

On the conservation of software defect

Les Hatton
CISM, University of Kingston*

August 18, 2007

Abstract

This paper begins by analysing general software systems using concepts from statistical mechanics which provide a framework for linking microscopic and macroscopic features of any complex system. This analysis provides a way of linking two features of particular interest in software systems, first the microscopic distribution of defects with component size and second the macroscopic distribution of component sizes in a typical system. The former has been studied extensively but the latter much less so. This paper shows that subject to an external constraint that the total number of defects is fixed *a priori*, then commonly used defect models for individual components directly imply that the distribution of component sizes in the overall system will obey a power-law Pareto distribution.

The paper closes by analysing a significant number of systems of different total size, different implementation languages and very different application areas and demonstrates that the component sizes overwhelmingly obey the predicted power-law distribution. Some possible implications of this are explored.

Keywords: Defects, Macroscopic system behaviour,
Component size distribution, Pareto

1 Overview

Software systems are typically of immense complexity, far larger than that of physical systems. For example, if a program function is assembled using 200

*L.Hatton@kingston.ac.uk, lesh@oakcomp.co.uk

independent decision statements in series, (this may only take around 1400 lines of code, [5]), the number of distinct paths through that function will be 2^{200} , a truly gigantic number. Given that software systems 1,000 times bigger than this are regularly constructed and systems 10,000 times bigger occasionally so, it is not difficult to appreciate that the systems we build may have a complexity far in excess of anything naturally occurring in the physical world.

In the physical sciences, the methodology used for dealing with extremely complicated physical systems is called statistical mechanics. One of the great simplifying properties of physical systems allows thermodynamic descriptions to be built with a very small number of variables which describe the properties of very complex systems extremely accurately. For example, for a gas, $PV = RT$ where P is the pressure, V the volume, T the temperature and R the gas constant is astonishingly accurate over a very wide range of pressures and temperatures. That a system containing roughly 10^{23} molecules per litre in random, rapid motion can be described with such accuracy by such a simple equation is a matter for both awe and relief. Such models have evolved both empirically and theoretically by the scientific method over the last few hundred years.

In the 20th and 21st centuries, once again, immensely complicated systems have evolved in the form of software systems, this time through human action and this raises the question as to whether there are simplifying models equivalent to the thermodynamic relationships of physics which can be applied to such systems. Thermodynamic relationships in systems deriving from human action have been discussed before, [3] (software systems), [10] (economic systems), but in this paper a rather different approach will be taken and observed defect models in components will be considered alongside the implications of this statistical mechanical approach and used to make predictions for component size distributions which will then be tested.

1.1 A statistical mechanical model of a software system

Suppose a software system of N lines be split into M components, each containing n_i lines where $i = 1, .. M$. Then the number of ways of organising this system is given by:-

$$W = \frac{N!}{n_1!n_2!..n_M!} \quad (1)$$

where

$$N = \sum_{i=1}^M n_i \quad (2)$$

Taking the natural log of equation (1) and using Stirling's approximation for a factorial as usual yields

$$\log W = N \log N - \sum_{i=1}^M n_i \log(n_i) \quad (3)$$

The validity of this approximation will be returned to later.

In physical systems, the most likely combination is found by maximising equation (3) subject to additional constraints. The first is obviously equation (2). The second constraint is usually of the form

$$U = \sum_{i=1}^M n_i \varepsilon_i \quad (4)$$

where ε_i is the energy of the i th component, determined by quantum mechanical arguments and U is therefore the total energy. In other words, they are externally imposed. Using the method of Lagrangian multipliers, the following variational is to be maximised

$$\log W = N \log N - \sum_{i=1}^M n_i \log(n_i) + \alpha' \{N - \sum_{i=1}^M n_i\} + \beta \{U - \sum_{i=1}^M n_i \varepsilon_i\} \quad (5)$$

Setting $\delta(\log W) = 0$ leads to

$$0 = - \sum_{i=1}^M \delta n_i \{\log(n_i) + \alpha + \beta \varepsilon_i\} \quad (6)$$

where α is derived from α' . This must be true for all variations δn_i and so

$$\log(n_i) = -\alpha - \beta \varepsilon_i \quad (7)$$

Using equation 2, this can be manipulated into the most likely distribution

$$n_i = \frac{N e^{-\beta \varepsilon_i}}{\sum_{i=1}^M e^{-\beta \varepsilon_i}} \quad (8)$$

So the probability $p_i = \frac{n_i}{N}$ that a component gets a share of E equal to ε_i is given by

$$p_i = \frac{e^{-\beta \varepsilon_i}}{\sum_{i=1}^M e^{-\beta \varepsilon_i}} \quad (9)$$

Conservation of defect Historically, defects in software systems have proven extremely resilient to attempts to remove them so here, constraint (4) will be adjusted to be the total number of defects in the system.

$$D = \sum_{i=1}^M d_i \quad (10)$$

Noting that this can be written as

$$D = \sum_{i=1}^M n_i \left(\frac{d_i}{n_i} \right) \quad (11)$$

leads directly to the identification of ε_i with $\left(\frac{d_i}{n_i} \right)$ in equation (8). In other words, each line of component i has a defect density associated with it given by $\left(\frac{d_i}{n_i} \right)$. It is interesting at this point to pause for a moment and consider empirically observed distributions of defects in components of real systems, $d_i = d_i(n_i)$.

1.2 Some existing component defect models

There have been numerous attempts at modelling microscopic defect behaviour as a function of component size in software over the years, some of which are summarised in Table 1 for different programming languages where d_i is the number of defects (i.e. faults that have failed) found in a component i with n_i lines of code. All are based on data fitting but the fifth also attempts to build a model first based on known properties of the human reasoning system, [6] and goes on to show good agreement with real systems in programming languages as disparate as assembler and Ada. Although functionally somewhat different, the general behaviour of these models is qualitatively similar in their slow growth with n_i , an observation which will be important in this analysis as one will be chosen as representative.

Source	Model
Akiyama, [1]	$d_i = 4.86 + 0.018n_i$
Lipow, [8]	$d_i = a_0n_i + a_1n_i \cdot \log(n_i) + a_2n_i \cdot \log^2(n_i)$
Gaffney, [4]	$d_i = 4.2 + 0.0015n_i^{\frac{4}{3}}$
Compton and Withrow, [2]	$d_i = 0.069 + 0.00516n_i + 0.00000047n_i^2$
Hatton, [6]	$d_i = \log(n_i) + 0.005n_i + 0.00000625n_i^2$

Table 1: Some defect models in the literature.

Identifying ε_i with $(\frac{d_i}{n_i})$ and using a representative model of useful functional behaviour from Table 1, (the first two terms of Lipow) conveniently reduce to

$$\left(\frac{d_i}{n_i}\right) = a_0 + a_1 \log(n_i) \quad (12)$$

Substituting in equation (8) yields

$$p_i = \frac{n_i^{-\beta a_1}}{Q(\beta)} \quad (13)$$

where

$$Q(\beta) = \sum_{i=1}^M n_i^{-\beta a_1} \quad (14)$$

In essence, equation (13) is none other than a Pareto power-law distribution, see for example [11] and [10]. *In other words, with the constraints of constant defect (equation (2)) and constant size (equation (10)), assuming a microscopic defect model representative of those shown in Table 1 leads directly to a macroscopic prediction that the most likely scenario will be that component sizes will be distributed according to a Pareto distribution.*

This will now be tested on a number of disparate real systems.

1.3 An analysis of component size distribution in real systems

There are a bewildering number of design methodologies and languages in use in building software systems. In spite of this, the distribution of component sizes has not generally been studied in depth. A recent and significant exception is the work of [9], who studied the distribution of object linkages to make inferences about the size and popularity of objects in OO systems. These authors specifically identified a power-law distribution in the observed sizes which led them to the important conclusion that "... the rhetoric of OO design is that large programs should be able to be constructed in just the same way as small programs, by encapsulating complexity within objects at one level of abstraction and then composing those objects together at the next. ... In fact the presence of a power law indicates the reverse: there is no evidence of typical sizes (a Lego brick) to objects at all.". In other words, the systems are scale-free. Other departures from expected behaviour in OO systems have also been described by [7].

There appears to be no equivalent analysis for non-OO systems so this intriguing result was followed up here by analysing twenty-one systems chosen in three different non-OO languages (C, Tcl and Fortran) from very disparate application areas to see if the prediction from the above section held true. Fifteen of the systems were written in C, two in Fortran and four in Tcl. For each language, the systems were partitioned where possible into two populations, one containing systems of more than 10,000 lines of code and the other less than this amount. This partition relates to one of the approximations used in the development of Equation (3). The nature of the approximation relates to the large numbers required for Stirling's formula to be a good approximation. In statistical physics, this is rationalised by appealing to the principle of aggregating smaller objects into large objects. However, it has long been known that the approximation holds up surprisingly well, [3], even when it might be expected to be less effective. It was expected here that if anything was to break down it would be in smaller systems, so these were analysed separately.

Displaying power-law distributions is somewhat complicated when distributions are discrete such as those occurring for component sizes in software systems. This can be ameliorated by plotting probability of occurrence by rank as a log-log distribution rather than by size, [9], but here they will be plotted as a cumulative density function to smooth out steps caused by gaps in component sizes, particularly as components get bigger, (for example it is normal to find components of size 500 and 1000 lines of code, say and nothing in between). The expected behaviour of the cumulative density function for a power-law distribution can be found by integrating a probability density function

$$p(s) = \frac{k}{s^\alpha} \quad (15)$$

to get

$$P(s < z) = \frac{1 - s^{1-\alpha}}{(1 - (S + 1)^{1-\alpha})} \quad (16)$$

for a system with maximum component size S where $\alpha \neq 1$. P is then the cumulative probability that a particular component has size less than z. This has the expected behaviour $P(s < 1) = 0$ and $P(s < (S+1)) = 1$. Figure 1 plots equation (16) for systems with different maximum component sizes = {150,450,750,1050,1350} and for two values of the power exponent, $\alpha = 1.1$ and 2.0 plotted against the maximum component size, normalised to 1.

Figures 2 - 7 show the results of analysing a number of real systems for their component size distributions plotted in exactly the same way as Figure

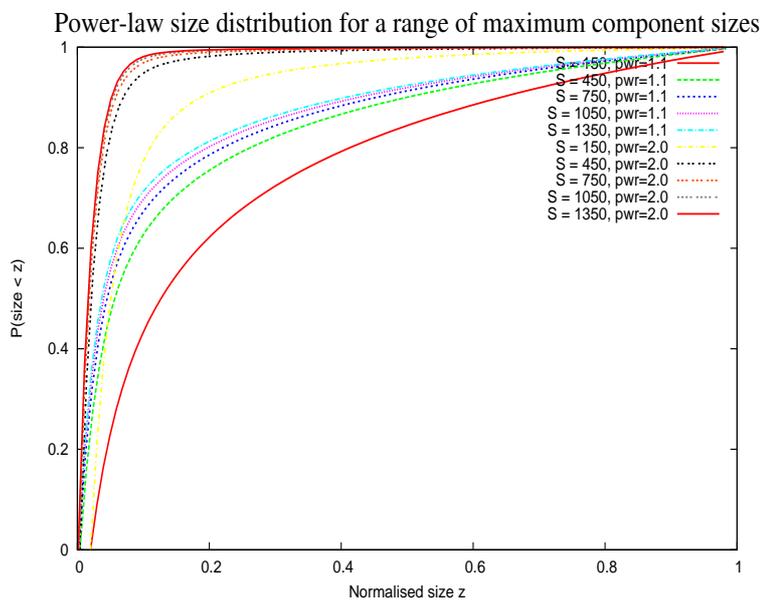


Figure 1: Theoretical distributions of equation (16) for systems with different maximum component size. The vertical axis is the cumulative distribution function, i.e. the fraction of all components less than a certain size in the system and the horizontal axis is the maximum size normalised to 1.

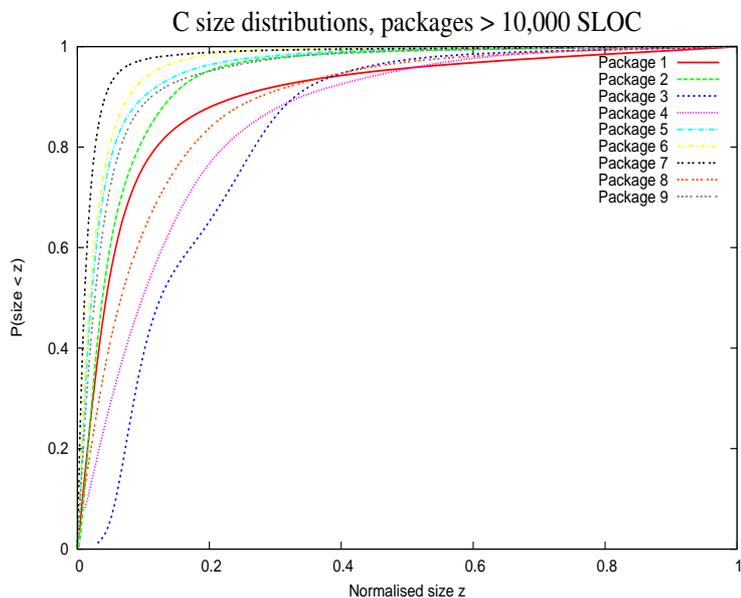


Figure 2: The distribution of component sizes in C systems in the range 10,000-43,700 lines

1. Figure 2 shows the component size distribution for nine systems written in the programming language C of greater than 10,000 lines of code. These systems include language parsers, statistical packages, an embedded automobile engine controller, a scientific modelling package and a linear programming package used in abstract interpretation in the range 10,000-43,700 lines. In other words, they are very disparate. In spite of this diversity, the distributions follow an obvious power-law distribution as exemplified by the model distribution in Figure 1.

Interestingly, exactly the same is true for systems of less than 10,000 lines of code written in C as shown in Figure 3 which includes three other embedded control systems including a television set-top box, as well as two graphics packages.

For comparison with different languages, two Fortran scientific modelling packages, one large package of 279,000 lines of code (a geophysical modelling package) and one of 15,000 lines of code (a weather prediction system and the smallest available) are shown in Figures 4 and 5 and four Tcl systems, two between 20,000-30,000 lines and two with 2,500 and 5,500 lines are shown in Figures 6 and 7 respectively. Again precisely the same behaviour appears.

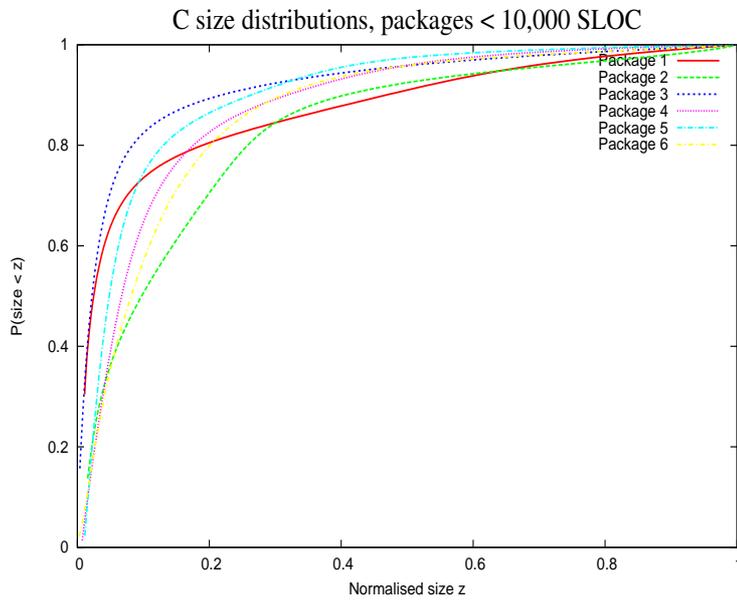


Figure 3: The distribution of component sizes in C systems in the range 1,300-4,600 lines

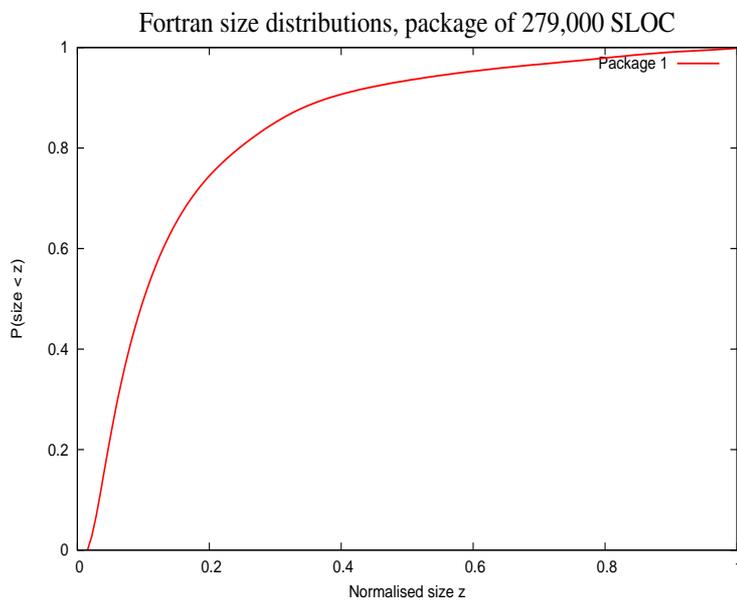


Figure 4: The distribution of component sizes in a Fortran system of $\geq 279,000$ lines

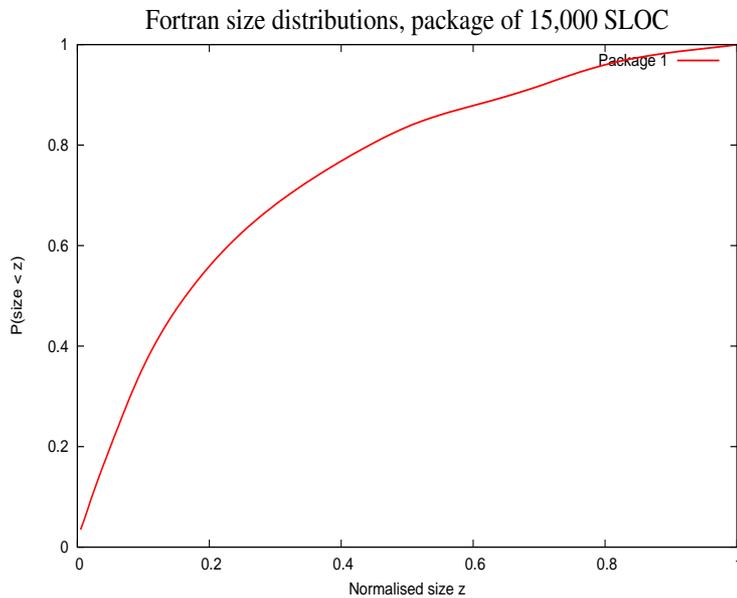


Figure 5: The distribution of component sizes in a Fortran system of $\leq 15,000$ lines

In the interests of both pedagogy and repeatable science, the full anonymised raw data for these experiments along with the analysis code are freely available for download and analysis ¹ in the form of a zip archive.

1.4 Discussion and Conclusions

This paper demonstrates that the externally imposed constraints of total size *and total defects* along with the microscopic phenomenon of typically observed defect behaviour models as a function of component size lead directly to a macroscopic prediction that the most likely component size distribution in a system will behave according to a Pareto power-law.

The paper then tests this prediction, analysing component size distributions across numerous systems implemented in three different programming languages with very different system sizes and application areas and shows this prediction to be true without exception.

¹http://www.leshatton.org/Data_Statmech_18-08-2007.html

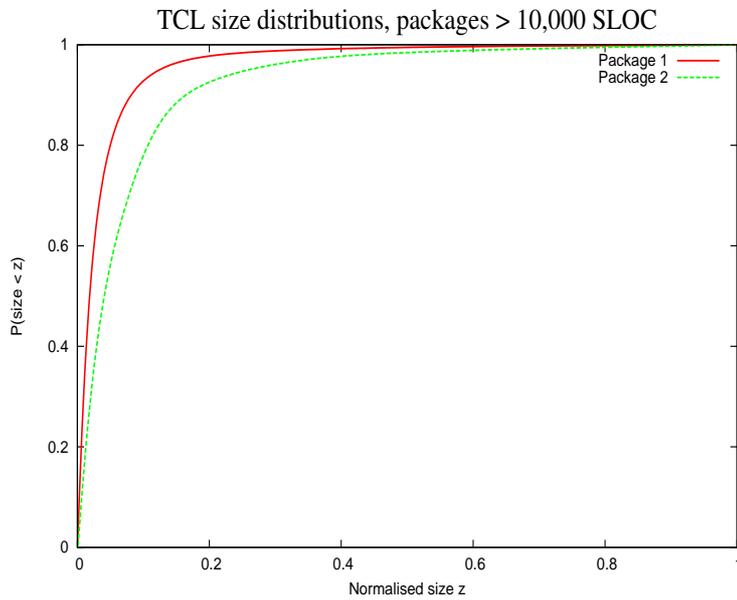


Figure 6: The distribution of component sizes in two TCL systems of $\geq 20,000$ and $29,000$ lines

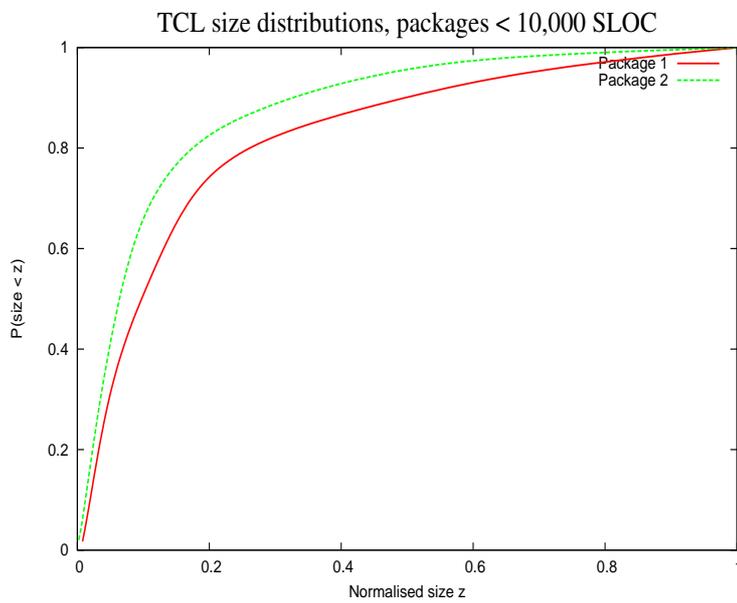


Figure 7: The distribution of component sizes in two TCL systems of $\leq 2,600$ and $5,500$ lines

This is an intriguing result. There are three related factors and it is not clear which, if any is the driver. The factors are:-

1. The assumed *a priori* enforced constraints of total size in lines and total number of defects,
2. Typically observed models for growth of defects in small to medium sized components as a function of component size,
3. The observed close adherence to a power-law component size distribution in systems of great diversity of application area.

Explicit constraints such as the first above often appear in complex systems, [10]. That the constraint here that the total number of defects is effectively specified at system design time along with the system size is by analogy with statistical mechanics and physical systems, merely a reflection that defects in software systems may well play a role very similar to that of entropy in a physical system. (This can be seen by considering equations (3) and (7) along with the identity of ε_i with $\frac{d_i}{n_i}$).

The question also arises whether the observed defect behaviour with component size drives the power-law component size distribution (the microscopic determining the macroscopic) or the other way round. As presented by [6], there is a model for logarithmic defect behaviour based on human reasoning so on balance it may be that the power-law distribution of component size observed so prevalently here follows directly from the defect signature placed on components by limitations in human reasoning given of course the underlying constraint of specifying the total number of defects *a priori*. As in physical systems, the microscopic would then appear to determine the macroscopic.

Perhaps the most thought-provoking part of this analysis is the appearance of general predictable macroscopic behaviour independent of representation language, size or application area from relatively simple assumptions and common observation of defect behaviour. This is entirely consistent with the important observations of [9], whereby extensive analysis of OO systems reveals scale-free behaviour which abrogates a number of the promised design benefits of that paradigm. Such scale-free behaviour across both OO and non-OO systems gives rich promise for reasoning about systems, particularly their defect behaviour, without being overwhelmed by the complexity of paradigm shifting and implementation detail.

References

- [1] F. Akiyama. An example of software system debugging. *Information Processing*, 71:353–379, 1971.
- [2] B.T. Compton and C. Withrow. Prediction and control of ada software defects. *Journal of Systems and Software*, 12:199–207, 1990.
- [3] R.P. Feynman. *Lectures on Computation*. Penguin, 1996.
- [4] J.R. Gaffney. Estimating the number of faults in code. *IEEE Transactions on Software Engineering*, 10(4), 1984.
- [5] L. Hatton. *Safer C: Developing software in high-integrity and safety-critical systems*. McGraw-Hill, 1995. ISBN 0-07-707640-0.
- [6] L. Hatton. Re-examining the fault density v. component size connection. *IEEE Software*, 14(2):89–98, 1997.
- [7] L. Hatton. Does oo sync with the way we think ? *IEEE Software*, 15(3):46–54, 1998.
- [8] M. Lipow. Number of faults per line of code. *IEEE Transactions on Software Engineering*, 8(4):437–439, 1982.
- [9] A. Potanin, J. Noble, M. Freat, and R. Biddle. Scale-free geometry in oo programs. *Comm. ACM.*, 48(5):99–103, May 2005.
- [10] P.K. Rawlings, D. Reguera, and H. Reiss. Entropic basis of the pareto law. *Physica A*, 343:643–652, July 2004.
- [11] G.K. Zipf. *Psycho-Biology of Languages*. Houghton-Mifflin, 1935.