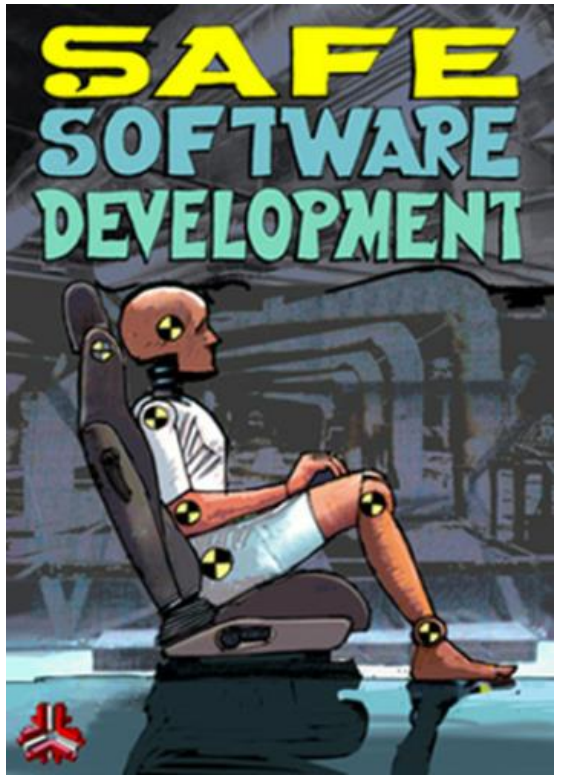
UT Dallas

Introduction to Software Safety



Part 2

What We Can Do About Software Safety

System Control Analysis

Software-specific Issues

Part 2 Agenda

Part 1

- Background, Introduction
- How Software Contributes to Safety and why we need to be concerned about it.
- The Safety Process

Part 2

- **What We Can Do About Software Safety**
- **System Control Analysis (the emerging approach)**
- **Software-specific Issues**
- **Research Reports**
  - (from Exercise 2 in Part 1)

Part 2 Agenda

➢What We Can Do About Software Safety

- System Control Analysis
- Some Software-specific Issues
- Research Reports

So What Should We Do About Safety Critical Software?

Mitigate the Risks

In other words, make the risks

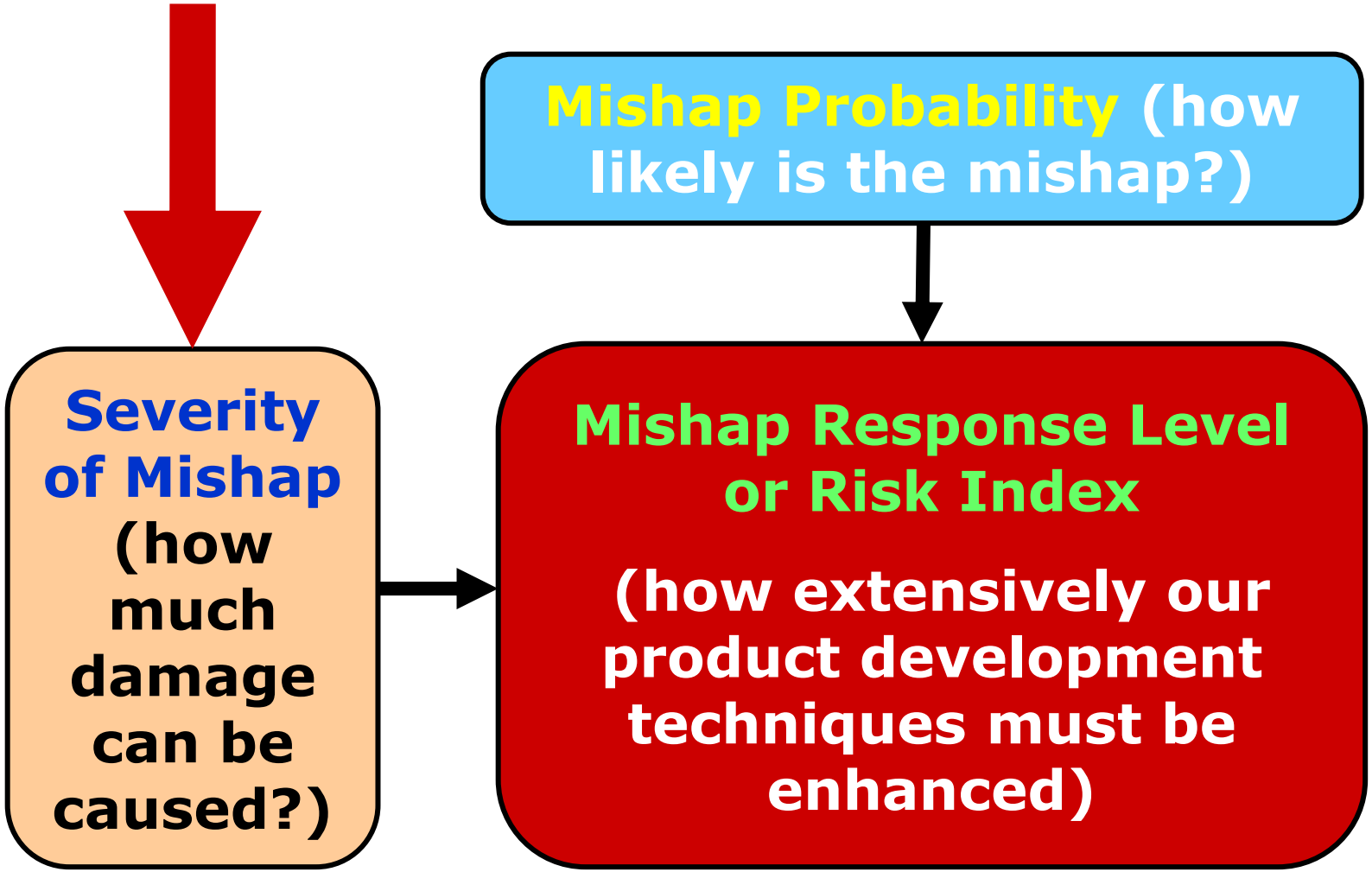
less likely to happen **and/or** **less harmful if they do happen**

Potential Software Safety Risk Mitigation Strategies

Not a Comprehensive List

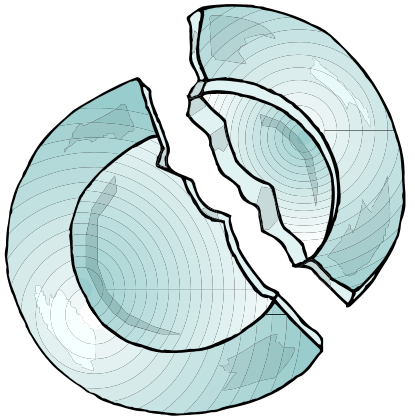
- Wrappers to Constrain Inputs/Outputs
- Timers/Heartbeats/ Counters
- Fault Trapping
- Error Trapping
- Command Retry
- Parity/Checksums/ CRCs
- Redundancy/Voting
- Monitoring
- Synchronization/Timing
- Timeout
- Reasonableness Procedures
- Interlocks/Inhibits
- Range/Sequence Checking
- Transition 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

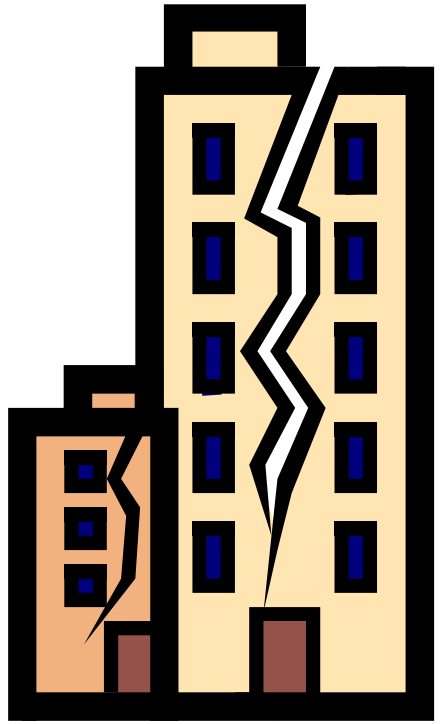
How Seriously Must we Treat Potential *System* Safety Issues?



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 Levels Used in Automotive Safety

The key issue here is controllability: the ability of the vehicle occupants to control the safety of the situation following a failure.

Example: failure of power window and door locks after an accident may prevent exiting a car after an accident.

Controllability	Acceptable Failure Rate
Uncontrollable	Extremely improbable
Difficult to Control	Very Remote
Debilitating	Remote
Distracting	Unlikely
Nuisance	Reasonably possible

Severity Levels in IEC 62304 – Medical Device Software Standard

The key issue here is possible injury: the ability of the device to cause injury to a patient or operator following a failure.

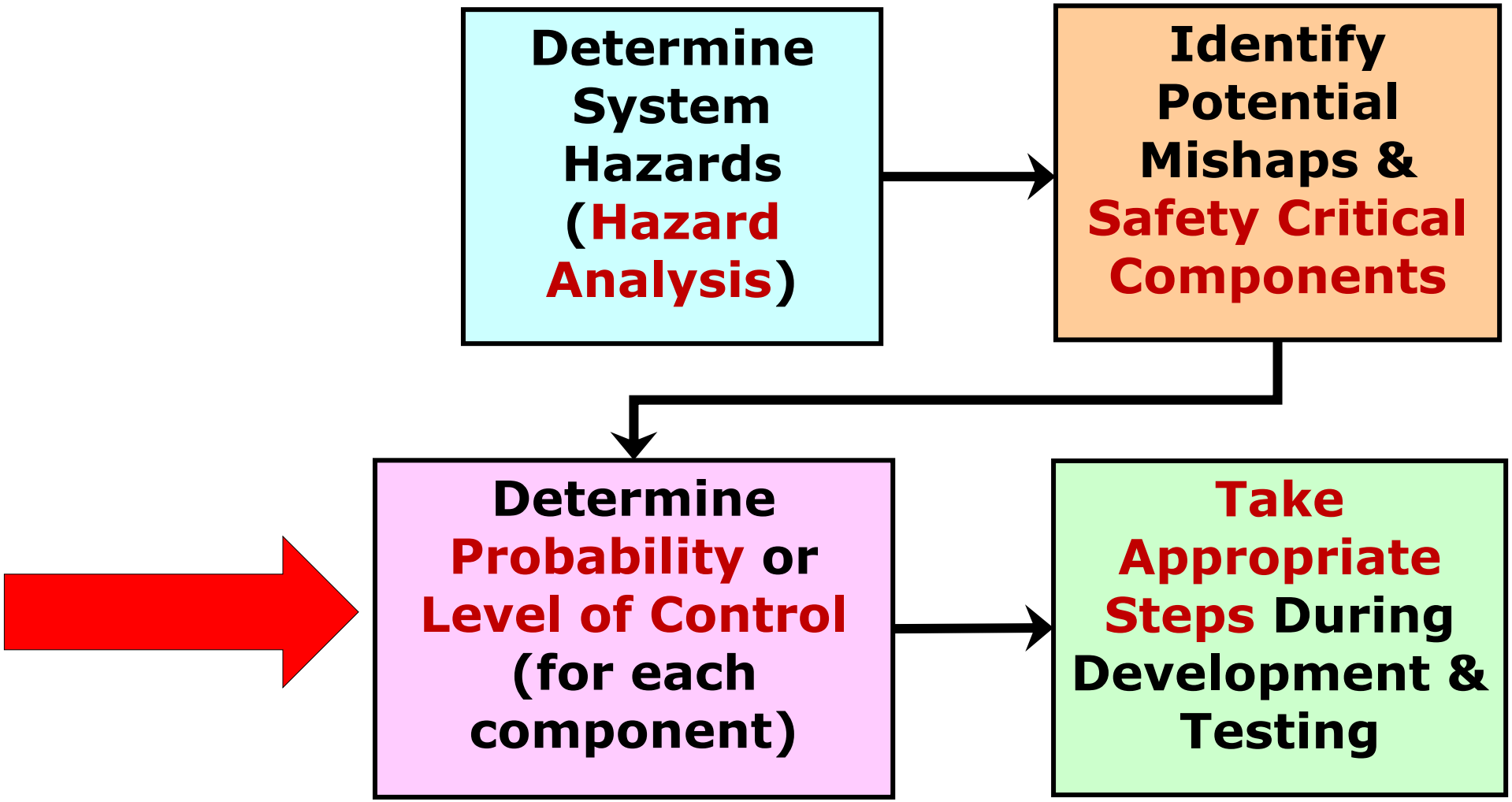
Any potential software contribution to a hazard or mishap requires assessment of severity level.

The issue is *how much damage* could be caused.

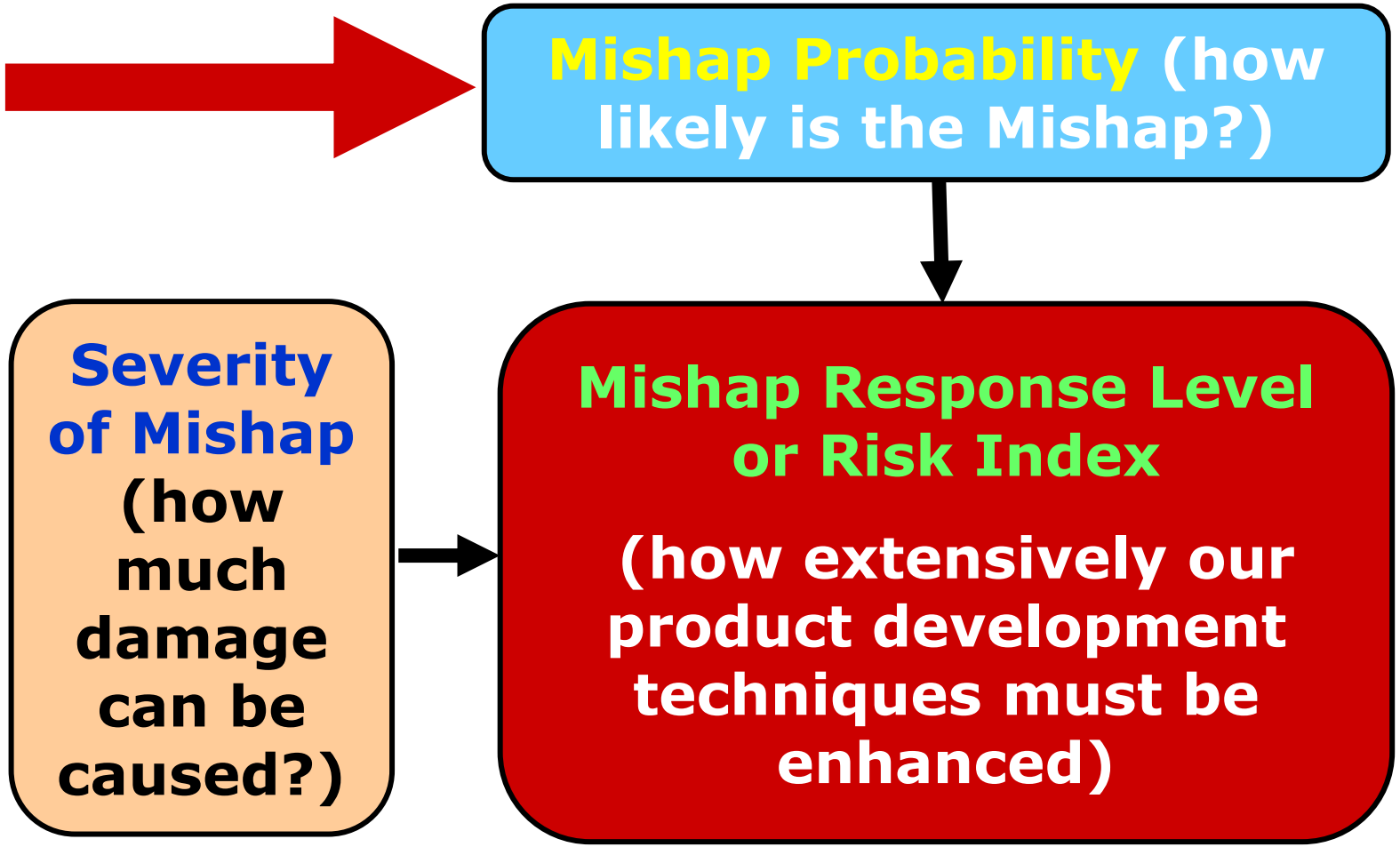
- **Class A – No injury or damage to health is possible**
- **Class B – Non-serious injury is possible**
- **Class C – Death or serious injury is possible**
  - Serious = life threatening, permanent impairment or damage, or requiring surgical intervention to prevent impairment or damage.

Mishap Severity Levels Commonly Used in Nuclear Power, Aircraft 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


Determine System Hazards (Hazard Analysis)
Identify Potential Mishaps & Safety Critical Components
Determine Probability or Level of Control (for each component)
Take Appropriate Steps During Development & Testing

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

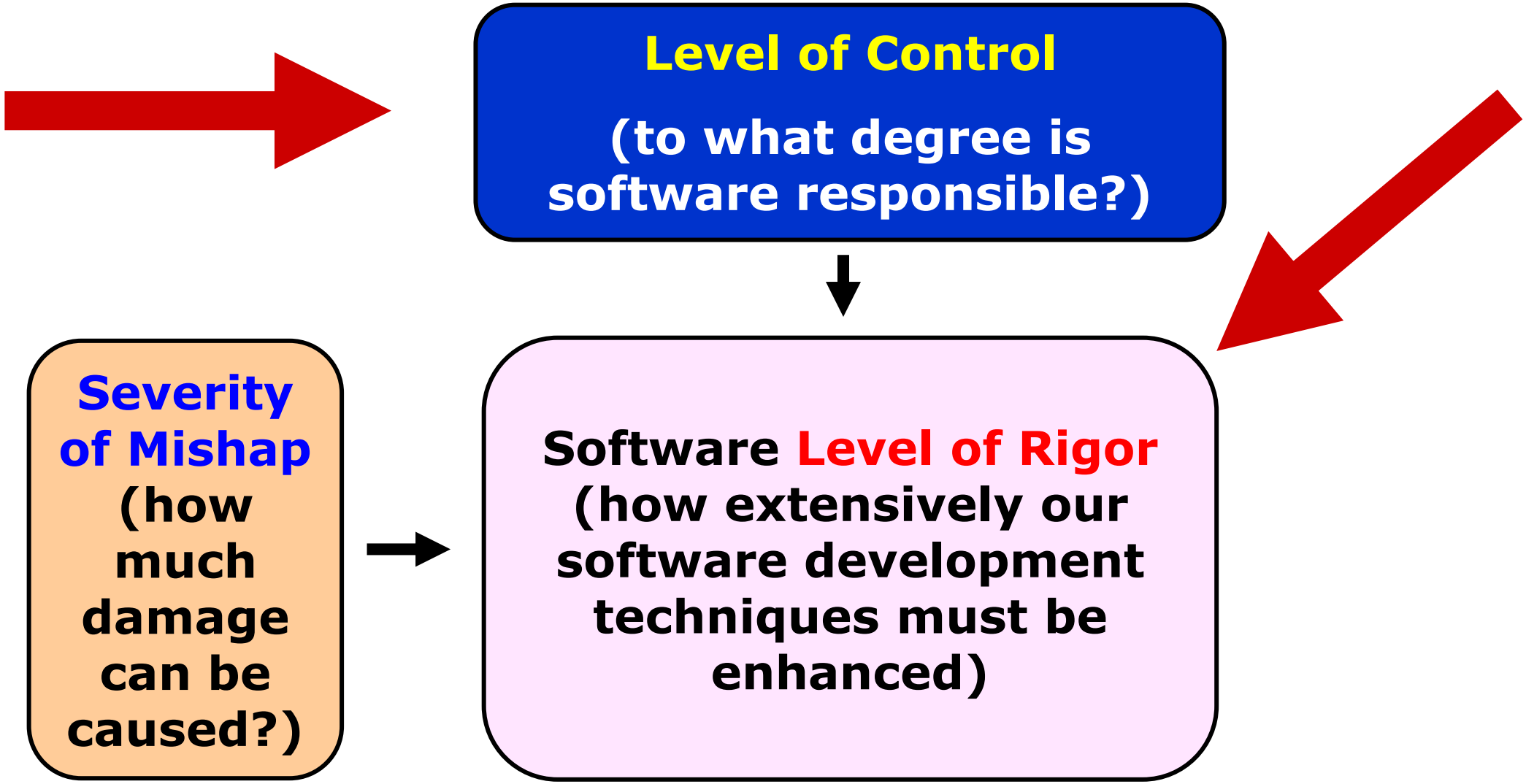
Typical Mishap Risk Index Calculation					
Probability	**A**	**B**	**C**	**D**	**E**
Severity	**Frequent**	**Probable**	**Occasional**	**Remote**	**Improbable**
I - Catastrophic	**U**	**U**	**M**	**M**	**N**
II - Critical	**U**	**U**	**M**	**N**	**N**
III - Marginal	**U**	**M**	**N**	**N**	**N**
IV - 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?)

There's a 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?)

The *Software Level of Control* is a measure of

the degree to which software is responsible for

- **the *behavior* of a system**

  or

- **the behavior 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 (1/6)

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*.

Example: Software Controlled Airbag

An Example of Software Levels of Control (2/6)

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.

Example: Backup Camera

An Example of Software Levels of Control (3/6)

IIb Information Time Critical –

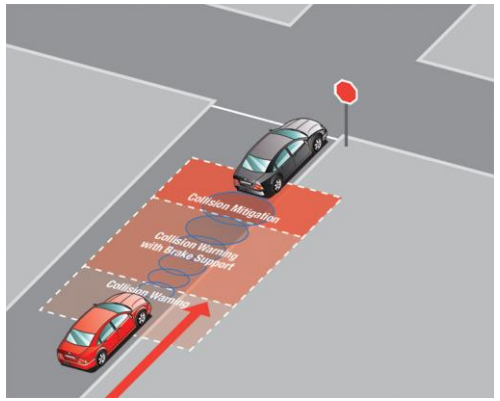
Software displays information

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*.

Example: Collision Warning System



Indyautoblog.com

An Example of Software Levels of Control (4/6)

IIIa Operator Controlled –

Software issues commands

Over a potentially hazardous hardware system, subsystem or component

Requiring human action to complete the control function.

There are several redundant, independent safety measures for each hazardous event.

Example: Air Traffic Control System warns pilot of nearby aircraft

An Example of Software Levels of Control (5/6)

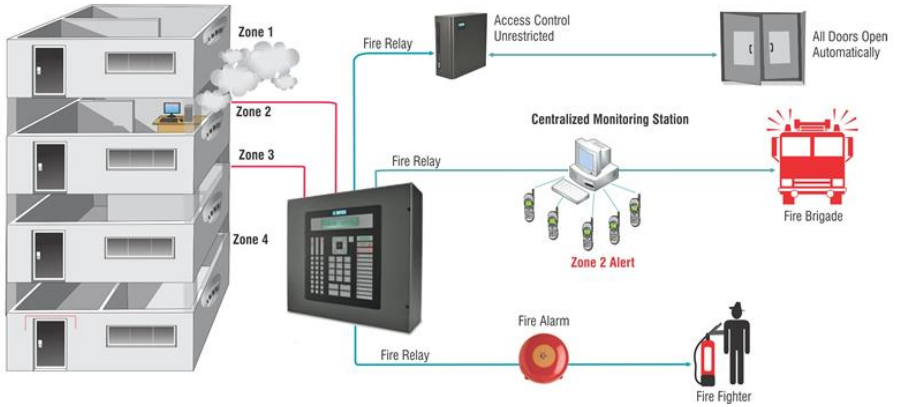
IIIb Information Decision –

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*.

Example: Software in fire alarm system notifies fire department of a fire



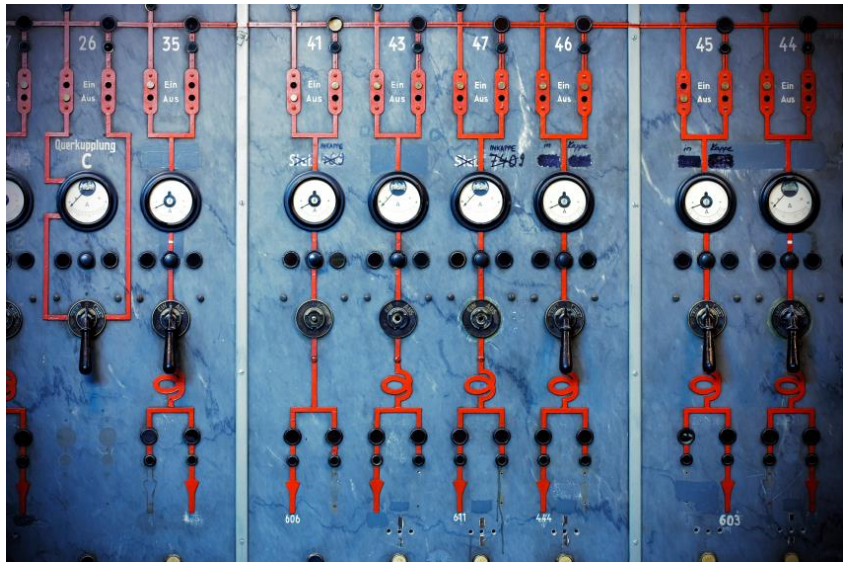
www.itrads.com

An Example of Software Levels of Control (6/6)

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:

I. result directly in a mishap (LOC I),

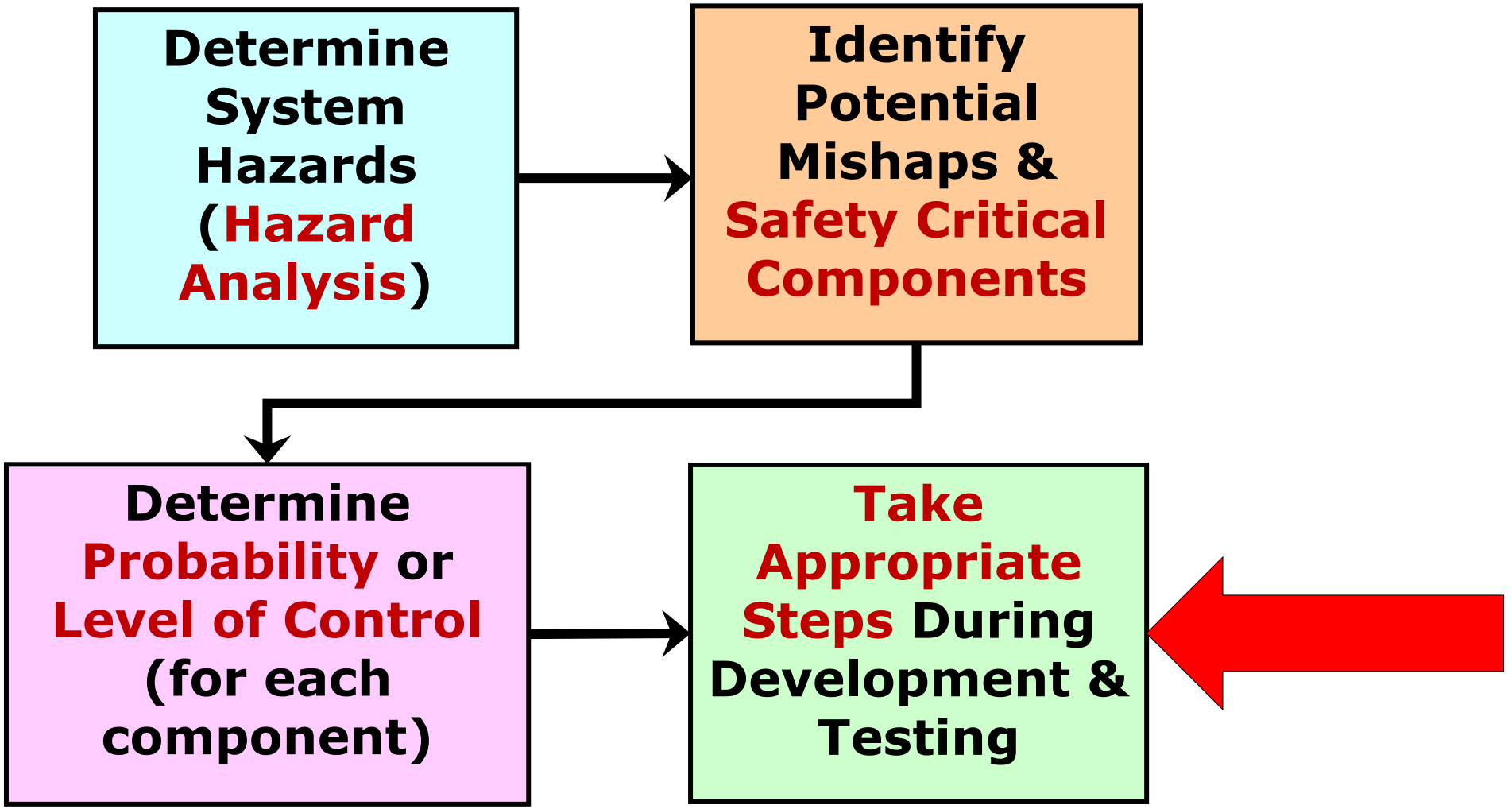
II. significantly impact the margin of safety (LOC IIa or IIb), but not result directly in a mishap, or

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

IV. Have no impact on safety (LOC IV)

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-1	LOR-1	LOR-2	LOR-2	LOR-2	N/A
II - Critical	LOR-1	LOR-1	LOR-2	LOR-2	LOR-2	N/A
III - Marginal	LOR-2	LOR-2	LOR-2	LOR-3	LOR-3	N/A
IV - Negligible	LOR-3	LOR-3	LOR-3	LOR-3	LOR-3	N/A


Determine System Hazards (Hazard Analysis)
Identify Potential Mishaps & Safety Critical Components
Determine Probability or Level of Control (for each component)
Take Appropriate Steps During Development & Testing

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	✓	✓	✓

Example of Appropriate Actions for IEC 62304 - Medical Device Software Standard

Software Documentation Required	Possible Injury from Device		
	Class A No Injury	Class B No Serious Injury	Class C Death or Serious Injury
Software development planning	X	X	X
Software requirements analysis	X	X	X
Software architectural design		X	X
Software detailed design			X
Software unit implementation and verification	X	X	X
Software integration and integration testing		X	X
Software system testing	X	X	X
Software release	X	X	X
X - Required			

Applying Level of Rigor in Automotive Applications

- **Languages and Compilers**
  1. Independently certified compilers with formal syntax and semantics
  2. Restricted subsets of standardized, structured languages with validated compilers
  3. Standardized, structured languages
- **Testing**
  1. 100% testing of all paths through the software at all development phases; semantic static analysis
  2. White box module testing; stress testing; syntactic static analysis
  3. Black box testing

Examples of Language Restrictions for C Language

- **Basic Problems:**
  - C has weak type checking and allows error-prone programs to compile
  - Little or no runtime error handling
- **Restrictions (examples)**
  - No reliance on undefined or unspecified behavior
  - No "commenting out" of any section of code
  - Document all implementation-dependent code
  - No reuse of identifier names in different modules
  - **Char** data type only used for characters
  - All ***typedefs*** should indicate size and signedness rather than using basic types or defaults

Requirements Analysis Phase Safety Actions (examples)

- **Formal definition of requirements with rigorous checking of completeness, consistency, etc.**
- **Safety requirements must be defined**
- **Software Hazard Analysis**
  - Identify hazards associated with different states of the software
- **Hazard Testing**
  - Determine possible faults and specify required actions and response times
  - Develop safety requirements

Design Phase Safety Actions (examples)

- **Formal design methods**
- **Safety critical functions must be isolated from non-safety-critical functions**
- **All safety critical elements must be identified as "safety critical"**
- **All safety critical functions must provide:**
  - Error handling
  - Fault containment (if fault occurs, it is detected and harm is limited)
  - Proper sequencing of safety critical commands
- **Termination of the software must result in a safe system state.**

Design Phase Safety Actions (continued)

- **Interfaces must be analyzed:**
  - Interdependent interfaces must be identified and all possible interactions must be analyzed
  - Between-component interactions are more prone to safety problems than component failures
- **Protocols must be analyzed:**
  - To assure that things like frequencies, location of parity bits, encoding algorithms, etc. are consistent
  - Deadlock situations must be prevented
- **Human factors analysis**
  - Human capabilities and limitations must be assessed

Coding Phase Safety Actions (examples)

- **Safe subsets of programming languages must be identified and coding standards must forbid use of features not in the safe subsets**
  - Unsafe features include uninitiated variables, dynamic memory allocation and de-allocation, side-effects of function calls on operands, transfer of control to arbitrary locations, …
- **Range checking must be used**
- **Code logic analysis**
- **…**

Testing Phase Safety Actions (examples)

- **Independent testing by individuals not connected with development**
- **Failure modes and effects testing**
- **Safety-critical interface testing**
- **Testing for possible failures, not just testing to find all bugs**
- **Root cause analysis of failures**
  - ***Forward analysis*** – what other failures might we have missed from the same root cause?

Other Software Safety Actions (examples)

- Physical or logical partitioning, to keep safety critical software separate from non-safety-critical software
- Tracing from requirements to design to code to tests
- Redundancy in design / backup software for critical functions
- Detection and recovery from hardware failures
- …

Part 2 Agenda

- What we can Do About Software Safety

- **System Control Analysis**

- Some Software-specific Issues

- Research Reports

Reminder

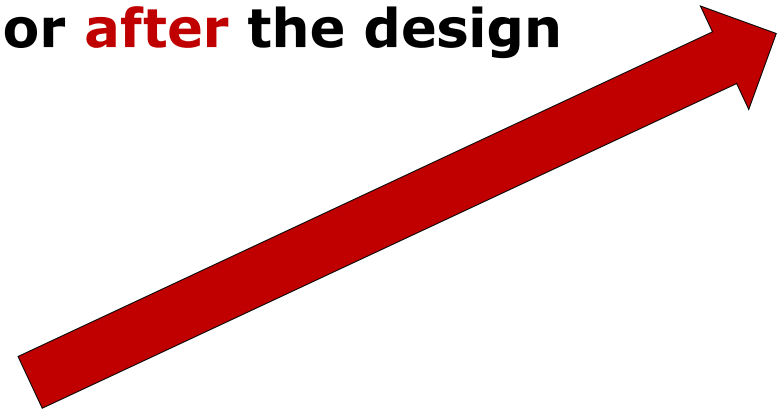
Hazard Analysis Approaches

Traditional - treat it as a reliability issue

- Identify the components that may fail and contribute to a hazard
- Use various approaches to make the failures less likely or less harmful
- Much of the analysis occurs during or after the design step

Emerging - treat it as a system control issue

- Identify the ways the system may fail and contribute to a hazard
- Create system requirements and constraints that forbid these conditions
- The analysis occurs before the design is started



System Control Analysis –
The Emerging Method of Analyzing Hazards

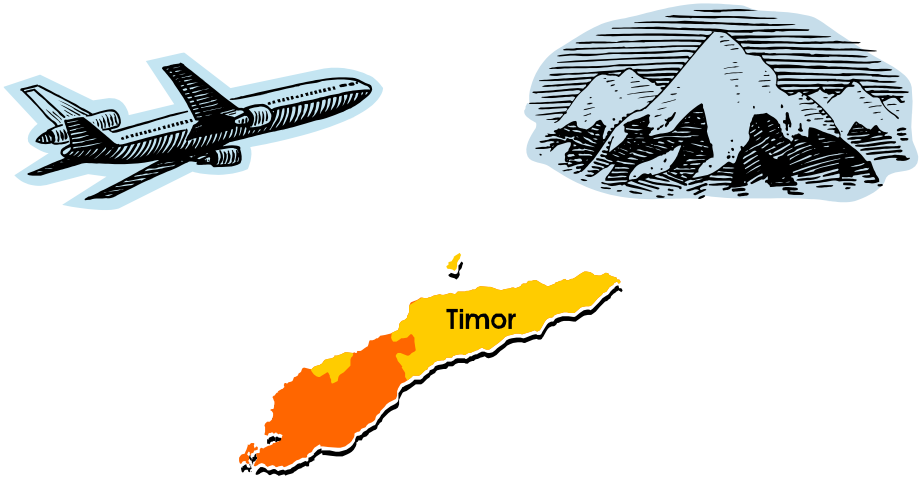
Look for Ways the System can Fail

For Reasons Other than Component Failure

Turn That Knowledge into Requirements and Constraints

[includes rigorous requirements specification and analysis]

Example of System Failures where no component fails



Airplane crashes into a mountain

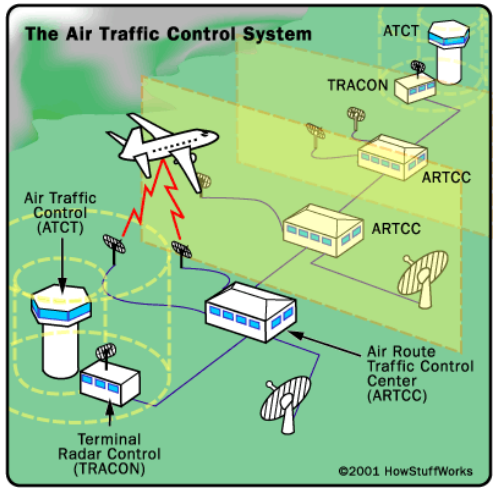
- Pilot is listening for air traffic control instructions
- Two nearby controllers in different countries give **conflicting instructions**
- Both control stations have ***similar sounding names***, so the pilot confuses which one he is listening to

Some Kinds of System Failures where no component fails (1/3)

System is too complex for human operator to operate

- Human error is often a symptom of a system design problem

- **Example**
  - **"NextGen air traffic control system is too complex for pilots to understand and operate safely and effectively."**



Source: flightdeck.ie.orst.edu/NextGen

Some Kinds of System Failures where no component fails (2/3)

Components interact in a manner not anticipated in the system design

- When components are developed independently, not all interactions are anticipated

- **Example**
  - **When I select "connect bluetooth device" on my car's audio system, the backup camera fails to work, even though I am backing up**

Some Kinds of System Failures where no component fails (3/3)

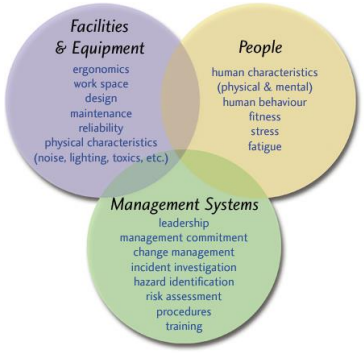
Organizations assume they can make small changes to "safe" systems

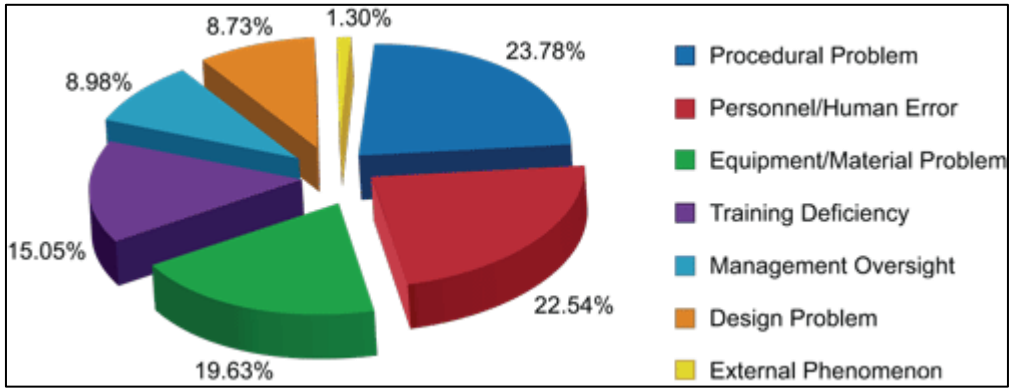
- Any change may introduce error
- Systems migrate to high risk

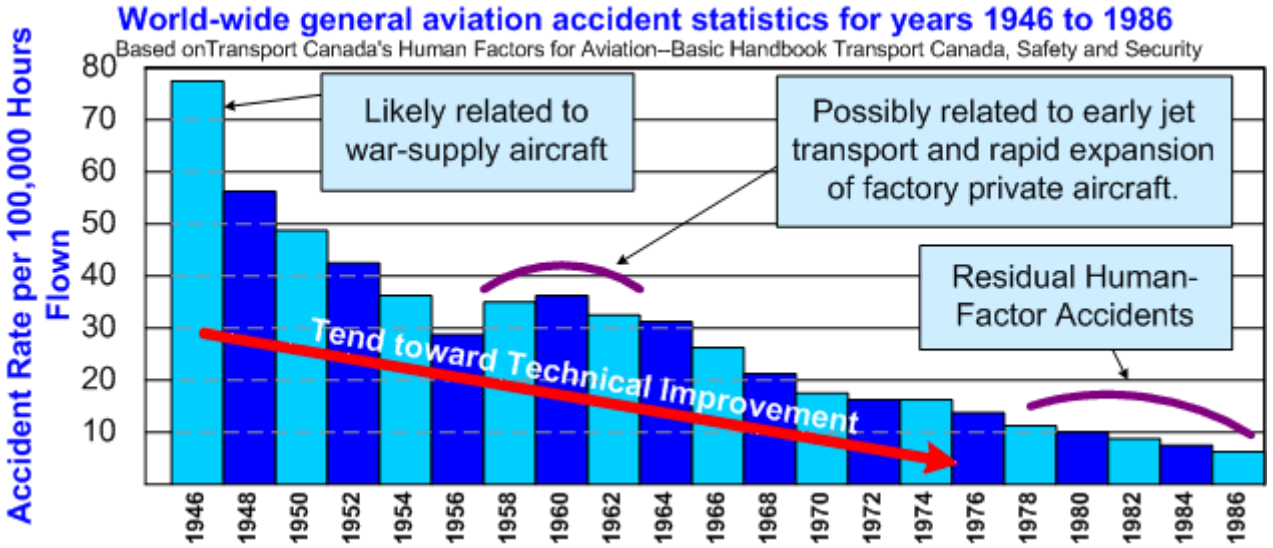
➢ **Example**
  - **Australian helicopter pilot training system**

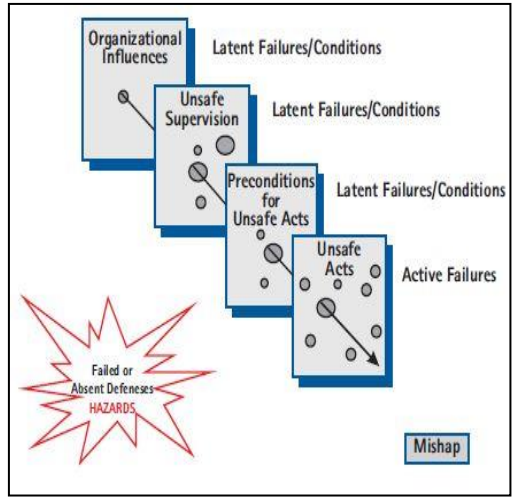
Dealing with These Issues

Human Factors Analysis is Vital









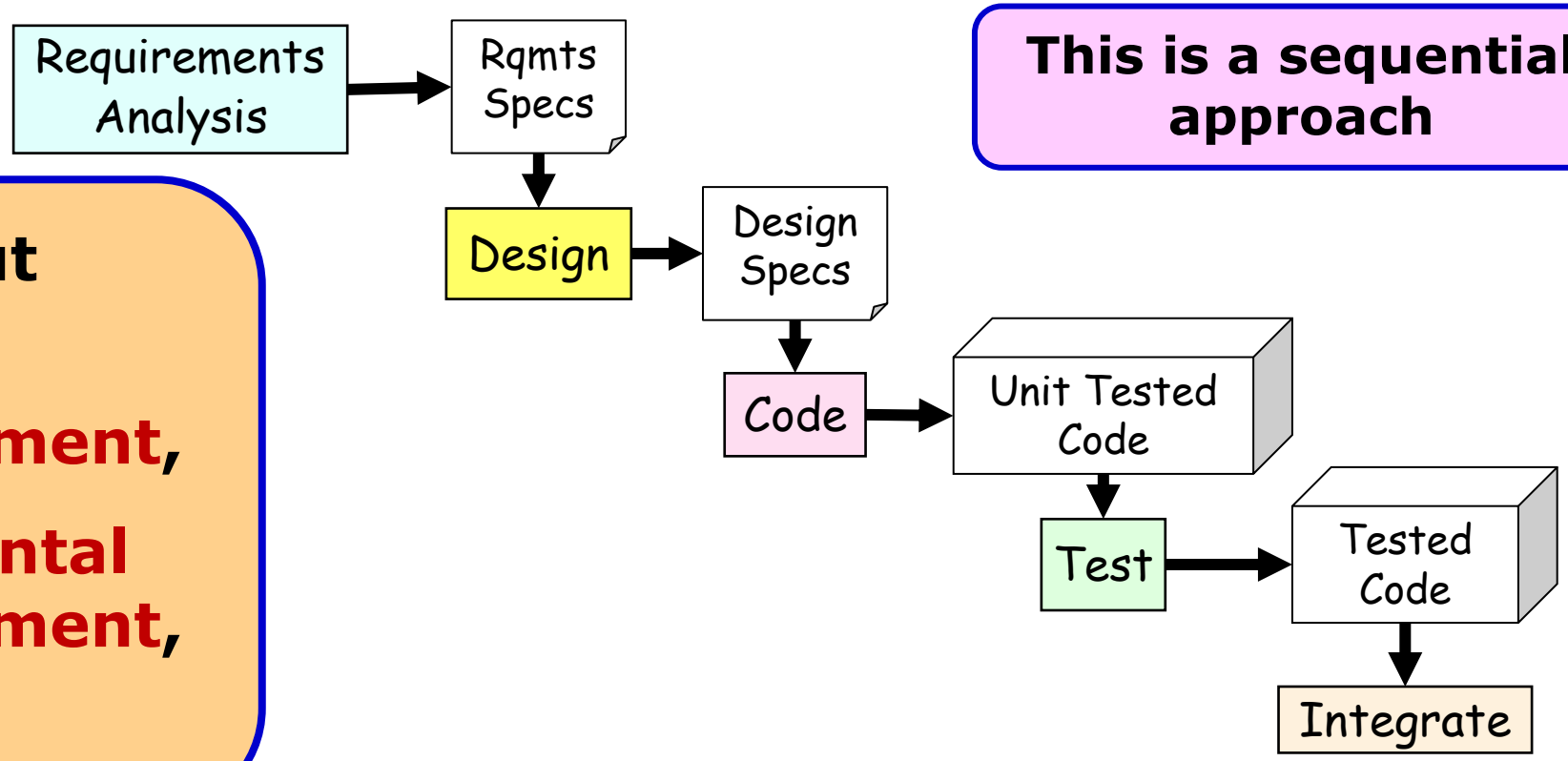
Part 2 Agenda

- What we can Do About Software Safety
- System Control Analysis

➢ **Some Software-specific Issues**

A Software Issue …

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



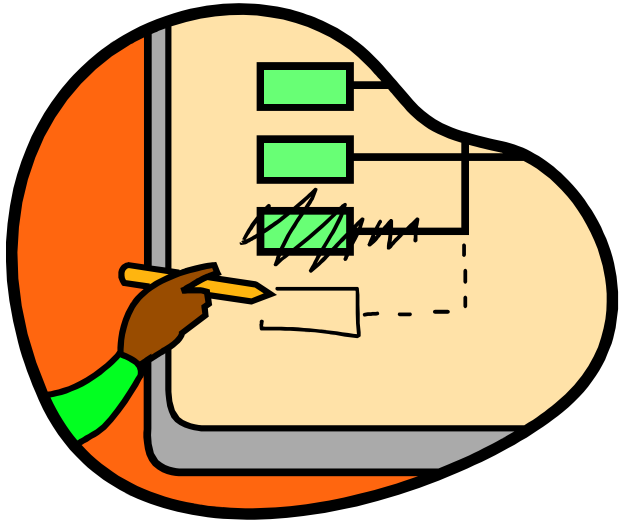
This is a sequential approach

What about

- **Agile Development,**
- **Incremental Development,**
- **etc.?**

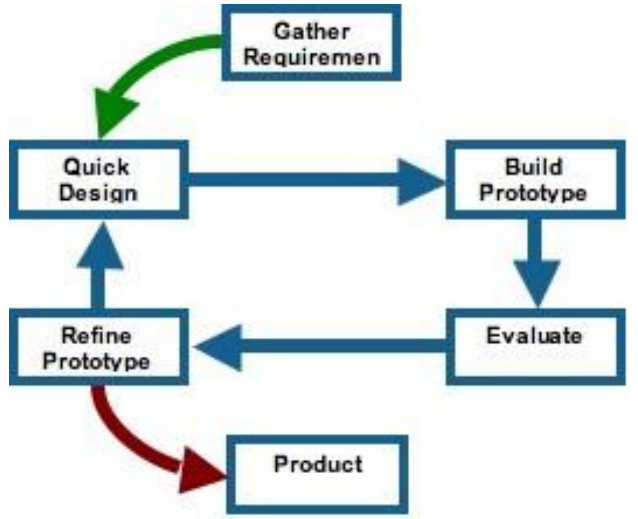
Challenges for Agile and Incremental Development

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



SW Safety Strategy for Agile & Incremental Development (1/3)

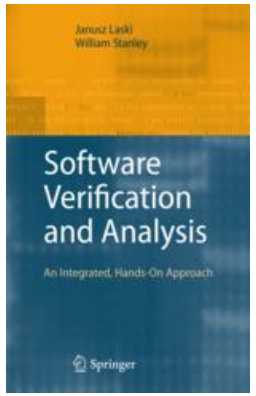
1. **Use agile/incremental development for experimental and prototype functionality**
   - I.e, use only for developing prototypes
2. **Then develop the final product in a more formal, sequential manner, including formal verification**
   - Partitioning the system to segregate prototype/experimental functionality from mainstream functionality



SW Safety Strategy for Agile & Incremental Development (2/3)

3. **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

4. **Lift restrictions when functions have successfully exited formal development and verification**



SW Safety Strategy for Agile & Incremental Development (3/3)

5. **Consider the economics of developing a robust, scaled-down version of a function for safety critical applications**

6. **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

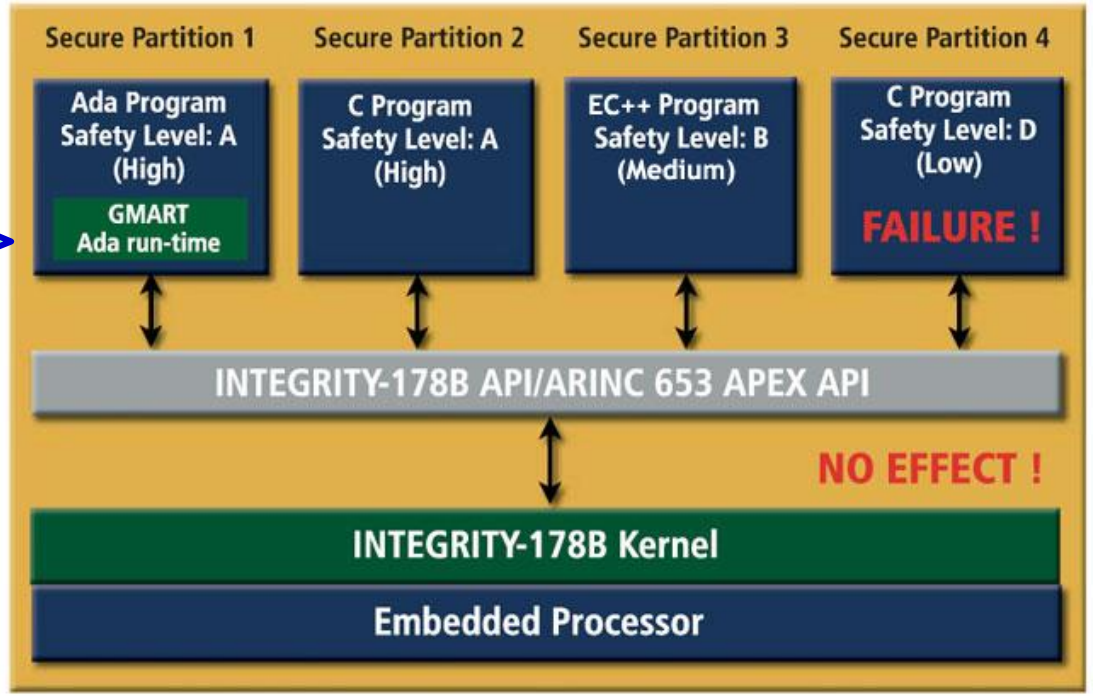
- Operating systems (VXworks™, Windows™)
- "standard" I/O drivers or interfaces
- Board support packages

- **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 (1 of 2)

- **Safety Certification – there are organizations that will certify the safety or security of a software package, such as an operating system.**

Partitioning to separate safety critical from non-safety critical functions.



Green Hills SOFTWARE

INTEGRITY Real-Time Operating System

www.ghs.com/products/rtos/integrity.html

Techniques for Safety Critical COTS Software (2 of 2)

- **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** – limit use of COTS software to limited-complexity applications where reliability can be readily and accurately assessed.
- **Redesign** - to eliminate COTS from the design.

"Free" and Open Source Software

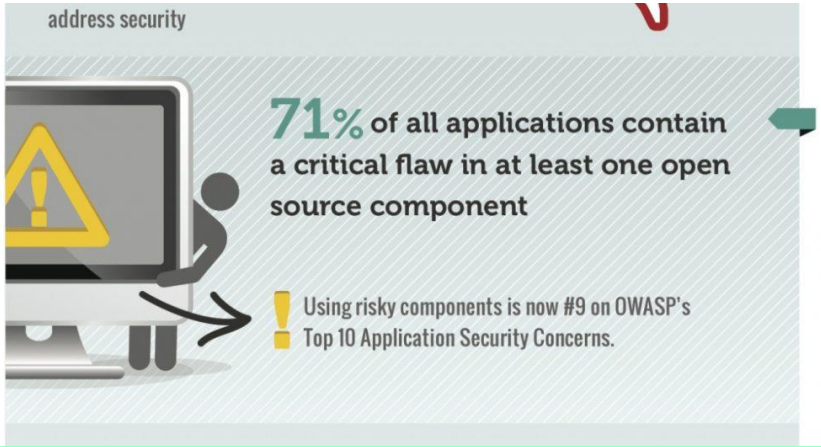
Why Open Source Software (OSS)

More about "Free" Software

- **FOSS examples**
  - Many components of a GUI
  - Compilers
  - Many popular applications such as Firefox, Open Office, Linux, Java. Ubuntu
- **FOSS has certain advantages over COTS**
  - You are usually able to see the source code
  - Most FOSS has been extensively used and tested
  - In some cases, you may have access to design documentation

Challenges for "Free" Software

- **FOSS has these additional challenges beyond COTS:**
  - You may be legally required to report changes back to the originator or the user community
  - The code may change over time with little notification or configuration control
  - You don't know who wrote the code or what mistakes or traps they might have embedded in the code



Source: best-infographics.com

- **In other words, there is less control and potential risk**

SOUP
Software of Unknown Provenance

IEC 62304

- **Suppose you have software that was not originally developed for your purposes**
- **Or which was developed by unknown sources, using unknown standards, processes and procedures**
- **IEC 62304 requires these additional steps:**
  - Configuration management of all such software
  - Extract and specify the functional and performance requirements
  - Specify the system hardware and software required
  - Include assessment in the software risk management process
  - Thoroughly evaluate all anomalies or unexpected results

Best Practices for FOSS, COTS, or SOUP

- **Strict requirements on supplier of software**
  - Must provide information on all known bugs
  - Must allow audits of processes and procedures
  - (others as deemed appropriate)

  } **May be deemed unacceptable by many suppliers.**

- **Care in software licensing practices**
  - Establish licensing procedures within buyer's organization
  - Object to undesirable licensing provisions
- **Keep software under YOUR configuration control**
  - In other words, you make all changes to it

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 not always 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**
- **…**

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
In preparation, the software research & development communities	→	must improve methods of dealing with software safety

Some Useful Reference Material (1/3)

- DACS, *The DACS Software Reliability Sourcebook,* Data & Analysis Center for Software
- EIA SEB6-A – *System Safety Engineering in Software Development*
- (U.S.) FAA *System Safety Handbook*, Appendix J: Software Safety
- (U.S.) FDA Center for Devices and Radiological Health - *Premarket Notification Submissions (510k), "General Principles of Software Validation"*
- Herrman, Debra S. *"Software Safety and Reliability"*
- Holzmann, Gerard J., *The Power of 10: Rules for Developing Safety-Critical Code*, IEEE Computer, June, 2006.
- IEC 62304 – *Medical Device Software – Software Life Cycle Processes (2015)*

Some Useful Reference Material (2/3)

- **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: 978-0201419725**
- **Leveson, Nancy G., *Engineering a Safer World: Systems Thinking Applied to Safety*, MIT Press, 2014, ISBN-13: 978-0262017629**
- **MIL-STD-882E – *Standard Practice for System Safety* – US Department of Defense**
- **NASA-STD-8719.13C – *Software Safety***

Some Useful Reference Material (3/3)

- Patrick R.H. Place & Kyo C. Kang, ***Safety-Critical Software: Status Report and Annotated Bibliography***, Software Engineering Institute, June 1993
- RTCA, Inc., DO-178C, ***Software Considerations in Airborne Systems and Equipment Certification*** (This is the commercial aviation standard)
- RTCA, Inc., DO-248B, ***Final report for Clarification of DO-178B***
- Society of Automotive Engineers - ***JA 1002 Software Safety and Reliability Program Standard***

Part 2 Agenda

- What we can Do About Software Safety
- System Control Analysis
- Some Software-specific Issues

➢ **Research Reports**

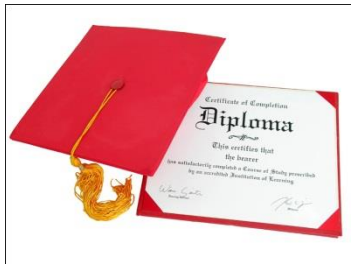
End of Part 2

Appendix A

Notes on Certification and Licensing

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 legal authorization 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?

- **Great Britain, Australia, & Canada have systems in place for licensing software engineers**
- **Other countries are considering similar systems**
- **The US National Society of Professional Engineers (NSPE) briefly had a way for software engineers to be licensed**
  - About 10 states instituted SW engineering licensing and others considered it
  - But this was to be discontinued in 2010 due to limited participation. Perhaps it was premature in the US.
- **The exam could be resurrected in the future**

Software Engineering Licensing – Must You Be Licensed?

- **Licensing is required in some countries for some kinds of software applications**
  - I have not kept up with what is required where
- **If licensing of software developers is ever required in the US, it will likely be for safety critical and security critical software applications,**
  - such as control of power plants, medical applications, etc.
- **As more and more of our lives become dependent on software, don't be surprised if politicians start requiring some sort of certification or licensing for safety critical software developers or even in other fields, such as financial and transportation applications.**