

# Software reliability and dependability: a roadmap

Bev Littlewood  
Lorenzo Strigini

Centre for Software Reliability, City University, London, UK  
{B.Littlewood, L.Strigini}@csr.city.ac.uk  
<http://www.csr.city.ac.uk>

## Outline:

- reliability: the political vs the technical
  - a range of different environments
- technical part :
  - reliability as an engineering discipline
  - what it can deliver now
- some open problems: technical
- some open problems: cultural

## The shrink-wrap market

- development culture:
  - features and time-to-market matter
  - uninterested (uneducated) in reliability
  - “no choice but upgrading”
- customer culture:
  - “computer will be flaky anyway”
  - “it is not working, I / you are hopeless with computers”
- the market:
  - “*what customers want*”, or “*dummies buying from crooks*” ?
- why is reliability engineering relevant here?
  - you may choose not to buy more features with their reliability cost
  - but how if the latter is unknown?

*driven by culture, market, law - no technical difficulties*

## The other extreme: traditional safety-critical industry

- care for dependability, conservatism
- regulatory checks-and-balances
- good record!

but

- assessment often unreal “we produce  $10^{-9}$  systems and we know it because we obey the standards”
- not sure *what* produces the good record for software
- which practices improve dependability (cost-effectively?)
- developers hampered by conservatism, regulators puzzled by progress (e.g., “smart” components)

*some hard technical problems*

# Plenty of problems with software reliability engineering

At the two ends of the range:

- shrink-wrap world: much knowledge available but not used
- safety-critical world: not enough knowledge for desired progress

## The technical side: state of the art

What do we know how to do?

- for *achieving* dependability:  
how to build
  - higher-quality, more robust software than most software now built
  - checks and defences in platforms and systems
- for *assessing* dependability:  
observing software in real (or realistic) operation,  
predict future behaviour (with uncertainty)
  - steady-state or with reliability growth
  - quite useful within its limits
  - although can't predict  $10^{-9}$  without enough observation

## Some open problems .. see paper

- focus on user-centred, system-level dependability qualities
- design for dependability assessment
  - failure prevention
  - system monitoring
- diversity and variation as drivers of dependability
- judgement and decision-making
  - Engineering approach
  - Choice of process for dependability
  - Formalism and judgement in assessment
- very large-scale systems
- integration with human reliability

## Open problems for the immediate future (1): Component-based reliability assessment

“what real engineers do”

- model system structure
- add data about components
- predict system reliability

why not with software systems?

( surely COTS parts must help! )

- software is not monitored
- reliability of software components varies
  - need to understand variation of demands
  - characterise how it affects uncertainty
- complex failure dependencies
- systems do not isolate components
  - *design for reliability* would be good for *reliability assessment* too

## Open problems for the immediate future (2) Judgement and combination of evidence

Reliability data often insufficient for desired confidence

We use other evidence: process quality, static analysis, proof

*but*

- what does each item of evidence imply?
- how do they add together? Confusing dependencies

Formal methods help: e.g., Bayesian

- guaranteed consistency *but* not guaranteed realism
- risks in relying on expert judgement - GIGO
- open research problems on getting reliable judgments

## Open problems for the immediate future (3) Dependability as *system* issue

Computers less and less isolated:

- e.g.  
{{computer+pilot+aircraft}+other aircraft}+air traffic control ...
- or office, company, society ...

failures from not understanding interactions

progress needed:

- cross-learning between disciplines
- system awareness in developers

*cf* lessons learned in early days of automation about wrong ways to automate  
*“....what is it doing now?”*

## Cultural side: outstanding non-technical problems

*Motherhood, but true:*

- Public perception of software dependability
  - misperceptions of computer systems and of risks
  - cf reactions to Y2K non-catastrophe
- Development culture: needs awareness of
  - reliability needs, system issues
  - available techniques
- Management culture: requires
  - acceptance of uncertainty
  - understanding of basic possibilities / concepts
  - reliability statements as decision aid not marketing issue
- Research culture: needs to
  - understand practitioners' problems
  - resist enthusiasm for baroque models

## Conclusions

Technical:

- some hard problems worth solving
- some real solutions worth using

Social implications:

- e.g.,  
could we build legally meaningful warranties that protect both users and producers of software-based systems ?