"Paradigm shift or measurement based feedback?"

JACC' 99, Oxford, 18th Sept, 1999

by

Les Hatton

Oakwood Computing, Surrey, U.K.

and

Computing Laboratory, University of Kent, UK

lesh@cybershed.co.uk

Version 1.0: 08/Sep/1999

Overview

- <u>Introduction</u>
- Evidence for repetitive failure
- Preventing repetitive failure
- Paradigm shift or measurement ?
- Other concerns - diagnosis

A growing problem

The amount of software in consumer electronic products is currently doubling about every 18 months.

- Line-scan TVs have ~250,000 lines of C.
- There are around 200,000 lines of C in a car.
- Most consumer devices, washing-machines and so on have a few K of software.
- The Airbus A340 and Boeing 777 are totally dependent on software, (more later...).

Problems - real or imagined ?

Airbus A340 G-VAEL 16/09/94, Heathrow:

- Software error calculated fuel incorrectly.
- Both screens went blank, " Please wait ..." .
- Plane turned right when instructed to turn left.
- Plane descended at 9 degrees when instructed to descend at 3 degrees.

Not all of these yet believed fixed.

Problems - real or imagined ?

and ships ...:

- The ultimate high-tech USS Yorktown was left dead in the water off the coast of Virginia for an hour on a weekend in September, 1997 due to software failure
- The ship had to be rebooted.
- (Perhaps there should be an international flag signal for " Please wait ...").

Problems - real or imagined ?

Cars too ...:

- 22/July/1999. General Motors has to recall 3.5 million vehicles because of a software defect. Stopping distances were <u>extended</u> by 15-20 metres.
- Federal investigators received almost 11,000 complaints as well reports of 2,111 crashes and 293 injuries.
- Recall costs ? (An exercise for the reader).

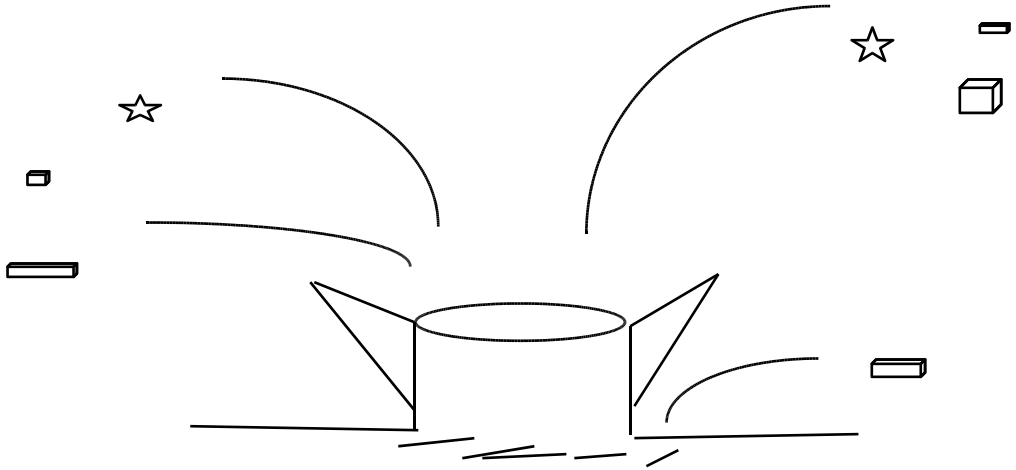
not forgetting ...

Ariane 5 :-(

How Ariane 5 failed ...

Actuator (steering) software actually written in Ada, but it is equivalent to the following C code:

```
short    s;
double   d;
...
s  = d;
```



Anatomy of another big bug ...

In Ariane 5, the following sequence occurred:-

- A 64-bit floating point number was forced into a 16-bit integer in a **non**-critical component
- When it overflowed, the programming language (Ada) generated an exception which was allowed to shut down the system including critical components for actuator control
- The system had twin-channel redundancy but the same software was running in each channel.
- If a sloppier language had been used, there would have been no crash

The PC picture ...

System	Defect Rate	% reboots
W'95 + Office	1every 42 minutes	28%
Mac OS + Office	1 every 188 minutes	56%
Unix (Software development)	< 1 per year	100%
Linux (Various)	None experienced in 2 years	-

A few more recent vignettes ...

- **Aug. 1999:** MCI Worldcom's international frame relay network suffers major software collapse.
- **Aug. 1999:** ERNIE, (UK premium bond generator) had software defects causing thousands of genuine winners not to be paid anything.
- **Sep. 1998:** LTCM, a financial house in New York lost 44% of fund value due to defects in computer model September, 1998.
- **Jul. 1998: Northern Examination Board, UK:** The exam results of 100,000 students were affected by defects in the marking software, July 1998

Overview

- Introduction
- <u>Evidence for repetitive failure</u>
- Preventing repetitive failure
- Paradigm shift or measurement ?
- Other concerns - diagnosis

An observable fact

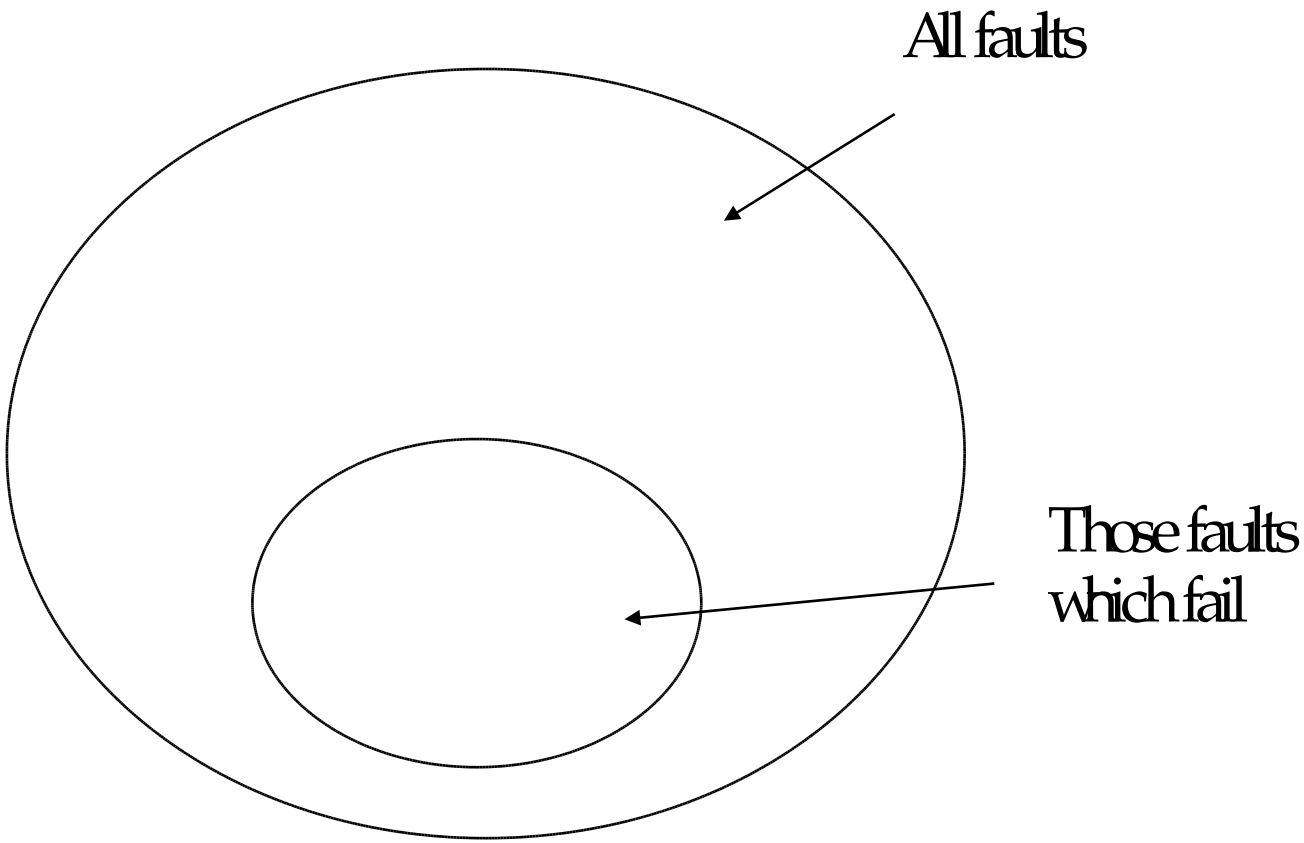
Software systems are unique amongst engineering systems in that their behaviour is dominated by repetitive failure.
Why ?

Definitions of fault and failure

Software failure is characterised by:-

- Fault
  - Static property (identified before run-time)
  - Can be identified in either design or source code as a ‘ mistake’
  - Not all faults fail
- Failure
  - Dynamic property (identified at run-time)
  - Defined to be difference between actual and expected run-time behaviour.
  - Every failure is caused by at least one fault.

The relationship between fault and failure



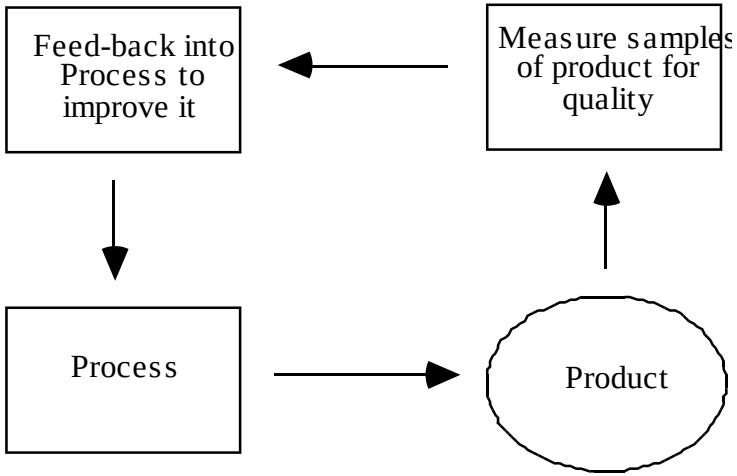
How do faults lead to failure ?

- **Ed Adams of IBM (1984) found that**
  - ~33% of all faults only failed < once every 5000 execution years
  - The most common failures, (> once every 5 years) were caused by only 2% of the faults.
  - Any correction had about a 15% chance of introducing a problem at least as big into the system.
- **Pfleeger and Hatton (1997) found (amongst other things) that:-**
  - static faults and dynamic failure were highly correlated in a high reliability system.

Overview

- Introduction
- Evidence for repetitive failure
- <u>Preventing repetitive failure</u>
- Paradigm shift or measurement ?
- Other concerns - diagnosis

Control Process feedback - the essence of engineering improvement

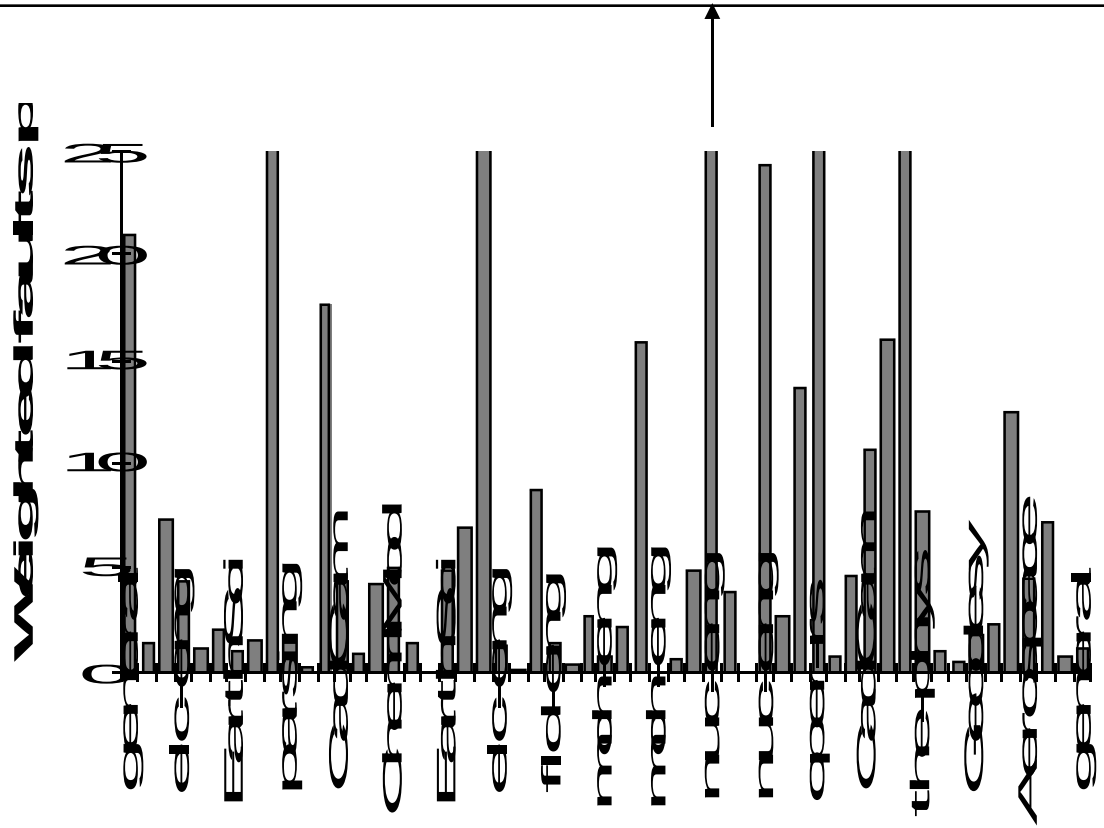


Language standardisation ...

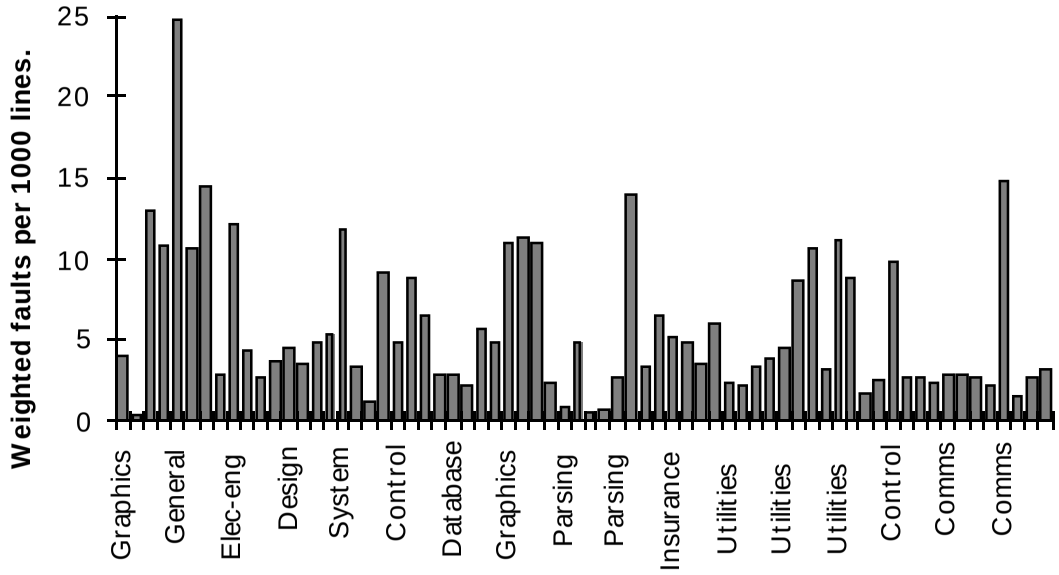
- **Language standardisation disobeys control process feedback in several important ways:-**
  - It is characterised by unconstrained creativity
  - It completely ignores measurement
  - The 'must not break old code' rule means feedback is crippled so although things are continually added, nothing ever gets taken out in practice.
  - It is a political as well as a technical process. Daft things get left in.

A control process engineer would conclude that languages can never get better.

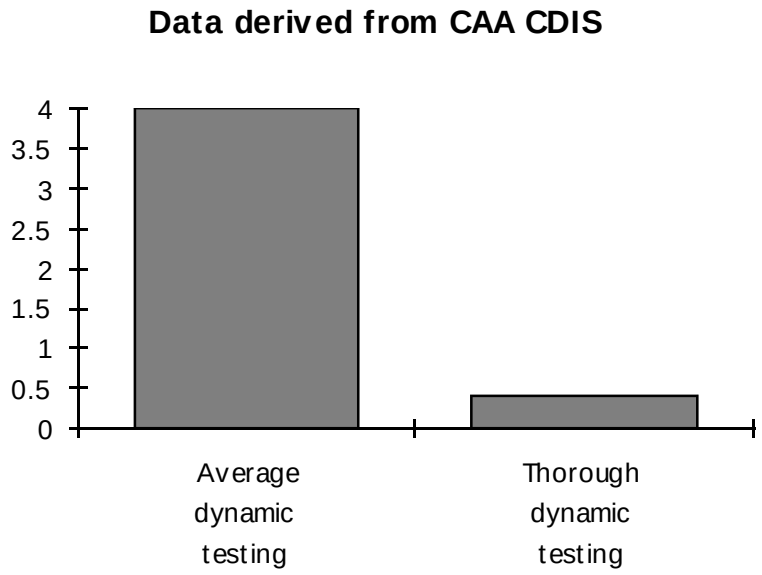
Occurrence rates in F77 applications



Dependence on known fault modes in C applications



Failure of statically detectable faults



Overview

- **Introduction**
- **Evidence for repetitive failure**
- **Preventing repetitive failure**
- **<u>Paradigm shift or measurement ?</u>**
- **Other concerns - diagnosis**

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

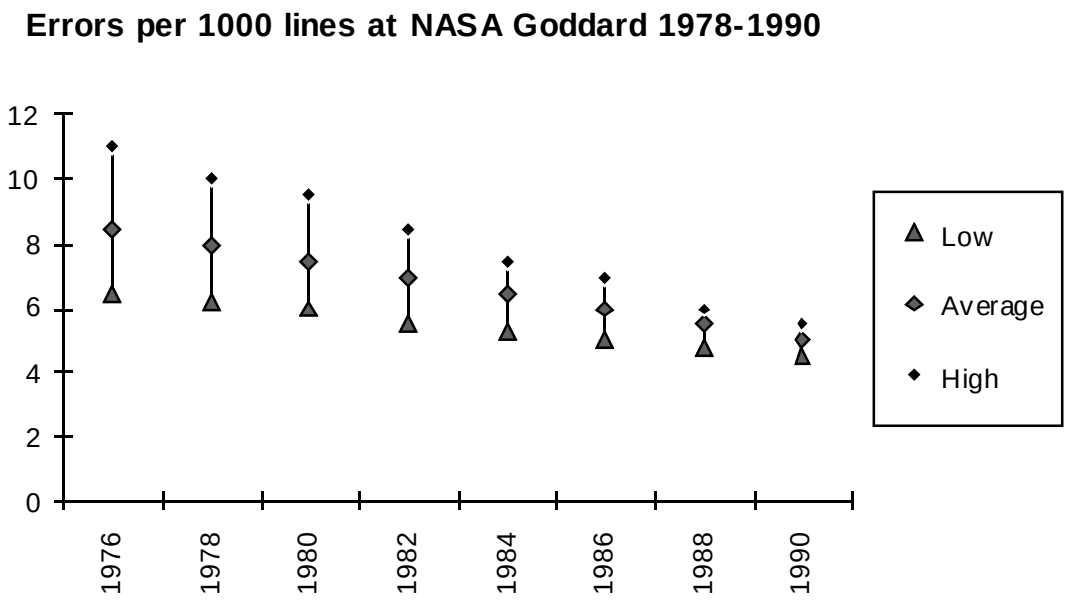
- **Three measurement-driven lessons:-**
  - Defect density is relatively independent of programming language
  - Defect density is almost constant with time in the last two decades
  - Independent implementations of the same algorithm in the same programming language can vary in size by factors of at least 3.

1. Defect density is relatively independent of language

Part of a worldwide survey of defect densities:

Source	Language	Errors / KLOC	Formal methods used	Life-cycle
Siemens - operating systems	Assemblers	6-15	No	Release
IPL - language parser	C	20-100	No	Dev. only
NAG - scientific libraries	Fortran	3	No	Release
Praxis - Air-traffic control	C	1.25	Yes	Release
Lloyds - language parser	C	1.4	Yes	Release
IBM cleanroom	Various	3.4	Part	Release
IBM normal	Various	30	No	Release
Loral - IBM MVS	Various	0.5*	Part	Projected
Basili & Perricone (1984)	Fortran	6-16	No	Release
Compton & Withrow (1990)	Ada	2-9	No	Release

2. Defect density is relatively constant with time



3. Independent implementations vary dramatically

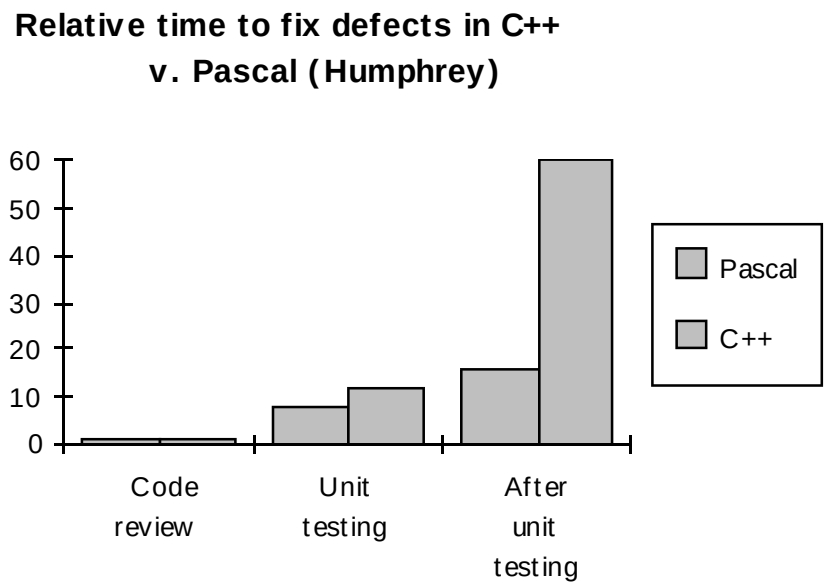
- **In the Knight-Leveson (1986) experiment:-**
  - 27 versions of the same algorithm were developed independently in Pascal
  - The smallest had around 300 lines and the largest was over 1000 lines.
  - The most reliable did not fail in 1 million trials, the least reliable failed nearly 10,000 times.

The OO paradigm shift

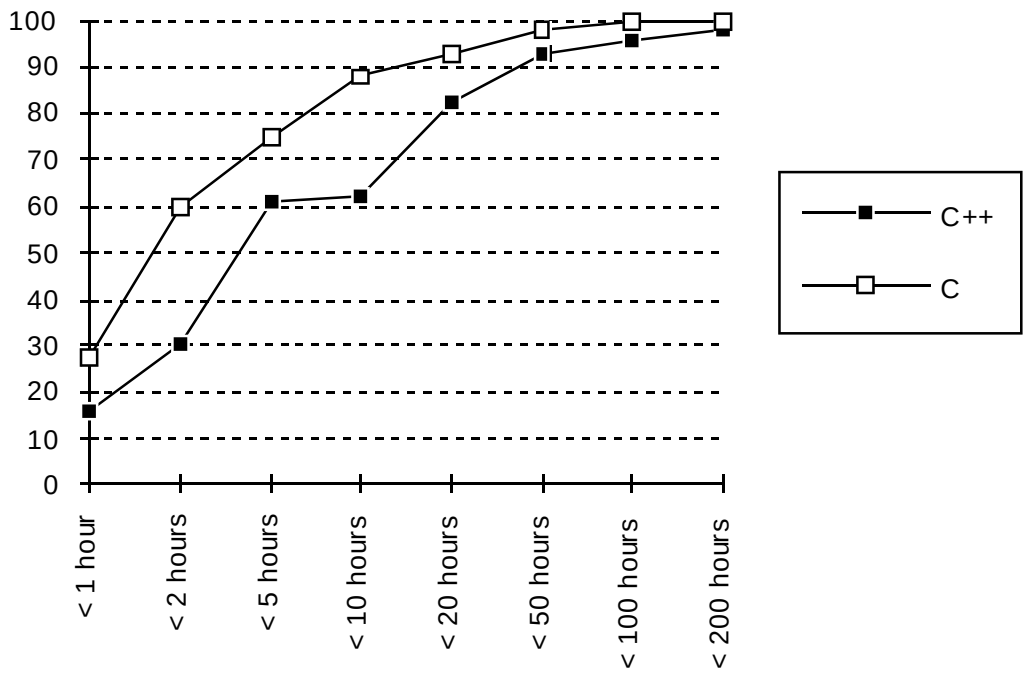
- **Object-Orientation and its implementation languages are a true paradigm shift characterised by:-**
  - Great certainty that it is ‘ the way to go’
  - Almost no measurement
  - Enormous marketing

  It may well be the way to go but without measurement based feedback, we simply don’ t know. Here are some recent measurements:-

Measurement feedback on object-orientation, (Humphrey 1995)



Measurement feedback on object-orientation, (Hatton, 1998)



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 between 2 and 3 times longer to fix than 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, (Shepherd & Cartwright, 1998).

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

Overview

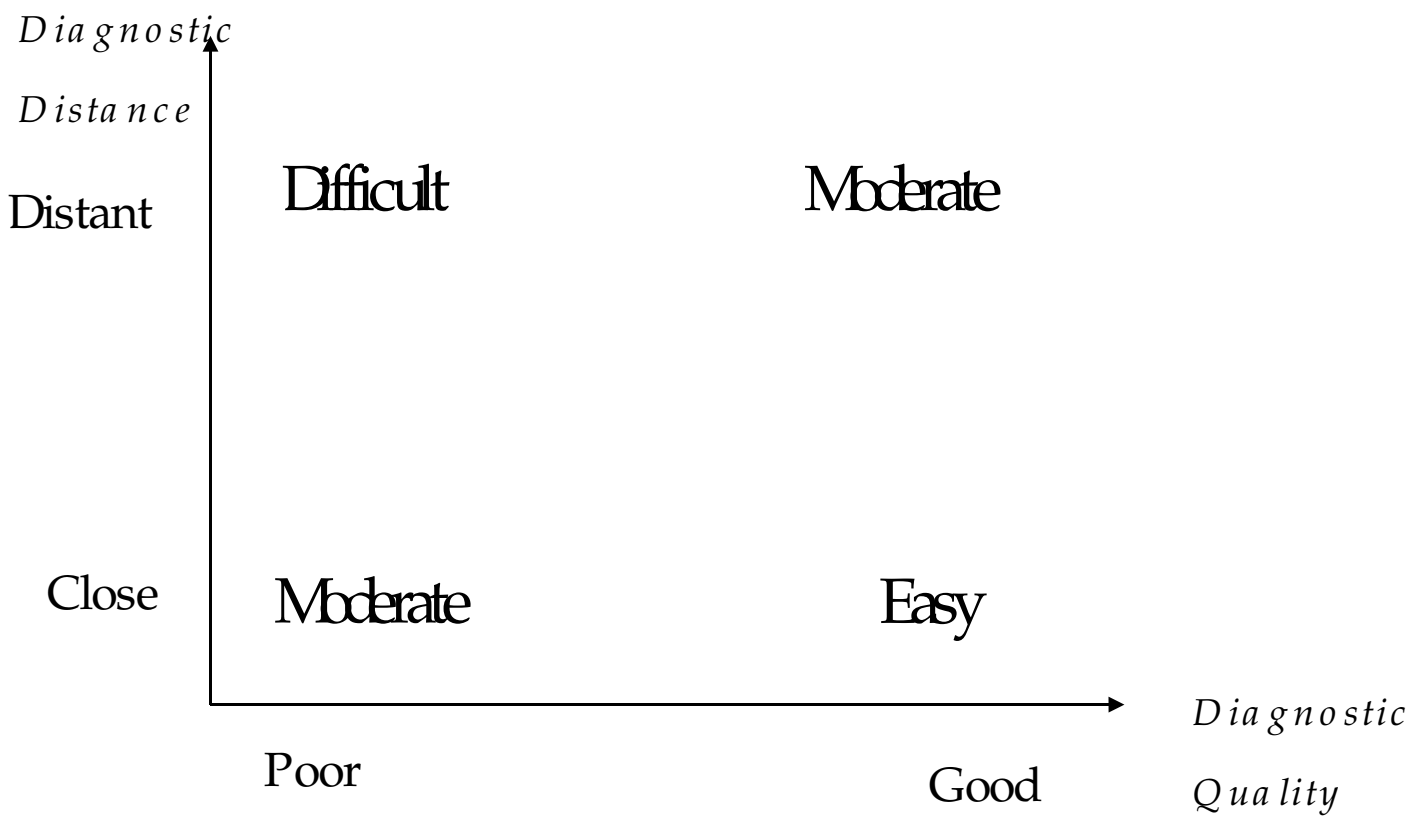
- **Introduction**
- **Evidence for repetitive failure**
- **Preventing repetitive failure**
- **Paradigm shift or measurement ?**
- **<u>Other concerns - diagnosis</u>**

Diagnosis

One of the central ways of improving feedback is good failure diagnosis. However, several factors inhibit diagnosis

- System complexity and coupling
- Engineer over-optimism leading to poor diagnostics and hence to poor diagnosis
- and measurement suggests, increasingly complex paradigms.

Diagnosis



Moderate, (distant/good)

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

Difficult, (distant/poor)

" Please wait ..."

- This appeared on the pilot's and co-pilot's main console on Airbus A340 G-VAEL in Sept, 1994 over Heathrow.

Difficult, (distant/poor)

“ Please wait ...”

- An acceptable substitute might have been:-

  “ The Flight Management System has collapsed, rebooting will take slightly less than N of your Earth minutes. Try whistling.” .

Moderate, (close/poor)

“ System stressed ...”

- This appeared on the cash registers of the author’ s local pub.

Moderate, (close/poor)

“ System stressed ...”

- An hour of exciting discussion about communication protocols, deadlocks and such later, the author and friend realised an acceptable substitute might have been:-

  “ Printer out of paper” .

Moderate, (close/poor)

" Error -23009: there are already more than 64 TCP or UDP streams open {tcp:104}"

- This appeared on the screen of the author's new Macintosh G3 desktop running Apple's latest and greatest version of OS 8, ("the computer for the rest of us").

Moderate, (close/poor)

“ Error -23009: there are already more than 64 TCP or UDP streams open {tcp:104}”

- Two desolate hours later the author realised that an acceptable substitute might have been:-

  “ Modem not responding” .

Moderate, (close/poor)

" Button push ignored"

- This appears on the Flight Management System of a McDonnell-Douglas MD-11, (Drury (1997))

  It is not clear what the programmer is trying to convey. " Paris is the capital of France" would have been equally useful.

  - The pilot also noted " *The airplane [computer system] manuals were written as though by creatures from another planet*" .

Documentation difficulties

Speaking of other planets, note the following excerpt from IBM OS Data management ...

- " If the control password has read without password protection, its secondary passwords must also have read without password protection. A request for a read only or for a read write secondary password will result in a read without password secondary password. If a read without password control password is changed to either a read only or read write control password , all its secondary passwords will automatically become read write secondary passwords."

Imagine trying to diagnose a failure back to a fault in this requirement.

Moderate, (close/poor)

" Line 23: Incompatible types"

- Typical C compiler speak.
- The annoying thing here is that the compiler writer knows precisely what the types are but is determined to bring a little excitement into the programmer's sad, miserable life.
- When the programmer searches the ISO C standard for enlightenment ...

Compatible types, oh joy ...

- **The ISO C standard states in essence:-**
  - Compatible types are the same, but compatible types need not be identical.
- **Now the Oxford English Dictionary states:-**
  - Identical = the same.

Easy (close/good)

Dereference pointer contents 0x0 at
strlen(...) called from
line 126 of myc_constexpr.c called from
line 247 of myc_evalexpr.c called from
line 2459 of myc_expr.c

This is called a *stack trace*. It points unerringly at the responsible code line and usually takes a matter of moments to fix.

The reasons for repetitive failure

The following factors very commonly appear:-

- We don't learn from our mistakes
- We are ruled by unconstrained creativity i.e. fashion, rather than sound engineering
- Software engineers expect their systems to work rather than accepting their inevitable failure and planning accordingly
- Paradigm shifting appears to be making diagnosis more difficult
- Software systems are getting much larger and more tightly coupled in general

Repetitive failure in the outside world

- **Just to show that reluctance to learn isn't solely the preserve of software engineers ...**
  - The DC-10 cargo door saga
    - In the *six months* prior to the dreadful crash of the Turkish Airlines DC-10 in Paris in March 1974, there had been *no less than 1000* cargo door incidents amongst the then 100 strong fleet of DC-10s. They were disregarded.
    - In this incident, the cargo door fell off, and the resulting de-pressurisation caused the cabin floor to collapse severing vital control cables..

Feedback in action ...