Public Safety Aspects of Software Engineering

Dennis K. Peters, Phd, P. Eng.
Memorial University
http://www.engr.mun.ca/~dpeters

Outline

- What is Software Engineering?
- Why should it matter to us?
- How can software be unsafe?
- How is SE different from other Engineering?
- Why use Software?
- Approaches to software safety.

What is Software Engineering?

- Software project management.
- Programming done by engineers.
- Programming done by anyone.
- A systematic approach to the analysis, design, implementation and maintenance of software. (*IEEE Standard Glossary of Software Engineering Terminology.*)

Why is SE Important?

- Software is a component of many systems.
- Engineers are responsible for the behaviour of the systems they design.
- Engineers are responsible for their designs, regardless of how flawed their tools may be.
- Software systems can be quite complex.
- Bad software can kill.

A Point to Debate

We don't know how to ensure that software is safe, so it should not be used in systems where its failure might be dangerous.

How can Software be Unsafe?

- Therac-25
  - Software controlled nuclear medicine device.
  - Killed several people due to software bugs.
- Airbus A320
  - 'Fly by wire' plane.
  - Crashes have been attributed to software errors.
- Darlington Nuclear Plant
  - Control and safety systems based on software.

Role of Computers in Disasters

- Control
  - Machine control
  - Electrical power systems
- Support for human operations
  - Diagnostic equipment
  - Air traffic control
  - Data systems
- Design & Manufacturing
  - CAD systems
  - Design calculations

Role of Computers in Disasters (cont'd)

- Non-physical Dangers
  - Privacy (personal & corporate)
  - Security
  - Financial systems
- Other pitfalls
  - Over-reliance/Over-confidence
  - Information overload
  - Inappropriate information presentation
  - Trivializing/complicating human roles
  - Hiding failures

How is Software Different?

- No natural internal boundaries
  - Requires special skills to choose designs.
  - Components can interact in many ways.
  - Design is not evident from implementation.
- It's not constructed from materials that obey physical laws.
  - Interpolation is rarely valid.
  - Difficult to build in safety margins.
  - Sensitive to minor errors.
  - Doesn't wear out or break.

How is Software Different? (cont'd)

- Possible interactions with other systems are essentially infinite.
- Very brittle.
- 'Easy' to change.

Advantages of Software

- Can do things that can't be done with other technology.
- Allows relatively easy increases in accuracy.
- Better information displays.
- Doesn't wear out.
- Easier to change, especially if there are multiple installations.
- Can use cheap, mass-produced hardware.
- Smaller.
- Uses less energy.

Approaches to Software Safety

- Reliability
  - Increasing safety requires increasing reliability of software.
  - Evaluating safety of system means estimating the reliability.
- Rigour
  - Safety requires correctness.
  - Use highly rigorous 'formal' methods.
- Hazard Analysis
  - Safety means eliminating hazards.
  - Work backwards to show impossibility of hazard occurring.

Reliability

- Reliability analysis
  - Estimate failure rates by testing.
  - Heavily dependent on *operational profile*.
  - "testing can be a very effective way to show the presence of bugs, but it is hopelessly inadequate for showing their absence." (E.W. Dijkstra)
- Redundant design
  - If one system fails, the other(s) will take over.
  - Assumes that failures are uncorrelated.

Rigour

- **Rigorous construction**: derive program from precise specifications.
- **Verification**: show that a program satisfies its specification.
- Both require that specification be written using mathematical notation.

Why not natural Language?

While acting as the bus controller, the MDM shall set the indicator to "failed" upon detection of transaction errors of selected messages to RTs whose FDIR is not inhibited in two consecutive processing frames within 100 msec of detection of the second transaction error if; a backup BC is available, the BC has been switched in the last 20 sec, the SPD card reset capability is inhibited, or the SPD card has been reset in the last 10 major (10 sec) frames, and either:

- the transaction errors are from multiple RTs, the current channel has been reset within the last major frame, or
- the transaction errors are from multiple RTs, the bus channel's reset capability is inhibited, and the current channel has not been reset within the last major frame.

Source: Easterbrook *et al.*, 1998.

Hazard Analysis

- Start by identifying hazards.
- Work backwards showing how hazards can never happen.
- Can be done on code or specifications, or both.
- Premised on assumptions.

Assumptions from Therac-25 HA

- Programming errors have been reduced by extensive testing on a hardware simulator and under field conditions on teletherapy units. Any residual software errors are not included in the analysis.
- Program software does not degrade due to wear, fatigue, or reproduction process.
- Computer execution errors are caused by faulty hardware components and by "soft" (random) errors induced by alpha particles and electromagnetic noise.

Source: Leveson, 1995

A Point to Debate

We don't know how to ensure that software is safe, so it should not be used in systems where its failure might be dangerous.

- Do we know how to ensure that software is safe?
  - Maybe, but even if we do, the cost is very high.
- Should it be used in systems where its failure might be dangerous?
  - It **is** being used in safety-critical systems, so what are we going to do about it?

Bibliography

- S. Easterbrook, R. Lutz, R. Covington, J. Kelly, Y. Ampo and D. Hamilton, "Experiences Using Lightweight formal Methods for Requriements Modeling," *IEEE Trans. Software Engineering*, vol. 24, no. 1, pp. 4-14, Jan. 1998.
- N. G. Leveson, *Safeware: System Safety and Computers*, Addison-Wesley, 1995.
- P. G. Neumann, *Computer-Related Risks*, Addison-Wesley, 1995.