“Software Defects are almost certainly inevitable"

Les Hatton

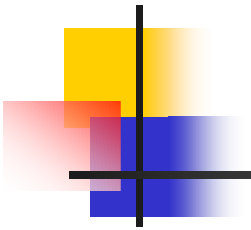
www.leshatton.org

Version 1.1: 24/May/2012

Famous last words

- Enduring misconceptions about software
  - “How could it fail, its been tested ?”. CTO of major car company after million+ recall of embedded software, 1999.
  - “The software works but is not well documented”, Article in Science, April 2012.

Overview



- A little about software systems
- Personal musings on software defects
- A hidden clockwork
- Conclusions

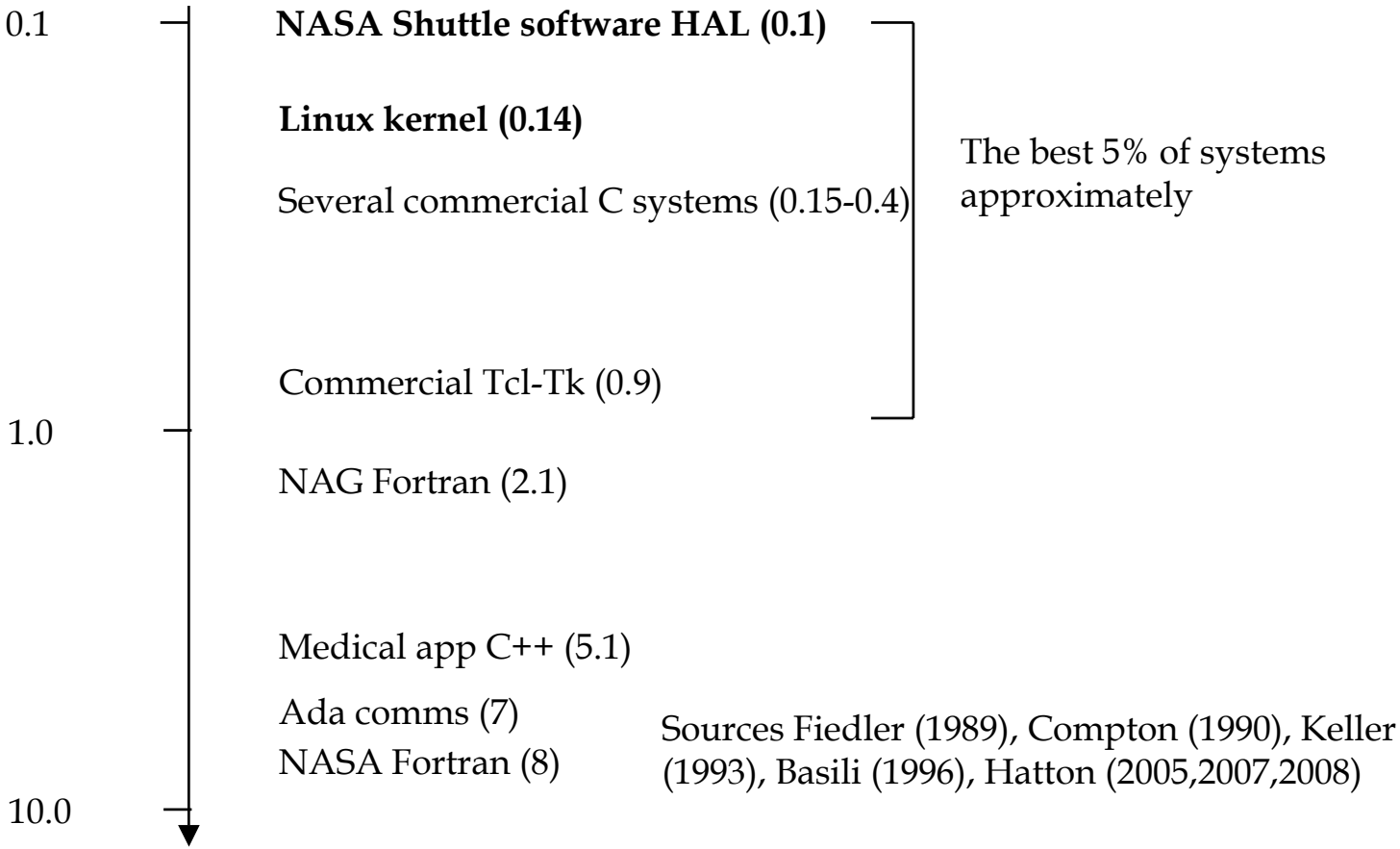
So what can we say about defect ?

- On quantification
  - Computer scientists have researched the average *density* of defect in code extensively
  - Where we have been much less successful is in quantifying the *effects* of such defect on numerical results.

So what can we say about defect ?

- On quantification of defect density
  - A “low defect” piece of software will exhibit less than 1 defect per thousand executable lines of source code in its entire lifetime.
  - Average software appears to be in the range 1-10.

A software quality scale based on defect density



Affecting you in ways you might never realise …

Scientific computation and reproducibility*

- Reproducibility is at the heart of the scientific method
- Without source code it is impossible to reproduce scientific results. (Even with it, there are considerable problems)
- Scientific work without accompanying data, source code and the means to reproduce is **not** science.

*See the discussion in Darrel Ince, Les Hatton and John Graham-Cumming (2012) "The case for open computer programs", Nature 482, p. 485-488, 23 Feb 2012.

Software growth …

Software size in consumer electronic products has been doubling about every 4-5 years for the last 20 years, (van Genuchten, Hatton, IEEE Software, Jul 2012)

- 300kLOC in an engine control unit, up to 100MSLOC in a luxury car spread across up to 80 electronic control units, (Mossinger, IEEE Software, March 2010).
- 10-100MSLOC in video player, (Bouchard, IEEE Software, June, 2010).
- Aircraft (e.g. http://oce.nasa.gov/oce/news)
  - The F/A-22 fighter has 1.7 million lines of code.
  - The Airbus A380 and Boeing 777 have 3-4 million lines.

Speaking of the F22 Raptor…

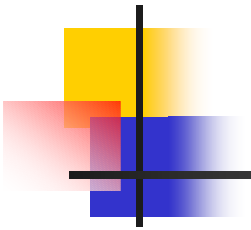
26th February 2007

6 F-22 Raptors were left without major systems when the systems crashed after crossing the International Date Line on route from Okinawa to Hawaii. “It was a software glitch – somebody made an error in a couple of lines of code out of millions.”

And Routemaster buses ...

27-Feb-2012: The launch of the new London Routemaster bus, (cost GBP 1.4 million each) was delayed by a week after a software problem meant it had to run with the rear platform shut. Perhaps they should have called it a Raptor.

Overview



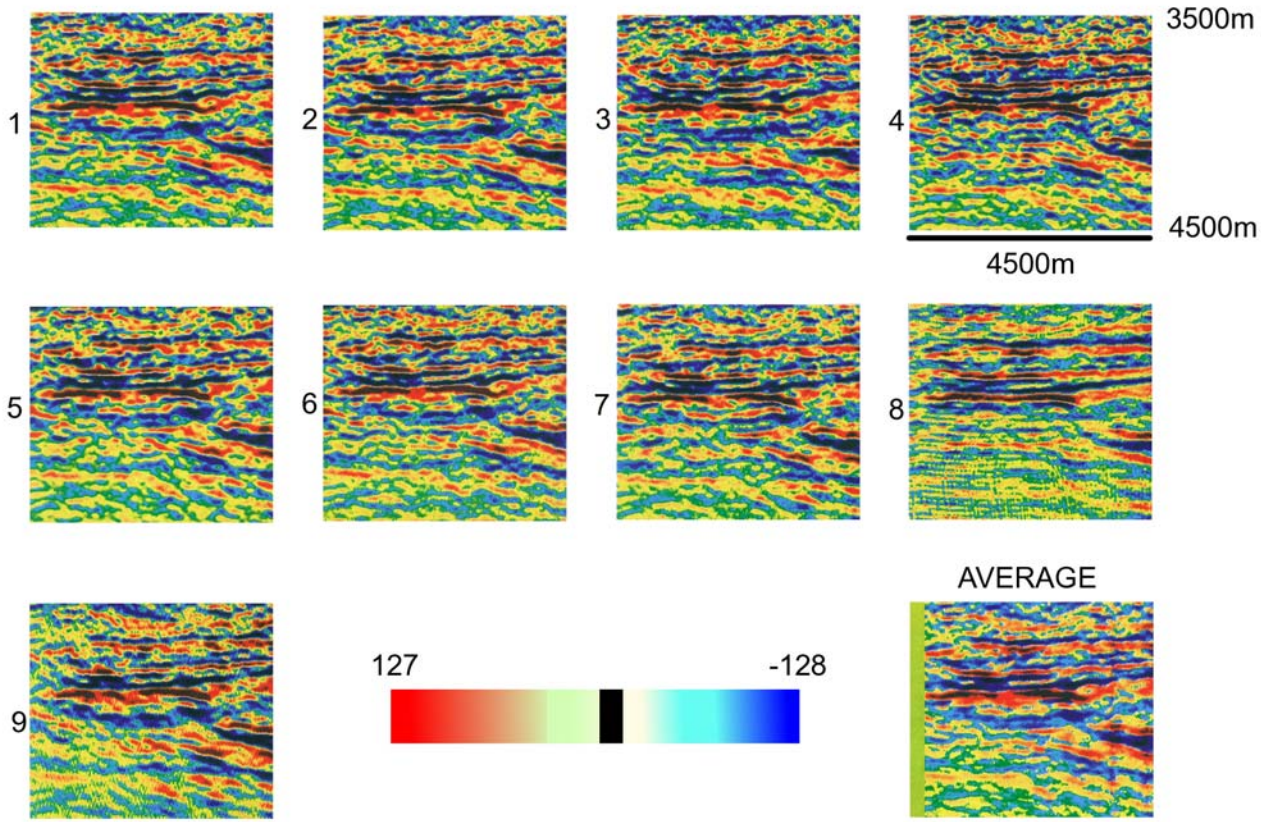
- A little about software systems
- Personal musings on software defects
- A hidden clockwork
- Conclusions

Some early thoughts

By 2010 I was reasonably convinced that:

- *N-version experiments*, although not fully independent are exceedingly valuable at highlighting differences, (for whatever reason), and effective at reducing those differences. (1994)
- Scientific software is littered with *statically detectable faults* which fail with a certain frequency (1997-1998)
- The language does not seem to make much difference. (1999-)
- Defects appear to be fundamentally statistical rather than predictive, (2005-8)
- There's a hidden clockwork (2007-10).

N-version experiments (1994) How big must a defect be before we notice it ?

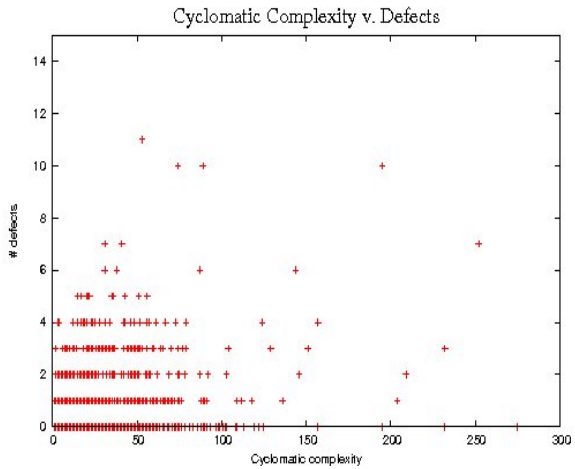


Comparison of 9 commercial packages using the same algorithms on the same data in the same programming language, (Hatton and Roberts (1994))

Are defects related to static complexity ?

- There is little evidence that complexity measures such as decision counts are of any use at all in predicting defects

Defects

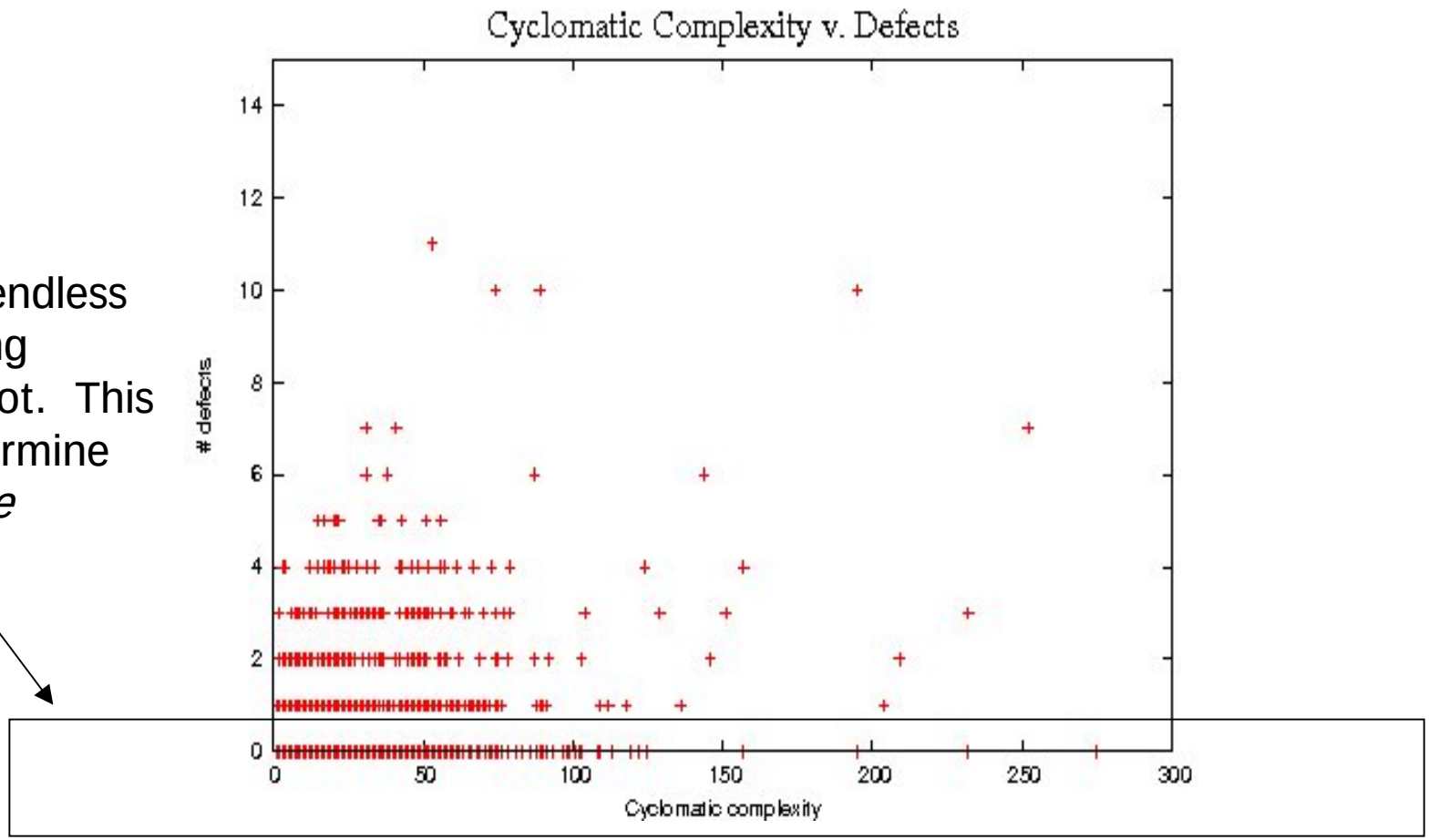


Decision counts

NAG Fortran library over 25 years (Hopkins and Hatton (2008))

Is there anything unusual about ‘zero’ defect ?

PCA and endless rummaging suggest not. This may undermine *root-cause analysis*.



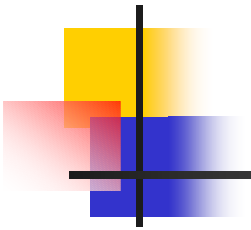
… programming language and defect

- On static defects
  - Modern programming languages are littered with many types of statically detectable defect, (for example reliance on evaluation order).
  - These typically occur around 5-10 per 1000 lines of executable code and fail at an unacceptably high rate. They must be removed by tools plus inspections.
- Given the undisciplined growth of programming languages, its hardly surprising…

… programming language and bloat

Language	Size 1	Size 2	Increase factor
Ada	270Kb (1983)	1093Kb (1995)	x4
C	191 pp. (1990)	401 pp. (1999)	x2
C++	808 pp. (1999)	1370 pp (2010 draft)	x1.7
Fortran	134 pp. (1978)	354 pp. (1990)	x2.5

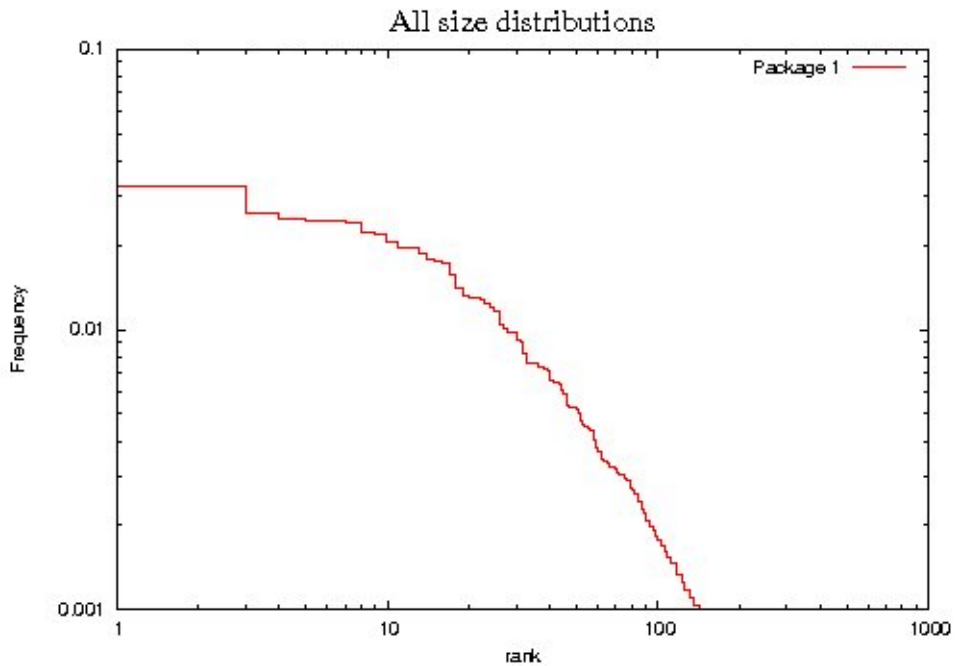
Overview



- A little about software systems
- Personal musings on software defects
- A hidden clockwork
- Conclusions

Software size distributions appear power-law in LOC

In spite of this, systems appear astonishingly similar in their information properties …



Smoothed (cdf) data for 21 systems, C, Tcl/Tk and Fortran, combining 603,559 lines of code distributed across 6,803 components, (Hatton 2009, IEEE TSE)

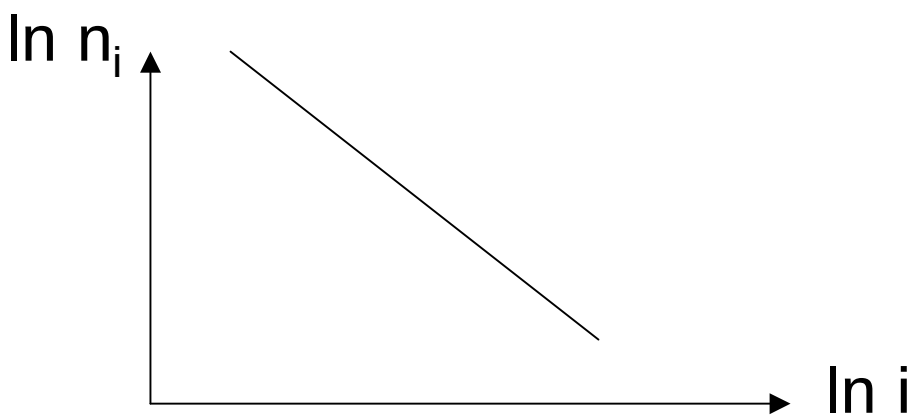
What is power-law behaviour ?

Frequency of occurrence n_i given by $$n_i = \frac{nc}{i^p}$$

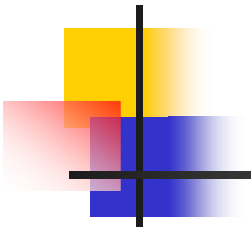
This is usually shown as

$$\ln n_i = \ln(nc) - p \ln i$$

which looks like



Searching for a hidden clockwork: building systems



- When we build a system we are making choices
  - Choices on functionality
  - Choices on architecture
  - Choices on programming language(s)
- There is a general theory of choice – Shannon information theory.
- If we constrain both choice and size in any symbol-based system we are led to …

Which leads to the alphabetic power-law theorem

the general theorem

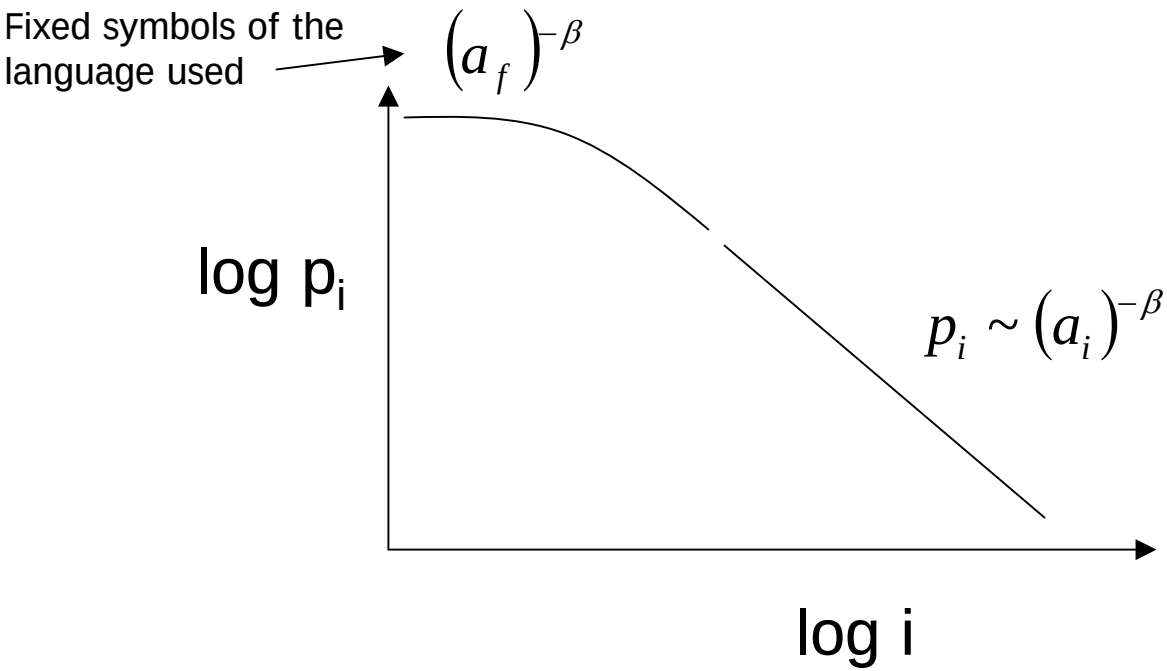
$$p_i \sim \left(a_i\right)^{-\beta}$$

This states that in any software system, conservation of size and information (i.e. choice) is overwhelmingly likely to produce a power-law in a_i which is the list of unique keywords and identifiers used to build the i^{th} component.

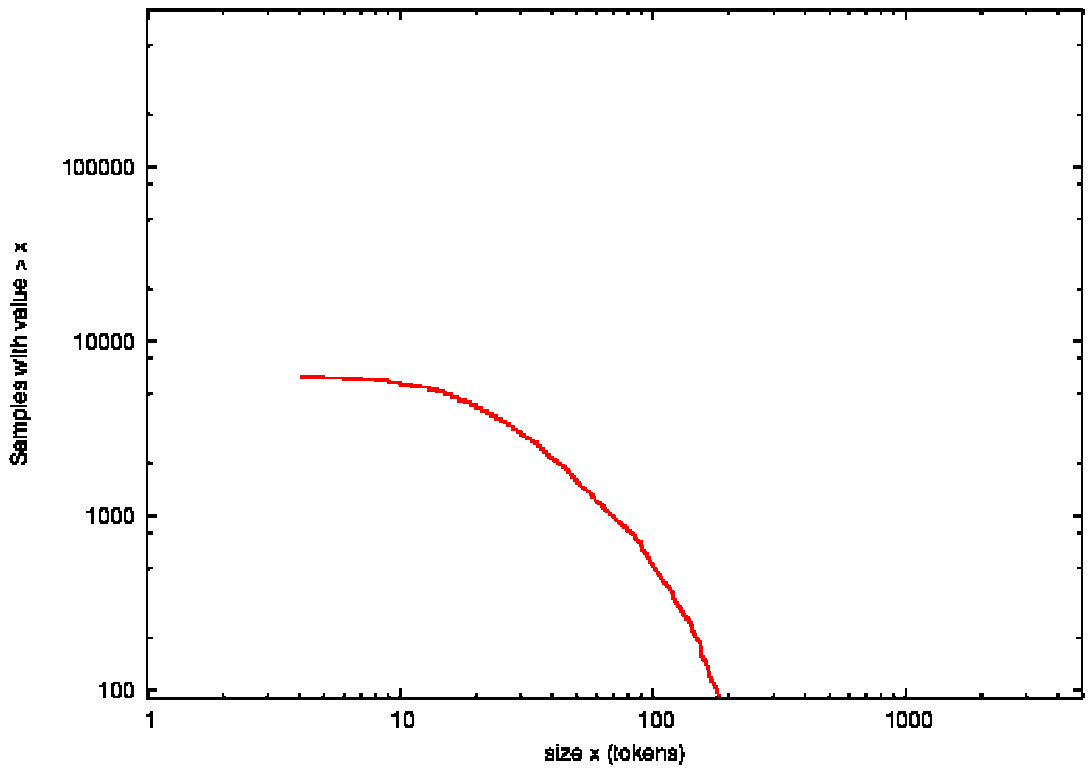
Hatton L (2011) IFIP, Boulder Colorado August 2011.

Application to software systems

So we are looking for the following signature

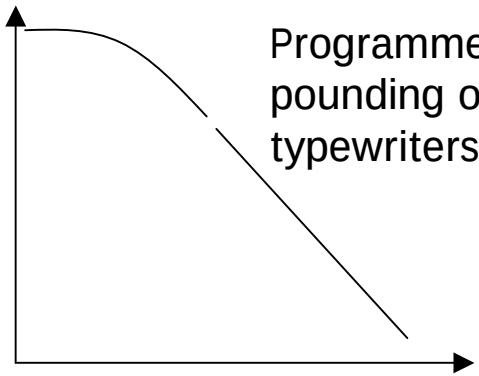


Equilibriation in 400,000 line chunks

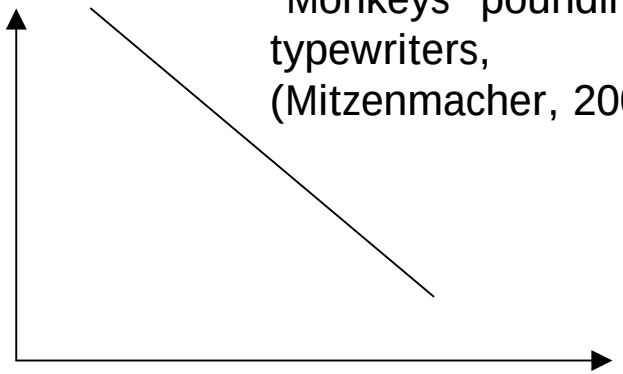


40 million lines of Ada, C, C++, Fortran, Java, Tcl-Tk from 80+ systems

Programming and Monkeys



Programmers pounding on typewriters



"Monkeys" pounding on typewriters, (Mitzenmacher, 2004)

So, what can we say about defects ?

- However, systems evolve such that the total number of defects D is conserved, (since they are finite), (Hatton, (2009) IEEE TSE, 35(4), p. 566-572), giving

$$D = \sum_{i=1}^{M} d_i \Rightarrow p_i = \frac{1}{Q(\gamma)} e^{-\gamma \frac{d_i}{t_i}}$$

- Combining this with

$$p_i \sim (a_i)^{-\beta}$$

The tloga theorem

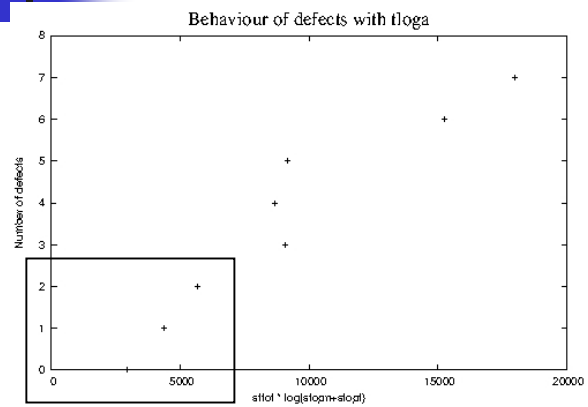
- Predicts the following defect distribution

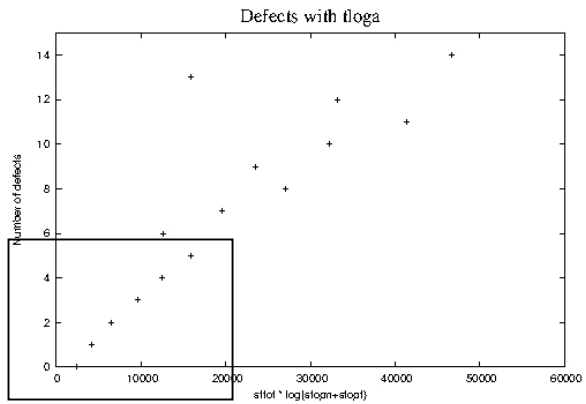
$$d_i \approx t_i \log a_i$$

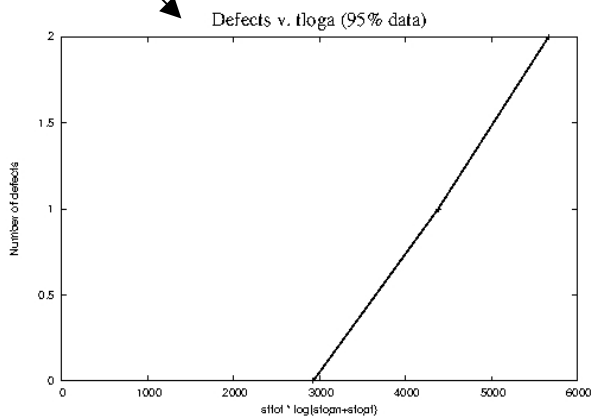
Here t_i is the total number of keywords and identifiers and d_i is the number of defects.

- Can we test this ?

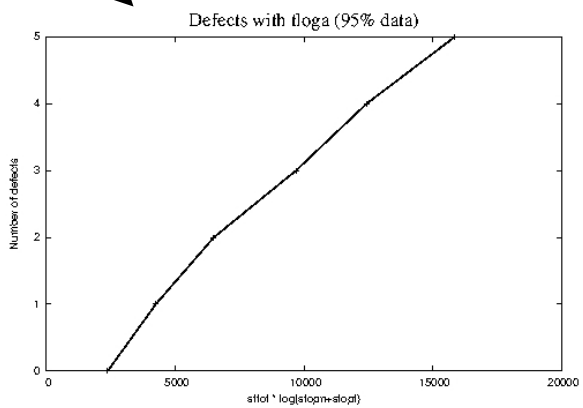
The tloga theorem







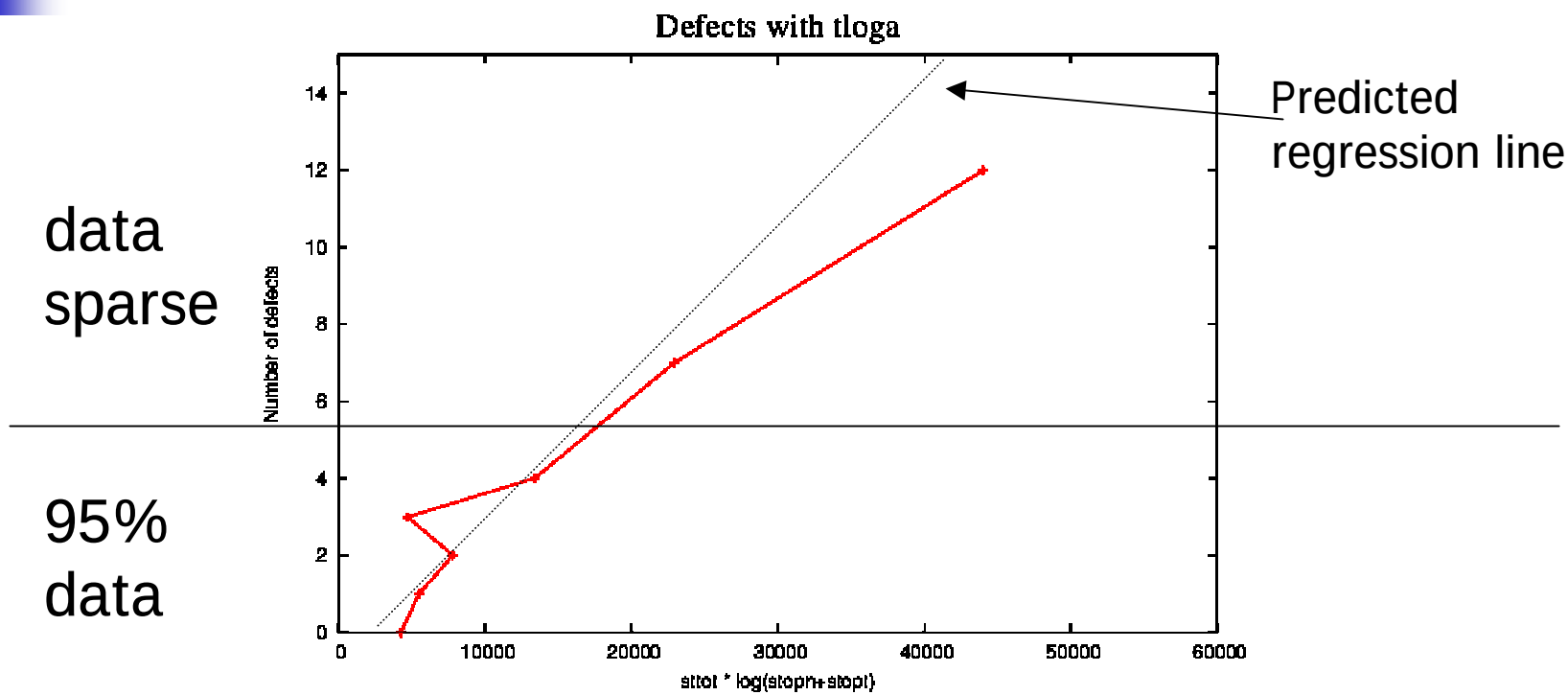
95%



NAG Fortran Library

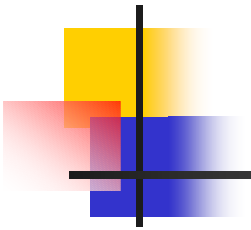
Eclipse IDE (Java)

Equilibriation to tloga 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.

The tloga theorem



- This should give us a handle on how well software has been used simply from the tloga linearity of its current defect distribution.
- It also explains the observed phenomenon of *defect clustering*

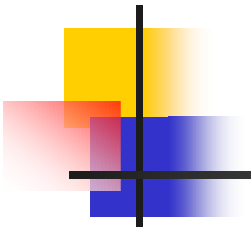
Defect clustering in the NAG Fortran library (over 25 years)

Defects	components	XLOC
0	2865	179947
1	530	47669
2	129	14963
3	82	13220
4	31	5084
5	10	1195
6	4	1153
7	3	1025
> 7	5	1867

80%

Zero-defect seems to be like winning the lottery. There is no systematic way of achieving it.

Overview



- A little about software systems
- Personal musings on software defects
- A hidden clockwork
- Conclusions

Conclusions

- The theorems appear to hold to high levels of significance $p_i \sim (a_i)^{-\beta}$ $d_i \sim t_i \log a_i$
- The implication is that software defects are fundamentally statistical. This we can exploit.
- The further implication is that the engineering concepts of redundancy and resilience will remain of top priority in any system involving software. We probably will never be able to guarantee the absence of defect.
- We need to understand the ***effects*** of software defect far better than we do at the moment but the size of current systems makes this increasingly difficult.

References

My writing site:-

http://www.leshatton.org/

Specifically,

http://www.leshatton.org/variations_2010.html

For comments on reproducibility in software systems,

http://www.nature.com/nature/journal/v482/n7386/full/nature10836.html

Thanks for your attention.