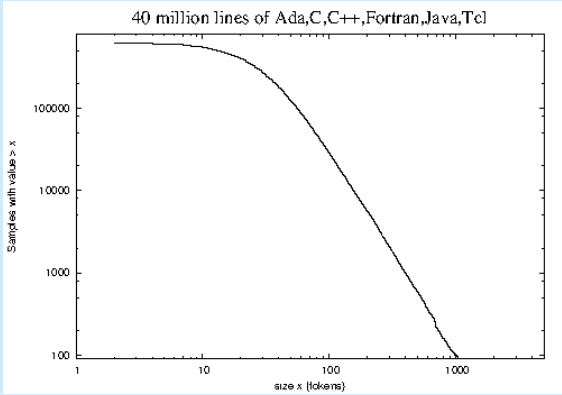
“Software’s Hidden Clockwork: Power-laws, defects and the conservation of information in token-based systems "

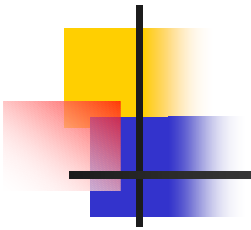
Les Hatton

www.leshatton.org

Version 1.1: 30/Oct/2012



Overview

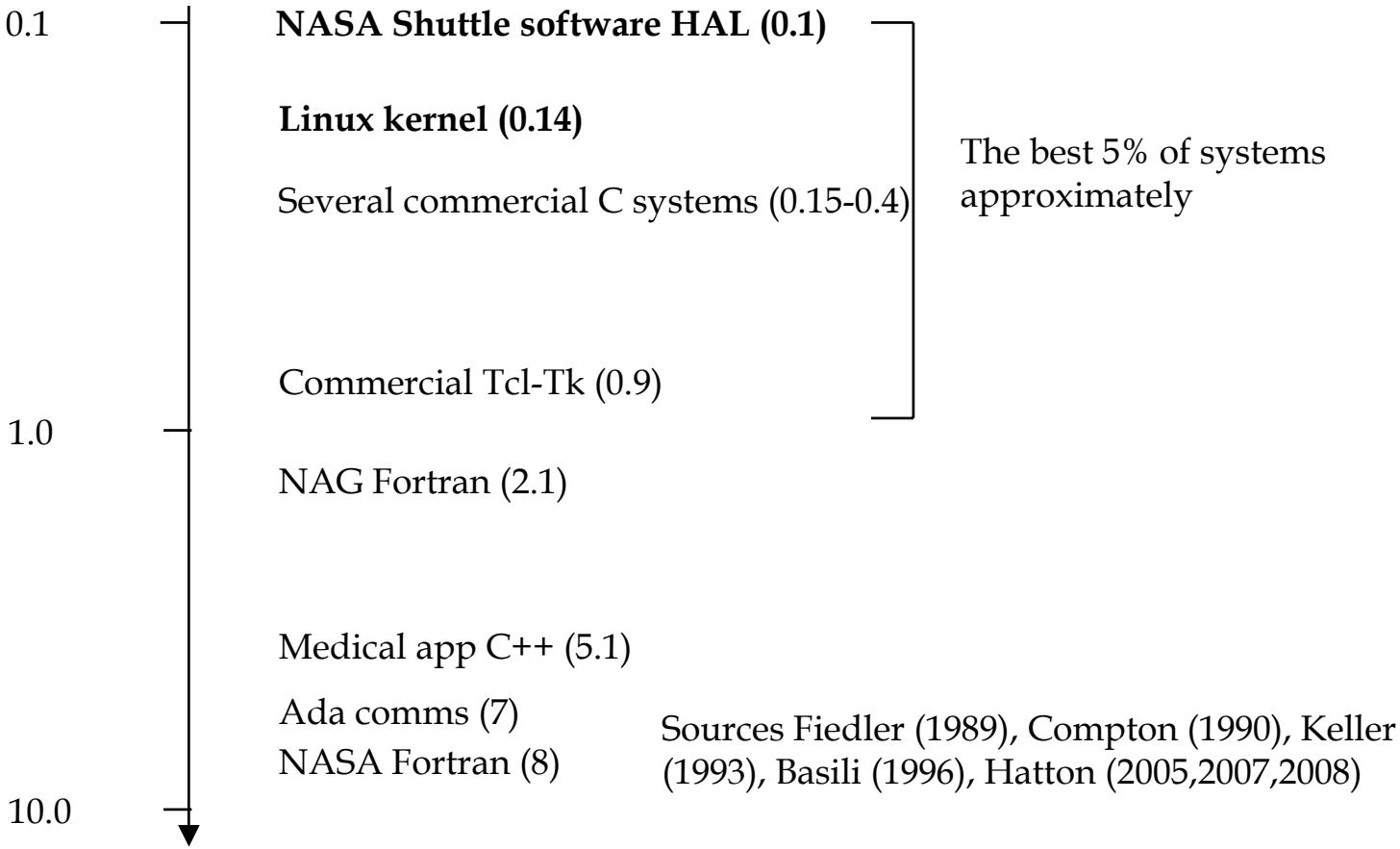


- A little about software systems
- A hidden clockwork
- Analogues with the genome
- Defects
- Conclusions

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.

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.

They occur in interesting places …

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 ...

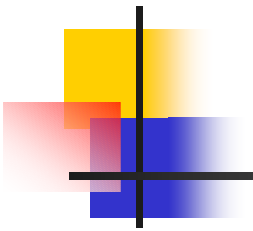
27-Feb-2012: The launch of the new London Routemaster bus, (cost GBP 1.4 million each) was delayed by a week after a software problem meant it had to run with the rear platform shut. Perhaps they should have called it a Raptor.

And in banks …

- 22-25 June 2012
  - *Routine* software update basically stops two of

    Britain's biggest banks (NatWest and RBS) processing payments.
    e tens of millions of pounds.

Some early thoughts



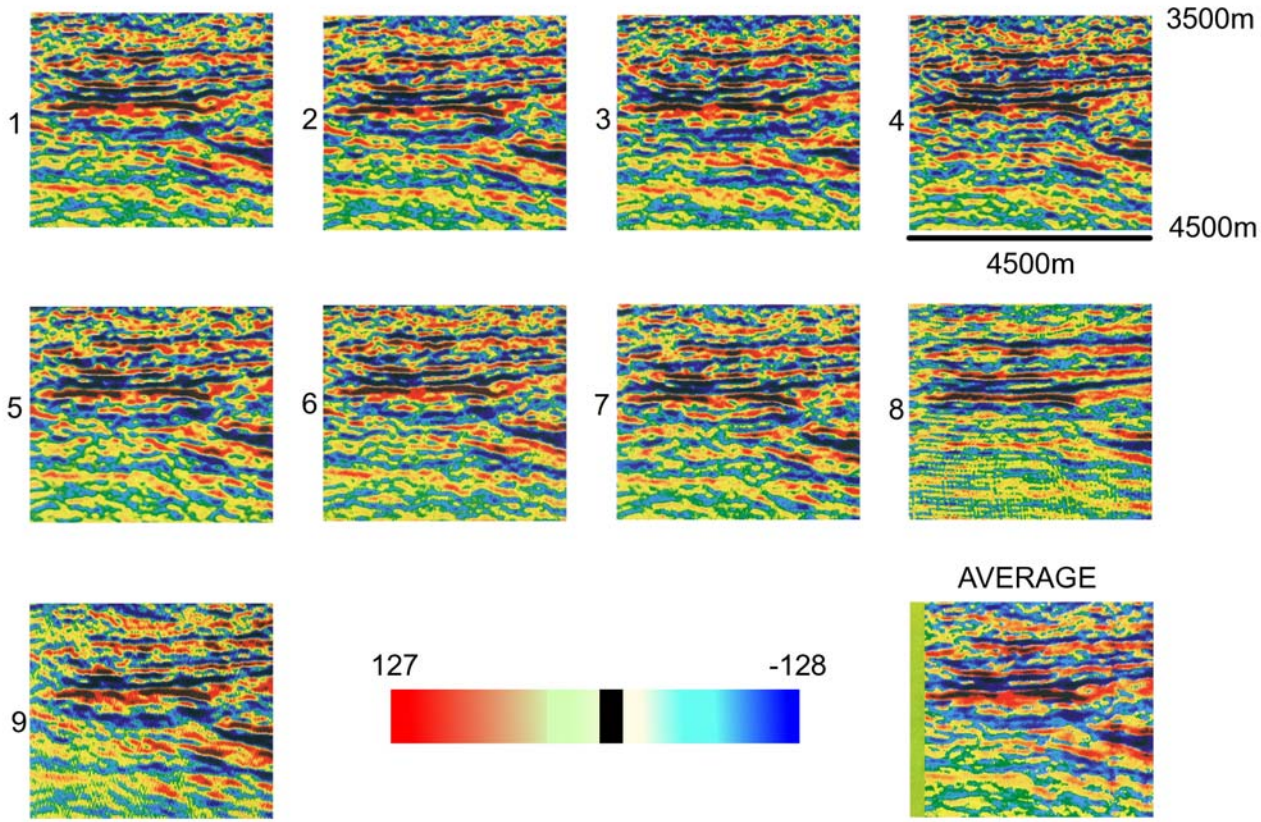
By 2011 I was reasonably convinced that:

- *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 (2007-11).

How big must a defect be before we notice it ?

- 11 October 2012
  - *French Woman stunned after phone company sends her EUR 11.8 quadrillion phone bill*
  - *This comfortably beats the previous record of …*
- 10 April 2006
  - *Malaysian man gets USD 218 trillion phone bill*
- Reassuringly, in both cases, the victim had difficulty convincing the phone company they were wrong.

How small must a defect be before we miss it ?

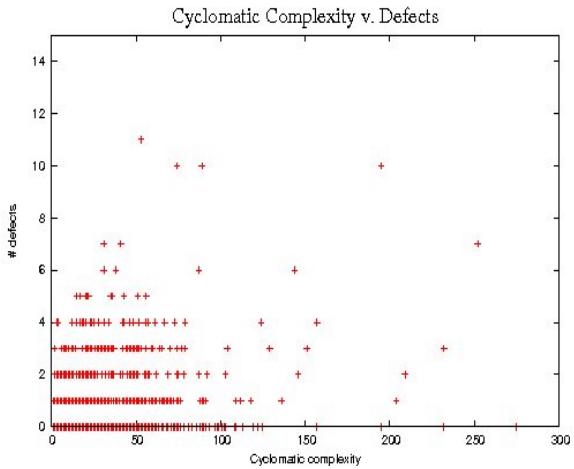


Comparison of 9 commercial packages using the same algorithms on the same data in the same programming language, (Hatton and Roberts (1994))

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

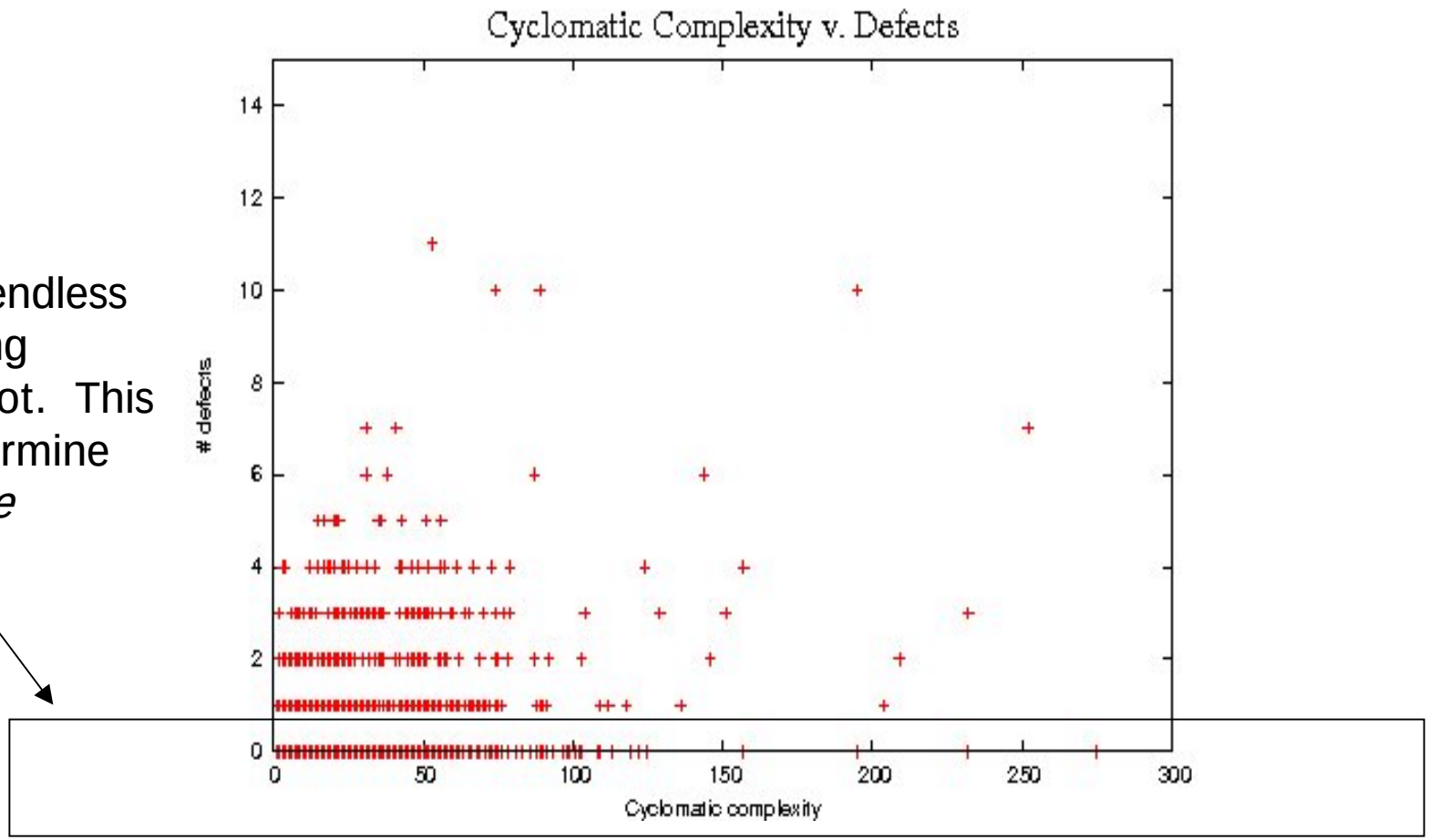


Decision counts

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*.



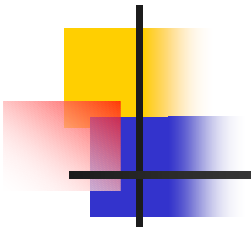
But they cluster …

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%

Unfortunately, zero-defect seems to be like winning the lottery. There is no systematic way of achieving it. (NAG Fortran library over 25 years)

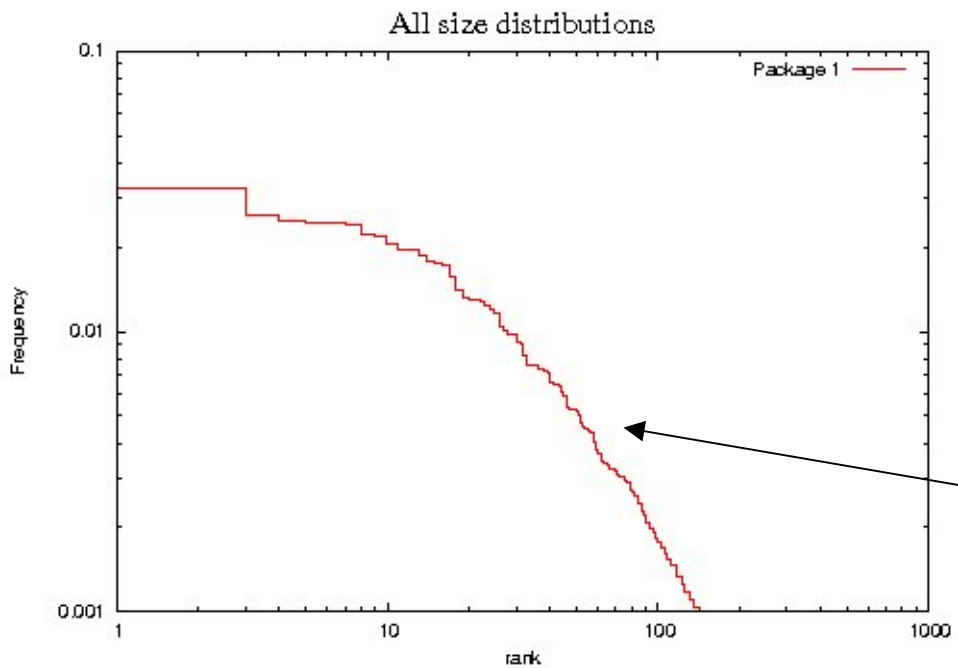
Overview



- A little about software systems
- A hidden clockwork
- Analogues with the genome
- Defects
- Conclusions

Software size distributions appear power-law in LOC

Systems appear astonishingly similar in their component size distributions …



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)

Power-law is linear on log-log plot

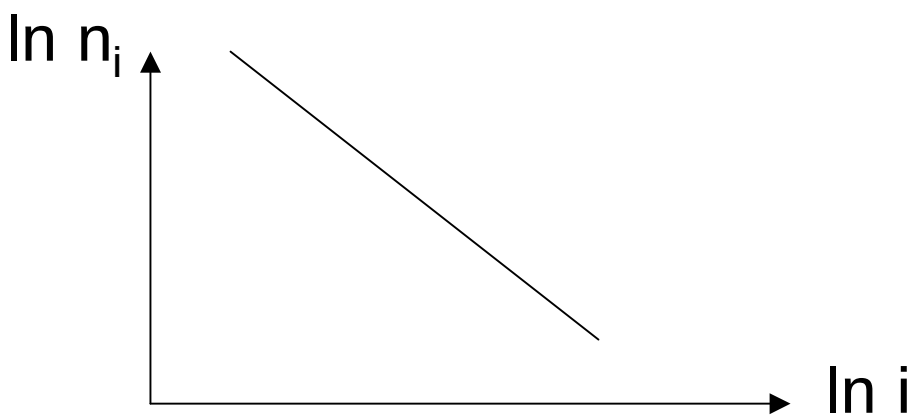
What is power-law behaviour ?

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

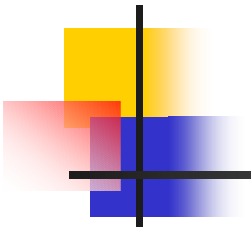
This is usually shown as

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

which looks like

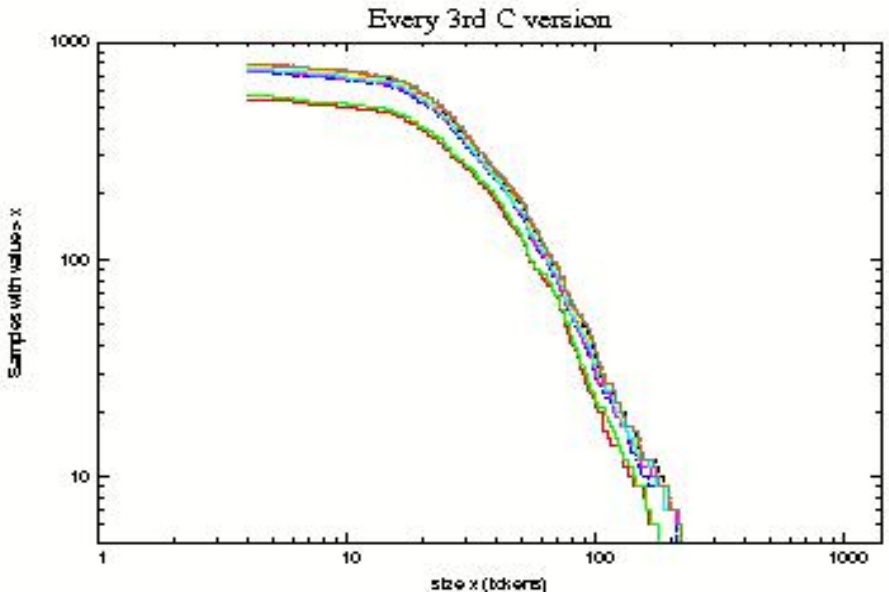


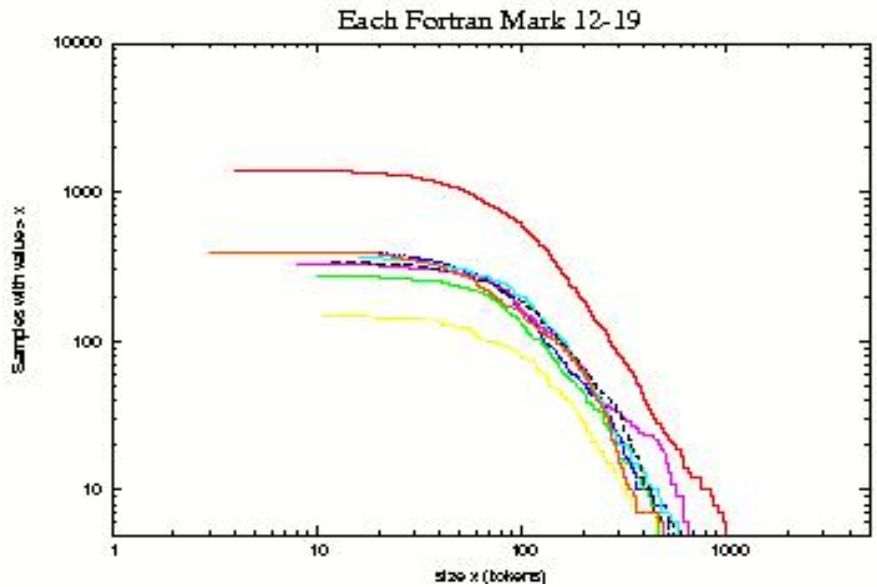
Is power-law behaviour persistent ?



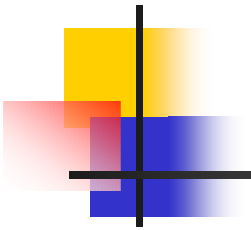
- Question: Does power-law behaviour in component size establish itself over time as a software system matures or is it present at the beginning ?

Is power-law behaviour persistent ?





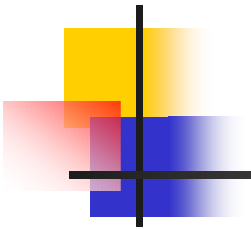
Is power-law behaviour persistent ?



- Answer: *Power-law behaviour in component size appears to be present at the beginning of the software life-cycle.*

- Given that this appears independent of programming language and application area, can we explain why ?

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.

A model for emergent power-law size behaviour using Shannon information

Suppose component i in a software system has t_i tokens in all constructed from an alphabet of a_i unique tokens.

First we note that

$$a_i = a_f + a_v(i)$$

Fixed tokens of a language, **{ } [ ] ; while ...**

Variable tokens, (id names and constants)

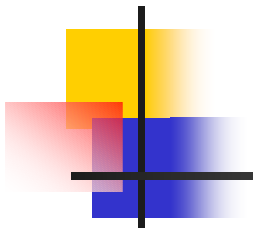
A model for emergent power-law size behaviour using Shannon information

For an alphabet a_i the Hartley-Shannon information content density I'_i per token of component i is defined by

$$t_i I'_i \equiv I_i = \log(a_i a_i \ldots a_i) = \log(a_i^{t_i}) = t_i \log(a_i)$$

We think of I'_i as fixed by the nature of the algorithm we are implementing.

How do we build components from tiny pieces (tokens) ?



Take an example from C:

Fixed (18)

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

+

Variable (8)

bubble a N i j t 1 2

```
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;
      }
    }
  }
}
```

Total (94)

How do we build systems from components ?

So, consider a general software system of T tokens divided into M pieces each with t_i tokens, *in which size and choice (Hartley-Shannon information I) is conserved.*

1	2	3	….			
			t_i, I'_i			
					…	M

$$T = \sum_{i=1}^{M} t_i$$

$$I = \sum_{i=1}^{M} t_i I'_i$$

General mathematical treatment

The most likely distribution of the I'_i (= I_i/t_i)subject to the constraints of T and I held constant

$$T = \sum_{i=1}^{M} t_i \qquad \text{and} \qquad I = \sum_{i=1}^{M} t_i I'_i$$

is

$$p_i \equiv \frac{t_i}{T} = \frac{e^{-\beta I'_i}}{\sum_{i=1}^{M} e^{-\beta I'_i}}$$

where p_i can be considered *the probability of appearance of a component i with a share I_i of I*. β is a constant.

General mathematical treatment and the clockwork theorem

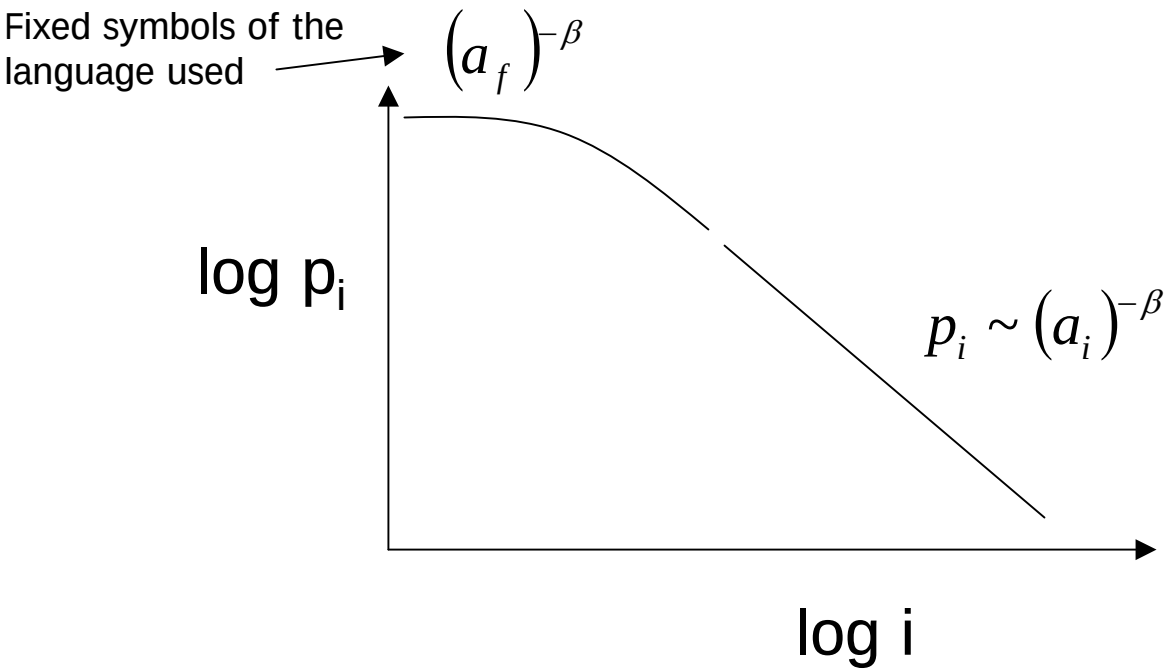
However $$I'_i = \left(\frac{I_i}{t_i} \right) = \left(\frac{t_i}{t_i} \log(a_i) \right) = \log(a_i)$$

Giving the general theorem $$p_i \sim \left(a_i \right)^{-\beta}$$

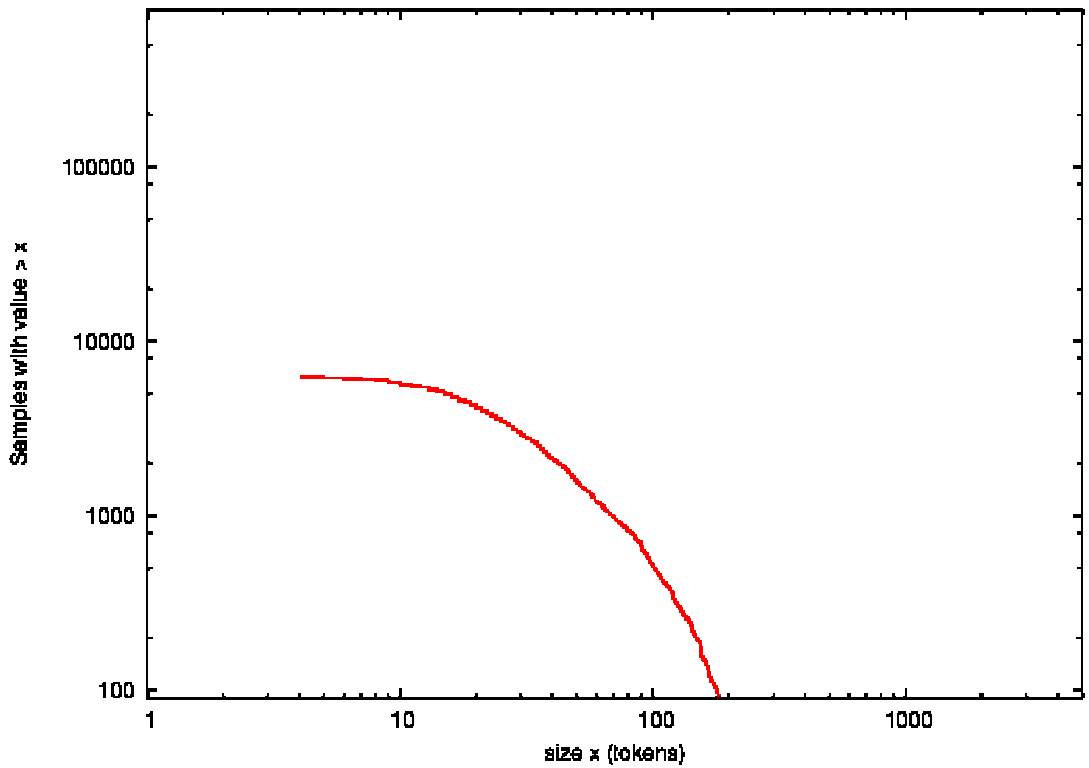
This states that in any software system, conservation of size and information (i.e. choice) is overwhelmingly likely to produce a power-law alphabet distribution. (Think ergodic here).

Application to software systems

Can we test this ? We are looking for the following signature

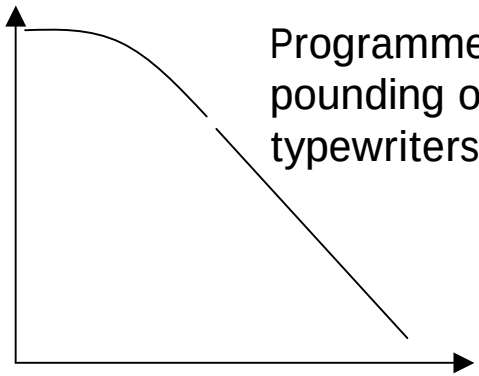


Equilibriation to the clockwork theorem in ~ 500,000 line chunks

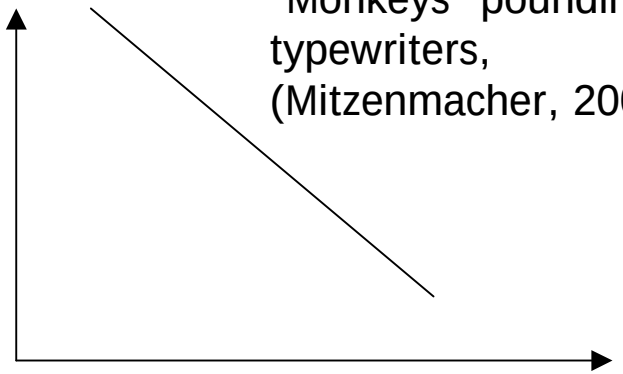


55 million lines of Ada, C, C++,
Fortran, Java, Tcl-Tk from 90+ systems

Programming and Monkeys

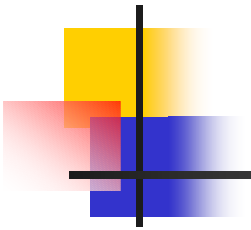


Programmers pounding on typewriters



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

Overview



- A little about software systems
- A hidden clockwork
- Analogues with the genome
- Defects
- Conclusions

Genetic example of alphabetic power-law theorem

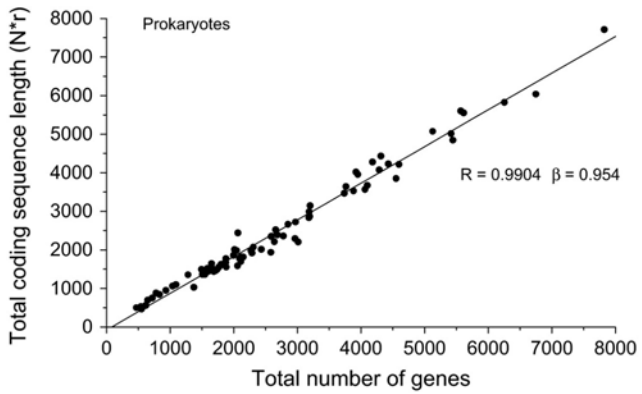
- Has a simple constant alphabet, $a_i = 4$.
- This implies that the random variable L representing gene length is uniformly distributed, giving

$$E(L) = k.\frac{T}{M}$$

T is the total length of the genomic sequence and M the number of genes. k is a constant.

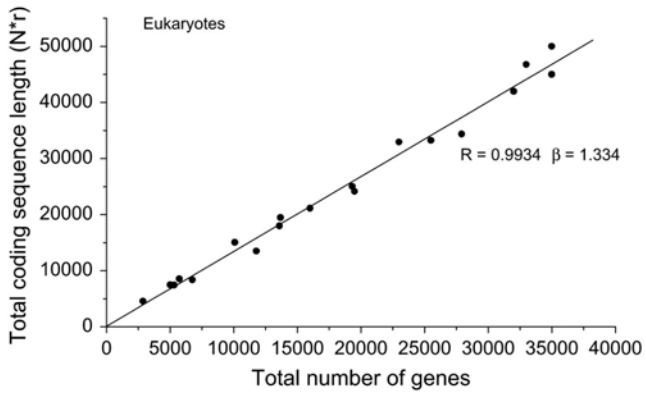
Genetic example of alphabetic power-law theorem

Prokaryotes



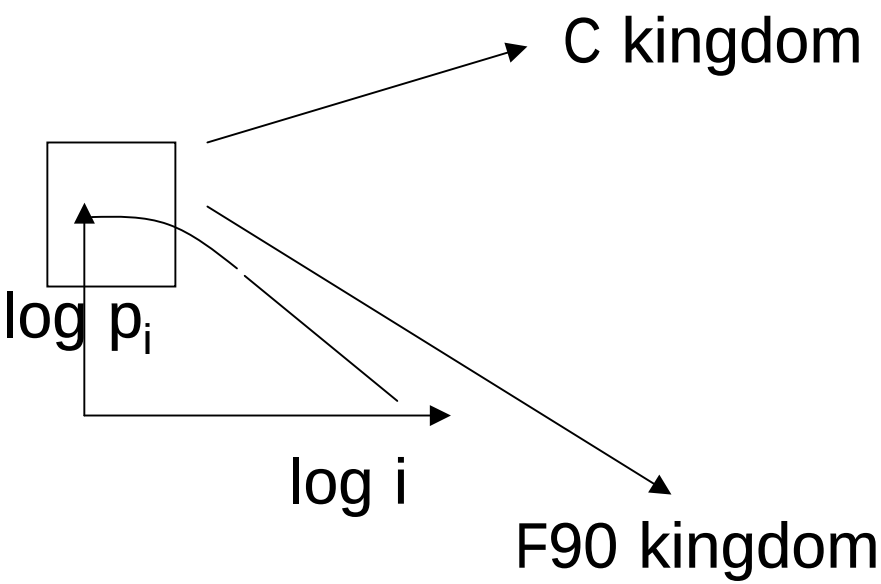
This is indeed observed :-

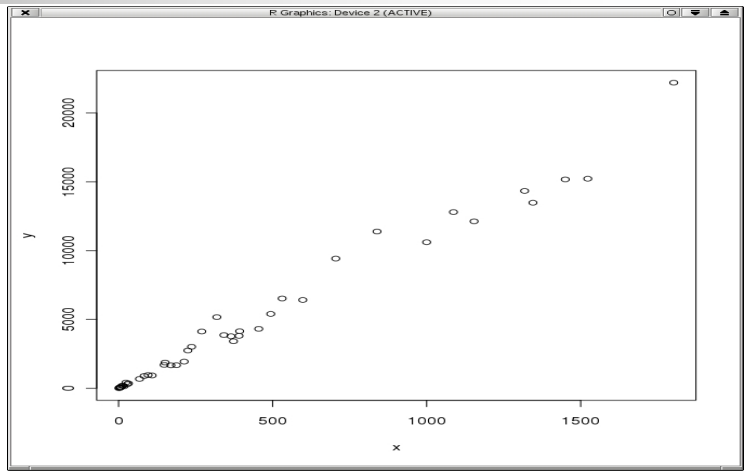
Eukaryotes

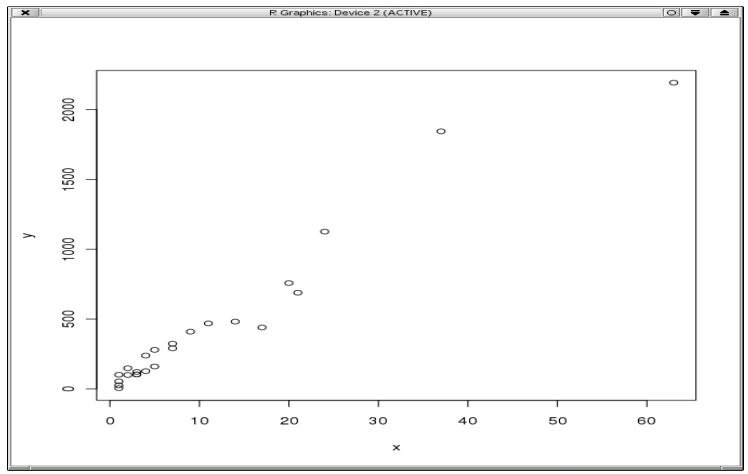


Xu et al. (2006) “Average gene length is highly conserved in Prokaryotes and Eukaryotes and diverges only between the two kingdoms”, Mol Biol Evol (June 2006) 23 (6), p. 1107-1108

Are there analogous programming kingdoms ?

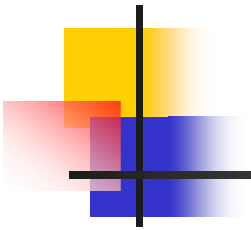






However if we confine ourselves too small software components, the same linearity should hold true

Overview



- A little about software systems
- A hidden clockwork
- Analogues with the genome
- Defects
- Conclusions

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 the clockwork theorem

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

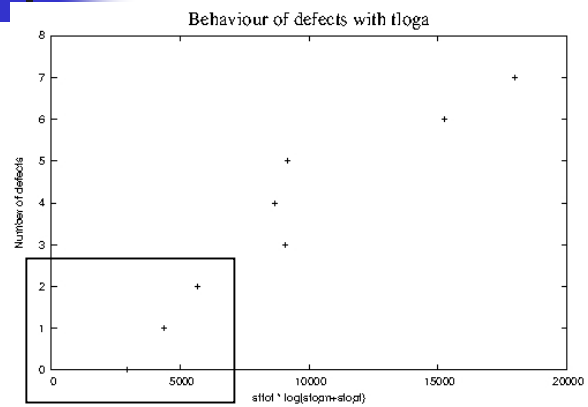
The tloga theorem

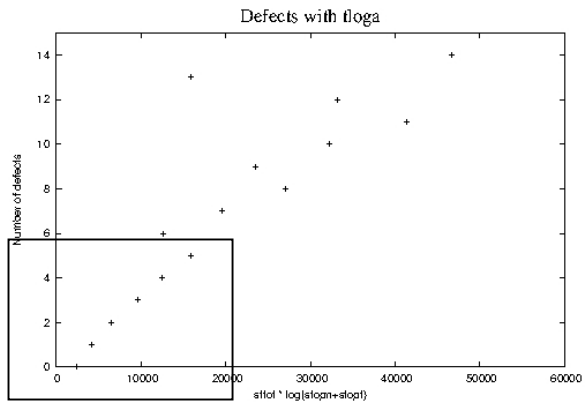
- Predicts the following asymptotic defect distribution

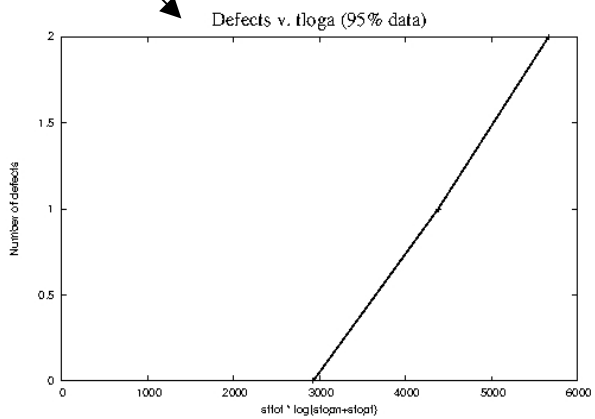
$$d_i \approx t_i \log a_i$$

Can we test this ?

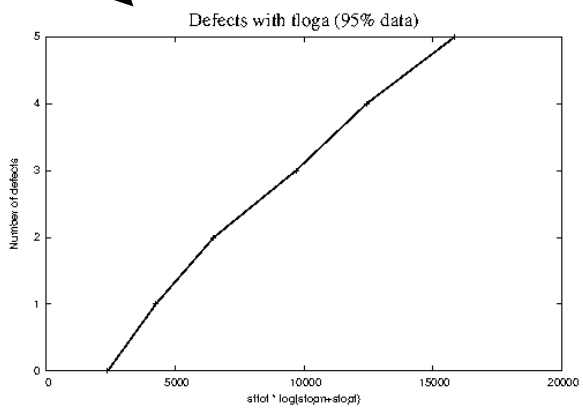
The tloga theorem







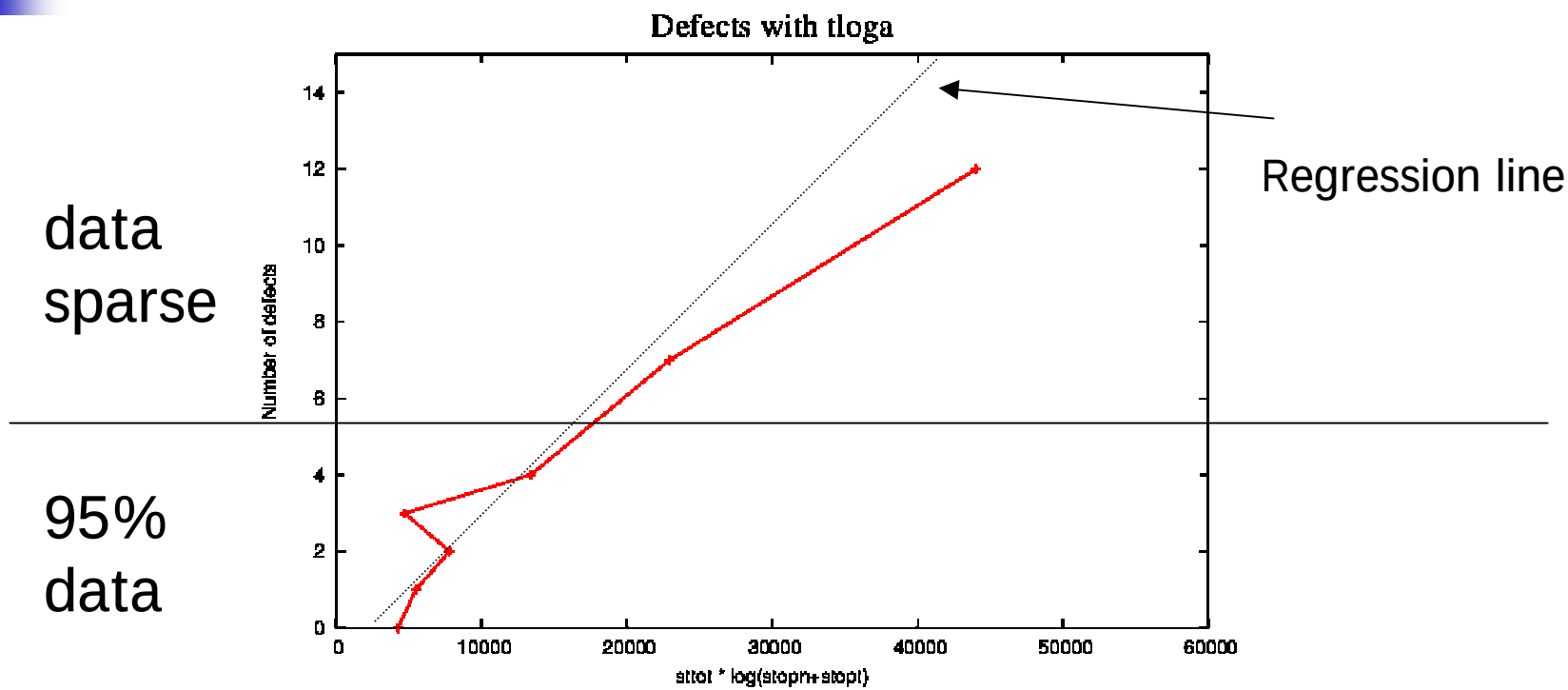
95%



NAG Fortran Library

Eclipse IDE (Java)

Equilibriation to tloga in the Eclipse IDE*

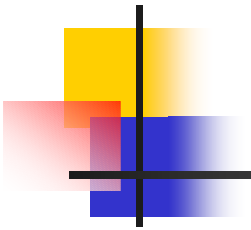


*With grateful thanks to Andreas Zeller et. al. (2007) for extracting the defect data and making it openly available. http://www.st.cs.uni-sb.de/softevo. The data comes from releases 2.0,2.1 and 3.0. There are 10,613 components in the release 3.0.

The tloga theorem

- *This should give us a handle on how well software has been tested* simply from the tloga linearity of its current defect distribution.
- In other words, we can tell the difference between good software and bad testing.
- (It also predicts the observed phenomenon of *defect clustering)*

Overview



- A little about software systems
- A hidden clockwork
- Analogues with the genome
- Defects
- Conclusions

Conclusions

$$p_i \sim (a_i)^{-\beta} \qquad d_i \sim t_i \log a_i$$

- Software component distributions appear to evolve the same way however we build them and whatever they do guided by the unseen hand of information conservation. (Equivalent to energy conservation in physical systems.)
- The genome also appears to conserve information leading to constant average gene length in kingdoms
- The smallest components of software systems are distributed similarly to genes
- Conservation of Information leads to defects asymptoting to tloga allowing the possibility of identifying poor testing

References

My writing site:-

http://www.leshatton.org/

Specifically,

http://arxiv.org/abs/1207.5027

http://arxiv.org/1209.1652

For comments on reproducibility in computational science,

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

Thanks for your attention.