Fault Localization Prioritization: Comparing Information Theoretic and Coverage Based Approaches

Shin Yoo, Mark Harman and David Clark, University College London

Test case prioritization techniques seek to maximise early fault detection. Fault localization seeks to use test cases already executed to help find the fault location. There is a natural interplay between the two techniques; once a fault is detected, we often switch focus to fault fixing, for which localization may be a first step. In this paper we introduce the Fault Localization Prioritization (FLP) problem, which combines prioritization and localization. We evaluate three techniques: a novel FLP technique based on information theory, FLINT (Fault Localization using Information Theory), that we introduce in this paper, a standard Test Case Prioritization (TCP) technique and a 'test similarity technique' used in previous work. Our evaluation uses five different releases of four software systems. The results indicate that FLP and TCP can statistically significantly reduce fault localization costs for 73% and 76% of cases respectively and that FLINT significantly outperforms similarity-based localization techniques in 52% of the cases considered in the study.

Categories and Subject Descriptors: D.2.5.7 [**Software Engineering**]: Testing and Debugging—*Debugging Aids*

General Terms: Algorithm

Additional Key Words and Phrases: Test Case Prioritization, Fault Localization, Information Theory

ACM Reference Format:
ACM Trans. Softw. Eng. Methodol. V, N, Article A (January YYYY), 29 pages.
DOI = 10.1145/0000000.0000000 http://doi.acm.org/10.1145/0000000.0000000

1. INTRODUCTION

Fault localization can be used to prioritize the statements of a program according to the likelihood that each contains a fault that causes a known failure [Jones et al. 2002; Renieres and Reiss 2003]. Regression test prioritization is used to order statements according to early achievement of some testing goal such as coverage [Elbaum et al. 2000; Li et al. 2007; Rothermel et al. 2001; Yoo and Harman 2012].

Regression testing and fault localization are naturally complementary: we test to see if change has introduced a fault and, if we discover that it has, we switch from regression testing to fault localization as a first step towards fault fixing. Having fixed the fault we switch back to regression testing and so the cycle continues. This 'test–find–fix' cycle is familiar to many software engineers.

Complete execution of all test cases in a regression test suite can be slow; execution times of up to seven weeks have been reported in the literature [Rothermel et al. 2001]. For larger test suites, it may not be realistic to expect the engineer to wait for all test cases to be executed before they can start work on fault fixing. Also, it may not be worth executing all test cases, once the first failure has been detected. In many scenarios, as soon as the first failure is detected, the engineer will want to switch from testing to fixing.

Even if the engineer does not wish to make this switch at the first failure, there may be, at some subsequent test execution, a failure for which the engineer will want to switch from testing to fixing: for example, if the engineer expects that the failure is important, or requires immediate action. Also, if the engineer expects that the fault which causes the currently detected failure will also cause many cascaded downstream failures, then it would be wasteful to have to wait for regression testing to continue, flagging up all such cascaded errors in the process.

There are many situations where this switch between testing and fixing may occur, particularly when many faults have been inserted. In such a situation, regression testing will be

most likely an iterative process: executing test cases until the first fault is found and then switching to fixing this first fault. Once the first fault is fixed, the engineer will switch back to regression testing until the next fault is found.

Of course, whenever the engineer switches from testing to fixing, the prioritization algorithm that provides automated support must also switch from test case prioritization for fault revelation to test case prioritization for fault localization. Since the two goals are different, the order in which the remaining test cases should be executed may also be different.

One approach would be to switch immediately to fault localization, using the set of test cases executed up to the point at which the failure was observed. In some situations, this may be all that is required. However, there may be too few test cases for effective fault localization, particularly if the first failure is observed early in the process. In this situation testing will need to continue for a while to provide a sufficiently large pool of test cases for effective localization. This raises the question that lies at the heart of this paper:

> "Having found a fault, what is the best order in which to rank the remaining test cases to maximize early localization of the fault?"

We call this problem the 'Fault Localization Prioritization' (FLP) problem. Existing approaches to fault localization need to be adapted to prioritize, not just statements (according to their suspiciousness), but also to prioritize the test cases according to how well they contribute to early elevation of faulty statement suspiciousness.

We believe that FLP, like many other testing problems, is a naturally information theoretic problem. That is, we seek to choose the next test case to be executed to be the test case that gives the most information about where the fault might lie. Therefore, we also introduce a novel information theoretic approach called FLINT: Fault Localization using INformation Theory. FLINT uses Shannon's Information Theory [Shannon 1948] to define an entropy reduction measure for test cases, such that the next test case in an ordering is the one that maximally reduces fault locality entropy. This makes Shannon's mathematical theory of information a natural choice; one that we formalize, implement and evaluate in this paper.

For the FLP problem, we order statements of the program according to their suspiciousness, as is common with other work on fault localization. However, we also order test cases according to their ability to increase the early identification of the suspiciousness of the faulty statement. In our experiments, we chose to use the widely studied Tarantula suspiciousness metric [Jones et al. 2002; Jones and Harrold 2005]. However, the choice of suspiciousness measurement is a parameter to any FLP problem, so one can easily incorporate metrics from other fault localization work [Cleve and Zeller 2005; Liblit et al. 2005; Renieres and Reiss 2003].

We evaluate our FLINT approach to FLP against three other alternatives. We use a random ordering as a baseline to see whether any of our orderings has any value compared to an arbitrary ordering. We also use two 'more intelligent' alternatives against which to compare our FLINT approach.

The first of these more intelligent approaches is to simply continue to order test cases according to their coverage, as would be done by the continuation of the TCP approach. It ought to make sense to continue to try to cover more of the code, particularly should little coverage have been achieved when the engineer switches from testing to fixing. Surely this would help with localization? This observation motives a comparison of 'standard TCP' with FLINT.

The second alternative against which we compare is based on the similarity between tests [Artzi et al. 2010]. The intuition behind this approach is that similar tests that have different pass/fail outcomes help to improve the effectiveness of fault localization, as the similarity would maximize the chance that the fault is correlated with the relatively small difference between tests. This suggests that we might consider prioritizing the remaining test cases

according to their similarity to the test case that revealed the fault. We call this the 'similarity ordering'.

We report the results of a set of empirical studies of the three techniques' suitability for FLP on five different releases of four software systems for which test suites and fault information are available. We start by evaluating the effect of FLP on the suspiciousness of the faulty statement. In order to qualify as a suitable method for FLP, a technique must order test cases so that the faulty statement will attain higher suspiciousness early in the prioritization.

Raising suspiciousness is only a necessary, but not a sufficient property for an FLP method to be useful. There is an interaction between the suspiciousness metric and the FLP technique and so the degree to which early suspiciousness elevation ensures efficient fault localization also needs to be studied empirically. It could be that the faulty statement has an early elevation of suspiciousness, as desired, but the effect is masked by co-lateral early suspiciousness elevation of fault-free statements. This can be assessed through an empirical study of the traditional fault localization cost of the induced prioritizations of statements according to suspiciousness. In our empirical study we report results for the elevation of suspiciousness and the traditional fault localization cost.

Reducing cost (for test cases with known coverage) is also insufficient, on its own, for FLP to be useful. In order to be useful a candidate FLP technique must reduce traditional fault localization cost, but it also needs to cope with an 'information impoverished environment'. That is, for those test cases already executed at the point we switch from testing to fixing, we shall have reliable coverage. However, for those that remain to be executed, we shall have to make do with coverage information available from the previous version of the software.

This information is only partly accurate, because changes to the previous version of the software that create the current version may also change test case coverage results recorded for the previous version. We therefore evaluate the performance of the two prioritization approaches for the 'test–find–fix' cycle in which coverage information is only available about the previous release. We also explore the correlation between good performance of the FLP techniques on the previous release of the software with their performance at the next release.

We may summarize the primary contributions of the paper as follows:

(1) The paper introduces a problem that we term the 'Fault Localization Prioritization (FLP) problem', which we believe captures the 'test–find–fix' cycle that many engineers adopt in practice.
(2) The paper introduces FLINT, an information theoretic approach to FLP.
Shannon entropy provides a quantitative theoretical foundation on which to build a new approach to fault localization in which both statements and test cases are prioritized. Statements are ordered by suspiciousness, while test cases are ordered by the degree to which they reduce the entropy inherent in fault localization.
(3) The paper presents the results of an empirical study that demonstrates that both our FLP approaches outperform arbitrary ordering, making them sensible FLP candidate techniques. Our study also reveals that information theoretic ordering outperforms the two more intelligent orderings: coverage-based TCP and similarity ordering.
(4) The paper presents results from a further empirical study that provides evidence to support the claim that information theoretic ordering copes well with imperfect, partial and noisy information. This makes the approach applicable after code changes have degraded existing test coverage information. Traditional test case prioritization based on coverage also performs well in this information impoverished environment, though less well than FLINT.
(5) The paper presents empirical evidence to support the claim that the engineer can use past performance as a guide to determine which FLP technique to use.

The rest of the paper is organized as follows: Section 2 explains the underlying concepts of the suspiciousness metric and test case prioritization as well as presenting a motivating example for FLINT. Section 3 presents the theoretical foundation for the FLINT approach and sets out the research questions. Section 4 describes the algorithms for FLINT. Section 5 discusses the experimental setup of the empirical study, the results of which are analyzed in Section 6. Section 7 discusses threats to validity. Section 8 presents related work and Section 9 concludes.

2. BACKGROUND

2.1. Test Case Prioritization

Test case prioritization concerns the ordering of test cases for early maximization of some desirable properties, such as the rate of fault detection [Yoo and Harman 2012]. It seeks to find the optimal permutation of the sequence of test cases with respect to a property. It does not involve selection of the test cases, and assumes that the set of test cases may be executed in the order of the permutation it produces, but that testing may be terminated at some arbitrary point during the testing process. More formally, the prioritization problem is defined as follows:

DEFINITION 1. ***The Test Case Prioritization Problem***

Given: *a test suite, T, the set of all permutations of T, PT, and a function from PT to real numbers, $f : PT \rightarrow \mathbb{R}$.*

Problem: *to find $T' \in PT$ such that $(\forall T'')(T'' \in PT)(T'' \neq T')[f(T') \geq f(T'')]$.*

The ideal choice of the priority function, f, would be one that would result in an ordering of tests with the maximum rate of fault detection. Since this information is unavailable before the testing is finished, various surrogates including code coverage [Elbaum et al. 2000; Rothermel et al. 2001], test execution history [Kim and Porter 2002] and expert knowledge [Tonella et al. 2006; Yoo et al. 2009] have been studied.

2.2. Fault Localization Metrics

Fault location techniques aim to reduce the cost of debugging by automating the process of searching for the location of the fault in the program. A widely studied approach to fault localization is to assign to each structural element in the program a *suspiciousness* value that corresponds to the relative likelihood of the element containing the fault [Liblit et al. 2005; Jones and Harrold 2005; Abreu et al. 2007]. For example, the Tarantula suspiciousness metric [Jones and Harrold 2005] for a statement s in a program is calculated as follows:

$$\text{Tarantula metric } \tau(s) = \frac{\frac{fail(s)}{totalfail}}{\frac{pass(s)}{totalpass} + \frac{fail(s)}{totalfail}} \quad (1)$$

In Equation 1, $fail(s)$ and $pass(s)$ represent the number of times the statement s was executed by failing and passing tests, respectively, whereas $totalfail$ and $totalpass$ represent the number of failing and passing tests.

The highest possible value for τ is 1 and the lowest is 0. If a statement s is executed by all tests, at least one of which fails, it gets assigned $\tau = 0.5$. A statement s' gets assigned $\tau = 1$ if and only if all failing tests and none of the passing tests executes s'. However, it is possible

that some statements other than the faulty statement get a higher τ value than s'. Suppose that s' causes a failure only for certain input values, whereas an error handling routine s'' is executed whenever s' fails: s'' will get assigned $\tau = 1$, whereas s' might get assigned τ less than 1 depending on the test input.

Table I. Motivating Example: coverage-based prioritization would execute t_3 after the first failure (t_2), resulting in sub-optimal suspiciousness metric values. However, if t_4 is executed after the first failure, the faulty s_7 will get assigned the optimal suspiciousness value.

Structural Elements	Test t_1	Test t_2	Test t_3	Tarantula Metric(τ)	Test t_4	Tarantula Metric(τ)
s_1	•		•	0.00		0.00
s_2	•		•	0.00		0.00
s_3	•		•	0.00		0.00
s_4	•			0.00		0.00
s_5	•		•	0.00		0.00
s_6		•		1.00	•	1.00
s_7 (faulty)		•	•	0.67	•	1.00
s_8		•		1.00	•	1.00
s_9	•	•		0.67	•	0.50
Result	P	F	P	-	F	-

2.3. Prioritizing for Fault Localization

Existing work on fault localization treated the calculation of suspiciousness metrics as a *post hoc* procedure. That is, fault localization was attempted only after the entire test suite was executed. However, this contradicts the assumptions behind test case prioritization, i.e. that there may not be enough time to execute the entire test suite.

Suppose that the tester encounters a failing test while executing a test suite prioritized for maximum fault detection capability. We argue that, after the initial failure, different tests contribute different amounts of information regarding the location of the faulty structural element. It follows that, after the initial failure, the tester should choose a test that would provide the most information as the next test case whenever possible, followed by other tests in the order of a decreasing amount of information provided.

Consider the motivating example in Table I. Test t_1 to t_4 is prioritized based on the structural coverage following the *additional* approach with resets [Elbaum et al. 2000]. The dots (•) show the coverage relation: for example, structural element s_1 is covered by test t_1 and t_3. The prioritized test suite detects the first fault with t_2, which covers the faulty element s_7. Suppose that there is only time to execute one additional test: the next two columns show what the final suspiciousness metric would look like if t_3 or t_4 is chosen to be executed. According to the coverage-based prioritization, the next test is t_3 and the faulty element will get assigned the suspiciousness of 0.67. However, this is misleading as s_6 and s_8 are assigned with higher suspiciousness values. On the other hand, if t_4 is executed, the faulty element is assigned the suspiciousness of 1.0 along with other elements, which would be a more precise result. This shows that the choice of the next test case can affect the accuracy of the suspiciousness metric if the testing is terminated at an arbitrary point.

In reality, it is impossible to predict whether a test would pass or fail. Therefore, it is also impossible to make the ideal choice for fault location. However, it is possible to formulate a probabilistic approximation that can be used as a surrogate, much in the same way as test case prioritization techniques use structural coverage as a surrogate for the measure of fault detection capability. We turn to Information Theory for this probabilistic approximation.

3. FAULT LOCALITY & ENTROPY

This section presents the formulation of fault localization as an entropy reduction process and outlines the research questions.

3.1. Problem Formulation

3.1.1. Assumptions & Basic Notations. Let $S = \{s_1, \ldots, s_m\}$ be the set of structural elements in the System Under Test (SUT); let $T = \{t_1, \ldots, t_n\}$ be the test suite with n tests. A single element in S contains the fault. Let $C : T \rightarrow 2^S$ be the mapping from tests to executed structural elements, i.e.:

$$C(t) = \{s \in S | t \text{ covers } s \text{ when executed}\}$$

Finally, let $F(t)$ be a boolean statement that says test t has failed. Similarly, let $B(s)$ be a boolean statement that says structural element s contains a fault and $\mathbf{P}$ be the mapping from events to probabilities. We will make the following assumptions for our approximation:

(1) The results from all tests in T are deterministic, i.e. $\forall t \in T : F(t) \vee \neg F(t)$.
(2) The suspiciousness metric is *competent* and does reflect the likelihood of faultiness, i.e. $\mathbf{P}(B(s)) \sim \tau(s)$.
(3) The mapping between tests and structural elements, C, is known (we relax this assumption in Section 6.2).

The first assumption underpins most existing work for software testing. The second assumption states that the suspiciousness metric we use will work as expected, i.e. higher suspiciousness of $s_i \in S$ means a higher chance of s_i being faulty. This assumption is supported by empirical evidence in the existing work [Jones and Harrold 2005; Abreu et al. 2007], the findings of which the paper replicates. It is important to acknowledge that our approach will only *amplify* the suspiciousness metric that is used for the parametric τ: the better the suspiciousness metric is at localising faults, the better our approach will be at maximising early fault localization. Regarding the third assumption, the empirical study in the paper considers both the case when it holds and the case when it does not. The assumption about the knowledge of coverage information may not be realistic in certain cases. However, when the exact information C is not known, it is possible to approximate C with information from the testing of the previous version, similar to the way in which test case prioritization techniques use the coverage information from the previous version.

Now we describe the situation in which the ith test fails during testing. Without losing generality, let T_{i-1} be the set of the first $i-1$ tests, $\{t_1, \ldots, t_{i-1}\}$, that have passed; let t_i be the first failing test. For the sake of brevity, let TP_i and TF_i be the total number of passing/failing tests, respectively, after executing the tests in T_i. Similarly, let $CP_i(s_j)$ and $CF_i(s_j)$ be the number of times s_j has been covered by passing/failing tests, respectively, after executing the tests in T_i.

3.1.2. Entropy of Fault Locality. Given a set of tests at least one of which fails, it is possible to calculate the suspiciousness of each statement based on the tests executed up to and including the first test that has failed. Given a set of tests $T_i = T_{i-1} \cup \{t_i\}$, let $\tau(s|T_i)$ denote the suspiciousness of s calculated using the tests in T_i. Based on the Assumption 2, the approximated probability that statement s_j contains the fault, based on the information observed with T_i, is calculated as the normalized suspiciousness metric for s_j:

$$\mathbf{P}_{T_i}(B(s_j)) = \frac{\tau(s_j|T_i)}{\sum_{j=1}^{m} \tau(s_j|T_i)} \tag{2}$$

The normalization is required to convert the set of suspiciousness metric values into a probability distribution. Using this, Shannon's entropy regarding the locality of the fault can now be defined as follows:

$$\mathbf{H}_{T_i}(S) = -\sum_{j=1}^{m} \mathbf{P}_{T_i}(B(s_j)) \cdot \log \mathbf{P}_{T_i}(B(s_j)) \tag{3}$$

Ideally, fault localization is complete when we add sufficient tests so as to arrive at some T_N with $\mathbf{H}_{T_N} = 0$: the probability $\mathbf{P}(B(s'))$ will be 1 for the faulty statement s' and 0 for the remaining statements. Our aim is to minimize $\mathbf{H}_{T_N}$ as much as possible. This means not only increasing the suspiciousness of the faulty statement, but also decreasing the suspiciousness of the non-faulty statements.

When locating a fault that can be detected deterministically (Assumption 1), it should be noted that the entropy of fault locality, calculated following Equation 3, is identical for the same set of tests, i.e. $T = T' \rightarrow \mathbf{H}_T(S) = \mathbf{H}_{T'}(S)$. That is, the same set of tests yields the same amount of information regarding the locality of the fault. The aim of FLINT is not, and cannot be, to increase the amount of information; rather, it is to order tests so that the maximum information is extracted as early as possible. It follows that the next test to execute, t_{i+1}, should be the one that yields the smallest $\mathbf{H}_{T_{i+1}}(S)$.

3.1.3. Entropy Lookahead. To estimate $\mathbf{H}_{T_{i+1}}(S)$ on the basis of what we know so far, $\mathbf{P}_{T_{i+1}}(B(s_j))$ needs to be approximated. Since it is not possible to predict whether t_{i+1} will pass or fail, we use conditional probability to express both cases, based on the law of total probability, as follows:

$$\begin{aligned} \mathbf{P}_{T_{i+1}}(B(s_j)) = \quad & \mathbf{P}_{T_{i+1}}(B(s_j)|F(t_{i+1})) \cdot \alpha + \\ & \mathbf{P}_{T_{i+1}}(B(s_j)|\neg F(t_{i+1})) \cdot (1-\alpha) \end{aligned} \tag{4}$$

where α is the probability of t_{i+1} failing. The conditional probabilities $\mathbf{P}_{T_{i+1}}(B(s_j)|F(t_{i+1}))$ and $\mathbf{P}_{T_{i+1}}(B(s_j)|\neg F(t_{i+1}))$ can be calculated using the Tarantula metric and Equation 2: we simply consider two separate cases (t_{i+1} passes or fails) and calculate the lookahead suspiciousness metric accordingly.

The remaining term, α, is the probability that the $(i+1)$th test fails, i.e., $\mathbf{P}_{T_{i+1}}(F(t_{i+1}))$. Instead of using an arbitrarily fixed value, we use the observed feedback from the execution of tests in T_i as follows:

$$\alpha = \mathbf{P}_{T_{i+1}}(F(t_{i+1})) \approx \frac{TF_i}{TP_i + TF_i} \tag{5}$$

$$1 - \alpha = \mathbf{P}_{T_{i+1}}(\neg F(t_{i+1})) \approx \frac{TP_i}{TP_i + TF_i} \tag{6}$$

Using the lookahead suspiciousness, Equations 5 and 6, it is possible to estimate Equation 4, i.e. the lookahead probability of each statement containing the fault. Once normalized, the lookahead probability distribution enables the calculation of the lookahead entropy that is expected from the execution of each candidate test case for t_{i+1}. For faster fault localization after the detection of the first failing test, the tester should select the next test case that is expected to yield the lowest entropy by the approximation.

It is important to note that, while we evaluate FLINT with the Tarantula metric in the paper, the entropy lookahead described in this Section is independent of the suspiciousness metric. The better the fault localization metric is, the more effective FLINT becomes.

Table II. Working Example: calculation of lookahead entropy with the test example shown in Section 2.2

Structural Elements	Test t_1	Test t_2	Test t_3	τ if $\neg F(t_3)$	τ if $F(t_3)$	Lookahead $P(B(s_j))$	Test t_4	τ if $\neg F(t_4)$	τ if $F(t_4)$	Lookahead $P(B(s_j))$
s_1	•		•	0.00	0.33	0.04		0.00	0.00	0.00
s_2	•		•	0.00	0.33	0.04		0.00	0.00	0.00
s_3	•		•	0.00	0.33	0.04		0.00	0.00	0.00
s_4	•			0.00	0.00	0.00		0.00	0.00	0.00
s_5	•		•	0.00	0.33	0.04		0.00	0.00	0.00
s_6		•		1.00	1.00	0.26	•	0.67	1.00	0.28
s_7 (faulty)		•	•	0.67	1.00	0.21	•	0.67	1.00	0.28
s_8		•		1.00	1.00	0.26	•	0.67	1.00	0.28
s_9	•	•		0.67	0.33	0.14	•	0.50	0.50	0.17
Result	P	F		P	F	H=0.77		P	F	H=0.59

3.2. A Working Example

Let us revisit the example test scenario in Section 2.2. The first failure is detected by t_2, after which the tester has to determine the next test to execute between t_3 and t_4. Table II explains the detailed steps of entropy lookahead. Column 4 and 8 show the coverage information of t_3 and t_4 respectively. With the coverage information and specific assumptions about the test results, it is possible to calculate the lookahead Tarantula metric. For example, assuming t_3 passes, the lookahead Tarantula metric value for s_9 is:

$$\tau(s_9) = \frac{\frac{fail(s_9)}{totalfail}}{\frac{pass(s_9)}{totalpass} + \frac{fail(s_9)}{totalfail}} = \frac{\frac{1}{1}}{\frac{1}{2} + \frac{1}{1}} = \frac{2}{3}$$

Column 5 and 6 contain Tarantula metric values while assuming t_3 passes (i.e. $\neg F(t_3)$) and fails (i.e. $F(t_3)$) respectively. Similarly, column 9 and 10 contain Tarantula metric values while assuming t_4 passes and fails. Normalizing these values results in a probability distribution of $P(B(s)|\neg F(t))$ and $P(B(s)|F(t))$.

After executing t_1 and t_2, we approximate the probability of the next test failing to be 0.5, since $\frac{TF}{TP+TF} = \frac{1}{1+1}$. Now we use Equation 4 to combine pass and fail cases. For example, with t_3, the lookahead probability pf s_9 containing the fault is calculated as follows:

$$\begin{aligned}\mathbf{P}(B(s_9)) &= \mathbf{P}(B(s_9)|\neg F(t_f)) \cdot \mathbf{P}(\neg F(t_f)) + \mathbf{P}(B(s_9)|F(t_f)) \cdot \mathbf{P}(F(t_f)) \\ &= \frac{0.67}{1+0.67+1+0.67} * 0.5 + \frac{0.33}{0.33+0.33+0.33+0.33+1+1+1+0.33} * 0.5 = 0.14\end{aligned} \tag{7}$$

Finally, combining the lookahead probability distribution with Equation 3, we can calculate the lookahead entropy for t_3 and t_4, which are 0.77 and 0.59 respectively. Since t_4 produces a lower lookahead entropy value, FLINT will advise the tester to execute t_4 before t_3.

3.3. Multiple Faults

Section 3.1 assumed that we are locating a single fault. Ideally, if the suspiciousness metric that is plugged into the fault locality entropy can deal with multiple faults, FLINT will be able

to cope with localizing multiple faults. For this, the ideal suspiciousness metric would have to produce high values for all the structural elements that contain the fault and low values for everything else. In reality, the structure of the program, coupled with the locations of multiple faults, is likely to confound fault localization metrics.

However, it is not unrealistic to expect the tester to benefit from FLINT even when there exist multiple faults. For certain classes of programs and faults, we conjecture that the tester would be able to connect different failures to corresponding faults. If the tester can act as a *filter* of failure information, it is possible to utilize FLINT to target multiple faults one by one.

3.4. Research Questions

The first set of empirical studies concerns the case when C, i.e. the coverage information for each test, is known. This corresponds to the use case in which the tester wants to use FLINT for *posteriori* debugging, i.e. seeking guidance for debugging by ranking tests after executing all the tests in the order of the amount of information they reveal regarding the locality of the fault. We will refer to this study as the "Precision" study because it utilizes the precise coverage information.

RQ1. Suspiciousness: Does FLINT increase the suspiciousness of the faulty statement during testing? If so, by how much?

RQ2. Expense: If FLINT successfully increases the suspiciousness of the faulty statement, does this result in a reduction in Expense metric, i.e. the percentage of statements to be inspected before the tester encounters the faulty statement?

RQ1 is answered by observing the suspiciousness metric of the faulty statement during the execution of the test suite in two different orders: coverage-based prioritization and entropy-based prioritization. **RQ2** is answered by analyzing the percentage of the number of statements that the tester has to investigate, following the suspiciousness ranking, until the faulty statement is encountered.

The second set of empirical studies considers the case when the coverage relation between tests and statements is not known and, therefore, has to be replaced by the coverage information from the previous version. This corresponds to the use case when the tester wants to use FLINT for the actual execution of tests, not posteriori debugging. We will refer to this study as the "Robustness" study because it investigates how robust FLINT is when coverage information is not precise.

RQ3. Robustness: Does the use of coverage information from the previous version affect the effectiveness of FLINT? If so, by how much?

RQ4. Correlation: Is there any correlation between the results obtained using outdated and current coverage information?

RQ3 is answered by observing the suspiciousness and Expense metric while applying FLINT with coverage information from previous versions. **RQ4** is answered by analyzing the statistical correlation between results obtained using outdated coverage (i.e. using coverage from version $n-1$ to prioritize version n) and current coverage (i.e. using coverage from version n to prioritize version n) for the same subject and the same fault. If there is a positive correlation, the tester can decide whether to use FLINT in version n based on the performance of FLINT measured using the coverage and faults information from version $n-1$.

Algorithm 1: Entropy Lookahead
EL(t)
Input: A candidate test, t
Output: Lookahead entropy, H
(1) **foreach** $s_j \in S$
(2) $c_p \leftarrow CP_i(s_j)$
(3) **if** $s_j \in C(t)$ **then** $c_p \leftarrow c_p + 1$
(4) $\tau_{p,j} \leftarrow$TARANTULA$(TP_i + 1, TF_i, c_p, CF_i(s_j))$
(5) $c_f \leftarrow CF_i(s_j)$
(6) **if** $s_j \in C(t)$ **then** $c_f \leftarrow c_f + 1$
(7) $\tau_{f,j} \leftarrow$TARANTULA$(TP_i, TF_i + 1, CP_i(s_j), c_f)$
(8) $P \leftarrow \{p_j = 0 | 1 \leq j \leq m\}$
(9) $P_{sum} \leftarrow 0$
(10) **foreach** $s_j \in S$
(11) $p_j \leftarrow \frac{TF_i}{TP_i+TF_i} \cdot \frac{\tau_{f,j}}{\sum_{j=1}^{m} \tau_{f,j}} + \frac{TP_i}{TP_i+TF_i} \cdot \frac{\tau_{p,j}}{\sum_{j=1}^{m} \tau_{p,j}}$
(12) $P_{sum} \leftarrow P_{sum} + p_j$
(13) $H \leftarrow \sum_{p_j \in P} -\frac{p_j}{P_{sum}} \cdot \log \frac{p_j}{P_{sum}}$
(14) **return** H

Algorithm 2: FLINT
FLINT(T)
(1) $G \leftarrow \{\}$
(2) **while** $|G| < |T|$
(3) $t \leftarrow$NEXTTCPORDER$(index)$
(4) Execute t and update TP_i, TF_i, CP_i and CF_i
(5) $G \leftarrow G \cup \{t\}$
(6) **if** t fails **then break**
(7) $R \leftarrow T - G$
(8) **while** $|R| > 0$
(9) Pick $t \in R$ s.t. $\forall (t' \in R)(t' \neq t)(\mathbf{EL}(t) \leq \mathbf{EL}(t'))$
(10) Execute t and update TP_i, TF_i, CP_i and CF_i
(11) $R \leftarrow R - \{t\}$

4. ALGORITHMS

4.1. Entropy Lookahead Algorithm

To keep the pseudo-code concise, let us assume that the counter functions described in Section 3.1, $TP, TF, CP_i, CF_i : S \rightarrow \mathbf{N}$, as well as the coverage relation, C, remain available throughout Algorithm 1 and 2. The variable i denotes, whenever present, that the algorithm is seeking to find the i-th test to execute.

Algorithm 1 presents the lookahead algorithm for entropy. The loop in Line (1) calculates the lookahead suspiciousness for each statement, s_j: Lines from (2) to (4) calculate $\tau_{p,j}$, i.e. the lookahead suspiciousness of s_j for the case when that test t executes s_j (i.e. $s_j \in C(t)$) and passes (hence the increase of c_p, pass counter, by 1). Similarly, Lines from (5) to (7) calculate $\tau_{f,j}$ for the case when test t executes s_j (i.e. $s_j \in C(t)$) and passes (hence the increase of c_f, failure counter, by 1). Both cases use Equation 1 to calculate the lookahead suspiciousness values. Line (11) combines passing and failing lookahead cases using the conditional probability defined in Equation 4 in Section 3.1.3, while Line (12) calculates the sum of all probabilities. Line (13) calculates the lookahead entropy using the result from Line (12) to normalize the probability distribution.

4.2. FLINT Algorithm

Algorithm 2 illustrates the top-level algorithm for FLINT. We assume the test suite T is already prioritized using the *additional* TCP approach [Elbaum et al. 2000]: NEXTTCPORDER() is an iterator over the prioritized test suite. The loop in Line (2) follows the TCP ordering until the first failure is encountered. Once the failure is detected, it is possible to calculate the lookahead entropy following Algorithm 1. The loop in Line (8) chooses the next test t such that the lookahead entropy of t is lower than any other choice. FLINT repeats this until all tests are prioritized.

The computational complexity of the FLINT algorithm depends on the number of tests in the test suite and the number of statements in SUT. For a test suite with n tests, Algorithm 1, the entropy lookahead, requires $O(n)$. For a SUT with m statements, Algorithm 2 needs to invoke EL(t) m times. Therefore, the overall computational complexity of FLINT is $O(mn)$.

5. EXPERIMENTAL SETUP

5.1. Subjects

Table III lists the subject programs studied in the paper. All four Unix utility programs are obtained from Software Infrastructure Repository (SIR) [Do et al. 2005] along with their test suites: `flex` is a lexical analyzer, `grep` is a text-search utility, `gzip` is a compression utility and `sed` is a stream text editor. We consider five consecutive versions for each program. Table IV presents the quantity of changes, measured in terms of the numbers of insertions and deletions in the source code, between consecutive versions. This was measured using the standard `diff` tool, with options to ignore white space and blank lines and to minimize the number of changes.

Since SIR only contains the fault matrices, statement coverage information was collected using the widely used GNU profiler, `gcov`. The number of executable lines in Table III is produced by `gcov` version 4.3.2 running on Linux version 2.6.27. Both the coverage-based test case prioritization and FLINT have been performed using only the executable lines.

For each subject program, we selected a test suite that can be applied across all five versions. This is to ensure that, when FLINT is being applied to version n of the program, there exists matching coverage data for each test from version $n - 1$. However, test suites for versions 4 and 5 of `sed` were completely re-written from those for the previous versions and, therefore, could not be used as part of the robustness study.

Table III. Subject Programs from SIR

Subject	# of Tests	Lines of Code	Executable Lines
`flex`	567	12,407–14,244	3,393–3,965
`grep`	199	12,653–13,363	3,078–3,314
`gzip`	214	6,576– 7,996	1,705–1,993
`sed`	360	8,082–11,990	1,923–2,172

Table IV. Insertions/Deletions in Source Code between Versions

Subject	V1 →V2	V2 →V3	V3 →V4	V4 →V5
`flex`	3,758/1,891	881/834	204/121	234/248
`grep`	1,196/621	2,376/2,232	2,310/2,324	9/76
`gzip`	820/196	196/140	2581/2517	1139/467
`sed`	7,321/3,406	7,603/9.543	47/92	4,004/7,343

5.2. Faults & Versions

SIR provides a total of 219 (both real and seeded) faults across the five versions of the four subject programs [Do et al. 2005]. Out of 219 faults, 35 were excluded because these faults were unreachable in the compiled binary for the experimental environment. Out of the remaining 184 faults, another 92 were excluded because they were not detected by any test from the chosen test suites. The paper thus considers the remaining 92 faults, which are listed in Table V.

Table V. Number of studied faults

Subject	V1	V2	V3	V4	V5
`flex`	15	14	7	9	2
`grep`	2	1	5	3	0
`gzip`	7	3	0	3	5
`sed`	0	5	6	1	4

The robustness study uses versions from 2 to 5 of subject programs, because it requires coverage information from the previous version. Therefore, the robustness study considers only 63 faults.

5.3. Evaluation

We compare both the FLINT approach and traditional Test Case Prioritization (TCP, hereafter) with random orderings of tests to assess the effectiveness of each for early fault localization. For each pair of subject programs and available faults, we have evaluated 30 random orderings and measured the average effort required for fault localization.

The precision study utilizes both TCP and FLINT using the coverage information of the current version, whereas the robustness study depends on the coverage information from the previous version.

The research question **RQ2** concerns a widely-studied metric that measures the effectiveness of fault localization, *Expense* [Renieres and Reiss 2003], which is defined as follows:

$$Expense = \frac{\text{rank of faulty statement}}{\text{\# of executable statements}} \cdot 100$$

The numerator is the rank of the faulty statement when sorted according to the suspiciousness metric: the rank assigned to tied statements are equal to the sum of the number of the tied statements and the number of statements ranked before them [Yu et al. 2008; Renieres and Reiss 2003]. The Expense metric represents the percentage of the source code the tester must investigate before the faulty statement will be encountered. Intuitively, higher suspiciousness for the faulty statement should result in a lower Expense metric. However, for the reasons discussed in Section 2.2, the suspiciousness metric and the Expense metric may not always agree with each other.

The Tarantula metric was originally developed as a visualization for fault localization and contained two components: *hue* and *brightness* [Jones et al. 2001]. We use the hue component as suspiciousness metric following other work [Abreu et al. 2007; Yu et al. 2008], reserving brightness as a mean of discriminating when hue gives a tied suspiciousness of two or more statements[1].

We execute the test suite following Random ordering, TCP ordering and FLINT ordering. The TCP ordering is obtained using the *additional* approach with resets [Elbaum et al. 2000]. After executing each test for each ordering, we calculate the suspiciousness of the faulty statement and the corresponding Expense metric, thus providing a time-series of both metrics for each ordering.

The improvement in suspiciousness is observed by measuring the increment in suspiciousness achieved by FLINT and TCP over Random; the improvement in Expense is observed by measuring the reduction in Expense achieved by FLINT and TCP over Random ordering. Finally, we compare the Expense reduction of FLINT and TCP to each other. We report the comparison of mean values and the results of the statistical hypothesis tests.

By definition, FLINT, TCP and Random all produce the same suspiciousness metric value for all statements and, therefore, the same Expense value after the entire test suite has been executed. However, our interest lies in the case when the testing is terminated at an arbitrary point in time. Since there is no set point at which we can announce the fault *has been localized*, it is not possible to calculate the specific number of tests required to localize a fault. Rather, the aim of prioritizing tests for fault localization is to ensure the cost of fault localization is as low as possible even if the testing is terminated prematurely. To measure the benefit of FLP techniques, we calculate the mean suspiciousness and the mean Expense reduction

[1]This use of brightness as a tie-breaker was suggested by one of the anonymous referees.

Table VI. Summary of results from Precision Study

Subject	Ver.	$\bar{\tau}_R$	$\bar{\tau}_T$	$\bar{\tau}_F$	$\Delta\bar{E}_{TR}$	$\Delta\bar{E}_{FR}$	$\Delta\bar{E}_{FT}$
flex	1	0.80	0.81	0.83	0.64	1.53	0.90
flex	2	0.79	0.88	0.88	5.08	7.29	2.22
flex	3	0.77	0.86	0.86	5.53	5.79	0.26
flex	4	0.87	0.87	0.90	-1.24	0.74	1.99
flex	5	0.96	0.94	0.95	-6.99	-1.46	5.53
grep	1	0.88	0.90	0.95	1.04	1.18	0.14
grep	2	0.67	0.66	0.83	1.91	4.88	2.98
grep	3	0.65	0.76	0.83	6.81	10.64	3.83
grep	4	0.93	0.87	0.84	-1.07	-2.33	-1.26
gzip	1	0.71	0.85	0.86	11.90	11.56	-0.35
gzip	2	0.73	0.75	0.72	0.27	0.75	0.48
gzip	4	0.65	0.98	0.97	32.42	32.30	-0.12
gzip	5	0.69	0.89	0.91	15.97	14.90	-1.06
sed	2	0.75	0.78	0.76	0.37	1.33	0.96
sed	3	0.89	0.96	0.97	6.81	6.67	-0.14
sed	4	0.59	0.94	0.94	35.28	35.28	0.00
sed	5	0.91	0.95	0.95	3.57	3.42	-0.15

from n observations made after executing 1, 2, $\ldots n$ tests. The mean values represent the suspiciousness and the Expense reduction that can be *expected* by the tester if the testing is terminated at an arbitrary point. We also apply statistical hypothesis test to the set of n observations and categorize the results into the following:

— **Positive with Significance** (PS): the technique shows statistically significant improvement over the random ordering.
— **Positive with No significance** (PN): the mean value of the metric does show improvement, but without statistical significance.
— **Equal**(EQ): the technique performs as well as the random ordering.
— **Negative with No significance** (NN): the mean value of the metric does show degeneration, but without statistical significance.
— **Negative with Significance** (NS): the technique shows statistically significant degeneration compared to the random ordering.

Category EQ is possible when, for example, the faulty statement is detected by the first test and its suspiciousness remains 1.0 throughout the testing, regardless of the ordering of tests: any ordering produced by TCP or FLINT will always result in the same suspiciousness values.

6. RESULTS

6.1. Precision Study

Table VI contains the results from the precision study as well as the statistical analysis. The statistical details for each individual fault can be found in Table XII in the Appendix. Columns $\bar{\tau}_R$, $\bar{\tau}_T$ and $\bar{\tau}_F$ contain the mean suspiciousness of the faulty statement, over the entire test suite, by Random, TCP and FLINT respectively. Columns $\Delta\bar{E}_{TR}$ and $\Delta\bar{E}_{FR}$ contain the mean reductions in the Expense metric achieved by TCP and FLINT over Random respectively. $\Delta\bar{E}_{FT}$ is the mean reduction in the Expense metric achieved by FLINT over TCP.

For FLINT to produce more effective fault localization, it should provide the tester with higher suspiciousness and lower Expense metric for the faulty statement. This is analyzed using the Mann-Whitney 'U' test. The Mann-Whitney 'U' test is a non-parametric statistical hypothesis test, i.e. it allows the comparison of two samples with unknown distributions. Columns H_1, H_2 and H_3 contain the result classifications of the Mann-Whitney 'U' test for the

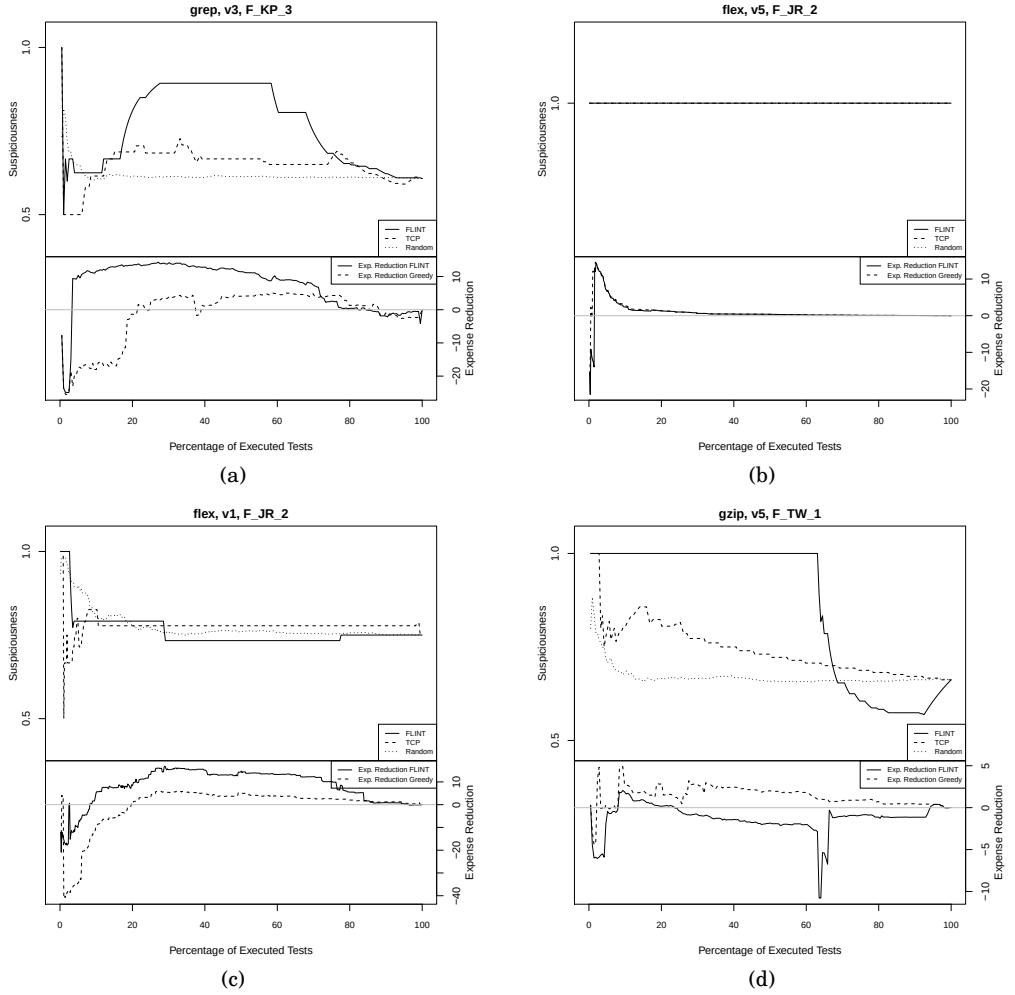


Fig. 1. Plots of suspiciousness and Expense reduction from the precision study. Figure 1(a) represents the cases when the FLINT approach produces higher suspiciousness and lower Expense. Figure 1(b) and 1(c) represent the cases when the reduction in entropy achieved by FLINT results in reduced Expense, even though the suspiciousness value of the faulty statement is largely equal to that of TCP (Figure 1(b)) or even lower (Figure 1(c)). However, in Figure 1(d), the suspiciousness metric is not in alignment with Expense: the increased suspiciousness metric resulted in higher Expense.

Expense metric. The null hypothesis for all three hypothesis tests is that there is no difference in the mean values between the compared approaches. For H_1, the alternative hypothesis is that TCP produces a lower mean Expense than Random does ($\bar{E}_T < \bar{E}_R$); for H_2, the alternative hypothesis is that FLINT produces a lower mean Expense than Random does ($\bar{E}_F < \bar{E}_R$). Similarly, for H_3, the alternative hypothesis is that FLINT produces a lower mean Expense than TCP does ($\bar{E}_F < \bar{E}_T$). The confidence level is 95%[2].

[2]Given a test suite with n tests, our 'sample' for the Mann-Whitney 'U' test is n comparisons of Expense between two permutations of the test suite, one after the execution of each test following the permutation respectively. Con-

Table VII contains the classification of the statistical hypothesis testing for the precision study. Both TCP and FLINT produce a lower mean Expense metric for more than 70% of the faults. However, Random also produces a lower mean Expense metric than both approaches for more than 25%. Considering the existing evidence that test case prioritization is effective for early fault *detection*, the results suggest that prioritization of test cases for early fault localization may be an entirely different task from prioritization for early fault detection.

The results for H_3 in Table VII also suggest that FLINT performs equally well or better than TCP for about 54% of the studied faults. Considering that each fault is unique not only in its own characteristics but also in its interaction with the structure of the SUT and the test suite, assessing the risk of applying FLINT to faults for which TCP does better may not be obvious. However, the robustness study in Section 6.2 provides a potential way forward in deciding whether or not to apply FLINT or TCP.

Figure 1 provides a more detailed explanation with exemplar cases. The upper panes of each of its 4 subfigures show how the suspiciousness metric for the faulty statement changes during the execution of tests, following the orders from Random, TCP and FLINT. The lower panes show the reductions in Expense (ΔE_{TR} and ΔE_{FR}).

Figure 1(a) shows a case when both TCP and FLINT produce mean Expense values that belong to a `PS` category. FLINT produces higher suspiciousness than the other two approaches during most of the duration of the testing, which results in reductions in Expense.

Figure 1(b) and 1(c) represent two interesting cases. Reduction in Expense is achieved despite the fact that the suspiciousness metric for the faulty statement from FLINT either remains identical to that of TCP at 1.0 (Figure 1(b)), or even becomes lower (Figure 1(c)). These results are achieved because choosing a test that produces the lowest entropy may not only increase the suspiciousness of the faulty statement but also lower the suspiciousness of the non-faulty statements (Section 3.1.2).

However, the results also show that an increment of the suspiciousness metric may not always result in a reduction in Expense, as discussed in Section 2.2 and Section 5.3. In Figure 1(d), although FLINT achieves a higher suspiciousness metric for the fault `F_TW_1` for version 5 of `gzip` than other approaches, it fails to make reductions in Expense. This provides evidence that a higher suspiciousness value for the faulty statement may not always result in a lower Expense for locating it.

Table VII. Hypothesis Test for Precision Study

Hypothesis	PS	PN	EQ	NN	NS
$H_1 : \bar{E}_T < \bar{E}_R$	72.83%	0.00%	0.00%	0.00%	27.17%
$H_2 : \bar{E}_F < \bar{E}_R$	76.09%	1.09%	0.00%	1.09%	21.74%
$H_3 : \bar{E}_F < \bar{E}_T$	40.22%	4.35%	9.78%	3.26%	42.39%

To answer **RQ1**, Table VI and XII provide evidence that FLINT achieves a higher suspiciousness for the faulty statement during the execution of the test suite for many of the studied faults. More importantly, the increased suspiciousness leads to reductions in Expense in many cases, which provides the answer to **RQ2**: FLINT reduces the Expense metric with statistical significance for 76% of the studied faults when compared to Random ordering. The cases presented in Figure 1 show that the reduction in Expense is possible, even when increasing the suspiciousness of the faulty statement is not possible.

sequently, our sample size is equal to the population size for comparing two permutations (i.e., we make all available comparisons). In the case with the random ordering, we compare FLINT and TCP against the average values from 30 random orderings.

Table VIII. Summary of results from Robustness Study

Subject	Ver.	$\bar{\tau_R}$	$\bar{\tau_T}$	$\bar{\tau_F}$	$\Delta\bar{E}_{TR}$	$\Delta\bar{E}_{FR}$	$\Delta\bar{E}_{FT}$
flex	2	0.79	0.87	0.88	4.79	5.95	1.16
flex	3	0.77	0.86	0.86	5.53	5.79	0.26
flex	4	0.87	0.87	0.90	-1.24	0.93	2.18
flex	5	0.96	0.94	0.95	-6.99	-1.48	5.51
grep	2	0.67	0.66	0.83	-3.35	4.90	8.25
grep	3	0.65	0.76	0.83	6.69	11.09	4.41
grep	4	0.93	0.93	0.90	1.88	0.62	-1.25
gzip	2	0.73	0.76	0.70	0.81	1.08	0.27
gzip	4	0.65	0.98	0.97	32.43	32.31	-0.12
gzip	5	0.69	0.89	0.86	16.32	15.58	-0.75
sed	2	0.75	0.78	0.77	0.27	0.78	0.52
sed	3	0.89	0.96	0.97	6.83	7.09	0.26

6.2. Robustness Study

Table VIII contains the results from the robustness study as well as the statistical analysis. The statistical details for each individual fault can be found in Table XIII in the Appendix. The results of the statistical hypothesis tests are summarized in Table IX. Both TCP and FLINT produce lower mean Expense than Random for about 70% of the studied faults.

Figure 2 presents the representative outcome of the robustness study. Figure 2(a) and 2(b) contain the plots for the same faults depicted in Figure 1(a) and 1(b). It shows that the changes in suspiciousness of the faulty statement retain similar patterns when FLINT is applied using coverage information from the previous version rather than the current version. The results shown in these two plots remain positive, as they did for the precision study.

Similarly, Figure 2(c) and Figure 2(d) show cases that correspond to Figure 1(c) and Figure 1(d). Even when FLINT depends on coverage information from the previous version, it is possible either to reduce Expense metric without producing higher suspiciousness (Figure 1(c)) or to increase suspiciousness without necessarily reducing Expense (Figure 1(d)).

To answer **RQ3**, comparing the results of the robustness study (Table XIII and IX, Figure 2) to those of the precision study indicates that the use of previous coverage information does not affect the performance of FLINT in any significant way. The trends and patterns observed in the precision study manifest themselves again in the robustness study. Both TCP and FLINT achieved statistically significant Expense reductions for about 70% of the studied faults.

To answer **RQ4**, we compare the reductions in Expense obtained using the coverage from current versions with those obtained using the coverage from the previous versions, for each of 63 faults from versions 2 to 5 of the four subject programs. The impact of outdated coverage information is analyzed by testing Pearson's correlation between the Expense reduction obtained with precise and outdated information. If the use of outdated coverage information has no impact, there will be a perfect Pearson's correlation with $\rho = 1.0$. A strong positive correlation will indicate that coverage information from the previous version can be used without damaging the performance of fault localization techniques.

Figure 3 shows the correlation observed in the ΔE_{TR}, ΔE_{FR}, and ΔE_{FT} values for the 63 faults studied for the robustness study. The x-axis denotes the reductions in Expense obtained for each fault using the precise (i.e. current) coverage information. The y-axis denotes the reductions in Expense for the same fault using the outdated (i.e. previous) coverage information. All three plots show strong positive correlations with statistical significance. This provides an answer to **RQ4**: the impact of outdated coverage information on the performance of fault localization techniques is not significant. In particular, Figure 3(c) provides supporting evidence

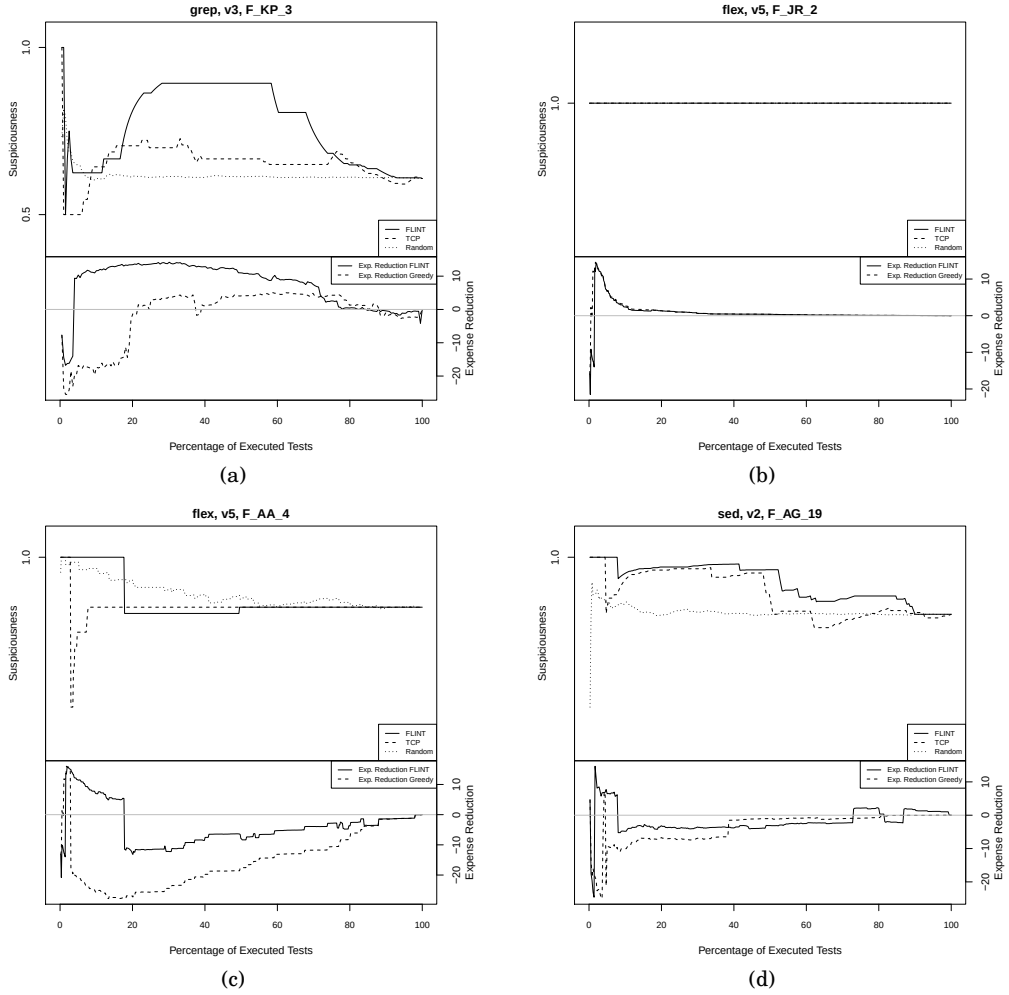
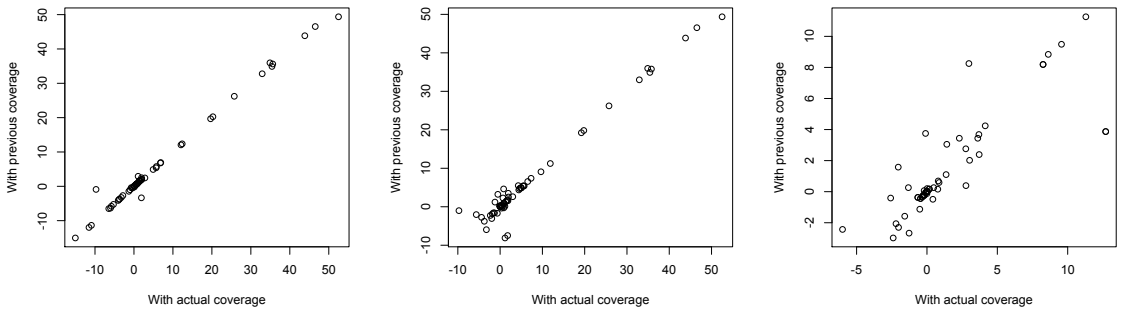


Fig. 2. Plots of suspiciousness and Expense reduction from the robustness study. Figures 2(a) and 2(b) correspond to Figures 1(a) and 1(b) respectively: FLINT achieves improved suspiciousness and reduced Expense for these faults, despite the use of outdated coverage information from the previous version. Figure 2(c) shows another case where FLINT achieves a reduction in Expense even though its suspiciousness is lower than that of Random, similar to Figure 1(c). Figure 2(d) shows a negative case when increased suspiciousness does not lead to a reduction in Expense.

that developers can decide which fault localisation technique to use, based on how well each technique performs for localising known faults from the previous version.

Table IX. Hypothesis Test for Robustness Study

Hypothesis	PS	PN	EQ	NN	NS
$H_1 : \bar{E}_T < \bar{E}_R$	69.84%	0.00%	0.00%	0.00%	30.16%
$H_2 : \bar{E}_F < \bar{E}_R$	69.84%	3.17%	0.00%	1.59%	25.40%
$H_3 : \bar{E}_F < \bar{E}_T$	44.44%	4.76%	9.52%	0.00%	41.27%



(a) $\Delta E_{TR}, \rho = 0.9952$ (b) $\Delta E_{FR}, \rho = 0.9851$ (c) $\Delta E_{FT}, \rho = 0.8355$

Fig. 3. Correlation plots that show the impact of using coverage information from previous versions. Each data point corresponds to the reductions in Expense with previous and current coverage with respect to localizing a specific fault. The closer to 1.0 the correlation coefficient is, the smaller the impact of using coverage information from previous versions will be. The p–values for the Pearson's correlation test for all three cases are less than 10^{-10}.

6.3. Comparison to Similarity Sorting

One potential approach to prioritizing test cases for faster fault localization would be to sort the test cases according to the similarity to the first test that has detected the fault [Artzi et al. 2010], hoping that similar tests would also fail and contribute to higher suspiciousness value for the real faulty statement[3]. To compare FLINT to this approach, we have implemented the similarity sorting approach. To measure the similarity between test cases, we adopted Hamming distance between the binary execution traces of test cases. For a program with n statements, the execution trace of a test is a binary string of length n: the i-th digit is 1 if the i-th statement was executed by the test, 0 if not. Binary execution trace has been used to effectively capture the similarity between test cases for regression testing techniques [Leon and Podgurski 2003; Yoo et al. 2009].

Algorithm 3 illustrates the pseudo-code of the similarity sorting approach. The algorithm is similar to FLINT described in Algorithm 2 except Line (7) and (11). In Line (7), the similarity sorting algorithm records the first failing test case, t_f. In Line (11), the algorithm chooses the test case with minimum Hamming distance to t_f as the next test to execute: function HD(t_1, t_2) denotes the calculation of Hamming distance between the execution trace of test t_1 and t_2.

[3]This 'similarity' approach was suggested to us by one of the anonymous referees.

Algorithm 3: SimilaritySort

SIMILARITYSORT(T)
(1) $G \leftarrow \{\}$
(2) **while** $|G| < |T|$
(3) $t \leftarrow$ NEXTTCPORDER($index$)
(4) Execute t and update TP_i, TF_i, CP_i and CF_i
(5) $G \leftarrow G \cup \{t\}$
(6) **if** t fails
(7) $t_f \leftarrow t$
(8) **break**
(9) $R \leftarrow T - G$
(10) **while** $|R| > 0$
(11) Pick $t \in R$ s.t. $\forall (t' \in R)(t' \neq t)(\text{HD}(t, t_f) \leq \text{HD}(t', t_f))$
(12) Execute t and update TP_i, TF_i, CP_i and CF_i
(13) $R \leftarrow R - \{t\}$

Table X presents the result of comparison between FLINT and the similarity sorting approach. Detailed results for each individual fault can be found in Table XIV in the Appendix. Both techniques are evaluated against the random baseline results: $\Delta\bar{E}_{FR}$ denotes the reduction of Expense produced by FLINT over the random, whereas $\Delta\bar{E}_{SR}$ denotes the reduction of Expense brought in by the similarity sorting approach over the random. Columns H contains the results of Mann-Whitney 'U' test in the manner described in Section 5.3. The null hypothesis is that there is no difference in mean Expense reduction between two approaches. The alternative hypothesis is that FLINT produces smaller Expense values than the similarity sorting approach.

Table X. Results of comparison to similarity sorting approach

Subject	Ver.	$\bar{\tau_R}$	$\bar{\tau_F}$	$\bar{\tau_S}$	$\Delta\bar{E}_{FR}$	$\Delta\bar{E}_{SR}$	$\Delta\bar{E}_{FS}$
flex	1	0.80	0.83	0.86	1.53	1.51	0.03
flex	2	0.79	0.88	0.81	7.29	5.11	2.18
flex	3	0.77	0.86	0.86	5.79	4.39	1.40
flex	4	0.87	0.90	0.86	0.74	-2.68	3.42
flex	5	0.96	0.95	0.76	-1.46	-23.09	21.63
grep	1	0.88	0.95	0.88	1.18	-3.75	4.94
grep	2	0.67	0.83	0.55	4.88	-0.89	5.78
grep	3	0.65	0.83	0.63	10.64	1.65	8.99
grep	4	0.93	0.84	0.83	-2.33	-1.99	-0.34
gzip	1	0.71	0.86	0.86	11.56	11.58	-0.02
gzip	2	0.73	0.72	0.69	0.75	0.17	0.57
gzip	4	0.65	0.97	0.97	32.30	32.33	-0.03
gzip	5	0.69	0.91	0.84	14.90	13.54	1.37
sed	2	0.75	0.76	0.68	1.33	-3.20	4.53
sed	3	0.89	0.97	0.95	6.67	7.11	-0.44
sed	4	0.59	0.94	0.94	35.28	35.28	0.00
sed	5	0.91	0.95	0.94	3.42	3.82	-0.40

Overall, it can be seen that the similarity sorting approach fails to achieve reductions in expense for many faults. Table X and XIV show that FLINT produces lower Expense, with statistical significance, than the similarity sorting approach for over 50% of the studied faults. It may appear surprising that the similarity sorting approach fails to consistently outperform

Table XI. Hypothesis Test for Similarity Sorting Study

Hypothesis	PS	PN	EQ	NN	NS
$H : \bar{E}_F < \bar{E}_S$	52.17%	3.26%	10.87%	10.87%	22.83%

the random prioritization. We conjecture that this is due to the discrepancies between the actual semantic similarity between tests and that between execution traces measured in Hamming distance. Depending on the location and the nature of a fault, sorting tests according to Hamming distance to the failing one may or may not prioritise reproduction of the fault (i.e. the next sorted test does execute the faulty statement but does not fail, therefore hindering the fault localization process). When the sorting does not contribute to the fault reproduction, the approach may fail to outperform the random prioritization.

7. THREATS TO VALIDITY

There are a few threats to validity regarding the generalization of the results presented in this paper. First, FLINT was evaluated using only Tarantula metric as the basis of probability distribution. Using other fault localization metrics may lead to different results. However, the overall approach of FLINT should apply, regardless of the choice of the fault localization metric, as long as the assumptions stated in Section 3.1.1 are met. Second, different faults or subject programs may affect the performance of FLINT. We have tried to include various types of faults across multiple versions of subject programs, but only additional study can further reduce this threat. Similarly, the degrees of change between consecutive versions can vary widely: changes that are significantly different from those we studied may result in different levels of effectiveness when FLINT is used with coverage information from the previous version. Finally, certain classes of faults may not satisfy the assumptions stated in Section 3.1.1. For example, concurrency faults or any failure that is caused in a nondeterministic manner can affect the performance of FLINT adversely. While it is beyond the scope of this paper, we conjecture that the entropy model of fault locality would have to be extended with probabilistic model in order for it to cater for nondeterministic factors.

8. RELATED WORK

Test case prioritization is a regression testing technique that aims to maximize the rate of fault detection when the testing is terminated at an arbitrary point [Yoo and Harman 2012]. Since the fault information is not known, test case prioritization techniques often rely on surrogates, for which structural coverage is widely used [Rothermel et al. 2001; Elbaum et al. 2000; Li et al. 2007]. Other prioritization criteria such as execution history [Kim and Porter 2002], execution profile [Leon and Podgurski 2003] and expert knowledge [Tonella et al. 2006; Yoo et al. 2009] have also been studied. However, test case prioritization techniques do not separately consider the case when a fault is actually detected during the execution of the prioritized test suite. This paper is the first to consider the effectiveness of fault localization under the impact of time constraints.

Fault localization is a debugging technique that aims to aid the tester to locate a detected fault [Renieres and Reiss 2003]. Existing work focuses on coverage-based metrics, program spectra analysis or the Program Dependence Graph (PDG) to locate faults [Renieres and Reiss 2003; Cleve and Zeller 2005; Liblit et al. 2005; Jones and Harrold 2005; Abreu et al. 2007]. Recent work uses a probabilistic causal inference model for better fault localization [Baah et al. 2010] or fuzzy set logic to improve the relationship between tests and fault locality [Hao et al. 2008]. There is existing work that investigates how fault localization is affected by test suite reduction [Yu et al. 2008] or test case prioritization [Jiang et al. 2009].

There are existing techniques for prioritizing tests for fault localization. Gonzalez-Sanchez et al. prioritized test cases for fault localization by minimizing the fault locality entropy sim-

ilarly to FLINT [Gonzalez-Sanchez et al. 2011]. The technique from Gonzalez-Sanchez uses Bayesian theory to calculate suspiciousness and prioritize the entire test suite for fault localization only. FLINT can use any existing fault localization metric to calculate the entropy and takes over from normal test case prioritization technique only after the first failing test is executed, thereby preserving both fault detection and localization capability. Artzi et al. proposed that fault localization can be aided by generating and executing tests that have similar execution paths to that of the failing one [Artzi et al. 2010]. The intuition is that similar tests with passing and failing results would maximize the chance that the fault is correlated with the small difference between them.

Information Theory [Cover and Thomas 1991], now an extensive branch of probability theory with many applications, was famously founded by Claude Shannon in a single paper [Shannon 1948]. It has been applied in many research areas related to computer science including machine learning, analysis of algorithms, and data mining. Applications to software engineering and particularly to programming languages, have been less common. Software metrics [Allen and Khoshgoftaar 1999] and software evolution [Arbuckle 2011] are both areas which have seen contributions but the most active area at the present time is program analysis for quantifying information flow.

Questions about quantified information flow (QIF) arise naturally in the theory of dependence, particularly in the theory of security, in order to measure the strength of dependence (e.g. for potential covert channels). It is not surprising that one of the the earliest applications of Shannon information to programming languages was in Denning's 1982 book on cryptography and data security [Denning 1982] where it appears in an informal discussion of how to analyze program constructs in terms of information flow, along with an attempt to define flow quantity. Although it subsequently became fashionable to use information theory in discussions of security properties for software systems, the first automatable analysis for QIF did not appear until 2002 [Clark et al. 2002]. This latter work was extended to a Turing complete language [Clark et al. 2007] and to a process language [Boreale et al. 2010]. In the last five years, a vibrant community of researchers into QIF has developed. However, all the applications to date have been concerned with flow security.

Our paper introduces a novel application of Shannon entropy to the analysis of programs and has potential for extension to a theoretical framework for a probabilistic approach to testing. It is not QIF based but it adopts a similar approach to Clarkson et alia's Bayesian-influenced paper on quantifying information flow [Clarkson et al. 2009]. The paper describes a security attack in which a series of experiments successively update the attacker's belief about the probability distribution on the space of secrets with the aim of refining that probability distribution to the one in which the actual secret input to the program has probability 1 while all others have probability 0.

9. CONCLUSION & FUTURE WORK

This paper presents the first use of Information Theory for fault localization. We build an entropy model for the locality of the fault. The probability distribution of the locality of the fault is approximated using an existing fault localization metric. The proposed technique, FLINT, aims to improve the effectiveness of fault localization by trying to reduce the Shannon entropy of the locality of the fault. While the paper uses the Tarantula metric, any fault localization metric can be plugged into FLINT.

We evaluate FLINT by evaluating its effectiveness for a novel problem: Fault Localization Prioritization (FLP). Once a test suite, prioritized for early fault detection, detects a fault, we switch to the FLINT approach to maximize the chance of early fault localization, even if testing is terminated prematurely. Empirical evaluation of FLINT shows that it is possible to

increase the suspiciousness metric and reduce the fault localization cost for more than 70% of the studied faults.

Future work will include the use of more advanced fault localization analysis for the lookahead method, with an emphasis on reducing the discrepancy between the suspiciousness and the Expense metric.

REFERENCES

ABREU, R., ZOETEWEIJ, P., AND VAN GEMUND, A. J. C. 2007. On the accuracy of spectrum-based fault localization. In *Proceedings of the Testing: Academic and Industrial Conference Practice and Research Techniques - MUTATION*. IEEE Computer Society, 89–98.

ALLEN, E. B. AND KHOSHGOFTAAR, T. M. 1999. Measuring coupling and cohesion: An information-theory approach. In *IEEE METRICS*.

ARBUCKLE, T. 2011. Studying software evolution using artefacts' shared information content. *Science of Computer Programming 76,* 12, 1078–1097.

ARTZI, S., DOLBY, J., TIP, F., AND PISTOIA, M. 2010. Directed test generation for effective fault localization. In *Proceedings of the 19th international symposium on Software testing and analysis*. ISSTA '10. ACM, New York, NY, USA, 49–60.

BAAH, G. K., PODGURSKI, A., AND HARROLD, M. J. 2010. Causal inference for statistical fault localization. In *Proceedings of the 19th International Symposium on Software Testing and Analysis (ISSTA 2010)*. ACM Press, 73–84.

BOREALE, M., CLARK, D., AND GORLA, D. 2010. A semiring-based trace semantics for processes with applications to information leakage analysis. In *Theoretical Computer Science*, C. Calude and V. Sassone, Eds. IFIP Advances in Information and Communication Technology Series, vol. 323. Springer Boston, 340–354.

CLARK, D., HUNT, S., AND MALACARIA, P. 2002. Quantitative analysis of the leakage of confidential data. *Electronic Notes in Theoretical Computer Science 59,* 3, 1–14.

CLARK, D., HUNT, S., AND MALACARIA, P. 2007. A static analysis for quantifying information flow in a simple imperative language. *Journal of Computer Security 15,* 3, 321 – 372.

CLARKSON, M. R., MYERS, A. C., AND SCHNEIDER, F. B. 2009. Quantifying information flow with beliefs. *Journal of Computer Security 17,* 5, 655–701.

CLEVE, H. AND ZELLER, A. 2005. Locating causes of program failures. In *Proceedings of the 27th international conference on Software engineering*. ICSE '05. ACM, New York, NY, USA, 342–351.

COVER, T. M. AND THOMAS, J. A. 1991. *Elements of Information Theory*. Wiley Interscience.

DENNING, D. E. R. 1982. *Cryptography and Data Security*. Addison-Wesley.

DO, H., ELBAUM, S. G., AND ROTHERMEL, G. 2005. Supporting controlled experimentation with testing techniques: An infrastructure and its potential impact. *Empirical Software Engineering 10,* 4, 405–435.

ELBAUM, S. G., MALISHEVSKY, A. G., AND ROTHERMEL, G. 2000. Prioritizing test cases for regression testing. In *Proceedings of International Symposium on Software Testing and Analysis*. 102–112.

GONZALEZ-SANCHEZ, A., PIEL, É., ABREU, R., GROSS, H.-G., AND VAN GEMUND, A. J. C. 2011. Prioritizing tests for software fault diagnosis. *Software: Practice and Experience 41,* 10, 1105–1129.

HAO, D., ZHANG, L., PAN, Y., MEI, H., AND SUN, J. 2008. On similarity-awareness in testing-based fault localization. *Automated Software Engineering 15*, 207–249.

JIANG, B., ZHANG, Z., TSE, T. H., AND CHEN, T. Y. 2009. How well do test case prioritization techniques support statistical fault localization. In *Proceedings of the 33rd Annual International Computer Software and Applications Conference (COMPSAC 2009)*. IEEE Computer Society Press, 99–106.

JONES, J. A. AND HARROLD, M. J. 2005. Empirical evaluation of the tarantula automatic fault-localization technique. In *Proceedings of the 20th International Conference on Automated Software Engineering (ASE2005)*. ACM Press, 273–282.

JONES, J. A., HARROLD, M. J., AND STASKO, J. 2002. Visualization of test information to assist fault localization. In *Proceedings of the 24th International Conference on Software Engineering*. ACM, New York, NY, USA, 467–477.

JONES, J. A., HARROLD, M. J., AND STASKO, J. T. 2001. Visualization for fault localization. In *Proceedings of ICSE Workshop on Software Visualization*. 71–75.

KIM, J.-M. AND PORTER, A. 2002. A history-based test prioritization technique for regression testing in resource constrained environments. In *Proceedings of the 24th International Conference on Software Engineering*. ACM Press, 119–129.

LEON, D. AND PODGURSKI, A. 2003. A comparison of coverage-based and distribution-based techniques for filtering and prioritizing test cases. In *Proceedings of the IEEE International Symposium on Software Reliability Engineering (ISSRE 2003)*. IEEE Computer Society Press, 442–456.

LI, Z., HARMAN, M., AND HIERONS, R. M. 2007. Search Algorithms for Regression Test Case Prioritization. *IEEE Transactions on Software Engineering 33,* 4, 225–237.

LIBLIT, B., NAIK, M., ZHENG, A. X., AIKEN, A., AND JORDAN, M. I. 2005. Scalable statistical bug isolation. In *Proceedings of the 2005 ACM SIGPLAN Conference on Programming Language Design and Implementation*. PLDI '05. ACM, New York, NY, USA, 15–26.

RENIERES, M. AND REISS, S. 2003. Fault localization with nearest neighbor queries. In *Proceedings of the 18th International Conference on Automated Software Engineering*. 30 – 39.

ROTHERMEL, G., UNTCH, R. J., AND CHU, C. 2001. Prioritizing test cases for regression testing. *IEEE Transactions on Software Engineering 27,* 10, 929–948.

SHANNON, C. E. 1948. A mathematical theory of communication. *Bell System Technical Journal 27*, 379–423 and 623–656.

TONELLA, P., AVESANI, P., AND SUSI, A. 2006. Using the case-based ranking methodology for test case prioritization. In *Proceedings of the 22nd International Conference on Software Maintenance (ICSM 2006)*. IEEE Computer Society, 123–133.

YOO, S. AND HARMAN, M. 2012. Regression testing minimisation, selection and prioritisation: A survey. *Software Testing, Verification, and Reliability 22,* 2, 67–120.

YOO, S., HARMAN, M., TONELLA, P., AND SUSI, A. 2009. Clustering test cases to achieve effective & scalable prioritisation incorporating expert knowledge. In *Proceedings of International Symposium on Software Testing and Analysis (ISSTA 2009)*. ACM Press, 201–211.

YU, Y., JONES, J. A., AND HARROLD, M. J. 2008. An empirical study of the effects of test-suite reduction on fault localization. In *Proceedings of the International Conference on Software Engineering (ICSE 2008)*. ACM Press, 201–210.

Appendix

Table XII: Detailed Statistical Analysis for Precision Study

Subject	Fault ID	$\bar{\tau_R}$	$\bar{\tau_T}$	$\bar{\tau_F}$	$\Delta\bar{E}_{TR}$	H_1	$\Delta\bar{E}_{FR}$	H_2	$\Delta\bar{E}_{FT}$	H_3
flex V1	F_AA_6	0.48	0.53	0.55	2.78	PS	1.88	PS	-0.90	NS
	F_AA_1	0.76	0.78	0.76	0.89	PS	9.46	PS	8.57	PS
	F_AA_2	0.99	1.00	1.00	0.40	PS	0.11	PS	-0.29	NS
	F_AA_3	0.53	0.59	0.75	4.56	PS	-0.23	NS	-4.79	NS
	F_JR_4	1.00	1.00	1.00	0.04	NS	-1.89	NS	-1.93	NS
	F_JR_6	1.00	1.00	1.00	0.05	PS	-4.36	NS	-4.41	NS
	F_JR_5	1.00	1.00	1.00	0.62	PS	0.56	PS	-0.07	NS
	F_JR_2	0.78	0.78	0.76	-0.46	PS	8.11	PS	8.57	PS
	F_JR_3	1.00	1.00	1.00	2.55	PS	-1.12	PS	-3.67	NS
	F_HD_3	0.99	1.00	1.00	0.82	PS	0.53	PS	-0.29	NS
	F_HD_1	0.51	0.54	0.56	-0.90	NS	3.26	PS	4.16	PS
	F_HD_6	0.53	0.51	0.50	-5.25	NS	4.19	PS	9.44	PS
	F_HD_7	0.93	0.95	0.99	0.72	NS	0.52	PS	-0.21	PS
	F_HD_4	0.51	0.53	0.56	2.57	PS	3.41	PS	0.84	PS
	F_HD_5	1.00	1.00	1.00	0.12	PS	-1.41	PS	-1.53	PS
flex V2	F_AA_4	0.63	0.63	0.61	1.50	PS	-0.54	PS	-2.04	NS
	F_AA_5	0.61	0.61	0.64	-2.84	NS	0.87	PS	3.71	PS
	F_AA_2	0.46	0.50	0.50	-6.46	NS	-2.32	NN	4.14	PS
	F_AA_3	0.46	0.99	0.99	52.50	PS	52.50	PS	0.00	EQ
	F_JR_6	0.48	0.97	0.97	35.62	PS	35.78	PS	0.15	PS
	F_HD_8	0.96	0.96	0.99	-10.92	NS	1.78	PS	12.70	PS
	F_JR_5	1.00	1.00	1.00	0.10	PS	-0.01	NS	-0.10	NS
	F_JR_2	0.97	0.96	0.99	-11.54	NS	1.16	PS	12.70	PS
	F_JR_3	1.00	0.99	1.00	-6.02	NS	-3.25	NS	2.77	PS
	F_JR_1	1.00	1.00	1.00	0.09	PS	-0.01	NS	-0.10	NS
	F_HD_2	0.88	1.00	1.00	12.38	PS	11.87	PS	-0.51	NS
	F_HD_6	0.99	1.00	1.00	0.36	PS	-5.64	NS	-6.00	NS
	F_HD_7	0.95	1.00	1.00	4.92	PS	4.92	PS	-0.00	NN
	F_HD_4	0.69	0.69	0.71	1.39	PS	5.00	PS	3.61	PS
flex V3	F_AA_4	0.49	0.50	0.50	-5.26	NS	-1.57	NS	3.69	PS
	F_AA_5	0.99	1.00	1.00	0.55	PS	0.31	PS	-0.24	NS
	F_AA_3	0.34	0.53	0.54	1.80	PS	1.54	PS	-0.26	NS
	F_JR_5	0.80	1.00	1.00	19.67	PS	19.21	PS	-0.46	NS
	F_JR_2	0.99	1.00	1.00	0.52	PS	0.28	PS	-0.24	NS
	F_JR_3	0.79	1.00	1.00	20.26	PS	19.79	PS	-0.46	NS
	F_HD_6	0.98	1.00	1.00	1.19	PS	0.95	PS	-0.24	NS
flex V4	F_AA_7	0.99	0.98	0.99	-1.01	NS	1.75	PS	2.76	PS
	F_AA_1	0.80	0.81	0.89	0.13	PS	0.93	PS	0.80	PS
	F_AA_2	0.99	1.00	1.00	0.70	PS	0.44	PN	-0.26	NS
	F_AA_3	0.51	0.54	0.63	-3.25	NS	4.99	PS	8.25	PS
	F_JR_4	0.99	0.99	0.99	-5.80	NS	-4.39	NS	1.41	PS
	F_JR_2	0.99	1.00	1.00	0.70	NS	-1.33	NS	-2.02	NS
	F_JR_3	1.00	1.00	1.00	0.57	PS	-0.08	PS	-0.65	NS
	F_JR_1	0.51	0.54	0.63	-3.83	NS	4.42	PS	8.25	PS
	F_HD_5	0.99	1.00	1.00	0.61	PS	-0.05	PS	-0.65	NS
flex V5	F_AA_4	0.92	0.89	0.91	-15.04	NS	-3.74	NS	11.29	PS
	F_JR_2	1.00	1.00	1.00	1.05	PS	0.81	PS	-0.24	NS

grep V1	F_KP_2	1.00	1.00	1.00	-0.16	PS	-1.77	NS	-1.61	NS
	F_DG_4	0.77	0.80	0.91	2.24	PS	4.14	PS	1.90	PS
grep V2	F_DG_1	0.67	0.66	0.83	1.91	PS	4.88	PS	2.98	PS
grep V3	F_KP_7	0.49	0.84	0.84	35.42	PS	35.42	PS	0.00	EQ
	F_KP_3	0.62	0.65	0.76	-1.31	PS	7.32	PS	8.62	PS
	F_DG_8	0.42	0.60	0.70	2.00	PS	0.67	PS	-1.32	NS
	F_DG_2	0.97	0.96	0.99	-4.04	NS	5.52	PS	9.56	PS
	F_DG_3	0.73	0.75	0.85	1.99	PS	4.29	PS	2.30	PN
grep V4	F_KP_8	0.93	1.00	1.00	6.79	PS	4.58	PS	-2.20	NS
	F_DG_3	0.89	0.62	0.62	-9.73	PS	-9.74	PS	-0.00	NS
	F_KP_6	0.97	0.98	0.90	-0.26	PS	-1.84	NS	-1.58	NS
gzip V1	F_KL_2	0.47	0.81	0.82	10.51	PS	11.62	PS	1.12	PS
	F_KL_6	0.50	0.41	0.42	0.34	NS	0.34	NS	0.00	EQ
	F_KP_10	0.57	0.99	0.99	40.25	PS	40.25	PS	0.00	EQ
	F_KP_11	0.79	0.77	0.83	-3.06	NS	-3.35	NS	-0.29	PS
	F_KP_9	0.79	1.00	1.00	20.07	PS	19.54	PS	-0.54	NN
	F_TW_3	0.90	1.00	1.00	10.68	PS	10.43	PS	-0.25	NN
	F_KP_1	0.94	0.98	0.97	4.52	PS	2.06	PS	-2.46	NS
gzip V2	F_KL_1	0.67	0.73	0.62	-0.43	PS	0.34	PS	0.77	PS
	F_KL_3	0.93	0.87	0.90	1.07	PS	1.93	PS	0.86	PS
	F_KL_8	0.58	0.66	0.65	0.17	PS	-0.03	PS	-0.20	NS
gzip V4	F_KL_1	0.50	0.99	0.99	46.53	PS	46.53	PS	-0.00	NS
	F_KL_8	0.90	0.96	0.94	6.88	PS	6.53	PS	-0.35	NS
	F_KP_3	0.53	0.99	0.99	43.85	PS	43.85	PS	0.00	EQ
gzip V5	F_KL_1	0.93	0.96	0.95	5.75	PS	5.42	PS	-0.33	NS
	F_KL_2	0.66	0.89	0.87	12.07	PS	9.66	PS	-2.40	NS
	F_KL_4	0.50	0.91	0.91	34.90	PS	34.90	PS	0.00	EQ
	F_KL_8	0.69	0.95	0.95	25.75	PS	25.75	PS	0.00	EQ
	F_TW_1	0.67	0.74	0.86	1.36	PS	-1.22	NS	-2.58	NS
sed V2	F_AG_20	0.62	0.64	0.63	0.71	PS	1.13	PS	0.42	PN
	F_AG_17	0.34	0.38	0.41	2.81	PS	2.99	PS	0.18	PS
	F_AG_12	0.94	0.96	0.97	0.49	NS	1.85	PS	1.36	PS
	F_AG_19	0.88	0.92	0.85	-3.69	NS	-0.66	PS	3.03	PS
	F_AG_2	0.95	0.98	0.95	1.51	PS	1.34	NS	-0.17	NS
sed V3	F_AG_15	0.93	0.94	0.95	0.10	NS	0.59	PS	0.48	PS
	F_AG_5	0.82	0.88	0.91	5.69	PS	5.69	PS	0.00	PN
	F_AG_17	0.99	0.99	0.98	-0.71	NS	-1.98	NS	-1.27	NS
	F_AG_6	0.67	1.00	1.00	32.91	PS	32.92	PS	0.01	PN
	F_AG_11	0.95	0.98	0.99	0.93	NS	0.83	NS	-0.10	PS
	F_AG_18	0.96	0.97	0.97	1.95	PS	1.97	PS	0.03	PS
sed V4	F_KRM_2	0.59	0.94	0.94	35.28	PS	35.28	PS	0.00	EQ
sed V5	F_KRM_8	0.97	0.99	0.99	1.18	PS	1.06	PS	-0.11	NS
	F_KRM_1	0.92	0.94	0.94	1.48	NS	0.92	NS	-0.56	PS
	F_KRM_2	0.97	1.00	1.00	3.38	PS	3.47	PS	0.09	PS
	F_KRM_10	0.78	0.87	0.89	8.23	PS	8.23	PS	0.00	EQ

Table XIII: Detailed Statistical Analysis for Robustness Study

Subject	Fault ID	$\bar{\tau_R}$	$\bar{\tau_T}$	$\bar{\tau_F}$	$\Delta\bar{E}_{TR}$	H_1	$\Delta\bar{E}_{FR}$	H_2	$\Delta\bar{E}_{FT}$	H_3
flex V2	F_AA_4	0.63	0.63	0.61	1.64	PS	3.23	PS	1.58	PS
	F_AA_5	0.61	0.60	0.63	-2.69	NS	-0.29	PS	2.39	PS
	F_AA_2	0.46	0.50	0.50	-6.49	NS	-2.26	PS	4.24	PS
	F_AA_3	0.46	0.96	0.96	49.37	PS	49.37	PS	0.00	EQ
	F_JR_6	0.48	0.97	0.98	35.62	PS	35.79	PS	0.17	PS
	F_HD_8	0.96	0.95	0.96	-11.36	NS	-7.48	NS	3.87	PS
	F_JR_5	1.00	1.00	1.00	0.10	PS	0.03	NS	-0.06	NS
	F_JR_2	0.97	0.95	0.96	-11.98	NS	-8.11	NS	3.87	PS
	F_JR_3	1.00	0.99	0.99	-6.35	NS	-5.96	NS	0.39	PS
	F_JR_1	1.00	1.00	1.00	0.09	PS	0.03	NS	-0.06	NS
	F_HD_2	0.88	1.00	1.00	12.37	PS	11.24	PS	-1.13	NS
	F_HD_6	0.99	1.00	1.00	0.39	PS	-2.04	NS	-2.43	NS
	F_HD_7	0.95	1.00	1.00	4.90	PS	4.92	PS	0.02	PN
	F_HD_4	0.69	0.69	0.71	1.44	PS	4.87	PS	3.44	PS
flex V3	F_AA_4	0.49	0.50	0.50	-5.26	NS	-1.58	NS	3.68	PS
	F_AA_5	0.99	1.00	1.00	0.55	PS	0.31	PS	-0.24	NS
	F_AA_3	0.34	0.53	0.54	1.80	PS	1.54	PS	-0.25	NS
	F_JR_5	0.80	1.00	1.00	19.67	PS	19.23	PS	-0.44	NS
	F_JR_2	0.99	1.00	1.00	0.52	PS	0.28	PS	-0.24	NS
	F_JR_3	0.79	1.00	1.00	20.26	PS	19.82	PS	-0.44	NS
	F_HD_6	0.98	1.00	1.00	1.19	PS	0.95	PS	-0.24	NS
flex V4	F_AA_7	0.99	0.98	0.99	-1.01	NS	1.75	PS	2.76	PS
	F_AA_1	0.80	0.81	0.87	0.13	PS	0.82	PS	0.69	PS
	F_AA_2	0.99	1.00	1.00	0.70	PS	0.45	PN	-0.25	NS
	F_AA_3	0.51	0.54	0.63	-3.25	NS	4.94	PS	8.19	PS
	F_JR_4	0.99	0.99	1.00	-5.80	NS	-2.75	NS	3.05	PS
	F_JR_2	0.99	1.00	1.00	0.70	NS	-1.60	NS	-2.30	NS
	F_JR_3	1.00	1.00	1.00	0.57	PS	0.20	PS	-0.37	NS
	F_JR_1	0.51	0.54	0.63	-3.83	NS	4.37	PS	8.19	PS
	F_HD_5	0.99	1.00	1.00	0.61	PS	0.24	PS	-0.37	NS
flex V5	F_AA_4	0.92	0.89	0.90	-15.04	NS	-3.77	NS	11.26	PS
	F_JR_2	1.00	1.00	1.00	1.05	PS	0.81	PS	-0.24	NS
grep V2	F_DG_1	0.67	0.66	0.83	-3.35	NS	4.90	PS	8.25	PS
grep V3	F_KP_7	0.49	0.84	0.84	34.91	PS	34.91	PS	0.00	EQ
	F_KP_3	0.62	0.65	0.76	-1.45	PS	7.39	PS	8.84	PS
	F_DG_8	0.42	0.59	0.70	2.02	PS	2.28	PS	0.26	PS
	F_DG_2	0.97	0.96	0.99	-4.11	NS	5.38	PS	9.49	PS
	F_DG_3	0.73	0.75	0.86	2.06	PS	5.49	PS	3.44	PS
grep V4	F_KP_8	0.93	1.00	1.00	6.79	PS	4.73	PS	-2.06	NS
	F_DG_3	0.89	0.82	0.81	-0.90	PS	-1.01	NN	-0.12	NS
	F_KP_6	0.97	0.98	0.90	-0.26	PS	-1.84	NS	-1.58	NS
gzip V2	F_KL_1	0.67	0.73	0.59	-0.52	PS	-0.36	NS	0.16	NS
	F_KL_3	0.93	0.91	0.95	2.92	PS	3.50	PS	0.59	PS
	F_KL_8	0.58	0.66	0.55	0.04	PS	0.11	PS	0.07	PN
gzip V4	F_KL_1	0.50	0.99	0.99	46.53	PS	46.53	PS	0.00	EQ
	F_KL_8	0.90	0.96	0.95	6.92	PS	6.56	PS	-0.35	NS
	F_KP_3	0.53	0.99	0.99	43.85	PS	43.85	PS	0.00	EQ

gzip V5	F_KL_1	0.93	0.96	0.94	5.75	PS	5.41	PS	-0.34	NS
	F_KL_2	0.66	0.89	0.86	12.07	PS	9.09	PS	-2.98	NS
	F_KL_4	0.50	0.93	0.93	35.96	PS	35.96	PS	0.00	EQ
	F_KL_8	0.69	0.96	0.95	26.22	PS	26.22	PS	0.00	EQ
	F_TW_1	0.67	0.72	0.63	1.62	PS	1.21	PS	-0.41	PS
sed V2	F_AG_20	0.62	0.65	0.60	0.60	PS	0.11	PN	-0.49	NS
	F_AG_17	0.34	0.38	0.40	2.44	PS	2.61	PS	0.17	PS
	F_AG_12	0.94	0.95	0.98	0.45	NS	1.56	PS	1.10	PS
	F_AG_19	0.88	0.92	0.94	-3.71	NS	-1.69	NS	2.02	PS
	F_AG_2	0.95	0.98	0.94	1.55	PS	1.32	NS	-0.23	NS
sed V3	F_AG_15	0.93	0.94	0.95	-0.21	NS	0.05	NS	0.26	PN
	F_AG_5	0.82	0.87	0.91	5.39	PS	5.41	PS	0.01	PS
	F_AG_17	0.99	0.99	0.99	-0.39	NS	-3.06	NS	-2.67	NS
	F_AG_6	0.67	1.00	1.00	32.78	PS	32.99	PS	0.20	NS
	F_AG_11	0.95	0.97	0.98	0.88	NS	4.63	PS	3.75	PS
	F_AG_18	0.96	0.98	0.98	2.50	PS	2.53	PS	0.03	PS

Table XIV: Detailed Statistical Analysis of Comparison to Similarity Sorting Approach

Subject	Fault ID	$\bar{\tau_F}$	$\bar{\tau_S}$	$\Delta\bar{E}_{FR}$	$\Delta\bar{E}_{SR}$	H
flex V1	F_AA_6	0.55	0.50	1.88	-11.73	PS
	F_AA_1	0.76	0.97	9.46	24.34	NS
	F_AA_2	1.00	1.00	0.11	0.25	NS
	F_AA_3	0.75	0.50	-0.23	-11.58	PS
	F_JR_4	1.00	1.00	-1.89	-1.91	PN
	F_JR_6	1.00	1.00	-4.36	-2.13	PS
	F_JR_5	1.00	1.00	0.56	0.49	PS
	F_JR_2	0.76	0.97	8.11	22.98	NS
	F_JR_3	1.00	1.00	-1.12	-20.14	PS
	F_HD_3	1.00	1.00	0.53	0.67	NS
	F_HD_1	0.56	0.50	3.26	-8.81	PS
	F_HD_6	0.50	0.98	4.19	43.32	NS
	F_HD_7	0.99	0.94	0.52	-0.01	PS
	F_HD_4	0.56	0.50	3.41	-12.89	PS
	F_HD_5	1.00	1.00	-1.41	-0.23	PS
flex V2	F_AA_4	0.61	0.77	-0.54	34.09	NS
	F_AA_5	0.64	0.52	0.87	17.12	NS
	F_AA_2	0.50	0.50	-2.32	-16.40	PS
	F_AA_3	0.99	0.99	52.50	52.50	EQ
	F_JR_6	0.97	0.97	35.78	35.84	PS
	F_HD_8	0.99	0.54	1.78	-28.51	PS
	F_JR_5	1.00	1.00	-0.01	0.04	PS
	F_JR_2	0.99	0.54	1.16	-29.13	PS
	F_JR_3	1.00	0.97	-3.25	-12.48	PS
	F_JR_1	1.00	1.00	-0.01	0.04	PS
	F_HD_2	1.00	1.00	11.87	12.08	PS
	F_HD_6	1.00	1.00	-5.64	0.08	NS
	F_HD_7	1.00	1.00	4.92	4.88	PN
	F_HD_4	0.71	0.52	5.00	1.46	PS

flex V3	F_AA_4	0.50	0.50	-1.57	-0.67	NS
	F_AA_5	1.00	1.00	0.31	0.44	NS
	F_AA_3	0.54	0.50	1.54	-10.20	PS
	F_JR_5	1.00	1.00	19.21	19.53	NN
	F_JR_2	1.00	1.00	0.28	0.41	NS
	F_JR_3	1.00	1.00	19.79	20.12	NN
	F_HD_6	1.00	1.00	0.95	1.07	NS
flex V4	F_AA_7	0.99	1.00	1.75	1.96	NN
	F_AA_1	0.89	0.81	0.93	-0.49	PS
	F_AA_2	1.00	1.00	0.44	0.42	PS
	F_AA_3	0.63	0.50	4.99	-7.88	PS
	F_JR_4	0.99	0.97	-4.39	-11.69	PS
	F_JR_2	1.00	1.00	-1.33	0.73	NS
	F_JR_3	1.00	1.00	-0.08	0.64	NS
	F_JR_1	0.63	0.50	4.42	-8.45	PS
	F_HD_5	1.00	1.00	-0.05	0.68	NS
flex V5	F_AA_4	0.91	0.52	-3.74	-27.66	PS
	F_JR_2	1.00	1.00	0.81	-18.53	PS
grep V1	F_KP_2	1.00	1.00	-1.77	-8.26	PS
	F_DG_4	0.91	0.76	4.14	0.76	PS
grep V2	F_DG_1	0.83	0.55	4.88	-0.89	PS
grep V3	F_KP_7	0.84	0.84	35.42	35.42	EQ
	F_KP_3	0.76	0.51	7.32	-8.77	PS
	F_DG_8	0.70	0.51	0.67	-0.34	PS
	F_DG_2	0.99	0.70	5.52	-11.96	PS
	F_DG_3	0.85	0.56	4.29	-6.07	PS
grep V4	F_KP_8	1.00	1.00	4.58	6.17	NN
	F_DG_3	0.62	0.61	-9.74	-9.93	PS
	F_KP_6	0.90	0.88	-1.84	-2.21	PS
gzip V1	F_KL_2	0.82	0.81	11.62	11.10	PS
	F_KL_6	0.42	0.42	0.34	0.34	EQ
	F_KP_10	0.99	0.99	40.25	40.25	EQ
	F_KP_11	0.83	0.84	-3.35	-4.22	PS
	F_KP_9	1.00	1.00	19.54	20.33	NN
	F_TW_3	1.00	1.00	10.43	10.39	PN
	F_KP_1	0.97	0.98	2.06	2.86	NS
gzip V2	F_KL_1	0.62	0.56	0.34	0.44	NS
	F_KL_3	0.90	0.87	1.93	0.27	PS
	F_KL_8	0.65	0.62	-0.03	-0.18	PS
gzip V4	F_KL_1	0.99	0.99	46.53	46.53	EQ
	F_KL_8	0.94	0.94	6.53	6.62	NN
	F_KP_3	0.99	0.99	43.85	43.85	EQ
gzip V5	F_KL_1	0.95	0.94	5.42	5.50	NN
	F_KL_2	0.87	0.85	9.66	7.42	PS
	F_KL_4	0.91	0.91	34.90	34.90	EQ
	F_KL_8	0.95	0.95	25.75	25.75	EQ
	F_TW_1	0.86	0.53	-1.22	-5.90	PS

sed V2	F_AG_20	0.63	0.53	1.13	-7.25	PS
	F_AG_17	0.41	0.40	2.99	3.26	NS
	F_AG_12	0.97	0.83	1.85	-6.63	PS
	F_AG_19	0.85	0.73	-0.66	-6.73	PS
	F_AG_2	0.95	0.93	1.34	1.36	PS
sed V3	F_AG_15	0.95	0.92	0.59	0.83	PS
	F_AG_5	0.91	0.89	5.69	5.70	NN
	F_AG_17	0.98	1.00	-1.98	1.01	NS
	F_AG_6	1.00	1.00	32.92	33.35	NS
	F_AG_11	0.99	0.93	0.83	-0.17	PS
	F_AG_18	0.97	0.97	1.97	1.97	NN
sed V4	F_KRM_2	0.94	0.94	35.28	35.28	EQ
sed V5	F_KRM_8	0.99	0.98	1.06	1.10	NN
	F_KRM_1	0.94	0.93	0.92	2.99	NS
	F_KRM_2	1.00	1.00	3.47	2.96	PS
	F_KRM_10	0.89	0.86	8.23	8.23	EQ