Software Safety

18-849b Dependable Embedded Systems

Michael Scheinholtz

March 2nd, 1999

Required Reading: High-Pressure Steam Engines and Computer Software, by Nancy Leveson

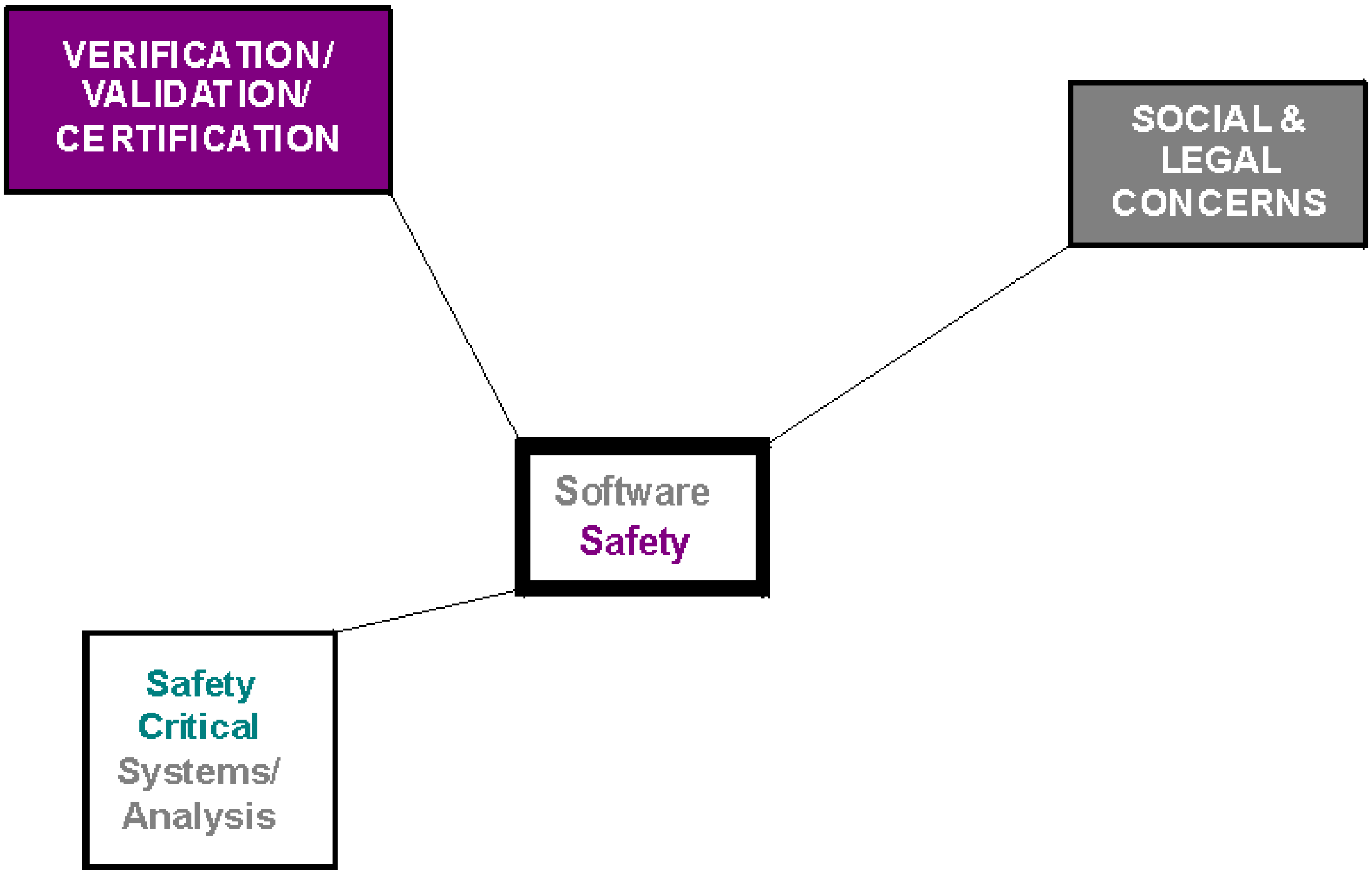
Best Tutorial: Chapter 8 of Safeware by Nancy Leveson

Carnegie Mellon

Overview: Software Safety

- **Introduction**
  - More and more hazardous systems are being controlled by software.
- **Key concepts**
  - Safety is an emergent system property.
  - How computers cause accidents.
  - Software design and system design.
- **Tools / techniques / metrics**
  - System safety techniques can be applied to software.
  - Validation of software safety.
- **Relationship to other topics**
  - Any topic dealing with software in a potentially dangerous system.
- **Conclusions & future work**

YOU ARE HERE MAP



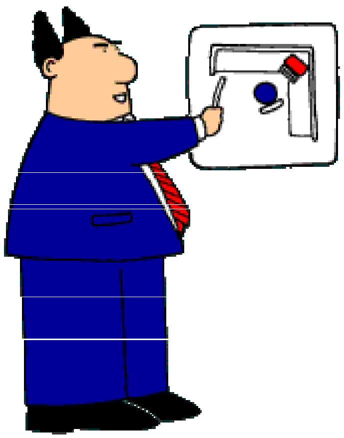
Description of Topic

- **Software Safety**
  - Ensuring the software will execute within a system context without resulting in unacceptable risk.

- **List Key Concepts**
  - Reliable software does not mean safe software
    - Specification errors can lead to unsafe system states
    - Poor user interface design can hurt operators understanding of "correct" software
  - Safety must be designed in from the beginning
    - Safety is similar to security. It is hard to retrofit.
  - Software fails differently than hardware or mechanical system
    - Mechanical systems are continuos, software systems are discrete.

Safety is an emergent system property

- **System safety looks at the whole system, not just its components**
  - Must involve software people in the analysis
  - Make sure component interactions are safe.
  - No single component is responsible fore safety.
- **Perfect software may not produce safe systems**
  - Difference between safe and reliable.
  - Most software errors come from requirement errors.
- **Safety concerns go beyond engineering**
  - Management must buy into safety.
  - Operators must be well trained.
  - A strong safety culture in needed.

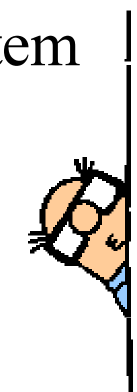


How Computers Cause Accidents

- **Problems validating and understanding software**
  - Machines exhibit continuous behavior.
  - Software is discrete.
  - Methods of validating software are not as rigorous as those for mechanical systems.
- **The lure of flexibility**
  - Can't we just add that one extra feature? Sure! Its only software!
- **Requirements errors and misunderstandings**
  - Should that valve be open or closed when the power goes off?
- **Operator issues**
  - Software should make the operators job easier.

Software Safety and System Safety

- **Safety should be part of software design from the start**
  - Much cheaper to eliminate hazards.
  - Modifying software late in the development cycle can cause more bugs.
- **Software engineers must not isolate software design from system design**
  - Software should not be a black box.
  - Software engineers should have some knowledge of the system larger system they are helping to build.



- **Communication is key**
  - Keep safety requirements updated.

Tools / Techniques

- **Software cannot realistically be made bug free.**
  - Formal methods help some.
  - Software testing helps some too.
  - Still depends on perfect requirements
- **Software can't really be made fault tolerant…**
  - N-version programming… etc.
- **Applying system safety methods to software**
  - You can apply system safety methods to Software to make it safer.
  - Safety Critical systems analysis can find behaviors that lead to unsafe system states.
  - Find the functions that are safety critical and validate them as much as possible.
  - Hardware interlocks aren't so bad.

Metrics

- **Verification is difficult**
  - Software should be designed to be verifiable.
  - Keep software small and simple
  - Separate safety critical functions from non-safety critical functions to minimize the amount of software to verify.
- **Dynamic analysis**
  - Execute the software and check it.
  - Check all of the safety features.
  - Can catch missed requirements.
- **Static analysis**
  - Formal verification, Software Fault Tree analysis
  - Static safety analysis very similar to a structured code walkthrough.

Relationship To Other Topic Areas

- **Verification, Validation, and Certification**
  - How can you tell if a piece of software is "safe"?
- **Safety Critical Systems Analysis**
  - A key part of software safety is getting the requirements correct.
- **Social and Legal Concerns**
  - How much safety can you afford?

Conclusions & Future Work

- **Correct software is hard; safe software is harder**
  - Even perfect software is not necessarily safe.
  - Requirements analysis is a major factor in getting safe software.
  - Don’t depend on software. Keep hardware checks.
  - Complexity is the enemy.
- **Safety is a system wide process, software must be an integral part**
  - Software engineers must be aware of what is outside the box.
  - Systems engineers should have an understanding of the software.
  - Everyone must buy into safety.
- **There is no way to measure safety, so the options are:**
  - Use good process, with structured safety reviews.
  - Hope best practice is good enough.

Steam Engines and Computer Software

- **"Exploding Software?"**
- **Major points**
  - Computers and software are profitable.
  - Computers are not as well understood as the devices they replace.
  - Are we putting too much faith in fledgling software engineering?
- **Some of Leveson's conclusions**
  - Need more rigorous understanding of software and the humans that build it.
  - Do we need strict regulation of safety critical software?