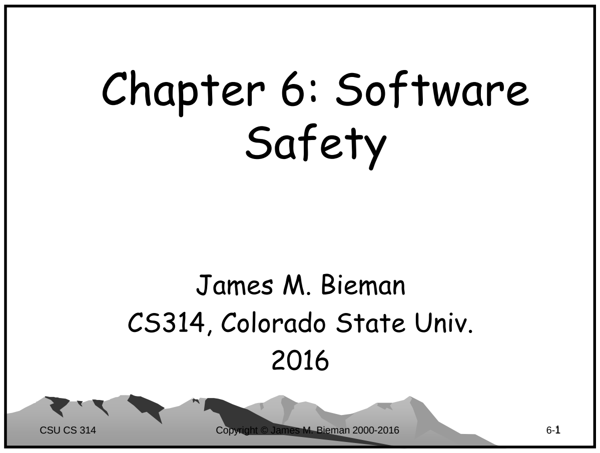
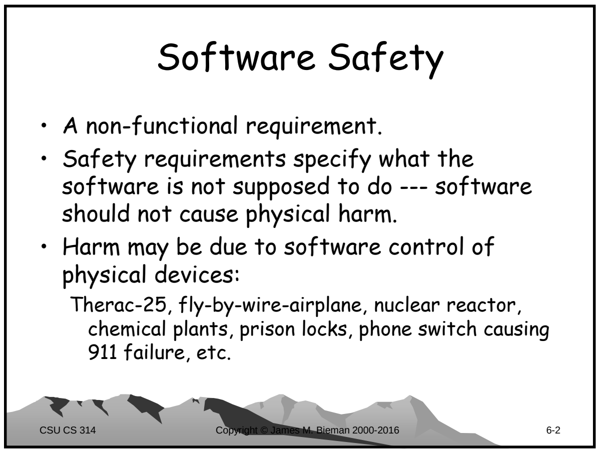


Chapter 6: Software Safety

James M. Bieman
CS314, Colorado State Univ.
2016

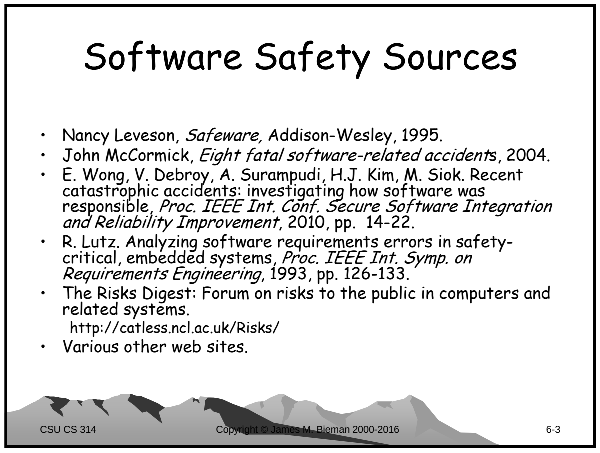
CSU CS 314 Copyright © James M. Bieman 2000-2016 6-1



Software Safety

- A non-functional requirement.
- Safety requirements specify what the software is not supposed to do --- software should not cause physical harm.
- Harm may be due to software control of physical devices:
  Therac-25, fly-by-wire-airplane, nuclear reactor, chemical plants, prison locks, phone switch causing 911 failure, etc.

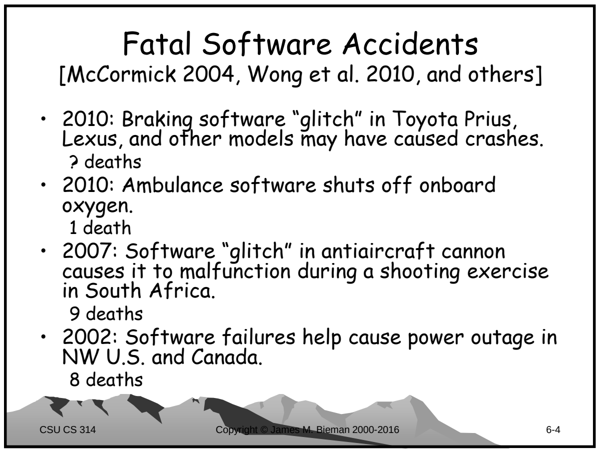
CSU CS 314 Copyright © James M. Bieman 2000-2016 6-2



Software Safety Sources

- Nancy Leveson, *Safeware,* Addison-Wesley, 1995.
- John McCormick, *Eight fatal software-related accidents*, 2004.
- E. Wong, V. Debroy, A. Surampudi, H.J. Kim, M. Siok. Recent catastrophic accidents: investigating how software was responsible, *Proc. IEEE Int. Conf. Secure Software Integration and Reliability Improvement*, 2010, pp. 14-22.
- R. Lutz. Analyzing software requirements errors in safety-critical, embedded systems, *Proc. IEEE Int. Symp. on Requirements Engineering*, 1993, pp. 126-133.
- The Risks Digest: Forum on risks to the public in computers and related systems.
  http://catless.ncl.ac.uk/Risks/
- Various other web sites.

CSU CS 314 Copyright © James M. Bieman 2000-2016 6-3

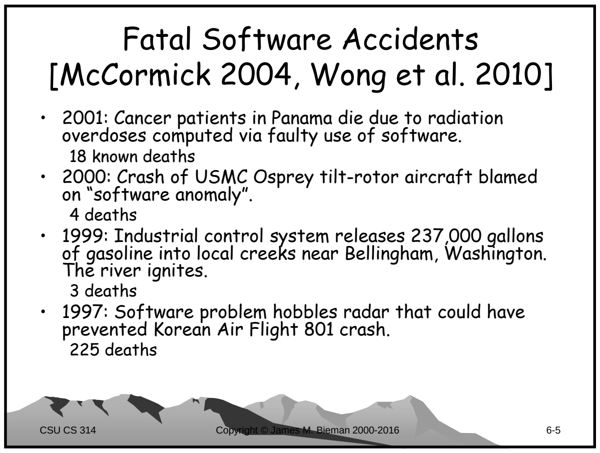


Fatal Software Accidents

[McCormick 2004, Wong et al. 2010, and others]

- 2010: Braking software "glitch" in Toyota Prius, Lexus, and other models may have caused crashes.
  ? deaths
- 2010: Ambulance software shuts off onboard oxygen.
  1 death
- 2007: Software "glitch" in antiaircraft cannon causes it to malfunction during a shooting exercise in South Africa.
  9 deaths
- 2002: Software failures help cause power outage in NW U.S. and Canada.
  8 deaths

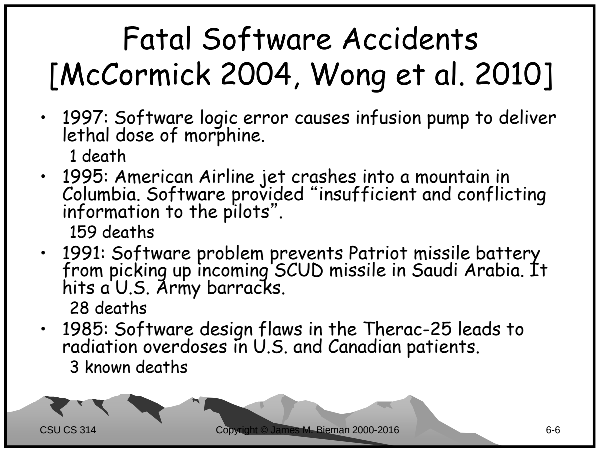
CSU CS 314 Copyright © James M. Bieman 2000-2016 6-4



Fatal Software Accidents [McCormick 2004, Wong et al. 2010]

- 2001: Cancer patients in Panama die due to radiation overdoses computed via faulty use of software.
  18 known deaths
- 2000: Crash of USMC Osprey tilt-rotor aircraft blamed on "software anomaly".
  4 deaths
- 1999: Industrial control system releases 237,000 gallons of gasoline into local creeks near Bellingham, Washington. The river ignites.
  3 deaths
- 1997: Software problem hobbles radar that could have prevented Korean Air Flight 801 crash.
  225 deaths

CSU CS 314 Copyright © James M. Bieman 2000-2016 6-5



Fatal Software Accidents [McCormick 2004, Wong et al. 2010]

- 1997: Software logic error causes infusion pump to deliver lethal dose of morphine.
  1 death
- 1995: American Airline jet crashes into a mountain in Columbia. Software provided "insufficient and conflicting information to the pilots".
  159 deaths
- 1991: Software problem prevents Patriot missile battery from picking up incoming SCUD missile in Saudi Arabia. It hits a U.S. Army barracks.
  28 deaths
- 1985: Software design flaws in the Therac-25 leads to radiation overdoses in U.S. and Canadian patients.
  3 known deaths

CSU CS 314 Copyright © James M. Bieman 2000-2016 6-6

Safety Risks in Software

- Generally in software that controls physical entity that can cause harm.
- Embedded systems
- Medical applications.
  - Software controlled treatment.
    - Therac-25
    - Surgical equipment: robots, scalpels, …
    - Pacemakers.
    - Diabetes insulin pumps.
    - Patient monitoring.
  - Diagnoses support:
    - Automatic x-ray interpretation.
    - Expert systems
  - Patient records

  Need for continuous availability.

To lower risks, knowledge of the entire application domain is critical.

Safety Myths

1. The cost of computers is lower than that of analog or mechanical devices.

   Software can be more expensive than hardware, especially when we include maintenances costs. The on-board Space Shuttle software is about 300K words and has an annual budget of $100M for software maintenance.
2. Software is easy to change.
3. Software provides greater reliability than hardware.

   Software does not wear out, but design errors are common.

Safety Myths (cont)

4. Increasing software reliability increases safety.

   Only a partial overlap between reliability and safety
5. Testing or proving correctness can remove all errors.
6. Reuse increases safety.

   Reuse outside of the original domain can lead to trouble. The Therac-20 software was used in the Therac-25. Aviation software can have problems in the Southern hemisphere.
7. Software reduces risk over mechanical systems due to better control.

   Safety margins may be cut, resulting in no safety gain.

Do Human Operators Cause Most Accidents?

- Were the operators responsible for the Tyler Therac-25 accidents?
- Human operators are often blamed for accidents.
  - “85% of unsafe work accidents are caused by unsafe acts by humans rather than unsafe conditions".
  - Such data may be biased and incomplete and often is based on reports of supervisors.
  - It can be convenient to blame the operators rather than the system design.

Example: 1979 DC-10 Crash

- In 1979, a DC-10 crashed into Mt. Erebus in Antarctica.
- The inquiry blamed the pilot. However,
  - the autopilot had been altered by other employees just before flight, and the pilot was not notified.
  - The pilot was blamed for flying too low. However, he flew low due to specific management instructions to fly low to improve sightseeing.

Example: False Nuclear Attack

- A false warning of a Soviet nuclear attack in 1979 was blamed on a "simple operator error".

  An operator mistakenly inserted a training tape that simulated an attack into the warning system in Cheyenne Mountain.
- However, at the time,
  - NORAD was deploying an upgraded system.
  - During deployment, some of the software development and testing was done on the on-line NORAD network, because no other computer system was available.

Example: Three Mile Island

- Operators were blamed for throttling back on 2 high-pressure injection pumps to decrease water pressure, thus allowing the core to become uncovered and overheat.
- However, unless the operator knows that there was a loss of coolant,
  - the standard practice was to throttle back to avoid other kinds of damage.
  - An indicator that would suggest a lack of coolant was on the back side of the control panel.
  - Also, operators were used to seeing faulty readings.

Consider the Situation at TMI

- 110 alarms were sounding;
- Key indicators were inaccessible and/or malfunctioning;
- Repair order tags covered warning lights;
- The data printout was running one or more hours behind;
- The room was filling with experts; and so on.

Blaming the operators seems like a way out.

Reported Poor User-interface Designs in Nuclear Power Plants:

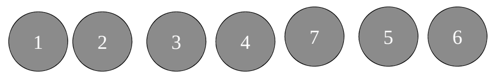
- Dials measuring the same quantities are calibrated using different scales.
- Normal ranges are not uniformly marked.
- Recorders are cluttered with excess information.
- Labels and colors are inconsistent and/or confusing.
- Left-hand displays are driven by right-hand controls.
- Meters cannot be read from a distance, but controls are far away.

Reported poor user-interface designs in nuclear power plants 2:

- Key displays are on the back of the console, while unimportant displays are on the front.
- Two identical (unmarked) scales are side-by-side. One differs from the other by a factor of 10.
- Labels on alarms differ from corresponding labels in the written procedures.
- Training control board is laid out differently from the actual control board.

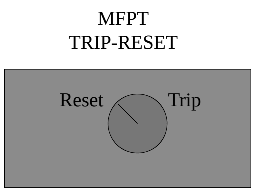
Examples of Poor Designs Seen in Real Control Rooms

Labeling on Pumps [Leveson 1995]



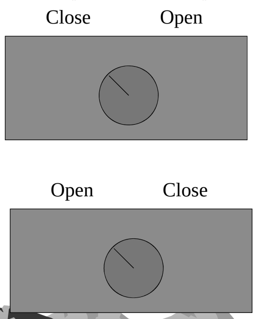
Reversal of Trip-Reset Positions

[Leveson 1995]



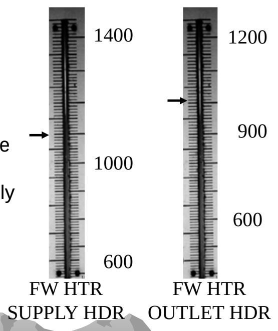
Another Inconsistency

[Leveson 1995]

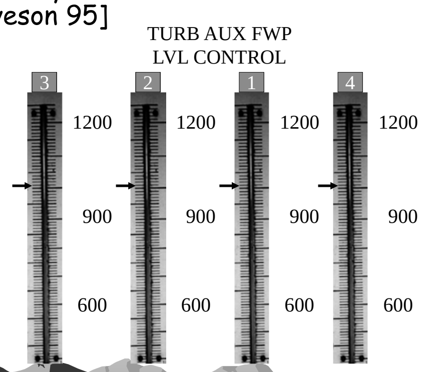


Heater Pressure Gauges [Leveson 95]

A hurried operator might believe that the outlet pressure is higher than the supply



A Strange Way To Count [Leveson 95]

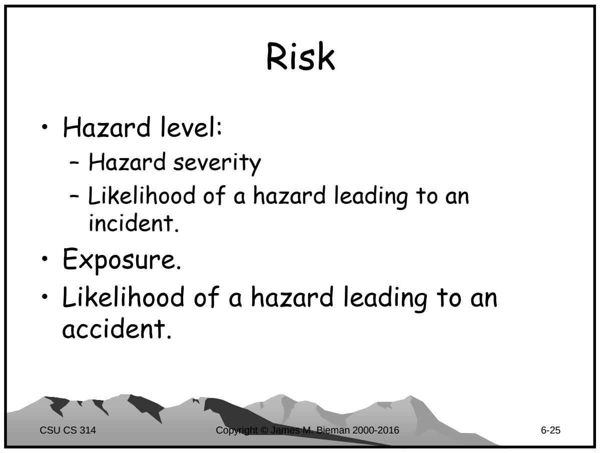


Root Causes of Software Accidents

- General attitude: overconfidence & complacency, safety given low priority.
- Ineffective organizational structure: nobody with authority is really in charge of safety.
- Ineffective technical activities: paperwork rather than real solutions. No attention to real risks.

Safety Definitions

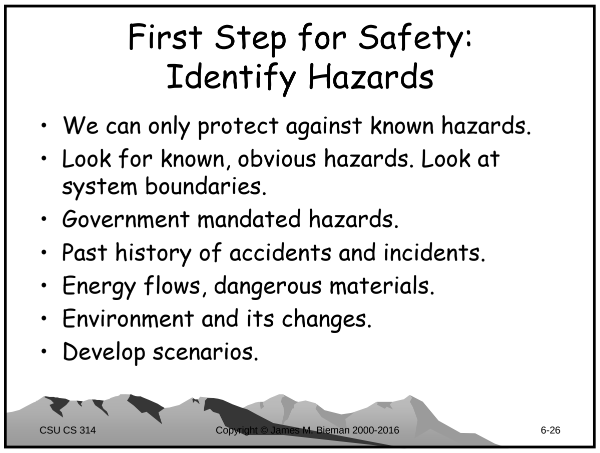
- Safety: freedom from accidents or losses.
- Accident: undesirable and unplanned event that results in a specified loss.
- Incident (also know as a near miss): an event that involves no loss, but with the potential for loss under different circumstances.
- Hazard: a state or a set of conditions of a system or object that, together with other environmental conditions, will lead to an accident.



Risk

- Hazard level:
  - Hazard severity
  - Likelihood of a hazard leading to an incident.
- Exposure.
- Likelihood of a hazard leading to an accident.

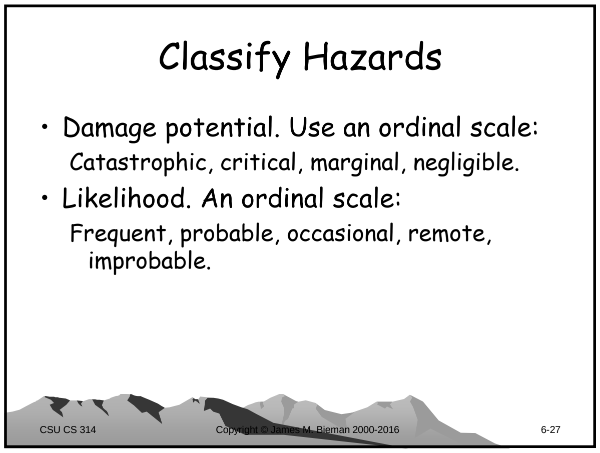
CSU CS 314 Copyright © James M. Bieman 2000-2016 6-25



First Step for Safety: Identify Hazards

- We can only protect against known hazards.
- Look for known, obvious hazards. Look at system boundaries.
- Government mandated hazards.
- Past history of accidents and incidents.
- Energy flows, dangerous materials.
- Environment and its changes.
- Develop scenarios.

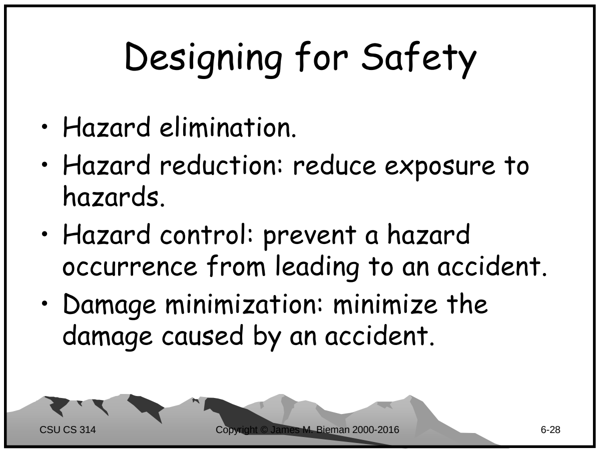
CSU CS 314 Copyright © James M. Bieman 2000-2016 6-26



Classify Hazards

- Damage potential. Use an ordinal scale:
  Catastrophic, critical, marginal, negligible.
- Likelihood. An ordinal scale:
  Frequent, probable, occasional, remote, improbable.

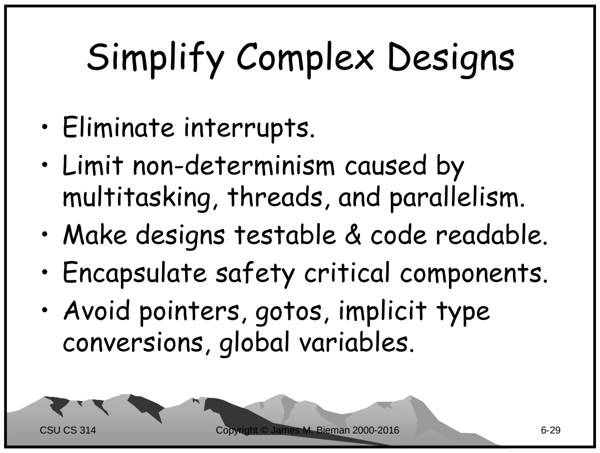
CSU CS 314 Copyright © James M. Bieman 2000-2016 6-27



Designing for Safety

- Hazard elimination.
- Hazard reduction: reduce exposure to hazards.
- Hazard control: prevent a hazard occurrence from leading to an accident.
- Damage minimization: minimize the damage caused by an accident.

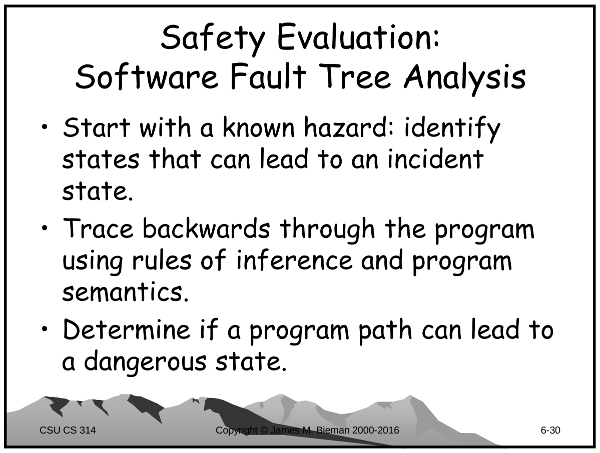
CSU CS 314 Copyright © James M. Bieman 2000-2016 6-28



Simplify Complex Designs

- Eliminate interrupts.
- Limit non-determinism caused by multitasking, threads, and parallelism.
- Make designs testable & code readable.
- Encapsulate safety critical components.
- Avoid pointers, gotos, implicit type conversions, global variables.

CSU CS 314 Copyright © James M. Bieman 2000-2016 6-29



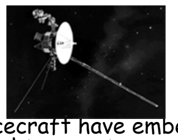
Safety Evaluation: Software Fault Tree Analysis

- Start with a known hazard: identify states that can lead to an incident state.
- Trace backwards through the program using rules of inference and program semantics.
- Determine if a program path can lead to a dangerous state.

CSU CS 314 Copyright © James M. Bieman 2000-2016 6-30

Software Safety: The Voyager & Galileo Spacecraft

Voyager



Galileo

Spacecraft have embedded software on their flight computers:
- Voyager (1977- ?): 18,000 LOC.
- Galileo (1989-2003): 22,000 LOC.

Lots of message passing, real-time monitoring, with complex timing.

Software Safety: The Voyager & Galileo Spacecraft

- Robyn Lutz [1993] studied 387 software errors found during testing.
- Cause and potential effect was documented.
- Out of all software errors, 87 on Voyager and 122 on Galileo are "safety critical".

Faults with "potential significant or catastrophic effects".

Recommendations to Limit Safety-related Software Faults

1. Focus on the interfaces,. Interfaces are where safety-related faults (especially timing faults) tend to be.
2. Identify safety-critical hazards early.
3. Use formal specifications to supplement natural language specs.
4. Improve communication within and between teams.
5. Communicate changing requirements to development and test teams.
6. Require defensive design with run-time safety checks, and backward analysis from critical failures.

View from Voyager

View from Galileo of Jupiter's Great Red Spot