DERTS event, IEE, May 2003, SOSA

“Embedded control systems and modern software development problems

by

Professor Les Hatton

the Computing Laboratory, University of Kent
Oakwood Computing Associates Ltd.
L.Hatton@ukc.ac.uk, lesh@oakcomp.co.uk

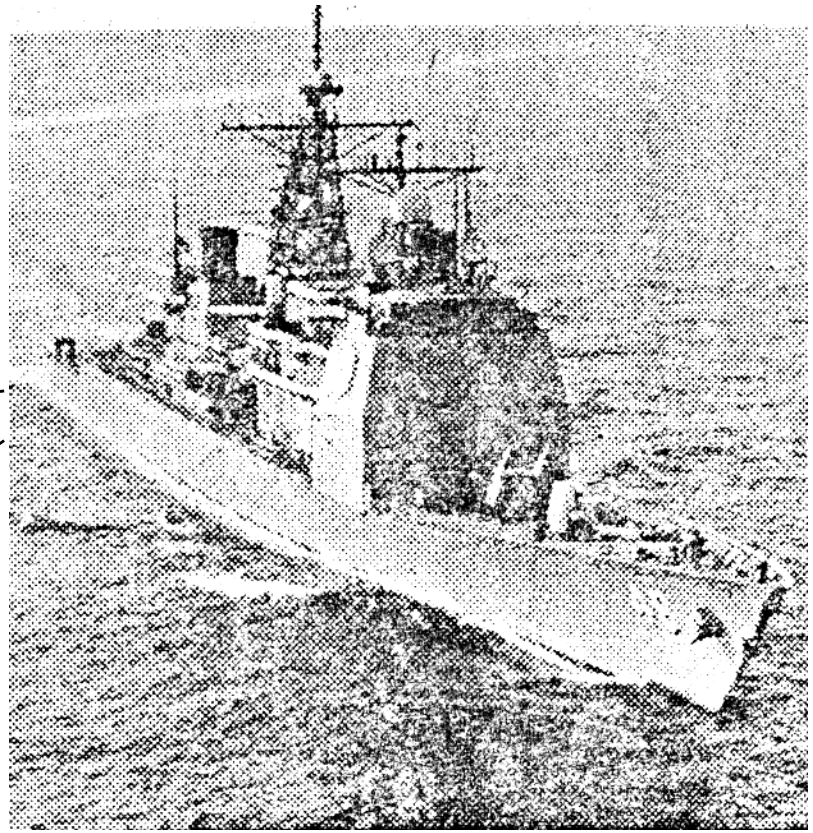
Version 1.1: 21/May/2003

©Copyright, L.Hatton, 2003-

Not a good decade:

The USS Yorktown in "Please wait ..." mode

"What does 'You have performed an illegal act' mean sir ?"



© L.Hatton, 2003-

Not a good decade:

An Airbus having a bad day

A Tarom airlines Airbus which performed an uncontrolled dive,
climb, roll and spin near Orly in 1995 due to ‘a fault in the automatic pilot’.
The plane landed safely, a tribute to the pilots’ skill.

© L.Hatton, 2003-

Not a good decade:

Ariane 5: What goes up ...

© L.Hatton, 2003-

Not a good week either

- 12/May/2003. “Thailand’s finance minister trapped in his BMW when the doors and windows locked automatically and the air-conditioning switched off when the computer crashed.” Reuters
- 12/May/2003. “A software failure in the Soyuz TMA-1 was responsible for the 400 mile navigation error.”
  - (Note that this is not the first time this software has failed. In 1997, the control computer first of all almost caused the two halves of the spacecraft to collide and shortly afterwards aligned the spacecraft for re-entry nose first rather than heat-shield first.)

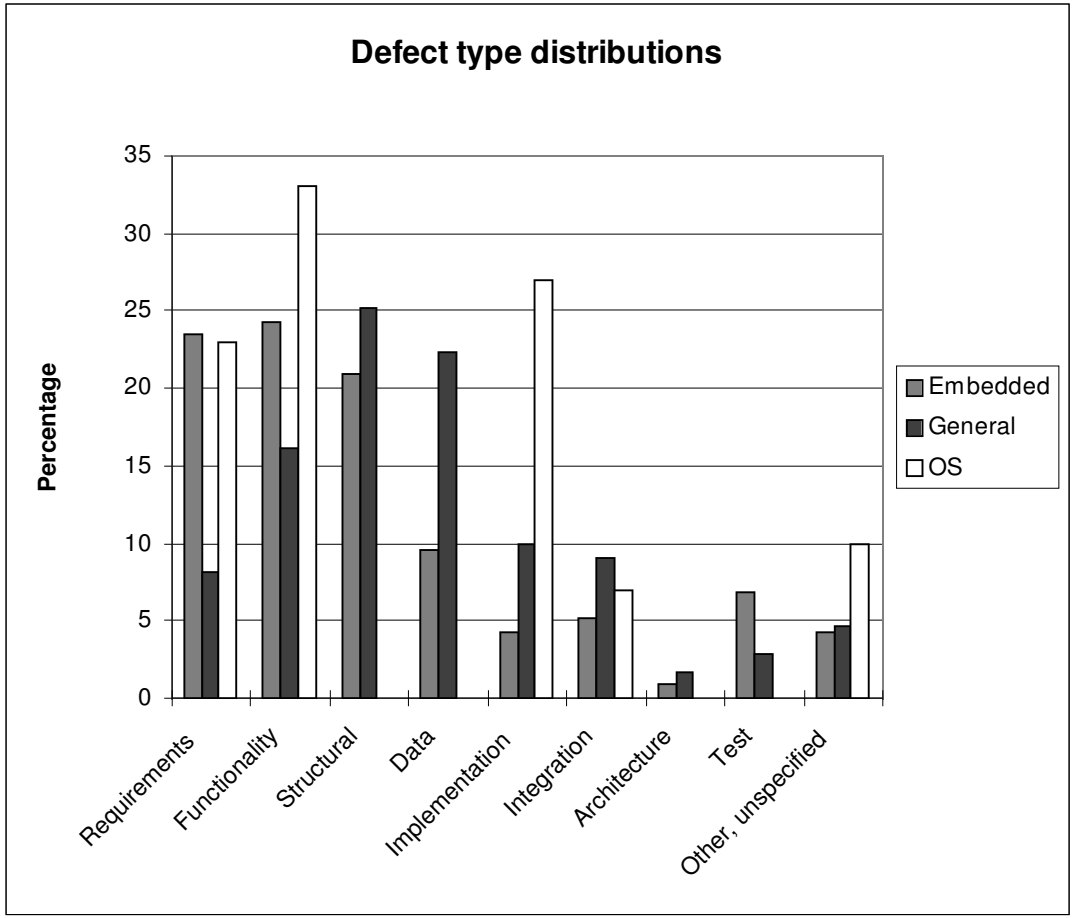
© L.Hatton, 2003-

Overview

- **<u>New kinds of problem ?</u>**
  - What do experiments tell us ?
- **Same old stuff but worse ?**
  - Size, complexity, testing and features
  - Long-cycle failures
  - Embedded systems and language support
- **Conclusions**
  - Unpromising aspects
  - Promising aspects

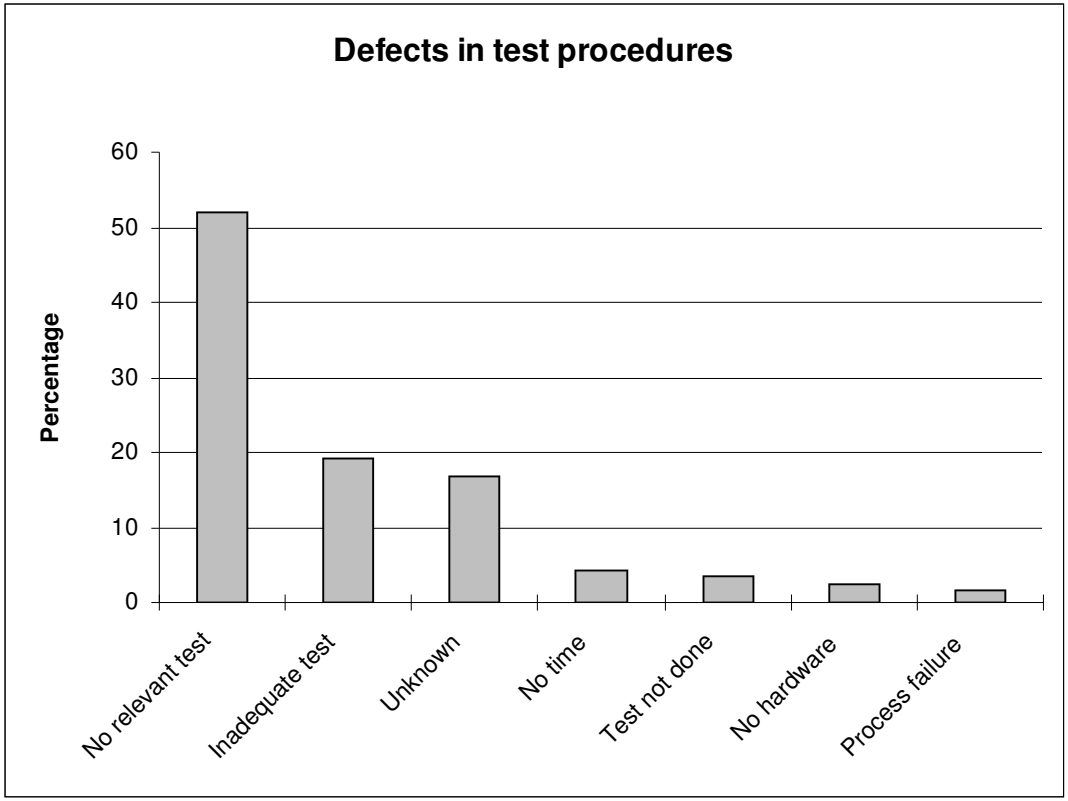
© L.Hatton, 2003-

Failure type distributions (Vinter and Poulson (1995))



© L.Hatton, 2003-

Defects in Unix OS test procedures (1995)



© L.Hatton, 2003-

Overview

- **New kinds of problem ?**
  - What do experiments tell us ?
- **<u>Same old stuff but worse ?</u>**
  - Size, complexity, testing and features
  - Long-cycle failures
  - Embedded systems and language support
- **Conclusions**
  - Unpromising aspects
  - Promising aspects

© L.Hatton, 2003-

Size, complexity, testing and features

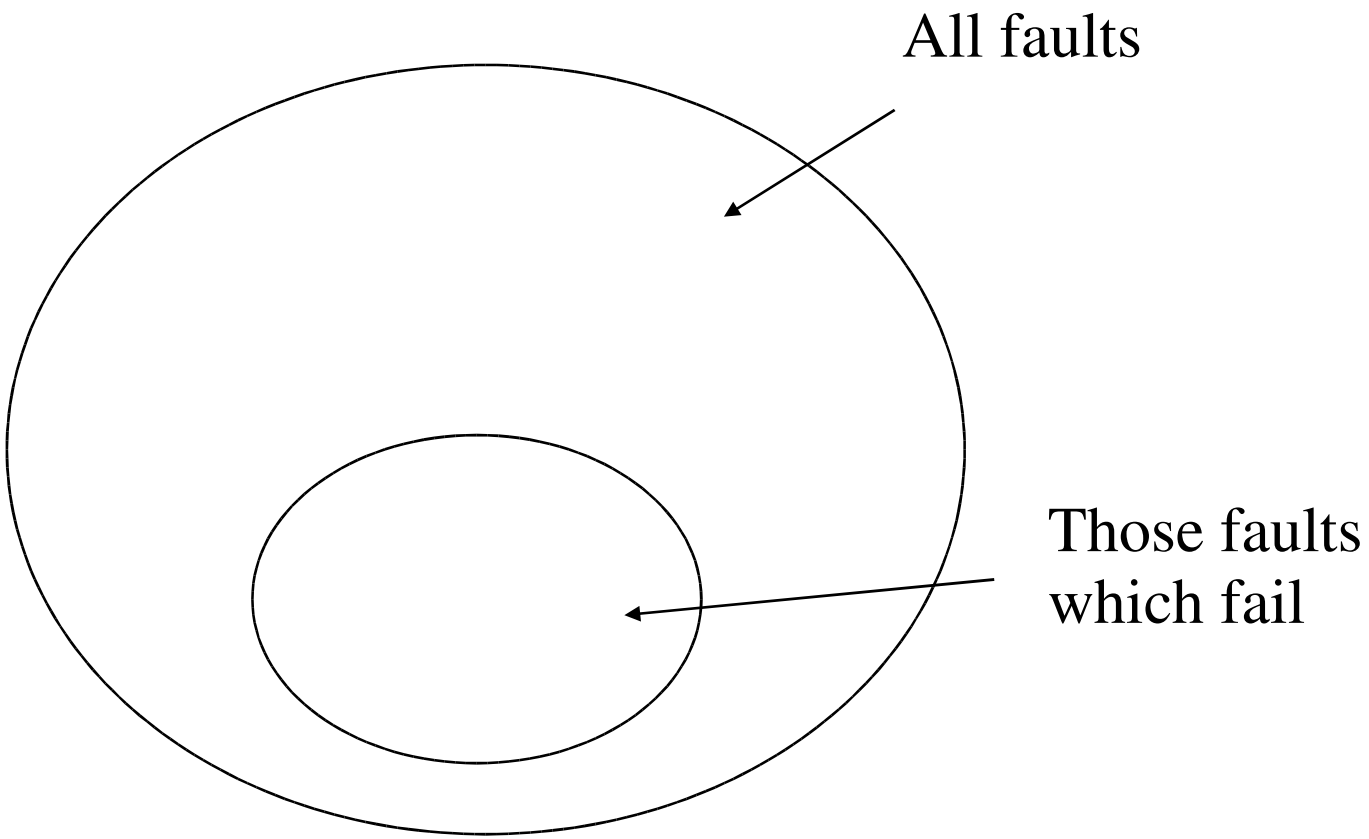
Principle characteristics of software failure

- Growth
- Sensitivity and chaotic failure
- Interaction between system components
- Cost
- Inconvenience

© L.Hatton, 2003-

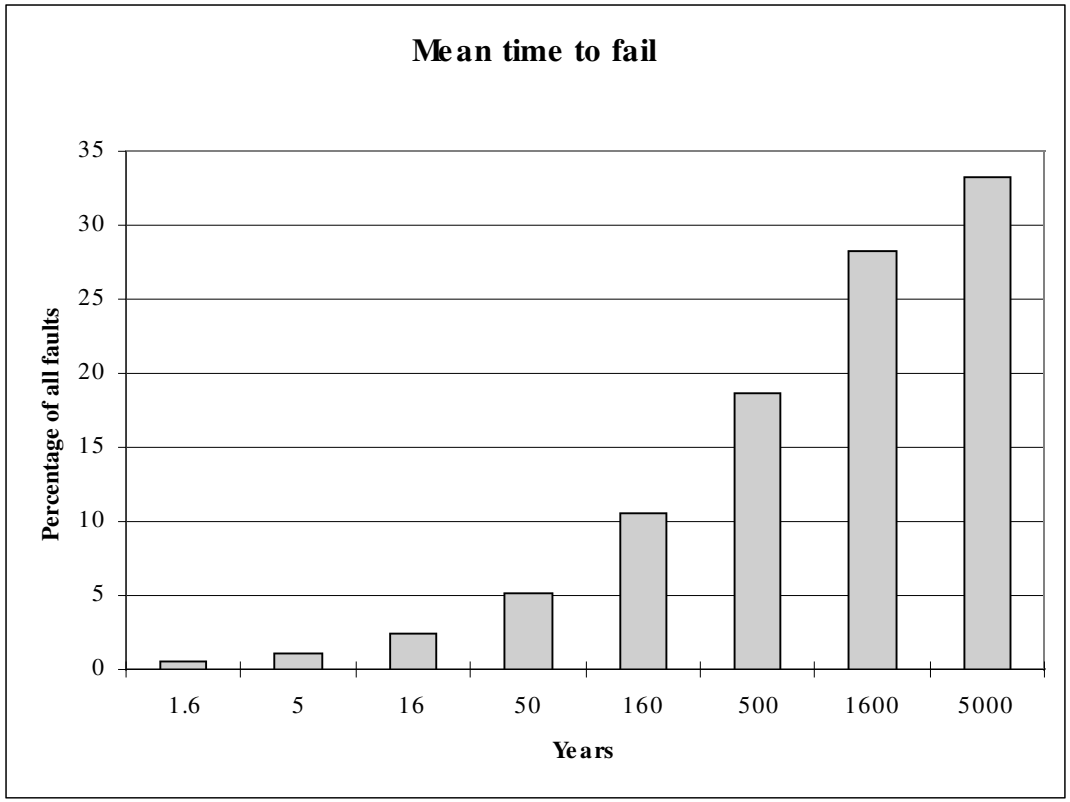
Long-cycle failures:

Faults that fail and faults that don't



© L.Hatton, 2003-

Mean time to fail in Adams (1984)



© L.Hatton, 2003-

Problems with defect longevity

If you find this hard to believe consider the following 1000-2000 year defects found in seismic data processing software as used in oil and gas exploration.

© L.Hatton, 2003-

How to collect seismic data

Borrow around 20 million dollars and buy one of these

If it doesn't work out you can always run booze cruises.

© L.Hatton, 2003-

Package similarity v. processing stage

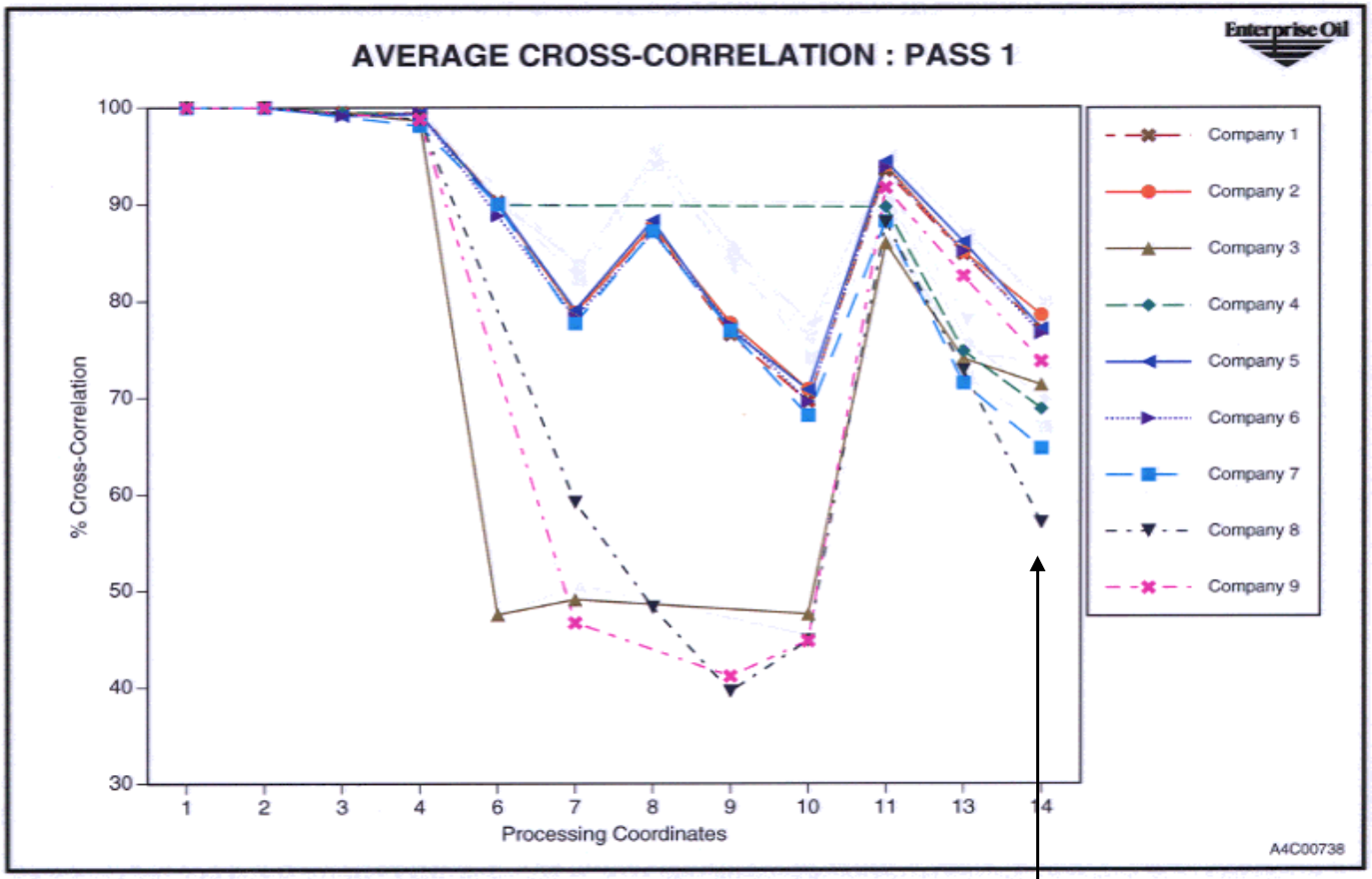
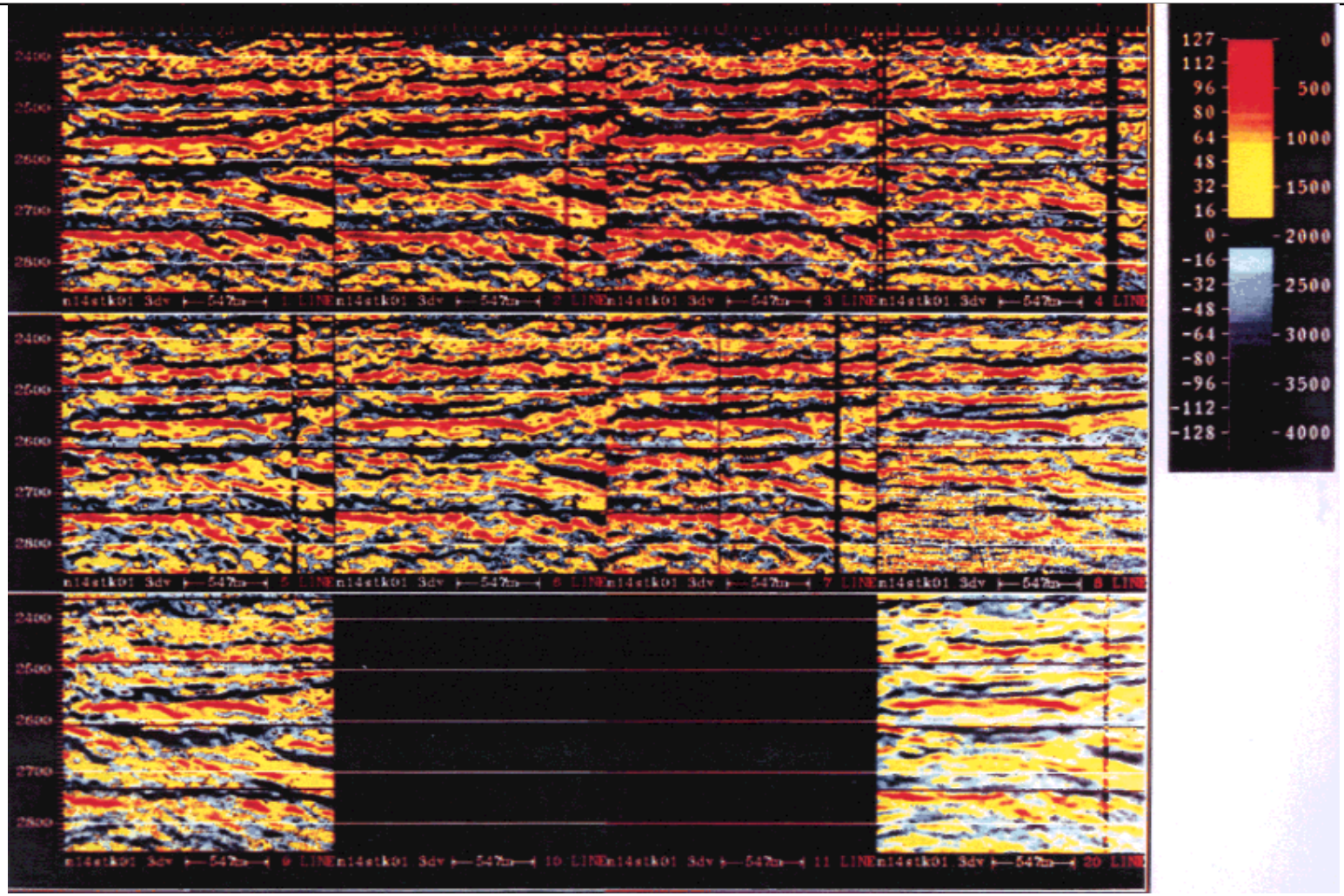


Figure 2.1

Geologists view data at this point

© L.Hatton, 2003-

The end product: 9 subtly different views of the geology



© L.Hatton, 2003-

Why are long-cycle failures important ?

Imagine two scenarios of 7x24x365 use:-

- Air-traffic control system, 20 copies
  - After 25 years, 80% of the faults which could fail have not yet had time to fail according to Adam's data - only 500 execution years are accumulated
- Embedded control system, 1,000,000 copies
  - 5000 execution year failures occur after two days.

© L.Hatton, 2003-

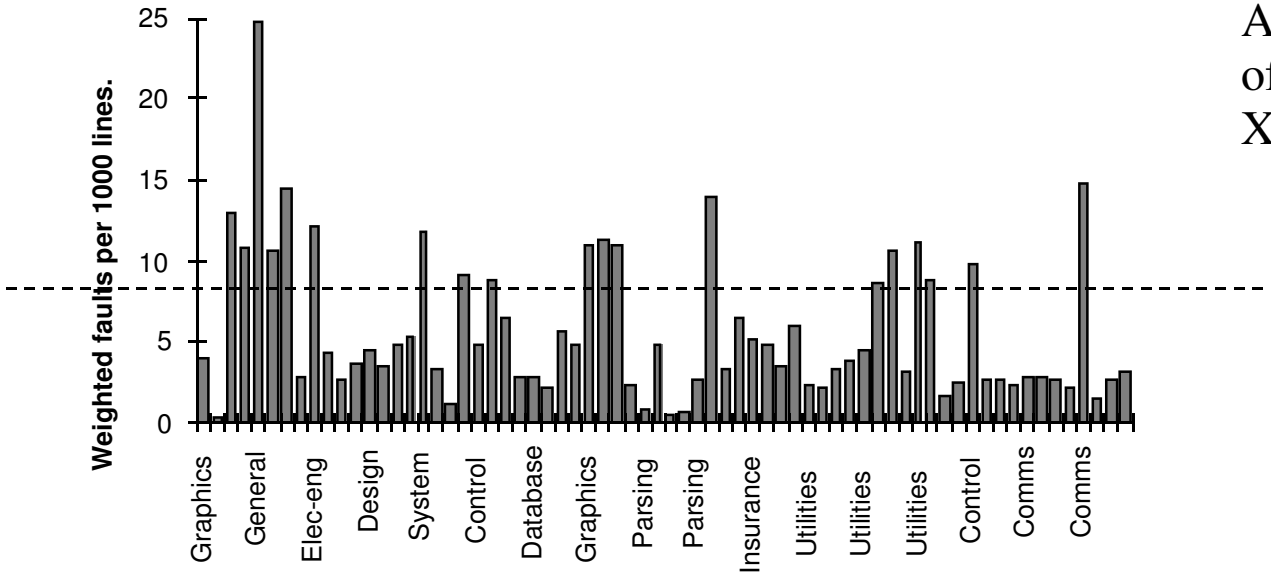
Embedded systems and language support

Today the dominant language appears to be C. Note the following:

- ISO C90 has been an extremely successful language for both embedded and non-embedded systems
- Regrettably, its standardisation progress has not served embedded systems developers well
- Like all programming languages, ISO C90 has well known but numerous fault modes …

© L.Hatton, 2003-

Fault frequencies in ISO C90 applications



Average of 8 per XKLOC

These are the frequencies of the 400 or so known fault modes in C which can be detected statically as measured in 54 released applications 1993-1998

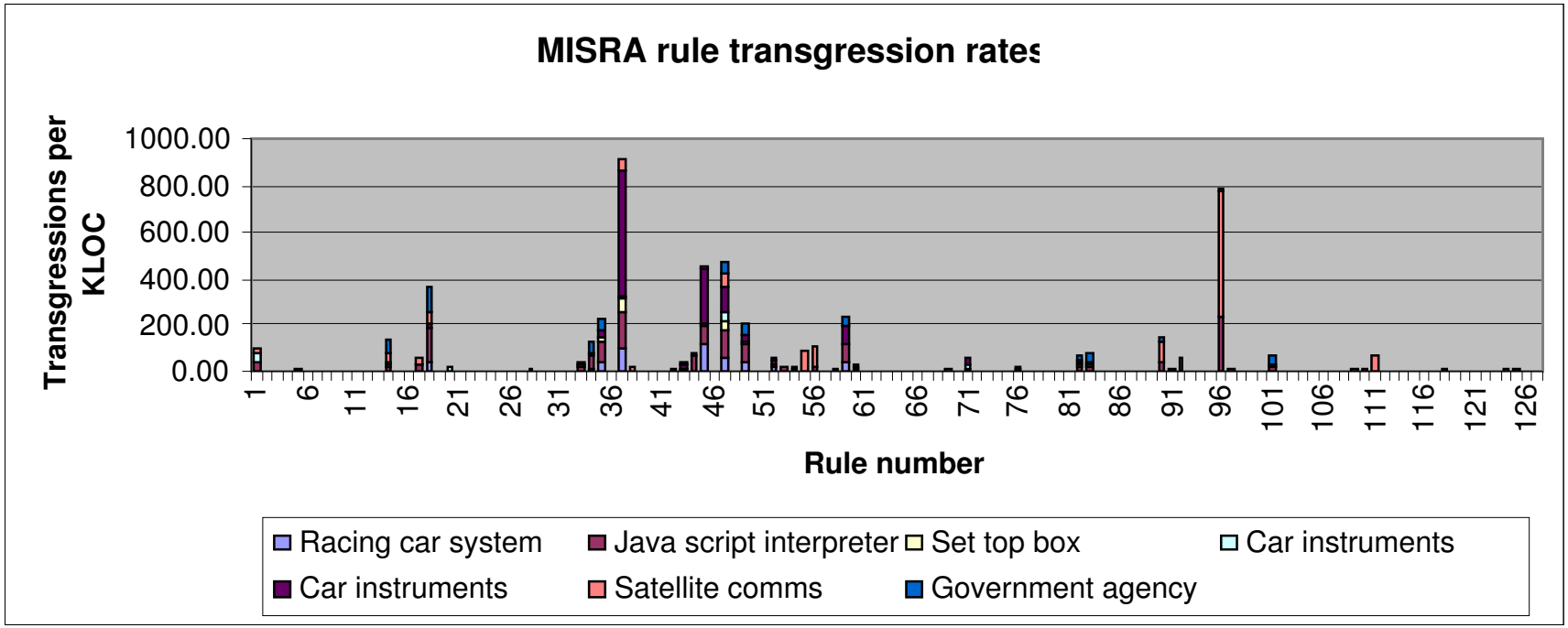
© L.Hatton, 2003-

Conflicting forces in programming language design

- **Forces generally beneficial**
  - Reducing the language by removing features which are known to fail (safer subsets – MISRA is quite good at this)
- **Forces usually not beneficial**
  - Extending the language by further standardisation (standards committees)

© L.Hatton, 2003-

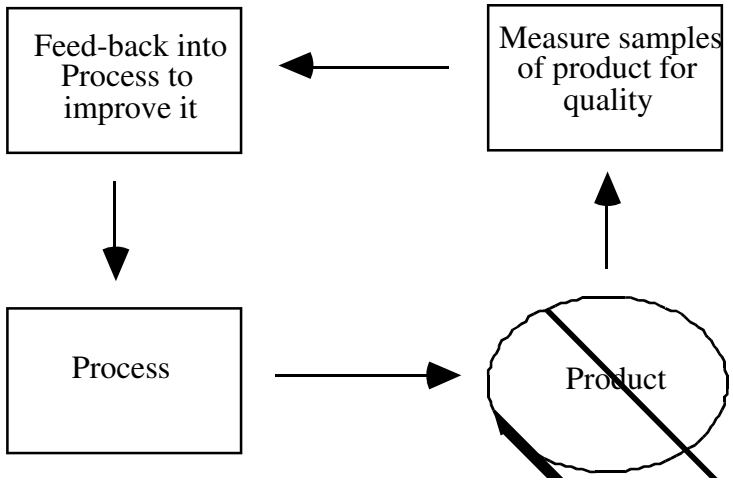
Individual rule transgression rates in MISRA C (version 1)



MISRA C currently suffers from too little measurement input and consequently a lot of noise but it is unquestionably a step in the right direction even if developers find it difficult to adhere to.

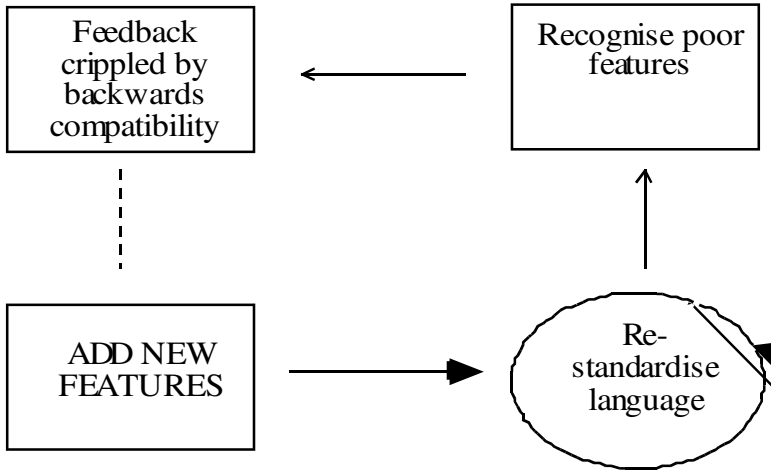
© L.Hatton, 2003-

Control Process feedback - the essence of engineering improvement



© L.Hatton, 2003-

Why languages can't improve



Using the model of control process feedback, we see that the feedback stage is crippled by the "shall not break old code" rule or "backwards compatibility" as it is more commonly known.

© L.Hatton, 2003-

Overview

- **New kinds of problem ?**
  - What do experiments tell us ?
- **Same old stuff but worse ?**
  - Size, complexity, testing and features
  - Long-cycle failures
  - Embedded systems and language support
- **<u>Conclusions</u>**
  - Promising aspects
  - Unpromising aspects

© L.Hatton, 2003-

Conclusions

- **Promising aspects**
  - With existing technology and the right kind of training, it is perfectly possible to write *extremely* reliable systems in C.
  - Examples:-
    - The Linux kernel, (now arguably the most reliable application the human race has produced).
    - CDIS air traffic control system, asymptotic defect density of around 0.7 per XKLOC, (200,000 XLOC).
    - Safer C toolset, asymptotic defect density of 0.31 per XKLOC, (127,000 XLOC)

These are all close to the reported asymptotic defect density of the NASA Shuttle software, (around 0.1 per XKLOC, (Keller)).

© L.Hatton, 2003-

Conclusions

- **Unpromising aspects**
  1. We still inject far too many defects during implementation when they could be avoided
  2. Most developers are still over-reliant on dynamic testing and pay little attention to inspection, a situation made worse by embedded environments
  3. The ‘backwards compatibility’ rule hallowed in language standardisation forces languages inexorably towards self-destruction
  4. We are not training developers sufficiently well in engineering concepts.

© L.Hatton, 2003-

Discussion points

1. Computer science is not scientific
   … it has no measurement basis
2. Software engineering isn't soft and has no engineering in it.
   … it does not rely on experiential feedback

Discuss …

© L.Hatton, 2003-