Safety-Critical Software as Social Experimentation - How Will Software Engineers Internalize Risk Concerns?

Clark Savage Turner

Basic Arguments Overview

- ***Safety-Critical*** Software development is a process of experimentation
- Social expectations on experimentation are well known
  - – Legal bounds on experimentation apply to the safety-critical software development process
    - liability decisions are explained by the relative social need for the information generated by the failure!
      - – recall Petroski argument

Roadmap

- The safety-critical software problem
- Technical and social progress
- Tort law
  - Products Liability and “defects”
- Software engineering as experimentation
- The Therac-25 as an example
  - analysis of some defects with experiment analogy
- Commonly heard technical defenses
- Recommendations for lower risks of liability

Safety-Critical Software

- Many software systems inherently risky
  - increasingly used in avionics, nuclear, medical
  - accidents *will* happen [Per84]
    - example: Therac-25 accidents [LT93]
      - 6 persons massively overdosed
      - 2 years continuing problems
        - engineers blind to main contributing causes
      - lawsuits resulted, large sums paid in settlements!
    - a hard problem: no “silver bullet” expected [Bro95]

Technology will progress

- Homo Faber: Man, the maker
  - technical progress is built on new knowledge
    - thus, progress is often built upon catastrophic technical failure
      - failure *necessary* to technical progress (Petroski)
- Risk level for software is uncertain [Par90]
  - technically it is unbounded
    - note: risk to life and property is a *social problem*

Human Progress

- Society seeks to protect and enhance the welfare of its members
  - – society is generally risk-averse
- Much of technical progress does indeed enhance social welfare
- Where is the balance struck?
    - tort law balance: accept risks that are likely to benefit society in the long run

Tort Law Underpinnings

- Basic rules of social interaction
  - how can society minimally enforce “civilization”
    - versus “law of the jungle” with survival of fittest
  - society collectively provides the “ground” for all civilized progress
    - this is part of the “social contract” required to maintain the “ground”
  - balance risks vs. benefits of social action
    - a truly Utilitarian principle

Experiment

- Science is a way to provide good theories
  - about the natural world
    - to explain natural “laws” (See Kuhn)
    - give science the power of explanation
      - and engineers use such knowledge to create the “artificial world” (Simon)
        - consider “artificial world” as another topic of study
- Science is a “process” of experimentation to answer questions regarding our theories

What is “Experiment” ?

- Scientific Method
  - – Observation
    - recognition of a problem or subject of interest
  - – Hypothesis
    - intelligent / intuitive “guessing”
    - human subjects: hypothesize about a **population**
  - – Test
    - process of experimentation to obtain data to refute or support the hypothesis
    - must be “repeatable”

Social Experimentation [MS89]

- Observation: life is not good (safe, etc.) *enough*
- Hypothesis: safe for intended purposes
- Population: users, passengers, patients, etc.
- Levels of experimentation
  - lab: counterexamples "fixed"
    - high control, low generalizability
  - field: possible lesson for state of the art [Pet85]
    - low control, high generalizability
- We experiment to *make progress*

Tort Law as Constraint on Social Experimentation

- Tort obligations are imposed regardless of contract (social obligations of a civilized society)
  - a decision on who will pay the *inevitable costs* of social experimentation
    - someone *always* pays
  - analog: social consent to experimentation in tort law?
    - **can these obligations be explained by the social value of the information generated by the failed experiment?**
- **Tort obligations are therefore implicit constraints on Software Requirements and Design**

Products Liability

- General Rule: “One … who sells […] a *defective* product is subject to liability for harm […] caused by the *defect.* [Draft Restatement of Products Liability, 1998]
  - this rule and its basic categories have *not* yet been applied to software
    - but there is general agreement that software is a “product” for purposes of the law

What is a Defect?

- Two important categories of product defect:
  - manufacturing defect
    - product departs from its intended design
      - *strict* standard for liability, “no fault” liability
  - design defect
    - design safety is not “enough”
      - a basic *negligence*, risk-utility standard for liability
      - “fault” is the very basis for liability
- Need to *know* legal category of defect to do any risk analysis!

Software “Manufacturing” Defect

- Hypothesis: ***This particular product*** offers the level of safety “promised” in the design / specs
- Liability - hypothesis false: product *fails to meet its own [internal] design standard* for safety
  - based on proof that actual product failed to meet its own design standards (specs)
    - legal question: is there any social value to random experimentation with people’s lives?
    - social consent vitiated by lack of value to information generated by the failed experiment
      - no Petroski-style learning going on :-)

Software Design Defect

- Hypothesis: ***This design itself*** offers a reasonable level of safety
  - a bigger question than just for the "product"
    - **it involves the "process"**
- Liability - hypothesis false: product design was not sufficiently safe by social standards
  - legal proof made that reasonably safe / cost effective alternate designs were available (see caselaw)
    - therefore little or no gain for the "state of the art" by this failed experiment!

Software Design Defect

- No liability - hypothesis proved true, consent based on *social need for the info*
  - this is the sort of information that furthers the state of the art!
    - it involves a social need outweighing the risk inherent in the experimental activity
      - there must be a benefit to society that is worth the risk
      - Social Risk and Social Benefit are inversely proportional
        - big social benefit allows for more acceptable risk

Two Therac Problems

1. Hamilton, Ontario accident:
   - engineers “fixed” a problem they could not reproduce
     - design change: 3 bit turntable location instead of 2
2. Tyler, Texas accident:
   - code increments (by 1) an 8 bit safety-crit var
     - that’s only set to zero to show a safe condition
     - rolls over to zero every 256 cycles
     - could this be a “manufacturing” defect? (hypothesize it)

Classify the Therac defects?

- 1. Is this a safe design decision?
  - Do we need to know what happens when we “fix” a problem we cannot reproduce?
- 2. Is this part of design intent?
  - Does a safety-critical variable that rolls over to zero, possibly falsely indicating a safe condition every 256 machine cycles a lesson for the state of the art?

Commonly Heard Excuses

- Software is so new, we don’t know enough
  - should we build such safety-critical systems?
- Our systems are so complex, we don’t fully understand them
  - same for Aerospace and other “systems”
  - should we build systems that exhibit “pseudo-random” behavior?
- “We used the best process!”
  - good for design, irrelevant to implementation defects
    - but what is the difference between “design” and “implementation” for software?

Conclude

- Internalize risk concerns via Legal Constraints
  - implicit in every safety-critical software development effort
- How to stay in the game?
  1. Implementation must meet safety specs
  2. Process of safety-critical software development must be rationalizable
     - safety design effort must be commensurate with the risk and level of danger involved

Bibliography

- [Bro95] Brooks, The Mythical Man-Month, Addison-Wesley, 1995
- [LT93] Leveson, Turner, “An Investigation of the Therac-25 Accidents,” IEEE Computer, July, 1993
- [MS89] Martin, Schinzinger, Ethics in Engineering, 2d Ed., McGraw-Hill, 1989
- [Par90] Parnas, “Evaluation of safety-Critical Software,” CACM June, 1990
- [PER84] Perrow, Normal Accidents: Living with High Risk Technologies, Basic Books, NY, 1984
- [PET85] Petroski, To Engineer is Human: The Role of Failure in Successful Design, Vintage, NY, 1992