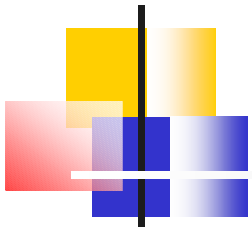
“Power-laws, persistence and the distribution of Information in software systems"

Les Hatton

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

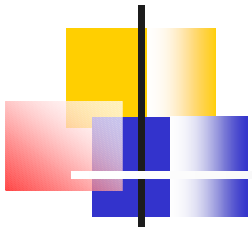
Version 1.1: 01/Mar/2010

Overview



- What is scale-free behaviour and where does it occur ?
- The story so far
- Is scale-law behaviour persistent in software systems ?
- A unifying model
- 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,

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

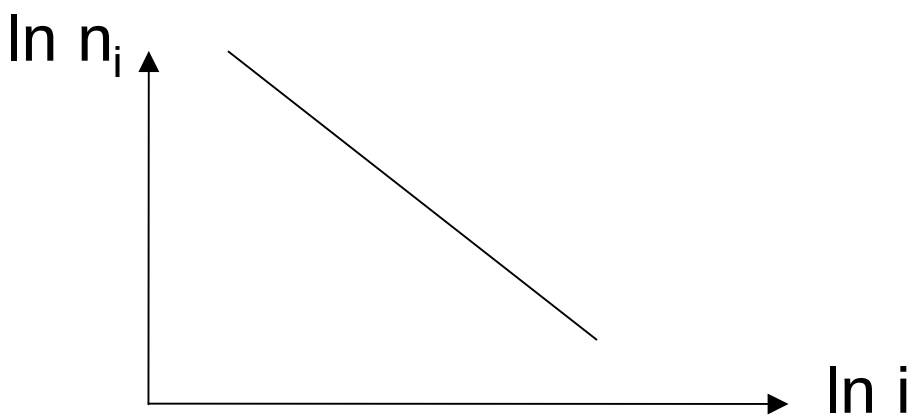
What is scale-free behaviour ?

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

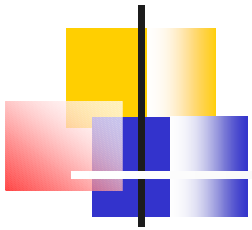
This is usually shown as

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

which looks like

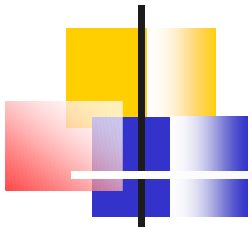


Examples from the real world



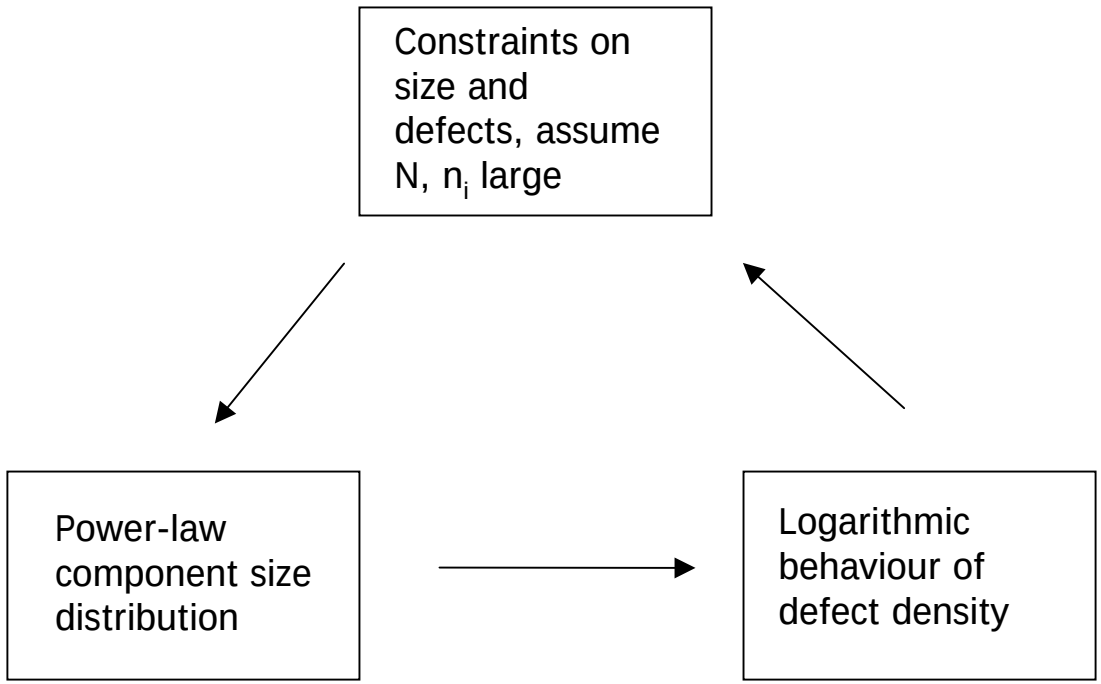
- Physics:- specific heat of spin glasses at low temperature, Caudron et al (1981)
- Biology: Protein family and fold occurrence in genomes, Qian et al. (2001)
- Biology: Evolutionary models, Fenser et al (2005)
- Economics: Income distributions, Rawlings et al (2004)
- Software systems: incoming and 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



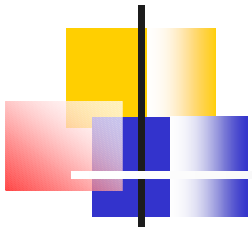
- What is scale-free behaviour and where does it occur ?
- The story so far
- Is scale-law behaviour persistent in software systems ?
- A unifying model
- Conclusions

Summary of early work with software systems



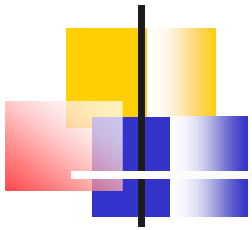
Any two imply the third but which is the driver, or do they evolve simultaneously ?

Where to go from here ?



- Is power-law behaviour persistent ?
- If so, why does it occur ?

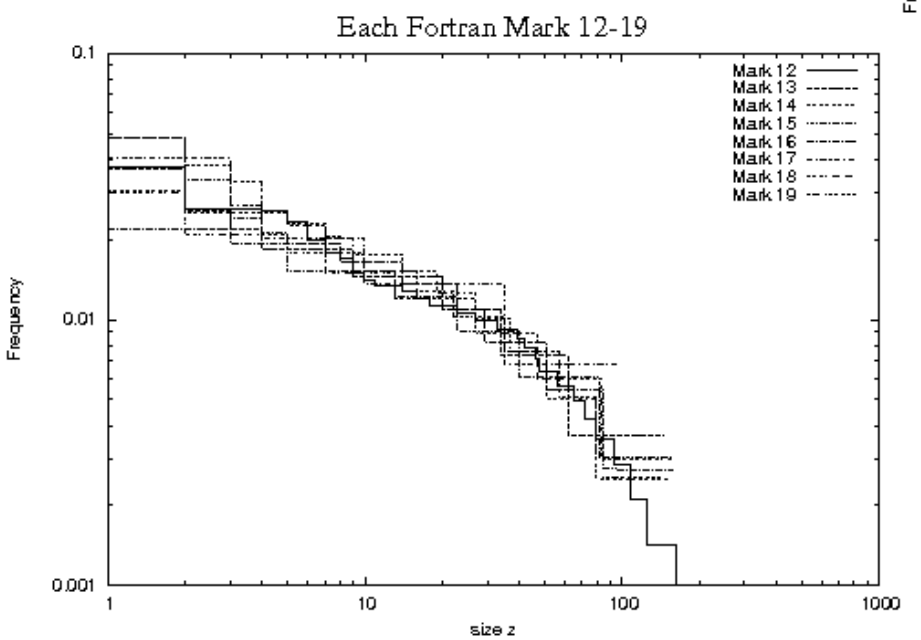
Overview

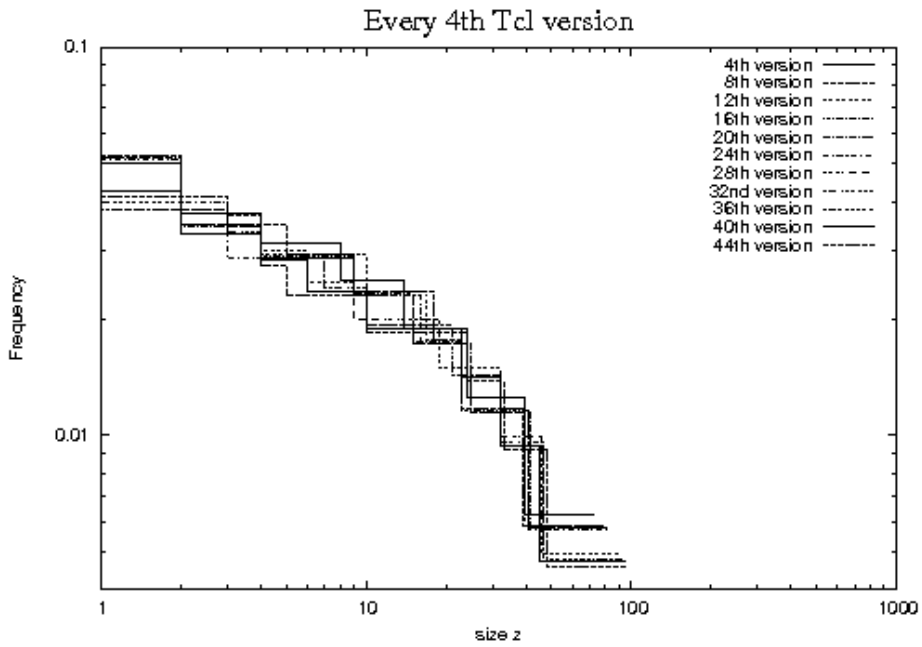


- What is scale-free behaviour and where does it occur ?
- The story so far
- Is scale-law behaviour persistent in software systems ?
- A unifying model
- Conclusions

Size distributions in major systems as function of time

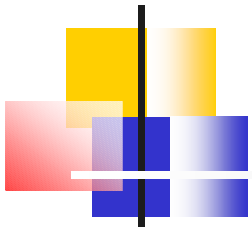
7 versions of the NAG Fortran library over 10 years





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

Overview



- What is scale-free behaviour and where does it occur ?
- The story so far
- Is scale-law behaviour persistent in software systems ?
- A unifying model
- Conclusions

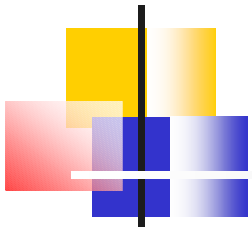
General mathematical treatment

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

1	2	3	….			
			t_i,e_i			
					…	M

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

General mathematical treatment



The number of ways of organising this is:-

$$W = \frac{T!}{t_1!t_2!...t_M!}$$

Stirling's approximation + logs as usual gives:-

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

For any system, we seek to find the most likely arrangement by maximising this subject to two constraints

$$T = \sum_{i=1}^{M} t_i \qquad \text{and} \qquad U = \sum_{i=1}^{M} t_i e_i$$

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

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

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

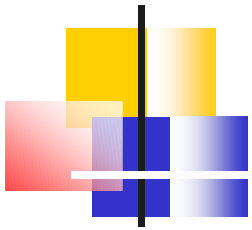
To summarise, for large T and t_i

The most likely distribution of the e_i subject to the constraints

$$T = \sum_{i=1}^{M} t_i \qquad \text{and} \qquad U = \sum_{i=1}^{M} t_i e_i$$

is $$p_i \equiv \frac{t_i}{T} = \frac{e^{-\beta e_i}}{\sum_{i=1}^{M} e^{-\beta e_i}}$$

Application to software systems



If we identify e_i with the Hartley-Shannon information density in a component (*therefore assuming this is externally enforced by some mechanism*), then the *total amount of information in a software system* is given by:-

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

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.

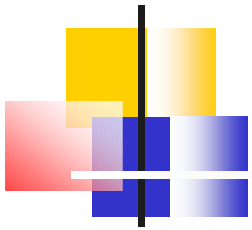
- Combinations of exponentials, e.g. Miller, Mandlebrot, Shannon – the meaning of symbols

A model for emergent power-law size behaviour using H-S information

Note !

- Information here is simply the Hartley-Shannon concept of *selection of signs* in communication theory. It is nothing to do with what we might think of as functionality or any meaning the signs might have.

A model for emergent power-law size behaviour using H-S information



- Programming languages contain two types of token
  - a_f - Fixed tokens, (e.g. goto, {, }, for, (C, C++, …))
  - $a_v(i)$ - User-defined tokens. (Constants, identifier names).

The total alphabet of tokens for component i can then be written as:-

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

A model for emergent power-law size behaviour using H-S information

- *Let I_i be the Hartley-Shannon information content of component i containing t_i tokens constructed from an alphabet a_i.*

$$I_i = \log a_i^{t_i} = t_i \log a_i$$

A model for emergent power-law size behaviour using H-S information

If we now define the total information of a system to be I and define e_i as the information density:-

$$I = \sum_{i=1}^{M} I_i = \sum_{i=1}^{M} t_i \left(\frac{I_i}{t_i} \right) \equiv \sum_{i=1}^{M} t_i e_i$$

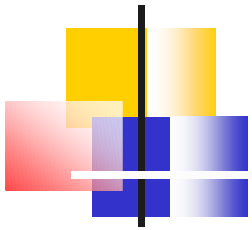
Then ...

$$p_i \equiv \frac{t_i}{T} = \frac{e^{-\beta e_i}}{\sum_{i=1}^{M} e^{-\beta e_i}} = \frac{e^{-\mu \log a_i}}{Q(\beta)}$$

and

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

A model for emergent power-law size behaviour using H-S information



Note !

- This result is very general. It says that for any complex system with constraints on size and total number of ways of selecting alternatives, the individual coagulations will most likely be distributed as a power-law in the number of alternatives available.

Let’s test this for software systems.

A model for emergent power-law size behaviour using H-S information

First, what are we looking for ?

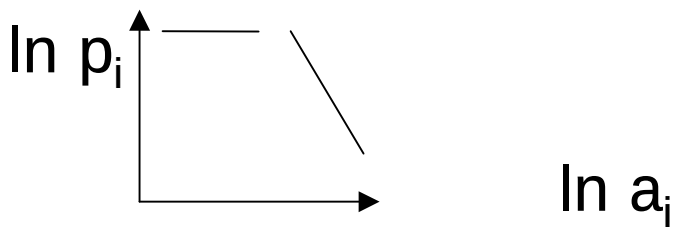
Small components, $a_f >> a_v(i)$

$$a_i^{-\beta} = a_f^{-\beta}(1+\frac{a_v(i)}{a_f})^{-\beta} \approx a_f^{-\beta}$$

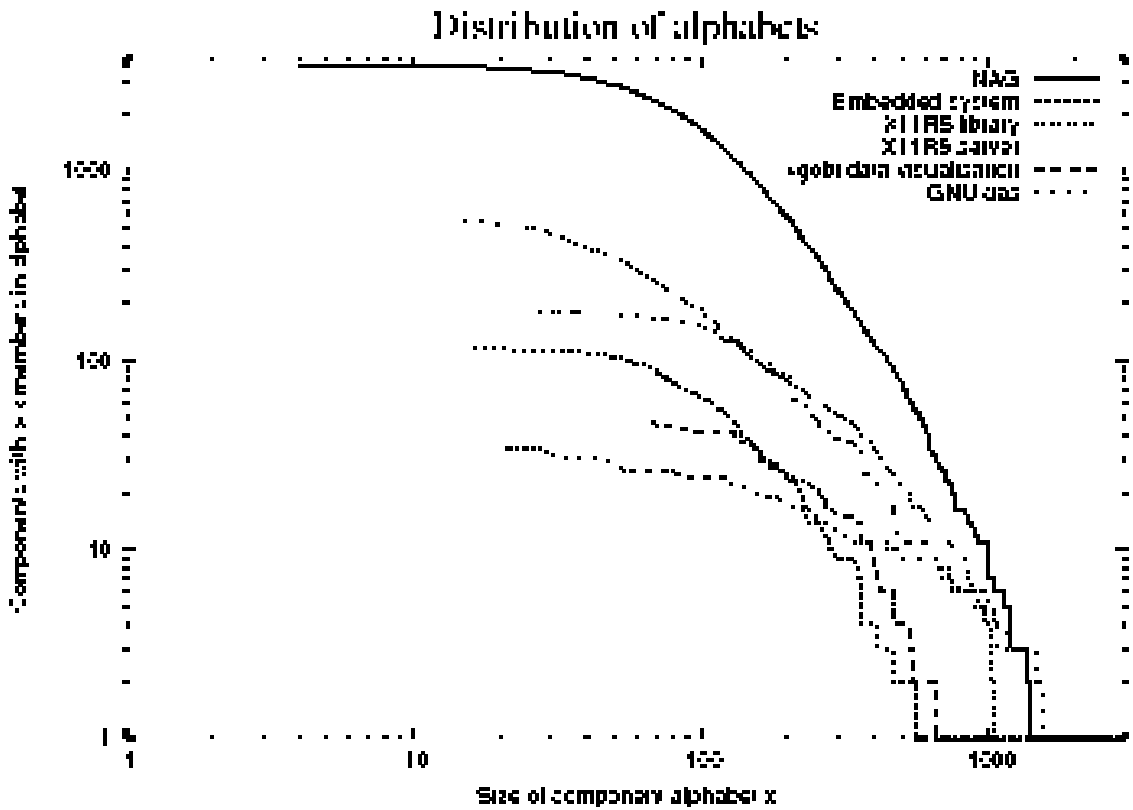
Large components, $a_f << a_v(i)$

$$a_i^{-\beta} = a_v^{-\beta}(i)(1+\frac{a_f}{a_v(i)})^{-\beta} \approx a_v^{-\beta}(i)$$

We are looking for .....

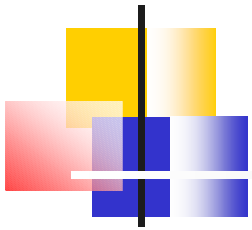


Application to software systems



NAG, embedded control system, X11R5 server and library, xgobi, gas

A model for emergent power-law size behaviour using H-S information



- So, for any software system however implemented, if the total size and total number of ways of selecting alternative tokens is fixed, the probability p_i that a component using an alphabet of a_i tokens will appear is given by

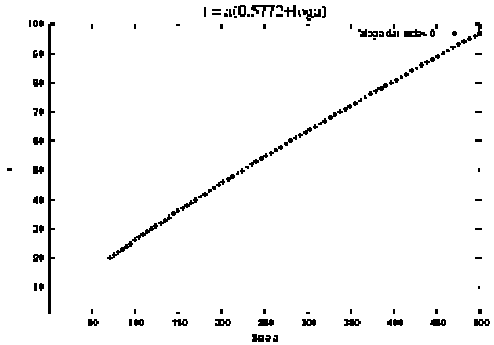
$$p_i \approx \frac{C}{a_i^{\beta}}$$

The implication is that power-law behaviour is nothing to do with functionality. It is to do with the presence of constrained alternatives only.

Some other predictions for software systems – linearity in user-defined names

Zipf's law implies that the total number of tokens in a component is approximately related to the unique alphabet by

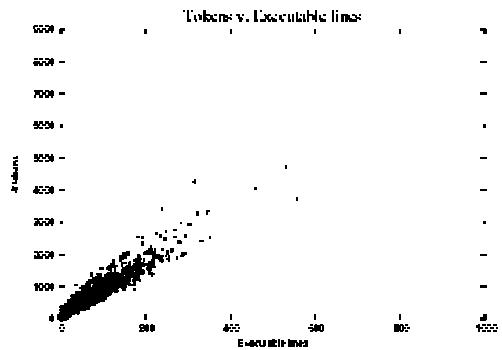
$$t_i \approx a_i(\gamma + \ln a_i)$$



$$\Rightarrow t_i \approx a_i \lambda$$

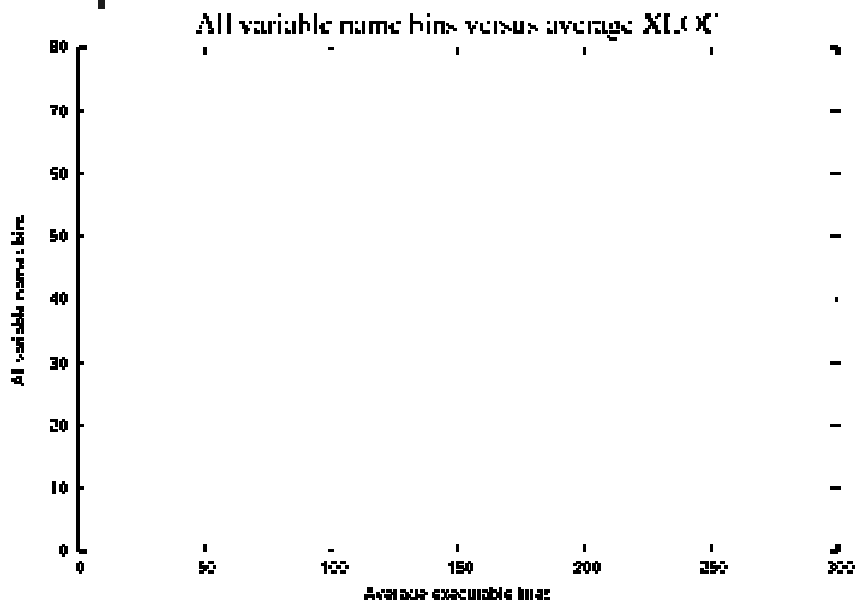
$$a_v(i) >> a_f \quad \Rightarrow t_i \approx a_v(i)\lambda$$

$$\Rightarrow n_i \approx a_v(i)\lambda'$$



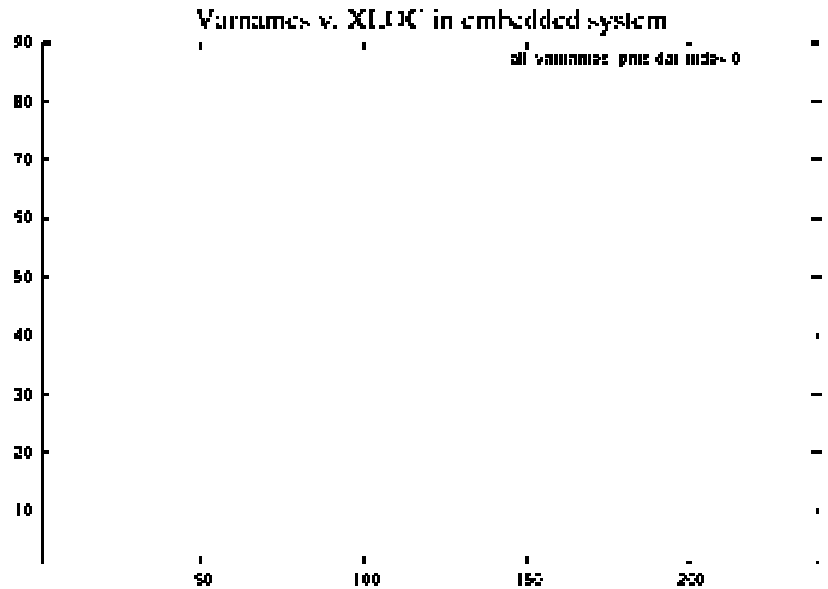
Number of user-defined names roughly linear with XLOC

Some other predictions for software systems – linearity in user-defined names

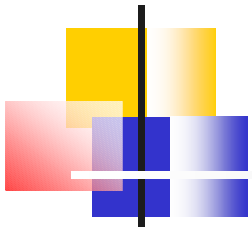


NAG Fortran

Embedded control system



Some other predictions for software systems – power-law behaviour in defect



Using Zipf again

$$t_i \approx a_i \lambda \qquad \Rightarrow p_i \approx \frac{C}{t_i^{\nu}}$$

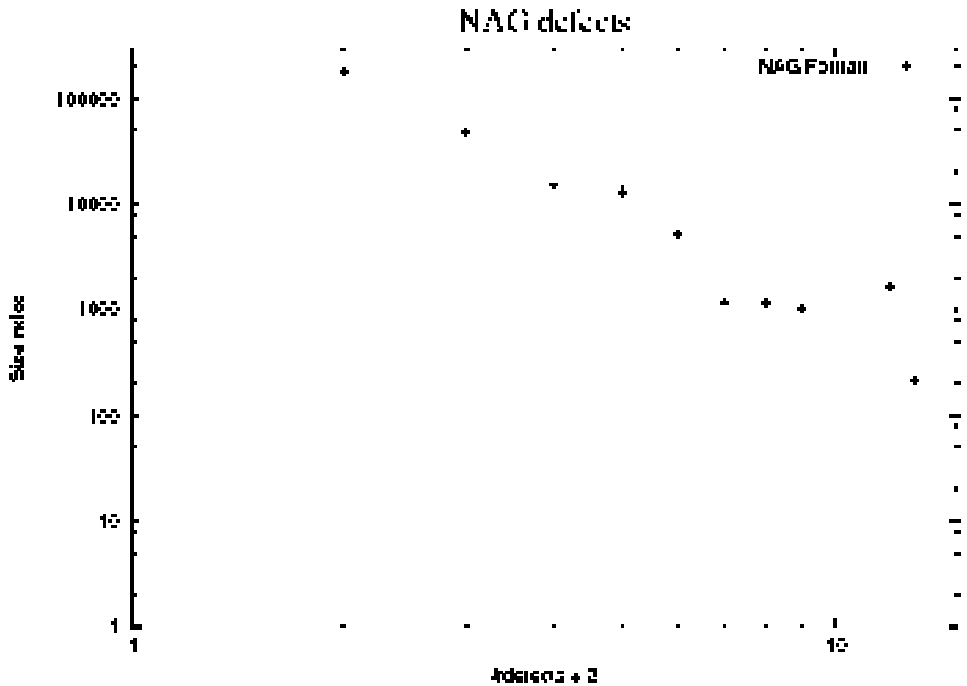
Now assume a fixed probability that any particular token is a defect.

$$\Rightarrow p_i \approx \frac{C}{d_i^{\nu'}}$$

Number of defects will also approximately obey a power-law.

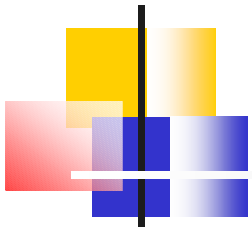
Some other predictions for software systems – power-law behaviour in defect

So does it ?



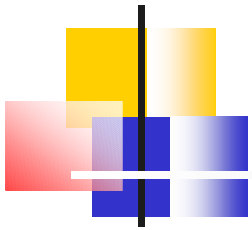
One implication of this is that defects cluster, (since 80% of the defects occur in 20% of the components – the Pareto distribution).

Some other predictions for software systems – power-law behaviour in defect



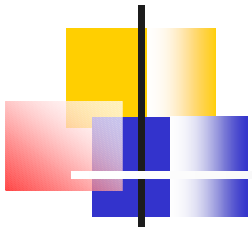
Defect clustering has been observed so often it appears as property number 4 in Boehm and Basili's software defect reduction top 10 list.

Overview



- What is scale-free behaviour and where does it occur ?
- The story so far
- Is scale-law behaviour persistent in software systems ?
- A unifying model
- Conclusions

Conclusions of early work

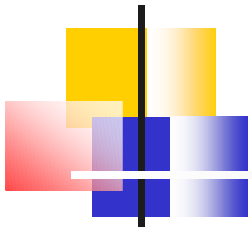


- Subject to simple constraints, the appearance of power-law behaviour in the following form seems inevitable, where p_i is the probability of occurrence and a_i is the alphabet used in component i.

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

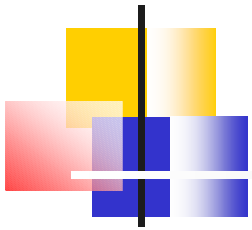
- This is able to predict other approximate patterns including
  - User-defined names are approximately linear with size
  - Defects will cluster

Tentative conclusions



- The model defined here may explain why power-law behaviour is so prevalent in nature. It is very likely to occur anywhere where a complex system evolves in components subject to constraints on size and total number of alternative choices.

References



My writing site:-

http://www.leshatton.org/