

"Testing: the influence of complexity, coupling
repetitive failure and diagnosis"

EuroStar' 99, Barcelona, 12th Nov, 1999

by

Les Hatton

Oakwood Computing, Surrey, U.K.

and

Computing Laboratory, University of Kent, UK

lesh@ oakcomp.co.uk

The great Hotel Princesa Sofia bug mystery, 9/Nov/1999

Note the following eccentric behaviour in the hotel messaging system-

- ◆ State: message waiting
 1. TV issues message waiting banner and requests hotel room
<Enter hotel room>
 2. TV accepts and says press OK to continue
<Press OK>
 3. TV returns to state 1
<Press TV switch>
 4. TV returns to state 1
<Press on/off switch>

5. TV appears to reboot and returns to state 1

<Pull plug out of wall>



Overview of talk

- ❖ **Some observations on testing**
- ❖ **The nature of fault**
- ❖ **Repetitive failure and diagnosis**
- ❖ **Conclusions**



Observations

- ❖ **Why do we test ?**
- ❖ **Growing problems**
- ❖ **Risk management**



Why do we test ?

- ❖ **The classic view is to find fault. A successful test causes the system to fail.**
- ❖ **A more modern view is that testing does two things:-**
 - Finds faults *and*
 - Quantifies the run-time behaviour
 - ◆ Risk management
 - ◆ Demonstration of standard of care
 - ◆ To study severity of system failure



Growing problems for testing

- ❖ **Testing is getting harder in modern systems because of:-**
 - Increasing complexity
 - Increasing coupling
 - Failures in education
 - ‘ Reduced time to market’
- ❖ **The balance between fault finding and risk assessment is also changing because of:-**
 - Increasing diagnostic problems



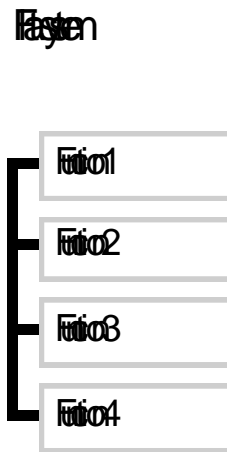
Increasing complexity

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.
- Modern commercial passenger aircraft have between 2 and 5 million lines of code in over a hundred computer systems



Increasing complexity



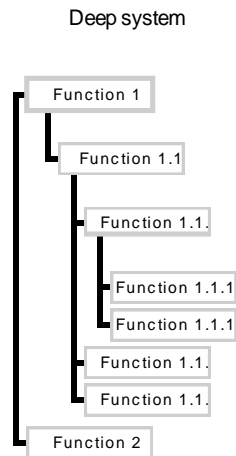
For M modules and N paths per module,
complexity - $M \times N$

Tests easy to construct but many required

This architecture is common in real-time systems



Increasing complexity



For M modules and N paths per module,

and binary fan-out of μ ,

$$\text{complexity} - N^{\left[\frac{1 - (2\mu)^{\frac{\ln M}{\ln 2}}}{1 - 2\mu} \right]}$$

Tests hard to construct but not so many required

This architecture is common in conventional systems



Increasing coupling

Coupling is the degree of interdependence between otherwise separate systems

- In telecommunications systems, coupling can be very high
- In consumer appliances such as cars, many computer systems communicate with each other giving high coupling



Failures in education

Note the following quotations:-

“ Our students graduate and move into industry without any substantial knowledge of how to go about testing a program. Moreover, we rarely have any advice to provide in our introductory courses on how a student should go about testing and debugging his or her exercises” .

“ Every programmer and programming organisation could improve immensely by performing a detailed analysis of the detected errors, or at least a subset of them of duty of care”

“ An efficient program debugger should be able to pinpoint most errors without going near a computer”



**The most depressing aspect about these is that they were made
in 1979 by Glen Myers.**

Overview of talk

- ❖ **Some observations on testing**
- ❖ **The nature of fault**
- ❖ **Repetitive failure and diagnosis**
- ❖ **Conclusions**



Easy faults ...

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. This is why pointer failures amount for a relatively small amount of failure in released systems.



... and hard faults

```
...  
if ( tolerance == acceptable_tolerance )  
...
```

This is a comparison of real valued variables. It is broken in most programming languages. In 1982, the author and a colleague spent 14 weeks trying to find this in the middle of 70,000 lines of signal-processing software, because it occasionally behaved slightly differently on one machine than another. An acceptance test found the symptoms.



How do faults lead to failure ?

❖ **Ed Adams of IBM (1984) found that**

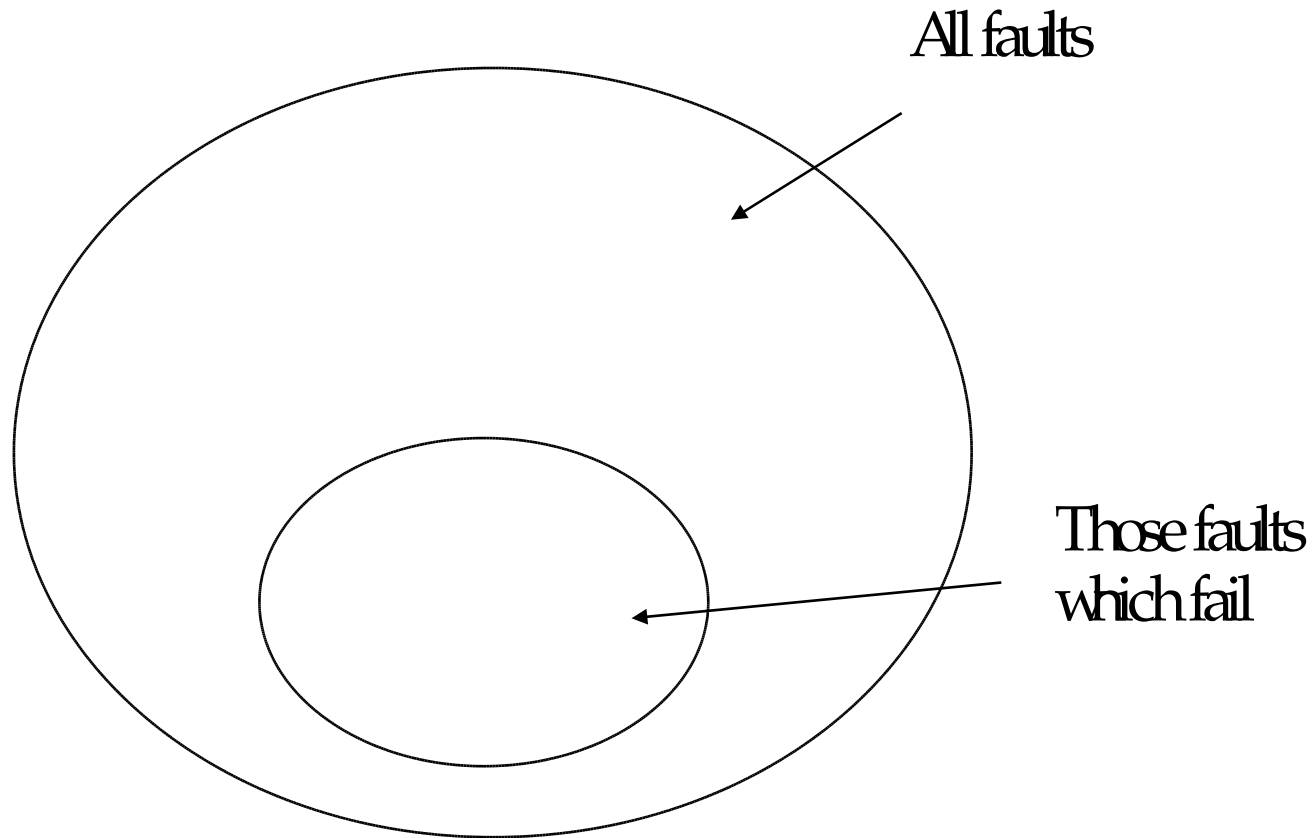
- ~33% of all faults only failed < once every 5000 execution years
- The most common failures, (> once every 5 years) were caused by only 2% of the faults.
- Any correction had about a 15% chance of introducing a problem at least as big into the system.

❖ **Pfleeger and Hatton (1997) found (amongst other things) that:-**

- static faults and dynamic failure were highly correlated in a high reliability system



The relationship between fault and failure



Fault mean free path

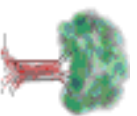
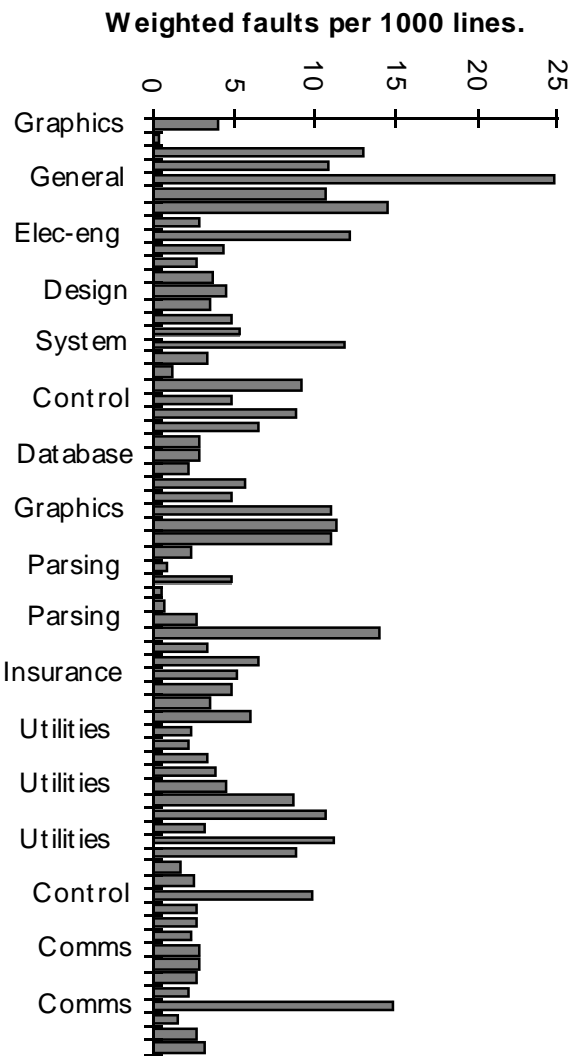
❖ **As Voas (1998) points out:-**

- A significant number of deliberately injected faults caused no change in the external behaviour of the program.

Note also that inspections find fault and dynamic testing finds failure.

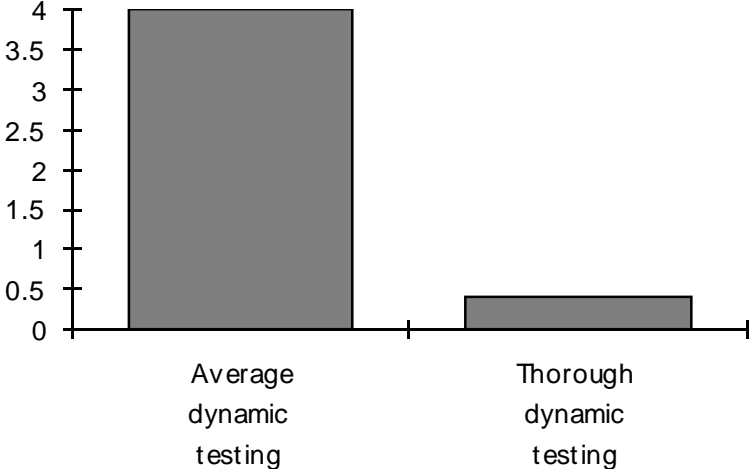


Inspection detectable faults in C applications



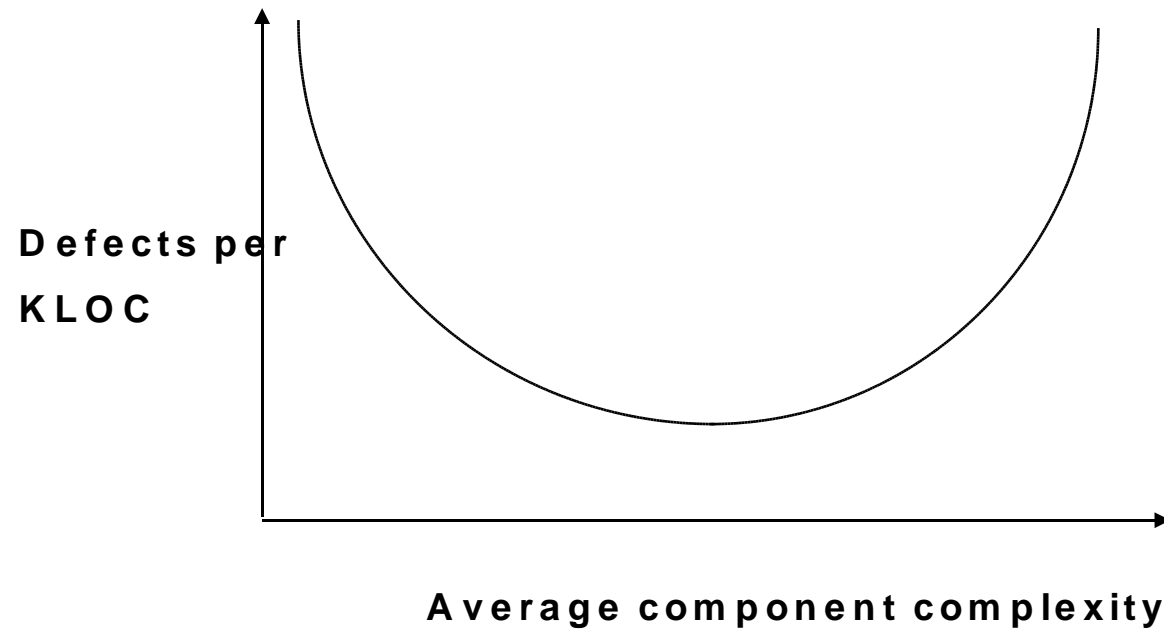
Failure rate of statically detectable faults

Data derived from CAA CDIS

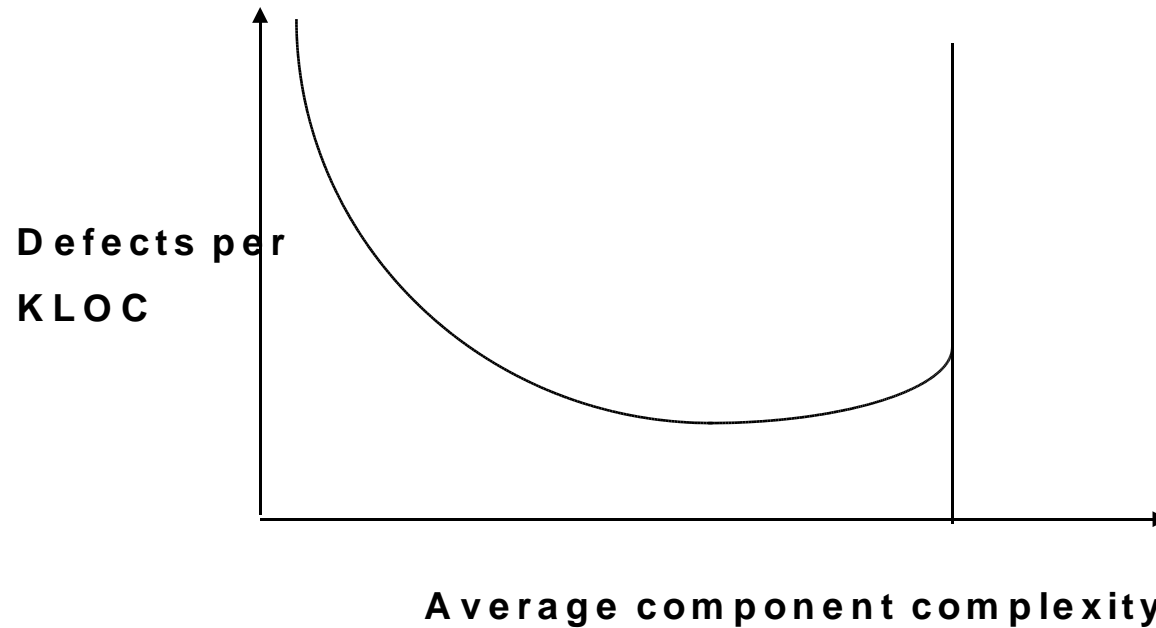


The failure density U curve

For Ada, assembler, C, C++, Cobol, Fortran, Pascal, and PL/M systems:



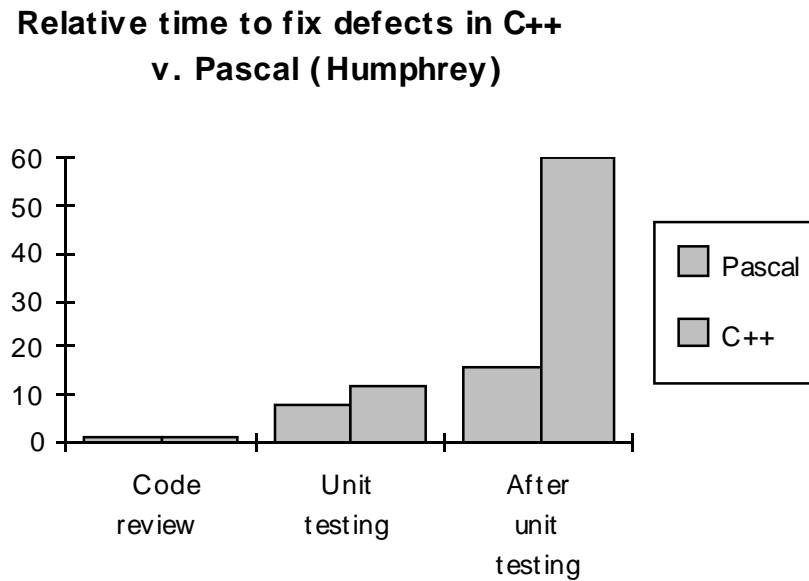
The failure density U curve - invasive truncation



In those systems where excessive complexity has been restricted, the curve is truncated. Here a component specification issue is closely involved with deciding the eventual optimal test strategy.



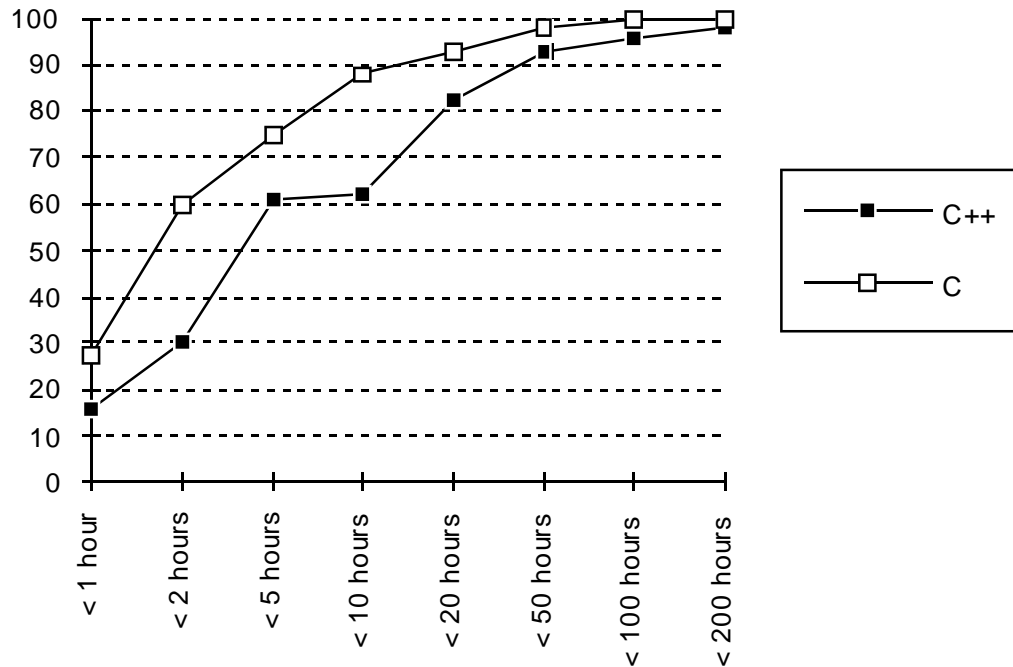
Some observations on OO (Humphrey's (1995) data)



In OO systems, the cost-detection curve appears to rise much quicker suggesting that inspection-weighted testing will be far more effective.



Some observations on OO (Hatton's (1998) data)



In these OO systems, around 5% of all failures took an extremely long time to fix, although they were found easily.



The nature of fault

❖ **We can conclude:-**

- Certain classes of fault yield much easier to testing than others.
- Certain classes of fault have no effect on the program's expected behaviour.
- Certain classes of fault fail but are entirely avoidable.
- Most faults never fail
- A knowledge of the design is necessary to determine the best way to test it.
- The balance between fault finding and risk assessment changes with design.



Overview of talk

- ❖ **Some observations on testing**
- ❖ **The nature of fault**
- ❖ **Repetitive failure and diagnosis**
- ❖ **Conclusions**



An observable fact

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



Risk Management

❖ **All studies of real systems failure show:-**

- It is overwhelmingly likely that systems will fail
- Systems are dominated by repetitive failure
- Relating failure to a responsible fault or faults is getting much harder leading to a substantial “ don’ t know” category - the diagnosis problem.

All this leads us inevitably to recognise that failure is an inevitable property of software systems and we should assess the risk by test quantification and plan for its management.

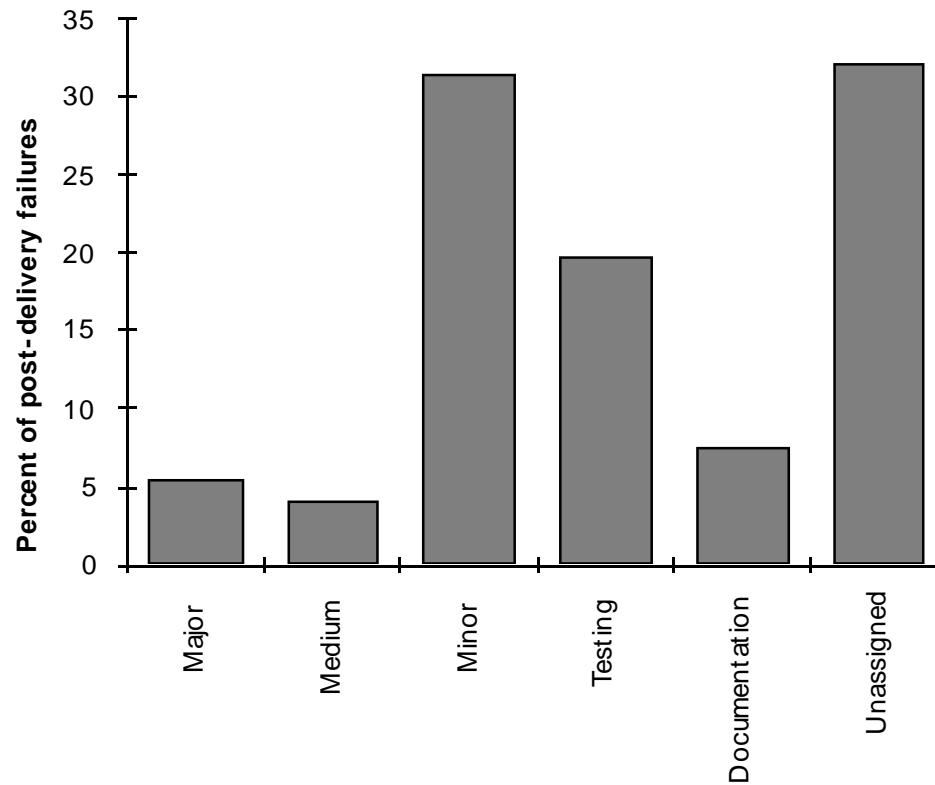


The Patriot missile problem, 1991

- The Patriot missile missed an incoming Scud in the 1991 Gulf War, which then killed 29 people.
- The tracking software failed because there were anomalously two different representations of the constant “ 0.1” . This in turn was multiplied by system uptime to give a spatial error.
- The defect was found in systems testing too late to fix.
- The system carried the message, “ System must be rebooted every 8 hours” . This keeps spatial error small enough. System was left up 48 hours.



More evidence for repetitive failure



This data suggests that major / minor failure ratio is somewhere between 5% and 10%. Note that fully 1/3 were not diagnosable.



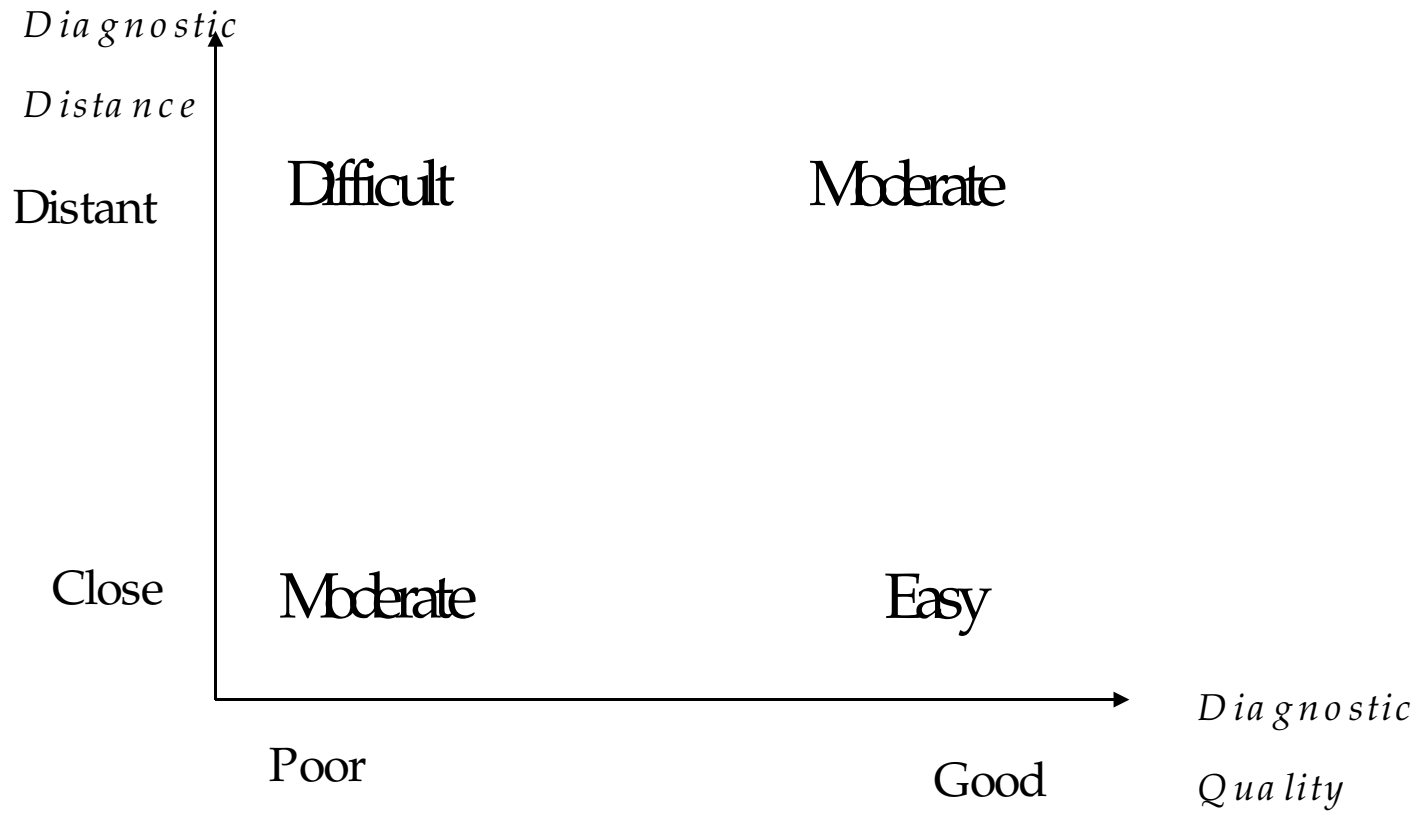
Diagnosis

Factors inhibiting diagnosis

- System complexity and coupling
- Engineer over-optimism leading to poor diagnostics and hence to poor diagnosis



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



Airbus A340 G-VAEL, Sept 1994

Symptom: The Flight management system hung (and lots of other exciting things).

Programmers effort:-

Please wait ...

Translation into English:-

The Flight Management System has crashed and will take slightly less than N of your earth minutes to reboot. Try whistling.

(This is still a problem after a very large amount of effort).



The great local bar disaster

Symptom: The author's local bar was unable to dispense beer.

Programmers effort:-

System stressed ...

Translation into English:-

The printer has run out of paper

(Two hours of author and friend, entirely wasted).



The computer for the rest of us

Symptom: The author' s lovely new G3 Mac would not log onto to his
Internet Service Provider

Programmers effort:-

*More than 64 TCP or UDP streams
open ...*

Translation into English:-

The modem is not switched on

(Nearly 3 hours wasted).



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*” .



The reasons for repetitive failure

The following factors very commonly appear:-

- Software engineers expect their systems to work rather than accepting their inevitable failure and planning accordingly
- Software systems are characterised by exceptionally poor diagnosis and frequently incomprehensible user manuals
- Software systems are getting much larger and more tightly coupled in general
- We don' t learn from our mistakes



Repetitive failure in the outside world

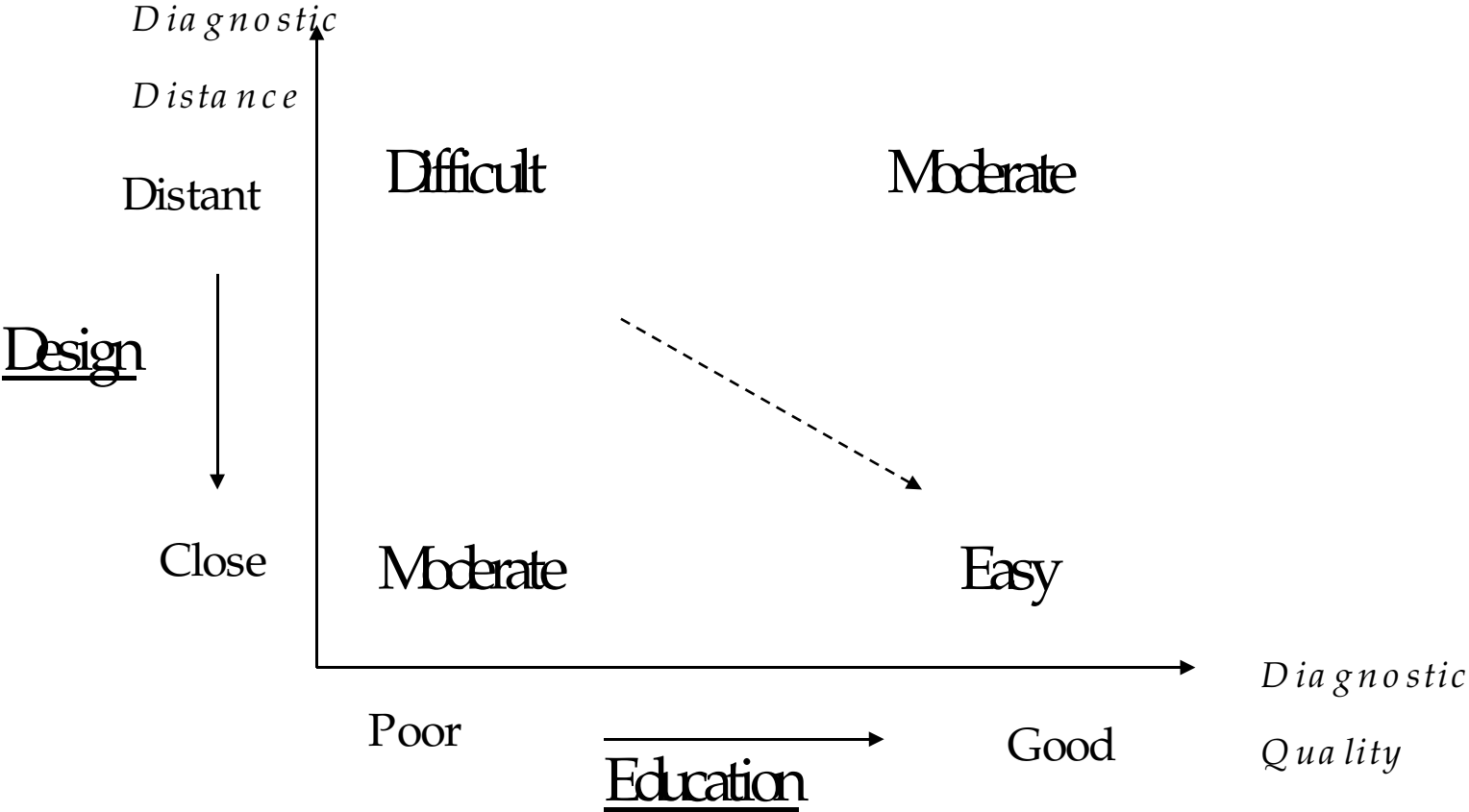
❖ **Just to show that reluctance to learn isn't solely the preserve of software engineers ...**

– 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 depressurisation caused the cabin floor to collapse severing vital control cables..



What we would like to do

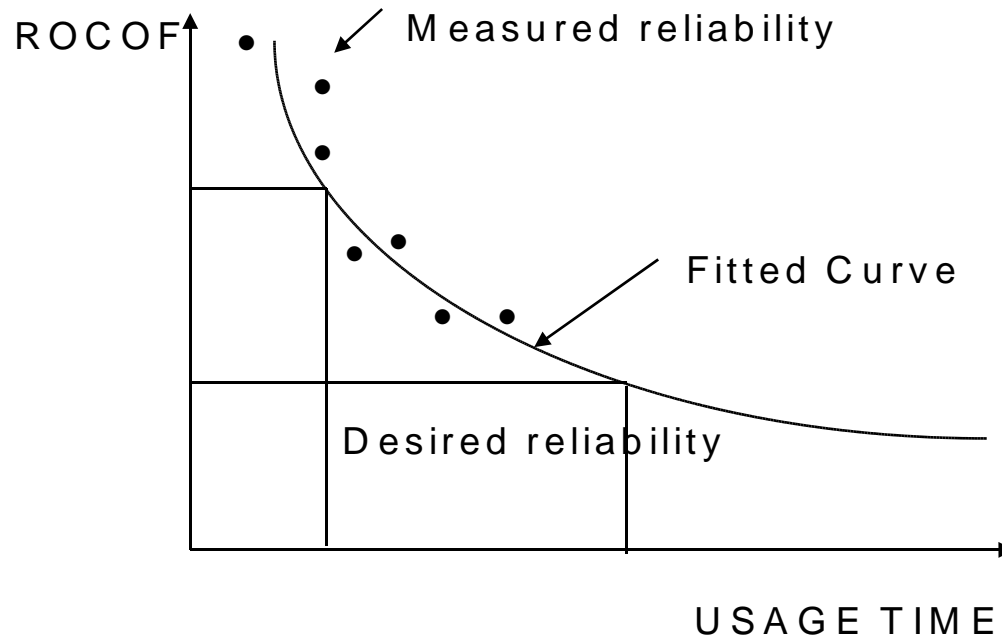


The influence of poor diagnosis on testing

- ❖ **In essence poor diagnosis has the following implications for testing**
 - Testing priority switches in favour of risk management. Tests find defects which cannot be corrected.
 - Testing is not only assessing the reliability of the product but its *sensitivity*. Well-designed products respond easily to corrective maintenance, (and don' t require much of it !). (Think of car engine evolution)
 - One of the responsibilities of testing is to assess diagnosability



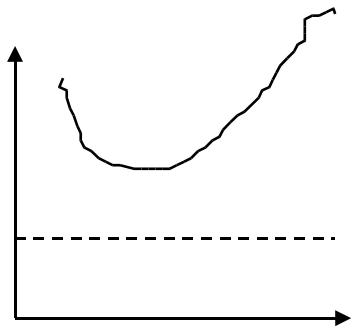
Testing is an economic trade-off



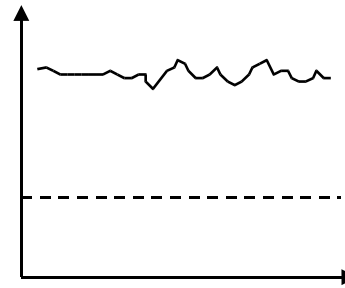
The point at which testing stops *is the point at which it is economically viable to stop given the trade-off between early availability and the risk of failure.* It is manifestly NOT an engineering decision.



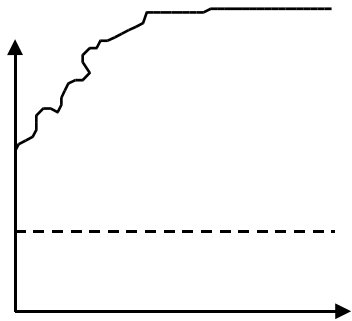
Testing is an economic trade-off



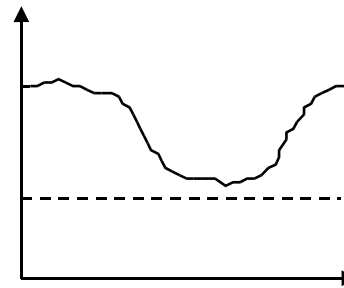
Late requirements change



Ambition /capability clash



Replacement of testers



August



Overview of talk

- ❖ **Some observations on testing**
- ❖ **The nature of defect**
- ❖ **Repetitive failure and diagnosis**
- ❖ **Conclusions**



Conclusions

- ❖ **Testing is getting harder**
- ❖ **Testing is increasingly concerned with risk assessment**
 - Diagnosis is much harder
 - Systems are much bigger and more tightly-coupled
- ❖ **Testing is itself a diagnostic for the development process**
- ❖ **Testing should play a much bigger role in design**

