

2003-

“Five variations on the theme:
Software failure: avoiding the avoidable and living with the
rest“

Variation 4: “Avoiding the avoidable”

Prof. Les Hatton

Computing Laboratory, University of Kent, Canterbury

Version 1.1: 18/Nov/2003

©Copyright, L.Hatton, 2003-

Introduction to software failure

In the absence of any technology for guaranteeing the absence of defect, engineers are always left with two obligations:-

- When the system fails, it must have the least effect on the user relative to the benefit it provides
- The nature of the failure must be sufficiently well diagnosed that its cause can be found and corrected quickly so the failure does not re-occur.

These are both *Design* issues



Overview

- ❖ **A case history in forensic analysis**
- ❖ **Some successful technologies**
- ❖ **The process v. product dilemma**
- ❖ **Why are we so bad at diagnosis ?**



A case history in forensic analysis and common mode failure

Precedence in programming languages



Precedence

Precedence in languages

- Languages vary greatly in precedence complexity.
 - ◆ Ada83 has 4 levels.
 - ◆ Fortran 77 has 9 levels
- The C-like languages, (C, C++ and Java) are characterised by very complex precedence tables.
 - ◆ C has 15 levels
 - ◆ Java and C++ have 17 levels
- Scripting languages can be even worse
 - ◆ Tcl has 12 levels but *Perl has 21 levels and PHP 23 levels.*



Operator precedence levels

OPERATORS	Associativity
() [] -> . ++ --	Left to Right
! ~ ++ -- + - * & (type) sizeof	Right to Left
* / %	Left to Right
+ -	Left to Right
<< >>	Left to Right
< <= > >=	Left to Right
== !=	Left to Right
&	Left to Right
^	Left to Right
	Left to Right
&&	Left to Right
	Left to Right
? :	Right to Left
= += -= *= /= %= &= ^= = <<= >>=	Right to Left
,	Left to Right



Precedence

How we have responded

– To attempt to ameliorate this, we introduce rules like:

♦ “No expression shall rely on precedence”

However, this rule for example is broken 4441 times in the ISO C validation suite, (about 1 per 50 lines).

– All the C-like languages have inherited the same table and added to it.



Precedence

How we could respond forensically

– Analyse failure modes

- ◆ `if (flags & FLAG != 0) ...`
- ◆ `while (c = getc(in) != EOF) ...`
- ◆ `if ((t=BTYPPE(pt1->aty)==STRTY) || t == UNIONTY) ...`
- ◆ `if (a < b < c) ...`

– Derive failure rule



Precedence

How we could respond forensically

Replace ...

- ◆ “No expression shall rely on precedence”

By ...

- ◆ “If a Boolean valued sub-expression appears in a predicate next to a bit operator, an assignment operator or a relational operator, it is nearly always wrong.”

This rule is much more precise. Even in the ISO C validation suite which is supposed to exercise everything, it only fires 11 times. In real code, it has a > 90% hit rate for real defects.



Overview

- ❖ **A case history in forensic analysis**
- ❖ **Some successful technologies**
- ❖ **The process v. product dilemma**
- ❖ **Why are we so bad at diagnosis ?**



Inspections

Inspections find fault, not failure. Historically, they are categorised into:-

- Design inspections, (much harder to do because of poor standardisation of design issues)
- Code inspections

There is some evidence to suggest that code inspections find more problems but design inspections find more *serious* problems



Inspections

❖ **Probably the most well-known are Fagan inspections, named after Michael Fagan of IBM. In essence a Fagan inspection contains:**

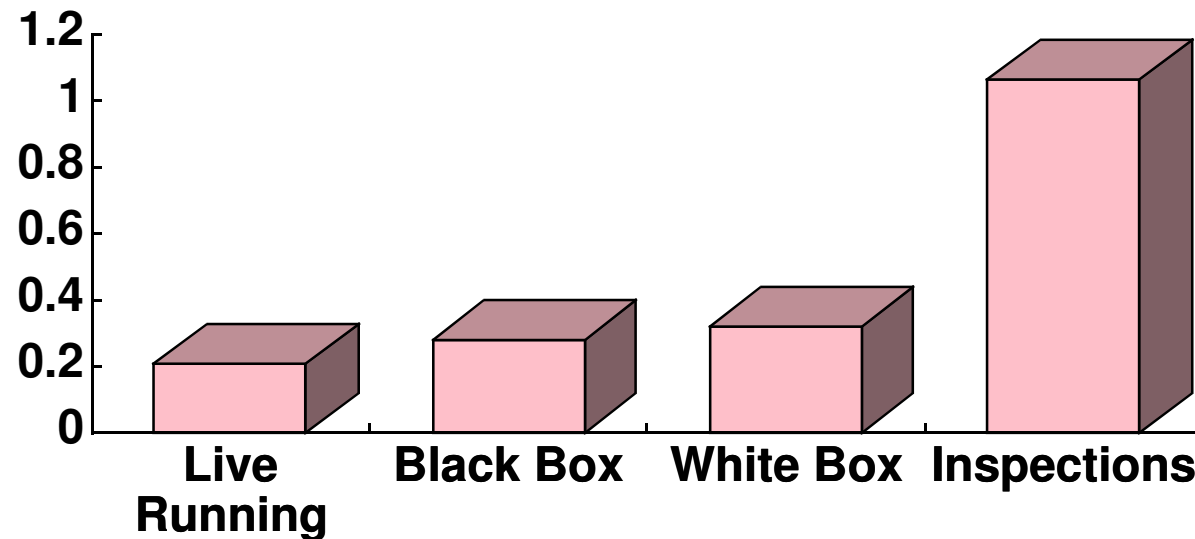
- Planning
- Overview
- Individual preparation
- Program inspection
- Rework
- Re-inspection

A walk-through is less formalised, about half as effective and contains only steps 4 and 5.



Inspection effectiveness- HP example

Code inspections are the most effective way of monitoring intrinsic quality. Consider the following rates of defects found per hour, (Grady & Caswell, Hewlett-Packard):



Inspections - applying control process feedback

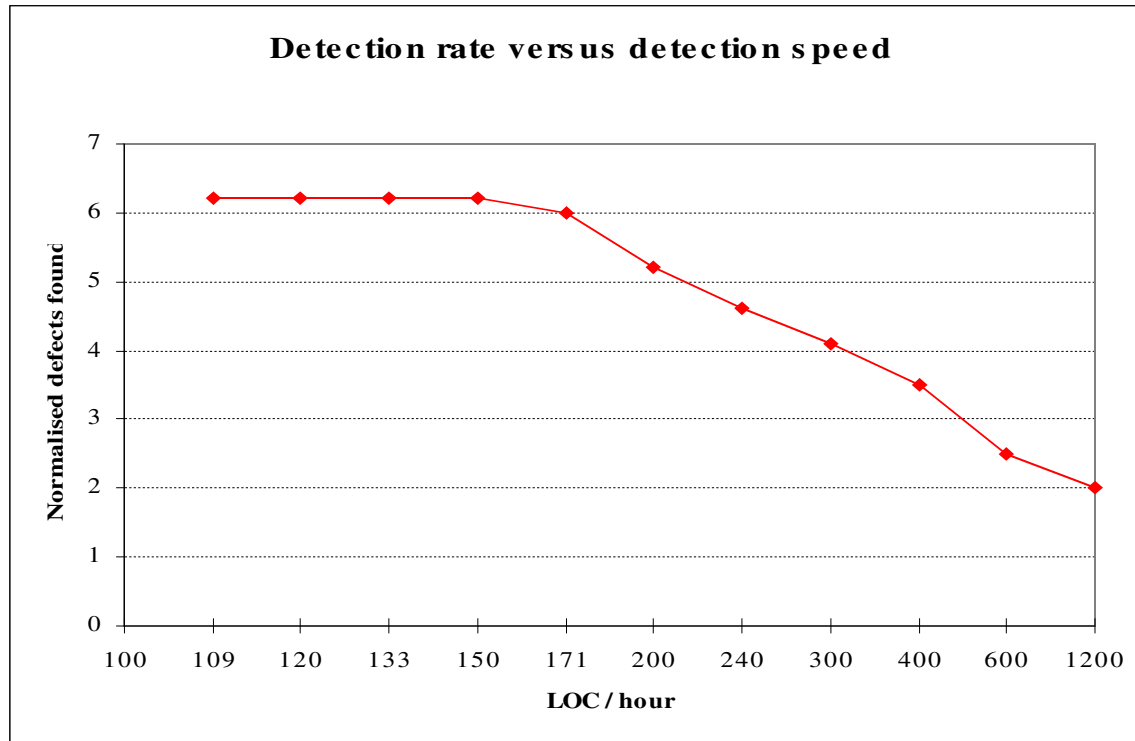
❖ **An inspection should measure the following:**

- Lines per hour reviewed
- Defects per hour found
- Defects per hour missed, (using hindsight)

It is known that inspection rates of > 100 lines per hour and around 400 lines per day lead to a significant deterioration in effectiveness.



Inspections – lines per hour

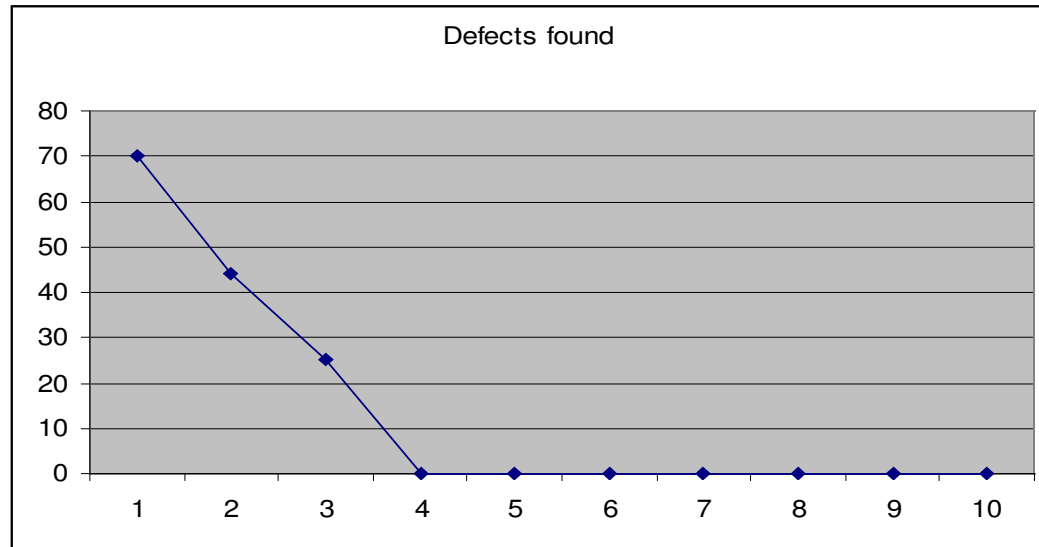


These measurements are due to Roper (1999) made on OO systems which have similar non-locality to embedded interrupt driven systems.



Inspections – lines per hour

Defect density
found



Inspection speed (100s LOC/ hour)

These measurements are due to Humphrey (1995) made on
25 C++ projects



Inspections

❖ **Code inspections inspect for:**

- Style transgressions
- Standard transgressions
- Dangerous use of language
- Differences between requirements and behaviour

Of these, the first three can be *automated* using static deep-flow tools, greatly increasing the efficiency of inspections. Code inspectors should be given code from which the first three categories are *absent*.



Inspections

❖ **Inspection yield**

Total number of defects found in inspection

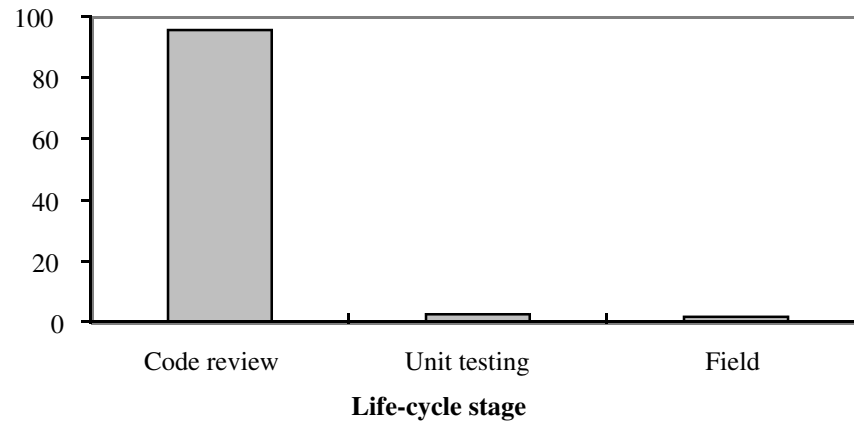
Total number of defects ever found

What is a sensible value for this ?



Defect discovery profile for effective inspections

% of all defects ever found and where found, SEMA (Gilb & Graham 1993)



Case histories of inspection yields

% of all defects found during Inspections



Overview

- ❖ **A case history in forensic analysis**
- ❖ **Some successful technologies**
- ❖ **The process v. product dilemma**
- ❖ **Why are we so bad at diagnosis ?**



Process v. Product Failure

❖ **Process failure**

- We either build the wrong thing or build the right thing too late or fail to build anything at all in the vast majority of cases

❖ **Product failure**

- Cessation
 - ◆ The product crashes
- Misleading results
 - ◆ The product gives misleading answers



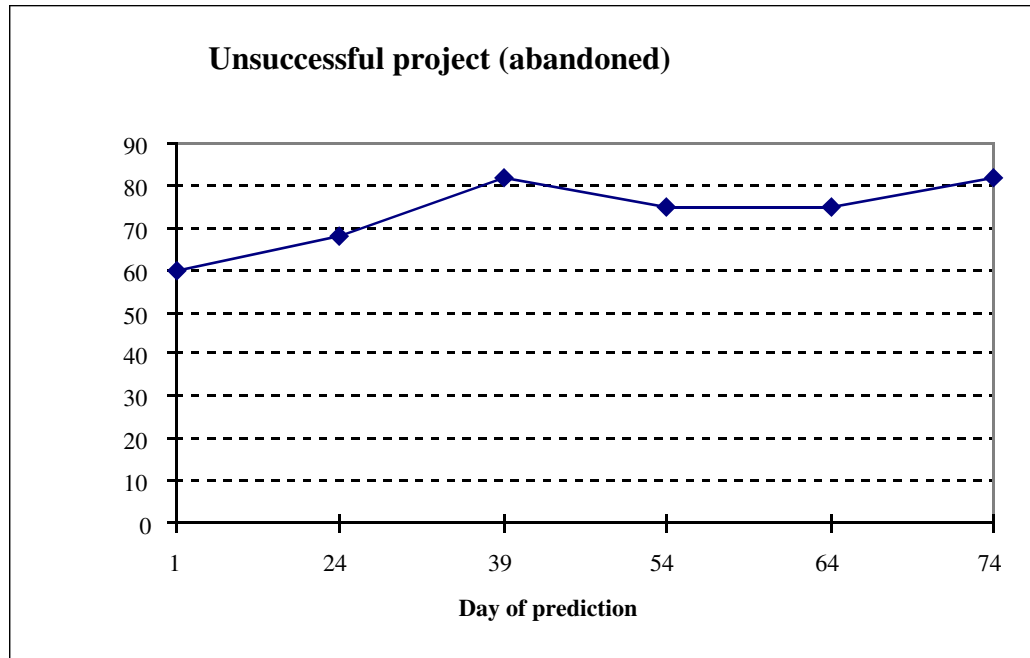
To plan or not to plan ?

“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

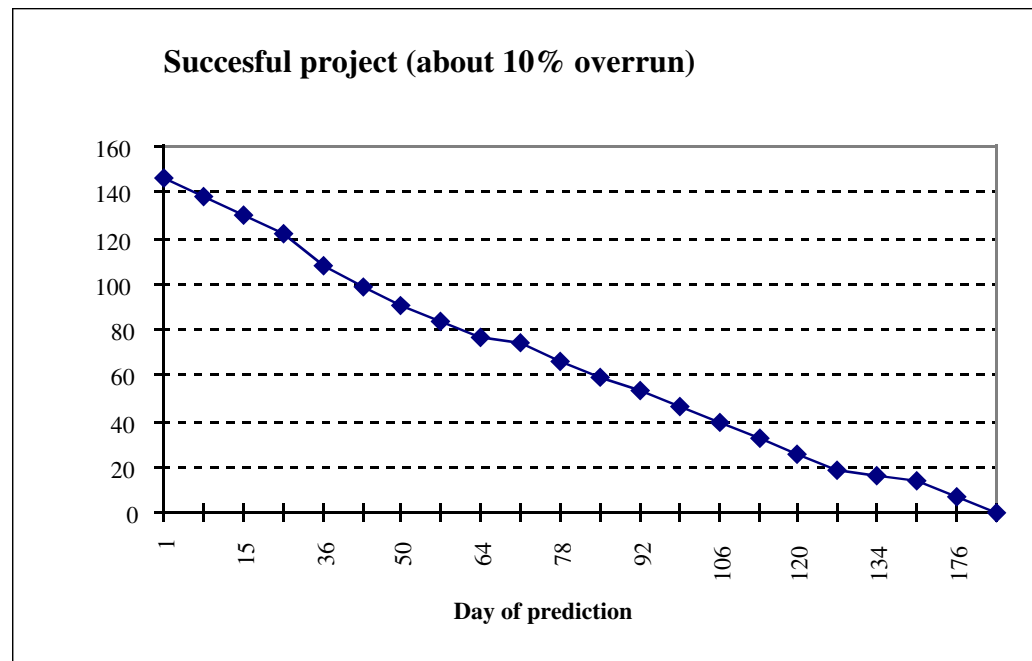


Planning data from a grand **'unified'** programming project.
(Produced after the project seemed to be struggling.)

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.



Software Process

Points to consider

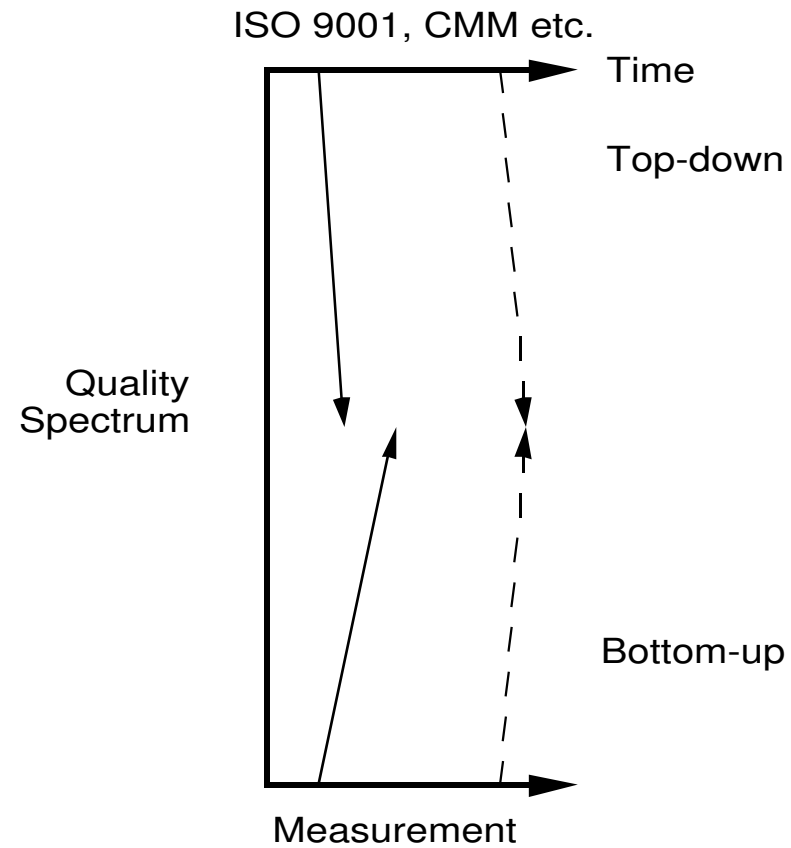
- There is an inherent belief that a good process implies a good product
- So why is Linux so good ?
 - ◆ Linux is categorically CMM level 1 (more shortly) so is the CMM wrong or does Open Source development have important properties that we don't understand well yet ?
 - ◆ Is the reliability of Linux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



Process quality v. Product quality

Process

Product



Software Process – the layman's guide to the CMM

A five level model developed on behalf of the US DoD at Carnegie-Mellon in the 80s and 90s

- Level 1 (Initial – used to be called chaos)
- Level 2 (Repeatable)
- Level 3 (Defined)
- Level 4 (Managed)
- Level 5 (Optimised or Godlike)

There are around 50 groups at level 5 in the world, around half of them in India, (who take software development a lot more seriously than we do). BUT ... what about Linux ?



The CMM levels

5	Optimised	^ ?
4	Managed	^ Full statistical process control; metrics used for defect prevention
3	Defined	^ Process database, process metrics collected and analysed; risks managed
2	Repeatable	^ Process focus, software engineering process group; training program throughout
1	Chaotic	^ Project planning, tracking; software quality assurance, configuration management



CMM statistics

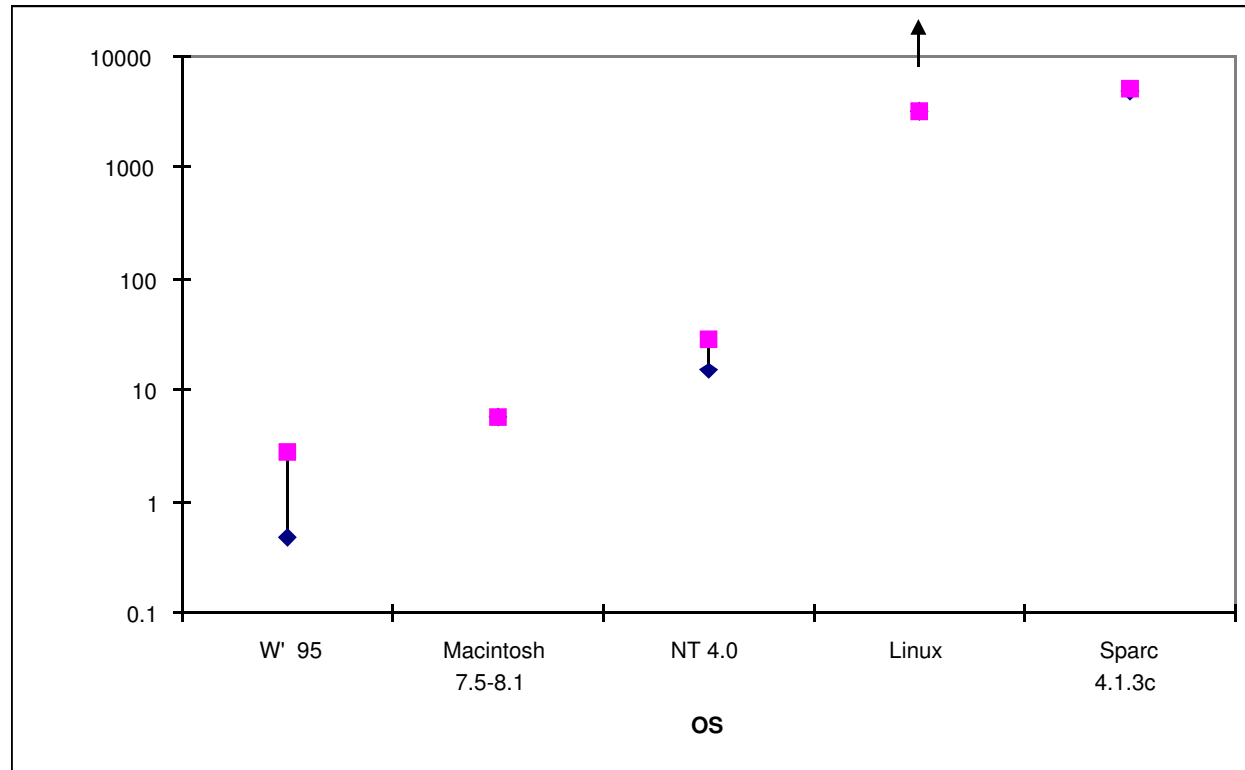
❖ **Who is where ?**

- About 70% of all companies believed to be at level 1
- About 20% believed to be at level 2
- About 9% believed to be at level 3
- Around 50 organisations in the world are at level 5 of which about half are in India
- It takes around 18 months to move between levels realistically
- There is significant evidence that software development costs are reduced considerably as higher levels are achieved.



The PC picture ...

MTBF



Mean Time Between Failures in hours of various operating systems



Similarities between Linux and Windows NT ...

- ❖ **Both operating systems are of comparable essential complexity**
 - Both are multi-tasking, multi-threaded, symmetric multi-processing systems
- ❖ **There appears to be about a factor of 10 in non-essential complexity**
 - Windows NT is around 20 million lines
 - Linux is around 2 million lines
- ❖ **Both started around 1991**



Differences between Linux and Windows

❖ **Visibility**

- Windows is proprietary
- Linux is open source and distributed under the GPL, (GNU public licence).



Differences between Linux and Windows

❖ **Testing approach**

- Windows is heavily dependent on dynamic testing with extensive beta testing campaigns
 - ◆ (Around 89,000 defects were reported to have been found in Windows 95 during beta test)
 - ◆ (Around 60,000 defects were reported to have been found in Windows 2000 during beta test).
- Linux is dependent on a mixture of large-scale code inspection and dynamic testing



The Linux Picture ...

Linux is also characterised by:-

- Excellent distributed version control
- Excellent development/test communication
- High average levels of experience
- Unusually high levels of perfective maintenance



The PC Picture ...

More notes on Windows:

- Availability and how to confuse the picture ...
- There are about 30 billion Windows crashes a year, (every machine at least one a week). (Dvorak, PC magazine, 4-Aug-2003)
- Does anybody add up the cost of this ? (Try this, each crash costs around 15 minutes of work say. At minimum wage, this is 30 billion pounds a year. Its probably much higher.)

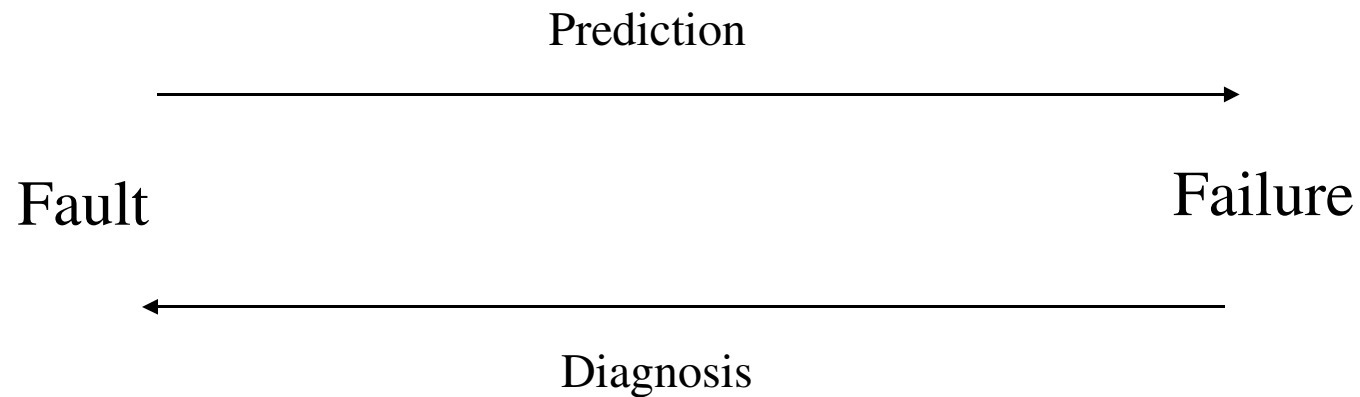


Overview

- ❖ **A case history in forensic analysis**
- ❖ **Some successful technologies**
- ❖ **The process v. product dilemma**
- ❖ **Why are we so bad at diagnosis ?**



The prediction problem



Prediction is in its infancy and diagnosis is becoming more difficult



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.



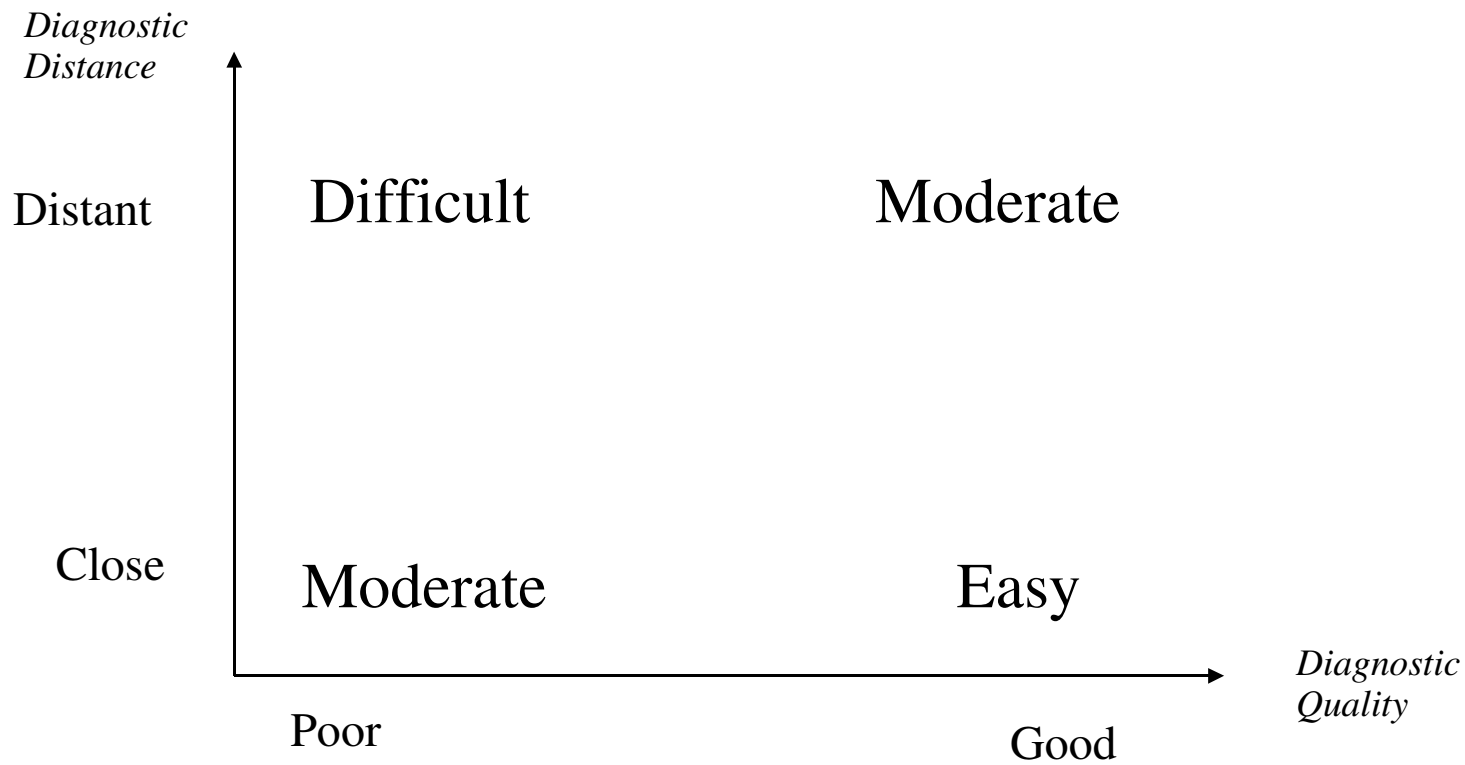
Increasing coupling

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 potentially high coupling



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



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 great local bar disaster

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

Programmers effort:-

System over-stressed ...

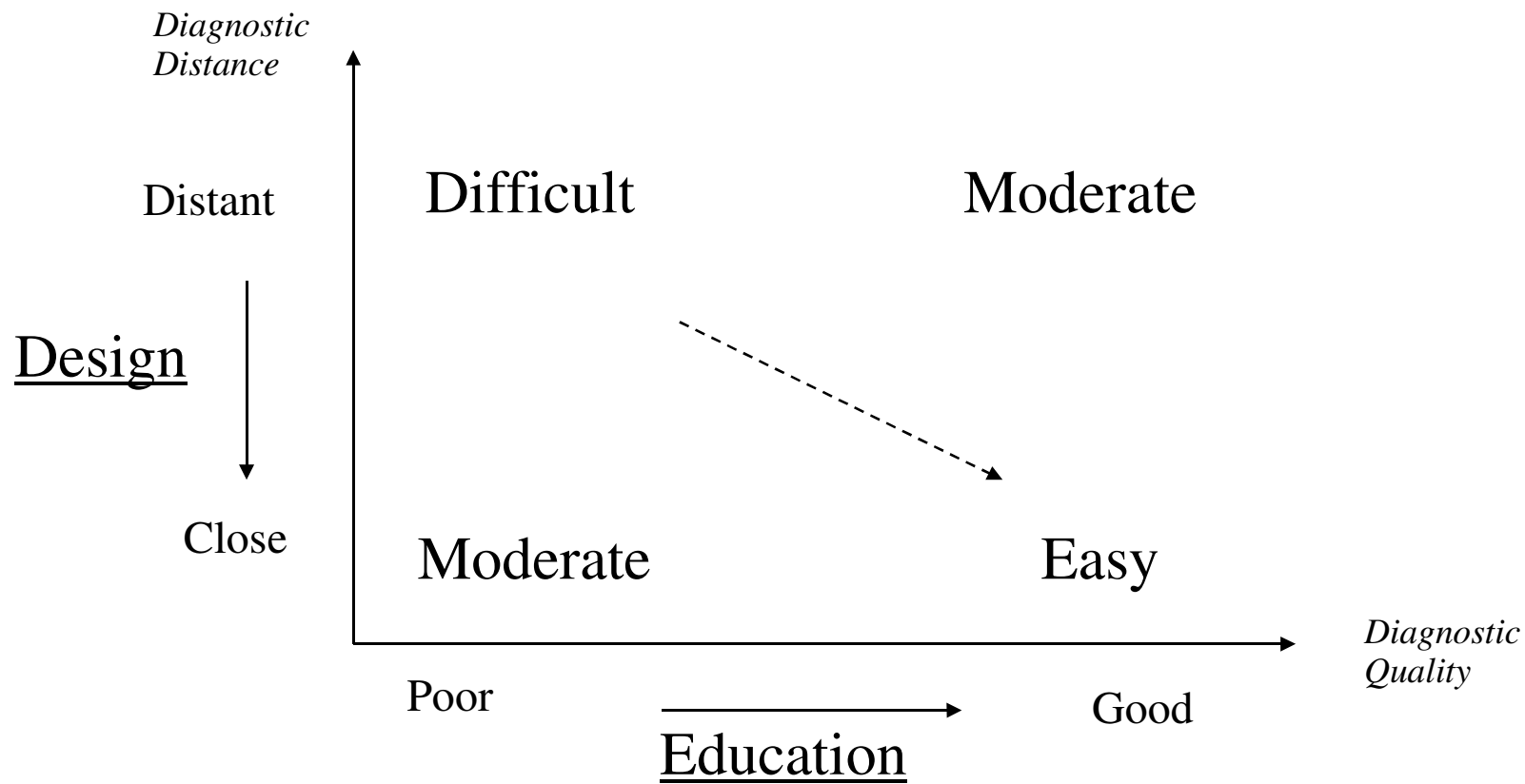
Translation into English:-

The printer has run out of paper

(Try explaining this to a thirsty native)



What we would like to do



Overall Summary

To conclude:

- On the negative side
 - ◆ We don't learn from our mistakes
 - ◆ System diagnosis is at a very poor stage of evolution
- On the positive side
 - ◆ Some technologies are extraordinarily effective

