

Software for Dependable Systems

EECS 4315

www.eecs.yorku.ca/course/4315/

Daniel Jackson, Martyn Thomas, and Lynette I. Millett, editors.
Software for Dependable Systems: Sufficient Evidence? The
National Academies Press. 2007.

- Professor at MIT.
- MA in Physics from Oxford University in 1984.
- PhD in Computer Science from MIT in 1992.



Source: people.csail.mit.edu/dnj

- British independent consultant and software engineer.
- Visiting professor at the Oxford University.
- Visiting professor at the University of Bristol.
- Fellow at the Royal Academy of Engineering.



Source: Twitter

- Director of the Forum on Cyber Resilience at the National Academies.
- MSc in Computer Science from Cornell University.



Source: LinkedIn

The remaining slides are based on slides by Daniel Jackson which can be found at the <http://people.csail.mit.edu/dnj/talks/depcert07/depcert07.pdf>.

- Growing role of mission-critical software,
- risks of undependable software,
- high cost of development, and
- uncertainty about value of certification.

A big question

How can software be made dependable in a cost-effective manner?

Extent of failures to date:

- software has already resulted in critical system failures,
- leading to death, injury and major economic loss.

Roots of failure:

- bugs in code account only for 3% of failures blamed on software,
- most failures blamed on interactions with operators and environment, and
- often poor understanding of requirements.

Development strategies:

- building dependable software is difficult and costly,
- quality is highly variable,
- certification regimes and standards have mixed record, and
- organizational culture has dramatic effect.

Incomplete and unreliable data about:

- extent and frequency of software failures,
- efficacy of development approaches, and
- benefits of certification schemes.

Consequences:

- mandating particular process does not guarantee dependability,
- avoid being too prescriptive about particular tools or techniques,
- put in place mechanisms for collecting industry-wide evidence, and
- make evidence focus of dependable system development.

Injury and loss of life:

- Korean Air 747 in Guam, 200 deaths (1997),
- 30,000 deaths and 600,000 injuries from medical devices (1985-2005)
perhaps 8% due to software?

Major economic loss:

- Code Red worm, \$2.75 billion in damage.

Critical application domains:

- Palmdale air-traffic control outage, 800 flights disrupted (2004), and
- blackout in Northeast (2003).

Widespread use of invasive devices:

- 200,000 pacemaker recalls due to software (1990-2000), and
- 23,900 Prius cars affected by software recall (2005).

Centralization leads to single point of failure:

- pharmacy database failure (Cook & O'Connor, 2005).

Certification Problems

In general:

- expensive and burdensome,
- certification \neq fewer vulnerabilities, and
- limited focus on security components.

In avionics:

- study of code at levels A and B finds no difference, and
- modified condition/decision coverage testing rarely exposes errors.

In medicine:

- heavy reliance on testing and process,
- hasn't prevented accidents due to bad practice, and
- 17 deaths in Panama (2001), similar incident to Therac-25 (1985).

Why Certification Helps

Promotes safety culture:

- seriousness, attention to detail, and
- rigorous process.

Helps justify safety investment:

- balances hurry to get product to market.

In medicine:

- 98,000 patients die annually from preventable errors,
- better tools for diagnosis and intervention, and
- effect of widespread IT on health would be major.

In avionics:

- detecting impending accidents,
- “controlled flight into terrain” responsible for most deaths,
- collisions during ground operations, and
- digital controllers to monitor engine performance.

In other areas:

- transportation: preventing car accidents,
- energy: monitoring generation and distribution, and
- telecommunications: better connectivity during emergencies.

Towards Dependable Software: the Three Es

Explicit:

- properties established,
- assumptions about domain and usage, and
- level of dependability.

Evidence:

- dependability case that properties hold,
- scientifically justifiable claims, and
- open to audit by a third-party.

Expertise:

- approach is technology-independent,
- demand for evidence stretches today's best practices, and
- deviate from best practice only with good reason.

Why be explicit?

- no system dependable in all respects, and
- so must choose, consciously or not.

What to make explicit?

- critical properties expected to hold,
- assumptions about environment and usage, and
- level of dependability claimed.

Radiotherapy example:

- property: emergency stop button turns off beam within 10ms,
- assumption: mechanical beam stop works, and
- level: 1 failure in 100 machines operating for 20 years.

What happened?

- Airbus A320, Warsaw 1993,
- aircraft landed on wet runway,
- aquaplaned, so brakes didn't work, and
- pilot applied reverse thrust, but disabled.

Why did that happen?

- reverse thrust disabled iff landing gears under compression.

Dependability case:

- an auditable argument for dependability, and
- $\text{software} \wedge \text{assumptions} \Rightarrow \text{properties}$.

For each element of argument, use most effective technique, for example,

- type checker – independence of modules,
- static analysis – no buffer overflows,
- theorem proving – code meets specification,
- model checking – protocol doesn't deadlock and
- testing – environmental assumptions hold.

Process:

- to preserve chain of evidence, and
- deployed code = analyzed code.

Testing and Analysis

Testing:

- tiny proportion of scenarios, so rarely justifies high confidence,
- sometimes exhaustive testing is possible, and
- automatic regression testing is an essential process practice.

Analysis:

- for local reasoning,
- formal and informal, but best if mechanized, and
- static analysis, model checking and theorem proving.

Justified claims:

- must state what inferences are drawn from analysis and testing, and
- bug finders are useful, but might not contribute much.

Role of Process

When to construct the case?

- too expensive to delay until system is complete, and
- construct hand-in-hand with system.

Chain of evidence:

- produced during development,
- preserved by careful checks and procedures, and
- leaves auditable records.



© Scott Adams, Inc./Dist. by UFS, Inc.

Approach is technology-independent:

- doesn't rely on particular tools, languages, methods,
- just following best practices is not good enough,
- but new approach demands expertise.

Examples of expertise required:

- prioritization and formalization of requirements,
- design of true data abstractions, not just lip service to OOP,
- substantive code standards: avoiding unsafe language features, and
- reflective bug tracking: back to origin.

No alternative:

- high confidence will require verification,
- cost of verifying entire code base too high, and
- therefore must design system with properties in mind.

Separation of concerns is key:

- establish critical properties in a few small modules,
- need independence arguments, and
- support with safe languages, virtual machines, etc.

Certification Regimes

Current regimes:

- few encompass the combination the book by Jackson et al. recommends.

In the future:

- certification = inspection and analysis of dependability case,
- by development organization, customer, or third-party, and
- no single regime for all circumstances.

Accountability:

- no fixed prescription,
- but must be clear at outset who's responsible for failure.

Culture Change Needed

Transparency:

- customers want to make informed judgments,
- criteria and evidence for claims must be transparent,
- publishing defect data boosts supplier's credibility, and
- certification process should be transparent (cf. e-voting).

Accountability:

- who is responsible if it fails?
- no fixed assignment, but must be clear.

Evidence and openness:

- dearth of evidence hampers technology and policy advances, and
- encourage collection, publication and analysis of failure data.

Education:

- demand for dependable software requires workforce,
- emphasis on software construction as systems building,
- high school: less on mechanism, more on problem solving, and
- university: more on security, usability, specification, argument.

Research:

- tools and techniques for constructing dependability cases,
- components and compositional dependability cases,
- how to bolster role of testing as evidence, and
- reasoning about fail-stop systems.

Now versus Future

<i>requirements</i>	<i>current</i>	
<i>design</i>	massive informal list	
<i>testing</i>	highly coupled	
<i>analysis</i>	expensive and unfocused	
<i>best practices</i>	in reviews, unrecorded	
<i>quality plan</i>	specify commenting style	
<i>certification</i>	long, unread, unchanging	
	testing and process checklist	

<i>future?</i>	
a few critical properties	
small trusted base	
environmental assumption	
proof of no deadlock	
guarantee no buffer overflow	
succinct, known, responsive	
audit of dependability case	