Failure-aware programming: and introduction and some predictions

Les Hatton
Computing Laboratory, University of Kent*

December 12, 2003

Abstract

In conventional science, physical models are built from which predictions can be made which when tested empirically can be used to reject inadequate models. In essence, this is the scientific method. This paper uses an existing empirical model of failure in software systems to make predictions of techniques which should improve system reliability. As a by-product of the model, it also introduces reasonable definitions of system complexity as a function of component complexity and defines the difference between essential and nonessential complexity in terms of defect. Some of the predictions are non-intuitive but collectively lead to a style of programming which is called here *Failure Aware Programming*.

At the time of writing, none of these predictions have been tested although some of them seem to accord well with accepted principles. More experimental work is necessary to exercise these testable predictions.

$Date: 2003/12/12 19:14:05 $

1 Overview

In [2], an empirical model for the number of faults which have failed (known here as *defects*) in a system as a function of the average size in lines of the component in which they appeared is presented. The model is shown in Figure 1 compared with actual recorded data for large systems written in Ada (where a component is a package) and assembler (where a component is a function). The model is based on known properties of the short-term and long-term memory in the human brain, see for example [3], but the recorded behaviour has been observed across numerous programming languages and systems in addition to those shown.

Figure 1 has two compelling features. First of all the data for systems written in programming languages as disparate as Ada and assembler is extraordinarily similar both quantitatively and qualitatively. This is strongly suggestive of the fact that defect growth is related to symbolic manipulation limits in the

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

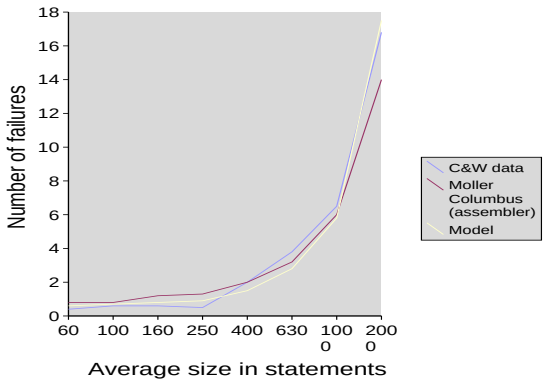

Figure 1: Combined plot Ada, assembler and modelled failure data as function of the average size of component in which the failures appeared.

human reasoning system. Second, the model predicts the qualitative behaviour very well if the size of component at which the quadratic behaviour appears is around 200 lines. Mathematically, the model looks like:-

$$D_i = k\{log(W_i) + \frac{1}{2}(\frac{W_i}{W'} + \frac{1}{2}(\frac{W_i}{W'})^2\} \quad (1)$$

where D_i is the number of defects (faults that failed), W_i is the size in executable lines of code of component i and W' is the size at which quadratic behaviour appears. Note that the only fitting parameter present in the original model was W' but an additional multipler k has been added for reasons which will become obvious later. In practice, this constant is close to 1. The model itself is derived from observed models of behaviour in the human short-term and long-term memory.

It should be noted that the observed data in Figure 1 arose without any attempt to interfere in the design process. The systems simply evolved into their particular distribution of component sizes without any external influence. So this data is saying that this defect (fault that failed) versus component size distribution evolved in conditions where there was no external influence on the particular distribution of component sizes other than the 'normal' way in which these rather disparate software systems naturally evolved. This is a crucial point because the simple extrapolation of model fitted data without any underlying conceptual basis is highly questionable if extrapolations are made outside the environment under which these measurements were made.

At the risk of labouring the point, it would for example be valuable to make sensible predictions about the defect behaviour of a system where all the components were constrained to be of the same or very similar size. However, it would be exceedingly unlikely for this to occur naturally so some kind of interference in the development process would be necessary. An underlying independent conceptual basis is vital if such predictions are to be valid as the scientific method of course is about the continual rejection of underlying conceptual models as understanding grows until they are of satisfactory predictive behaviour.

There is one area in which the data shown in Figure 1 has been extended by influence in the development process and that is the upper end. There is some evidence, also cited in the original reference, that enforcing a maximum component size in a particular system development simply truncates the curve. No similar data exists for any kind of interference at the lower end of component size to the author's knowledge.

2 Implications of this model

The first concept we will explore is whether or not individual component complexity can be combined together in a compact way to provide a definition of system complexity.

2.1 System level implications

2.1.1 Defining System Complexity

The total number of defects D in a system is obtained simply by summing together all the individual defects.

$$D = \sum_{i=1}^{N} D_i \tag{2}$$

so using equation (1),

$$D = k\{\sum_{i=1}^{N} logW_i + \frac{1}{2}\{\sum_{i=1}^{N} \frac{W_i}{W'} + \frac{1}{2}\sum_{i=1}^{N}(\frac{W_i}{W'})^2\}\} \tag{3}$$

We can rewrite this as:-

$$D = k\{log\prod_{i=1}^{N} W_i + \frac{1}{2W'}\{\sum_{i=1}^{N} W_i + \frac{W'}{2}\sum_{i=1}^{N}(\frac{W_i}{W'})^2\}\} \tag{4}$$

This is revealing. It suggests that for systems which consist of a population dominated by components smaller than W' (therefore making the second and third terms on the right hand side relatively small), it is perfectly reasonable to define a measure of system complexity W as:-

$$W = \prod_{i=1}^{N} W_i \tag{5}$$

These developments permit the association of the number of defects directly with a concept of entropy in a way exactly analogous to Boltzmann's equation for physical systems:-

$$D = klogW + \frac{k}{2W'}\{\sum_{i=1}^{N} W_i + \frac{W'}{2}\sum_{i=1}^{N}(\frac{W_i}{W'})^2\} \tag{6}$$

Another broad brush way of interpreting this is that the presence of the linear and quadratic terms alongside the logarithmic term means that programmers construct systems whose complexity is very unlikely to occur in a natural physical system where only the logarithmic term is present. On the other hand, restricting the maximum size of components so that the analogue with Boltzmann's equation is strong means that there may be other useful results from statistical mechanics which can be used in describing the failure behaviour of software systems. This will be explored in a separate paper. Note that the evidence described earlier whereby the curve is truncated when developers are constrained in component size means that the second and third terms on the right hand side of the equation above disappear.

2.1.2 Exploring component size distributions

Support for functional languages Suppose a system has a total of L lines and N components. Then

$$L = \sum_{i=1}^{N} W_i \tag{7}$$

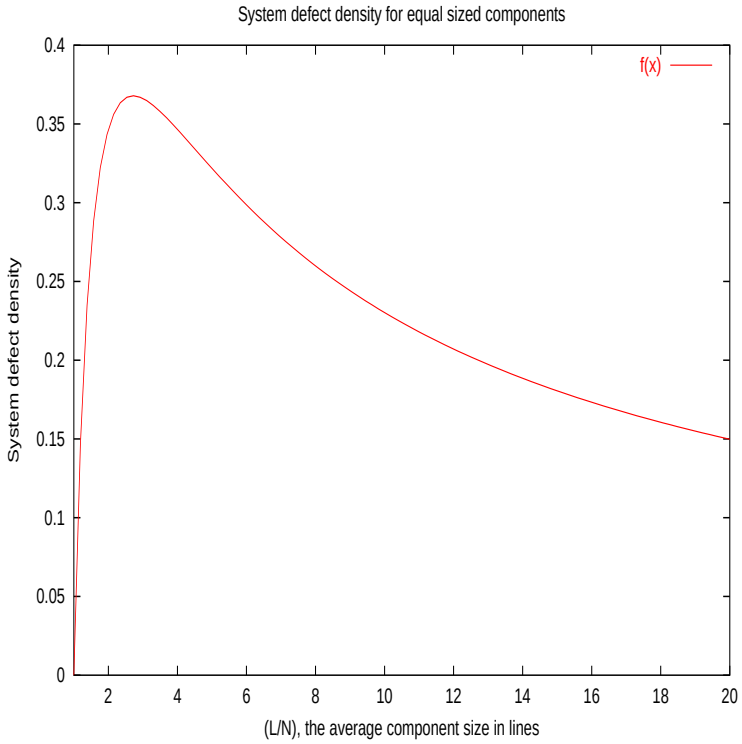

Figure 2: The system defect density as a function of component size for a system constructed from equal size components

Initially, the linear and quadratic terms will be ignored and a software system with logarithmic behaviour will be explored. For such a system, the total number of defects, D is given by

$$D = k \sum_{i=1}^{N} logW_i \tag{8}$$

Consider a system with equal sized components so that $W_i = \frac{L}{N}$. Then

$$D = kNlog\frac{L}{N} \tag{9}$$

Defining the system defect density as $D_s = \frac{D}{N}$ and the component size as $C_s = \frac{L}{N}$,

$$D_s = k\frac{1}{C_s}logC_s \tag{10}$$

Figure 2 plots the system defect density against the component size as the component size is varied for a system of fixed size.

Figure 2 has three particularly interesting properties.

- As components are reduced in size within the same system, the system defect density gradually gets worse
- There is a global *maximum* for components of a few lines
- There is an abrupt *minimum* for components of size 1. This may indicate support for systems built from effectively single line components for example, in functional languages, that they are the most reliable way of implementing a system, (for which the author confidently expects to be made the patron saint of functional languages).

Reducing defect in an existing system Differentiating with respect to L gives

$$\frac{\partial D}{\partial L} = k\frac{N}{L} \tag{11}$$

Differentiating with respect to N gives

$$\frac{\partial D}{\partial N} = k\{log\frac{L}{N} - 1\} \tag{12}$$

Three scenarios can be distinguished in which the total number of defects is reduced:-

1. $\frac{\partial D}{\partial L} > 0$ keeping N constant and letting L *decrease*. This corresponds to reducing the total number of lines for the same number of components. The closest analogue to this is the mechanism of re-use. For a given system, the effect is largest when $\frac{N}{L}$ is large, for example when the average component size in a system is small.

1. $\frac{\partial D}{\partial N} > 0$ keeping L constant and letting N *decrease*. This corresponds to reducing the number of components for the same total number of lines. This increases the trend to larger monolithic components providing they do not approach the W' point where linear and quadratic behaviour become important.

1. $\frac{\partial D}{\partial N} < 0$ keeping L constant and letting N *increase*. This corresponds to reducing further the size of components which are already small whilst keeping the total number of lines constant. Components around the maximim in Figure 2 would benefit the most.

2.2 Component level implications

2.2.1 Variations between individual programmers

In [4], wide variations of as much as 80% (7 ± 2 implying a range of $5 \rightarrow 9$) are described between different individuals. This corresponds to moving the W' point between say 120 and 280. Irrespective of what the 7 ± 2 actually refers to in programming, (see discussion below), this could give variations of perhaps a factor of 2-3 between different programmers. Differences rather larger than this appear to get reported anecdotally although there seems little supporting evidence unfortunately. However, it should also be noted that in a system where there are no components beyond the W' point, there should be much less difference in the defect count between individual programmers. This is a testable hypothesis.

2.2.2 Abstraction is a benefit

Whilst this may seem punishingly obvious, its effect can be seen clearly in this model. Since the curve for assembler and Ada data are so similar, this predicts that for the same component size distribution, Ada systems will be significantly more reliable than assembler systems simply because a line of Ada code corresponds to perhaps 4 or 5 assembler lines on average. This corresponds to the known property of the human short-term memory whereby the 'size' of the objects being manipulated there is critically related to how well they are understood. It is as if what is being manipulated in short-term memory is simply very reliable tags to other concepts. If a subject is not understood well, the tags may be very primitive and fine-grained. On the other hand, if the subject is well understood, the tags may be much more sophisticated and coarser-grained.

As an example, think of the sequence

1. copy file
2. remove original file

The notion of "copy file" is a high-level tag and would correspond to a large number of assembler instructions responsible for editing the directory list but it is phonemically easy to think about and manipulate in the short term memory. Furthermore, there are only two parameters to remember, the source file name and the destination file name. The whole sequence would be considered to be phonemically easy to work with in the short-term memory sense even though together, it would generate many assembler instructions. This phenomenon is also known as "chunking", [1] and [5].

3 Summary of significant predictions so far

3.1 Prediction 1

If programmers could be compared for small and large components, there should very little difference in their injected defect rates for small components but much bigger variation for larger components.

3.2 Prediction 2

Increasing abstraction whereby a single line in any given representation corresponds to more functionality is expected to continue to improve systems reliability. This supports the notion of code generation from a higher level model as a systems reliability increasing feature.

3.3 Prediction 3

The total defect density will be at a minimum for a system built from 1 line components.

3.4 Prediction 4

Re-use will have the biggest beneficial effect on reliability when the average component size is small.

4 Necessary and unnecessary complexity

Unnecessary complexity will here be *defined* as the complexity of any component whose defect behaviour is linear or quadatic.

To explore the effects of this, consider a system consisting of a number N' of equal sized small components with total number of lines L' and a relatively smaller number N" of equal sized large components totaling L" lines so N = N' + N" and L = L' + L". For such a system, the total number of defects

$$D = k\{\sum_{i=1}^{N} logW_i + \frac{1}{2}\{\sum_{i=1}^{N} \frac{W_i}{W'} + \frac{1}{2}\sum_{i=1}^{N}(\frac{W_i}{W'})^2\}\} \tag{13}$$

can be re-arranged as

$$D = k\{N'log\frac{L'}{N'} + \frac{1}{2W'}\{L'' + \frac{1}{2W'}N''(\frac{L''}{N''})^2\}\} \tag{14}$$

from which the following can be derived

$$\frac{D}{L} = k\{\frac{N'}{L}log\frac{L'}{N'} + \frac{1}{2W'}\{(1-\frac{L'}{L}) + \frac{L(1-\frac{L'}{L})^2}{2W'N(1-\frac{N'}{N}}\}\} \tag{15}$$

Now let $\alpha = \frac{L'}{L}$ and $\beta = \frac{N'}{N}$. The following equation then results

$$\frac{D}{L} = k\{\beta\frac{N}{L}log\frac{\alpha L}{\beta N} + \frac{1}{2W'}\{(1-\alpha) + \frac{L(1-\alpha)^2}{2W'N(1-\beta)}\}\} \tag{16}$$

Now let $w = \frac{L}{N}$ and the equation reduces to

$$\frac{D}{L} = k\{\beta wlog\frac{\alpha w}{\beta} + \frac{1}{2W'}\{(1-\alpha) + \frac{(1-\alpha)^2}{2W'(1-\beta)}w\}\} \tag{17}$$

Figure 3 gives a qualitative view of the gradual breakdown of the small component model showing the influence of a few large components on the system defect density. As can be seen, even a few big components have nowhere as bad an effect as a preponderance of smaller components.

5 Conclusions

This paper presents some simple developments of an abstract model of defect injection developed from the memory model of [2] and calibrated against observations from rather different real systems. Some of these have far-reaching

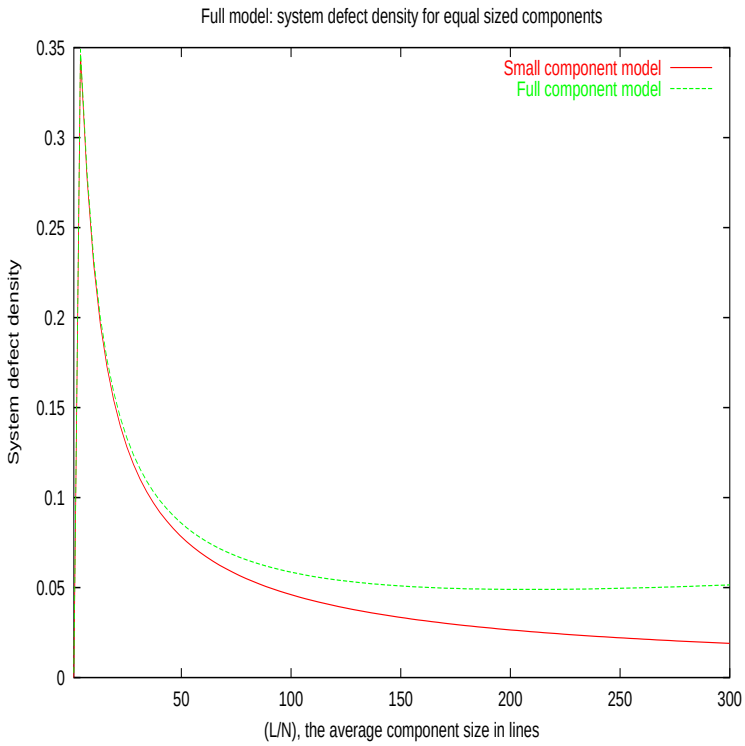

Figure 3: The small component model and the full model compared for some reasonable values of $\alpha = 0.0025, \beta = 0.0001$

.

implications for the production of reliable software and are not particularly intuitive. It remains to be seen how well these stand up to further experiment. As well as making predictions based on this model, this paper also introduces a natural definition for system defect based on the defects in individual components and also defines necessary and unnecessary complexity the same way. These definitions serve to bring out the close relationship with simple principles from statistical physics such as the Boltzmann relation.

Failure-aware programming is introduced here as the science of producing reliable programs using prior knowledge of the observed defect behaviour of systems generally.

References

[1] Greeno L. (1997) *The structure of memory and the process of problem solving* No.37 Human Performance Centre, University of Michigan

[2] Hatton L. (1997) *Re-examining the fault density v. component size connection* IEEE Software, 14(2), p.89-98

[3] Hilgard E.R., Atkinson R.C. and Atkinson R.L. (1971) *Introduction to Psychology* Harcourt Brace Jovanovich, New York

[4] Miller G.A. (1957) *The magical number 7 plus or minus 2; some limits on our capacity for processing information* Psychological Review, 63, p. 81-97.

[5] Shneiderman B. (1980) *Software Psychology* Winthrop Publishers, Cambridge MA