

“Unifying power-law behaviour, functionality and defect distribution in general software systems”

Les Hatton

CISM, Kingston University
L.Hatton@kingston.ac.uk

Version 1.1: 29/Apr/2009

Overview



- What is scale-free behaviour ?
- The story so far
- Is scale-law behaviour persistent in software systems ?
- A tentative unifying principle
- Conclusions

What is scale-free behaviour ?

- In this context, scale-free behaviour refers to a phenomenon whose *frequency of occurrence* is given by a *power-law*.
- Consider word-counting in a document. If n is the total number of words in a document and n_i is the number of occurrences of word i , then ***it is observed*** (originally by Zipf (1949)), that for many texts,

$$f_i = \frac{c}{i^p} \quad \text{where } c, p \text{ are constants and} \quad f_i \equiv \frac{n_i}{n}$$

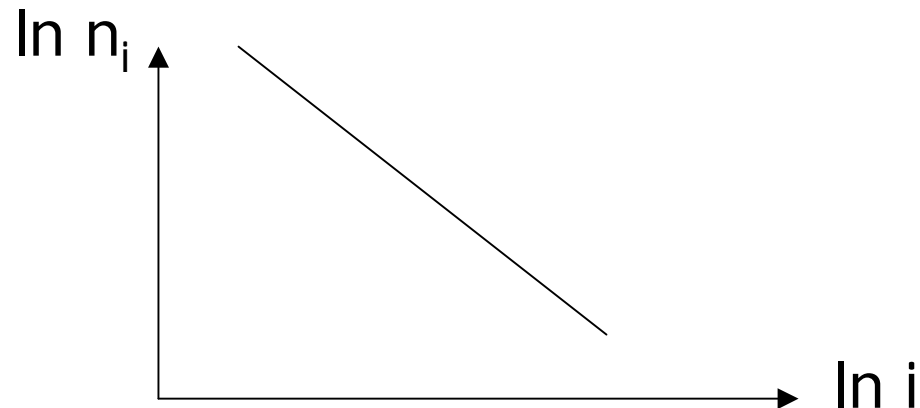
What is scale-free behaviour ?

Re-writing as $n_i = \frac{nc}{i^p}$

This is usually shown as

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

which looks like



Examples from the real world



- Physics:- specific heat of spin glasses at low temperature, Caudron et al (1981)
- Biology: Protein family and fold occurrence in genomes, Qian et al. (2001)
- Biology: Evolutionary models, Fenser et al (2005)
- Economics: Income distributions, Rawlings et al (2004)
- Software systems: incoming and outgoing references and class sizes in OO systems, Potanin et al (2002)
- Fractals also exhibit scale-free behaviour (Miro):-
 - <http://cism.kingston.ac.uk/people/details.php?AuthorID=577>
- Studies of C systems also reveal scale-free behaviour (Derek)
 - <http://www.knosof.co.uk/cbook/cbook.html>

Overview



- What is scale-free behaviour ?
- The story so far
- Is scale-law behaviour persistent in software systems ?
- A tentative unifying principle
- Conclusions

General mathematical treatment

Consider a general system of N atomic objects divided into M pieces each with n_i objects, each piece having an *externally imposed* property e_i associated with it.

1	2	3			
			n_r, e_r			
				...		M

$$N = \sum_{i=1}^M n_i$$

General mathematical treatment

The number of ways of organising this is:- $W = \frac{N!}{n_1!n_2!\dots n_M!}$

Stirling's approximation + logs as usual gives:-

$$\ln W = N \ln N - \sum_{i=1}^M n_i \ln n_i$$

In physical systems, we seek to find the most likely arrangement by maximising this subject to two constraints

$$N = \sum_{i=1}^M n_i$$

and

$$U = \sum_{i=1}^M n_i e_i$$

Assume fixed externally so not varied

General mathematical treatment



Using Lagrange multipliers and setting $\delta(\ln W) = 0$

leads to the most likely distribution being given by

$$p_i \equiv \frac{n_i}{N} = \frac{e^{-\beta e_i}}{\sum_{i=1}^M e^{-\beta e_i}}$$

where p_i can be considered *the probability of piece i occurring with a share e_i of U* . β is a constant.

General mathematical treatment

To summarise, for large N and n_i

The most likely distribution of the e_i subject to the constraints

$$N = \sum_{i=1}^M n_i \quad \text{and} \quad U = \sum_{i=1}^M n_i e_i$$

is

$$p_i \equiv \frac{n_i}{N} = \frac{e^{-\beta e_i}}{\sum_{i=1}^M e^{-\beta e_i}}$$

but if e_i is logarithmic with n_i ,
the result is a ***power-law***

$$p_i \approx \frac{C}{n_i^\beta}$$

Application to software systems



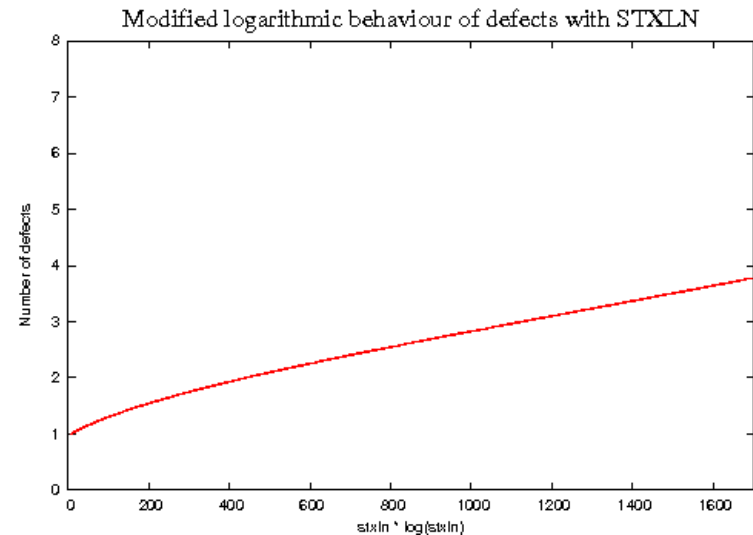
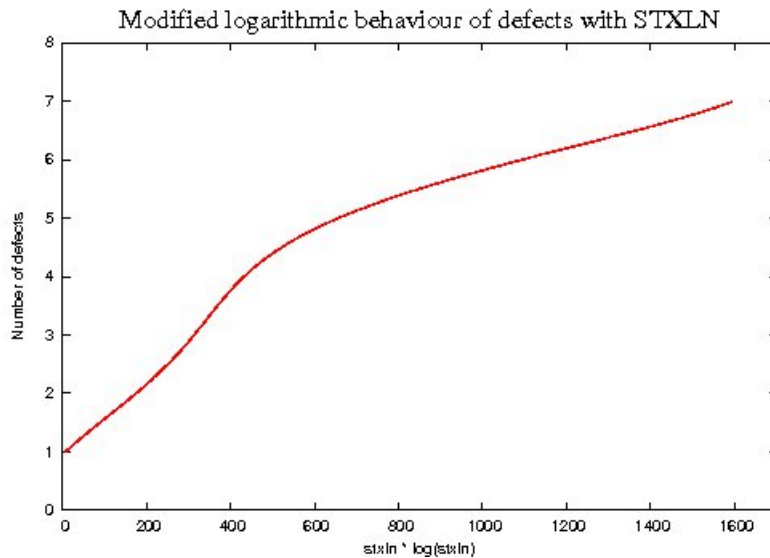
If we identify e_i with the defect density in a component (*therefore assuming that defect density is externally enforced by some mechanism*), then the *total number of defects in a software system* is given by:-

$$U = \sum_{i=1}^M n_i e_i$$

Application to software systems

However, it has frequently been observed that the defect density e_i in a component behaves as:-

$$e_i \approx \ln n_i$$



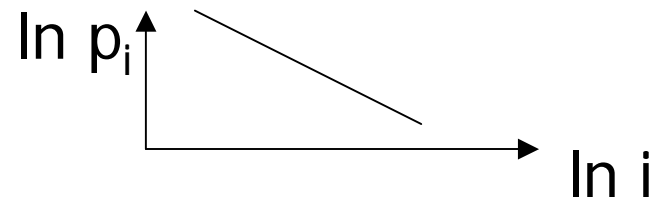
Examples from NAG library, Hatton and Hopkins (2008). See also Lipow (1982)

Application to software systems

The implication is that if we design a software system of a certain size, and *the total number of defects is fixed for some reason*, then the most likely system to emerge will have component sizes obeying a power-law distribution.

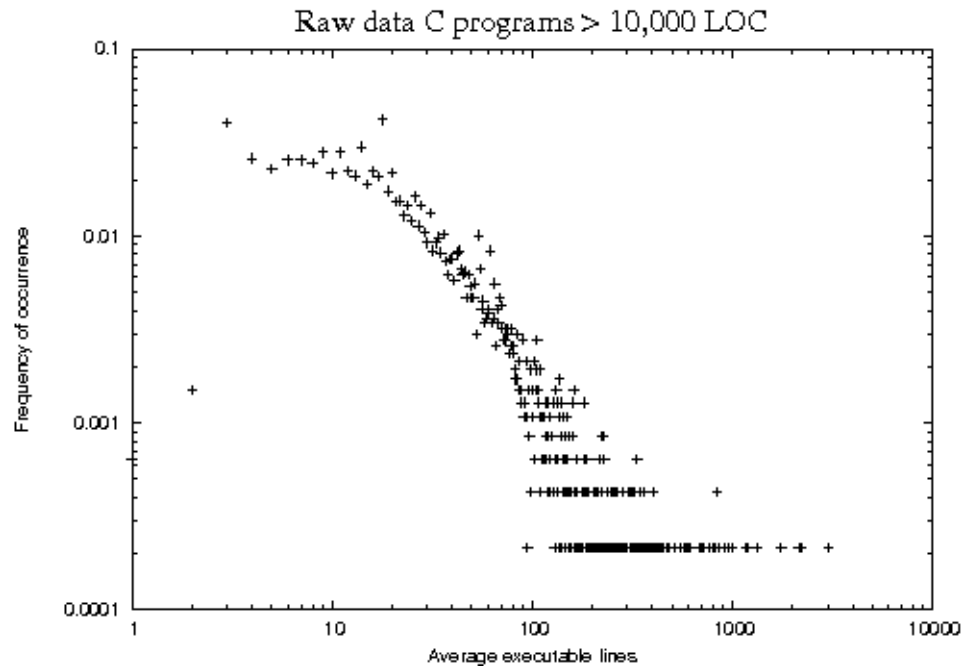
But do we see this ? The following is a study of component sizes in 21 software systems in 3 different languages, Fortran, C and Tcl.

We are looking for



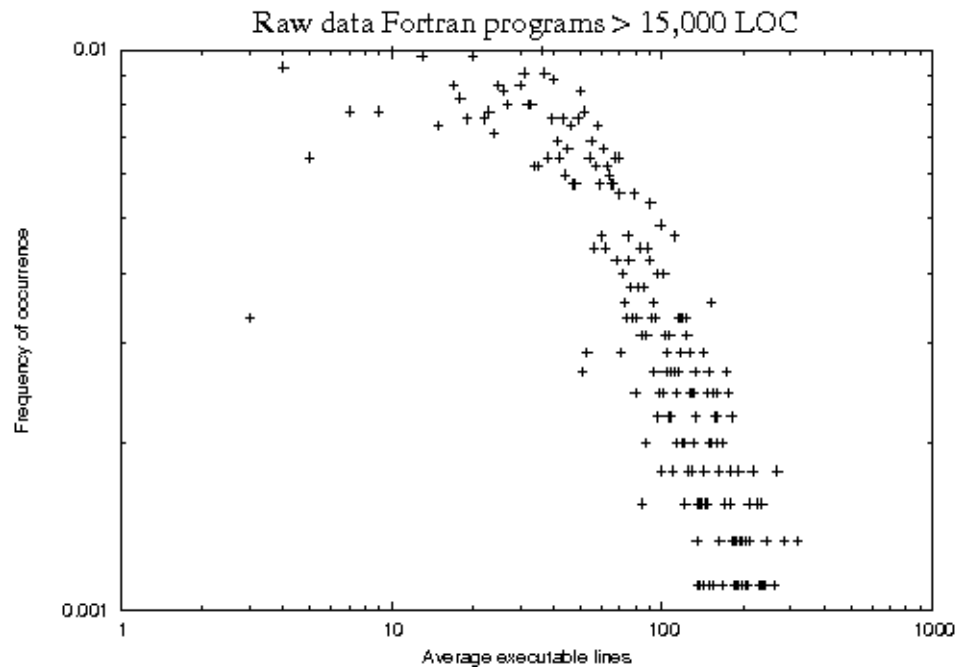
Application to software systems

Raw data for 9 systems in C > 10000 LOC



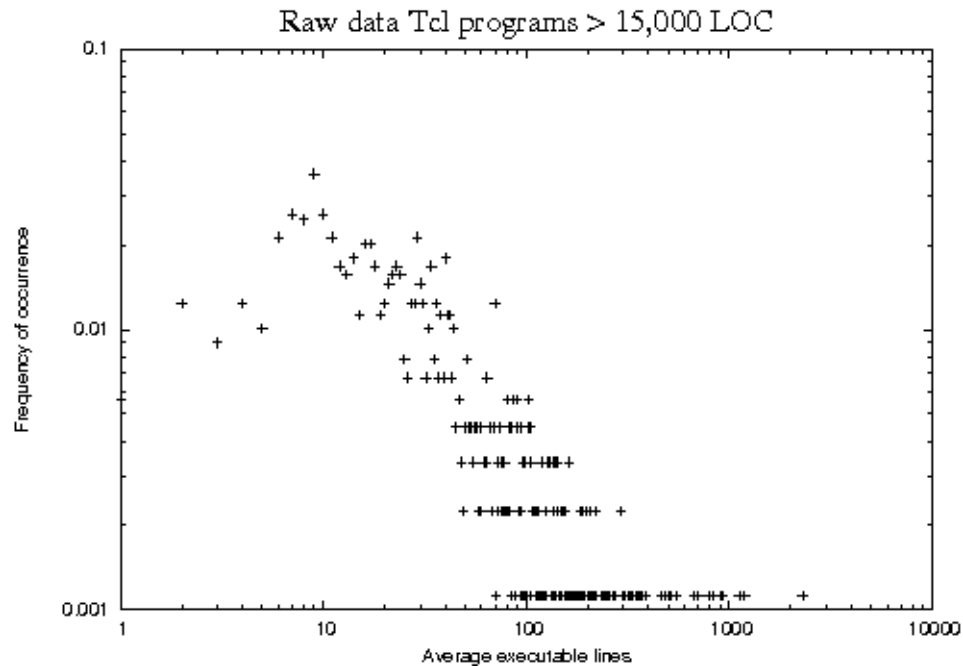
Application to software systems

Raw data for 10 Fortran systems > 15,000 – 250,000 LOC



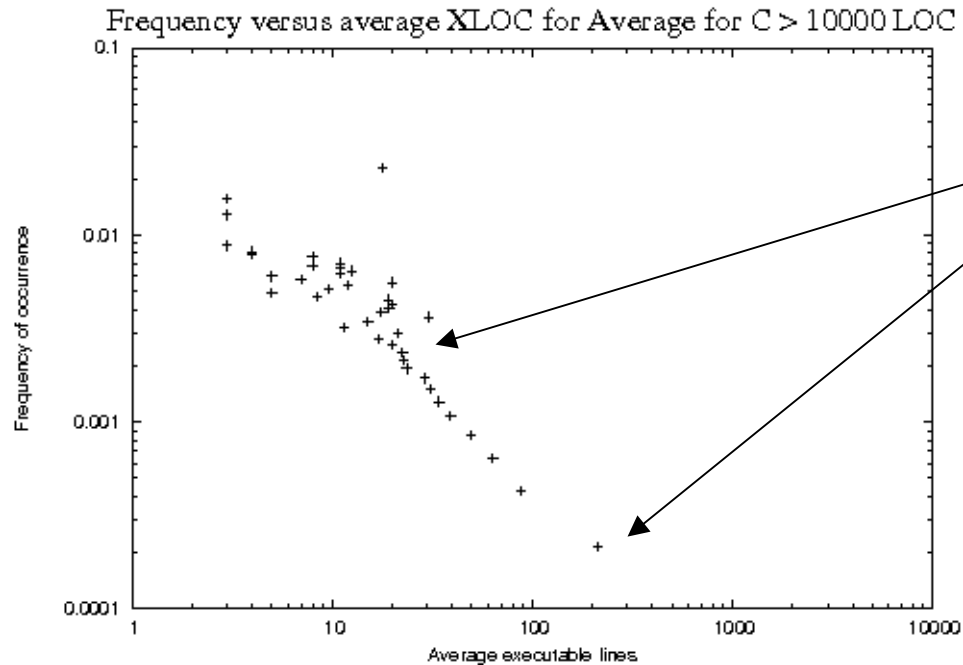
Application to software systems

Raw data for 2 Tcl/Tk systems



Application to software systems

Smoothed data for 21 systems, C, Tcl/Tk and Fortran,
2,000 – 250,000 LOC



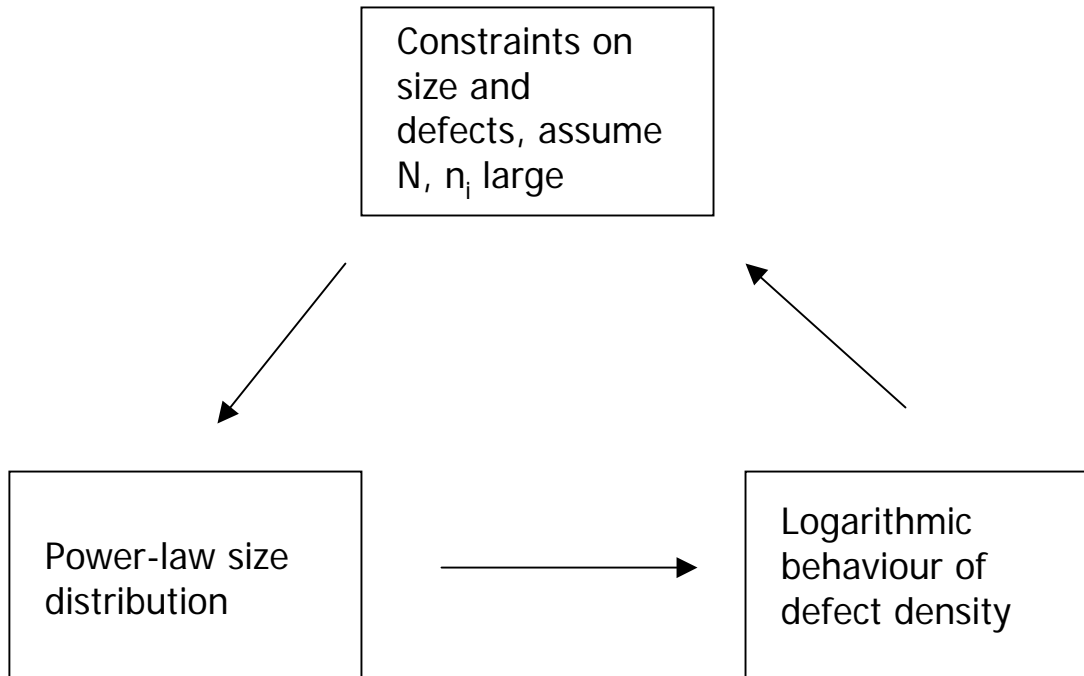
Averaging shows
clear linear
behaviour where
 N, n_i are large

Conclusion from early work



There is strong evidence for the existence of power-law behaviour for component sizes across a range of languages, system sizes and application areas for components larger than about 10 lines.

Summary of early work



Any two imply the third but which is the driver, or do they evolve simultaneously ?

Overview

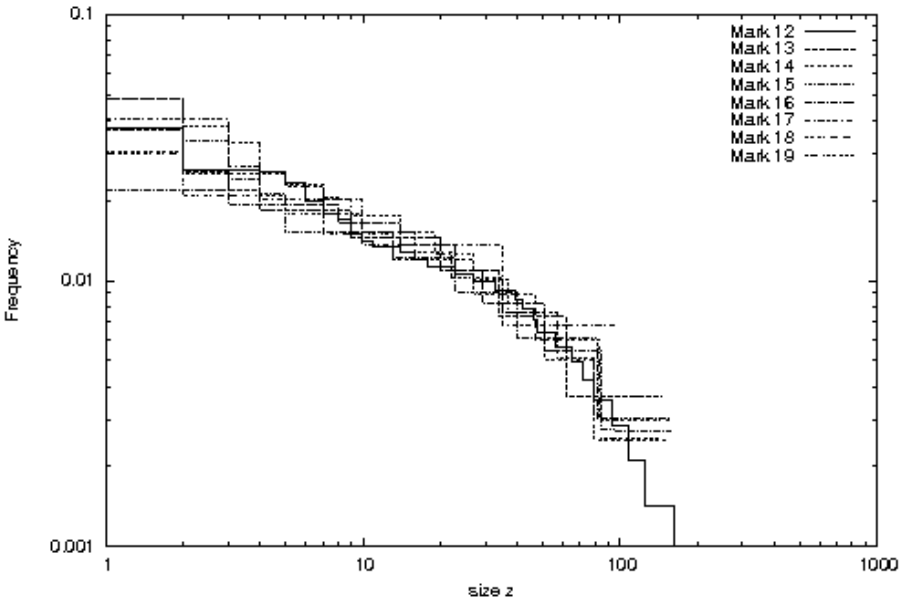


- What is scale-free behaviour ?
- The story so far
- Is scale-law behaviour persistent in software systems ?
- A tentative unifying principle
- Conclusions

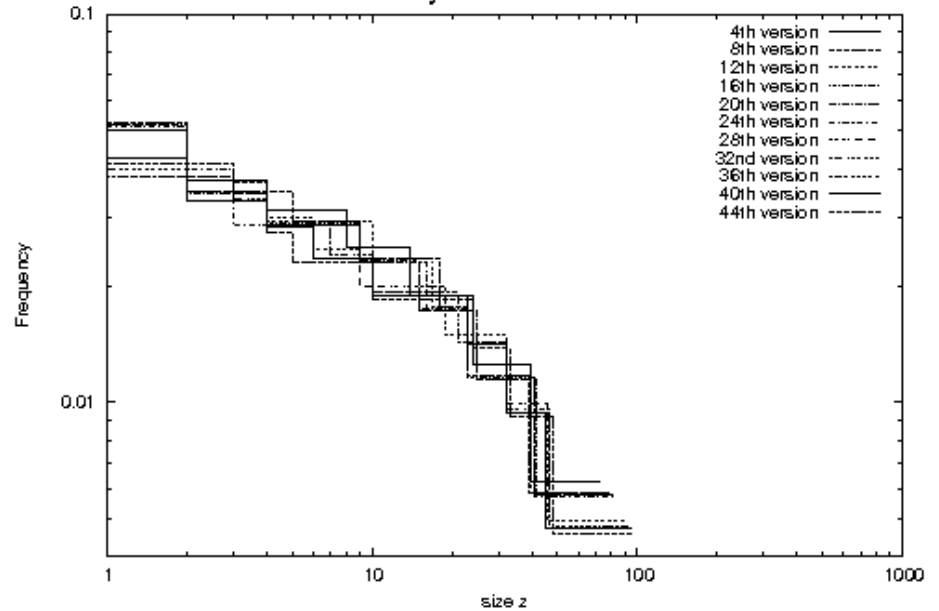
Size distributions in major systems as function of time

7 versions of the NAG Fortran library over 10 years

Each Fortran Mark 12-19

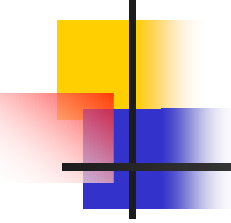


Every 4th Tcl version



41 versions of a TCL commercial application over 6 years from birth to present

Conclusion from this work



There is good evidence for the existence of *a priori* power-law behaviour for component sizes in two languages, system sizes and application areas.

This implies that defects evolve in a way which is governed at system design time. In other words *logarithmic defect density behaviour is inevitable as a system ages*.

So why might power-law behaviour in component size present from the beginning ?

Overview



- What is scale-free behaviour ?
- The story so far
- Is scale-law behaviour persistent in software systems ?
- A tentative unifying principle
- Conclusions

Mechanisms leading to power-law behaviour, Newman (2006).



- Inverse of quantities
- Random walks
- Yule process
- Phase transitions
- Self-organised criticality
- Squinting closely at log-normal distributions.
- Combinations of exponentials, e.g. Miller, Mandelbrot, Shannon – the meaning of symbols

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



- Measuring functionality
 - Lines of code – difficulties with definition.
 - Function points – generally speaking, no better than lines of code, (c.f. Barbara Kitchenham's work)
- *Is there anyway in which functionality might be defined such that power-law size behaviour emerges naturally ?*

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



If there are on average k symbols for each of n_i lines of code, then for an alphabet $S(n_i)$, the total number of combinations is $S^{kn_i}(n_i)$ and the Hartley-Shannon information content $I(n_i) = \log S^{kn_i}(n_i) = k n_i \log S(n_i)$

The normal criticism of this use of information content is that Hartley-Shannon *only relates to the symbols and not their meaning*.

To circumvent this, *we define the symbols to be the names of the variables used*.

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

- *Define the functionality of a component f_i to be the Hartley-Shannon information content of the alphabet S , of non-redundant user-defined variable names used in that component. Similarly let f'_i be the functionality density.*

$$n_i f'_i \equiv f_i = \log S^{kn_i}(n_i) = kn_i \log S(n_i)$$

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

If we now define the total functionality of a system to be U then:-

$$U = \sum_{i=1}^M n_i f'_i$$

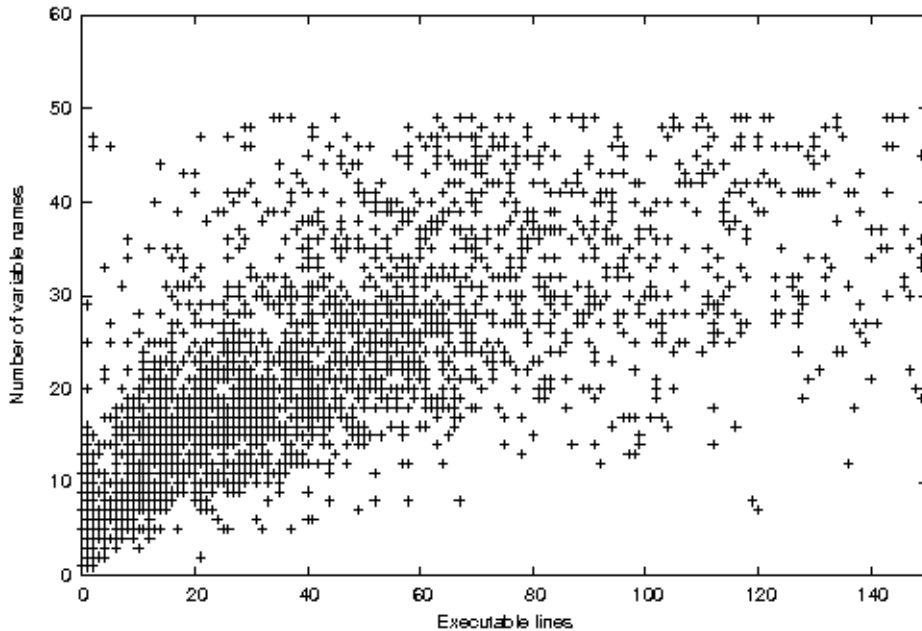
Repeating the statistical mechanical argument for component size ...

$$p_i \equiv \frac{n_i}{N} = \frac{e^{-\beta f'_i}}{\sum_{i=1}^M e^{-\beta f'_i}} = \frac{e^{-\mu \log S(n_i)}}{Q(\beta)}$$

If $S(n_i)$ turns out to be linear in n_i we are getting close.

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

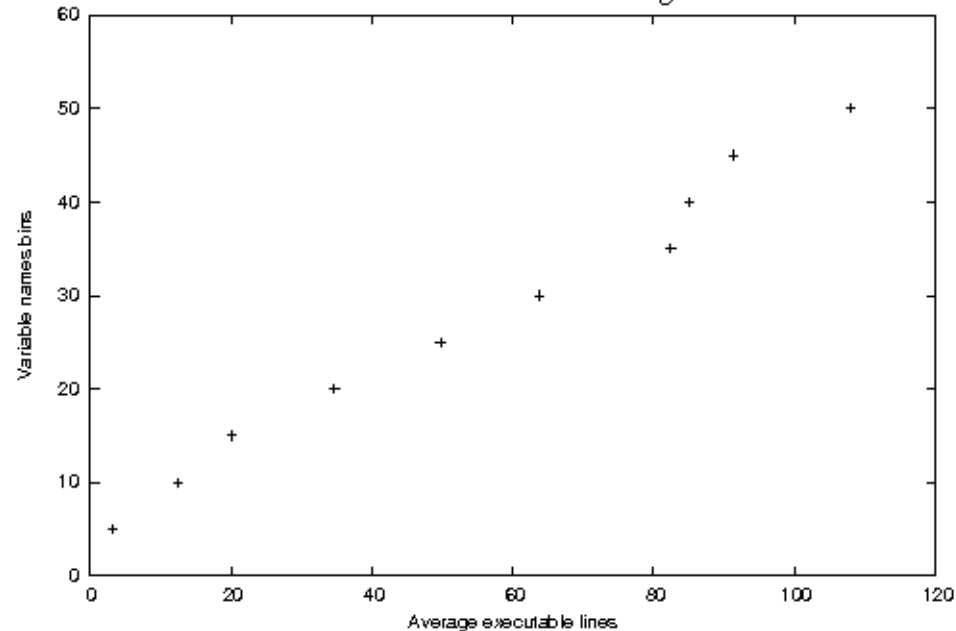
Number of variable names versus XLOC



Number of user-defined names v.
component size in lines – binned to
reduce noise

Number of user-defined names v.
component size in lines – raw data.

Variable name bins versus average XLOC



Overview



- What is scale-free behaviour ?
- The story so far
- Is scale-law behaviour persistent in software systems ?
- A tentative unifying principle
- Conclusions

Conclusions of early work



- Component sizes in software systems of very different size and language obey power-law size distributions for components > about 10 lines
- Standard arguments from statistical mechanics show that this is intimately linked to logarithmic behaviour in defect density.

Conclusions so far



- Power-law size behaviour appears to be *a priori*
- Extending the argument using statistical mechanics shows that if we define the information content using an alphabet of user-specified names, this will lead to a power-law size distribution which in turn leads to a logarithmic defect density distribution as the system approaches equilibrium
- Encouraging so far but need to look at more systems.

References



My writing site:-

<http://www.leshatton.org/>