ESCOM 2001

"15 years of measuring the effects of defect on scientific and other software:

How do we satisfy customers in the long-term ?"

by

Les Hatton

Oakwood Computing Associates, Surrey, U.K. and
the Computing Laboratory, University of Kent
lesh@oakcomp.co.uk

Version 1.1: 15/Feb/2001

Usability

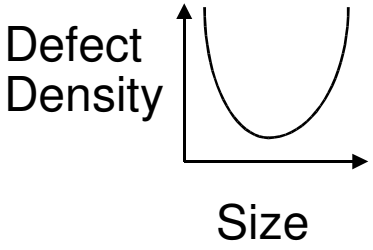
Usability = Use Ability

I couldn't think of anything more to say on this topic. :-)

A personal view of software failure

1990-92:
T1: ~10 static faults/KLOC in F77, C. (C++ worse)

1989-1995



1995-1999:
Win'95 1 defect every 42 mins.
Mac - 1 defect every 188 mins.
Linux - Almost never.

1984-1988:
Porting same F77 package gave 4 sig.fig. agreement on different platforms.

1990-1993:
T2: 9-version dynamic experiment.
Only 1 sig. fig. agreement left at end.

1995:
Formal methods => 3:1 better
Static fault highly correlated to dynamic failure.

1996:
O-O/C++ has 2-3 times corrective maintenance cost.

1997:
Compression and accuracy

1995-1996:
100% statement coverage often implicated in high-integrity systems.

1998-1999:
Why do we have so much repetitive failure in software ?

1999-:
Necessary and unnecessary complexity

Preparing the ground

Fixing the definitions

- A *fault* is a statically detectable property of a piece of code or a design
- A *failure* is a fault or set of faults which together cause the system to show unexpected behaviour at run-time

Overview

- **1984-1988: Portability experiments**
- **1988-1997: Fault experiments**
- **1990-1996: Failure experiments**
- **1996-1997: Correlating fault and failure**
- **1995-2000: Does paradigm shift help ?**
- **2001-: Some interesting questions**

1984-1988: Portability

The Seismic Kernel System (SKS)

- About a million lines of Fortran 77 developed for processing seismic data
- Ported to 10 different architectures, Cray down to Data General with attached FPS array processor. Porting time about 2 weeks.
- Portable graphics based on GKS
- Inhouse portable meta description language for array processing.
- Cost about $3million to develop

1984-1988: Portability

The Seismic Kernel System

- Achieved 4 significant figures of agreement (eventually*) across all architectures on typical seismic data processing benchmarks. Single precision floating point arithmetic used, 32-38 bit.

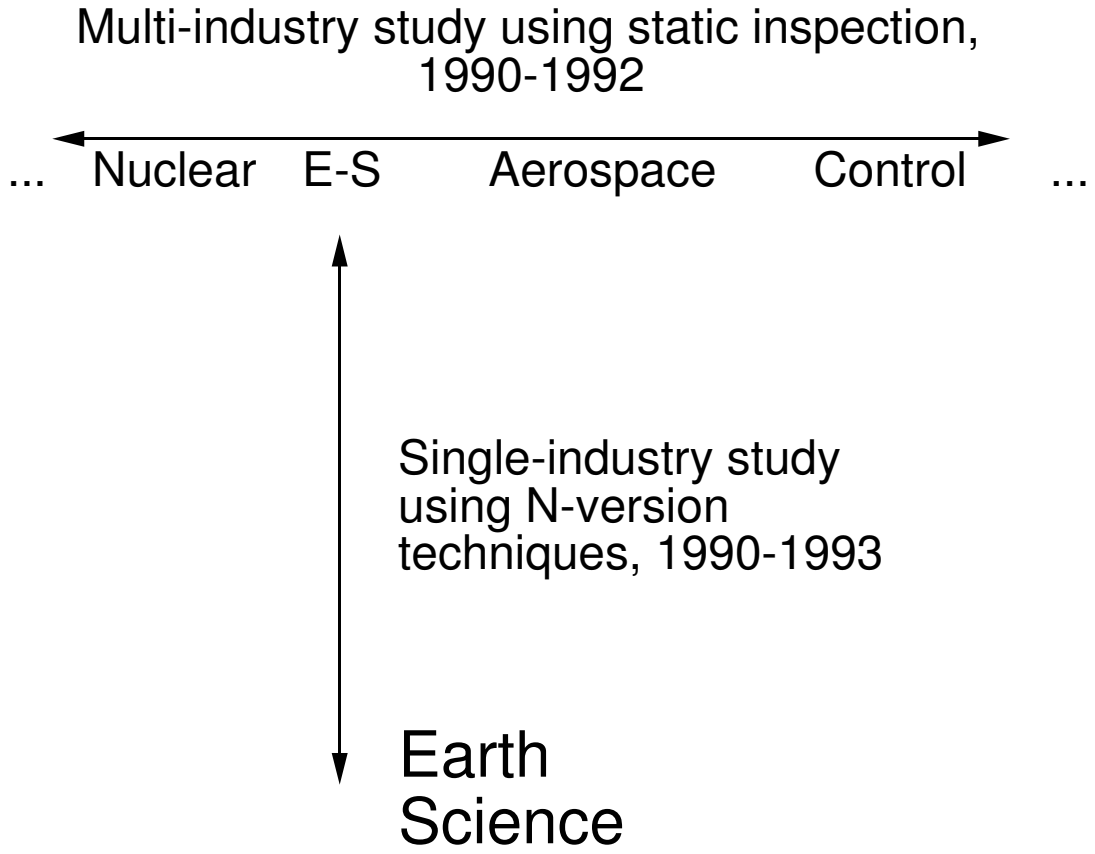
  * The following statement cost 2 of those until it was found in the middle of a 2-D Fourier Transform:

  ```
  if ( ABS(a-b) .gt. 1E-3 ) then ...
  ```

Overview

- 1984-1988: Portability experiments
- 1988-1997: Fault experiments
- 1990-1996: Failure experiments
- 1996-1997: Correlating fault and failure
- 1995-2000: Does paradigm shift help ?
- 2001-: Some interesting questions

The T-experiments

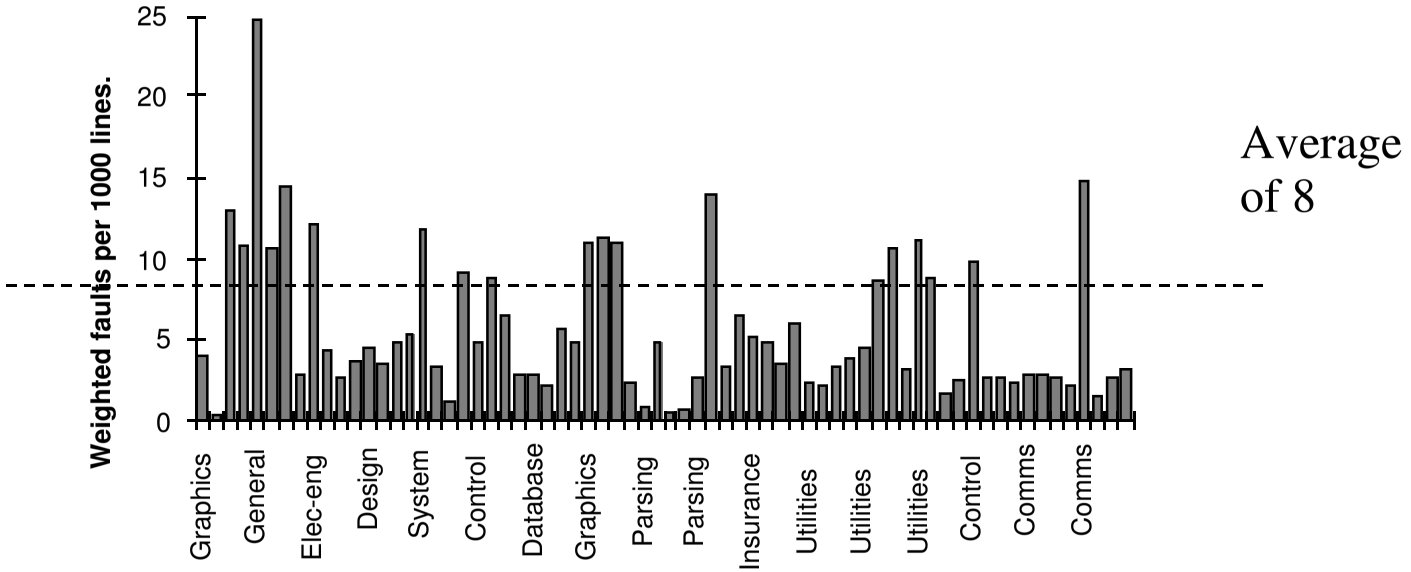


1988-1997: The T1 Fault experiments

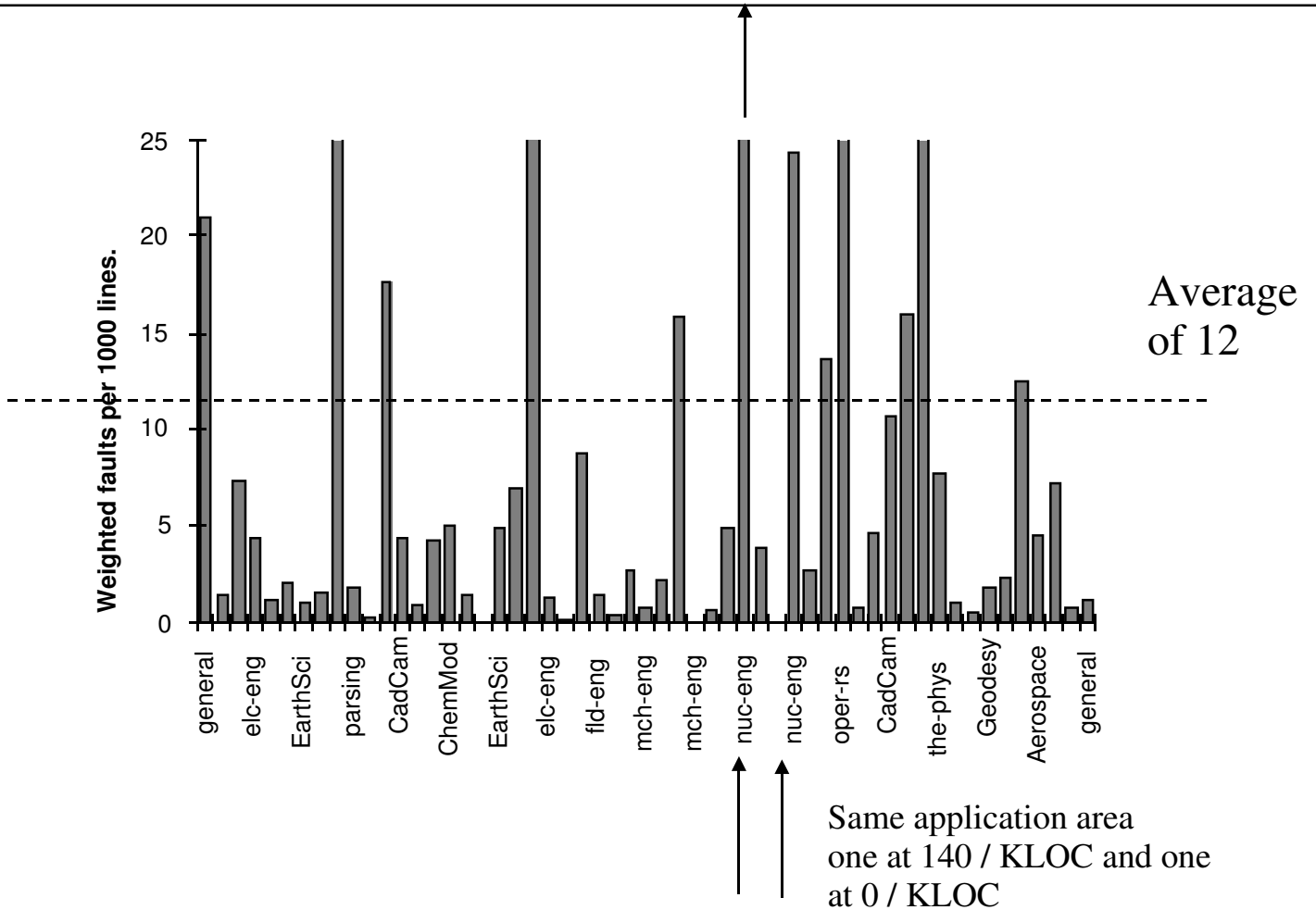
Stages

- Observed many repeating faults in development of SKS
- Developed F77 parsing engine to study other packages, 1988-1992
- Developed C parsing engine to study similar problems in different language, 1990-1994
- Measured around 100 major systems 1988-1997
- Developed more advanced C parsing engine 1996-2000, restart experiments on embedded control systems

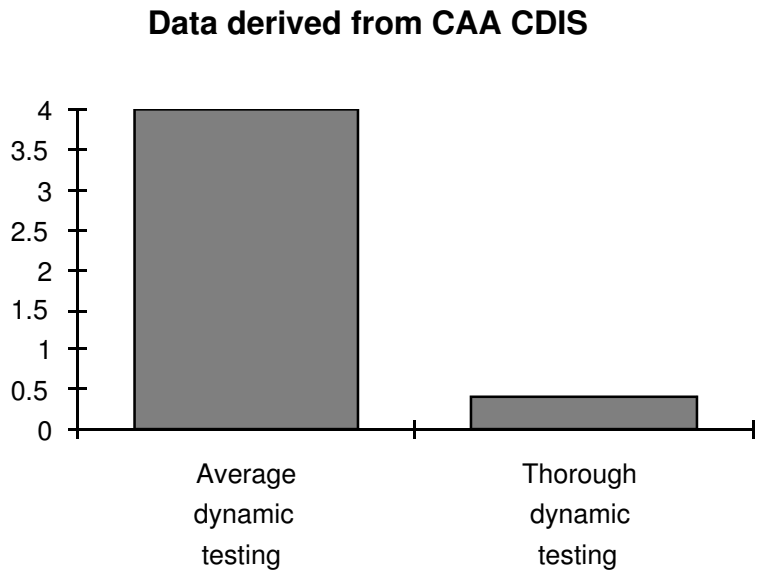
Fault frequencies in C applications



Fault frequencies in Fortran 77 applications



Where and how do faults fail historically ?



This study shows that statically detectable faults do in fact fail during the life-cycle of the software.

Overview

- **1984-1988: Portability experiments**
- **1988-1997: Fault experiments**
- **1990-1996: Failure experiments**
- **1996-1997: Correlating fault and failure**
- **1995-2000: Does paradigm shift help ?**
- **2001-: Some interesting questions**

1990-1996: Failure experiments

Stages

- An observation: Failure experiments are REALLY expensive compared with fault experiments
- "T2" experiment, 1990-1993
  - Funded by Enterprise Oil plc in the UK
  - Compared the output of 9 packages all in Fortran 77 developed independently
  - Carried out with a colleague Andy Roberts

T2 details

- 9 independently developed commercial versions of same ~750,000 F77 package of signal processing algorithms.
- Same input data tapes.
- Same processing parameters, (46 page monitored specification document).
- All algorithms published with precise specification, (e.g. FFT, deconvolution, finite-difference wave-equation solutions, tridiagonal matrix inversions and so on).
- All companies had detailed QA and testing procedures.

Basic goals of T2 experiment

- **Overall goals were:**
  - To estimate the magnitude of disagreement.
  - To see what form disagreement took.
  - To identify poorly implemented processes.
  - To attempt to improve agreement by feedback confirming nature of fault.
  - To preserve complete confidentiality.

Data analysis

- **Analysis goals were:**
  - Analyse at 14 "primary" calibration points and 20 "secondary" calibration points.
  - Analyse data in multiple windows.
  - Use two sets of independently developed analysis software to improve confidence.

Similarity v. coordinate: No feedback

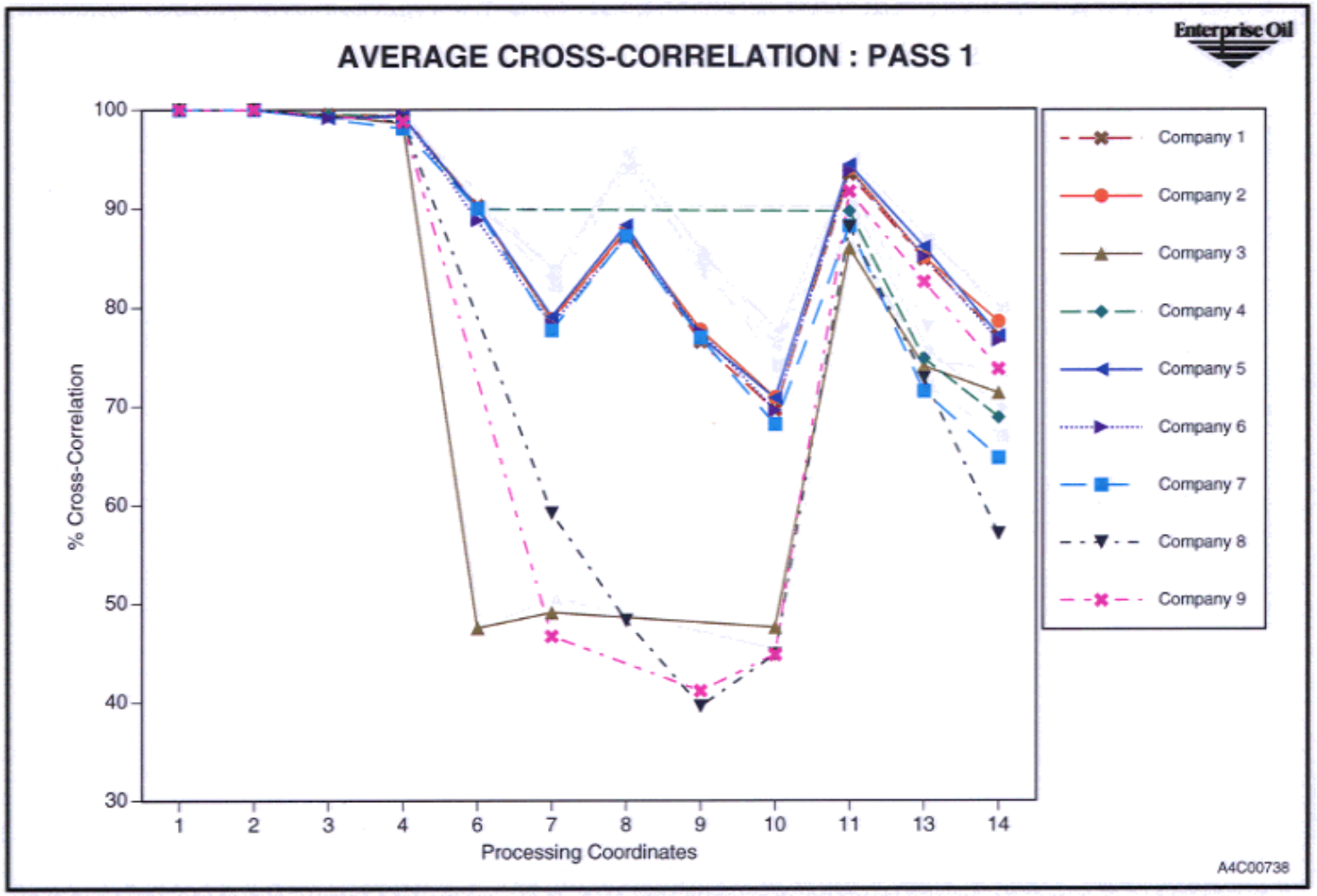
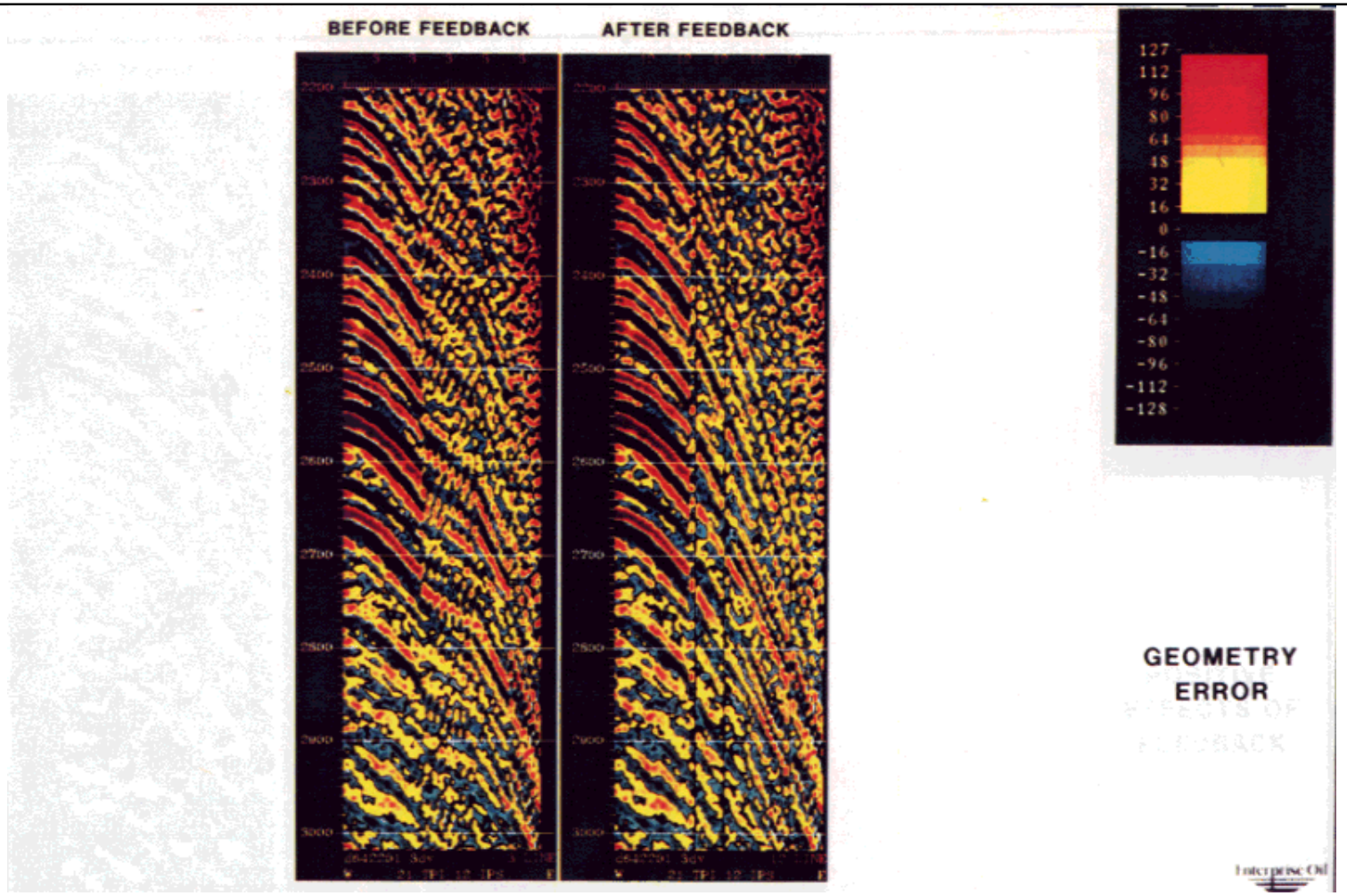


Figure 2.1

Defect example 1: feedback detail



Similarity v. coordinate: Feedback to company 8

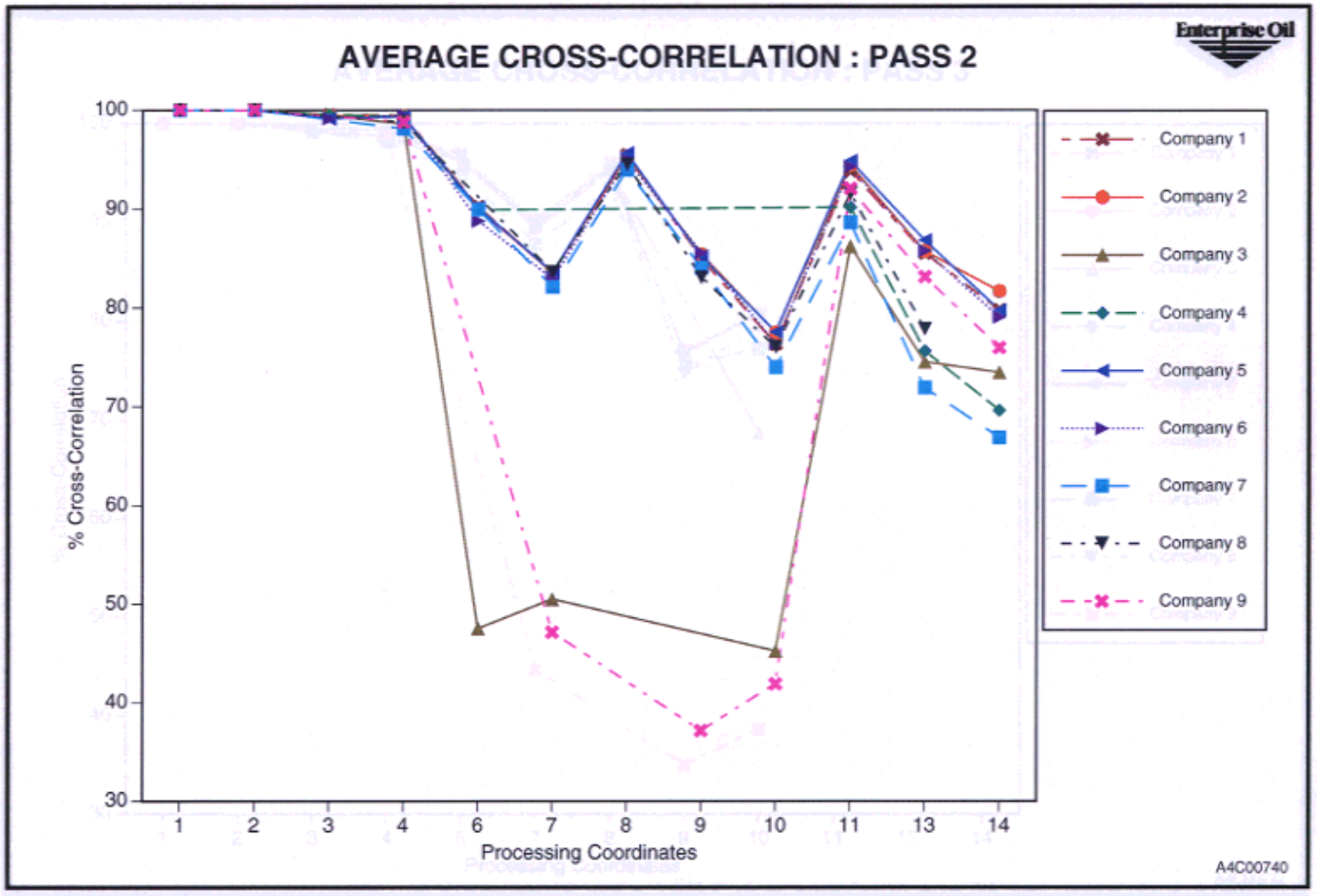
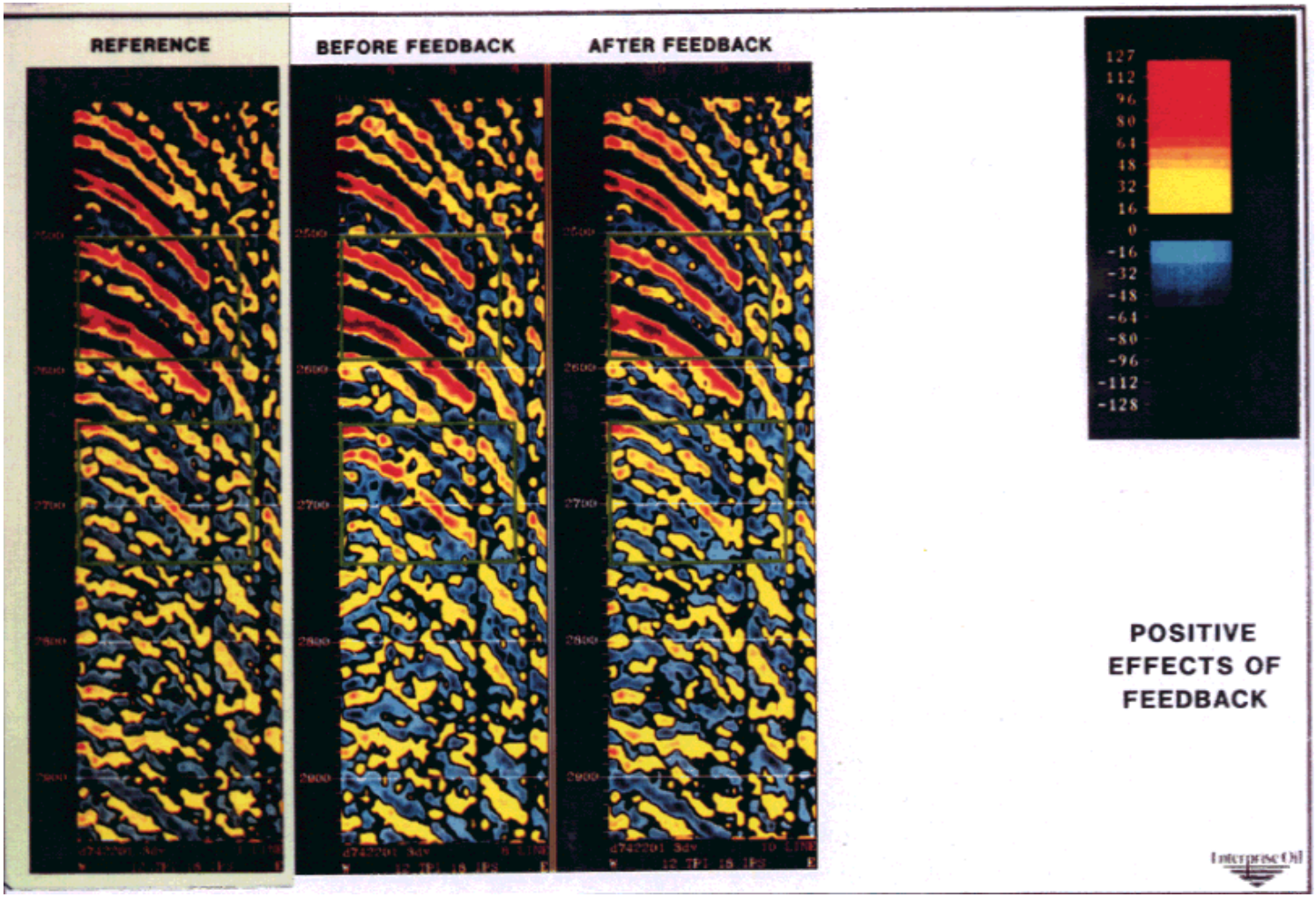


Figure 2.2

Defect example 2: feedback detail



Similarity v. coordinate: Feedback to company 3

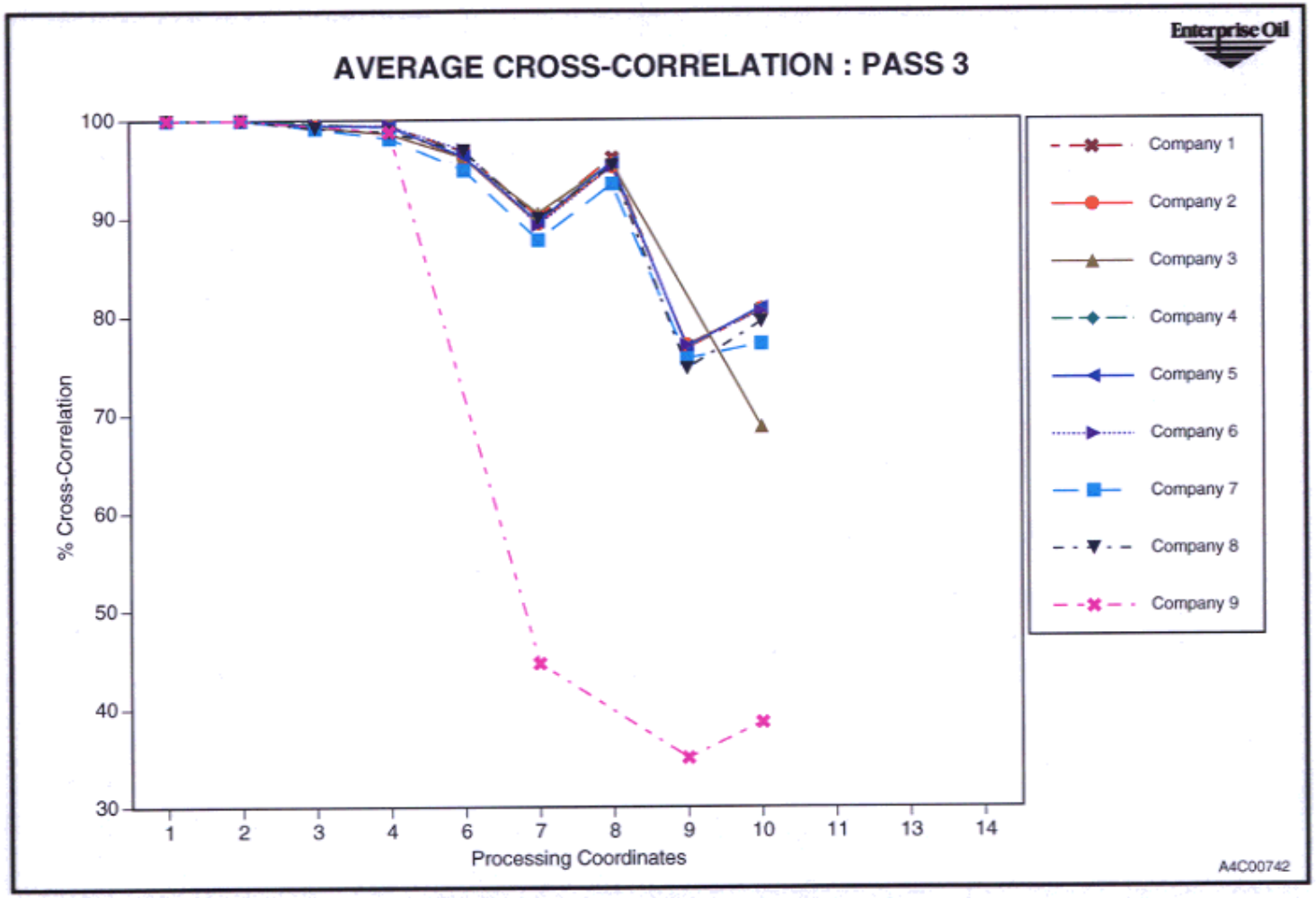
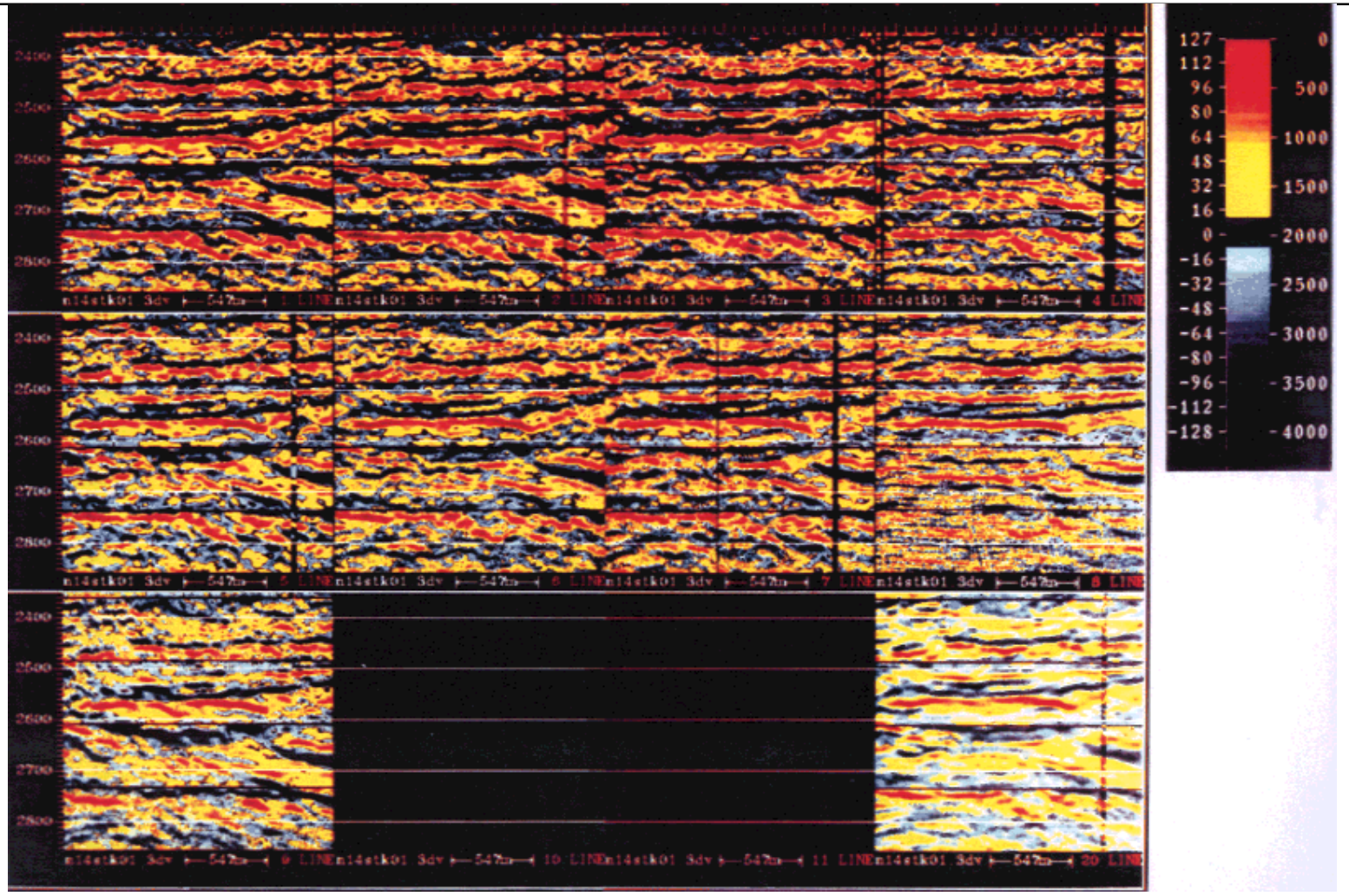


Figure 2.3

The end product: 9 subtly different views of the geology



T2 Results

- **The accompanying slides illustrate:**
  - Only 1-2 significant figures agreement after processing.
  - Disagreement is non-random and alternate views seem equally plausible
  - Feedback of anomalies along with other evidence confirms source of disagreement as software failure.

A summary of 10 years of failure experiments

Seismic processing software environment	Number of significant figures agreement
32 bit floating point arithmetic.	6
Same software on different platforms, same data.	4
Same software on same platform, 5-1 lossy compression.	3-4
Same software subjected to continual ' enhancement'	1-2
T2: different software, same specs, same data, same language, same parameters.	1

Overview

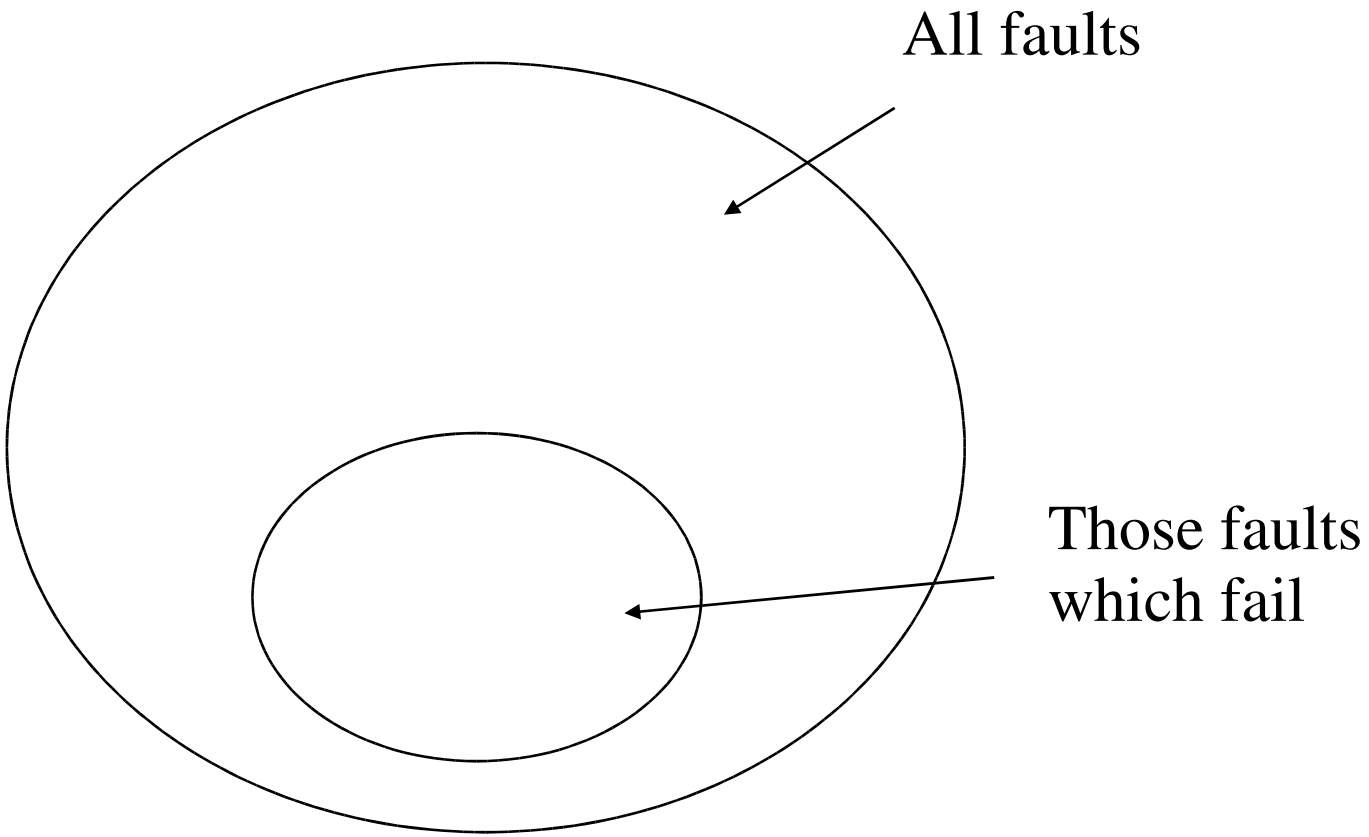
- **1984-1988: Portability experiments**
- **1988-1997: Fault experiments**
- **1990-1996: Failure experiments**
- **1996-1997: Correlating fault and failure**
- **1995-2000: Does paradigm shift help ?**
- **2001-: Some interesting questions**

1996-1997: Correlating fault and failure

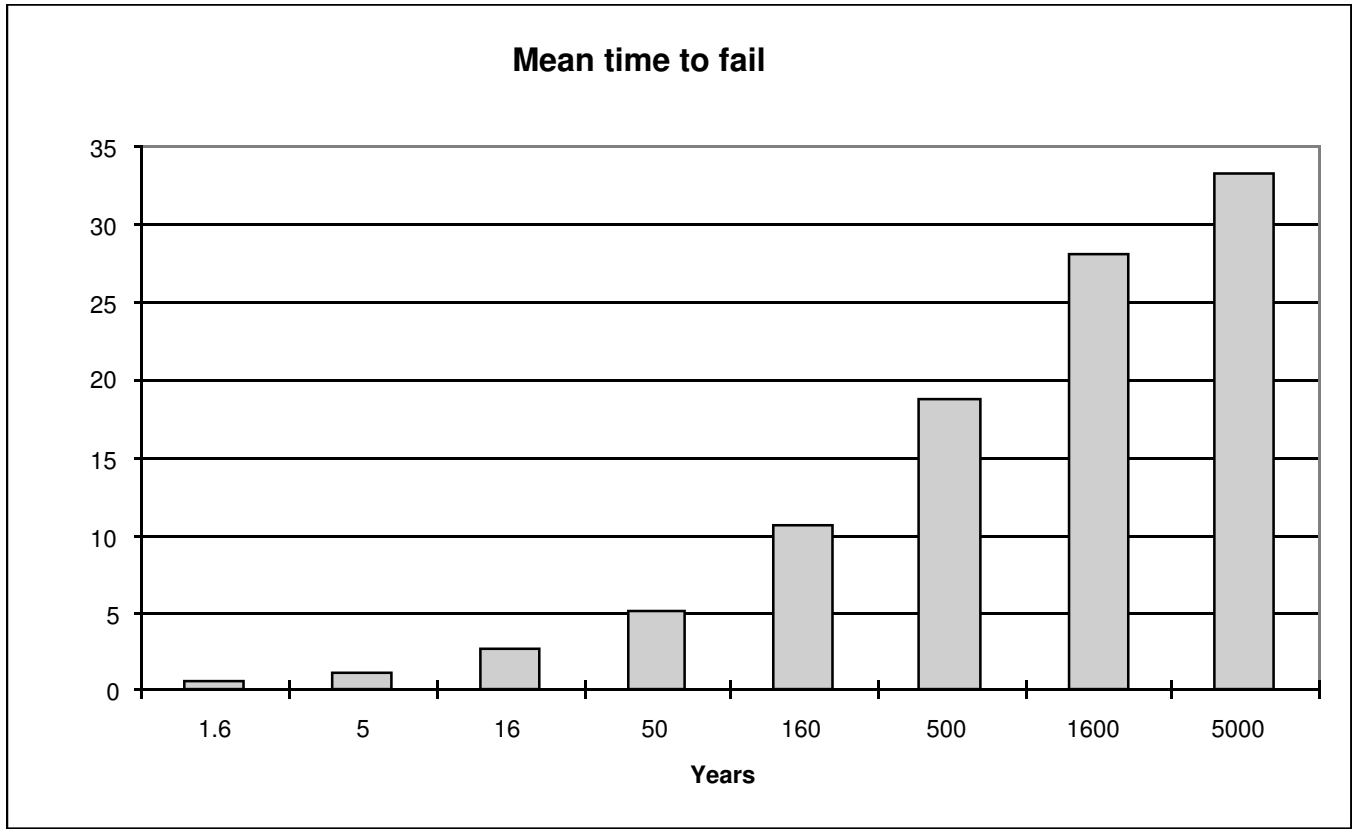
Stages

- How and when do faults fail ?
- Does Mathematics help - the Heathrow air-traffic control system

Where and how do faults fail historically ?



Mean time to fail in Adams (1984)

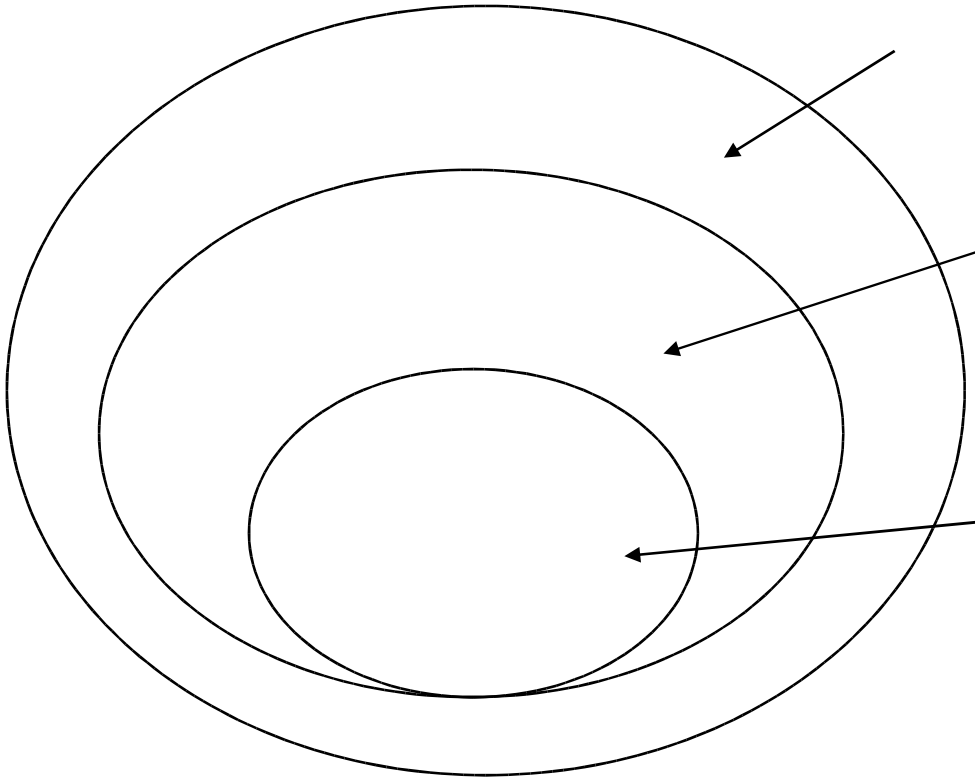


The balance between static and dynamic testing

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.

The balance between static and dynamic testing

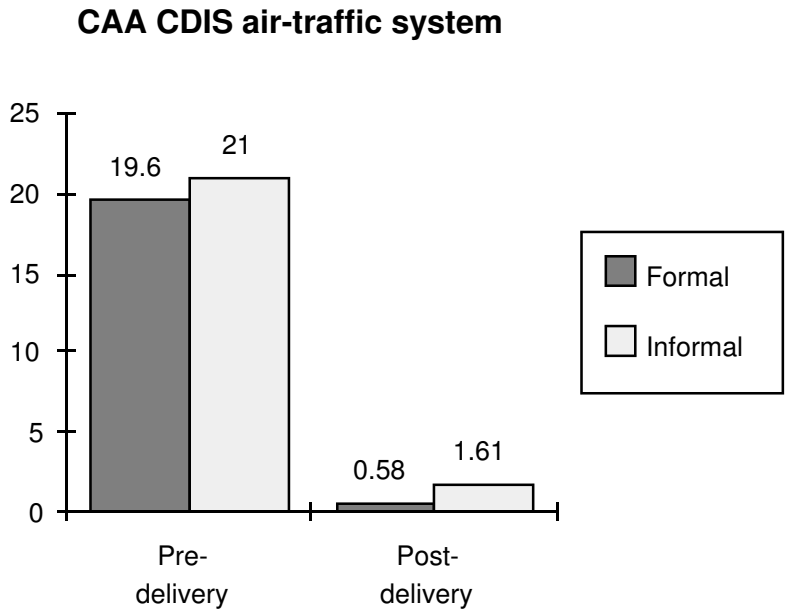


The balance between static and dynamic testing

Conclusion:

- For systems which are shipped in large numbers such as embedded control systems or web page software, static testing is even more important than for ordinary systems

Does mathematics help ?



Overview

- **1984-1988: Portability experiments**
- **1988-1997: Fault experiments**
- **1990-1996: Failure experiments**
- **1996-1997: Correlating fault and failure**
- **1995-2000: Does paradigm shift help ?**
- **2001-: Some interesting questions**

1995-2000: Does paradigm shift help ?

Points to consider

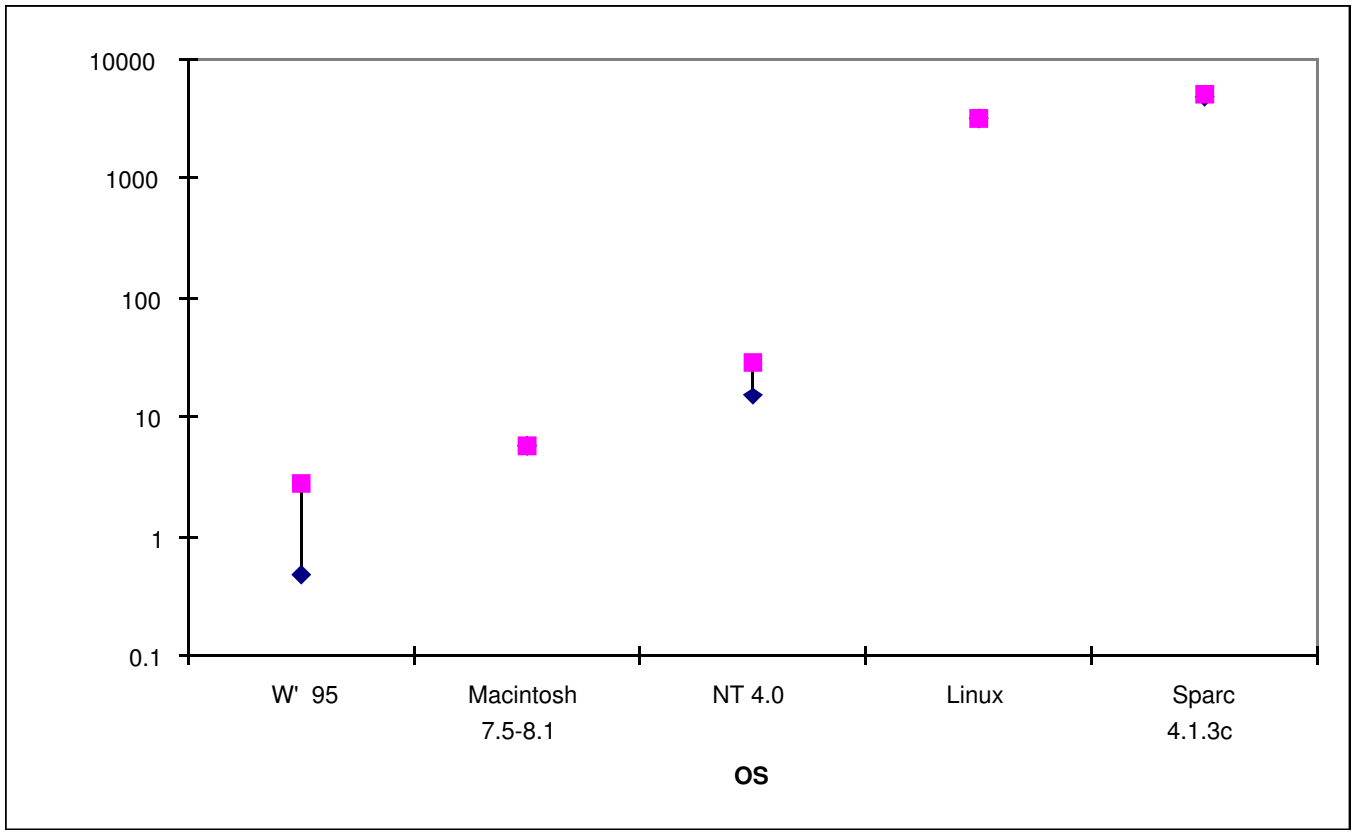
- Software Process
- Development paradigms, for example OO
- Control process feedback

Software Process

Points to consider

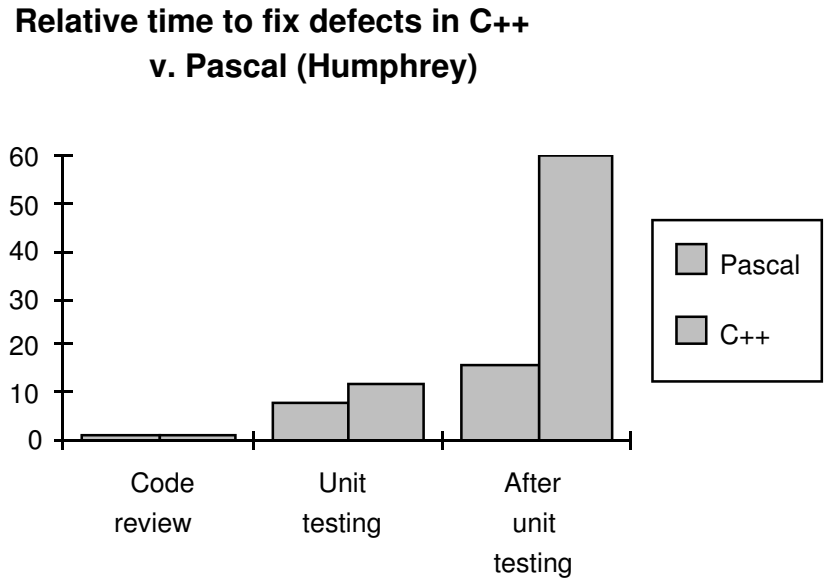
- There is an inherent belief that a good process implies a good product
- Why is Linux so good ?
  - Linux is categorically CMM level 1 so is the CMM wrong or does Open Source development have important properties that we don't understand well yet ?
  - Is the reliability of Linux incremental ?

Software Process and Linux

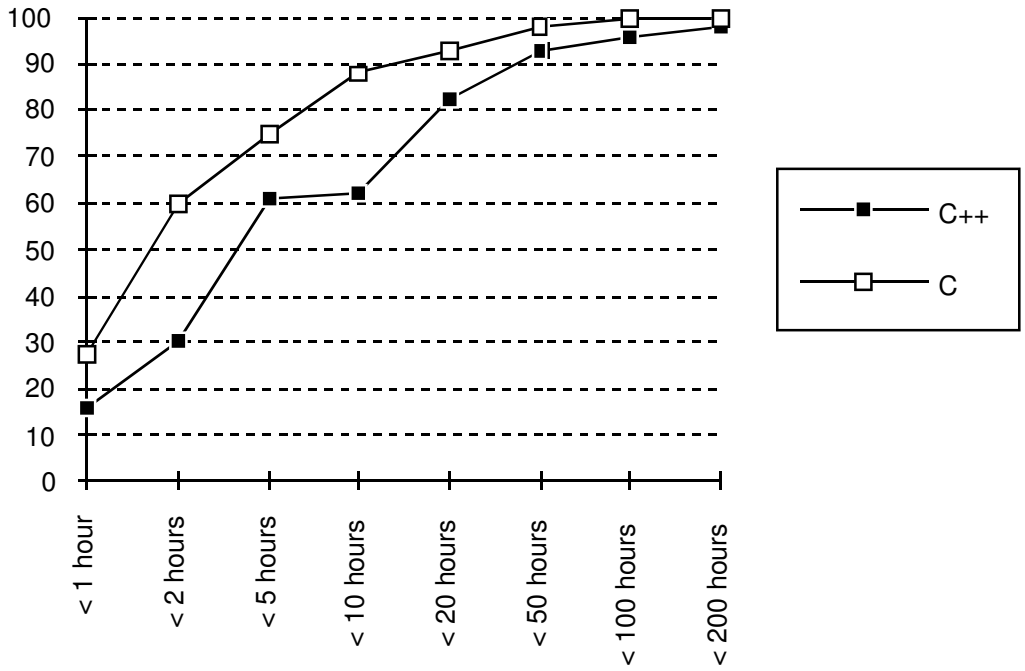


Mean Time Between Failures of various operating systems

Measurement feedback on OO development, (Humphrey)



Measurement feedback on OO development, (Hatton)



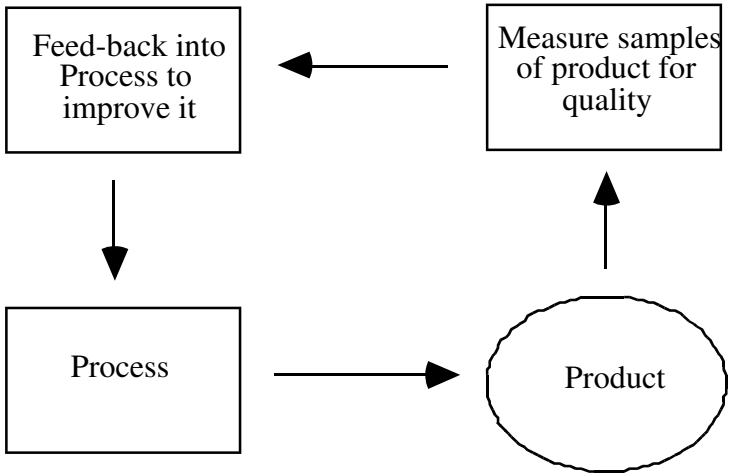
Measurement feedback on OO development

Summary of known measurements

- C++ OO systems have comparable defect densities to conventional C or Pascal systems.
- Each defect in a C++ OO system takes about twice as long to fix as in a conventional system. This is true for both simple defects AND difficult ones. The whole distribution is right shifted.
- Components using inheritance have been observed to have 6 times the defect density.

How much of this is attributable to C++ is unknown.

Control Process feedback - the essence of engineering improvement



If you want to improve reliability, measure and analyse failures.

Overview

- **Paradigm shift is characterised by:-**
  - Fashion / marketing focus
  - Creativity driven
  - The complete absence of measurement
  - Maximises things the engineer CAN do.
- **Control process feedback is characterised by:-**
  - Engineering focus
  - Measurement and analysis of failure
  - Ruthless elimination of known failure modes
  - Maximises things the engineer can NOT do.

Overview

- **1984-1988: Portability experiments**
- **1988-1997: Fault experiments**
- **1990-1996: Failure experiments**
- **1996-1997: Correlating fault and failure**
- **1995-2000: Does paradigm shift help ?**
- **2001-: Some interesting questions**

2001-: Some interesting questions

Points to consider

- Can we reverse or even halt linguistic decay ?: Aristotleans v. Babylonians
- Why do defects cluster and/or why are they not linearly distributed ?
- Necessary and unnecessary complexity

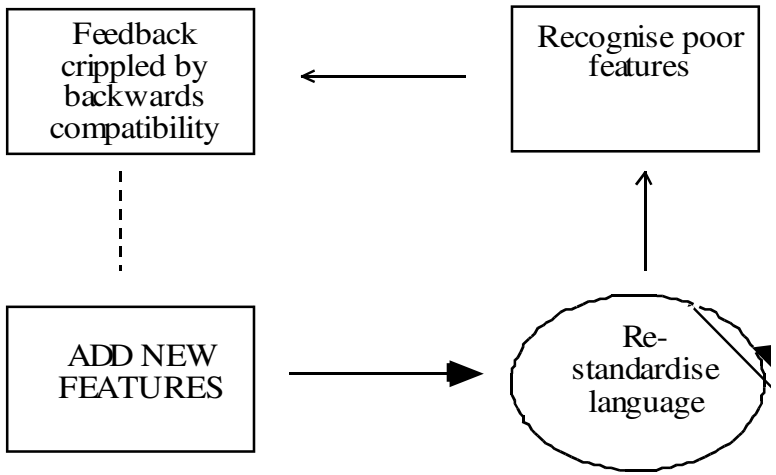
Linguistic decay

In my career, I have been forced to write programs in:-

- Focal
- Atlas Autocode
- Algol
- Assembler
- Fortran 66, 77
- C
- Pascal
- Ada (briefly)
- C++
- Java
- Various scripting languages, Perl, Tcl/Tk, Bash, Javascript
- C again, (this time from choice)

12 changes in 32 years; average is 32/12 = 2.7 years

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.

An example: C itself

Type of poorly-defined behaviour	ISO C90	ISO C99
Unspecified	22	49
Undefined	97	191
Implementation-defined	76	111
Locale-specific	6	15
Total	201	366
Defect reports	119	???

Even new languages struggle:-

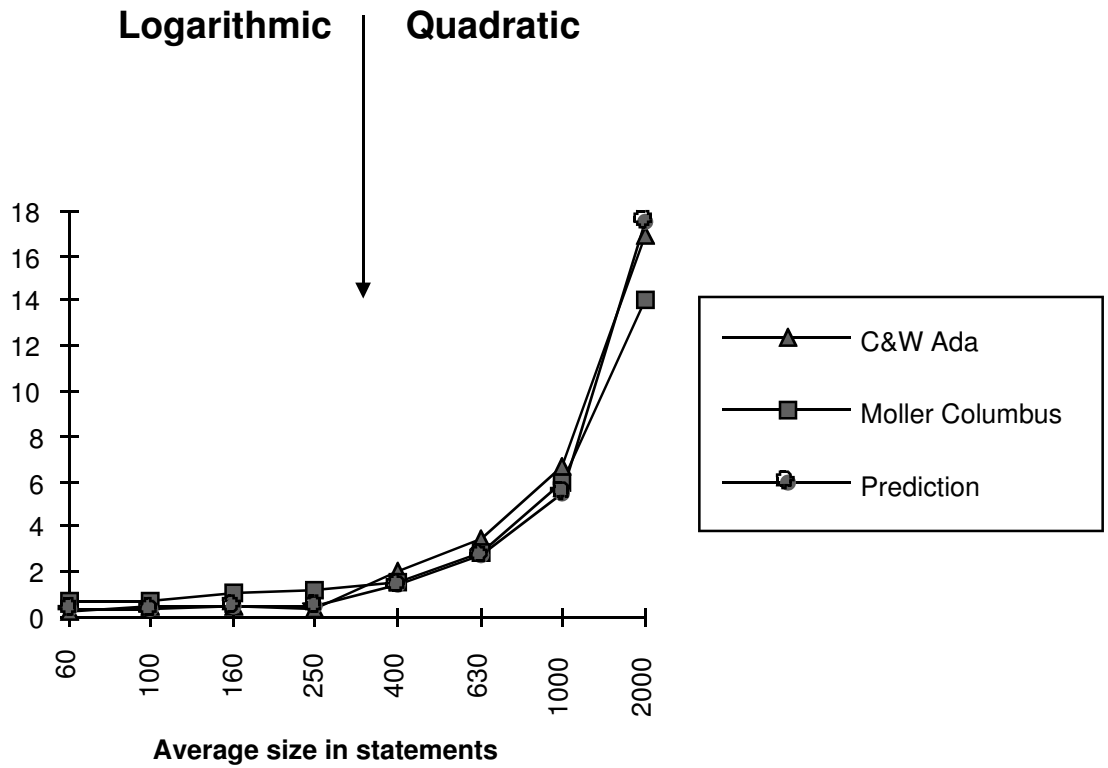
All the following languages inherit at least some of C's built-in defects, often most:-

- C++: In ISOC++99, we also find the words:-
  - Undefined, 1825 times
  - Unspecified, 1259 times.
- Javascript: Even precedence was not defined explicitly
- Java: Removed some defects, added some new ones
- Perl: 21 levels of precedence ...

IEC 1131: another new standard:

- Removes need to declare variables first 'for programmer flexibility'

Defect clustering



Note the non-linear growth. Why does it grow so slowly ?

Necessary and unnecessary complexity

- **In the Knight-Leveson (1986) experiment:-**
  - 27 versions of the same algorithm were developed independently in Pascal
  - The smallest had around 300 lines and the largest was over 1000 lines.
  - The most reliable did not fail in 1 million trials, the least reliable failed nearly 10,000 times.
- **AT&T in the '70s and '80s:-**
  - it was frequently observed that rewriting the same algorithm 2 or 3 times reduced the size by about the same factor, e.g. diff.

Summary

To conclude:

- On the negative side
  - We are ignoring systematic errors in our software *and* known ways of detecting them
  - Our languages do not seem to be improving. They just change
  - There is too little measurement based feedback
  - Different defect types have different signal-to-noise
- On the positive side
  - There are some exciting possibilities for improvement

References

- Hatton, L. et. al. (1988). “SKS: an exercise in large-scale Fortran portability”, Software Practice and Experience, 18(4), p. 301-329.
- Hatton, L. and Roberts A. (1994) “How accurate is scientific software”, IEEE TSE, 20(10), p. 785-797.
- Hatton, L. (1995) “Safer C: Developing for High-Integrity and Safety-Critical Systems. McGraw-Hill, ISBN 0-07-707640-0.
- Hatton, L. (1997a) Re-examining the fault density - component size connection, IEEE Software, March-April 1997.
- Hatton, L. (1997b) The T experiments: errors in scientific software, IEEE Computational Science & Engineering, vol 4, 2
- Hatton, L. (1998) Does OO sync with the way we think ?, IEEE Software, May/June 1997
- Humphreys, W. (1995) “A discipline of software engineering”, Addison-Wesley, ISBN 0-201-54610-8
- Kinght, J.C. and Leveson, N. (1986) “An experimental evaluation of the assumption of independence in multi-version programming”, IEEE TSE, 12(1), p. 96-109
- Pfleeger, S and Hatton L. (1997) “Investigating the influence of formal methods”, IEEE Computer, 30(2), p. 33-43.