Invited talk, SSS'17, Bristol, 7-9 February 2017

"Balancing Safety with Rampant Software feature-itis"

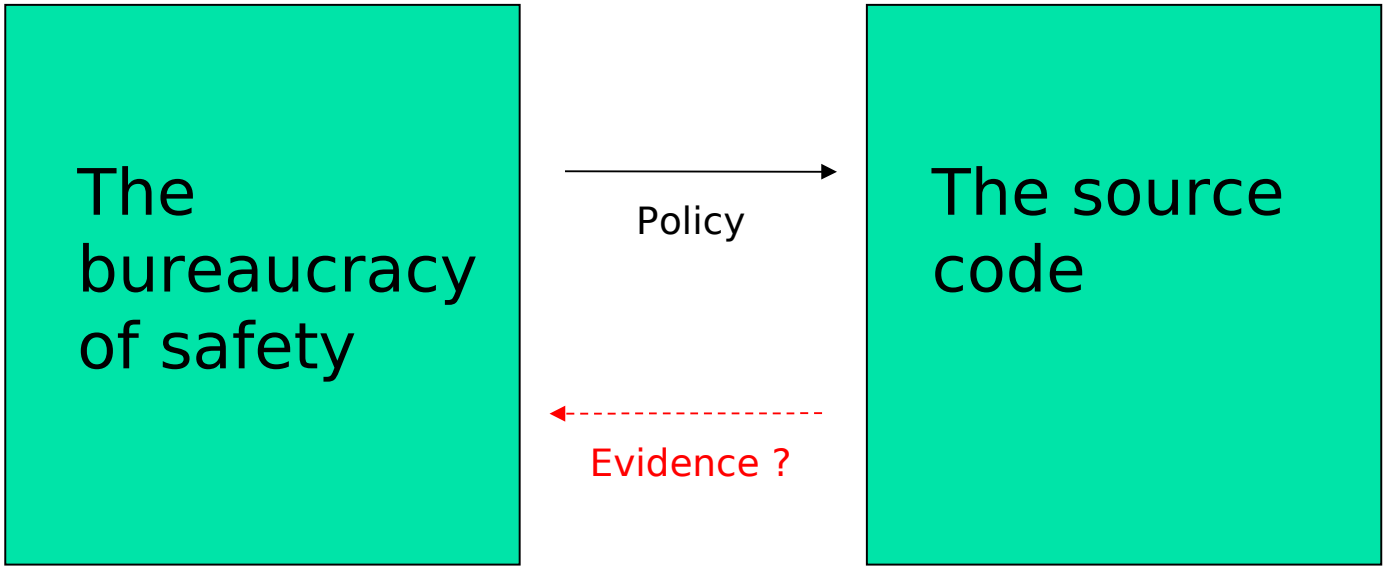
Version 1.2: 07/Feb/2017

© Les Hatton
Emeritus Professor of Forensic Software Engineering,
Kingston University, London

Overview

- A brief review
- Software growth and safety-related systems
- Quantifying software growth
- The distribution of software component lengths: some fundamental results
- Implications for assessing systems containing substantial software.

A brief review of safety-related software systems



Continuing inadequacies in empirical software engineering have undermined the natural feedback between failure experience and policy at the heart of all successful engineering.

Software growth and safety-related systems

- The Impact columns (IEEE Software 2010-)
  - About 30 systems described, several of them safety-related.
  - The majority appear in the 1-10 million SLOC range (including the safety-related systems).
  - Automotive systems appear to be in the 10-100 million SLOC range as we enter the autonomous driving age.
- The Toyota unintended acceleration bug(s?)
  - Some observations from expert witness accounts

A snapshot of recent automobile recalls

- 2016. Stout Risius Ross Warranty reports software related recalls have gone from 5% in 2011 to 15% by end of 2015. (189 distinct recalls in 5 years covering 13 million vehicles with 141 presenting higher risk of crashing).

  http://popsci.com/software-rising-cause-car-recalls
- 2016 Sep. GM to Recall 4.3 Million Vehicles for Software-Related Airbag Defect, http://fortune.com/2016/09/09/gm-recall-software/
- 2016. First known self-driving death in Tesla.
- 2015 June. Ford recall 433,000 cars for engines that don't shut off.
- 2015-. Drive by hacking. (Just google "Hacking automobiles").
- and Toyota …

Toyota unintended acceleration 2009-2013; some expert witness notes

Source code

- *> 11,000 global variables (NASA analysis)*
- *"Spaghetti code". 67 functions with cyclomatic complexity > 50, (throttle angle function > 100)*
- Recursion with no stack monitoring, (2005 Corolla had this !)
- Toyota's coding standard used only 11 MISRA-C rules and 5 were violated in the code.

Systems

- No adequate safeguards against memory corruption
- Redundancy problems – all fail-safes in same Task.
- No EDAC (Error Detecting And Correcting) memory

http://www.safetyresearch.net/Library/BarrSlides_FINAL_SCRUBBED.pdf

Toyota unintended acceleration 2009-2013; some expert witness notes

- These notes appear to consist of a mixture of good engineering analysis (e.g. the task death monitor) and software engineering folklore (global variables, cyclomatic complexity, lots of words ending in –ability …)
- The Toyota software clearly has serious concerns but our methods of analysing it are inadequate and riddled with opinion
- But the systems are still growing and we have autonomous driving on the way …

Quantifying software growth

- So, how fast are systems growing ? This we can answer …
  - On a corpus of 800 million lines of code, the 95% confidence interval of annual growth is (18%, 22%). So they double about every 42 months.
  - Safety-related systems appear to be about half as fast.

(Hatton, Spinellis, van Genuchten, 2017).

The distribution of lengths: some fundamental results

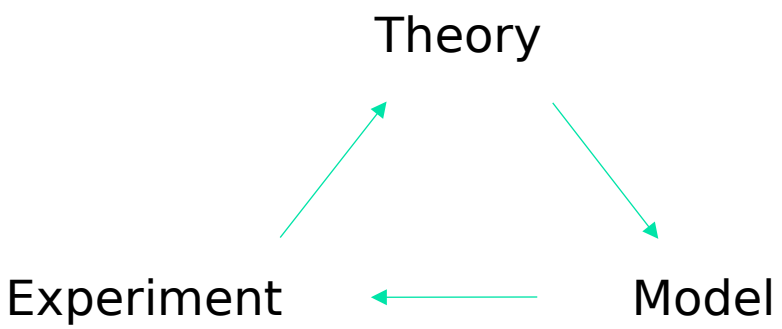
- The Toyota case specifically raised points of folklore (by which I mean no supporting statistically robust evidence).
  - Global variables are bad (i.e. in the sense of defects).

    but Tim Hopkins and I found NO statistically significant link over 25 years, (Hopkins and Hatton, 2008).
  - Spaghetti code, high cyclomatic complexity and other "measures" are bad, i.e. related to increased defect.

    After 9 years of trying, I believe I can now say with overwhelming statistical support that this is also irrelevant …..

The distribution of lengths: some fundamental results

- The scientific method depends on systematic improvement of the eternal triangle:

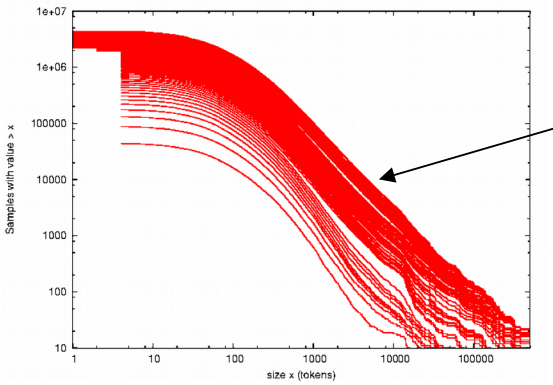


- but in software engineering we only have ...

The distribution of lengths: some fundamental results

- So back to the drawing board in search of theory:-
  - (2007, TSE) Found the same length pattern in 21 packages but realised LOC are useless as a measure for length … so use tokens instead.
  - (2008-2011) Wrote 7 compiler front-ends and analysed 100 million lines of code. Started noticing very strong evidence of *power-laws and scale-independence* …

Frequency of occurrence



Straight line on log-log -> power-law

Length of component in tokens

Which got me thinking of …
Emmy Noether's amazing theorem (1918)

For physical systems (close enough)), every *conservation principle* is associated with a *symmetry*.

- Energy -> invariance in time
- Linear momentum -> invariance in displacement
- Angular momentum -> invariance in direction.

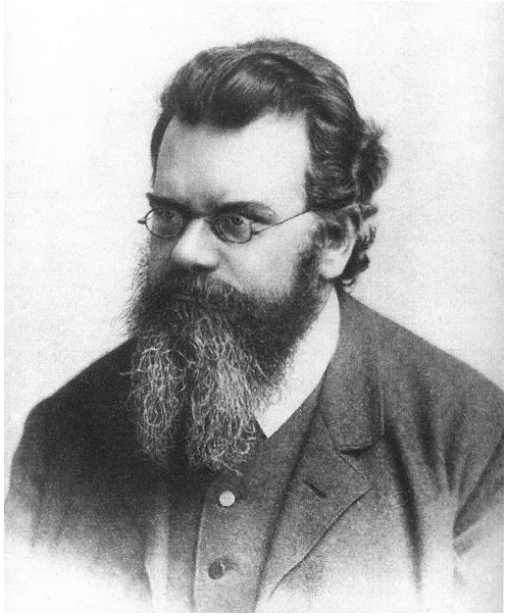
Scale-independence is a *symmetry*, so can we find *a Conservation Principle for discrete systems ?*

I needed a context INdependent measure. Enter Hartley-Shannon …

Hartley-Shannon information (1928,1948) is basically the log of the number of ways of arranging tokens *irrespective of what they mean.*

But, what happens when we study systems which conserve information ?

More big guns …
and my first effort (2014)



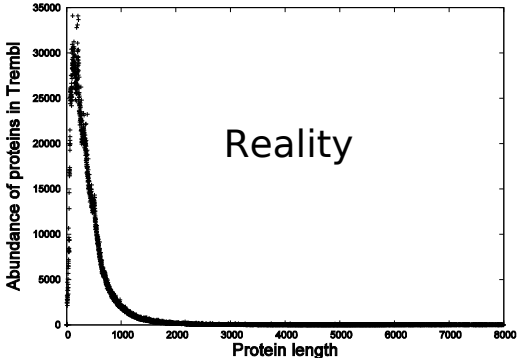
Conserve total tokens →

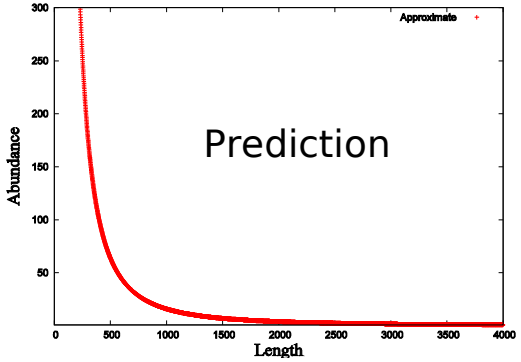
Conserve total Information →

Boltzmann's magical statistical mechanics machine

↓

Linear distributions



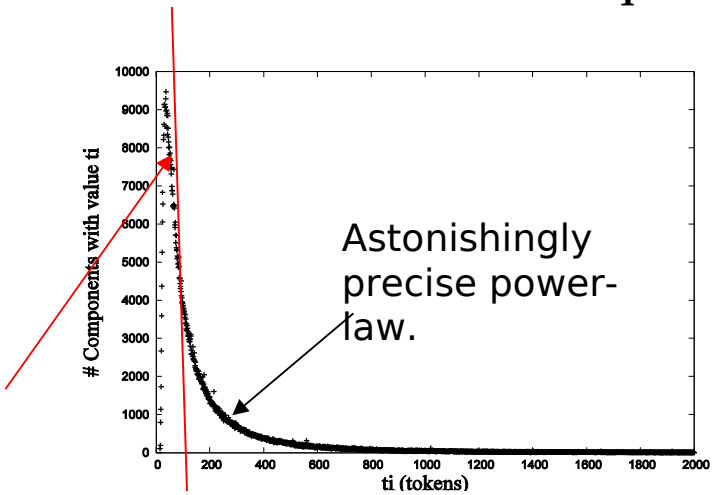


The distribution of lengths: some fundamental results

…

- (2014, TSE) So, using statistical mechanics, I had finally managed to prove using *Conservation of Hartley-Shannon information* that the power-law in long components is inevitable *for any software system*. So far so good, but look at the linear frequency distribution …

Frequency of occurrence



100 million lines of code

What's going on for small components ?

Length of component in tokens

The distribution of lengths: some fundamental results

…

- (2017, ?) Finally, after a Eureka moment with a box of chocolates, I finally managed to prove that the length distribution for any software system *for large components*

$$a_i \sim t_i^{-1/\beta}$$

(2014)

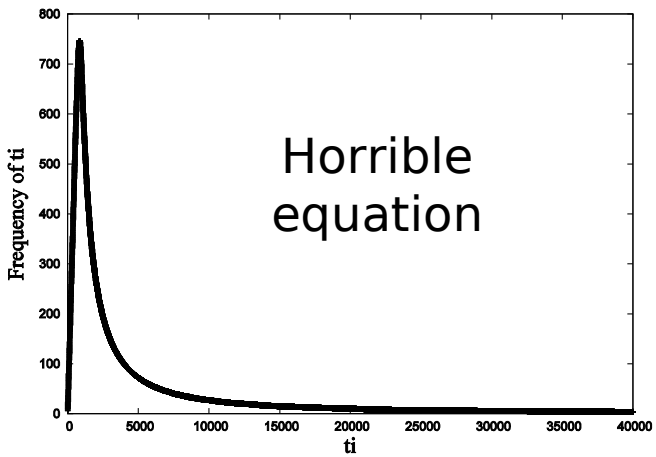
for all scales becomes …

(2016)

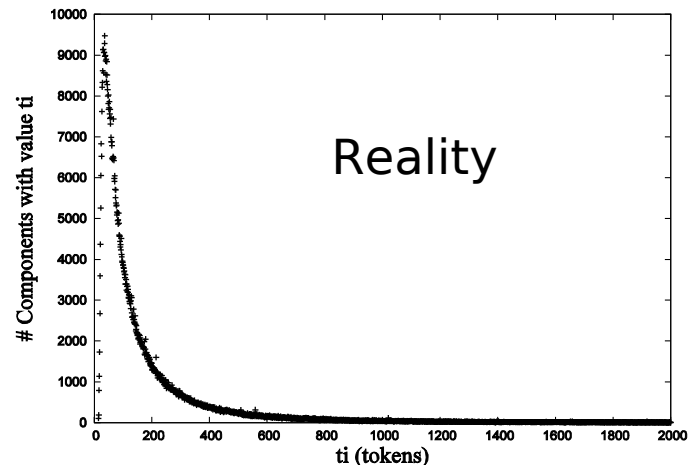
$$a_i = A.(exp(-\frac{1}{2t_i}))^{1/\beta}.(1 - a_i/t_i).(exp(\beta\frac{a_i(1 - a_i/t_i + 1/2t_i)}{t_i(1 - a_i/t_i)}))^{1/\beta}.t_i^{-1/\beta}$$

The distribution of lengths: some fundamental results

- and we're there ...



Predicted length distribution

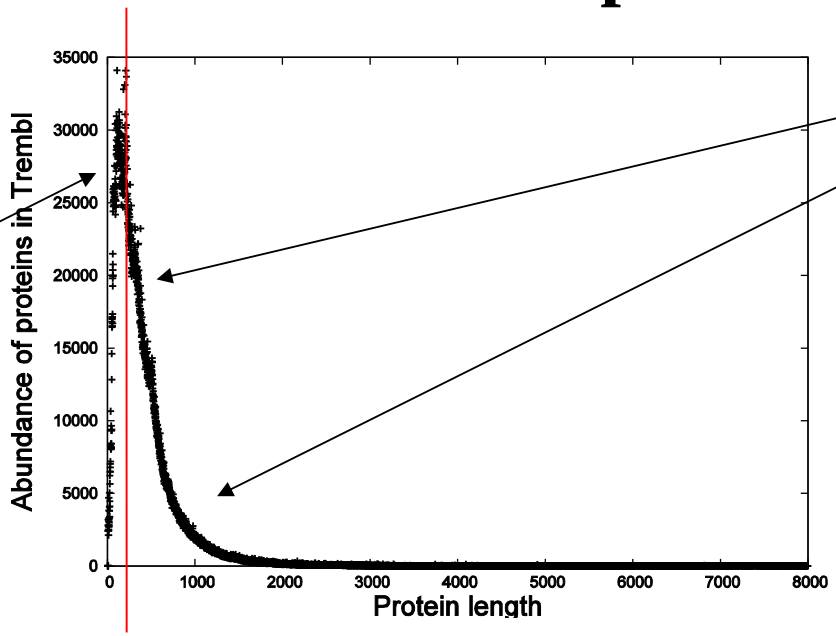


Observed length distribution of 100 MSLOC

This distribution is sharply unimodal with linear slopes and power-law thereafter exactly as observed. It doesn't matter what the system does, what language it is in or who wrote it, human volition does not appear to play a part, Conservation of Information is enough.

The distribution of lengths: some fundamental results

- If it troubles you that human volition has no apparent influence on software length distributions, Natural Selection has no influence on **protein** lengths either.



Almost perfect power-law

Weird pointy bit identical to software

13,532,084 proteins built from 5,392,041,307 amino acids in the TrEMBL database 15-07.

Implications of Conservation of Information

- We have no effective control over component sizes in software systems.
  - All systems will have big components with about the same frequency because power-laws don't die away very quickly. It is not a sign of poor quality, its inevitable. (Th, Ex)
- # Defects in any system α T ln(A).
  - This is consistent with defects being *random* and leads to the prediction *that most components will show no defects* (Th, Ex)
    - Inferring defect models from code properties is almost certainly futile.
    - Studying apparently zero defect components tells you nothing about the ones with defects and vice versa.
    - (On the plus side, this favours layering and more compact system representations.)
- The average size of software components in any system is about the same, (proteins too !) (Th, Ex)

Implications for systems containing substantial software

- The fabric of source code probably doesn't contain any useful patterns other than flaws in language use as revealed by inspection. It's just symbols.
- As software grows, defects will grow super-linearly with size given the same development environment
- BUT, since software size is growing exponentially with time, so will defects grow exponentially with time
- *To control this, greater emphasis will be needed on traditional system virtues such as architecture, redundancy, resilience, fault containment and so on. Good luck !*

References

My writing site:-

http://www.leshatton.org/

lesh@oakcomp.co.uk

Bibliography

Hartley R.V. (1928) “Transmission of Information”, Bell Systems Journal, 7(3), p. 535-563

Hatton L. (2007) “Power-law distributions of component size in general software systems“, IEEE Transactions on Software Engineering, 35(4), p. 566-572

Hatton L. (2014) “Conservation of Information: Software’s Hidden Clockwork ?”, IEEE Transactions on Software Engineering, 40(5), p.450-460

Hatton L. and Warr G (2015) “Protein Structure and Evolution: Are they Constrained Globally by a Principle Derived from Information Theory” PlOS ONE doi:10.1371 /journal.pone.0125663

Hatton L. and Warr G (2017) “Proteins and Computer Programs; Inextricably linked by Information Theory”, submitted to PRSB.

Hatton L., Spinellis D. and van Genuchten M (2017) “The long-term growth rate of evolving software: Empirical results and implications.” *Journal of Software: Evolution and Process*, 2017. To appear. (doi:10.1002/smr.1847)

Hopkins T. and Hatton L. (2008) “Exploring defect correlations in a major Fortran numerical library”, http://leshatton.org/NAG01_01-08.html

Noether E. (1918) “Invariante Variationsprobleme”, Nachr. v..d. Ges. d. Wiss zu Gottingen, p. 235-257

Shannon C.E. (1948) “A Mathematical Theory of Communication”, Bell Systems Journal, 10.1002/j.1538-7305.1948.tb01338.x