" Forensic Software Engineering:
avoiding the avoidable and living with the rest"

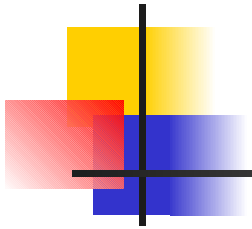
by

Professor Les Hatton

CIS, University of Kingston
L.Hatton@kingston.ac.uk, lesh@oakcomp.co.uk

Version 1.1: 23/Sep/2004

Overview



Principles

Scope

Conclusions

An observation

- All the evidence suggests that many if not most failures exhibited by software controlled systems could have been avoided by techniques we *already* know how to apply.

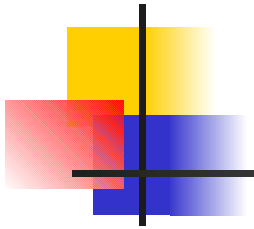
The two engineering obligations

- When a system fails (and it will), it should be designed in such a way as to minimise deleterious effects on its user by means of built-in redundancy or otherwise
- When a system fails, the diagnostic system should *always* be able to provide an efficient means for finding the corresponding fault or faults so they can be corrected.

How to get it wrong
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.

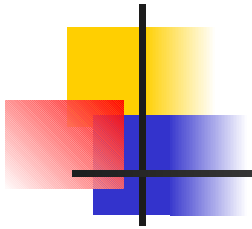
How to get it wrong:
Flight management system



An example from real life, Airbus A320 AF319, 25/8/88, (Mellor (1994)):-

- MAN PITCH TRIM ONLY, followed in quick succession by ...
- Fault in right main landing gear
- Fault in electrical flight control system computer 2
- Fault in alternate ground spoilers 1-2-3-5
- Fault in left pitch control green hydraulic circuit
- Loss of attitude protection
- Fault in Air Data System 2
- Autopilot 2 shown as engaged when it was disengaged
- LAVATORY SMOKE

Overview

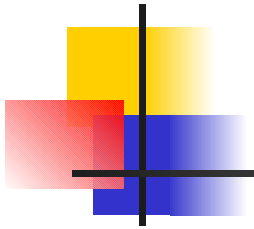


Principles

Scope

Conclusions

Scope



Forensic Software Engineering encompasses:-

- Forensic Process Analysis
  - Project failures
- Forensic Product Analysis
  - Implementation failures (e.g. linguistic), test failures …
- Forensic Systems Analysis
  - OS reliability, security, environment failures (eg arithmetic), compiler quality, implications for design …

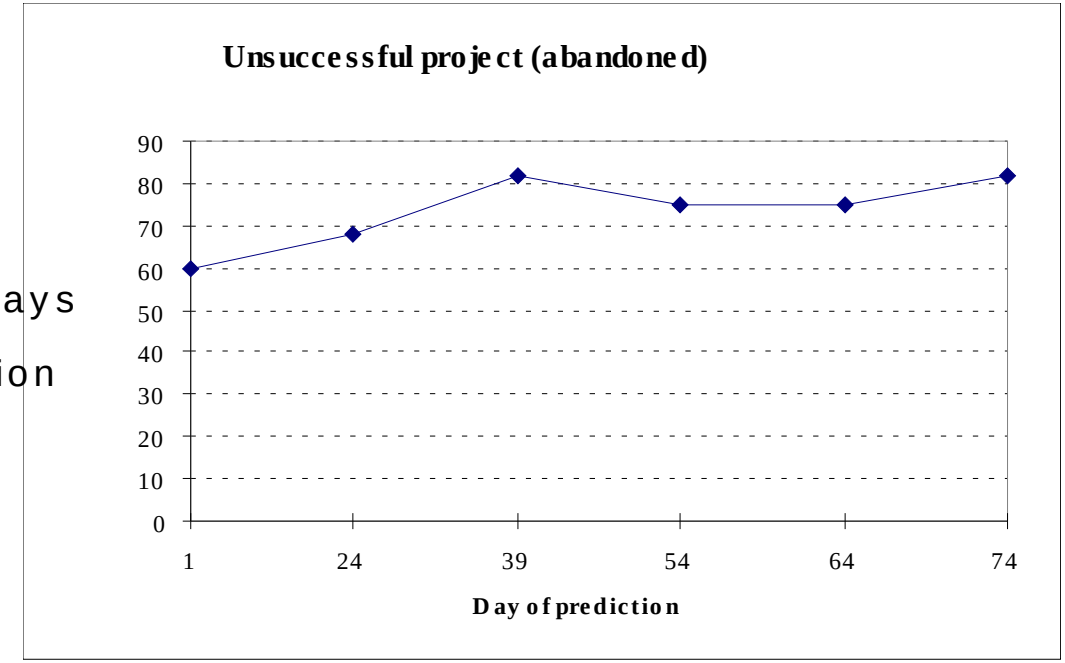
In each area we are trying to answer the question "Why?" to avoid future occurrences

Forensic Process Analysis

"Planning is an unnatural process. Its much more fun to get on with it. The real benefit of not planning is that failure comes as a complete surprise and is not preceded by months of worry."

Sir John Harvey Jones.

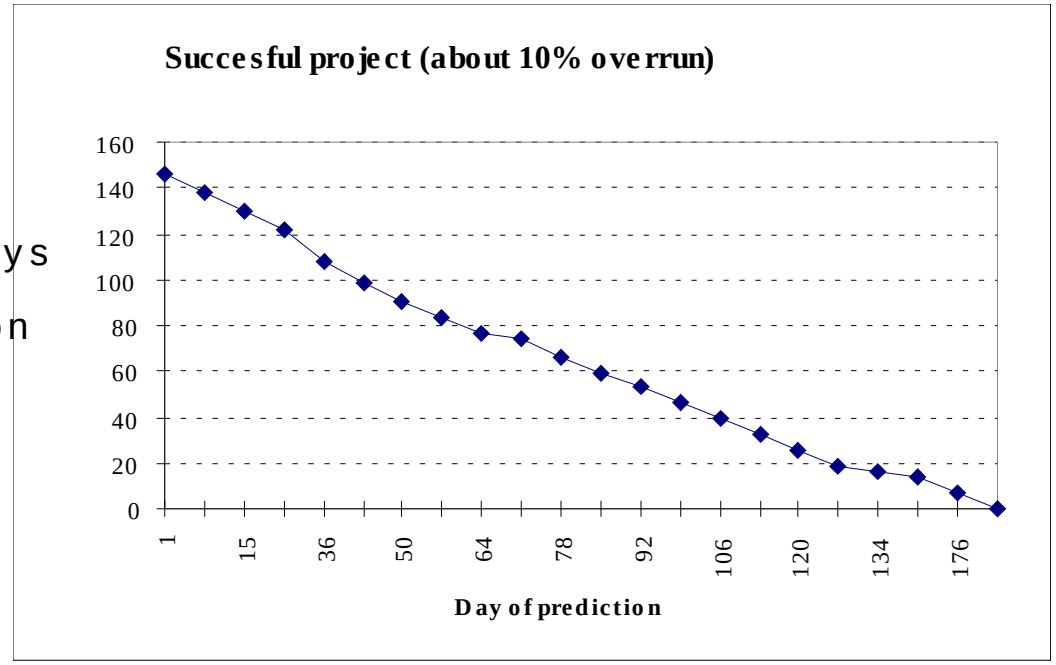
When the train of ambition pulls away from the platform of reality



Planning data from a grand **‘ unified’** programming project.
(Produced after the project seemed to be struggling.)

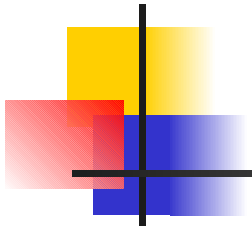
Note that *unify* appears next to *unintelligible* in the OCD.

Ruthlessly controlling tasks



Project restarted with (far) less ambitious goals and tracked weekly with results published on staff notice board.

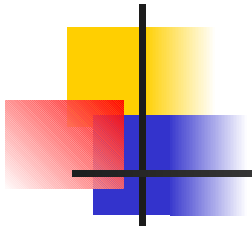
Forensic Process Analysis:
results so far



The following are necessary (but may not be sufficient) for satisfactory project planning:-

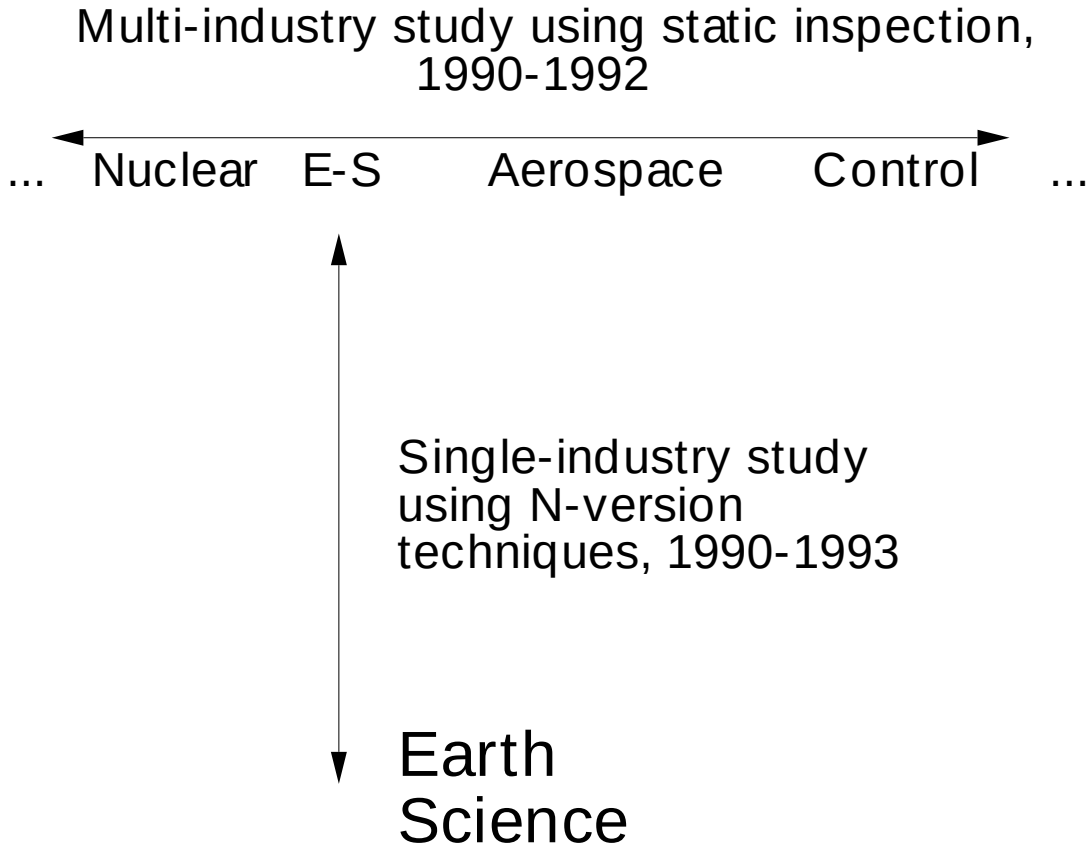
- No sub-task with software systems should be longer than 1 week
- Projects should be tracked weekly with progress published
- Programmers underestimate the time taken to do things by about 50%

Forensic Product Analysis:

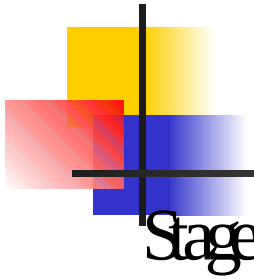


Here we are essentially analysing the software product itself to understand the nature of its failures.

The T-experiments



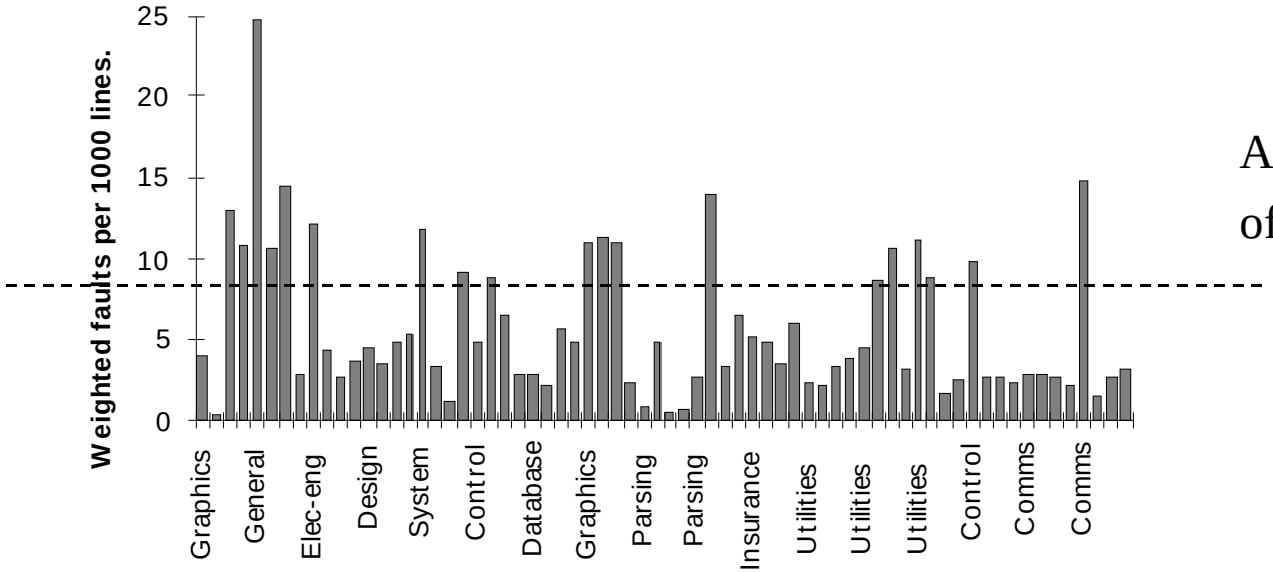
1988-1997: The T1 Fault experiments



Stages

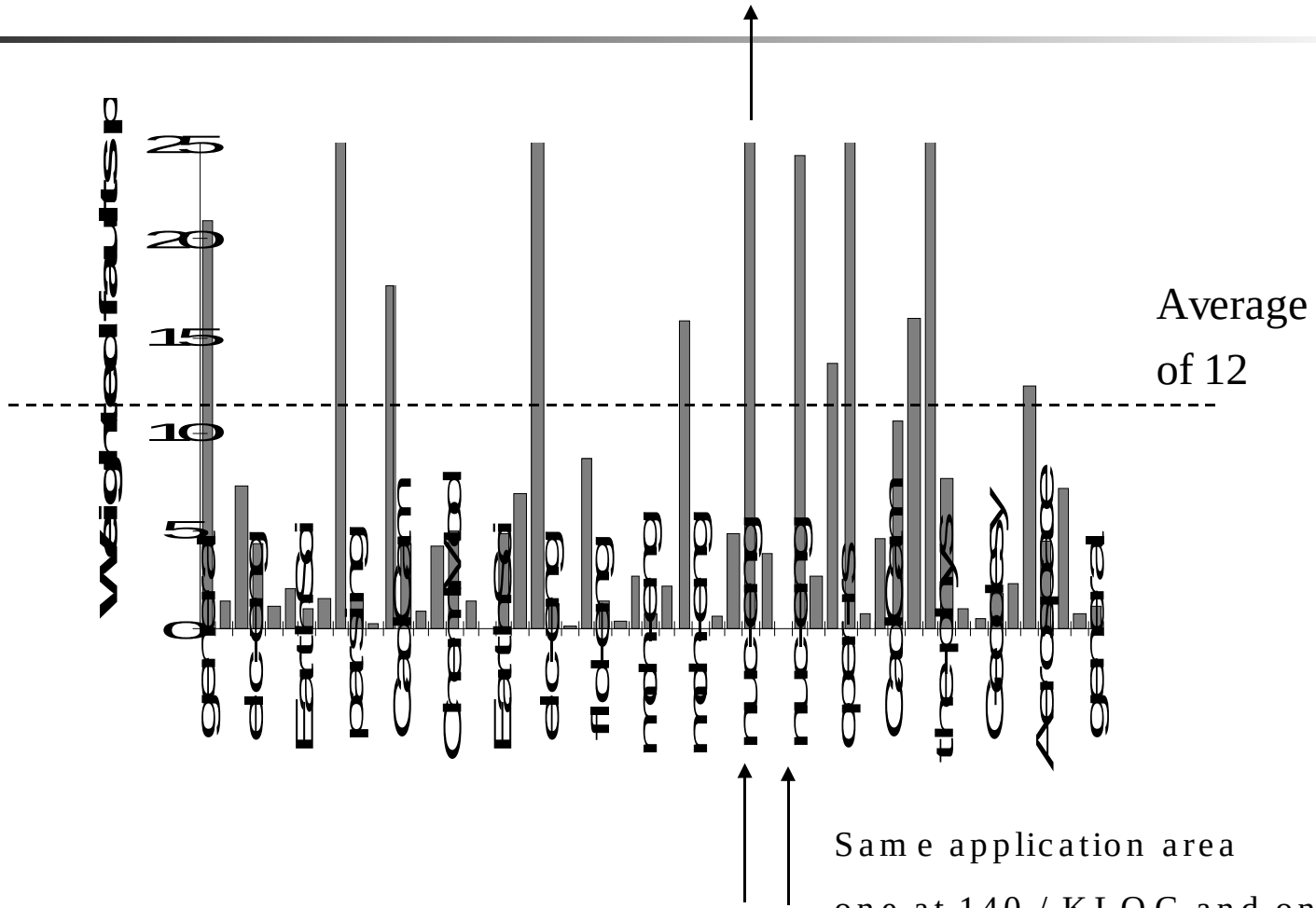
- Observed many repeating faults in development of SKS
- Developed F77 parsing engine to study other packages, 1988-1992
- Developed C parsing engine to study similar problems in different language, 1990-1994
- Measured around 100 major systems 1988-1997
- Developed more advanced C parsing engine 1996-2000,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

Fault frequencies in C applications

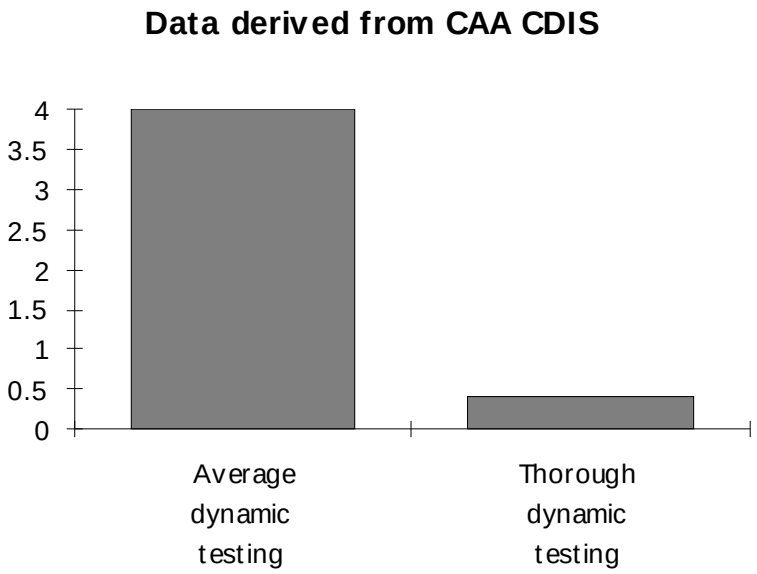


Average of 8

Fault frequencies in Fortran 77 applications

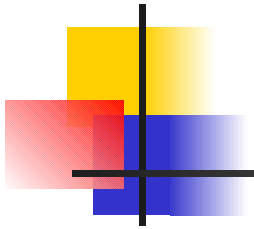


Where and how do faults fail historically ?



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

1990-1996: Failure experiments

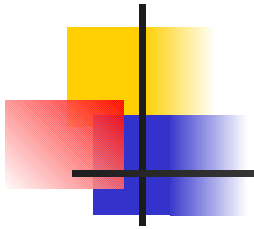


Stages

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

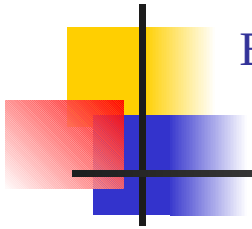
9 independently developed commercial versions of same ~750,000 F77 package of signal processing algorithms.

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:

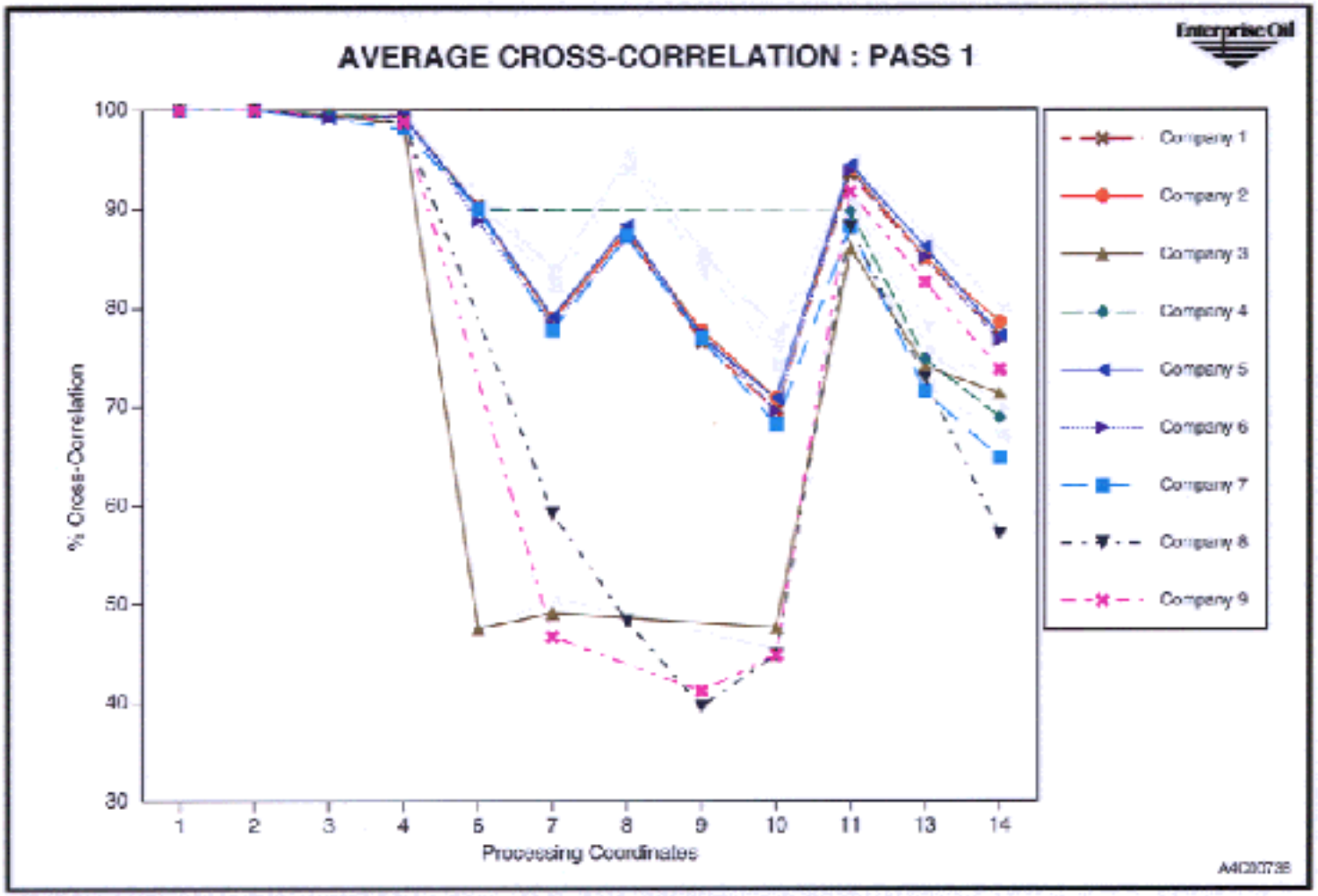
- To estimate the magnitude of disagreement.
- To see what form disagreement took.
- To identify poorly implemented processes.
- To attempt to improve agreement by feedback confirming nature of fault.
- To preserve complete confidentiality.

Data analysis

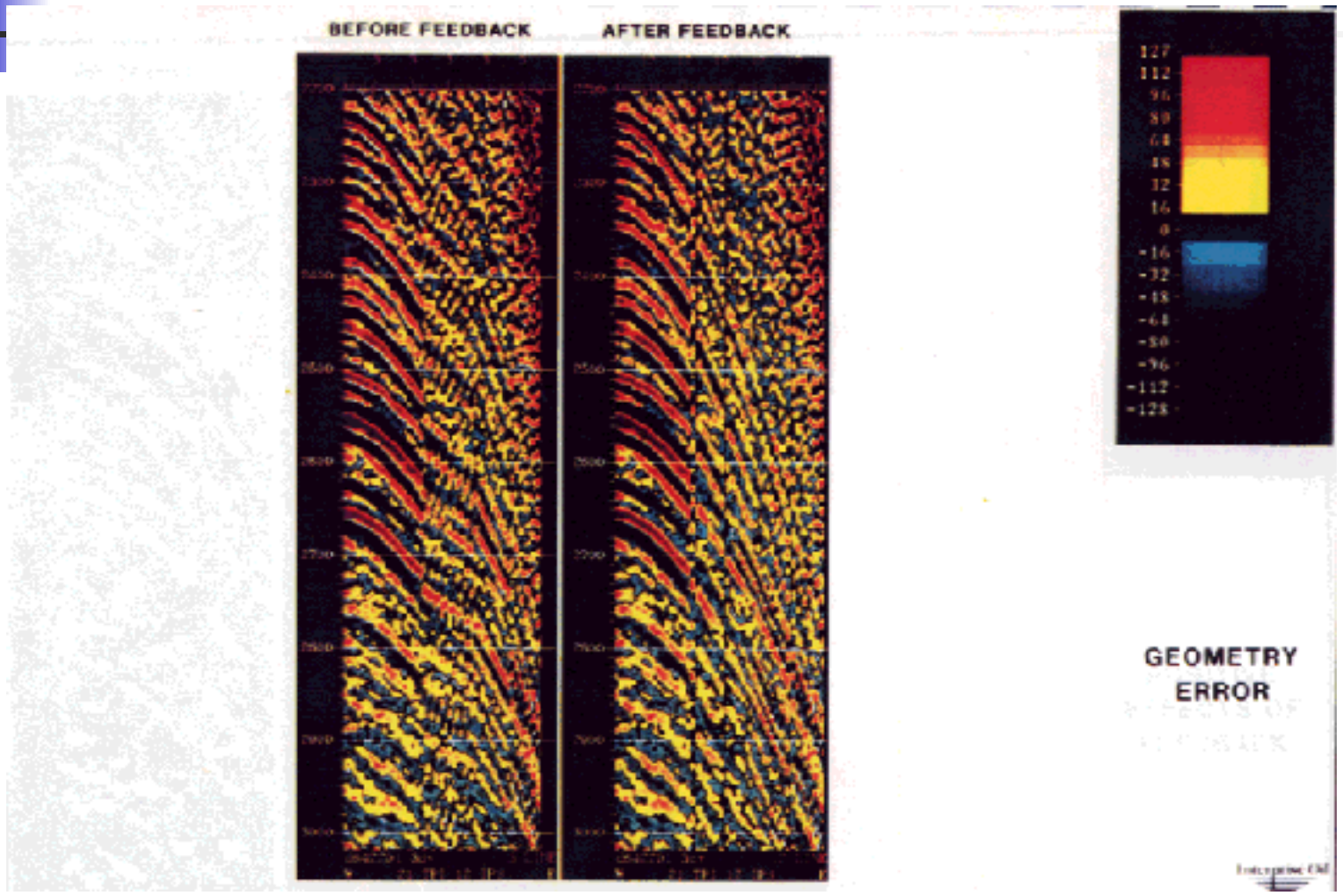
Analysis goals were:

- 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.

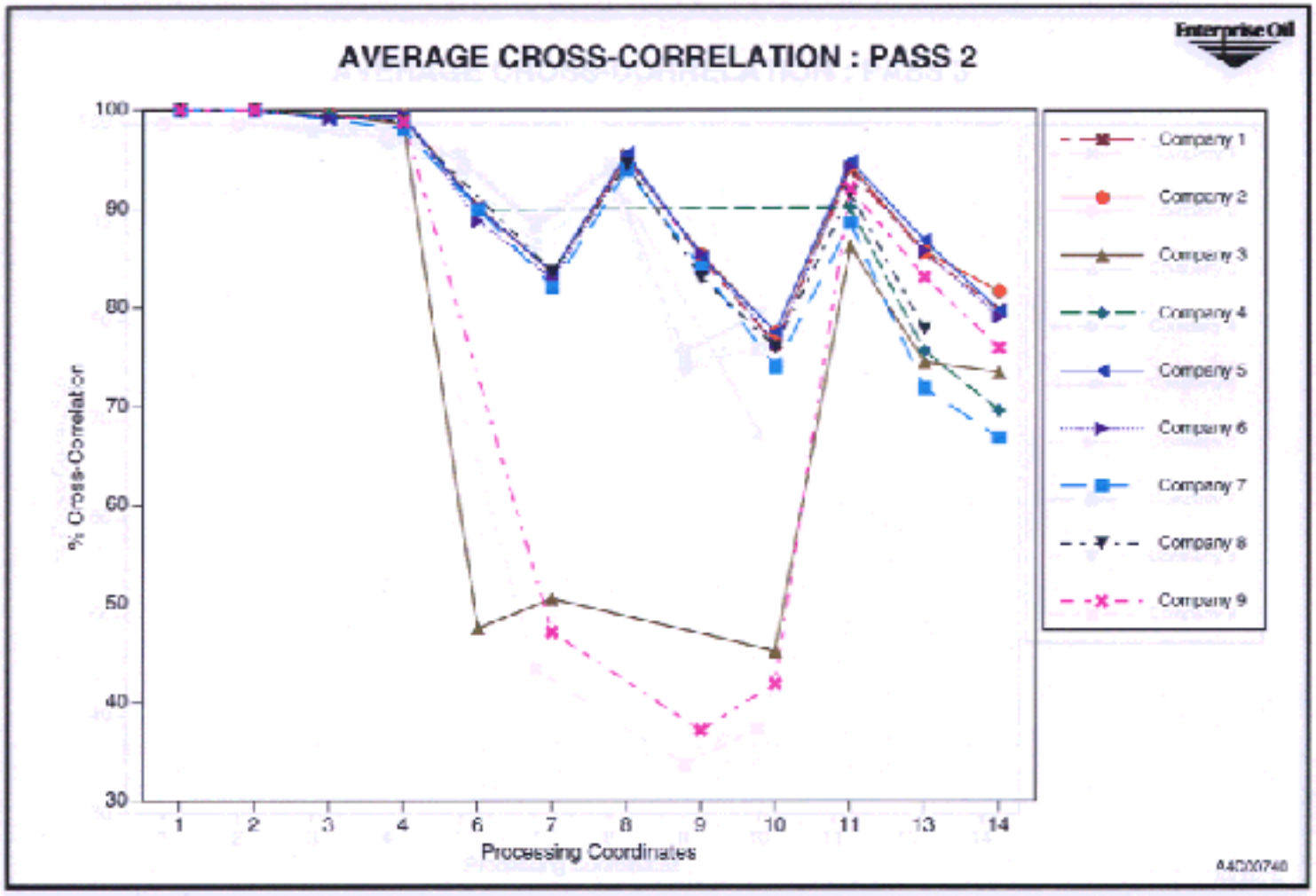
Similarity v. coordinate: No feedback



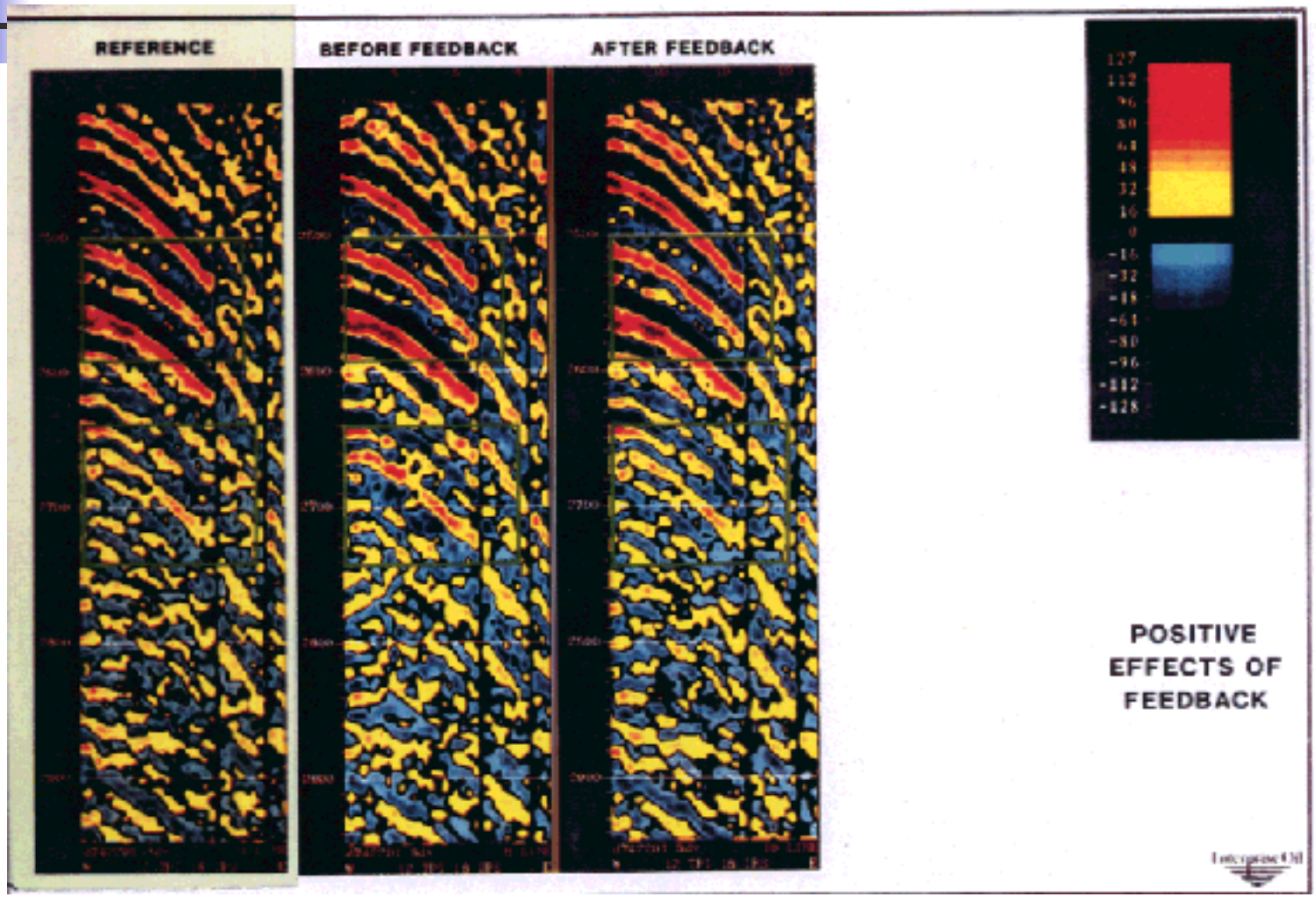
Defect example 1: feedback detail



Similarity v. coordinate: Feedback to company 8



Defect example 2: feedback detail



Similarity v. coordinate: Feedback to company 3

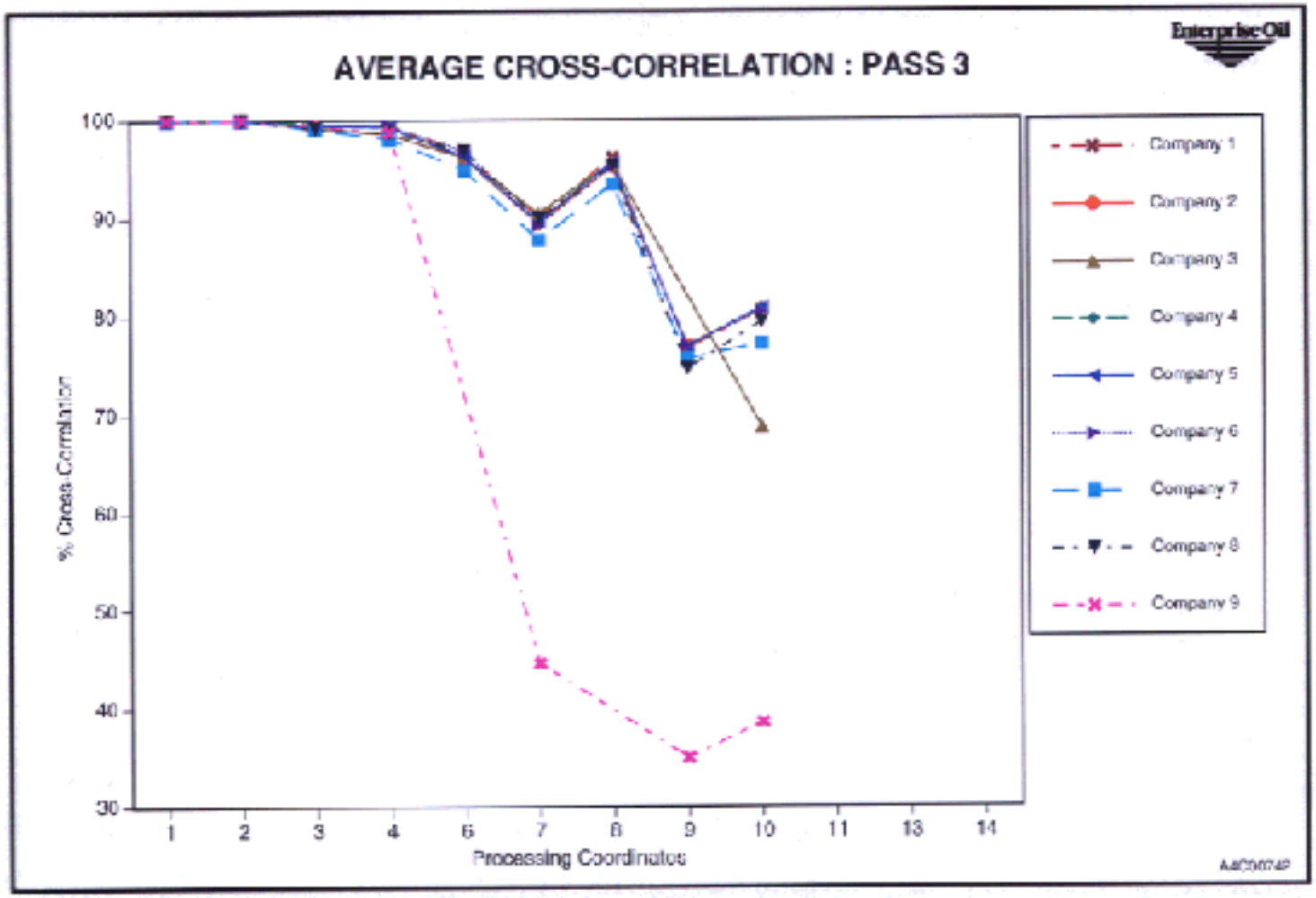
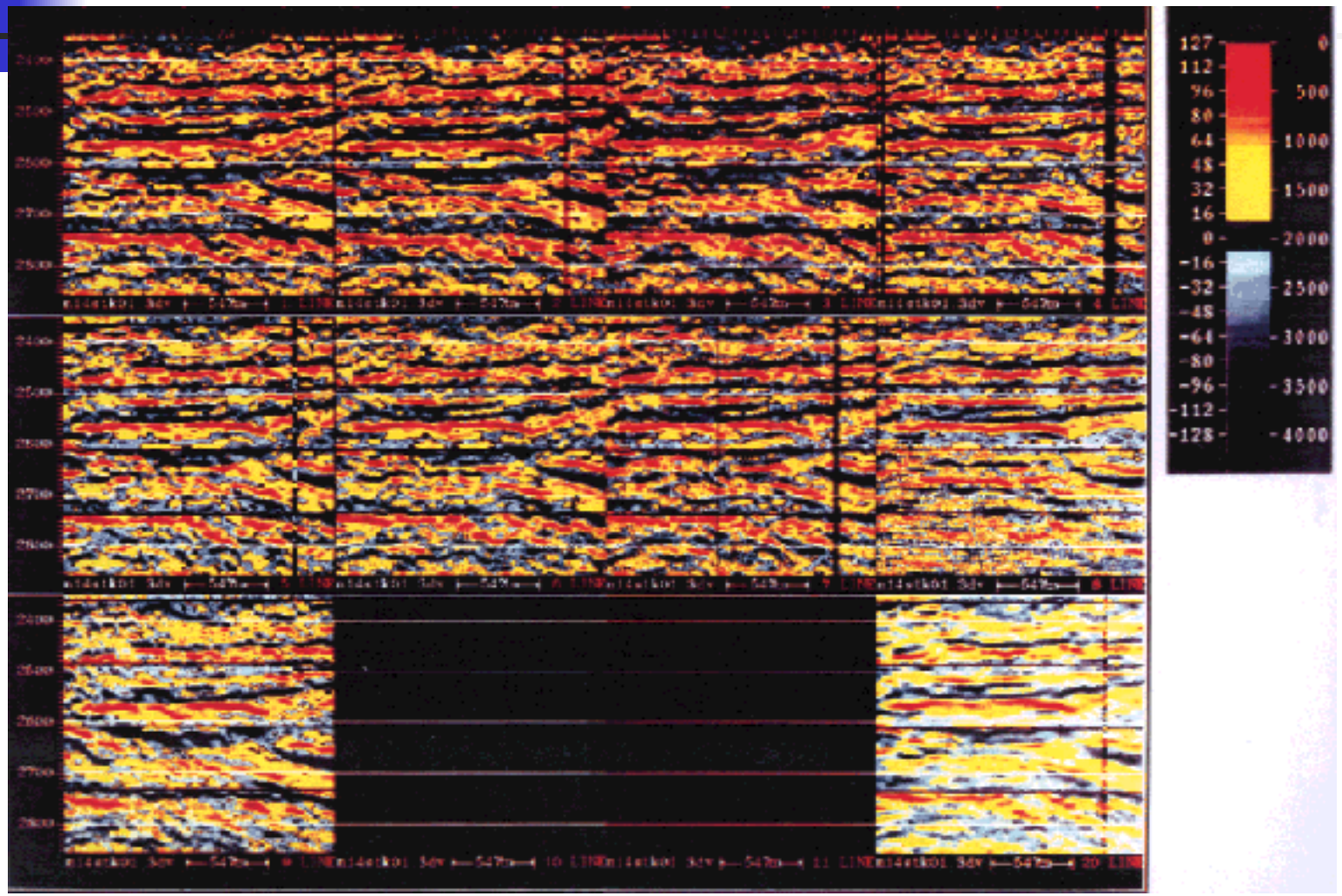
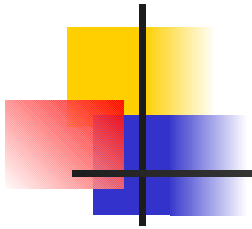


Figure 2.3

The end product: 9 subtly different views of the geology



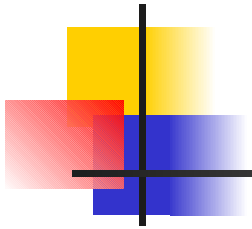


T2 Results

The accompanying slides illustrate:

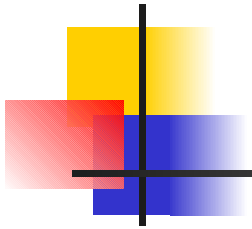
- Only 1-2 significant figures agreement after processing.
- Disagreement is non-random and alternate views seem equally plausible
- Feedback of anomalies along with other evidence confirms source of disagreement as software failure.

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

Forensic Product Analysis:
results so far

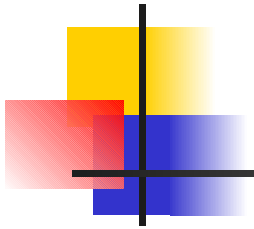


The following seem well supported

- Modern programming languages are riddled with poorly defined behaviour which programmers regularly fall prey to
- Even in high-quality environments, scientific software results degrade quite rapidly with the amount of computation even though the results may look plausible
- The choice of programming language is irrelevant, it is the fluency of the programmers in that language which matters most.

Forensic Systems Analysis:

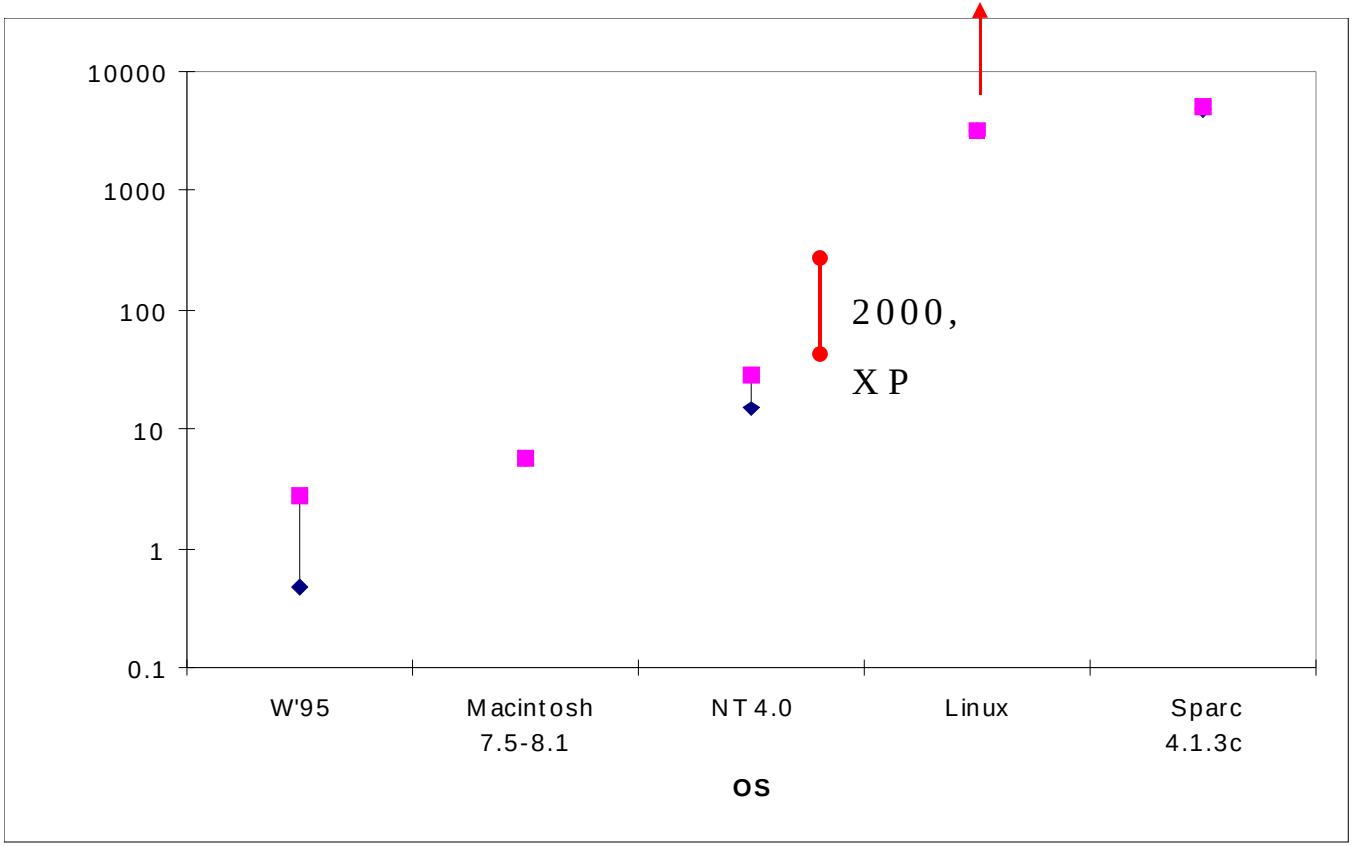
Here we are essentially analysing the systems environment in which software functions to understand the nature of its failures.



We can identify at least the following areas:

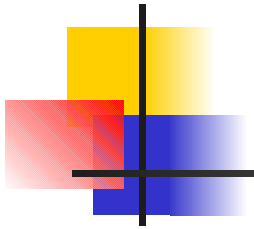
- OS reliability
- Security
- Arithmetic environment
- Compiler quality

OS Reliability



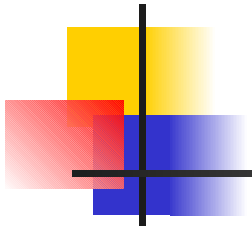
Mean Time Between Failures of various operating systems

Security



A very big subject which includes:-

- Monolithic v. modular design
- The use of binary format files
- How permissions and users are defined
- Software failures, (many security breaks are due to buffer overflow caused by programmers using inappropriate functions, (e.g. strcpy instead of strncpy))

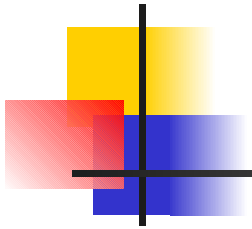


Even in 2004, computers still get arithmetic wrong:-

- Embedded System Paranoia extends the venerable *paranoia* to embedded control systems with similar results:-

  http://www.leshatton.org/ESP_903.html

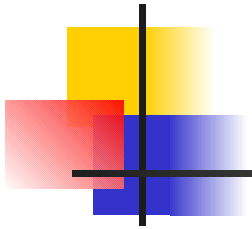
Compiler Quality



Note the following:-

- In April 2000, NIST formally stopped validating compilers *in any language*
- Most compilers fail the existing validation suites in some way or another – these departures are not documented, you assume the risk
- It seems very likely that the situation will get worse as languages get more complicated

Forensic Systems Analysis:
results so far



The following seem well supported

- If you need a reliable OS environment (MTBF > 500 hours) do not use Windows
- If you need a secure OS environment do not use Windows or binary file formats
- Test your computer arithmetic, it will probably have inconvenient flaws and may have major failures
- Do not take your compiler quality for granted. Seek written assurances from the supplier if possible.

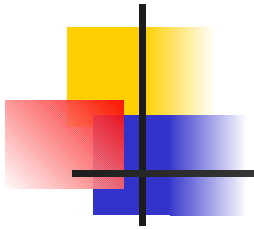
Overview

Principles

Scope

Conclusions

Conclusions

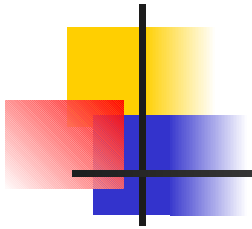


Forensic Software Engineering seeks:

- To analyse failures in various categories and to disseminate this information in a searchable way to allow developers and scientists to avoid future occurrences of these failures.

All the evidence suggests that relatively simple use of avoidance strategies can lead to extraordinarily reliable applications

Reference site



For more information, downloadable papers and software, see:-

http://www.leshatton.org/

l.hatton@kingston.ac.uk, lesh@oakcomp.co.uk