2006-

“A measurement perspective on testing”

by

Les Hatton

Professor of Forensic Software Engineering,
CISM, University of Kingston, UK
lesh@leshatton.org, l.hatton@kingston.ac.uk

Version 1.1: 17/Sep/2006

This presentation is available at http://www.leshatton.org/

Overall philosophy

To improve a process, you must

- Define a measurable criterion for improvement
- Analyse the measurements regularly to understand what process changes will lead to improvement

For software testing, this means

- We have to have good objective measures. Since a successful test finds defects, *released defects* is an ideal candidate.
- We have to analyse all defects to link them to deficiencies in the test process, if any.

How good is good ?

A useful criterion

- Define a defect as a fault that has failed
- Define an executable line of code as any line of code which generates an executable statement
- Define asymptotic defect density as the upper bound of the total number of defects ever found in the product's entire life-cycle divided by the lines of code

If your asymptotic defect density is < 1 defect per KXLOC (thousand executable lines of code), you are doing about as well as has ever been achieved.

How bad is bad ?

NIST (US National Institute of Standards and Technology)

- 2002 report estimating costs of software failure in US alone at $60 billion per year
- 80% of software development costs are finding and fixing defects
  - Economist Science Technology Quarterly 19/Jun/2003

Royal Academy of Engineering (UK) 2004, reported

- Only 16% of projects in the UK were considered successful
- This suggests that around GBP 17 billion will be wasted in 2003/2004 alone.
  - “The challenges of complex IT projects”, 22/Apr/2004

Overview

- **Where do we start ?**
  - Aristotleans v. Babylonians: the role of measurement
  - Some examples
- **Complicating factors**
- **Measurements and how to test them**
- **Searching for patterns**

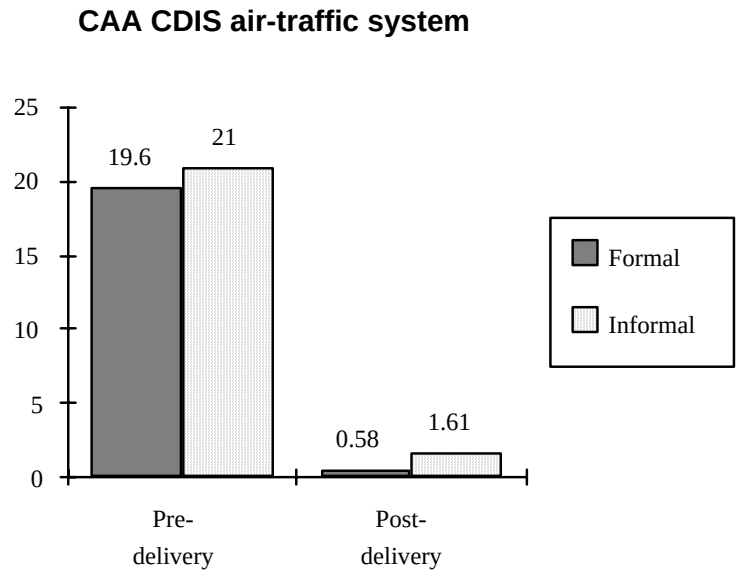
Aristotleans v. Babylonians

- **Aristotleans …**
  - Decide everything by deep thought
- **Babylonians …**
  - Learn the hard way by sticking their fingers in light sockets.

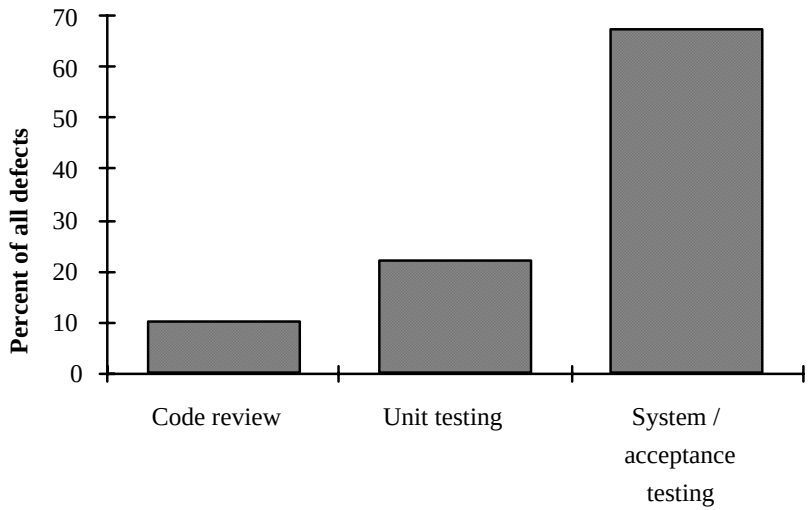
Development is an Aristotlean activity …

… Testing is a Babylonian activity

An example from air-traffic control



But before delivery they have no noticeable effect !



All is revealed: Over-reliance on dynamic testing to remove defects

Given: This company thinks formal specifications are wonderful

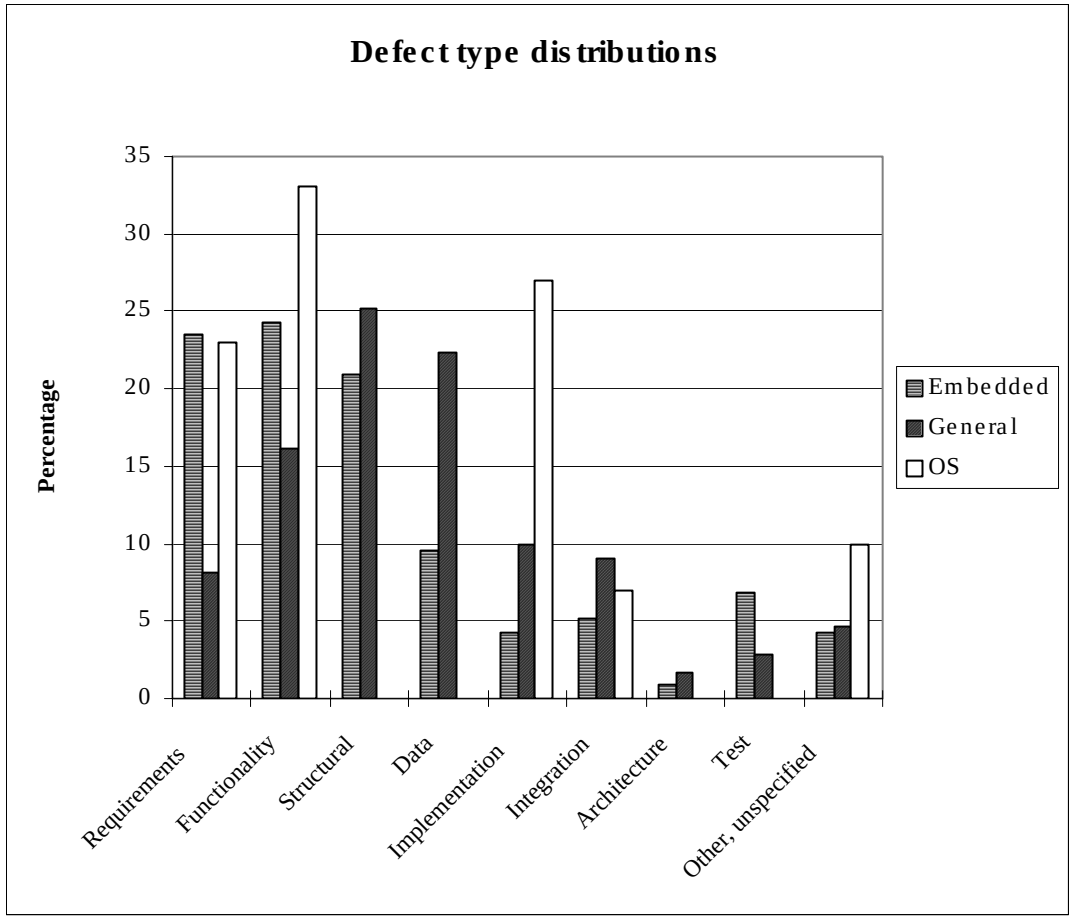
Clues from air-traffic control

- **There is evidence that people over-depend on a favourite technique**
- **There is evidence that different techniques attack different kinds of defect**
- **There is over-whelming evidence that if you measure where your defects occur, you can improve the testing process to remove them in future products**
- **There is strong evidence that the speaker is a serious nerd**
- **There is evidence that some kinds of defect are much more numerous than others**

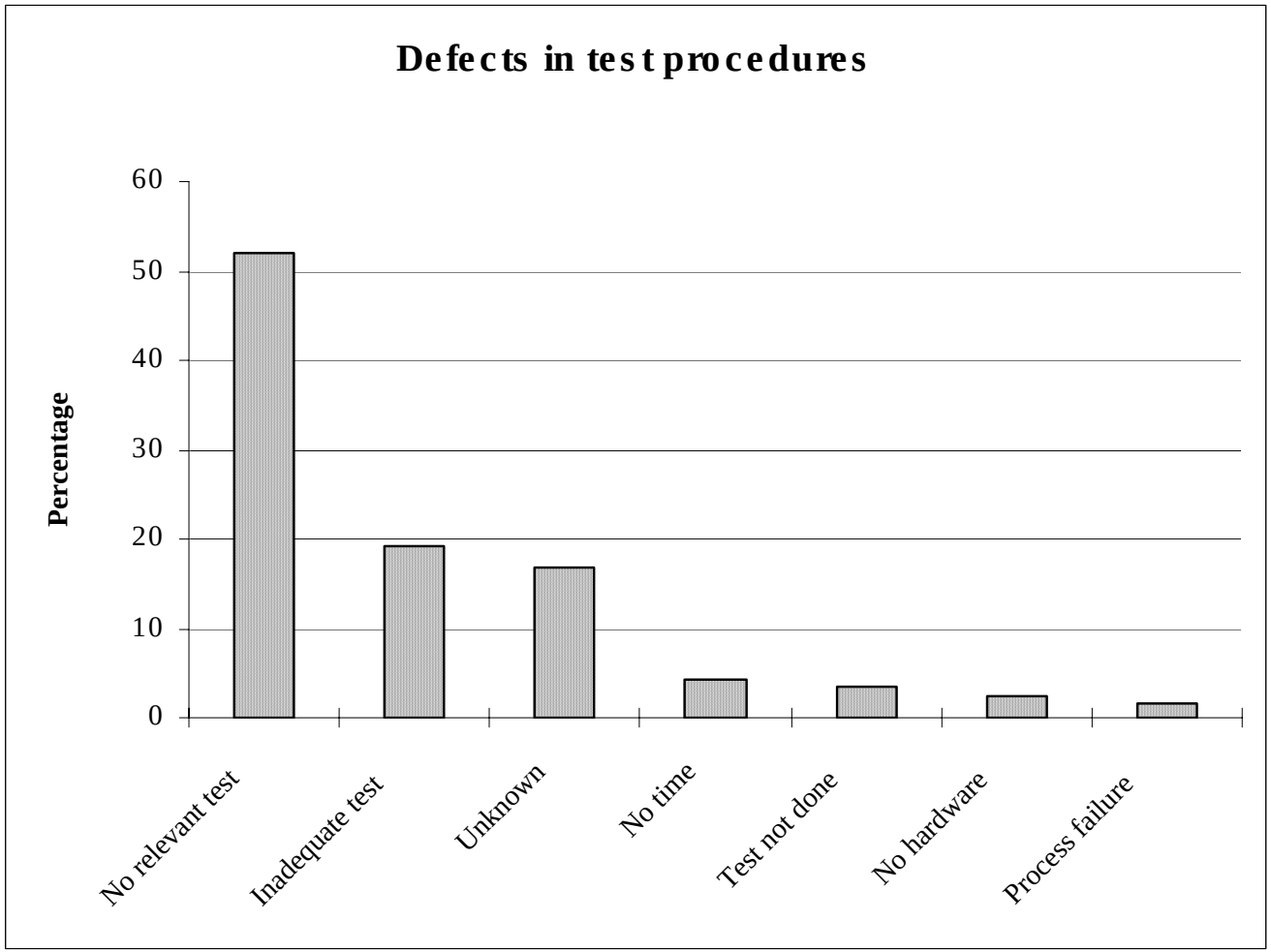
But which …

Typical defect profiles

Source:
Vinter and Poulson (1996)



Defects in test procedures - OS



Overview

- **Where do we start ?**
- **Complicating factors**
  - Architecture
  - We should test “systems” as well as software
  - Spreadsheets
  - Testers are still not involved early enough
  - The underlying architecture is not good enough
- **Measurements and how to assess them**
- **Searching for patterns**

Architecture

- **Some architectures are much more testable than others**
  - Good
    - Client / Server
  - Not so good
    - GUI
    - Embedded systems
  - Terrible
    - Real-time, high-integrity systems

We test "systems" as well as software

- **Systems inconveniently include human users**
- **Testing involves testing the system as a whole**
  - This may include a mixture of computerisation and additional, poorly documented, time varying and somewhat erratic human behaviour

The speaker versus British Gas, August 2006

- **Note the following**
  - Gas Bill based on two estimates:
    - Human sub-system – They now require me to enter my own reading on their telephone entry system
    - Software sub-system – a telephone entry system attached to their billing database
  - My attempts …
    - Enter my account number.
      - Accepted without repeating to check
    - Enter my gas bill reading
      - Repeats to verify and then says "does not agree with our records"
    - Enter same number
      - Repeats to verify and accepts

The speaker versus Barclays Bank, September 2006

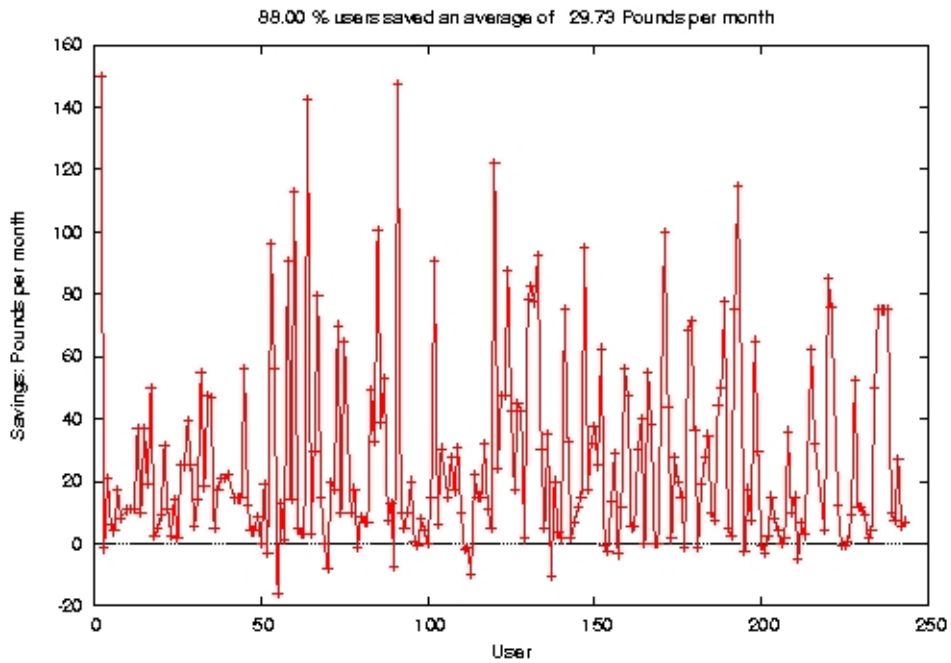
- **An attempt to transfer money from a company USD account to a GBP account in the same bank.**
  - Procedures
    - Human sub-systems – System changed without notification to central facility; new user account number / password system
    - Software sub-system – accounts database with additional account number verification fields

The speaker versus Barclays Bank, September 2006

- **An attempt to transfer money from a company USD account to a GBP account, continued …**
  - My attempt (a 25 minute phone call altogether)
    - Person 1
      - Goes through normal password procedures and then asks for account number but does not know what to do if latter missing so …
    - Person 2:
      - Does the same ☹ and then tells me new account number will reach me in two weeks and passes me to …
    - Person 3: goes through procedures again
      - Goes through procedures again but cannot do USD -> GBP transfers
    - Person 4:
      - Does the transfer. When solicited for direct number it turns out to be the number I used to call.

More complexity layering - Mobile phone charges

Illustrates the natural growth of complexity *when technology allows*



Fuzzy global optimisation web server finding minimum phone charges

Average saving £29.73 per month

http://www.betterdeal.co.uk/

Spreadsheets

- **One of the great liberating applications of the 80s and 90s**
- **One of the major headaches of the 21st century**
  - Weird arithmetic
    - -x^2+1 != 1-x^2 (Allan Stevens 2005)
  - People keep data in them instead of in databases
    - This a major fly in the ointment in most companies because people cannot exchange data.
  - They are hard to test and consequently full of defects
    - 90% of all spreadsheets had errors which led to more than 5% error in the results. Ray Panko (University of Hawaii)

There are already problems with in-car electronics

Some examples …

- 14/Apr/2004. Ford is recalling 363,440 of its 2001-2003 Ford Escape vehicles due to software problems in power-train causing engine stalling.

  Detroit News

- 17/Mar/04. 2003 US vehicle recalls hit 19.5 million in spite of ‘engineering never being better’. Experts cite problem-prone computers as significant factor.

- 09/Mar/04, Toyota faces US safety investigation and potential recall of 1 million of its best-selling Camry and Lexus ES300 sedans because of reports of unexpected acceleration causing 30 crashes.

  Detroit Free Press

There are already problems with in-car electronics

More examples …

- 06/02/2005. Whole string of problems, shaking Mercedes, Ford that bakes back seat passengers …

  http://www.nytimes.com/2005/02/06/automobiles/06AUTO.html

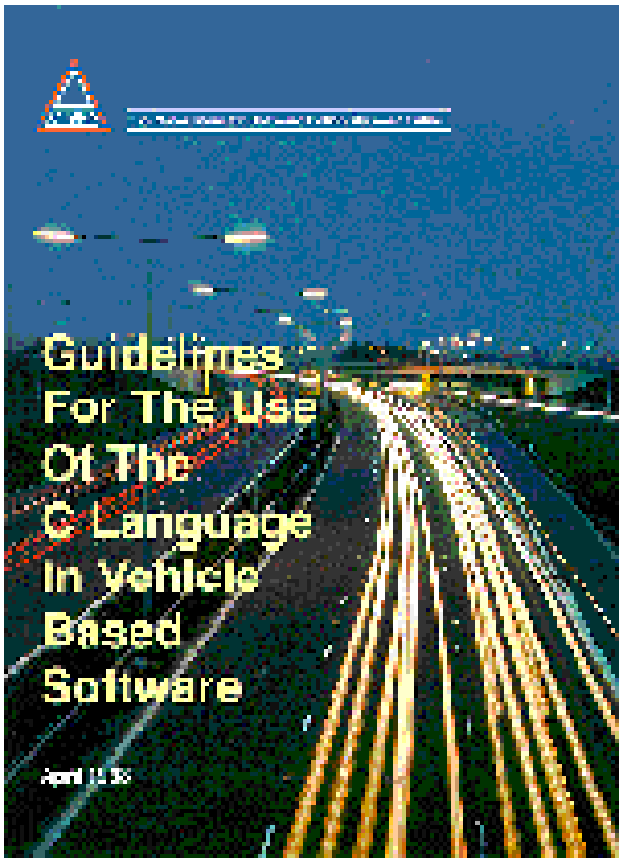
- 26/10/2004. BMW disables dynamic stability control and ABS. Two police drivers vindicated after investigation.

  http://www.daserste.de/plusminus/beitrag.asp?iid=254

Solution

Interfere without any form of test input or measurement support and make it worse …

No test input: Coding standards -MISRA C



- **The test suite was constructed after the standard was published**
  - *24% of the rules were sufficiently imprecise that test cases could not be constructed at all or could not be constructed without breaking other rules.*

  (http://www.leshatton.org/ for more details)

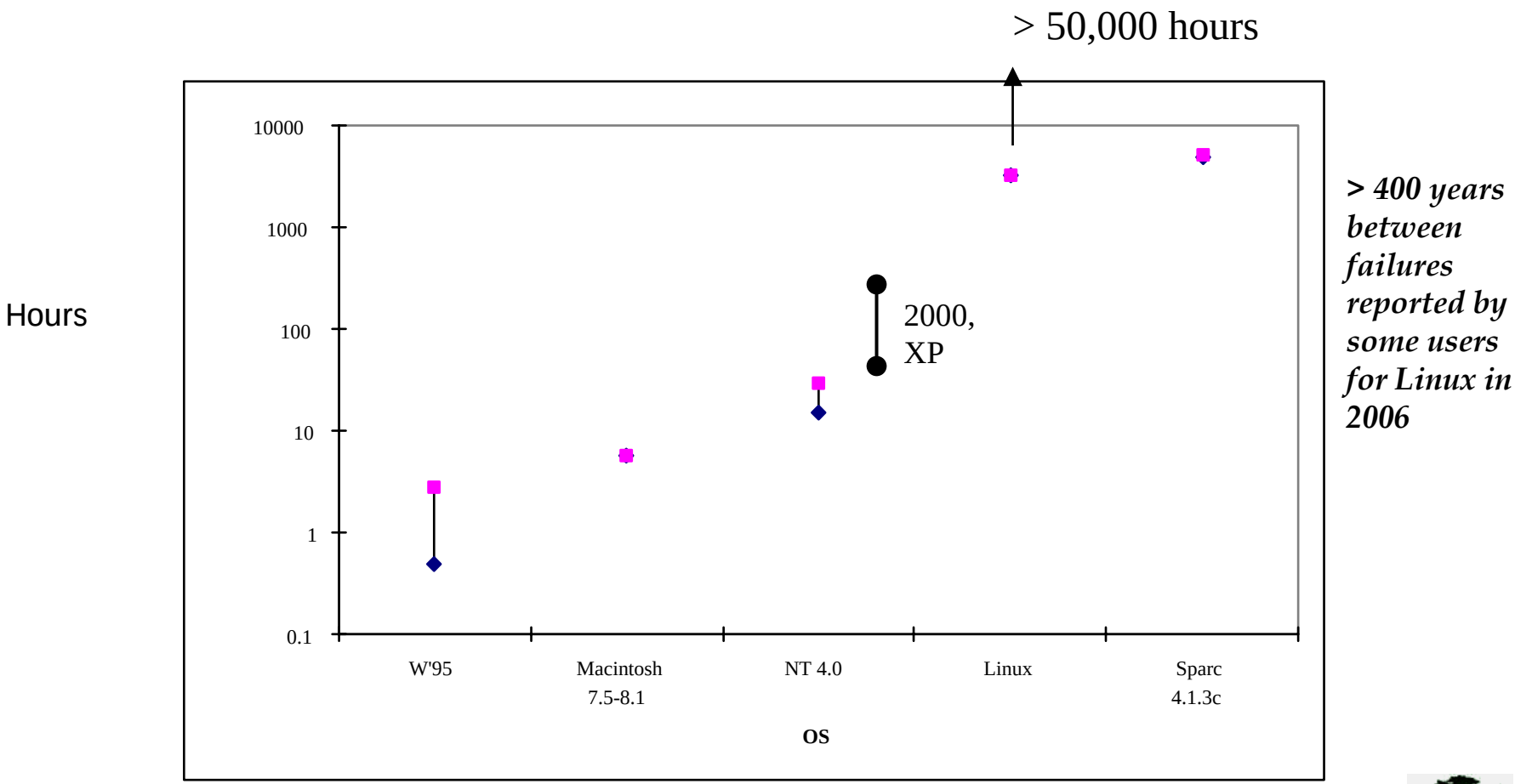
  Tests should be designed concurrently with specifications to make sure they are feasible.

Result …

Very high false positive ratio, (> 50:1)

- The code is statically tested against MISRA C and deviations must generally be corrected
- Ed Adams (1984) showed that about 1 in 7 corrections introduces a new defect, (the “if it ain’t broke, don’t fix it” rule)
- An analysis shows that this will almost certainly make the code worse than before …
    - http://www.leshatton.org/MISRA_comp_1105.html and IST (2006).

OS Reliability



Mean Time Between Failures of various operating systems

OS Reliability

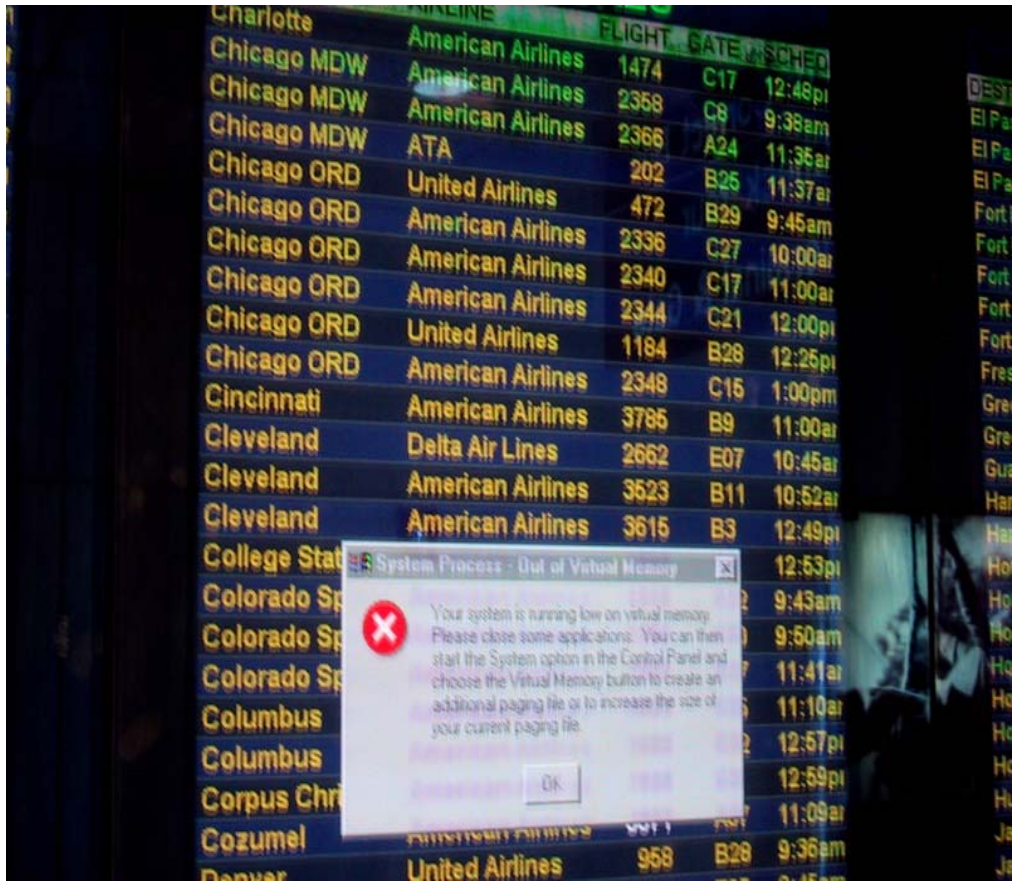
24.5 million XP crashes per day

- http://www.pcmag.com/article2/0,4149,1210067,00.asp

5% of Windows Computers crash more than twice a day

- http://www.nytimes.com/2003/07/25/technology/25SOFT.html

OS Reliability



OS Reliability

Modern responsibilities for Testers

- **Tell people loud and clear when they are proposing or building stupid systems**
- **Get involved early**
- **Include the underlying architecture**

Overview

- **Where do we start ?**
- **Complicating factors**
- **Measurements and how to assess them**
  - How not to present a result
  - How to present a result
- **Searching for patterns**

Case History 1 – How NOT to present numerical results

The proportion of defects found by external users in this case history of a client server architecture is as follows, (Hatton (2006), IEEE Computer):-

Component	Proportion of defects found externally	Proportion of defects found internally
GUI Client	***57.2%***	42.8%
Computation Server	***39.1%***	61.9%

Conclusion:- *External users are more sensitive to defects in GUI clients than in computational servers.*

Nonsense !

Case History 1 – How to present numerical results

Use the z-test for proportions and assume as a null hypothesis that the distribution of defects found externally by users in the GUI client (p_c)and the computation server (p_s) are in the same proportion.

Then …

$$z = \frac{p_s - p_c}{\sqrt{\hat{p}\hat{q}\left\{\frac{1}{n_1} + \frac{1}{n_2}\right\}}} \sim N(0,1)$$

This gives z = -0.84 which is NOT significant.

We cannot reject the null hypothesis and *we cannot infer any such pattern*

Case History 1 – What can we infer ?

Be very careful to test results for significance before using them !

Sometimes, very useful and highly significant patterns emerge …

Continued testing after delivery reduces the defect density the user sees by about half, providing we can update them regularly.

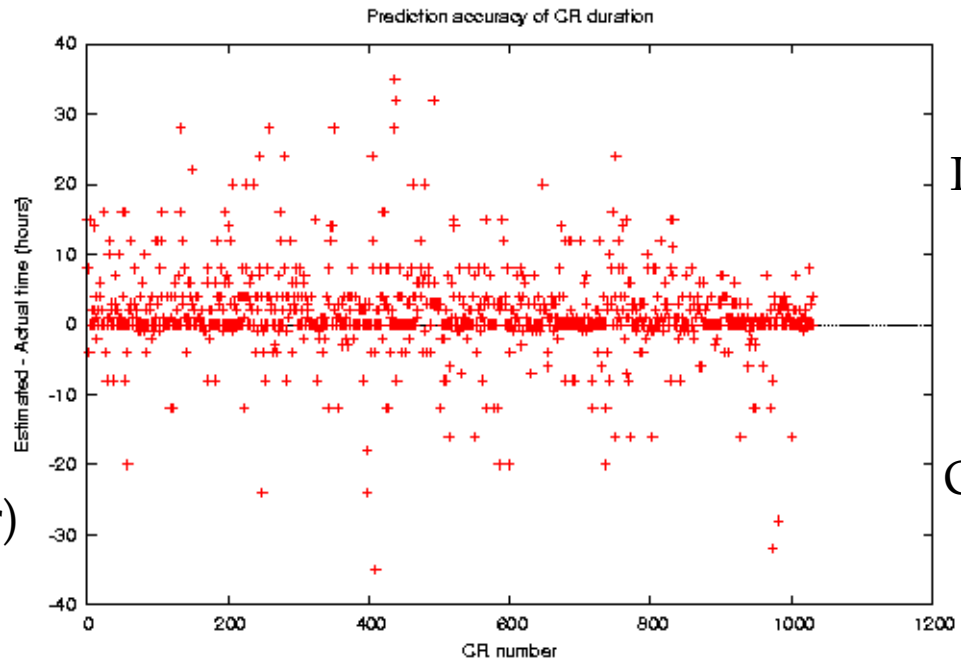
This result is statistically highly significant.

Case History 2 – How good are we at estimating tasks ?

Time →

The difference between Estimated and Actual times of maintenance tasks in a software development project.

(Hatton (2006) IEEE Computer)



Pessimistic

Optimistic

There is very highly significant systematic pessimistic bias but does it change with time ?

Case History 2 – Treading more carefully

The average amount by which engineers over-estimated maintenance tasks in this case history was:-

Data set	Average over-estimate in hours
First half	***2.45***
Second half	***1.2***

Before springing to conclusions, we test it this time …

Case History 2 – How to present numerical results

This time use the z-test for the difference of means in populations, split the population in half and assume as a null hypothesis that the systematic bias does not change.

Then …

$$z = \frac{\overline{X}_1 - \overline{X}_2}{\sqrt{\left\{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}\right\}}} \sim N(0,1)$$

This gives z = 3.16 which is VERY HIGHLY significant.

We reject the null hypothesis and *infer the bias is getting less, since*

X1 = 2.45 hours and X2 = 1.2 hours

Case History 2 – What can we infer ?

Referring to the same paper, averaging across all the data the following are statistically highly significant …

On average, engineers over-estimate how long maintenance tasks take (corrective, adaptive or perfective) by about 35%

Engineers systematically over-estimate how long a short task will take and under-estimate how long a long task will take.

Engineers improve in estimation skills significantly as projects develop.

Lessons for Testers

Keep careful defect data

Analyse it for statistical significance

Use ONLY statistically significant results to improve your tests and your resource estimation

Overview

- **Where do we start ?**
- **Complicating factors**
- **Measurements and how to test them**
- **Searching for patterns**
  - Problem: Defect data is usually disorganised
  - Searching for relationships in disorganised data

Chance discovery

- **Chance discovery is a class of algorithms to discover relationships in unstructured data**
- **Chance Discovery uses sophisticated statistical correlations after analysing plain text for significant phrases.**
- **Defect data is usually kept in an unstructured way, for example the Common Vulnerabilities Database**
  - 454,000 lines, 17Mb of unstructured English

Chance discovery

- **An example from the Common Vulnerabilities Database, CVE-2006-4304:-**

  “Buffer overflow in the ppp driver in FreeBSD 4.11 to 6.1 and NetBSD 2.0 through 4.0 beta allows remote attackers to cause a denial of service (panic), obtain sensitive information, and possibly execute arbitrary code via crafted Link Control Protocol (LCP) packets with an option length that exceeds the overall length, which triggers the overflow in (1) pppoe and (2) ippp.”

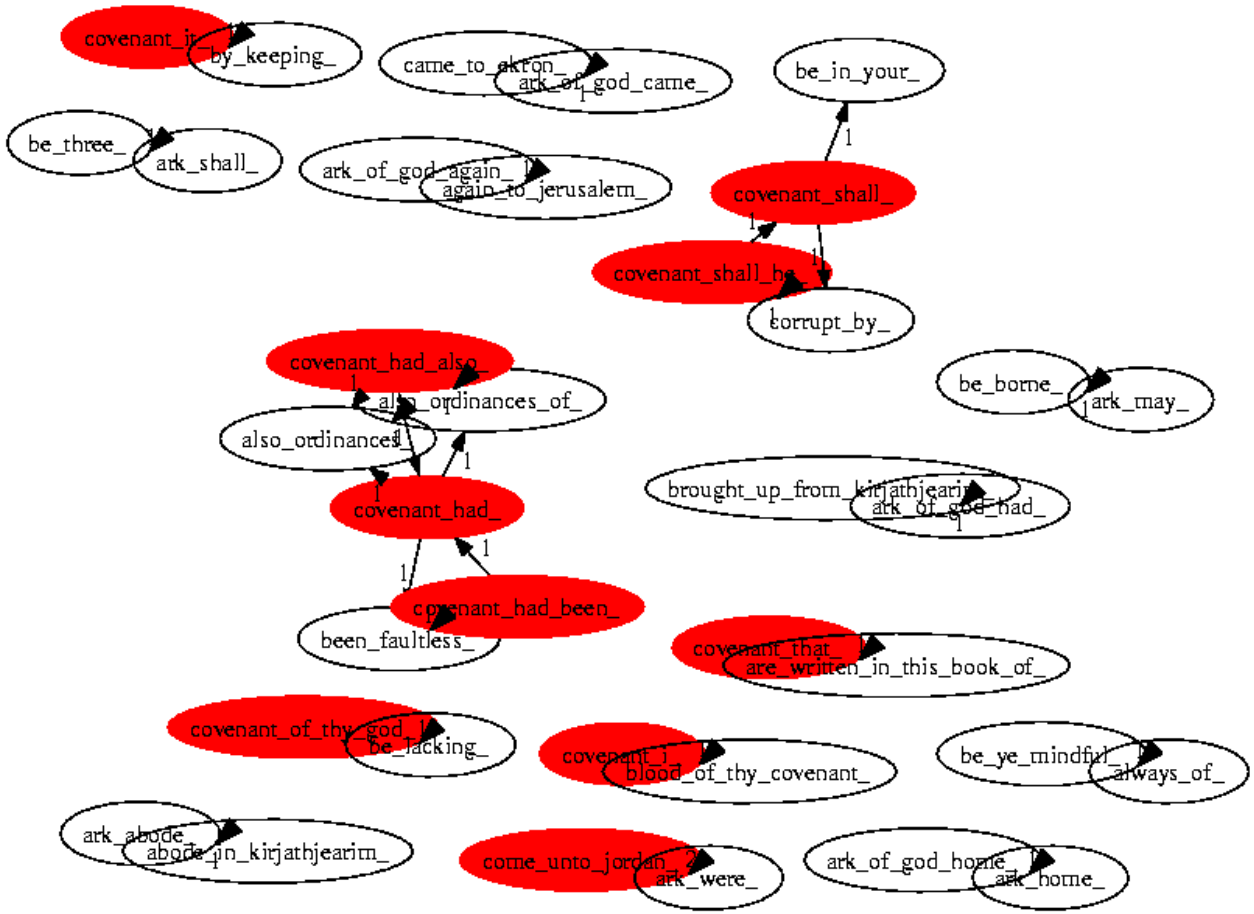
Chance discovery – example 1

Let's start slowly with the King James Bible, (downloaded from Project Gutenberg, http://www.gutenberg.org/). This is about 4 times smaller than the Common Vulnerabilities Database.

- **Chance Discovery Analysis looking for relationships between "Ark" and "Covenant" …**

Chance discovery – example 1

Search took 7 secs

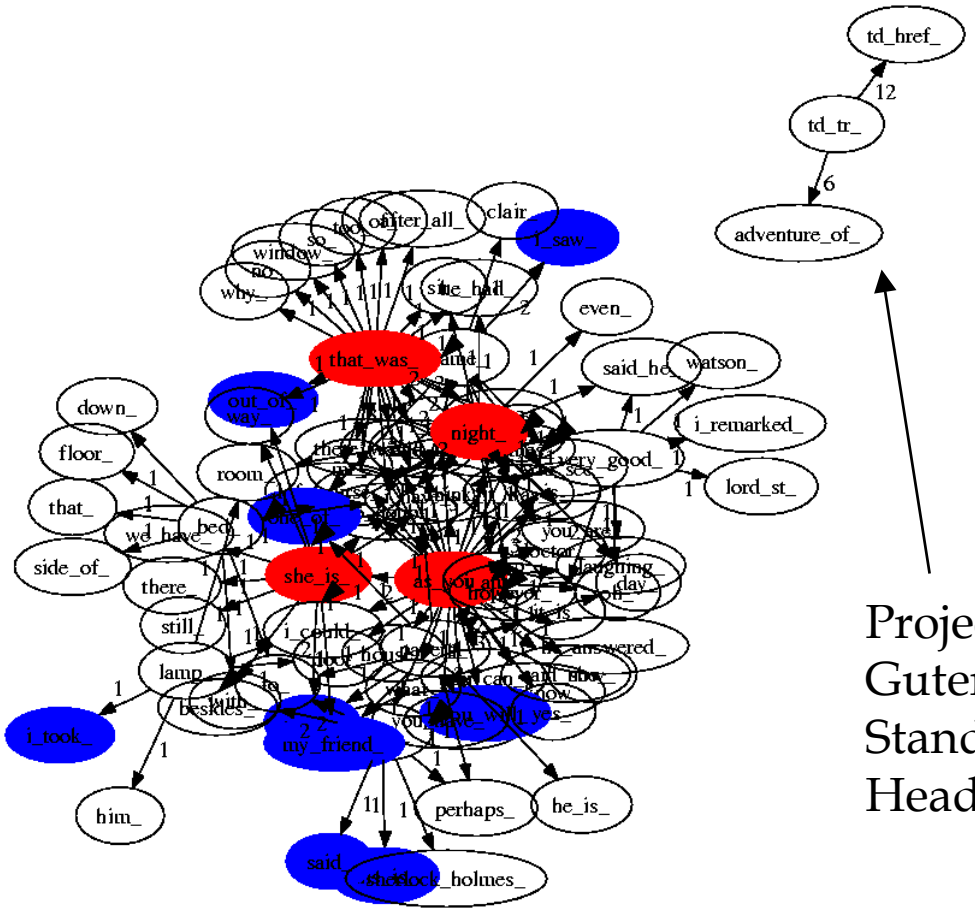


Chance discovery – example 2

- **The Adventures of Sherlock Holmes, (downloaded from Project Gutenberg)**
- **Chance Discovery Analysis looking for any relationships …**

Chance discovery – example 2

Search took 12 secs



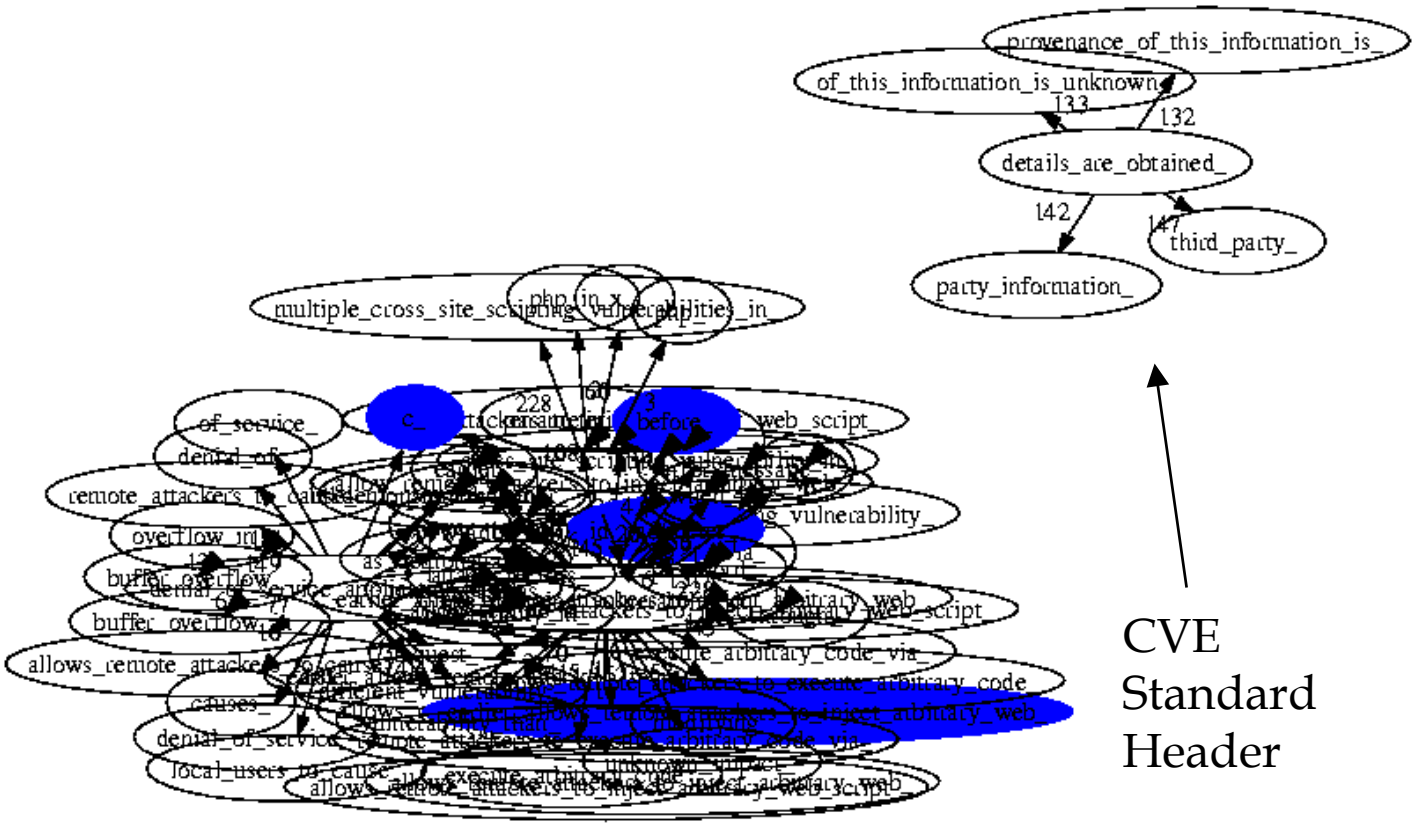
Project Gutenberg Standard Header

Chance discovery – example 3

- **The Common Vulnerabilities Database, (downloaded from http://www.mitre.com/)**
- **Chance Discovery Analysis looking for any relationships …**

Chance discovery – example 3

Search took 12 minutes



General conclusions

- **Improving test strategy efficiency requires:-**
  - Test and defect measurements and the patterns concealed in them
  - Very early involvement with design and specification
  - Awareness of the properties of failure and the intended use and ubiquity of the product
  - Awareness of how good is good
  - Better education
- **It also makes sense to …**
  - Continue to test after a product is released, (assuming product updates are economic to do)

For further information …

This paper and many other downloadable papers and pieces of software can be found at:-

http://www.leshatton.org/