IEEE Software BLOG

Sunday, September 12, 2021

Information Loss Leads to Robustness

W.B. Langdon, J. Petke, D. Clark

CREST, Department of Computer Science, University College, London

It is tacitly accepted that real software is far from perfect. That everyday computer systems deliver real economic advantage to their users even though they contain both documented and undocumented bugs. Often computer scientists are embarrassed by this imperfection and teach students that programs should be free from errors. Yet software quality managers know the fastest way to bankrupt their company is to prevent it shipping code until all its errors have been corrected. We turn to information theory to explain why software is resilient to both source code and run time errors and often yields a usable answer despite its flaws [1].

During the second world war Claude Shannon of Bell Labs, New Jersey, formalised the notion of information, defining the amount of information using entropy. The information content of a message defines the minimum number of bits needed to code it (e.g. in order to send it via a telephony link).

Pretty much every operation in a digital computer is irreversible. Meaning that its inputs cannot be determined from its output. From an information theory point of view, every irreversible operation loses information. Figure 1 shows computer operations as information funnels, with more information going into their wide mouths than leaves via their output.

Figure 1 shows two numbers being added together (addition is irreversible). To make the example more concrete, we assume the two inputs are independent single digits (1–9, using Benford's law), and therefore their sum lies in the range 2–18. The red bar graphs show the probabilities of the two inputs and of the output. We also show their entropies. Entropy is defined on probability distributions. The more evenly spread the distribution, the more information it contains and the higher the entropy. The output (2–18) being more uniformly spread, has a higher entropy (3.69 bits) than either input, but it is still less than the entropy of the combined inputs together. That is, in this case, the addition information funnel has lost 2.06 bits of information.

Not only is this information loss inevitable, but it is cumulative. That is, once information is lost, it cannot be restored.

Secondly although pretty much everything, bar simple assignment statements, losses information, some operations loose more than others. For example, testing if a value is greater than a threshold outputs a single bit, containing (at most) 1 bit of information, even if the variable being tested could contain 32 bits of information. Similarly a common technique to increase resilience to errors is to use wrappers. E.g. to prevent run-time exceptions by forcing a module's input to lie in an expected range. Thus the module may have a float input, but in reality expects values to be three digit integers. Hence the developer may design a wrapper to return a default value for negative numbers and numbers greater than 999.0 and round other inputs to the nearest integer. Assuming it is sometimes necessary

IEEE Computer Society

IEEE COMPUTER SOCIETY

IEEE Software

IEEE Software

Blog Archive

▼ 2021 (1)
▼ September (1)
Information Loss Leads to Robustness ...
► 2020 (5)
► 2019 (24)
► 2018 (10)
► 2017 (30)
► 2016 (45)
► 2015 (7)

Associate Editors

- Jeffrey Carver (Practitioners' digest)
- Dario Di Nucci (Testing)
- Niko Mäkitalo (Microservices/Software Architecture)
- Sofia Ouhbi (Requirements Engineering and Software Sustainability)
- Varun Gupta (Global developments)
- Jinghui Cheng (Human Aspects)
- Muneera Bano (User Centric/Human Aspects)
- Ronald Jabangwe (Software Engineering Process Models)
- Mehdi Mirakhorli (Design/ Architecture and Requirements)
- Brittany Johnson (Issue and SE Radio Summary)
- Sarah Nadi (Software release and configuration management)
- Stefano Zacchiroli (Open source software systems)
- Federica Sarro (Mobile applications and systems)
- Sridhar Chimalakonda (Software Quality and Software Reuse)
- Danilo Pianini (Pervasive computing)
- Karim Ali (Programming Languages)
- Mei Nagappan (Practitioner perspectives)
- Xabier Larrucea (Practitioner perspectives)

(e.g. the input is sometimes 5.01), such a robustness increasing wrapper will destroy information.

Figure 1: An information funnel. All computer operations lose information. Even something as fundamental as addition outputs less information than it took in. Example (see red bar charts): addition of two independent numbers (drawn using Benford's Law from 1 to 9, each contain 2.88 bits of information, total 5.75 bits). The result, 2 to 18, contains only 3.69 bits.

Figure 2 shows a large nested function composed of floating point arithmetic operations (+, -, × and ÷). Each function acts as a two input information funnel and loses information. The red subtree at the base of Figure 2 is a bug and the coloured chain shows the execution of nested functions which call the bug. The whole function is executed on 48 test cases and the size of the coloured nodes represents the number of tests when the error and the correct code are not identical at that point in the execution. The colour of these nodes represents the average difference. In this example, the error is visible on all the tests at the function immediately after it. However the number of test cases where the evaluation is not identical relentlessly falls as we move up the chain of coloured nodes towards the outer most function. Indeed, in this example, it falls to zero before we reach the end of the expression. Meaning none of the test cases are able to detect the error.

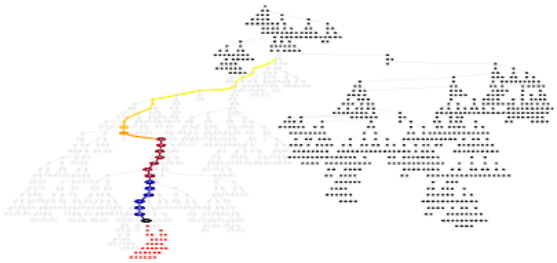

Figure 2: Impact of error (red subtree) dissipates the further away we measure its change on test values. Size of functional nodes shows number of disrupted test cases. The colour shows, on a log scale, the average difference in evaluation on the remaining disrupted nodes. Brightest yellow shows smallest non-zero difference (RMS 3.1 10^{-10}). Eventually this large change has no impact at all [2] [3].

Obviously the loss of information and details of error hiding will depend on the details of the expression and test suite. Nonetheless in functional parts of the code, the monotonic fall in the effectiveness of testing with distance from the error is always true and holds regardless of whether the error is caused by a bug in the source code or a runtime induced fault. Preliminary results hint that the effectiveness of test suites increases only slowly with the number of tests (in proportion to log of the number of tests) and suggests that independent tests are more effective at uncovering faults.

In traditional imperative code the data flow need not follow the execution flow. In particular, the effect of an error may be stored in a variable which is used later. Nonetheless when it is used (computed on) the information it contains about the error may be (partially) lost, particularly if the impact of

the error passes through a long chain of operations.

The information funnel view is entirely consistent with Voas' PIE (propagation, infection, and execution) framework [4] for explaining software errors. Voas says to have any observable effect the error must be executed, that execution must make a difference, i.e. infect the state of the computation, and crucially that state change must propagate to the program's output. What we see is where the change of state passes through a chain of operations (each of which loses information) information about the error may disappear on one or more test cases. That is, information theory backs up the intuition that deeply nested errors will be more difficult to find using testing but also they may have no impact on many test cases. It also justifies the use of unit testing, as not only may the bug be easier to locate, but also, as the code is less heavily nested, bugs may manifest themselves more readily.

The fact that software is not fragile [5] is only tacitly recognised, is perhaps partially explained by the large number of software engineering phenomena which express it using a range of terminology from different research silos:

- In mutation testing the problem of equivalent mutants is wide spread. And yet, an equivalent mutant is simply a code change which can not be separated from the original code by testing.
- Schulte et al. [6] described software mutational robustness when they showed in examples of production C and assembler code that more than 30% of random code changes made no difference under testing.
- Similarly Harrand et al. [7] found neutral program variants when they mutated Java code.
- Members of the same Stockholm team [8] use the term correctness attraction to describe the imperturbability of ordinary software to injection of minimal changes. That is, the Java program gives the same output (assumed to be correct) even though a value set by executed code has been changed by a small amount at run time.
- We [1] suggest the phrase disruption propagation failure as a generalisation of failed error propagation [9], to recognise that the change in program state need not be an error or traditional source code bug and instead she includes all manner of errors, mutations and run-time perturbations.
- Coincidental correctness and fault masking can be placed in Voas' PIE framework in that an error is said to have occurred but that it did not manifest (e.g. due to not infecting the state in the first place or failing to propagate).

In future we will see software systems of increasing complexity. Such systems have long past the point of being comprehensible by any one person and indeed any team of people. Any yet globally we depend on the reliability of software.

In a world addicted to software, managing its quality will become ever more important. Information theory and entropy loss analysis offers a unified view of bugs, faults, transient errors, hardening code to resist tampering and attack, defect resilience and can offer insights into software verification and validation (V&V), particularly testing.

You might also enjoy reading

- Wikipedia article on information theory and Shannon entropy.
- Correctness Attraction: A Study of Stability of Software Behavior Under Runtime Perturbation, 12 November 2018. Martin Monperrus and his co-authors discuss results on 10 Java programs where injection of minimal run time changes made no difference in most (68%) cases.

- Bit-Rot: Computer Software Degrades over Time. IEEE Software blog, 11 March 2020.
- Genetic Improvement. How search is being used to improve existing programs. IEEE Software blog, 2 February 2016.
- Automated bug fixing. An interview with Westley Weimer and Martin Monperrus (published in Ubiquity, 2015, March, pp 1–11).
- Automated bug fixing in Facebook. Mark Harman and his coauthors describe genetic improvement at a global scale [10].
- Benford's law of anomalous numbers. Wikipedia article on why leading digits are often small.

References

[1] Justyna Petke, William B. Langdon, and David Clark. Software robustness: A survey, a theory, and some prospects. In Paris Avgeriou and Dongmei Zhang, editors, ESEC/FSE 2021, Ideas, Visions and Reflections, Athens, Greece, 23-28 August 2021. ACM.

[2] William B. Langdon. Fitness first. In Wolfgang Banzhaf, Leonardo Trujillo, Stephan Winkler, and Bill Worzel, editors, Genetic Programming Theory and Practice XVIII, East Lansing, MI, USA, 19-21 May 2021. Springer. Forthcoming.

[3] William B. Langdon, Justyna Petke, and David Clark. Dissipative polynomials. In Nadarajen Veerapen, Katherine Malan, Arnaud Liefooghe, Sebastien Verel, and Gabriela Ochoa, editors, 5th Workshop on Landscape-Aware Heuristic Search, GECCO 2021 Companion, Internet, 10-14 July 2021. ACM.

[4] Jeffrey M. Voas. PIE: a dynamic failure-based technique. IEEE Transactions on Software Engineering, 18(8):717–727, Aug 1992.

[5] William B. Langdon and Justyna Petke. Software is not fragile. In Pierre Parrend, Paul Bourgine, and Pierre Collet, editors, Complex Systems Digital Campus E-conference, CS-DC'15, Proceedings in Complexity, pages 203–211. Springer, September 30-October 1 2015. Invited talk.

[6] Eric Schulte, Zachary P. Fry, Ethan Fast, Westley Weimer, and Stephanie Forrest. Software mutational robustness. Genetic Programming and Evolvable Machines, 15(3):281–312, September 2014.

[7] Nicolas Harrand, Simon Allier, Marcelino Rodriguez-Cancio, Martin Monperrus, and Benoit Baudry. A journey among Java neutral program variants. Genetic Programming and Evolvable Machines, 20(4):531–580, December 2019.

[8] Benjamin Danglot, Philippe Preux, Benoit Baudry, and Martin Monperrus. Correctness attraction: a study of stability of software behavior under runtime perturbation. Empirical Software Engineering, 23(4):2086–2119, 1 August 2018.

[9] Gunel Jahangirova, David Clark, Mark Harman, and Paolo Tonella. An empirical study on failed error propagation in Java programs with real faults. ArXiv, 24 Nov 2020. abs/2011.10787.

[10] John Ahlgren, Maria Eugenia Berezin, Kinga Bojarczuk, Elena Dulskyte, Inna Dvortsova, Johann George, Natalija Gucevska, Mark Harman, Ralf Laemmel, Erik Meijer, Silvia Sapora, and Justin Spahr-Summers. WES: Agent-based user interaction simulation on real infrastructure. In Shin Yoo, Justyna Petke, Westley Weimer, and Bobby R. Bruce, editors, GI @ ICSE 2020, pages 276–284, internet, 3 July 2020. ACM. Invited Keynote.

Posted by Federica Sarro at 8:02 AM

No comments:

Post a Comment

Enter your comment...

Comment as: Google Account

Publish Preview

Home Older Post

Subscribe to: Post Comments (Atom)