishap (1/3)

| Example SW Contribution | Symptom | Example Potential Cause |
|---|---|---|
| Incorrect Safety-Critical Alerts and Warnings. | Safety Critical alerts are incorrect, or are not triggered by Safety Critical Events. Alerts fail to warn the user of an unsafe condition, and or an Unsafe System State. | Software fails to alert the operator to unsafe condition and/or state. Alerts can be audio, visual, or in text format in a log, etc.. |
| Incorrect Data Transfer Messages (transmit and receive). | Data transferred in the wrong format and the Safety Critical Data are interpreted incorrectly. | Failure to validate data transfer with the appropriate parity, check sums to validate Safety Critical data. |

# Example Software Contribution to a Mishap (2/3)

| Example SW Contribution | Symptom | Example Potential Cause |
|---|---|---|
| Data Storage Failures (Safety Critical Data corrupted and or lost) | Safety-Critical Displays are confusing, and/or incorrect in presenting Safety Critical Data. | Safety critical data were not properly check-summed; data overwritten by mistake |
| Incorrect data transfer between processors | Incorrect message received | Failure to perform verification checks in both processors prior to transferring Safety Critical data. |

# Example Software Contribution to a Mishap (3/3)

| Example SW Contribution | Description | Example Potential Cause |
|---|---|---|
| Timing and Interrupt failures | Interrupts occur at the wrong time and potentially interrupt a Safety Critical process, or introduce a potential hazard. | Interrupts are out of synch with system time, and/or interrupt a Safety Critical Process/Path which introduces a potential hazard. |
| Incorrect Modes | Software signals the system to fire a weapon when it should not. | Switch from training to "live" mode is not correctly reflected in the "mode" variable. |

# So What Should We Do About Safety Critical Software?

**Mitigate the Risks**

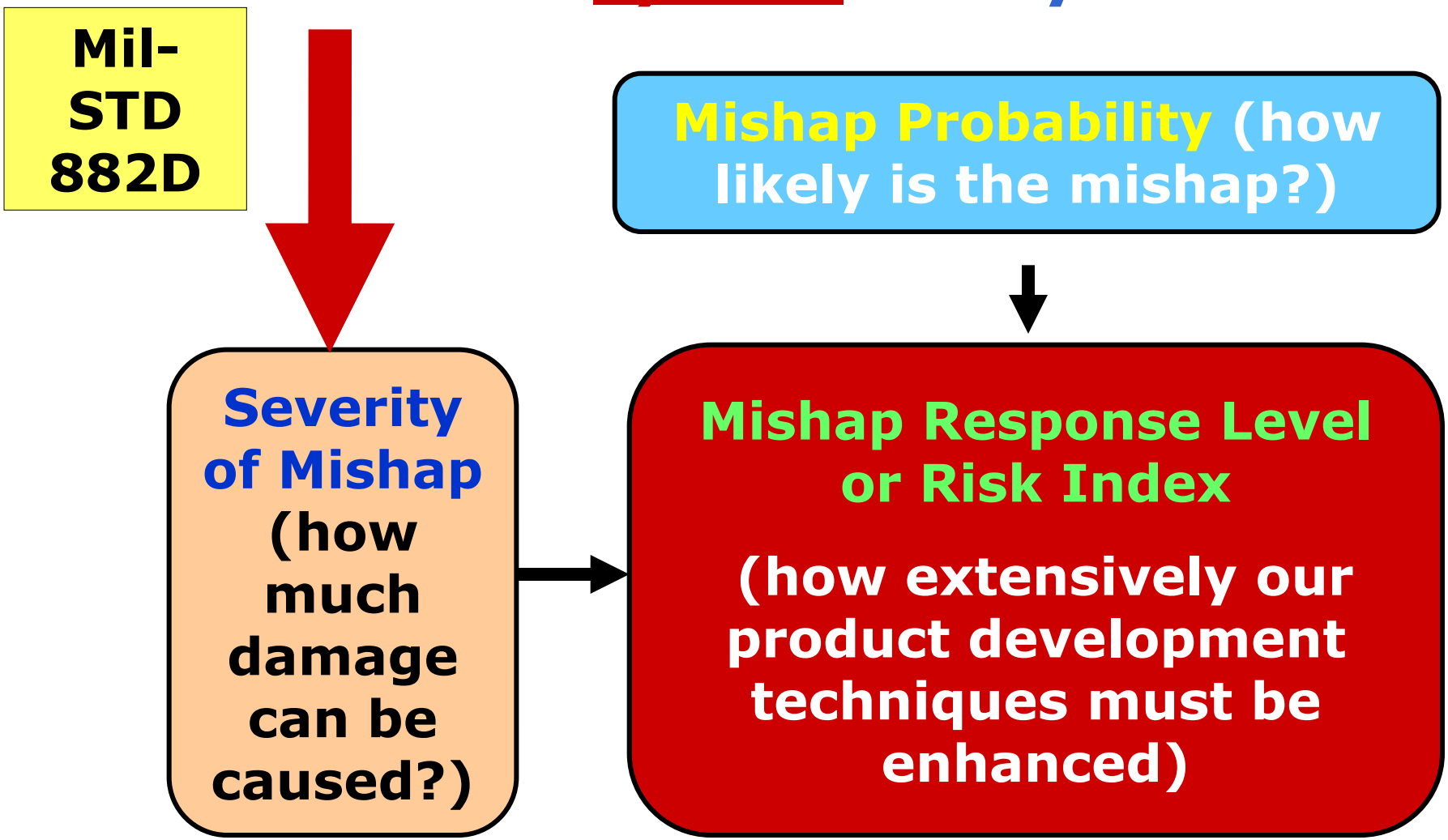
**In other words, make them**
**less likely to happen**
**and/or**
**less harmful if they do happen**

# Potential Software Mitigation Strategies

Not a Comprehensive List

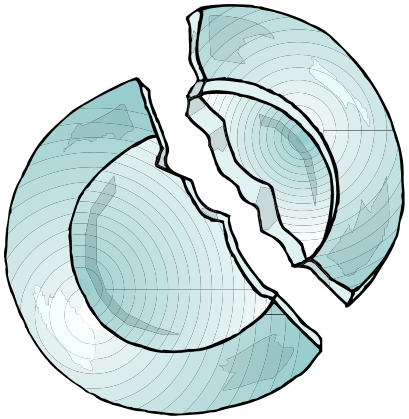
- Wrappers to Constrain Inputs/Outputs
- Timers/Heartbeats/ Counters
- Fault Trapping
- Error Trapping
- Command Retry
- Parity/Checksums/ CRCs
- Redundancy/Voting
- Monitoring
- Synchronization/Timing
- Timeout
- Reasonableness Procedures
- Interlocks/Inhibits
- Range/Sequence Checking
- Transition Checking
- Hysteresis for Man-Machine Interactions to allow time for Human Interpretation
- Partitioning/Isolation
- Redundant but Dissimilar Algorithms
- Functional Separation
- Failsafe Strategies
- Fault/Error Tolerant Strategies

# How Seriously Must we Treat Potential *System* Safety Issues?



# Severity of Mishap (how much damage can be caused?)

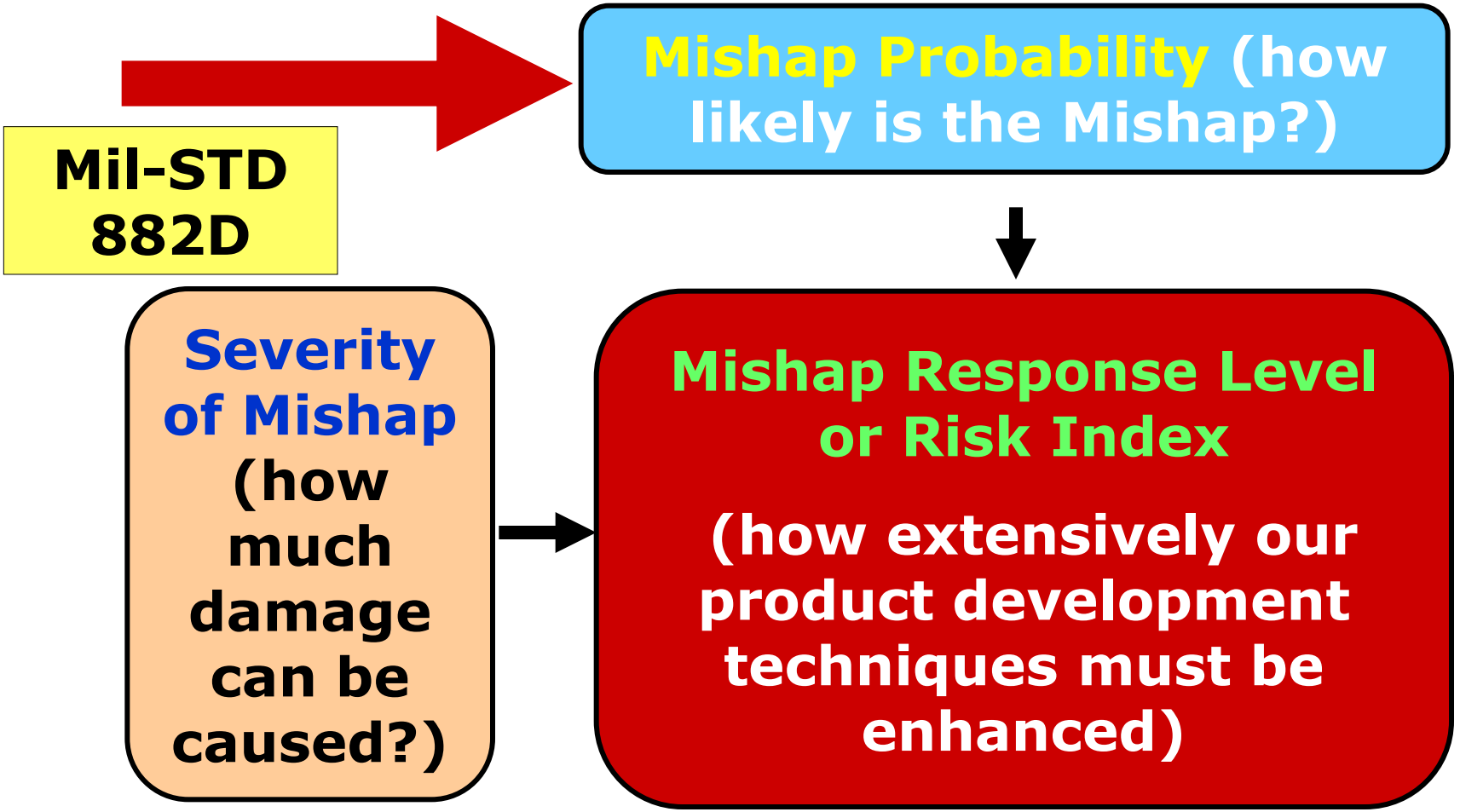
- This can be a very subjective evaluation
- Any mishap should be avoided, but some are worse than others
- The safety community for any application domain will, by and large, agree on a set of severity categories (see next slide)



# Mishap Severity Categories Commonly Used in Major Commercial and Military Safety Standards

- **Catastrophic** – could result in death, permanent total disability, loss exceeding $1M, or severe environmental damage violating law or regulation
- **Critical** – could result in permanent partial disability, injuries or illness affecting at least 3 people, loss exceeding $200K, or reversible damage to environment violating law or regulation
- **Marginal** – could result in injury or illness resulting in loss of 1 or more work days, loss exceeding $10K, or mitigatable environmental damage without violation of law or regulation where restoration activities can be accomplished
- **Negligible** – could result in injury or illness not resulting in lost workdays, loss exceeding $2K, or minimal environmental damage not violating law or regulations

# How Seriously Must we Treat Potential *System* Safety Issues?



# Mishap Probability (how likely is the Mishap?)

| Typical Mishap Probabilities | | | |
|---|---|---|---|
| **Level** | **Description** | **Description** | **Probability** |
| **A** | **Frequent** | **Will Occur** | **1 in 100** |
| **B** | **Probable** | **Likely to Occur** | **1 in 1000** |
| **C** | **Occasional** | **Unlikely but Possible** | **1 in 10,000** |
| **D** | **Remote** | **Very Unlikely** | **1 in 100,000** |
| **E** | **Improbable** | **Can Assume Will Not Occur** | **1 in 1,000,000** |

**Probability Level is Used to Select Appropriate Mitigation Actions**

# Mishap Risk Index Combines Severity and Probability

| Typical Mishap Risk Index Calculation | | | | | |
|---|---|---|---|---|---|
| **Probability** | **A** | **B** | **C** | **D** | **E** |
| **Severity** | **Frequent** | **Probable** | **Occasional** | **Remote** | **Improbable** |
| **I - Catastrophic** | **U** | **U** | **M** | **M** | **N** |
| **II - Critical** | **U** | **U** | **M** | **N** | **N** |
| **III - Marginal** | **U** | **M** | **N** | **N** | **N** |
| **IV - Negligible** | **M** | **N** | **N** | **N** | **N** |
| **U - Unacceptable** | **Mitigate at High Cost** | | | | |
| **M – Marginal Risk** | **Mitigate at Moderate Cost** | | | | |
| **N – No Significant Risk** | **Mitigate at Low Cost** | | | | |

# Mishap Probability (how likely is the Mishap?)

## The Problem with Probability for Software

- One can deduce the probability of hardware failure by examining the properties of materials, laws of physics, data on material fatigue, etc.
- But one cannot deduce the probability of software failure in such a manner.
- So for software, most safety experts use a different approach (next slide).

# How Seriously Must we Treat Potential *Software* Safety Issues?

**Level of Control** (to what degree is software responsible?)

**Severity of Mishap** (how much damage can be caused?)

Software **Level of Rigor** (how extensively our software development techniques must be enhanced)

# Level of Control (to what degree is software responsible?)

- The ***Software Level of Control*** is a measure of the degree to which software is responsible for the behavior of a system or of a specific system action that may lead to a safety mishap

- The higher the level of control, the more rigorous the process for software development

# An Example of Software Levels of Control (1/4)

- **I – Autonomous/Time Critical** - *Software exercises autonomous control* over potentially hazardous hardware systems, subsystems, or components *without the possibility of intervention* to preclude the occurrence of a mishap. Failure of the software or a failure to prevent an event leads directly to a mishap's occurrence. This implies *no other failure is required to cause the mishap*.

# An Example of Software Levels of Control (2/4)

- **IIa - Autonomous/Not Time Critical** - *Software exercises control* over potentially hazardous hardware systems, subsystems, or components *and allows time for intervention* by independent safety systems to mitigate the mishap. This implies that corrective action is possible and a second fault or error is required for this mishap to occur.

- **IIb Information Time Critical** - *Software displays information requiring immediate operator action* to mitigate a hazard. Software failures will allow or fail to prevent the occurrence of a mishap. This implies that *the operator is made aware of the existence of the mishap and intervention is possible*.

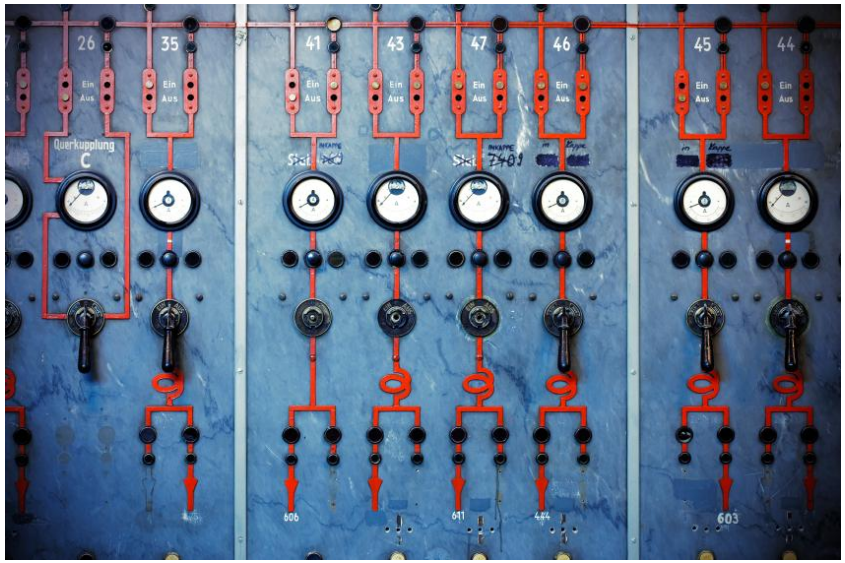
# An Example of Software Levels of Control (3/4)

- **IIIa Operator Controlled** - *Software issues commands* over potentially hazardous hardware system subsystems or components *requiring human action to complete* the control function. *There are several redundant, independent safety measures for each hazardous event*.

- **IIIb Information Decision** - *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*.

# An Example of Software Levels of Control (4/4)

- **IV Not Safety Software** - *Software does not control safety-critical hardware systems*, subsystems, or components and *does not provide safety-critical information*.



# Summary of Software Levels of Control

**A software fault may:**

**(1) result directly in a mishap (LOC I),**

**(2) significantly impact the margin of safety (LOC IIa or IIb), but not result directly in a mishap, or**

**(3) have a minor impact on the margin of safety for a mishap (LOC IIIa or IIIb).**

# Software Level of Rigor (how extensively our SW development techniques must be enhanced)

| Mishap Severity | Software Level of Control | | | | | |
|---|---|---|---|---|---|---|
| | I – Autonomous / Time Critical | IIb – Information / Time Critical | IIa - Autonomous/ Not Time Critical | IIIa - Operator Controlled | IIIb - Information Decision | IV - Not Safety Software |
| I - Catastrophic | LOR-3 | LOR-3 | LOR-2 | LOR-2 | LOR-2 | N/A |
| II - Critical | LOR-3 | LOR-3 | LOR-2 | LOR-2 | LOR-2 | N/A |
| III - Marginal | LOR-2 | LOR-2 | LOR-2 | LOR-1 | LOR-1 | N/A |
| IV - Negligible | LOR-1 | LOR-1 | LOR-1 | LOR-1 | LOR-1 | N/A |

# Examples of Appropriate Actions

**Take Appropriate Steps During Development & Testing**

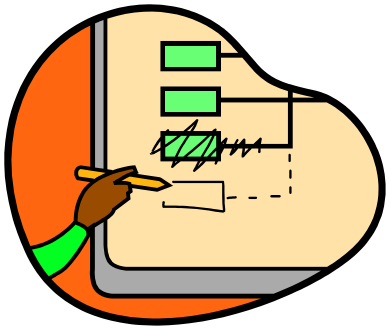
| Phase | Actions | LOR 1 | LOR 2 | LOR 3 |
|---|---|---|---|---|
| Requirements | Safety Requirements Documented | ✓ | ✓ | ✓ |
| | Hazard Analysis | ✓ | ✓ | ✓ |
| | Trace Hazards to Requirements | ✓ | ✓ | ✓ |
| Design | Trace Hazards to Design | ✓ | ✓ | |
| Code | Check Code for Safety Issues during Code Inspections or Walkthroughs | ✓ | | |
| Test | Functional Testing | ✓ | ✓ | ✓ |
| | Stress Testing | ✓ | ✓ | |
| | Stability Testing | ✓ | ✓ | |
| | 100% Branch Testing | ✓ | ✓ | |
| During All Phases | Change Requests must be Evaluated for Impact on Safety Critical Software | ✓ | ✓ | ✓ |
| | Maintain Hazard Control Records | ✓ | ✓ | ✓ |

# Most of the Techniques Described Make Sense in a Waterfall Type of Development Process

**What about Agile Development, Incremental Development, etc.?**

# Challenges for Agile and Incremental Development

- **Changes are constantly and frequently being made, so things you do to assure safety, such as comprehensive tests and verifications, may need to be done over and over again and may slow development down significantly**



# SW Safety Strategy for Agile & Incremental Development (1/2)

- **Experimental and prototype functionality cannot be used in critical applications until formally developed and verified**
  - partitioning must be used to segregate prototype/experimental functionality from mainstream functionality
- **Place limitations on the use of functions that have not yet met all safety criteria**
  - e.g., allow only non-critical applications to use capabilities until all safety criteria are satisfied

# SW Safety Strategy for Agile & Incremental Development (2/2)

- **Lift restrictions when functions have successfully exited formal development and verification**
- **Consider the economics of developing a robust, scaled-down version of a function for safety critical applications**
- **Partition critical functions from non-critical functions**
  - a failure/ malfunction in a non-critical function must not cause a failure/ malfunction in a critical function

**All of the Above Assumes *You***
**Write the Software**
**(and therefore control the development process)**

**What if Somebody Else Writes It?**

# Challenges for COTS Software

- **COTS (Commercial Off-the-Shelf) software examples**
  - Operating systems (VXworks™, Windows™)
  - "standard" I/O drivers or interfaces
  - Board support packages
- **COTS has these challenges:**
  - You don't control the software development process, standards, etc.
  - You may not be allowed to examine the source code

# Techniques for Safety Critical COTS Software

- **Safety Certification** – there are organizations that will certify the safety or security of a software package.
- **Wrappers** - that control the inputs and outputs of COTS software to inhibit likely failure modes.
- **Analyses** – to assess the reliability and failure modes of COTS software.
- **Extensive Tests** – covering a wide range of operational conditions under limited resources, including stress testing, stability testing, and others.
- **Limit use** - to limited-complexity applications where reliability can be readily and accurately assessed.
- **Redesign** - to eliminate COTS from the design.

# Challenges for FOSS Software

- **FOSS (Free or Open Source Software) examples**
  - Many components of a GUI
  - Compilers
  - …
- **FOSS has these additional challenges beyond COTS:**
  - You may be legally required to report changes back to the originator
  - The code may change over time with little notification or configuration control
  - You don't know who wrote the code or what traps they might have embedded in the code

# Other Software Safety Issues

- **Software development is rapidly changing as we learn of new and faster ways to develop software**
  - Faster development techniques often bypass the safeguards needed to address safety concerns
- **Software is often used in applications well beyond what was originally intended**
  - How do you know the software is suitable for a safety critical application?
- **System safety is often not seriously addressed**
  - So software safety may not even be considered

# Future Issues with Software Safety

- **Certifications or licensing for *developers* of "safe" software**
  - CISSP – Certified Information Systems Security Professional
- **Teaching safety issues in software engineering courses**
- **Legal liabilities for those who develop software that has safety implications**
- **…**

# Certification vs Licensing
## Certification

- **A certificate is a document whereby some organization attests that you meet certain qualifications**
  - A college diploma
  - A certificate for completion of a course
  - Microsoft certified system engineer
  - Cisco certified network architect


- **You get a certificate to show that you have some capability, skill, or education**
  - ASQ CSQE - Certified SW Quality Engineer
  - IEEE CSDP – Certified SW Development Professional
- **Any given certification is only as good as the organization it's from**
- **Some employers require certifications for specific types of jobs**

# Certification vs Licensing
# Licensing

- **A license is a <u>legal authorization</u> to practice, usually in a field involving public safety, health or welfare**
  - A medical license
  - A license to practice law
  - A plumbing license
  - A licensed professional engineer
- **Licenses are granted by government agencies**
  - Usually based on experience and examinations
  - Sometimes also based on other factors, such as allowing only so many taxi licenses in a given city
- **Some organizations require that contractors have licensed employees**
  - For example, licensed civil engineers for highway construction projects

# Software Engineering Licensing

- **Can you be licensed?**
  - Texas currently licenses software engineers but the process for further licenses is on hold until an examination is prepared at the national level
  - A national examination is in preparation, sponsored by several organizations and under the technical guidance of IEEE-USA
  - Many states are expected to start licensing SW engineers once the examination is available
  - An ABET accredited SW engineering degree is likely to be preferred but not required
- **Must you be licensed?**
  - If SW licensing is ever required, it will likely be only for safety critical software applications, such as control of power plants, medical applications, etc.

# To Summarize

- **As software is used to control more of the systems that we rely on, software safety will continue to grow in importance**
- **As society recognizes the role of software in these systems, there is likely to be more pressure for training, certification and licensing in software safety**
- **At the same time, the software research & development communities must improve methods of dealing with software safety**

# Some Useful Reference Material (1/2)

- EIA SEB6-A – System Safety Engineering in Software Development
- FAA System Safety Handbook, Appendix J: Software Safety
- Holzmann, Gerard J., The Power of 10: Rules for Developing Savety-Critical Code, IEEE Computer, June, 2006.
- IEEE 1228 – IEEE Standard for Software Safety Plans
- Leveson. Nancy G., Safety-Critical Software Development. In T. Anderson, editor, Safe and Secure Computing Systems, pages 155-162. Blackwell Scientific Publications, 1989.
- Leveson, Nancy G., Safeware: System Safety and Computers, Addison-Wesley, 1995, ISBN-13: 978-0201119725

# Some Useful Reference Material (2/2)

- MIL-STD-882D – Standard Practice for System Safety – US Department of Defense
- NASA-STD-8719.13A – Software Safety
- Patrick R.H. Place & Kyo C. Kang, Safety-Critical Software: Status Report and Annotated Bibliography, Software Engineering Institute, June 1993
- RTCA, Inc., DO-178B, Software Considerations in Airborne Systems and Equipment Certification
- RTCA, Inc., DO-248B, Final report for Clarification of DO-178B
- The DACS Software Reliability Sourcebook Data & Analysis Center for Software

# Questions?