Information Transformation: An Underpinning Theory for Software Engineering

David Clark*, Robert Feldt†, Simon Poulding† and Shin Yoo*
* Department of Computer Science
University College London, Gower Street, London
†Department of Software Engineering, Blekinge Institute of Technology, Sweden
Email: david.clark@ucl.ac.uk, robert.feldt@bth.se, simon.poulding@bth.se, shin.yoo@ucl.ac.uk

***Abstract*—Software engineering lacks underpinning scientific theories both for the software it produces and the processes by which it does so. We propose that an approach based on information theory can provide such a theory, or rather many theories. We envision that such a benefit will be realised primarily through research based on the quantification of information involved and a mathematical study of the limiting laws that arise. However, we also argue that less formal but more qualitative uses for information theory will be useful.**

The main argument in support of our vision is based on the fact that both a program and an engineering process to develop such a program are fundamentally processes that transform information. To illustrate our argument we focus on software testing and develop an initial theory in which a test suite is input/output adequate if it achieves the channel capacity of the program as measured by the mutual information between its inputs and its outputs. We outline a number of problems, metrics and concrete strategies for improving software engineering, based on information theoretical analyses. We find it likely that similar analyses and subsequent future research to detail them would be generally fruitful for software engineering.

I. Introduction

We believe that information theory is more useful to software engineering than hitherto has been realised. Not only is information theory a useful way to think about and to solve problems in our discipline but it is possibly the only contender for an underlying theory that could produce laws, explanations and predictions, all connected by a common perspective.

Information Theory offers the following three things to software engineering:

1) A theory of communication and encoding, due to Shannon, that allows us to view software, specifications, verification and validation in a unified way as a collection of information transformation channels (information transformers) [1].
2) A theory of the information content of objects, due to Kolmogorov and others, that allows us to assess the complexity of objects and to compare them at a highly useful level of abstraction [2]. This is the very notion of the information in an object which Brooks asserted would be so useful to software engineering [3].
3) A non-mathematical theory of information useful for describing the process and human aspects of software engineering—such as how individuals and teams best pool their resources to efficiently develop software systems [4].

Feynman famously categorised lines of enquiry in physics into Babylonian and Greek, i.e. focused on the phenomena or on the underlying order [5]. This paper is a very much a Babylonian paper but with a Greek promise. By tackling different problems in software engineering in a piecemeal way, we believe that there is a very real possibility that the approaches could eventually be systematised into an underlying theory.

In what follows, after some brief definitions and intuitions, we offer an example of the power of information theory to explain and predict by considering the input/output channel of a program and its potential uses in software testing. We then list a set of open problems in software engineering that have an obvious information theory connection. These range through testing, metrics, software evolution and system development.

II. What is Information Theory?

It is hard to do justice to information theory within this brief paper. Here, we merely describe the underlying concepts.

Shannon information, often called entropy, is a statistic of a probability distribution and measures the amount of variation in the probability distribution on a random variable. It is a negative quantity that measures ignorance – uncertainty about the outcome of the experiment of sampling the random variable. For a random variable X with sample values x with probability $p(x)$, it is defined as

$$\mathcal{H}(X) = -\sum_{x \in X} p(x) \log_2 p(x)$$

Since its inception as a mathematical theory of communication, information theory has broadened and found applications in many other areas, e.g. natural language processing, pattern detection, and statistical inference. Information theory answers fundamental questions about the limits of signal processing, transmission, and data transformation [1].

Kolmogorov complexity, also called algorithmic information, measures the amount of information in a string (or object) as the length of the shortest program that can produce that object from empty input. It historically post-dates but conceptually precedes Shannon information and has a strong connection to it. While it is not computable, Cilibrasi, Vitanyi

978-1-4799-1934-5/15 $31.00 © 2015 IEEE
DOI 10.1109/ICSE.2015.202

and others have demonstrated that compression of strings can be used successfully as upper approximations in practice [2].

III. Towards a Theory for Software Testing

To illustrate the potency of information theory in software engineering, we present a sketch of a theoretical framework for software testing based on information theory. We begin by looking at software as information transformation.

1) The input/output channel: Every program, viewed statically, has an input/output channel. The input to the channel, $\mathcal{I}$, is the random variable in all possible initial states; the output, $\mathcal{O}$, is the random variable in all possible final states. In the case of stateless programs, the initial state is defined by the values of arguments to the program and the random variable $\mathcal{I}$ has a joint probability distribution over these arguments. For stateful programs, $\mathcal{I}$ additionally encompasses the current state of the program. Non-termination, although not observable, can be considered for completeness of the model as one possible final state. Following this view, we immediately have a number of interesting quantities that have significant implications in software testing.

2) Mutual information: Mutual information between inputs and outputs was used by Shannon to model transmission of information. $\mathcal{M}(X;Y)$ is the quantity of information from X that arrives at (or is shared with) Y. Assume that the program we wish to test is deterministic, i.e. without any probabilistic schedulers or key generation so that for a program, P, $P(\mathcal{I}) = \mathcal{O}$. Then the random variable in outputs is entirely dependent on the random variable in inputs and all the variation in $\mathcal{O}$ comes from $\mathcal{I}$. This fact allows us to simplify the mutual information between inputs and outputs in a nice way: because of this dependency $\mathcal{H}(\mathcal{I},\mathcal{O}) = \mathcal{H}(\mathcal{I})$ so, using the chain rule of entropy [1], we have $\mathcal{H}(\mathcal{O}|\mathcal{I}) = \mathcal{H}(\mathcal{I},\mathcal{O}) - \mathcal{H}(\mathcal{I}) = 0$. Then $\mathcal{M}(\mathcal{I};\mathcal{O}) = \mathcal{H}(\mathcal{O}) - \mathcal{H}(\mathcal{O}|\mathcal{I}) = \mathcal{H}(\mathcal{O})$.

3) The channel capacity: Shannon defined the channel capacity as the largest possible amount of mutual information, considered over all possible input distributions. In our deterministic program setting this also reduces to something very simple. The general, formal definition is

$$\max_{\sigma \in \Sigma_{\mathcal{I}}} \mathcal{M}(\mathcal{I};\mathcal{O})$$

where $\Sigma_{\mathcal{I}}$ is the set of all possible distributions for the random variable $\mathcal{I}$. In our setting this simplifies to

$$\max_{\sigma \in \Sigma_{\mathcal{I}}} \mathcal{H}(\mathcal{O}) = \log_2(|\mathcal{O}|)$$

i.e. no higher than the highest possible value of $\mathcal{H}(\mathcal{O})$, achieved when the probability distribution of $\mathcal{O}$ is uniform. So the channel capacity only depends on the number of outputs. To return to the question of testing, we might consider it reasonable that any test suite reflects an input distribution that achieves the channel capacity of the program. Otherwise, how can it be adequate? If it doesn't achieve this there must be some part of the behaviour of the program that has not been tested. There are, in general, for any program, many input distributions that achieve the channel capacity. We can go further than this in what follows and show what these test suites look like. For the moment we have done enough to put forward our first suggestion:

A test suite is I/O channel adequate if it achieves the channel capacity of the input/output channel of program.

In practice, given that the channel capacity is the log of the number of outputs, this suggestion can be realised by finding a test suite that finds all the outputs of the program. Evidence for the usefulness of this suggestion (law) comes from recent work on test output diversity [6], [7]. The most recent paper demonstrates that "output uniqueness", i.e. incrementally choosing a test suite by increasing the number of unique outputs that the suite produces, has high Pearson correlation to white box testing with either statement, branch and path coverage. In addition, the output diversity test suite selection technique increases the number of real faults found by 47% in comparison to selecting the test suite using any of the three coverage metrics. The original paper demonstrates that the approach works in a black box testing context as well [6]. We argue that the power of the approach rests on the fact that it is a method for constructing a test suite that achieves the channel capacity of the program.

This is in line with the more general suggestion by Feldt et al. [8] that test case diversity measured with an approximation of the Kolmogorov complexity of test case execution traces could help create better test suites. By varying the information used to characterise the test cases, different types of diversity between them could be exploited, such as output diversity. This is an extension of the results on output uniqueness since the level of diversity can be directly measured, with a provably universal diversity metric based on Kolmogorov complexity [2]. Instead of only counting unique outputs, information theory suggests that we can utilise the diversity between outputs to further improve testing. Subsequent investigation of these information theoretical notions of uniqueness and diversity in software testing might prove a very fertile ground for important, future results.

4) Conditional entropy: The conditional entropy of the input/output channel in the program is given by $\mathcal{H}(\mathcal{I}|\mathcal{O})$. Using the arguments we used above in the case of a deterministic program we have, by the chain rule, $\mathcal{H}(\mathcal{I}|\mathcal{O}) = \mathcal{H}(\mathcal{I},\mathcal{O}) - \mathcal{H}(\mathcal{O}) = \mathcal{H}(\mathcal{I}) - \mathcal{H}(\mathcal{O})$. This quantity can be interpreted as the information that is lost through executing the program (over all possible runs). There is an equivalent to channel capacity for this quantity: the maximal conditional entropy of program over all possible input distributions [9] but we suspect that, for conditional entropy, the uniform distribution on inputs (or any other distribution obtained via the Maximum Entropy Principle) is more useful. This quantity is, in a sense, a measure of how difficult it is to test a program. The more information the input/output channel loses, the less useful an oracle observing outputs becomes and the more necessary a greater testing effort becomes, whether this is manifested as additional oracles on internal states, the provision of higher order mutation testing, or the use of a search-based method for finding a test suite that has properties that go beyond meeting the information transformer adequacy condition. Evidence for the usefulness of this metric is provided in the work of Androutsopoulos et al., in which they show that there is a high correlation between the probability of coincidental correctness for oracles observing output and the conditional entropy of certain internal channels [10]. The work of Masri et al. has demonstrated that coincidental correctness is prevalent in programs to varying degrees and that it exercises a reducing effect on the usefulness of syntactic coverage criteria [11]. This leads us to put forward another suggestion:

The conditional entropy of the input/output channel of a program (under the uniform distribution on inputs) is a measure of the difficulty in testing a program.

We can normalise and consider $\frac{\mathcal{H}(\mathcal{I}|\mathcal{O})}{\mathcal{H}(\mathcal{I})} \in [0,1]$ as a testability measure. The most difficult programs will have measure 1 and the easiest measure 0.

5) Achieving the channel capacity of the program's I/O channel: What do we mean by the phrase "achieves the channel capacity" precisely? Now that we have discussed conditional entropy we can illustrate this. The partition property of entropy [1] allows us to reformulate conditional entropy in a useful way. Suppose that a deterministic program P has an input/output semantics expressed by the function f. Let $f^{-1}o$ be the random variable in the inverse image of $o \in \mathcal{O}$. The inverse images of elements of $\mathcal{O}$ partition I. For each $o \in \mathcal{O}$, $\sigma_{\mathcal{O}}(o) = \sum_{i \in f^{-1}o} \sigma_{\mathcal{I}}(i)$ so $\sigma_{\mathcal{O}}$ is the probability distribution for the random variable in the partitions induced by the inverse images. These inverse images partition the input space. By the partition property

$$\mathcal{H}(\mathcal{I}) = \mathcal{H}(\mathcal{O}) + \sum_{o \in \mathcal{O}} p(o)\ \mathcal{H}(f^{-1}o)$$

so we can write the conditional entropy of the input/output channel as the sum of the entropies of the inverse images of the outputs:

$$\mathcal{H}(\mathcal{I}|\mathcal{O}) = \mathcal{H}(\mathcal{I}) - \mathcal{H}(\mathcal{O}) = \sum_{o \in \mathcal{O}} p(o)\ \mathcal{H}(f^{-1}o)$$

A graphical interpretation of this definition of conditional entropy for the input/output channel is in figure 1.

From the diagram it is clear that, to achieve the channel capacity with a test suite, we can select one test input from each inverse image. This corresponds to an input distribution which is uniform on the test suite but 0 for every other input and this distribution will achieve the channel capacity. The closer the conditional entropy of the program is to $\mathcal{H}(\mathcal{I})$ the more choices we have but the harder it is to make those choices so as to avoid coincidental correctness as the testability of the program decreases.

Of course the question remains as to how easy it is to construct a test suite that achieves an adequacy criterion based on information theory, such as achieving the channel capacity or a uniform distribution over the outputs. This is an example of where taking an information theoretical perspective can motivate improvements to existing software engineering techniques. Recent work in search-based software testing has demonstrated that by using metaheuristic search algorithms, it is feasible to derive a probability distribution over the input domain of a program, i.e. a specific distribution for the random variable $\mathcal{I}$, that induces a desired target distribution over, for example, the execution of structural elements in the program [12]. In the existing implementation of this technique, the test engineer is able to choose this target distribution based on their own experience or best practice—in other words, by applying the more informal type of information discussed in the introduction.

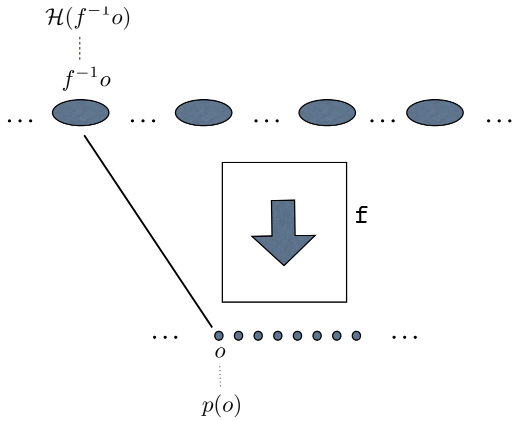

Fig. 1. Information destroyed by program execution.

The results above suggest an alternative objective for this search-based approach that takes a more formal information theoretical view. The new objective would be to derive a distribution over the input, $\mathcal{I}$, that induces a target distribution over the output, $\mathcal{O}$, of a program that optimises an information theoretical metric such as the channel capacity. Test inputs may then be sampled in order to construct the test suite. In practice, the best results—for example, the best fault-detecting ability—might be realised by combining both types of information: the informal knowledge of the human engineer and a metric based on the more formal view of information.

Consider: we have looked at one channel, but we can also view the program as a collection of transformers. In the theory of secure software flows this has become an important idea [13] and can model many properties of software systems including differential privacy, side-channel attacks, information hiding, integrity, anonymity, and cache leaks. We believe that a theory of software testing has a natural fit with information theory and in time many testing phenomena can be modelled and better understood using information theory.

IV. Future Software Engineering Research

We list here some problems in software engineering that have connections to information theory. Some have existing publications, others do not.

The Input/Output channel in software testing: Consider two programs, ID and ML. ID is a complex implementation of the identity function. ML is a complex machine learning classifier with two outputs, *yes* and *no*. The functional semantics of ID means that, with zero conditional entropy, I/O channel adequacy may not be achievable. For ML it is easy to achieve but not very helpful. What other channels can assist in devising a test suite? How far short of I/O channel adequacy can be tolerated?

Coupling and cohesion metrics: Allen et al. used information theory in the context of counting edges on graphs to define coupling and cohesion for software components [14]. But coupling between components can be measured in terms of the strength of the information flow between them. A high channel capacity indicates strong coupling. This fits closely with the use of coupling in safety critical systems.

Clustering test cases: The use of Kolmogorov complexity ideas to cluster test cases produces a 'natural' measure of diversity. This idea has barely been explored as yet [8].

Modelling properties of software: Clark et al. have inspired much research into measuring properties of software using information theory [15]. Bucketing continuous quantities allows time, power consumption and other non-functional properties to be interpreted in terms of information [16].

The Kolmogorov complexity of the software under test: Gates et al. have argued that there is something to be learned about the difficulty of testing software from the Kolmogorov complexity of the source code [17].

Using soft information for informal modelling: An information theoretical 'point of view' can be useful also for less mature areas where formal modelling is either hard or as yet unexplored. As an example, in industrial projects, the applicability of a software engineering method often depends on the type and number of information sources and amount of information the method needs. Methods that require a large amount of information from many different sources and types of information are less likely to be economically feasible [18].

Software maintenance and evolution: Arbuckle has demonstrated the practicality of using Kolmogorov complexity in understanding software evolution [19].

Path feasibility: It seems likely that channel capacity can be used to conservatively estimate path feasibility and the entropy of the input space for a path be used to estimate *how* feasible a feasible path is.

Guiding search using information: It has been understood for decades that information is useful for guiding search, whether it is being employed in the context of Big Data, machine learning, Monte Carlo simulations or even software testing. Yoo et al. have used it in this way to optimise test input ordering for regression testing [20].

We consider the problems above to give only a 'tip of an iceberg' sense of the important, future, software engineering research based on and inspired by information theory. Our vision is of a more wide-spread research programme where information theory can provide theoretical underpinnings for software engineering in general and an information theoretical approach can be broadly useful for software engineering both in theory and in practice. Both the software produced and the engineering processes to produce it are information transformation processes and will obey fundamental limits that can be studied mathematically. We urge more researchers to join in this exploration.

References

[1] T. M. Cover and J. A. Thomas, *Elements of information theory, 2nd ed.* John Wiley & Sons, 2012.

[2] P. M. Vitányi, F. J. Balbach, R. L. Cilibrasi, and M. Li, "Normalized information distance," in *Information theory and statistical learning*. Springer, 2009, pp. 45–82.

[3] F. P. Brooks, Jr., "Three great challenges for half-century-old computer science," *Journal of the ACM*, vol. 50, no. 1, pp. 25–26, 2003.

[4] K. Krippendorff, *Information theory: structural models for qualitative data*. Sage, 1986, vol. 62.

[5] L. Mlodinow, *Feynman's Rainbow: A Search for Beauty in Physics and in Life*. Warner Books, 2003.

[6] N. Alshawan and M. Harman, "Augmenting test suites' effectiveness by increasing output diversity," in *Proceedings of the 34th International Conference on Software Engineering*, 2012, pp. 1345–1348.

[7] N. Alshahwan and M. Harman, "Coverage and fault detection of the output-uniqueness test selection criteria," in *Proceedings of the International Symposium on Software Testing and Analysis*, 2014, pp. 181–192.

[8] R. Feldt, R. Torkar, T. Gorschek, and W. Afzal, "Searching for cognitively diverse tests: Towards universal test diversity metrics," in *Proceedings of International Conference on Software Testing, Verification and Validation Workshops*, 2008, pp. 178–186.

[9] D. Clark and R. Hierons, "Squeeziness: an information theoretic metric for avoiding fault masking," *Information Processing Letters*, vol. 12, no. 8–9, pp. 335–340, 2012.

[10] K. Androutsopoulos, D. Clark, H. Dan, R. M. Hierons, and M. Harman, "An analysis of the relationship between conditional entropy and failed error propagation in software testing," in *Proceedings of the International Conference on Software Engineering*, 2014, pp. 573–583.

[11] W. Masri and R. A. Assi, "Prevalence of coincidental correctness and mitigation of its impact on fault localization," *ACM Transactions on Software Engineering Methodology*, vol. 23, no. 1, pp. 8:1–8:28, 2014.

[12] R. Feldt and S. Poulding, "Finding test data with specific properties via metaheuristic search," in *Proceedings of the 24th International Symposium on Software Reliability Engineering*, 2013, pp. 350–359.

[13] D. Clark, S. Hunt, and P. Malacaria, "A static analysis for quantifying information flow in a simple imperative language," *Journal of Computer Security*, vol. 15, no. 3, pp. 321–371, 2007.

[14] E. B. Allen, T. M. Khoshgoftaar, and Y. Chen, "Measuring coupling and cohesion of software modules: An information-theory approach." in *IEEE METRICS*, 2001, pp. 124–134.

[15] C. Mu and D. Clark, "A tool: quantitative analyser for programs," in *Proceedings of the 8th International Conference on Quantitative Evaluation of Systems*, 2011.

[16] B. Köpf and D. A. Basin, "Automatically deriving information-theoretic bounds for adaptive side-channel attacks," *Journal of Computer Security*, vol. 19, no. 1, pp. 1–31, 2011.

[17] A. Q. Gates, V. Kreinovich, and L. Longpre, "Kolmogorov complexity justifies software engineering heuristics," *Bulletin of the European Association for Theoretical Computer Science (EATCS)*, pp. 150–154, 1998.

[18] R. Feldt, "Do system test cases grow old?" in *Proceedings of the IEEE International Conference on Software Testing, Verification, and Validation*, 2014, pp. 343–352.

[19] T. Arbuckle, "Studying software evolution using artefacts' shared information content," *Science of Computer Programming*, vol. 76, no. 12, pp. 1078–1097, 2011.

[20] S. Yoo, M. Harman, and D. Clark, "Fault localization prioritization: Comparing information-theoretic and coverage-based approaches," *ACM Transactions on Software Engineering Methodology*, vol. 22, no. 3, pp. 19:1–19:29, 2013.