Squeeziness: An Information Theoretic Measure for Avoiding Fault Masking

David Clark[a], Robert M. Hierons[b]

[a]Department of Computer Science, University College London, Gower Street, London, WC1E 6BT United Kingdom
[b]School of Information Systems, and Computing Mathematics, Brunel University, Uxbridge, Middlesex, UB8 3PH, United Kingdom

Abstract

Fault masking can reduce the effectiveness of a test suite. We propose an information theoretic measure, Squeeziness, as the theoretical basis for avoiding fault masking. We begin by explaining fault masking and the relationship between collisions and fault masking. We then define Squeeziness and demonstrate by experiment that there is a strong correlation between Squeeziness and the likelihood of collisions. We conclude with comments on how Squeeziness could be the foundation for generating test suites that minimise the likelihood of fault masking.

Keywords: Software engineering; formal methods; software testing; fault masking

1. Introduction

This paper explores relationships between fault masking and information theoretic measures. What follows assumes a knowledge of basic information theory although a reader who simply takes the mathematical definitions on trust should find it rewarding.

In recent years information theory has been used to quantify information flow (QIF) in software. The context has been to ensure correct behaviour with respect to security properties [5, 11, 3, 8]. This paper explores fault masking in software testing and its dependence on what we call *collisions*. Since execution paths can be used to generate tests in a coverage approach, we argue that ranking paths using a measure for the effect of collisions can assist in avoiding fault masking. We define an information theoretic measure, Squeeziness. We compare Squeeziness with the previously defined Domain to Range Ratio metric (DRR) and show that they induce different orderings. We show that there is a strong statistically significant correlation between the likelihood of collisions and Squeeziness, which seems to be invariant as domain size increases, unlike the weaker correlation between collision likelihood and DRR. Finally, we discuss possibilities for estimating Squeeziness.

Email addresses: david.clark@ucl.ac.uk (David Clark), rob.hierons@brunel.ac.uk (Robert M. Hierons)

2. Fault Masking and Collisions

In software testing we execute the *implementation under test (IUT)* with an input (*test case*) and observe the output. Typically we cannot use all inputs and choose a set of test cases, a *test suite*. Often we choose a test suite that satisfies a particular test criterion. In this paper we are interested in white-box testing, in which a test criterion refers to the code. Most white-box test criteria require that elements of the code are executed. Such criteria are called *coverage criteria*. For example, 100% statement coverage requires that each statement of the program is executed (covered) in testing.

One challenge is to produce test cases that are likely to lead to failures: observations not consistent with the specification. A fault is a mistake in the IUT that can lead to a failure and to simplify the discussion we assume that a fault is associated with a particular statement s of the IUT. In order for a fault at s to lead to a failure, the execution of the IUT with test case t must lead to a path π that includes s; t *exercises* the fault. Testing is complicated by the fact that a test case can exercise s but not lead to a failure. Even if the program state after s is not that expected, later statements can lead to the correct output being produced, a phenomenon often called *fault masking*. The fault might lead to a failure if executed in a different path or with different input. It has been observed, in the PIE framework [16], that for a failure to be caused by faulty statement s it is necessary for s to be Executed, this must Infect the state (create an incorrect program state) and the infection must Propagate to the output or final state.

We illustrate the idea with an example. Consider the intended program P_1

```
x = x + 2;
if (x > 0) x = x % 4; else x = x;
```

and the faulty program P_2

```
x = 3 * x;
if (x > 0) x = x % 4; else x = x;
```

For an input that executes the first statement of P_2 (Execution), the program state immediately after the execution may not be that expected (Infection). An input for P_2 which leads to `x` being bigger than 0 after `x = 3 * x;` will not always discover this (no Propagation) but one which leads to `x` being less than or equal to 0 will (Propagation). For example, consider test cases $t_1 : x = 3$ and $t_2 : x = -5$. P_1 leads us to expect outputs of 1 for t_1 and -3 for t_2. P_2 produces 1 and -15 respectively; t_1 does not uncover the fault, whereas t_2 does. The problem is that, in both programs, t_1 has an execution path that executes `x = x % 4;`. For P_1, `x` is 5 after `x = x + 2;` and for P_2, `x` is 9 after `x = 3 * x;`, but executing `x = x % 4;` produces 1 in both cases. This is not the case for t_2 as both programs take the false branch and Propagation is guaranteed.

Let us restate what is necessary for fault masking. The faulty program (we assume a single fault for simplicity) must create a semantic condition, an “incorrect” state, immediately after the fault. The fault free program would create a “correct” state for the same input. The subsequent behaviour of the program maps both of these states to the same output. We say that the subsequent behaviour of the program causes the "correct" and "incorrect" states to *collide* in the output. If there were no possible collisions then fault masking would not be possible.

This is the key intuition in this paper. Some execution paths are more effective at revealing faults than others and an optimal path has fewest collisions. This is important when generating test suites on the basis of coverage: given a choice of paths that cover a structural element we can minimise the possibility of fault masking by choosing a covering path that minimises the potential for collisions.

3. Evidence for Fault Masking

This section briefly reviews two pieces of evidence of fault masking through collisions in real code. We start by discussing mutation testing, where a program p is changed (mutated) at one point to produce mutant m. The effectiveness of test suite T is judged by determining the proportion of mutants in a set M killed by T. Under strong mutation testing, $m \in M$ is killed by $t \in T$ if m and p produce a different output on t. In weak mutation testing t kills m if the execution of p and m with t leads to different states after the point mutated. Thus, if t kills m under weak mutation testing but not under strong mutation testing then fault masking has occurred through collision. Experiments, using 11 programs, found that a test suite T that killed all mutants under weak mutation also killed 85% or more under strong mutation but not all [12, 13]. For mutant m, killed under weak mutation, to not be killed under strong mutation we required fault masking for *all* $t \in T$ that weakly kill m. Thus, these experiment provide evidence regarding a lower bound on fault masking due to collisions.

In regression testing we have an original program p, a new program p' produced by changing p, and we test p' to check that new faults have not been introduced. Traditionally, we use test cases that exercise the statements changed. However, the MATRIX method [2] chooses test cases that exercise a changed statement and are likely to propagate differences in the state to output (i.e. to not suffer from collisions). A (small) value d is chosen and the method uses dependence analysis, for chains with depth at most d starting at the change point, and symbolic evaluation. Experiments suggest that this works [2]. For example, in experiments with nine versions of two programs, if the tester simply chose test cases that executed the change points then in all but one case fewer than 23% of the test suites revealed a change in behaviour[1]. The results were much better using the MATRIX method and $d = 3$. For one program over 90% of the test suites revealed a change in behaviour for all four versions. The results for the other program also showed a significant improvement. Recent work has used more efficient approaches to analysing propagation conditions [14] with empirical studies again showing a significant improvement over approaches that simply aim to test the change point [14]. While these studies did not aim to assess fault masking, the methods aim to lead to test cases in which there is less potential for collisions and the results provide additional evidence for the significance of fault masking through collisions.

[1] All results were averaged over 50 runs.

4. Avoiding Collisions

In this section we design a metric that can be used to compare paths, ranking them by measuring the effect collisions have on the transfer of information along the path. We can treat execution paths as if they are programs by observing that any path, π, can be made into a program by introducing a special termination state, ABORT, for branches not on π. We call these *path programs*. For example the execution path in program P_1 above that follows the true branch would become the following and the effect of ABORT statements can easily be discounted when computing the metric.

```
x = x + 2;
if (x > 0) x = x % 4; else ABORT;
```

We make some reasonable assumptions about the nature of the programs we are analysing, namely that they are deterministic and can be given a functional semantics. By this we mean that outputs are a function of inputs; the programs terminate and there is no internal non-determinism. This covers a very large and important class of programs. Using these assumptions a program can simply be treated as a function.

$$\text{execution path} \longleftrightarrow \text{program} \longleftrightarrow \text{function}$$

Our measure will be the loss of information from the input space caused by the execution path. Information is only lost by inputs leading to the same output (collisions). The smaller the loss the weaker the effect of collisions. No loss of information means no collisions. In what follows we assume familiarity with basic information theory.

We consider total, onto functions with fixed, finite, discrete domains. That is, a function f is equipped with a fixed input domain, I, and an output range, O, so that $f : I \to O$ and (overloading f) $O = fI$ and $I = f^{-1}O$.

Definition 1. *Function Collision. A pair of inputs to function f, $t, t' \in I$,* collide *if $t \neq t' \wedge ft = ft'$.*

We take a probabilistic view of the behaviour of f. We overload I and O to also represent random variables equipped with probability distributions, σ_I and σ_O respectively. Shannon [15] measured the information content, or entropy, of a random variable X with probability distribution p as follows.

Definition 2. *Entropy of a random variable.*

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

Since random variable O is completely determined by f's action on I all the information in O stems from I [7] so $\mathcal{H}(I) - \mathcal{H}(O)$ is the amount of information destroyed by f. We call this quantity *Squeeziness*. If the function is one to one there are no collisions and the Squeeziness of the function is 0 since $\sigma_I = \sigma_O$ and $\mathcal{H}(I) = \mathcal{H}(O)$.

Definition 3. *Squeeziness. The* Squeeziness *of total function $f : I \to O$, $Sq(f)$, is defined as the loss of information after applying f to I*

$$Sq(f) = \mathcal{H}(I) - \mathcal{H}(O)$$

The partition property of entropy [9] allows us to reformulate Squeeziness in a more useful way. Let $f^{-1}o$ be the random variable in the inverse image of $o \in O$. The inverse images of elements of O partition I. For each $o \in O$, $\sigma_O(o) = \sum_{i \in f^{-1}o} \sigma_I(i)$ so σ_O is the probability distribution for the random variable in the partitions induced by the inverse images. The inverse images partition I. By the partition property

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

hence

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

The RHS is a weighted sum of terms and $\mathcal{H}(f^{-1}o)$ is the amount of information contained in (the random variable in) the set of elements mapped to a single output.

It is mostly likely that, at the point of testing software for the first time, developers will have little idea of the probability distribution. A possible strategy is the principle of maximum entropy which produces a counting metric.

Maximum Entropy Principle: The principle of *maximum entropy* states that if a probability distribution is not known the best strategy is to assume a distribution consistent with the known constraints that maximises the entropy [1]. If there are no constraints on the distribution, maximum entropy is attained when the distribution is uniform. If the input distribution is uniform the entropy in the normalised probability distribution on the inverse image of an output o is $log_2|f^{-1}o|$ where $|f^{-1}o|$ is the number of elements in the inverse image. The weight of a single element of σ_I is $1/|I|$ and the weight of each inverse image is $|f^{-1}o|/|I|$. So the squeeziness of f is

$$Sq(f) = \frac{1}{|I|} \sum_{o \in O} |f^{-1}o|\ log_2(|f^{-1}o|)$$

Another possibility is to work with a worst case scenario.

Maximum Loss of Information: When the probability distribution on inputs is not known it is possible to calculate maximum loss of information via a characterisation of the class of distributions that achieve the upper bound, $\bigsqcup_{\sigma_I} Sq(f)$. An input distribution that achieves maximum Squeeziness is one which is uniformly distributed on one of the maximally sized inverse image partitions and zero everywhere else. Let $o' \in O$ be such that $\forall o \in O, |f^{-1}o'| \geq |f^{-1}o|$. Define for all $i \in I$, $\sigma_{\mathcal{I}}(i) = 1/|f^{-1}o'|$ if $i \in |f^{-1}o'|$ and $\sigma_{\mathcal{I}}(i) = 0$ otherwise. If $\sigma_{\mathcal{I}}$ is the probability distribution associated with I, we have $Sq(f) = log_2|f^{-1}o'|$. For any other probability distribution on inputs, σ_I, we have

$$Sq(f) = \sum_{o \in O} p(o)\ \mathcal{H}(f^{-1}o) \leq log_2|f^{-1}o'|$$

that is, $log_2|f^{-1}o'|$ is an upper bound for information loss. This is a consequence of the following lemma.

Lemma 1. *Let* $\{A_i\}_{1 \leq i \leq n}$, $\{\alpha_i\}_{1 \leq i \leq n}$ *be such that* $\forall i, A_i, \alpha_i \in \mathbb{R}, A_i, \alpha_i \geq 0$*, and* $\forall i, A_1 \geq A_i$*, and* $\sum_{1 \leq i \leq n} \alpha_i \leq 1$*, then* $\sum_{1 \leq i \leq n} \alpha_i A_i \leq A_1$.

It is straightforward to demonstrate this by induction on n. Given the lemma, it is only necessary to observe that, by construction, $\forall o \in O, \mathcal{H}(f^{-1}o) \leq log_2|f^{-1}o'|$.

5. Squeeziness and the Domain to Range Ratio

The only previous metric for fault masking we are aware of is Woodward and Al-Khanjari's Domain to Range Ratio (DRR) [17]. They claim that it is a good candidate for a metric which is useful to *discuss* (our emphasis) concepts such as fault masking and testability. Part of their rationale is the claim that it provides an approximate measure of information loss, by which they mean the proportion of what we call collisions. In what follows we compare DRR with Squeeziness as ranking measures for collisions.

Definition 4. *Domain to Range Ratio*
For a total, onto function $f : I \to O$ define the Domain to Range Ratio*, $DRR(f)$, as $DRR(f) = \frac{|I|}{|O|}$.*

This is not an information theoretic measure and only measures information loss in an informal sense of the term. In what follows we show that Squeeziness is not a refinement of DRR or vice versa. In fact: DRR is a cruder measure than Squeeziness, making fewer distinctions; and they produce inconsistent orderings on functions.

For simplicity we assume the maximal entropy principle. The examples are the simplest for which the observations hold.

Observation 1. *Fix $|I|$ and $|O|$. Let $F = \{f \mid f : I \to O\}$ every $f \in F$ has $DRR(f) = \frac{|I|}{|O|}$ but $\exists f_1, f_2 \in F, Sq(f_1) \neq Sq(f_2)$.*

Suppose $I = \{1, 2, 3, 4, 5, 6\}$ and $O = \{A, B\}$, so every $f \in F$ has $DRR(f) = 6/2 = 3$. Consider $f_1 \in F$ where $f_1 x$ is A if $x = 1$ and B otherwise so $Sq(f_1) = (5.log_2 5 + log_2 1)/6 = 1.935$. Let $f_2 \in F$ where $f_2 x$ is A if $x \leq 3$ and B otherwise so $Sq(f_2) = (2.3.log_2 3)/6 = 1.585$.

Observation 2. *There exist functions, f_1, f_2 such that $DRR(f_1) < DRR(f_2)$ but $Sq(f_1) > Sq(f_2)$.*

Let $I = \{0, \ldots, 15\}$, let $f_1 : I \to O_1$ be defined by $f_1 x$ is: 6 if $x < 7$; and x otherwise. We have $O_1 = f_1 I = \{6, \ldots, 15\}$ so $|O_1| = 10$, $DRR(f_1) = 16/10 = 1.6$ and $Sq(f_1) = (7.log_2 7)/16 = 1.23$. Let $f_2 : I \to O_2$ be defined by: $f_2 x$ is: 7 if $x < 2$; 8 if $2 \leq x < 4$; 9 if $4 \leq x < 6$; 10 if $6 \leq x < 8$; 11 if $8 \leq x < 10$; 12 if $10 \leq x < 12$; 13 if $12 \leq x < 14$; and x otherwise.

$O_2 = f_2 I$ is the set of states in which x takes values in $\{7, \ldots, 15\}$ so $|O_2| = 9$ while $DRR(f_2) = 16/9 = 1.78$ and $Sq(f_2) = (7.2.log_2 2)/16 = 14/16 = 0.875$. Hence the observation holds. In the next section we look at the relationship between Squeeziness, DRR and the likelihood of collisions.

6. Squeeziness and the Likelihood of Collisions

In this section we assume that all input domains have a uniform probability distribution. We show that there is no monotonic relationship between the probability of a pair of inputs to function f being in a collision and the Squeeziness of f but provide evidence of a strong statistical correlation.

Fault masking occurs when the expected state at statement s is σ, the actual state is $\sigma' \neq \sigma$ but the expected output is produced. If f denotes the function defined by starting the program at s, this occurs if and only if there exists $o \in O$ such that $\sigma, \sigma' \in f^{-1}o$; there is a collision and one state is not "correct". A collision is necessary but not sufficient for fault masking as some collisions could be between "correct" states.

Assume $O = \{o_1, \ldots, o_n\}$ and I_i denotes the set $f^{-1}o_i$ with size m_i. Further, let $d = \sum_{i=1}^{n} m_i$ denote the size of the input space. Given a uniform distribution on inputs, the probability of both σ and σ' being in I_i is $p_i = \frac{m_i*(m_i-1)}{d*(d-1)}$, since $\sigma \neq \sigma'$. Thus, the probability of collisions occurring is the sum of the p_i, giving

$$PColl(f) = \sum_{i=1}^{n} \frac{m_i * (m_i - 1)}{d * (d-1)}$$

This can be seen as the probability of collisions, when there is a uniform distribution on inputs. The relationship between $PColl$ and Sq is not in general monotonic.

Observation 3. $\exists f_1 : I_1 \rightarrow O_1, f_2 : I_2 \rightarrow O_2, Sq(f_1) < Sq(f_2)$ *but* $PColl(f_2) < PColl(f_1)$.

Let f_1 take $\{A, B, C\}$ to X and $\{D, E\}$ to Y, while f_2 takes $\{A, B, C\}$ to X and D to Y. Then $PColl(f_1) = 0.4$ and $PColl(f_2) = 0.5$ however $Sq(f_1) = 1.35$ and $Sq(f_2) = 1.1887$.

We now report on simulations that compared $PColl$; Squeeziness (Sq); and Domain to Range Ration (DRR). All three are defined in terms of the sizes of the subdomains ($f^{-1}o$). We therefore produced all three for randomly generated partition sizes.

Simulations operated as follows. We fixed the size d of the input space and maximum subdomain size m. Random integers in $[1.m]$, representing subdomain sizes (the m_i), were generated until the values summed to d, the last value being reduced if the sum exceeded d. The three measures were then computed. This was repeated 100 times for the given d and m. We then computed the Pearson correlation coefficient[2] between $PColl$ and Sq and between $PColl$ and DRR. For each pair of values for d and m we performed this process twice, the results to three significant figures being shown in Table 1; all results were significant at $p = 0.01$[3].

The results show a strong, statistically significant, correlation between Sq and $PColl$, all correlations being above 0.96. There is also a correlation between DRR and $PColl$ but this is noticeably lower; while the correlation is as high as 0.88 in some cases it is as low as 0.58 in others and there appears to be a tendency for this correlation to drop as the domain size increases.

7. Squeeziness in Practice

$PColl$ appears to be just as useful a metric for collision avoidance as Squeeziness. On the other hand they don't produce the same ordering on paths, but since we are

[2] The results for Spearman Rank Correlation were almost identical and are not reported.

[3] Due to space restrictions we only report a few results but we repeated these experiments many more times, varying the parameters, always obtaining similar results.

Table 1: Simulations relating $PColl$ with Sq and DRR

Domain Size	Max Sub.	Corr with Sq	Corr with DRR
10^5	200	0.981	0.849
10^5	200	0.986	0.889
10^5	2000	0.981	0.849
10^5	2000	0.986	0.889
10^6	200	0.971	0.748
10^6	200	0.964	0.686
10^7	200	0.968	0.645
10^7	200	0.975	0.606
10^8	200	0.978	0.584
10^8	200	0.975	0.668

ultimately looking for statistical correlations, this may be of theoretical interest only. Both $PColl$ and Squeeziness are intractable to compute directly. Squeeziness has the advantage of being a QIF measure and enjoying a close connection to an existing body of research and an active community of researchers who are seeking useful approximate methods. Possibilities for adaptation to approximate computation of Squeeziness include Heusser and Malacaria's bounded model checking approach [10], Chatzikokolakis et alia's statistical approach to estimating flow quantity [4], or a syntax based approximation [6]. We briefly sketch how the BCMC model checking approach of [10] can be adapted to estimating Squeeziness.

Suppose we want to cover a particular statement with a test input. Further, we have identified a set of post statement paths and we want to select the least squeezy one to optimise the effectiveness of the test input we will generate. We can use the BCMC model checker to give us very rough information about the *maximum Squeeziness* of each path, sufficient to identify one or more paths as the best option(s) among the set. We would proceed as follows: we set a threshold for the maximum Squeeziness and run the model checker on each path. For each path it tells us whether its maximum Squeeziness is below, equal to, or above our threshold. Having hopefully eliminated some paths we have the options of refining our results by choosing another threshold and running BCMC on the remaining paths, or arbitrarily selecting a remaining path.

The BCMC (Bounded C Model Checker) checks properties of C programs through the use of a driver that encodes the property to be checked. We note that the bounded nature of the model checker refers to the depth of unwinding of loops and is not relevant to path programs, which contain none.

Recall that the maximum Squeeziness of a program is determined by the maximum size of the inverse images over all outputs. We set a threshold, N, for the maximum size of the inverse image for any output from the path program, function Pi, discounting the inverse image of the special termination state, `ABORT`. The driver function tries to produce a counter-example, an output of Pi with an inverse image larger than N. Inputs are modelled by a non-deterministic choice function, `input( )`. To check that the maximal inverse image is N the model checker repeatedly runs the driver function.

On each run the driver function produces inputs $i_1, \ldots, i_{N+1}$ by $N+1$ calls to `input()` and then calls Pi on each input to produce outputs $o_1, \ldots, o_{N+1}$. The driver then uses an `assume` statement after these calls, in this case to assume that the $N+1$ inputs are all different, that the first N outputs are all equal, and that no output is equal to `ABORT`. Then the driver uses an `assert` statement to assert that o_{N+1} is not equal to o_1.

The model checker only considers input sets satisfying the assume statement and rejects other sets. Then it tries to find a counter-example which satisfies the assumption but negates the assertion.

There are three possible outcomes from the model checking: The `assume` clause cannot be satisfied, in which case the Maximum Squeeziness is less than $log_2 N$. The `assume` clause can be satisfied but the negation of the `assert` clause cannot be satisfied in which case the maximum Squeeziness of the path is $log_2 N$. The `assume` clause can be satisfied and the negation of the `assert` clause can be satisfied so the maximum Squeeziness is greater than $log_2 N$.

8. Conclusions

This paper investigated fault masking through collisions and introduced a new information theoretic measure, called Squeeziness, which can be applied to program execution paths. The results of initial experiments show that there is a strong correlation between the likelihood of collisions along a path and Squeeziness.

If we can estimate the likelihood of collisions along a set of paths then we might choose a path that both tests a targeted part of the program and simultaneously minimises the scope for fault masking; that is, in practice we might search for suitable covering paths that minimise Squeeziness. The development of such methods will be an important part of future work.

Basing a future practical approach on an information theoretic measure opens up the potential to exploit work on methods for scalably estimating such measures in the context of secure information flow. We have sketched how one such method might be employed in practice using the BCMC model checker. For software testing we are interested in ranking paths rather than computing Squeeziness estimates safely and it would be interesting to investigate approximation methods that take advantage of our weaker requirements.

Acknowledgements

This work has benefited from discussions with Sebastian Hunt and Laurence Tratt. We are also grateful for the valuable comments from the anonymous reviewers.

References

[1] http://en.wikipedia.org/wiki/Principle_of_maximum_entropy.

[2] T. Apiwattanapong, R. A. Santelices, P. K. Chittimalli, A. Orso, and M. J. Harrold. Matrix: Maintenance-oriented testing requirements identifier and examiner. In *TAIC PART 2006*, pages 137–146. IEEE Comp. Soc., 2006.

[3] M. Boreale, D. Clark, and D. Gorla. A semiring-based trace semantics for processes with applications to information leakage analysis. In *6th IFIP TC 1/WG 2.2 Int. Conf. TCS 2010, Part of WCC2010 Proceedings*. Springer, 2010.

[4] K. Chatzikokolakis, T. Chothia, and A. Guha. Statistical measurement of information leakage. In *TACAS*, pages 390–404, 2010.

[5] K. Chatzikokolakis, C. Palamidessi, and P. Panangaden. Anonymity protocols as noisy channels. *Information and Computation*, 206(2-4):378–401, 2008.

[6] D. Clark, S. Hunt, and P. Malacaria. Quantitative analysis of the leakage of confidential data. In A. D. Pierro and H. Wiklicky, editors, *Electr. Notes Theor. Comput. Sci.*, volume 59. Elsevier, 2002.

[7] D. Clark, S. Hunt, and P. Malacaria. A static analysis for quantifying information flow in a simple imperative language. *Journal of Computer Security*, 15(3):321 – 372, 2007.

[8] M. Clarkson and F. Schneider. Quantification of integrity. In *Proc. IEEE Comp. Security Foundations Symposium*, 2010.

[9] T. M. Cover and J. A. Thomas. *Elements of Information Theory*. Wiley Interscience, 1991.

[10] J. Heusser and P. Malacaria. Quantifying information leaks in software. In *26th Ann. Comput. Security App. Conf. (ACSAC)*, pages 261–269, 2010.

[11] C. Mu and D. Clark. An interval-based abstraction for quantifying information flow. *Electr. Notes Theor. Comput. Sci.*, 253(3):119–141, 2009.

[12] A. J. Offutt and S. D. Lee. How strong is weak mutation? In *Symposium on Testing, Analysis, and Verification*, pages 200–213, 1991.

[13] A. J. Offutt and S. D. Lee. An empirical evaluation of weak mutation. *IEEE Trans. on Software Engineering*, 20(5):337–344, 1994.

[14] R. A. Santelices and M. J. Harrold. Applying aggressive propagation-based strategies for testing changes. In *IEEE 4th Int. Conf. on Software Testing, Verification and Validation (ICST 2011)*, pages 11–20. IEEE Computer Society, 2011.

[15] C. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27:379–423 and 623–656, July and October 1948. Available on-line at http://cm.bell-labs.com/cm/ms/what/shannonday/paper.html.

[16] J. Voas and G. McGraw. *Software fault injection: inoculating programs against errors*. John Wiley and Sons, 1998.

[17] M. R. Woodward and Z. A. Al-Khanjari. Testability, fault size and the domain-to-range ratio: An eternal triangle. In *ISSTA*, pages 168–172, 2000.