STAR' 99 East, May 10-14 1999

"Testing is not a phase"

by

Les Hatton

Oakwood Computing, U.K.
and Computing Laboratory, University of Kent

lesh@oakcomp.co.uk

Last update: 8th March 1999

©Copyright, L.Hatton, 1999

Overview of talk

- ❖ Some observations on testing
- ❖ The nature of defect
- ❖ Symptoms of narrow interpretation
- ❖ Testing is not a phase

© OC Copyright, 1999

Observations

- Why do we test ?
- Views of testing
- Failures in education
- Introducing Risk management

© OC Copyright, 1999

Why do we test ?

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

© OC Copyright, 1999

Why do we test ?

- If defects are easy to fix, the test leads to the product getting more reliable
- If they are impossible to fix, the test leads to a potential risk to the user, necessitating management.

© OC Copyright, 1999

- **The following misconceptions are still widely held:-**
  - Testing is easy
  - Testing naturally terminates, (" Have you finished testing that program yet ...")
  - Testing requires no training, simply a ready supply of coffee.
  - Testing is the most expendable when deadlines press
  - Testing is just another phase in the software process.

© OC Copyright, 1999

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.

© OC Copyright, 1999

- **All studies of real systems failure show:-**
  - It is overwhelmingly likely that systems will fail
  - Systems are dominated by repetitive failure
  - Test effectiveness is generally very poor
  - 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.

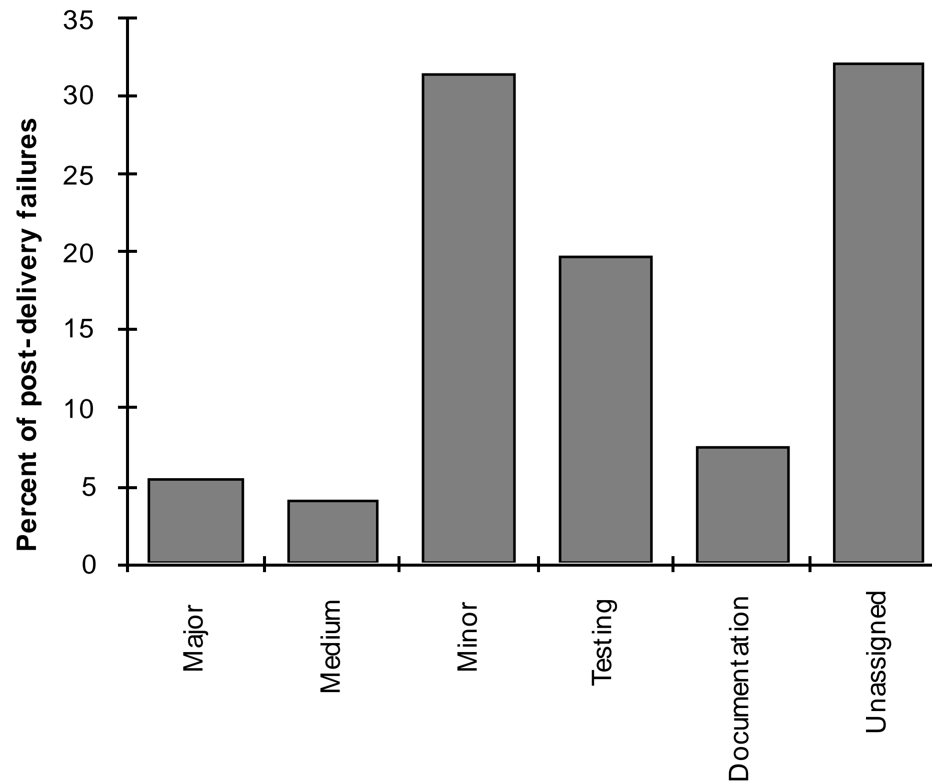
© OC Copyright, 1999

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.

© OC Copyright, 1999

Categories of post-delivery failures in CAA Study, 1997



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

© OC Copyright, 1999

Overview of talk

- Some observations on testing
- <u>The nature of defect</u>
- Symptoms of narrow interpretation
- Testing is not a phase

© OC Copyright, 1999

Easy defects ...

```
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.

© OC Copyright, 1999

... and hard defects

```
...
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.

© OC Copyright, 1999

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.

© OC Copyright, 1999

Defect mean free path

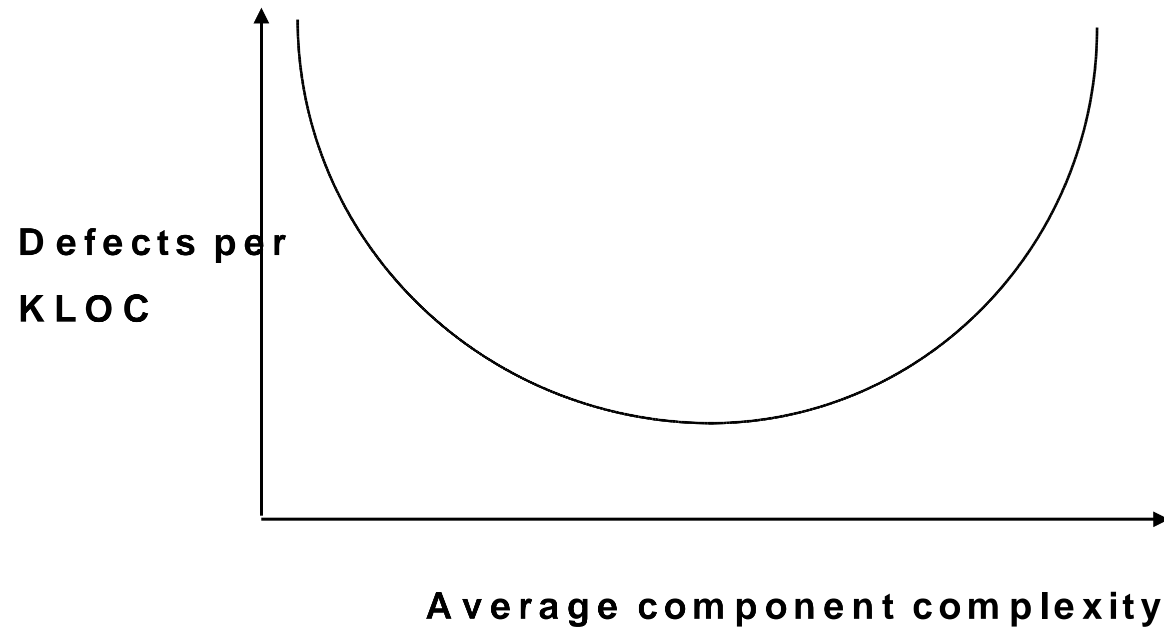
- **As Voas (1998) points out:-**
  - A significant number of deliberately injected defects caused no change in the external behaviour of the program.

We can re-phrase one of the goals of testing as not simply finding defect, but finding defect which leads to unexpected behaviour. This gives a new slant on inspection. Many of the defects found by inspection have no such effect.

© OC Copyright, 1999

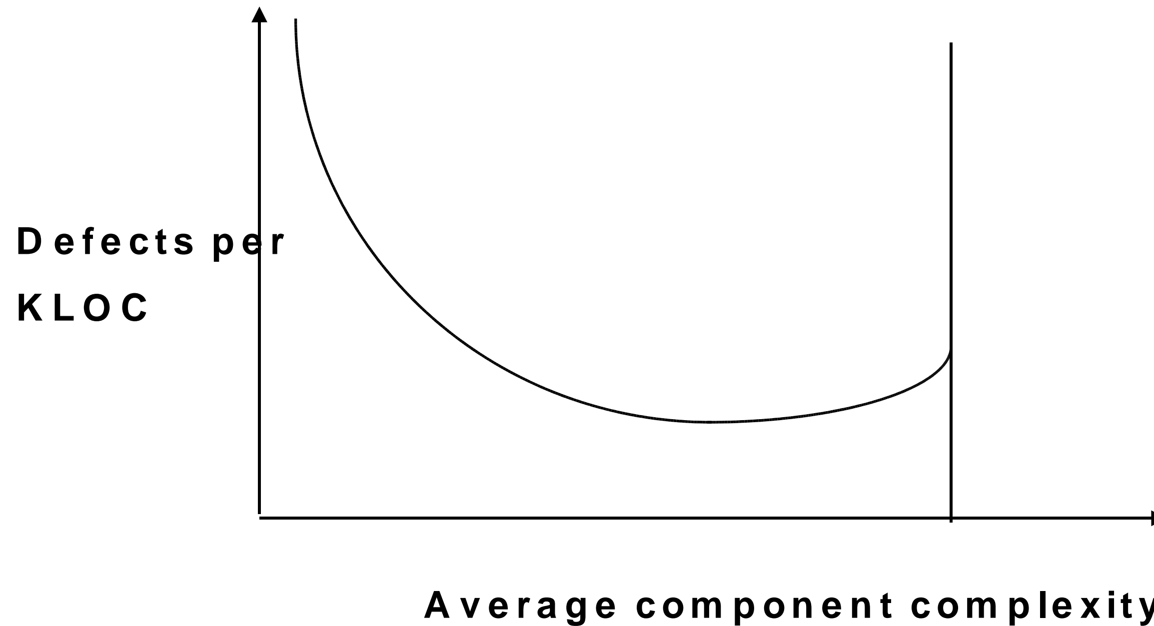
The defect density U curve

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



© OC Copyright, 1999

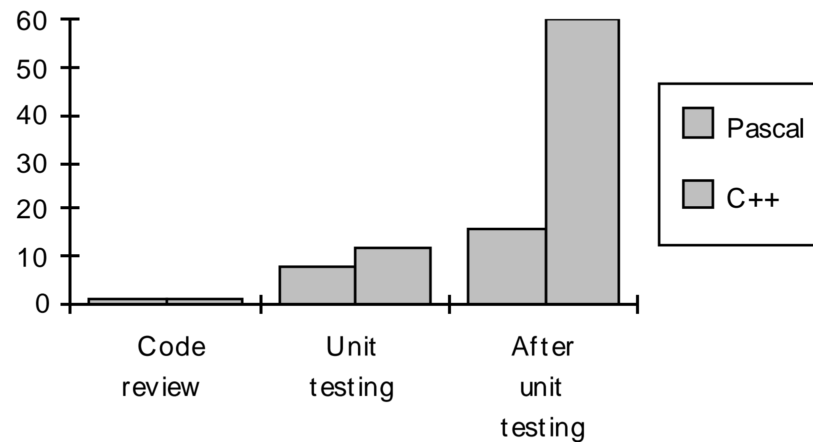
The defect density U curve - invasive truncation



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.

© OC Copyright, 1999

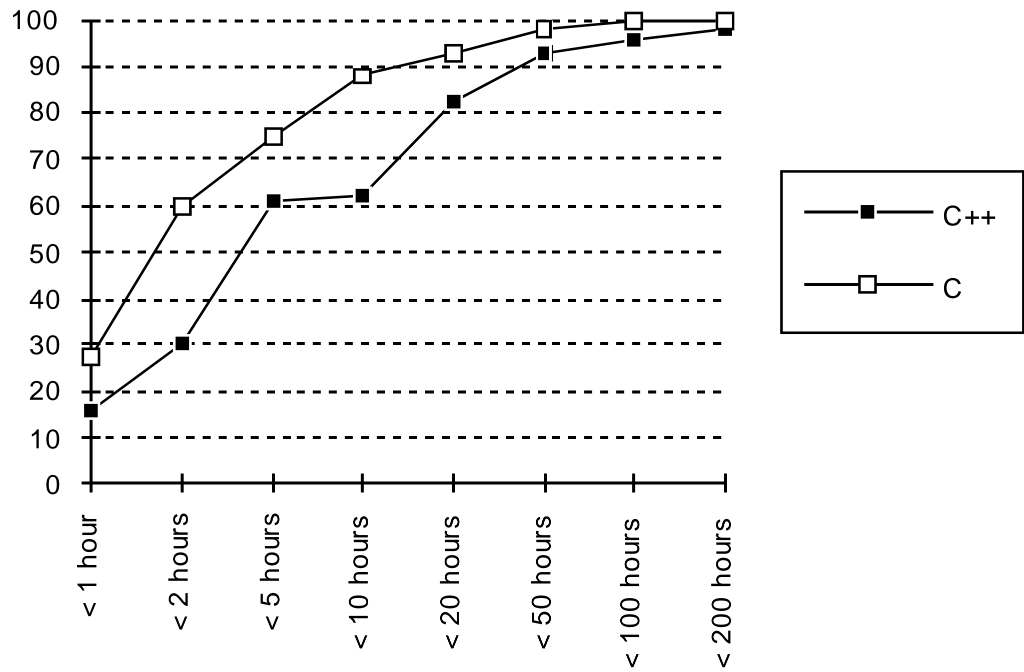
Some observations on OO (Humphrey's (1995) data)



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

© OC Copyright, 1999

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



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

© OC Copyright, 1999

The nature of defect

- **We can conclude:-**
  - Certain classes of defect yield much easier to testing than others.
  - Certain classes of defect have no effect on the program's expected behaviour.
  - A knowledge of the design is necessary to determine the best way to test it.
  - The balance between defect finding and risk assessment changes with design.

So testing clearly interacts and indeed underlies other parts of the software process.

© OC Copyright, 1999

Overview of talk

- Some observations on testing
- The nature of defect
- Symptoms of narrow interpretation
- Testing is not a phase

© OC Copyright, 1999

Narrow views of testing

- **The following are symptoms of a narrow view of testing:-**
  - Poor testability
  - Poor diagnosability
  - Failure to understand that testing embodies economic trade-offs.

© OC Copyright, 1999

Poor testability

- Hatton (1995) showed that around 10% of all Fortran and C functions were untestable by any criteria.
- This is simply a failure to recognise that testing is a design issue and that test planning should take place at each phase of the software process.

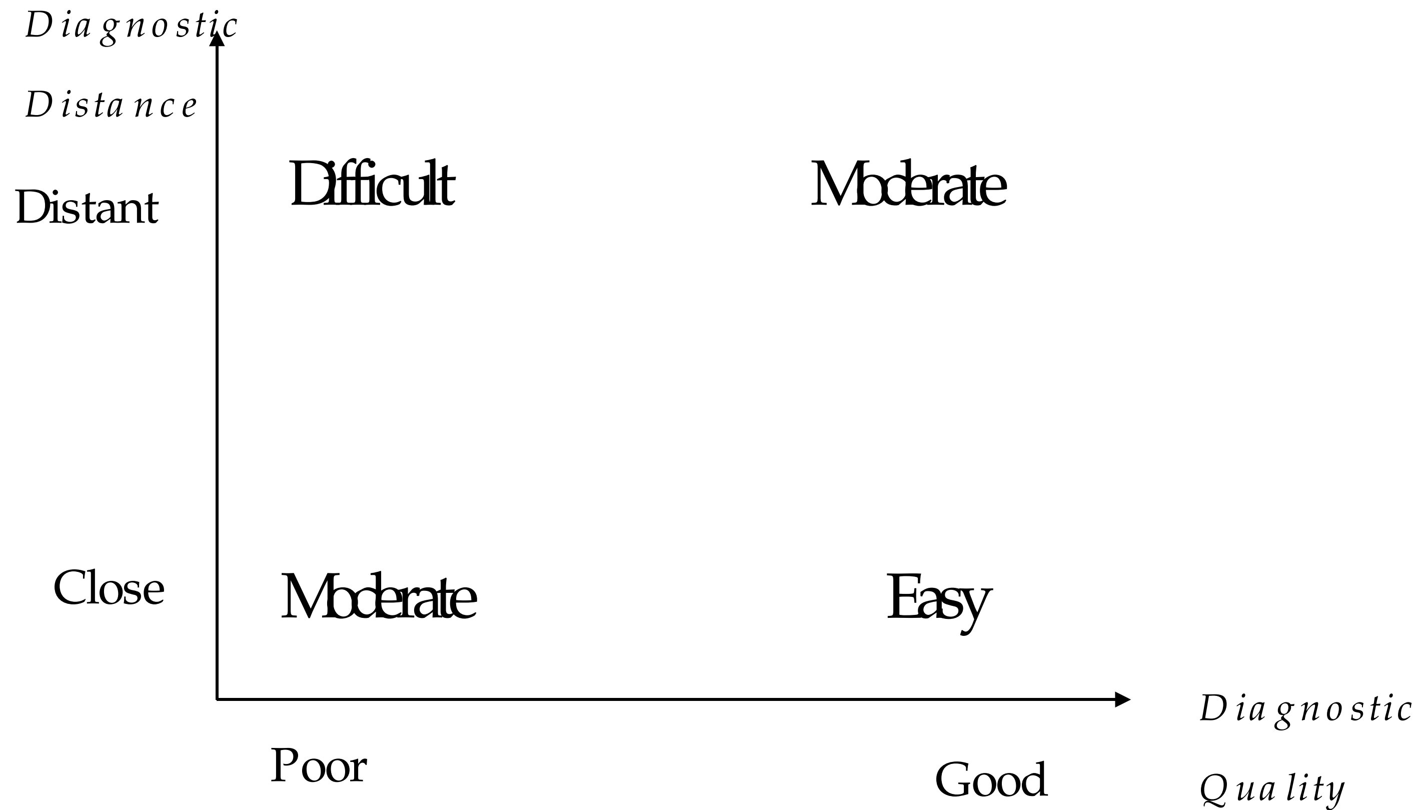
© OC Copyright, 1999

Poor diagnosability

- This is a symptom that testing is believed to have removed all defects.
- It is also a symptom that testing is not considered during design.

© OC Copyright, 1999

Diagnosis



© OC Copyright, 1999

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).

© OC Copyright, 1999

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).

© OC Copyright, 1999

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 ...

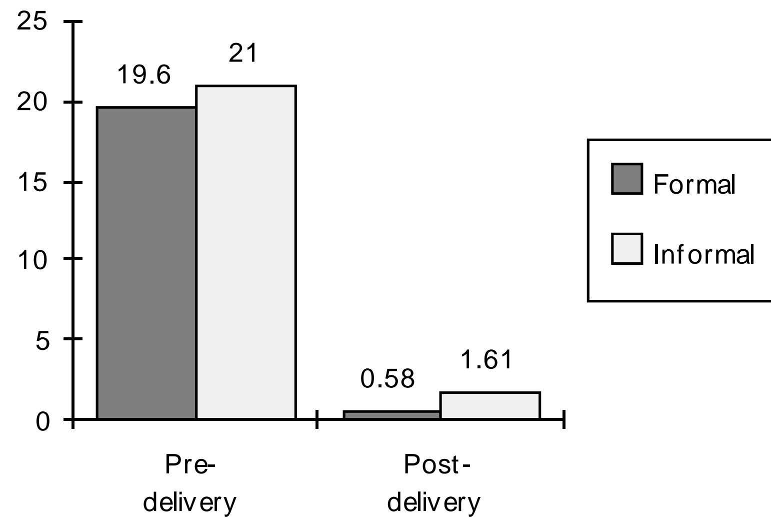
Translation into English:-

The modem is not switched on

(Nearly 3 hours wasted).

© OC Copyright, 1999

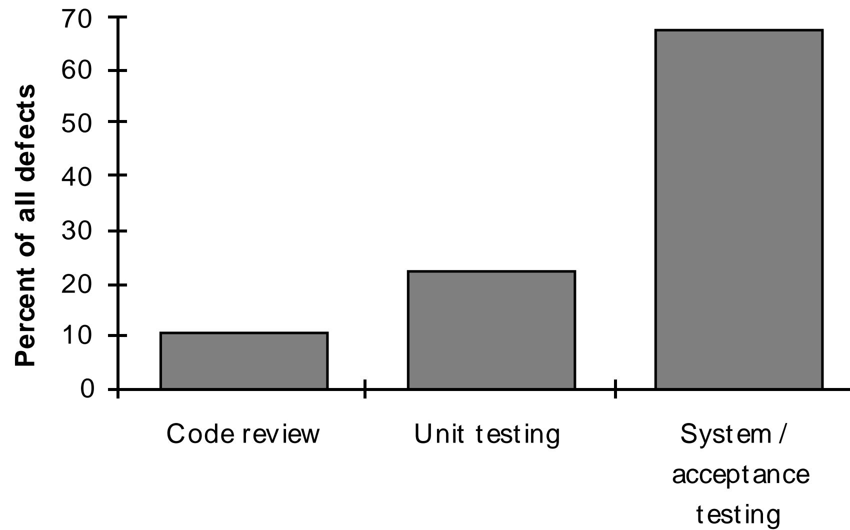
Getting the balance right



The CAA CDIS system is an excellent example whereby testing interacted with all stages of development. However, even then, there were lessons to learn ...

© OC Copyright, 1999

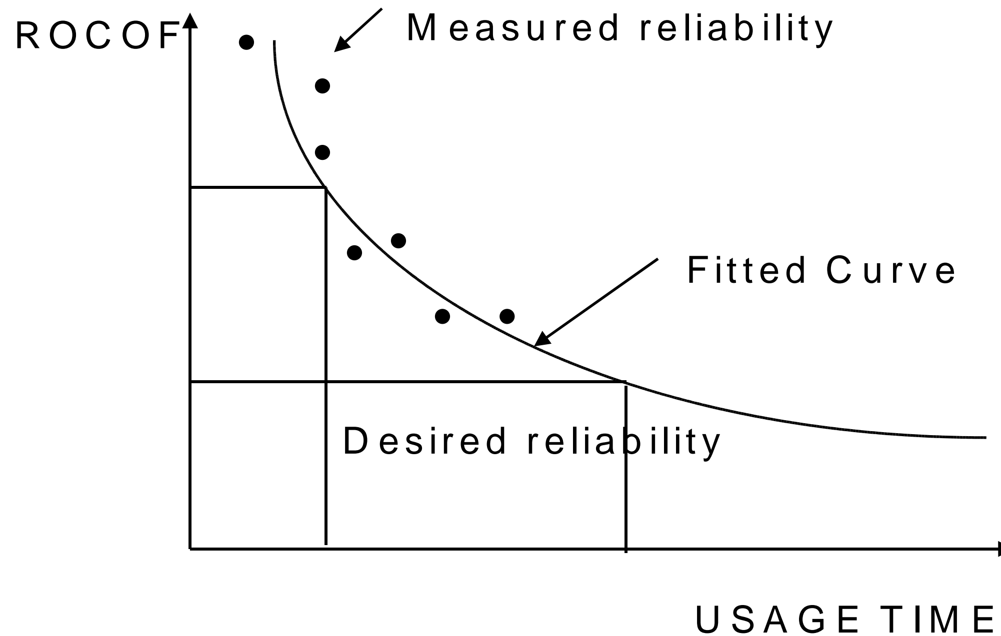
Getting the balance right



This data suggests that in this case they led to over-reliance at the expense of code inspection. This curve is the opposite of what it should be.

© OC Copyright, 1999

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.

© OC Copyright, 1999

Overview of talk

- Some observations on testing
- The nature of defect
- Symptoms of narrow interpretation
- Testing is not a phase

© OC Copyright, 1999

Testing is not a phase

- **As we have seen, testing issues arise at:-**
  - Requirements: can a requirement be tested ?
  - Design: does the design fail benignly- safety analysis.
  - Implementation: Is the code testable ? Do the coding defects propagate to the exterior ?
  - Release: has a good enough job of risk management been undertaken - litigation ?

Testing is manifestly NOT a phase. It is an attitude of mind which permeates the entire software process.

© OC Copyright, 1999