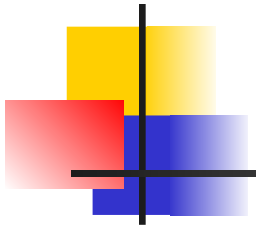
What do we do about “untestable” components ?

Les Hatton

Emeritus Professor of Forensic Software Engineering
SEC, Kingston University, London
lesh@oakcomp.co.uk

Version 1.3: 06/Nov/2017

Overview



There has never been a time when software testers were needed more …

Recalls in the automotive industry…

- 2016. Stout Risius Ross Warranty reports software related recalls have gone from 5% in 2011 to 15% by end of 2015. (189 distinct recalls in 5 years covering 13 million vehicles with 141 presenting higher risk of crashing).

  http://popsci.com/software-rising-cause-car-recalls

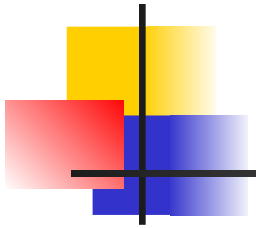
- 2016. First known self-driving death in Tesla.
- 2015 June. Ford recall 433,000 cars for engines that don't shut off.
- 2009 - Oct 2013. Toyota unintended acceleration

  http://safetyresearch.net/Library/BarrSlides_FINAL_SCRUBBED.pdf

- 20/Mar/2013 GM recalls vehicles for software and fuel-system problems

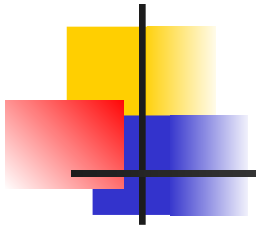
  Wall Street Journal

Current size of software systems



- The Impact columns (IEEE Software 2010-)
  - About 30 systems described, several of them safety-related.
  - The majority appear in the 1-10 million SLOC range (including the safety-related systems).
  - Automotive systems appear to be in the 10-100 million SLOC range as we enter the autonomous driving age.

Quantifying software growth



- So, how fast are systems growing ? This we can answer …
  - On a corpus of 800 million lines of code, the 95% confidence interval of annual growth is (18%, 22%). So they double in size about every 42 months.
  - Safety-related systems appear to be about half as fast.

(Hatton, Spinellis, van Genuchten, 2017).

Cheating and attempts to put it right …

- 27/Jul/2017. Porsche to recall 22,000 cars over emission software.

  http://www.bbc.co.uk/news/business-40740886

- 2015 September. VW finally admit to emissions software cheat which will cost them around $21 billion.

  https://www.theguardian.com/business/2017/mar/10/volkswagen-vw-pleads-guilty-criminal-charges-emissions-cheating

  Interestingly, it seems as though the fix itself has introduced further bugs in some cars …

  https://www.honestjohn.co.uk/news/volkswagen-emissions-scandal/8-reports-of-volkswagen-tiguans-losing-power-after-ea189-emissions-fix/

Security implications ? …

Car Navigation Systems

- 04/08/2017. “Researchers hack self-driving car”.

  https://www.autoblog.com/2017/08/04/self-driving-car-sign-hack-stickers/

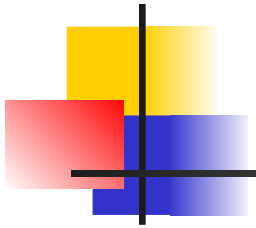
- 01/08/2016. “Jeep hackers are back”.

  https://www.wired.com/.../jeep-hackers-return-high-speed-steering-acceleration-hacks/

- 21/07/2015. “Hackers remotely kill Jeep on highway”.

  https://www.wired.com/2015/07/hackers-remotely-kill-jeep-highway/

Overview

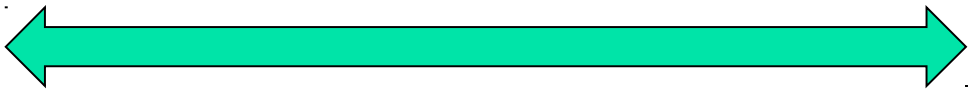


- The role of evidence
- The bureaucratic urge
- A case history – untestable components

The role of evidence

Science	Politics
Evidence => Policy	Opinion => Policy
Improvement via Scientific Method	Improvement Random

Testing ?



The evidence matrix

Level of evidence	Example area	Source of threat
Irrefutable	Engineering	Generally immune – hard to resist the laws of physics
Considerable	Medicine	Media, Political
None -> Little	Software systems	Media, Political, managerial/economic

Overview

- The role of evidence
- The bureaucratic urge
- A case history – untestable components

The bureaucratic urge

- Proliferation of documentation
- “Something must be done” – panic and proportion, sometimes with opposing evidence
- Risk assessment – a frightening new weapon in the battle against common sense

Proliferation of documentation: ISO Human interface standards

9126

9241

13407

13406

61997

18529

14598

10741

10075

18021

18019

14915

11581

18789

14754

15910

16982

16071

20282

Proliferation of documentation: The Nimrod explosion, 2006

Daily Mail, 29/Oct/2009

- MOD Head of Air: “Complete failure to do his job in relation to the Nimrod Safety Case”
- MOD Safety Manager “Most of the time he was clearly out of his depth”
- BAE Systems Chief Airworthiness Engineer “Pushed through too quickly because he was too ambitious and assumed it was safe anyway”
- BAE Flight Systems and Avionics manager “Significant responsibility for poor planning, poor management and poor execution of project”
- QINETIQ Task manager of Safety Case project “agreeing on behalf of Qinetiq to sign off project without seeing or reading the reports”
- QINETIQ Technical Assurance Manager for Nimrod Safety Management “Guilty of allowing the manifestly inappropriate BAE reports to be approved”

Note the extensive safety bureaucracy and titles

Proliferation of documentation: The Nimrod explosion, 2006

"The Nimrod Safety Case was a lamentable job from start to finish. It was riddled with errors, it missed key dangers, its production is a story of incompetence, complacency and cynicism."

Charles Haddon-Cave QC

Panic and proportion: Swine flu – millions will die

- Massive over-reaction, stoked by a rapacious media
- 10x diagnosis rate in the UK because of remote diagnosis – knee infections and in some cases meningitis were treated as swine flu.
- Significant side-effects from Tamiflu in children and ultimately the anti-flu vaccination in adults

A frightening new bureaucratic weapon: Risk Assessment

If R is the Risk, F the Frequency and C the Consequence:

R = F x C

So unlikely catastrophic events have a similar risk to very frequent but unimportant events.

Mathematicians always seek to quantify risk.

Detecting risk assessors in the wild

http://www.flickr.com/photos/fraserspeirs/4090224, student canteen at Brunel University

Problems of measurement - A genius's view of risk

"The risk of the end of the universe is definitely less than 1 in 10^5."

Risk factor from the Large Hadron Collider, CERN.

"If a guy tells me that the probability of failure is 1 in 10^5, I know he's full of crap."

Richard P. Feynmann, Nobel Laureate commenting on the NASA Challenger disaster.

Doing risk assessments so you don't get asked again

- Step 1: Insist on using R = F x C in your assessment. This will panic Human Resources.
  (People go into Human Resources to avoid nasty things like multiplication.)
- Step 2: Put "end of universe" as risk number 1
  (Rationale: R = F x C. Since the end of the universe has an infinite consequence C, then no matter how small the frequency F, the Risk is also infinite)
- Step 3: Ignore all other risks as insignificant
- Step 4: Wait for call from Human Resources

Overview

- The role of evidence
- The bureaucratic urge
- A case history – untestable components

Toyota unintended acceleration 2009-2013; some expert witness notes

Source code

- *> 11,000 global variables (NASA analysis)*
- *"Spaghetti code". 67 functions with cyclomatic complexity > 50, (throttle angle function > 100)*
- Recursion with no stack monitoring, (2005 Corolla had this !)
- Toyota's coding standard used only 11 MISRA-C rules and 5 were violated in the code.

Systems

- No adequate safeguards against memory corruption
- Redundancy problems – all fail-safes in same Task.
- No EDAC (Error Detecting And Correcting) memory

http://www.safetyresearch.net/Library/BarrSlides_FINAL_SCRUBBED.pdf

Toyota unintended acceleration 2009-2013 "Untestable" Components

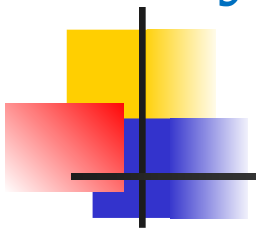
"Spaghetti code". 67 functions with cyclomatic complexity > 50, (throttle angle function > 100)

Cyclomatic complexity = #decisions + 1

Notes

- In testing terms, we often use 10 as a watershed of difficulty (McCabe 1976).
- The implication is that this is "bad".

Toyota unintended acceleration 2009-2013



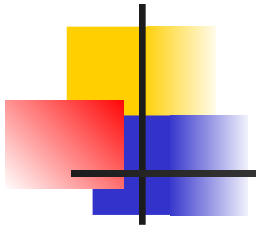
- These notes appear to consist of a mixture of good engineering analysis (e.g. the task death monitor) and software engineering folklore (global variables, cyclomatic complexity, lots of words ending in –ability …)
- The Toyota software clearly has serious concerns but our methods of analysing it are inadequate and riddled with opinion
- Can we answer these two questions ?
  - Why do we get “big” functions ?
  - Is it easier to test 10 functions with 10 decisions or 2 with 50 ?

Why do we get big functions ?

- **Software**: each box is a function and different coloured beads are different tokens
- **Proteome**: each box is a protein and different coloured beads are amino acids.

Box i: ti beads, ai different colours

General Systems Theory



We need a theory for software systems independent of

- what they do
- what language they are written in
- who wrote them
- what technology they used
- how big they are, provided they are ‘reasonably’ big.

Tokens in programming languages

Take an example from C:

```
void bubble( int a[], int N)
{
  int i, j, t;
  for( i = N; i >= 1; i--)
  {
    for( j = 2; j <= i; j++)
    {
      if ( a[j-1] > a[j] )
      {
        t = a[j-1]; a[j-1] = a[j]; a[j] = t;
      }
    }
  }
}
```

Fixed (18)

void int ( ) [ ] { , ;
for = >= -- <= +
+ if > -

+

Variable (8)

bubble a N i j t 1 2

ai = 18 + 8 = 26

Total (ti =94)

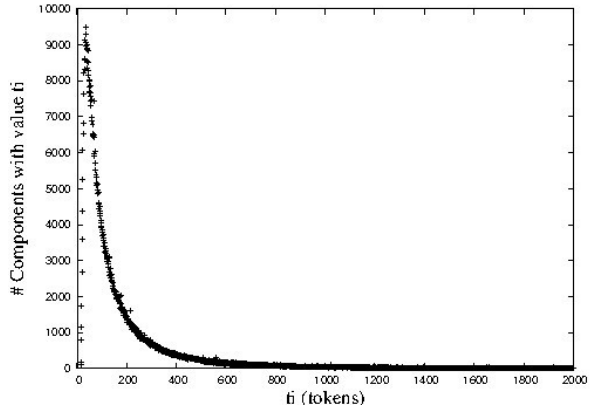
The chocolate box theorem

The CoHSI* (Conservation of Hartley-Shannon Information) length distribution of **any** discrete system is overwhelmingly likely to obey

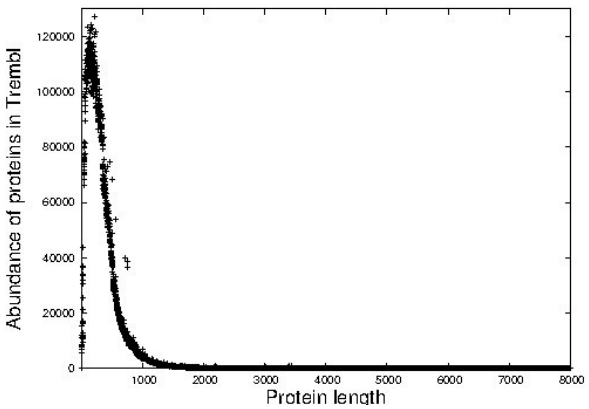
$$\log t_i = -\alpha - \beta \frac{d}{dt_i} \log N(t_i, a_i; a_i)$$

- Hatton and Warr (2017) “Information theory and the length distribution of all discrete systems”, https://arxiv.org/abs/1709.01712
- Hatton (2014) “Conservation of Information: Software’s Hidden Clockwork”, IEEE Transactions on Software Engineering, 40(5), http://ieeexplore.ieee.org/document/6784340/

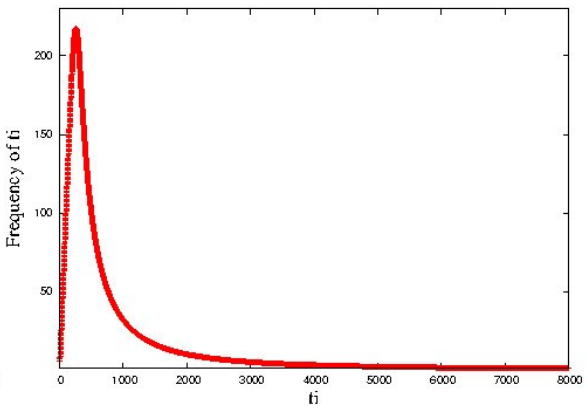
CoHSI – Conservation of Hartley-Shannon Information



80 million lines of C

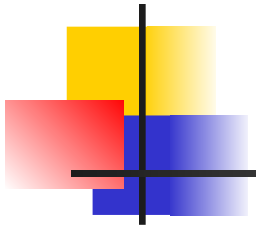


80 million proteins from the European Protein database, uniprot.org



The CoHSI distribution

CoHSI – Conservation of Hartley-Shannon Information



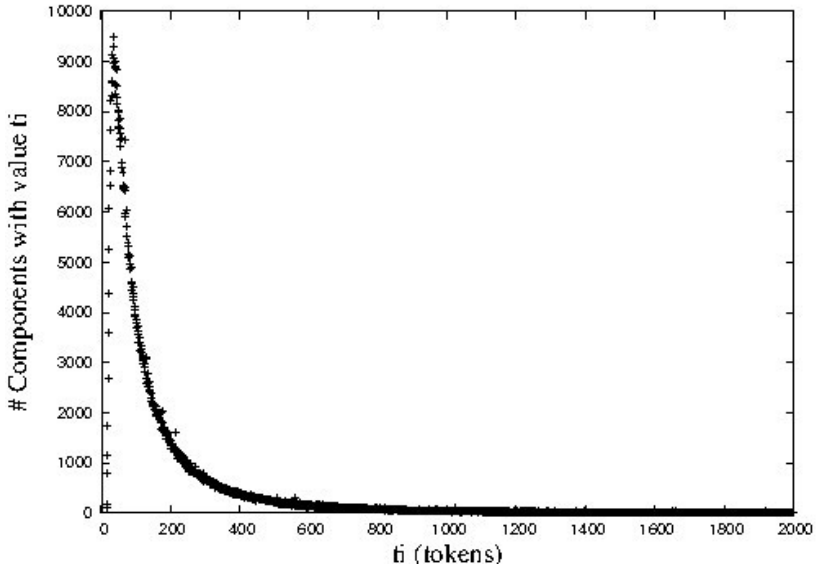
This signature is sharp unimodal peak followed by an extremely precise power-law tail.

The implications for the power-law tail of lengths in software development are profound. *It means that developers have no real control over the lengths of software components so very long components are an inevitable implication of CoHSI (Conservation of Hartley-Shannon Information).*

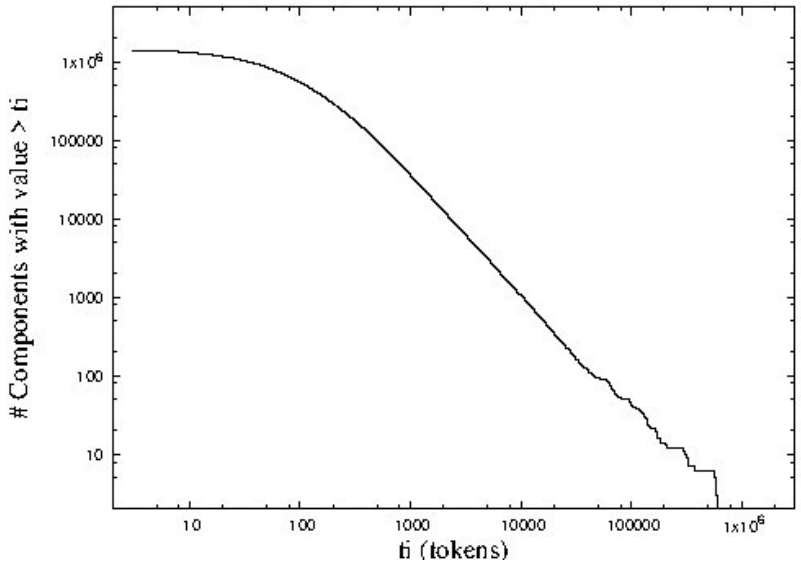
(Note the analysis of the Toyota unintended acceleration bug.)

Structure and component size: The hidden clockwork of software

How quickly does this establish itself ? The classic signature as a pdf and ccdf is shown below.

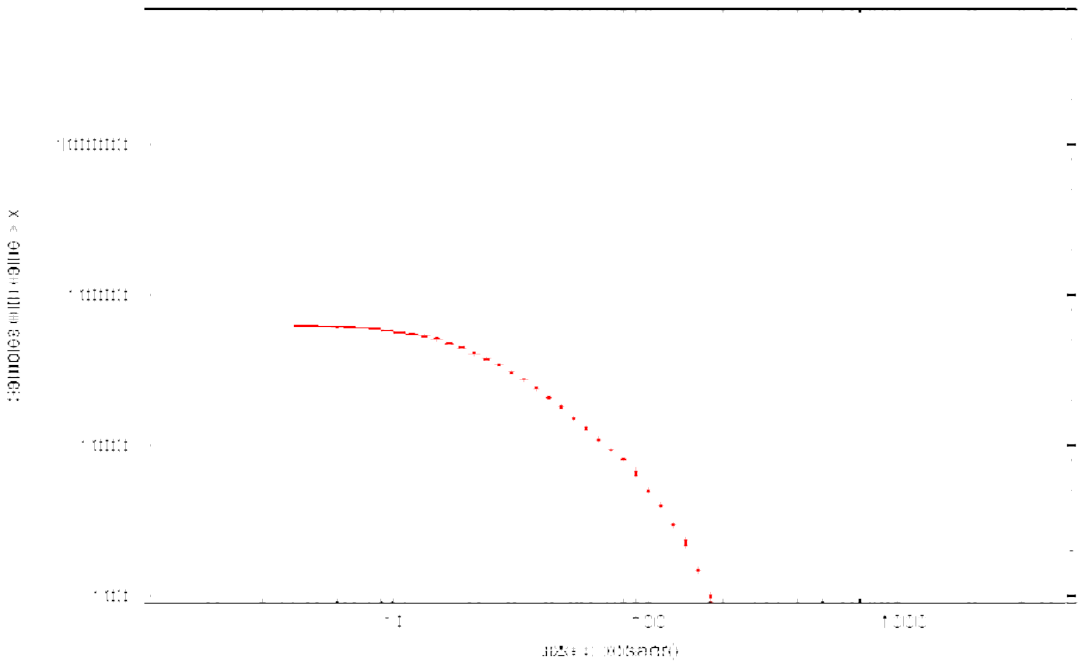


pdf



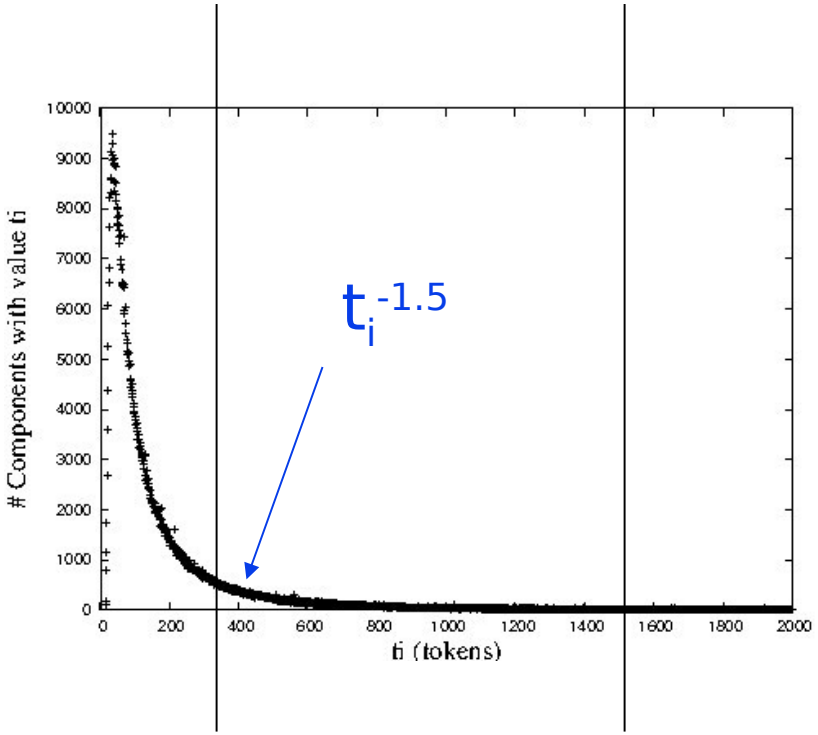
ccdf

Emergent behaviour in 40 MSLOC



40 million lines of Ada, C, C++,
Fortran, Java, Tcl-Tk from 80+ systems

Structure and component size: The hidden clockwork of software



10 decisions

50 decisions

In any software system, for every eleven 10 decision components there will on average be one 50 decision component. The bigger the system, the more accurate this becomes and you have no control over this.

The t log a defect theorem

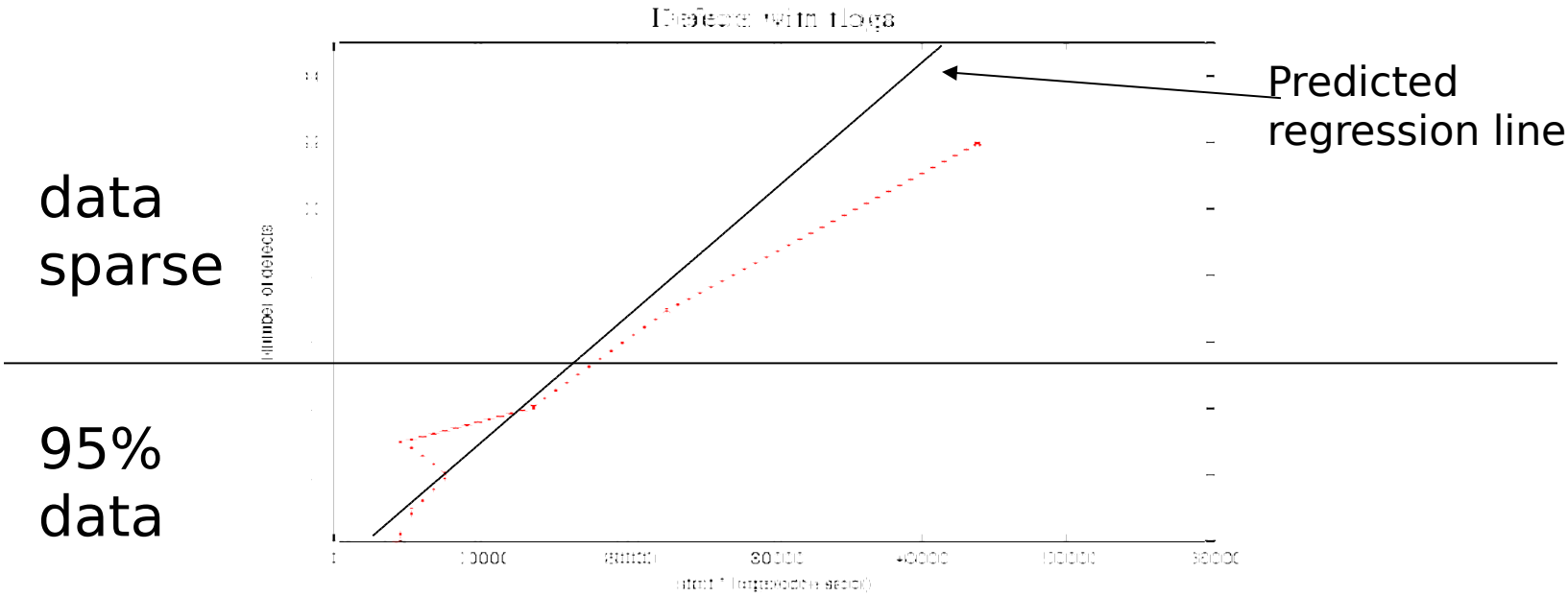
- Predicts the following defect distribution, (Hatton 2012, http://arxiv.org/abs/1209.1652)

$$d_i \approx t_i \log a_i$$

Here t_i is the total number of keywords and identifiers and d_i is the number of defects.

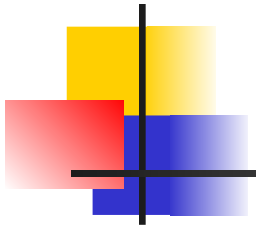
- Can we test this ?

Equilibriation to t log a in the Eclipse IDE*



*With grateful thanks to Andreas Zeller et. al. (2007) for extracting the defect data and making it openly available. http://www.st.cs.uni-sb.de/softevo. The data comes from releases 2.0,2.1 and 3.0. There are 10,613 components in the release 3.0.

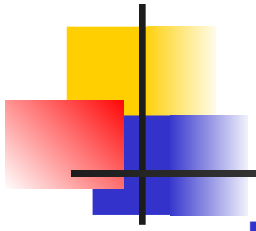
Testing 10x10 decision components v. 2x50 decision components



- Since the object of testing is to discover defects, which is likely to have the most ? We are comparing (approximately) …

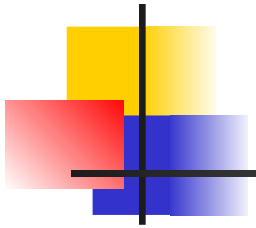
  10x300 log(50) with 2x1500 log(60). These are (statistically) very similar, so it doesn't make much difference in defect terms.

Notes for Software Testing



- Software systems are growing around 20% per year,
- Recalls due to defect are growing (hardly surprising),
- We are wallowing in largely ineffective bureaucracy, still driven by opinion not evidence,
- Large components in a system are inevitable and are not a sign of poor design.
- Large components (probably) do not add to test load. (The decisions have to be somewhere.)

References



Loads of stuff on Wikipedia:-

http://www.wikipedia.org/

My writing site:-

http://www.leshatton.org/

Bibliography

Hartley R.V. (1928) “Transmission of Information”, Bell Systems Journal, 7(3), p. 535-563

Hatton L. (2007) “Power-law distributions of component size in general software systems“, IEEE Transactions on Software Engineering, 35(4), p. 566-572

Hatton L. (2014) “Conservation of Information: Software’s Hidden Clockwork ?”, IEEE Transactions on Software Engineering, 40(5), p.450-460

Hatton L. and Warr G (2015) “Protein Structure and Evolution: Are they Constrained Globally by a Principle Derived from Information Theory” PlOS ONE doi:10.1371 /journal.pone.0125663

Hatton L. and Warr G (2017) “Proteins and Computer Programs; Inextricably linked by Information Theory”, submitted to PRSB.

Hatton L., Spinellis D. and van Genuchten M (2017) “The long-term growth rate of evolving software: Empirical results and implications.” *Journal of Software: Evolution and Process*, 2017. To appear. (doi:10.1002/smr.1847)

Hopkins T. and Hatton L. (2008) “Exploring defect correlations in a major Fortran numerical library”, http://leshatton.org/NAG01_01-08.html

Noether E. (1918) “Invariante Variationsprobleme”, Nachr. v..d. Ges. d. Wiss zu Gottingen, p. 235-257

Shannon C.E. (1948) “A Mathematical Theory of Communication”, Bell Systems Journal, 10.1002/j.1538-7305.1948.tb01338.x