software for dependable systems: sufficient evidence?

Daniel Jackson, MIT

NSA HCSS · Baltimore, MD · May 9, 2007

THE NATIONAL ACADEMIES
Advisers to the Nation on Science, Engineering, and Medicine

project status

report

- draft approved by National Academies
- prepublication on website this week
- books available later this summer

participants

committee

Daniel Jackson, MIT, Chair
Joshua Bloch, Google
Michael Dewalt, Certification Systems, Inc.
Reed Gardner, University Of Utah School Of Medicine
Peter Lee, CMU
Steven Lipner, Microsoft Trustworthy Computing Group
Charles Perrow, Yale
Jon Pincus, Microsoft Research
John Rushby, SRI International
Lui Sha, UIUC
Martyn Thomas, Martyn Thomas Associates
Scott Wallsten, Progress and Freedom Foundation
David Woods, Ohio State

staff

Lynette Millett, Study Director
David Padgham, Associate Program Officer
Jon Eisenberg, Director, CSTB

why this study?

sponsors

- National Science Foundation
- National Security Agency
- Office of Naval Research
- Federal Aviation Administration

concerns

- growing role of mission-critical software
- risks of undependable software
- high cost of development
- uncertainty about value of certification

a broad perspective

a big question

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

a diverse committee

- researchers and practitioners
- engineering, economics, psychology, sociology
- expert domains, esp. avionics, medical, security

assessment

what we know

extent of failures to date

- software has already resulted in critical system failures
- 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, environment
- 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
- organizational culture has dramatic effect

what we don't know

incomplete and unreliable data about

- extent and frequency of software failures
- efficacy of development approaches
- 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
- make evidence focus of dependable system development

notable accidents

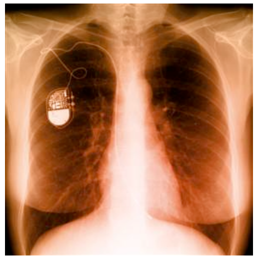
injury and loss of life

- 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, $2.75 billion in damage

near misses?



critical application domains

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

widespread use of invasive devices

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

centralization leads to single point of failure

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

"Accidents are signals sent from deep within the system about the vulnerability and potential for disaster that lie within"

—Richard Cook and Michael O'Connor. Thinking About Accidents And Systems.
In K. Thompson, H. Manasse, eds. Improving Medication Safety, ASHP, Washington, DC.

certification problems

security: Common Criteria

- expensive and burdensome
- certification ≠ fewer vulnerabilities (eg, Windows 2000 vs. 2003)
- limited focus on security components

avionics: DO178B

- study of code at levels A and B finds no difference
- SSAC respondents: MCDC rarely exposes errors

medicine: FDA premarket approval

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

why certification helps

promotes safety culture

- seriousness, attention to detail
- rigorous process
- self-selection of engineers

helps justify safety investment

- balances hurry to get product to market

"The software is checked **very carefully** in a bottom-up fashion... But **completely independently** there is an independent verification group, that takes an **adversary attitude** to the software development group, and tests and verifies the software **as if it were** a customer of the delivered product... A discovery of an error during verification testing is **considered very serious**, and its origin **studied very carefully** to avoid such mistakes in the future."

—Richard Feynman. Report of the Presidential Commission on the Space Shuttle Challenger Accident, June 1986.

software for a safer world

in medicine

- 98,000 patients die annually from preventable errors
- better tools for diagnosis and intervention
- 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
- digital controllers to monitor engine performance

in many other areas

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

approach

a systems perspective

may be surprising

- eg, graceful degradation may thwart monitor

software as component

- dependability not an inherent property of software
- software is always part of a larger system
- property of interest is in the world, not at the interface!

accidental systems and criticality creep

- eg, adding wireless access to data in hospital
- eg, pilot comes to depend on moving-map display

operators as components

- if operator relied upon, then include in system analysis
- too easy to blame failures on operator error

three Es

explicit

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

evidence

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

expertise

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

explicitness

why be explicit?

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

what to make explicit

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

radiotherapy example

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

environmental assumptions

what happened

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

why

airborne ⇔ disabled

airborne ⇔ not WheelPulse ⇔ disabled

ENV ✗ MACHINE ✓

simplified; for full analysis, see [Ladkin 96]

evidence

dependability case

- an auditable argument for dependability
- software ∧ assumptions ⇒ properties

for each element of argument, use most effective technique, eg

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

process

- to preserve chain of evidence
- eg, deployed code = analyzed code

testing and analysis

testing

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

analysis

- for local reasoning and for assembling end-to-end case
- formal and informal, but best if mechanized
- static analysis, model checking and theorem proving

justified claims

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

role of process

when to construct the case

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

chain of evidence

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

MY ACCOMPLISHMENTS THIS MONTH INCLUDE COMPLYING WITH OUR ISO 9000, SARBANES-OXLEY AND SEI-5 POLICIES.
AND IF YOU MAKE A NEW POLICY, I WILL COMPLY WITH IT SO FAST IT WILL MAKE YOUR HEAD SPIN!
IS IT MY IMAGINATION OR IS PRETENDING TO WORK GETTING EASIER?
scottadams@aol.com
www.dilbert.com
8-5-05 © 2005 Scott Adams, Inc./Dist. by UFS, Inc.
© Scott Adams, Inc./Dist. by UFS, Inc.

expertise

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
- reflective bug tracking: back to origin

simplicity

"Simplicity does not precede complexity, but follows it."
—Alan Perlis

no alternative

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

separation of concerns is key

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

broader issues

certification regimes

current regimes

- few encompass the combination this report recommends

in the future

- certification = inspection and analysis of dependability case
- by development organization, customer, or third-party
- 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
- 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
- encourage collection, publication and analysis of failure data

education and research

education

- demand for dependable software requires workforce
- emphasis on software construction as systems building
- high school: less on mechanism, more on problem solving
- 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
- reasoning about fail-stop systems
- etc...

a brave new world

a caricature, but gives basic sense

	current	*proposed*
requirements	massive informal list	a few critical properties
design	highly coupled	small trusted base
testing	expensive and unfocused	environmental assumptions
analysis	in reviews, unrecorded	proof of no deadlock
best practices	specify commenting style	guarantee no buffer overflow
quality plan	long, unread, unchanging	succinct, known, responsive
certification	testing and process checklist	audit of dependability case

summary

assessment

- need improvements to keep pace with demand for dependable software
- more data badly needed

recommended approach

- dependability case based on **explicit** claims, **evidence**, **expertise**
- process and testing: necessary but not sufficient
- simplicity is essential: complexity, dependability, economy (pick two)
- certification = analysis of dependability case
- demand accountability

policy issues

- transparency essential for improving dependable software market
- failure data should be collected, published and analyzed
- education and research should be focused on dependability