Assessing Test Case Prioritization
on Real Faults and Mutants

Qi Luo, Kevin Moran, Denys Poshyvanyk
Department of Computer Science
College of William & Mary
Williamsburg, VA
Email: {qluo,kpmoran,denys}@cs.wm.edu

Massimiliano Di Penta
Department of Engineering
University of Sannio
Benevento, Italy
Email: dipenta@unisannio.it

Abstract—**Test Case Prioritization (TCP) is an important component of regression testing, allowing for earlier detection of faults or helping to reduce testing time and cost. While several TCP approaches exist in the research literature, a growing number of studies have evaluated them against synthetic software defects, called *mutants*. Hence, it is currently unclear to what extent TCP performance on *mutants* would be representative of the performance achieved on *real faults*. To answer this fundamental question, we conduct the first empirical study comparing the performance of TCP techniques applied to both real-world and mutation faults. The context of our study includes eight well-studied TCP approaches, 35k+ mutation faults, and 357 real-world faults from five Java systems in the Defects4J dataset. Our results indicate that the relative performance of the studied TCP techniques on mutants may not strongly correlate with performance on real faults, depending upon attributes of the subject programs. This suggests that, in certain contexts, the best performing technique on a set of mutants may not be the best technique in practice when applied to real faults. We also illustrate that these correlations vary for mutants generated by different operators depending on whether chosen operators reflect typical faults of a subject program. This highlights the importance, particularly for TCP, of developing mutation operators tailored for specific program domains.**

I. INTRODUCTION

Regression Testing is defined as the process of running a collection of compact tests, aimed at testing discrete functionality that underlies a software program, when that program changes in an evolutionary context. This type of testing allows for the discovery of software *regressions*, faults that cause an existing feature to cease functioning as expected. Regression test suites tend to be large for complex projects and are often run every time code is checked into a repository, leading to longer than desired testing times in practice. For example, Google has reported that, across its code bases, there are more than twenty code changes per minute, with 50% of the files changing per month, leading to long testing times [13], [59]. To limit the number of test cases to execute, *Test Case Prioritization* (TCP) has been developed. TCP aims to prioritize test cases in a test suite to detect regressions more quickly or reduce testing time.

A large body of research has been dedicated to designing and evaluating TCP techniques [3], [6], [7], [11], [14], [15], [37], [44], [51], [58], [72], [73], [75], [76]. Such evaluations have typically been conducted by comparing the Average Per-

centage of Faults Detected (APFD) metric, or cost cognizant APFDc metric [20], [24], [33], [60], for different techniques.

While in principle the evaluation of TCP techniques requires the availability of real program faults, very often the lack of real fault data has encouraged researchers to use artificial faults, called *mutants*, each comprised of a simple syntactic change to the source code [17], [29], [30], [45], [78]. The underlying assumption of such evaluations is that there is a strong correlation between prioritized sets of test cases that kill high numbers of mutants and sets that detect a high number of real faults. This assumption raises key questions: *How well do TCP techniques perform on real faults?*; *Is the performance of TCP techniques on mutants representative of their performance on real faults?*; *What properties of mutants affect the representativeness of this performance?*

Previous studies have examined the relationship between real faults and mutants in order to understand the applicability of mutants in software testing. In particular, these studies have investigated: (i) whether mutants are as difficult to detect as real faults [1], [2]; (ii) whether mutant detection correlates with real fault detection [12], [35], [54]; (iii) whether mutants can be used to guide test case generation [63]; and (iv) whether tokens contained in patches for real-world faults can be expressed in terms of mutants [27].

Despite the studies outlined above, no previous study has investigated whether mutants are representative of real faults in the context of evaluating TCP approaches. Indeed, such evaluations aim to measure the *rate* at which large sets of mutants are detected by *prioritized sets of test cases* according to APFD(c), fundamentally differing from the experimental parameters of the studies outlined above. For example, Just *et al.* [35] focused on the relationship between real faults and mutants measured by the ability of fault detection for a *whole test suite*, which may not imply a similar relationship in terms of APFD(c) values. Furthermore, TCP approaches have not previously been extensively evaluated in terms of their capabilities for detecting real-world faults, indicating that the practical performance of these techniques is largely unknown.

To address this research gap, we perform an extensive empirical study to understand the effectiveness of TCP techniques when evaluated in terms of real faults, and examine whether mutants are representative of real faults when evaluating

TCP performance. We implemented eight well-studied TCP techniques and applied them on (i) a dataset of real faults, Defects4J [34], containing 357 real faults from five large Java programs, and (ii) over 35k+ mutants seeded using Pit [57].

The results of our study show that for the subject programs studied, mutation-based performance of TCP techniques, as measured by APFD(c), tends to *overestimate* performance by $\approx 20\%$ on average when compared to performance on real faults *unless trivial and subsumed mutants are removed*. When trivial and subsumed mutants are controlled for, performance measured according to the resulting mutants tends to slightly *underestimate* performance by $\approx 3\%$ on average compared to real faults. Furthermore, our results indicate that the performance of TCP techniques *relative to one another* on mutants, may not correlate to relative performance on real faults. However, the above findings tend to vary depending on the subject program. Additionally, we find that, as a whole, static TCP techniques tend to perform slightly better than dynamic techniques on real faults, but differences are not statistically significant. Finally, when examining the fault sets in terms of mutant coupling and operator type, we found that the representativeness of mutants (compared to real faults) varies *within* programs. This suggests that different mutant combinations could be derived to more closely resemble likely real faults in a program.

To the best of our knowledge, this is the first comprehensive empirical study that evaluates the performance of eight well-studied TCP techniques on a large set of *real faults* and compares the results to mutation-based performance in order to determine the representativeness of mutants in this context. The results of this study, and the code utilized are available in an online appendix to facilitate reproducibility [47].

II. BACKGROUND & RELATED WORK

In this section we formally define the TCP problem, introduce the studied TCP techniques, and discuss related work.

A. TCP Problem Formulation

TCP is formally described by Rothermel *et al.* [61] as finding a prioritized set of test cases $T' \in P(T)$, such that $\forall'', T'' \in P(T) \land T'' \neq T' \Rightarrow f(T') \geq f(T'')$, where $P(T)$ refers to the set of permutations of a given test suite T, and f refers to a function from $P(T)$ to real numbers. While there are many types of existing TCP techniques [16], [23], [29], [38], [65], [79], one common dichotomous classification, *static* [31], [42], [80] and *dynamic* techniques [18], [19], [22], [41], [53], [64], [71], [78], relates to the type of information used to perform the prioritization. Static approaches utilize information extracted from source and test code and dynamic techniques rely on information collected at runtime (*e.g.,* coverage information per test case) to prioritize test cases [46]. Additional classifications exist, such as the distinction between *white-box* and *black-box* techniques [30]. The techniques that require source code of subject programs (or information extracted from source code) are typically classified as *white-box* techniques [36]. Conversely, those that

only require program input or output information are classified as *black-box* techniques. Approaches that only require test-code have been classified as black-box [30], [46], however, in this paper we more accurately refer to these techniques as *grey-box* since they require access to test code with references to the underlying program. There are also other approaches [32], [62], [69] that use other types of information to perform the prioritization, such as code-changes and requirements, that do not fall neatly into these categories.

B. Studied TCP Techniques

In the context of our empirical investigation, we selected eight well-studied white and grey-box TCP techniques as they provide a effective means of comparing the performance of widely disseminated techniques on real faults to their reported performance on mutants in prior work. Furthermore, the original papers that introduce our studied approaches have been collectively cited over $1.4k$ times, indicating their importance within the research community. Together we consider four dynamic white-box techniques that utilize run-time code coverage for prioritization [33], [43], [60], two static gray-box techniques that operate only on test code [42], [66], and two static white-box approaches that use call-graph information [80]. In the remainder of this subsection we give an overview of these eight techniques (where the *Greedy* and *Call-Graph-based* techniques have two variants). Implementation details for these techniques are given in Section III-D.

1-2) Greedy Techniques (Total & Additional) The traditional greedy TCP techniques use two strategies, *total* and *additional*, to prioritize test cases using coverage information [60]. The total strategy always prioritizes test cases with highest total coverage. Conversely, the additional strategy prioritizes tests that cover the most *additional* code compared to the prioritized set.

3) Adaptive Random Testing The Adaptive Random Testing (ART)-based TCP technique was proposed by Jiang *et al.* [33]. ART approaches are used to spread test inputs evenly over the entire input domain of a program in order to detect failures more quickly as compared to random testing [10]. Initially, this technique randomly generates a candidate set of test cases and randomly selects a test case. Then, a new candidate set is iteratively generated in a random fashion and the test case that is farthest away from the already-prioritized set, in terms of coverage-based Jaccard distance, is added. In our study, we choose the *minimum* distance to measure the distance between each test case and the set of prioritized test cases, since it has been shown to be the most effective [33].

4) Genetic Algorithm-based Technique Li *et al.* proposed a set of approaches that prioritize test cases using search-based techniques, which are able to avoid producing sub-optimal results that find only local solutions within the test input space [43]. They used two meta-heuristics, namely hill climbing and Genetic Algorithms (GAs), and used coverage information as a prioritization objective. Their experimental results show that GAs perform better than hill climbing, thus, we utilize the GA-based approach in this study.

5-6) Call-Graph-based Techniques (Total & Additional)
Zhang *et al.* proposed an approach to prioritize test cases based on call-graph information instead of coverage [80]. The approach builds a call-graph for each test case, and uses the information to measure the testing abilities for all test cases. The test cases covering more methods are favored in the prioritization scheme. Similar to greedy techniques, there are two variations: *total* and *additional*.

7) String-based Technique Ledru *et al.* proposed a TCP approach that uses textual information extracted from test cases [42]. The underlying idea is that a set of textually diverse test cases may have a better chance at uncovering more faults than a textually similar set. This technique treats each test case as a string, and introduces four types of string distances to estimate the difference between a pair of test cases, prioritizing test cases that are furthest from the prioritized set in a pairwise manner. Experimental results show that Manhattan distance performs best in terms of fault detection [42], and thus, we use this setting in our study.

8) Topic-Model-based Technique This technique aims to utilize the textual information in test cases, such as identifiers and comments to build topic models for prioritization [66]. The technique relies on topic models to approximate the abstract functionality of each test case, and estimates the distances between topics of test cases to favor those that are able to test different high-level functionalities of the program. Similar to the string-based TCP technique, the topic-model-based technique uses Manhattan distance to measure the distance between a single test case and the prioritized set. The test furthest away from the prioritized set is chosen during each iteration to maximize diversity. In our study, we implement this approach as proposed by Thomas *et al.* [9], [66].

C. Threats to the Validity of Mutation-Based TCP Performance Evaluations

There is a large body of work that has addressed the problem of Test Case Prioritization [21], [32], [50], [62], [74]. A common link between the evaluations of various techniques is the utilization of *fault-detection rates*, typically in terms of the APFD(c) metrics [30], [45], [67]. However, due to the fact that finding and extracting real-world faults from software is an intellectually intensive and laborious task, real faults are rarely used when evaluating testing related research [34], [35], including existing work on TCP. Instead, *mutation analysis* can be utilized. During mutation analysis small, automatically-generated syntactic faults are seeded throughout subject programs according to a set of clearly-specified *mutation operators*. Then APFD(c) values are calculated according to the number of mutants that are detected (*i.e.,* killed) by prioritized test cases. To make results of such an evaluation generalizable to realistic settings, APFD(c) fault-detection rates for mutants should correlate with detection rates of *real faults*. Unfortunately, in the context of the typical methodology used to evaluate TCP techniques, the relation between performance on mutants and real faults is not well understood. This gives rise to the potential for threats to

the validity of TCP evaluations relating to (i) the overall performance of TCP techniques on mutation vs. real faults (*is performance over or under-estimated?*); (ii) the relative performance or performance correlation of real vs. mutation faults (*does a mutation based-analysis properly illustrate the most effective technique on real faults?*); and (iii) the impact that different properties of mutants and programs have on mutation-based performance of TCP.

D. Studies Examining the Relationship Between Mutants and Real Faults

While our study is the first to examine the representativeness of mutation faults in terms of real faults as it pertains to the domain of TCP, we are not the first to investigate this relationship in a general sense. Daran and Thévenod-Fosse [12] performed the first empirical comparison between mutants and real faults, finding that the set of errors and failures they produced with a given test suite are quite similar. Andrews *et al.* [1], [2] compared the fault detection capability of test-suites on mutants, real-faults, and hand-seeded faults, reaching two conclusions. First, mutants (if carefully selected) can provide a good indication of a test suite's ability to detect real faults. Second, the use of *hand-seeded* faults can produce an underestimation of a test suite's fault detection capability. Gopinath *et al.* conducted an empirical study that explored the characteristics of a large set of changes and bug-fixes and how these related to mutants [27].

Just *et al.* studied whether a test suite's ability to detect mutants is coupled with its ability to detect real faults, controlling for code-coverage [35]. Their results indicate that mutant detection correlates more closely with real fault detection than with code coverage. Additionally, their study also provided suggestions regarding how mutant taxonomies can be improved to make them more representative of real faults through examination of how mutants are coupled to real faults. Just *et al.* introduced a valuable dataset of 357 real faults across five Java programs in an artifact called Defects4 [34], which we utilize in this paper. Shamshiri *et al.* conducted an empirical study of automatic test generation techniques to investigate their ability to detect real faults in the Defects4J [63]. Finally, Chekham *et al.* conducted a study examining how mutation, statement and branch coverage correlate to fault revelation [8]. They found that *strong* mutation testing has the highest fault revelation capability and fault revelation only tends to significantly increase once high coverage levels are attained.

While the aforementioned research has investigated several aspects of the relationship between real-faults and mutants, there is no prior work examining this relationship in the context of typical TCP evaluation methodologies. Thus, it is unclear whether results from these studies hold in the context of TCP, as the experimental settings in such cases (and hence in our study) *fundamentally differ* from past work. These differences manifest in the typical methodology used to assess the effectiveness of TCP techniques, which involves seeding mutants into a *single* version of a program and calculating fault-detection *rates* according to the APFD(c) metrics.

Subject Programs	#Real	#Detected	#Subsuming	#All
JFreeChart	26	32,790	1,796	102,629
Closure Compiler	133	82,572	9,731	111,826
Commons Maths	106	80,059	5,016	113,680
Joda-Time	27	24,555	3,066	34,147
Commons Lang	65	25,173	2,129	31,214
Total	357	245,767	21,738	393,496

In order to more clearly illustrate the need for this study, it is useful to consider how our experimental setup differs from prior work such as Just et. al's [35]. The most analogous study to ours that Just et. al conduct is that which measures the correlation between a test suite's ability to detect real faults and the test suite's mutation score, *without considering the impact of test case ordering within a suite*. In contrast, our study is, in essence, aimed at investigating how test case ordering (and prioritization) can impact the representativeness of mutants in terms of fault detection *rates*. This nuance has important implications for TCP because the *distribution* of faults across a program can impact the rate at which a test suite detects faults depending on test case ordering. For example, mutants could be distributed throughout a program in a more uniform manner than real faults, causing differences in the effectiveness of the prioritization schemes of different TCP techniques. Previous studies that only examine test suites in their entirety were not capable of exploring this phenomenon.

III. EMPIRICAL STUDY

The *goal* of this study is to analyze the extent to which mutation analysis can support the evaluation of Test Case Prioritization, as opposed to using data from real faults. The study *context* consists of data from five Java open source projects (Defect4J [34], [35]), mutants generated by PIT [57], and the eight TCP techniques described in Section II.

A. Research Questions (RQs):

The study aims at answering the following three **RQs**:

- **RQ$_1$:** How *effective* are TCP techniques when applied to detecting real faults?
- **RQ$_2$:** Is the performance of TCP techniques on mutants *representative* of performance on real faults?
- **RQ$_3$:** How do the *properties* of real faults and mutants affect the performance of TCP techniques?

B. Study Context

In order to properly evaluate the performance of TCP approaches when applied to detecting real-world faults, our study requires a well understood set of verified, real program faults preferably containing coupling information between real faults and mutants. To satisfy this criteria, we utilize the Defects4J [34] dataset, which contains 357 real faults extracted from five Java subject programs, listed in Table I, and has been utilized in past studies [35], [63]. Defects4J isolates the real faults from the version control and bug tracking system of each subject program. For each isolated fault there exists a faulty program version and a corresponding fixed version. Table I shows the distribution of isolated real faults across the five subject programs, with Closure Compiler and Commons Maths representing the largest numbers of faults.

For each real fault (*i.e.,* faulty version), Defects4J provides a test suite including at least one test case that is able to trigger the fault but pass successfully in the corresponding fixed version. Additionally, it provides the code locations (*i.e.,* method and class names) that were modified to fix the fault. Test cases were extracted at test-method granularity rather than the test-class granularity, as TCP techniques have been shown to perform best under such experimental settings [46].

The *primary goal* of this study is to determine how well mutation-based measures of TCP effectiveness reflect the performance of these techniques on real faults. More generally, we aim to answer the following question: "If one prioritizes test cases using mutants, would this prioritized set likely be as effective on real faults?" In order to explore this question we seeded mutants using the PIT mutation tool [57] with all built-in operators enabled. During this seeding process we excluded mutants that cannot be killed (*i.e.,* triggered the test case to fail), by any test case in the existing JUnit suites for two reasons: i) to mitigate a potential threat to validity from equivalent mutants, ii) they do not affect our studied metrics according to the definitions of APFD(c) as defined in Section III-C. The number of detected mutants and the total number of seeded mutants are shown in columns 3 and 5 of Table I respectively (see online appendix for further information [47]).

To perform mutant seeding that allows for comparison between real faults and mutants, for each (real) *faulty* version of a program in Defects4J, we create one mutated program instance by seeding a randomly selected mutant into the *latest* corresponding program version, and then repeat this process 100 times (*e.g.,* for Closure: 133 versions with real faults × 1 mutant × 100 instances = 13,300 total mutants). For instance, taking the JFreeChart program as an example, one mutant was randomly selected from the set of 32,790 mutants able to be detected by at least one test case, until a set of 26 mutant versions of JFreeChart were accumulated (matching the number of real faulty versions). This procedure is then repeated 100 times. This results in 100 groups of 26 mutants, or 2,600 mutants being evaluated for JFreeChart in total. We repeat the seeding process 100 times due to the fact that the selected mutant is a random variable, and we aim to provide a reliable statistical analysis and the best possible approximation for TCP evaluations from prior work. In initial experiments, excluded due to space limitations (but included in our online appendix [47]), we computed the APFD(c) values using five randomly seeded mutants per instance (instead of one), following the settings of previous work [29], [45], [46], [50], [77]. The results for this analysis generally agree with the presented results, and thus we do not expect that the number of mutants per instance will dramatically impact findings.

The intention for choosing these experimental settings is to evaluate whether past mutant-based *methodologies* measuring TCP efficacy hold for real faults. Thus, we seeded mutants

TABLE II
STUDIED TCP TECHNIQUES.

Type	Tag	Description
Static	TCP_{cg-tot}	Call-graph-based (total strategy)
	TCP_{cg-add}	Call-graph-based (additional strategy)
	TCP_{str}	The string-distance-based
	TCP_{topic}	Topic-model-based
Dynamic	TCP_{total}	Greedy total (statement-level)
	TCP_{add}	Greedy additional (statement-level)
	TCP_{art}	Adaptive random (statement-level)
	TCP_{search}	Search-based (statement-level)

in accordance with past studies [46], [50], [62], [77], [78], applying mutation analysis to the *latest* version of each subject program. This means that test suites from each faulty program version in Defects4J are prioritized for both each faulty version and the latest version (with mutants injected). As Section V describes, this makes for a reasonable comparison due to the fact that past work measuring the impact of software evolution on TCP efficacy has shown that mutation-based performance remains consistent when applying test cases from an earlier program version to both the earlier and later versions [45].

Furthermore, two recent works outline the potential impact of trivial/subsumed mutants for mutation-based analysis [30], [55]. Thus, we investigate our RQs both with and without trivial and subsumed mutants removed from the set of seeded mutants. We follow the methodology defined in prior work [55], which is the best approximation for the removal of subsuming mutants, as this has been proven an undecidable problem. The number of subsuming mutants is shown in Table I. Thus we will discuss results in terms of two different mutant *sets*: the *full mutant set*, and the *subsuming mutant set*.

C. Methodology

1) RQ$_1$: TCP Effectiveness on Real Faults: The *goal* of this research question is to investigate the performance of TCP techniques when they are applied to detect real faults. We first ran these eight TCP techniques on 357 Defects4J program versions containing real faults to obtain ranked lists of test cases for each faulty version. The tests are run at the *test-method* level, since past work has shown method-level yields more effective TCP results [45], [46]. Then, to measure the effectiveness in terms of fault detection for each studied technique, we calculated two well-accepted metrics, the Average Percentage of Faults Detected (APFD) [21], [60] and its cost cognizant counterpart APFDc [20], [24]. Formally, APFD is defined as follows: Let T be a test suite and T' is a permutation of T, the APFD value for T' is given by

$$APFD = 1 - \frac{\sum_{i=1}^{m} TF_i}{n \times m} + \frac{1}{2n} \qquad (1)$$

where n is the number of test cases in T, m is the number of faults, and TF_i is the position of the first test case in T' that detects fault i. Intuitively, the higher the APFD value, the higher the rate of fault detection by the prioritized test cases. In order to derive a more holistic understanding of the relationship between TCP performance on real faults and mutants we also consider APFDc. This metric takes both execution cost and fault severity into account. Since there is no clearly-defined nor widely-used method for estimating fault

severity, we consider severity to be the same for all faults. Therefore, in the context of this study APFDc reduces to the following formal definition:

$$APFDc = \frac{\sum_{i=1}^{m} \left(\sum_{j=TF_i}^{n} t_j - \frac{1}{2} t_{TF_i} \right)}{\sum_{j=1}^{n} t_j \times m} \qquad (2)$$

where t_j is the execution cost for the test case ranked at position j in the ranked test suite. Intuitively, APFDc, as defined above, will be higher for prioritized test suites that both find faults faster and require less execution time. Since we study five subjects, each with a different number of real faults, we computed the average APFD(c) values of the different versions across all five systems to understand the effectiveness of each studied approach. Additionally, to statistically analyze the differences between TCP techniques in terms of APFD and APFDc values, we perform an Analysis of Variance (ANOVA) and a Tukey Honest Significant Difference (HSD) test [68] on the average APFD(c) values across the five subjects. The ANOVA analysis is used to test whether there are statistically significant differences between the performance of TCP techniques when applied to real faults versus when applied to mutants. The Tukey HSD test classifies the TCP techniques into different groups based on their performance in terms of APFD(c) values. For both statistical procedures we consider a significance level $\alpha = 0.05$.

2) RQ$_2$: Representativeness of Mutants: The *goal* of RQ$_2$ is to understand whether mutants are representative of real faults in the evaluation of TCP techniques. Thus, we applied mutation analysis according to the description given in Section III-B. As mentioned in Section III-B, two sets of mutants are examined, the *full mutant set* and the *subsuming mutant set*. Then, we ran all studied TCP techniques on these sets of mutant versions and calculated the average APFD(c) values (see Equations 1 and 2) across all 100 mutant groups for all five subject programs, in order to examine the mutant-based performance of our studied TCP techniques.

At this point, we are able to evaluate the effectiveness of TCP techniques in terms of both real fault and mutant detection according to APFD(c), which we refer to as the *absolute performance*. In addition to the absolute performance, we are also concerned with how different techniques perform relative to one another across different fault sets, which we refer to as the *relative performance*. If the ranking of TCP techniques from most to least effective is similar when measured across both mutants and real faults, the relative performance would be *positively correlated*, otherwise it would be *negatively correlated*. To calculate this, we utilize the Kendall rank correlation coefficient τ [40] to measure the relationship. This correlation metric is commonly used to measure the ordinal association between two quantities [39] and has been widely used in the area of software testing for such purpose [26], [56], [77]. Consider the APFD values of real fault detection and mutant detection across all studied techniques as a set of pairs (R, M), where R is the APFD/APFDc values of real fault detection and M is the APFD/APFDc values of mutant

detection. Any pair of (r_i, m_i) and (r_j, m_j) (APFD values for TCP$_i$), where $i \neq j$, are concordant if $r_i > r_j$ and $m_i > r_j$ or if $r_i < r_j$ and $m_i < m_j$. They are discordant if $r_i > r_j$ and $m_i < m_j$ or if $r_i < r_j$ and $m_i > m_j$ [52]. The Kendall τ rank correlation coefficient is formally defined as the ratio of the number of concordant pairs less the number of discordant pairs and the total number of pairs. Thus, its value ranges from -1.0 to 1.0. Results closer to 1.0 indicate the observations of two variables have similar rank (*e.g.,* which in the context of this study translates to similar rates of fault discovery), whereas when it is closer to -1.0 when the observations of two variables have dissimilar ranks (*e.g.,* suggesting a negative correlation between fault discovery rates). Following previous work [26], we use the Kendall τ_b statistic as it accounts for ties and does not require a linear relationship.

3) RQ$_3$: Effects of Fault Properties: The *goal* of this research question is to understand how different *properties* of faults impact two phenomena in the context of TCP: (i) the performance of techniques, and (ii) the utility of mutants as a proxy for real faults (*i.e.,* performance correlation).

The *first* fault property we investigated is **the level of coupling between real faults and mutants**. In order to determine the level of coupling for real faults to mutation operators, we utilize Just *et al.*'s previous work [35], which classified the 357 real faults from the Defects4J dataset into four coupling levels: (i) those coupled with mutants (denoted in the study using the keyword "Couple"), (ii) those requiring stronger mutation operators (denoted as "StrongerOP"), (iii) those requiring new mutation operators (denoted as "newOP"), and (iv) and those not coupled with mutants (denoted as "Limitation") Formally speaking, a real fault (*i.e.,* a complex fault) is coupled with a set of mutants (*i.e.,* simple faults) if a test case that detects all the mutants also detects the real fault [35]. We contacted the authors to obtain this classification scheme, which includes 262 real faults coupled to mutants, 25 real faults requiring stronger mutation operators, seven real faults requiring new mutation operators, and 63 real faults not coupled to mutants. In order to examine the impact that fault coupling has on *performance*, we pruned our initial dataset from the previous two research questions and calculated APFD(c) values for each coupling level of real faults and for all mutants considered in RQ$_2$ for each subject program. In order to examine the *correlation* between these APFD(c) values, we again utilize the Kendall τ_b coefficient.

The *second* fault property we examined is the **mutation operator type**. Intuitively, this investigation should help shed light on which mutation operators are more representative of real faults for our studied subject programs in the context of TCP performance. We classified mutants based on their corresponding operators, that is, the mutation faults that are generated by the same mutation operator are classified into the same group. We consider the 15 built-in operators in PIT [57], and for each subject program classified all mutant versions into groups according to these operators. For each subject program and operator type, we then randomly sampled from these groups until we had a set of faults corresponding to

the number of real faults existent in each subject program respectively. We then repeat this process 100 times. If there are not enough mutants to create 100 groups, we repeat this process until we exhaust the mutants. In the end, for each subject program, we derive 100 mutant groups for each type of operator (given enough mutants), with each group containing the same number of mutants (all of the same operator type) as real-faults for the subject. To understand the impact that different types of mutation operators have on the *performance* of TCP techniques, we calculated the APFD(c) values based on the new groups of mutants, and then used the Kendall τ_b coefficient to understand the *correlation* between the APFD(c) values calculated in terms of real faults and mutants. We also explored the effects of mutant locations, however, we found no significant trends. Thus, we forgo discussion of these results and point interested readers to our attached appendix.

D. Experiment Tools and Hardware

We reimplemented all studied TCP techniques in accordance with the technical descriptions in their corresponding papers (see Sec. II). Three of the authors, and an external expert on TCP, carefully reviewed the code, ensuring the reimplementation is reliable. To collect coverage information, we used the ASM bytecode manipulation and analysis toolset [4]. In our empirical study, we chose to use statement-level coverage information, as this allows for optimal performance of TCP techniques [46]. Furthermore, we utilize JDT [25] to extract textual information for each test method, which is used by string-based and topic-based approaches. Specifically for the topic-based approach, we use Mallet [49] to build a latent Dirichlet allocation (LDA) topic model [5] for each test case, after pre-processing the textual information (*e.g.,* splitting, removing stop words and stemming). Following previous research [46], we use WALA [70] to build RTA call graphs [28] for each test method and traverse each call graph to obtain its static coverage to implement the TCP_{cg} techniques.

The experiments were carried out on eight servers with 16, 3.3 GHz Intel Xeon E5-4627 CPUs and 512 GB RAM, and one server with eight Intel X5672 CPUs and 192 GB RAM.

IV. RESULTS

In this section, we describe the results of our empirical study. Furthermore, we provide an online appendix with additional results [47].

A. RQ$_1$: TCP Effectiveness on Real Faults

The values of the APFD(c) metrics and the results of the Tukey HSD test for *real* faults are reported at the top of Table III. From these experimental results we make the following observations. First, for real faults, all techniques tend to perform better when measured by APFD as compared to APFDc. This is not surprising, and it is due to the incorporation of execution cost. For some techniques, in particular TCP_{total} and TCP_{cg-tot}, the differences between APFD and APFDc are comparatively larger. This observation is most likely due to the fact that these techniques always prioritize test cases

TABLE III
AVERAGE APFD & APFDc VALUES FOR ALL EIGHT TCP TECHNIQUES, FOR REAL, FULL MUTANT AND SUBSUMING MUTANT FAULT SETS, ACROSS ALL SUBJECT PROGRAMS. ADDITIONALLY, THE GROUPING RESULTS FOR THE TUKEY HSD TEST ARE SHOWN IN CAPITALIZED LETTERS (*e.g.,* AB). S.MUTANTS REFERS TO SUBSUMING MUTANTS.

Faults	Static Techniques								Dynamic Techniques							
	TCP_{cg-tot}		TCP_{cg-add}		TCP_{str}		TCP_{topic}		TCP_{total}		TCP_{add}		TCP_{art}		TCP_{search}	
	APFD	APFDc	APFD	APFDc	APFD	APFDc	APFD	APFDc	APFD	APFDc	APFD	APFDc	APFD	APFDc	APFD	APFDc
Real	0.594	0.480	0.597	0.591	0.696	0.594	0.7	0.635	0.61	0.419	0.583	0.454	0.657	0.677	0.6	0.556
	A	BC	A	ABC	A	ABC	A	AB	A	C	A	C	A	A	A	ABC
Full Mutant	0.743	0.598	0.818	0.835	0.834	0.788	0.832	0.802	0.757	0.549	0.897	0.829	0.8	0.841	0.784	0.725
	B	BC	AB	A	AB	AB	AB	A	B	C	A	A	AB	A	B	ABC
S.Mutant	0.561	0.407	0.612	0.639	0.620	0.572	0.612	0.570	0.534	0.305	0.664	0.565	0.622	0.671	0.578	0.508
	AB	BC	AB	A	AB	AB	AB	AB	B	C	A	AB	AB	A	AB	ABC

TABLE IV
RESULTS OF THE ANOVA ANALYSIS AND THE KENDALL τ_b COEFFICIENT FOR THE OVERALL APFD(C) VALUES SHOWN IN TABLE III.

Faults	ANOVA p-value		τ_b	
	APFD	APFDc	APFD	APFDc
Real	0.011	3.22e-4	-	-
Mutant	8.02e-4	3.77e-5	0.143	0.571
S.Mutant	0.011	1.38e-4	-0.071	0.643

with higher coverage first, leading to longer execution costs for the top test cases, in turn leading to lower APFDc values.

Second, the static TCP techniques perform slightly better overall as compared to dynamic TCP techniques for both APFD and APFDc metrics, but these differences are not statistically significant. To determine whether static techniques outperformed dynamic ones to a statistically significant degree, we performed a Wilcoxon signed rank test across the raw APFD(c) values achieved by the collective sets of static and dynamic techniques across all subject systems. Results indicate that while static techniques do generally outperform dynamic techniques, the differences are not statistically significant, and the Cliff's delta effect sizes are negligible to small. We provide complete analysis results in our appendix [47].

Third, the TCP_{add} technique does not outperform the TCP_{tot} strategy, contradicting findings from past studies where the TCP_{add} has been shown to perform best overall [21], [46], [61]. Fourth, the results of the Tukey HSD test suggest that for APFD, the performance of the TCP programs does not vary in a statistically significant manner. However, for APFDc, we found statistically significant differences across techniques. It should be noted that the results of the statistical tests are derived from a smaller dataset as compared to the traditional method of using thousands of mutation faults, due to the number of faults included in Defects4J.

B. RQ$_2$: Representativeness of Mutants

The values of the APFD(c) metrics for mutants across the different TCP techniques are also shown in Table III. From this data, we make several notable observations. First, the APFD and APFDc metrics calculated using the full mutant set generally tend to *overestimate* absolute performance compared to real faults by $\approx 20\%$ on average. This finding is relevant, as it implies that mutation-based evaluations measuring the absolute performance of TCP techniques that do not control for subsumed and trivial mutants tend to overestimate *real-world* absolute performance of these techniques. Conversely, there is a slight *underestimation* ($\approx 3\%$ on average) for the APFD(c) values calculated using the subsuming mutant set

when compared to real faults. Moreover, the studied TCP techniques perform differently across different fault sets, with the absolute performance of techniques on both the full mutant set and subsumed mutant set differing from absolute performance on real faults. This is a significant finding as it suggests that, according to results for our set of five subject programs, *a TCP approach that performs well according to mutation analysis may not exhibit the same performance on a set of real faults for the same program(s)*. As we will discuss later, this points to the need for *careful selection of mutants* when performing a mutation-based evaluation of TCP techniques in order to ensure that the results obtained also hold for *real faults*. Additionally, removing subsumed mutants should prove beneficial in practice to avoid performance overestimation.

While the absolute performance in terms of APFD and APFDc may not be similar when comparing performance on mutants to performance on real faults, it is possible that performance is *positively correlated* between the different fault sets. That is, the performance of TCP techniques on real faults *relative to each other* is consistent across different fault sets. To measure this, we examine the Kendall τ_b correlation coefficient across the results from the two types of faults (shown in Table IV). Note that two rankings are considered as independent to one another when τ_b is closer to zero. Our results indicate a very weak positive correlation between mutants and real faults when examining APFD (τ_b=0.143 for all-killed mutants and for τ_b=-0.071 subsuming mutants) and a medium to strong positive correlation when considering APFDc (τ_b=0.571 for all-killed mutants and for τ_b=0.643 subsuming mutants). Removing subsumed mutants does not impact the correlation results. This observation implies that, in general, a mutation-based TCP performance evaluation carried out in terms of APFDc will more strongly correlate to performance in terms of APFDc on real faults. However, when relying on a mutation-analysis based APFD evaluation, *as many previous studies do*, there is no guarantee that the results will correlate to similar levels of performance on real faults. However, as we illustrate in the course of answering RQ$_3$, this correlation tends to vary across both the studied techniques and mutation operators. Furthermore, the actual rankings of different techniques across the different fault sets can impact the significance of these results. For example, if the weak Kendall correlation for the APFD metric is merely due to slight shuffling of a set of consistently top-ranked techniques, this may indicate that the relative performance of techniques according to *mutants* may be closer to real faults than implied by our correlation analysis. However, this

TABLE V
RESULTS FOR THE KENDALL τ_b RANK CORRELATION COEFFICIENT BETWEEN APFD(C) VALUES FOR TCP TECHNIQUES ON DETECTING MUTATION FAULTS AND DETECTING EACH TYPE OF REAL FAULTS DESCRIBED IN SECTION III-C3.

Real Faults	Chart		Lang		Math		Time		Closure		Mean	
	APFD	APFDc	APFD	APFDc	APFD	APFDc	APFD	APFDc	APFD	APFDc	APFD	APFDc
Couple	0.429	0.786	-0.071	0.286	0.5	0.429	-0.143	0	0.714	0.714	0.2858	0.443
Limitation	-0.214	0.214	-0.214	0.143	0	0.286	-0.071	0.143	0.5	0.286	0.0002	0.2144
StrongerOP	0.214	0.857	0.182	0.327	0	0.357	-0.286	0.5	0.714	0.571	0.1648	0.5224
NewOP	0.109	0.286	-	-	-0.143	0.286	-	-	0.571	0.571	0.179	0.381
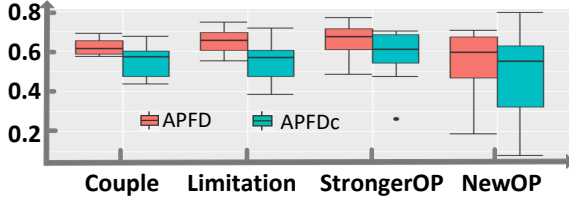
Fig. 1. APFD(c) values for TCP techniques in terms of detecting different types of real faults.

```
- currentPropertyNames = implicitProto.getOwnPropertyNames();
+ if (implicitProto == null) {
+     currentPropertyNames = ImmutableSet.of();
+ }
+ else {
+     currentPropertyNames = implicitProto.getOwnpropertyNames();
+ }
```

(a) Closure-2 Bug Fix.

```
- System.arraycopy(array2, 0, joinedArray, array2.length, array2.length)
+ try {
+     System.arraycopy(array2, 0, joinedArray, array2.length, array2.length)
+ } catch (ArrayStoreException ase) {
+     ...
+     ...
+     throw ase; // No, so rethrow original
+ }
```

(b) Lang-37 Bug Fix.

Fig. 2. Examples of bug fixing changes.

is not what we observe; for APFD, on average, the relative performance ordering according to real faults indicates that TCP_{topic} performs best and TCP_{add} performs worst. However, for both the full and subsuming mutant sets, TCP_{add} performs best, representing a reversal in the ordering of the best and worst ranked techniques. We observe similar trends for APFDc.

C. RQ₃: Effects of Fault Properties

In this subsection we investigate the effect that different fault properties have on the performance of TCP techniques, and on the relationship between mutants and real faults. As stated earlier, we discuss results for real faults in terms of different *coupling levels*, and mutants in terms of *operators*. In the context of **RQ₃**, we use the full mutant set for analysis instead of subsuming mutants, due to the limited number of subsuming mutants, particularly when grouping them based on mutation operators. However, interested readers can find the results for subsuming mutants in our attached appendix.

1) Effects of Coupling Between Mutants and Real Faults: To investigate the effect that fault coupling has on the performance of TCP across real-faults we consider four different fault types discussed in Section III-C3. The *performance* results for the APFD(c) metrics broken down by coupling level are illustrated in Figure 1. The *correlation* results for APFD(c) across subjects are given in Table V.

As Figure 1 shows, TCP techniques perform differently in terms of detecting different *types* (*e.g.,* coupling levels) of real faults. This result yields a few notable observations. First, TCP techniques tend to perform best (in terms of APFD and APFDc values) on real faults that are classified as needing *stronger operators* to be properly represented by mutants. This finding is encouraging, as it highlights that the studied approaches are capable of prioritization schemes that effectively uncover faults which are *not* closely coupled to mutants. When examining the correlation results, we find that the APFD τ_b coefficient values of *coupled* real faults are, unsurprisingly, substantially higher than for other types of real faults. This implies that TCP performance on real faults, which are more tightly coupled to mutants, is more strongly correlated with

performance on mutation faults in TCP evaluations. However, for APFDc we find that real faults requiring stronger operators tend to exhibit the highest correlation. Finally, as Table V shows, the correlation results vary across different subject programs. For instance, on one hand, the τ_b values for Closure are quite large across all levels of coupling, implying that TCP performance on mutants is more indicative of performance on real faults for this particular subject. On the other hand, the τ_b values for Lang are much closer to zero.

2) Effects of Different Mutation Operators: The *performance* distributions for the APFD(c) metrics across different operators are depicted as box plots in Figure 3. The *correlation* results across operators between the performance of mutants and real faults are shown in Table VI. The observations that can be made from this data help further explain the results of RQ₂. The *performance* results illustrate that the different TCP techniques tend to exhibit slight performance variances across different types of mutation operators, with the Switch and VoidMethodCall operators trending toward the positive and negative extremes respectively. This result implies that, even if a researcher is performing *only* a mutation-based analysis, the set of mutation operators selected can cause variations in the results. More importantly, the *correlation* results indicate that the degree to which performance on mutation faults correlates to performance on real faults, in terms of APFD(c), *varies dramatically across* systems as a a whole, and across different operator types *within* a single subject. For instance, when examining APFD values for specific systems, we found that mutation-based performance for both Closure and Math exhibits a strong positive correlation to performance on real faults across nearly all mutation operators. At the same time, operators such as `ConstructorCall` and `VoidMethodCall` exhibit a negative correlation within Chart, which tends to have a weak positive correlation overall. Overall the mutation-based APFDc metric is more strongly coupled to real faults than APFD, corroborating results from RQ₂.

Intuitively, our findings suggest that the characteristics of a

Mutation Faults	Chart		Lang		Math		Time		Closure		Mean	
	APFD	APFDc	APFD	APFDc	APFD	APFDc	APFD	APFDc	APFD	APFDc	APFD	APFDc
NegateConditionals	0.357	0.857	-0.143	0.143	0.429	0.571	-0.214	0.286	0.643	0.643	**0.214**	**0.500**
RemoveConditional	0.5	0.857	-0.143	0.143	0.429	0.571	-0.214	0.286	0.643	0.643	**0.243**	**0.500**
ConstructorCall	-0.143	0.714	0	0.286	0.5	0.714	-0.214	0.071	0.714	0.5	**0.171**	**0.457**
NonVoidMethodCall	0.214	0.786	0	0.286	0.357	0.5	-0.214	0.286	0.714	0.5	**0.214**	**0.471**
Math	0.286	0.714	-0.286	0.286	0.286	0.5	-0.071	-0.071	0.786	0.643	**0.200**	**0.414**
MemberVariable	0.214	0.929	-0.786	0.357	0.429	0.5	0.286	0.357	0.5	0.357	**0.129**	**0.500**
InlineConstant	0.286	0.929	-0.143	0.286	0.429	0.429	0	0.071	0.786	0.571	**0.272**	**0.456**
Increments	0.214	0.714	-0.214	0.143	0.286	0.571	0	0	0.786	0.714	**0.214**	**0.428**
ArgumentPropagation	0.143	0.857	0	0.214	0.357	0.286	-0.429	0.286	0.643	0.643	**0.143**	**0.457**
ConditionalsBoundary	0.357	0.786	-0.071	0.286	0.429	0.643	0.071	0.214	0.714	0.571	**0.300**	**0.500**
Switch	0.214	0.714	-0.214	-0.071	0.429	0.357	-0.214	0.214	0.786	0.429	**0.200**	**0.329**
VoidMethodCall	-0.143	0.714	-0.214	0.071	0.214	0.571	-0.357	0.357	0.857	0.643	**0.071**	**0.471**
InvertNegs	0.357	0.857	0.143	0.071	0.143	0.5	-0.214	0	0.714	0.714	**0.229**	**0.428**
ReturnVals	0.357	0.857	-0.429	0.357	0.357	0.714	-0.214	0.286	0.786	0.571	**0.171**	**0.557**
RemoveIncrements	0.214	0.786	-0.143	0.071	0.429	0.429	0.143	0.071	0.714	0.714	**0.271**	**0.414**

program may influence how representative mutation-analysis based TCP performance would be in terms of real faults. Therefore, we examined some of the bug fixing commits for a subject program that showed a strong correlation (Closure) and for a program that showed negative correlation (Lang). When looking into the fixing commits of these two subject programs, we found that Closure, being a compiler, trends heavily toward complex control flows managed by conditionals, compared to Lang, which trends more towards string and array manipulation. Two illustrative examples of bug fixes are shown in Figure 2. The first example for Closure (*i.e.,* Figure 2(a)) shows that developers fixed this bug by simply modifying conditionals. In particular, the bug shown in Figure 2(a) is exactly the same as one of the PIT mutation operators, *i.e.,* the NonVoidMethodCall mutator. Bug fixing in Lang mostly involved other more complex changes such as adding exception handlers due to the nature of its domain (see Figure 2(b)). This investigation suggests that TCP performance correlations between real faults and mutants is low when seeded mutants do not properly reflect faults occurring in a given domain or a program.

V. THREATS TO VALIDITY

Threats to Internal Validity concern potential confounding factors of the experiments that might introduce observed effects. One such factor is represented by faults that were seeded into the programs. We chose PIT to perform mutation analysis, which contains different operators compared to other mutation tools, such as Major [48]. While PIT features many operators common across other tools, it is possible that different mutation tools might have led to different observations. We leave exploration of additional tools as future work.

Internal validity threats also arise due to the assumptions made about the validity of the coupling between mutants and real faults obtained from Just *et al.* [35]. This is because mutants seeded in the same code where the fault occurred may differ from the real fault, hence leading to possible mis-classifications. Future research could further examine the affects of such coupling relationships to mitigate this threat.

Another potential confounding factor is the fact the studied test suites are written by developers. However, these test suites have been shown by prior work [35] to generally be of high quality, exhibiting high coverage, mitigating this threat. Additionally, threats may arise due to the utilization of the same test suite between program versions containing real faults and the latest program version (to which mutants were seeded). However, previous studies illustrate that the performance of mutation-based TCP techniques tend to be similar across different program versions for the same test suite [30], [45].

Threats to Construct Validity concern the relation between experimental theory/constructs, and potential effects on observed results. As explained in Section III-C, in the context of this study we re-implemented all of the TCP techniques following the approach descriptions in their respective papers. Our re-implementations may differ slightly from the original versions. However, we closely followed the methodology of the previous work, and three authors and one external TCP expert reviewed the code to ensure a reliable implementation. Executing the studied TCP techniques on all of the program versions was time-consuming, totaling more than five months of computation time. To make the GA-based technique tractable we reduced the maximum number of generations to 50. Also, to allow for a fair comparison, we used the same GA settings for both real faults and mutants.

Threats to External Validity. We limited our focus to eight TCP techniques, which require only source code, test code, and coverage information to perform prioritization. These eight TCP techniques are well-understood and widely used/studied in recent research work [45], [46], and since we aimed to understand how techniques differed from previous studies when applied to real faults, this set of techniques is suitable. We encourage researchers to extend this study to additional TCP approaches and technique configurations.

In order to provide a rigorous experimental procedure, we applied mutation analysis to TCP techniques in particular experimental settings discussed in Sec. III-B and III-C. Thus, it is possible that these results may differ for other TCP evaluation methodologies. However, we chose the experimental methodology set forth in this paper due to the fact that it has been widely used in previous studies [29], [42], [45], [46], [50], [77] and is likely to be used in the future.

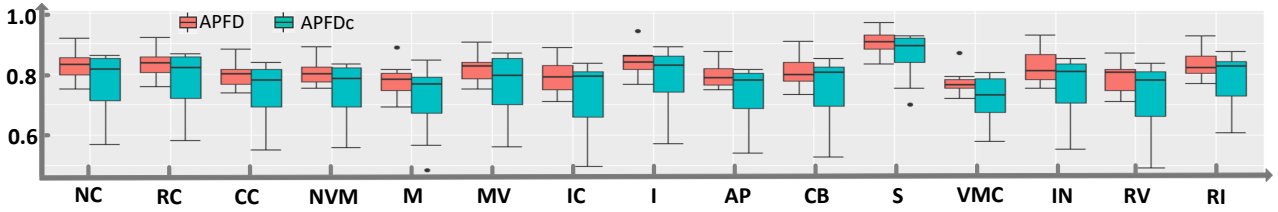We use the Defects4J dataset to understand the effective-

Fig. 3. Average APFD(c) values across different operators: NC = NegateConditional, RC = RemoveConditional, CC = ConstructorCall, NVM = NonVoidMethodCall, M = Math, MV = MemberVariable, IC = InlineConstant, I = Increments, AP = ArgumentPropagation, CB = ConditionalsBoundary, S = Switch, VMC = VoidMethodCall, IN = InvertNegs, RV = ReturnVals, and RI = RemoveIncrements.

ness of TCP techniques in terms of real-fault detection. It is possible that there are different types of faults (varying in complexity) in other subject programs written in other program languages compared to those in Defects4J. However, Defect4J is one of the largest and most studied [34], [35], [63] publicly available databases of real faults, containing 357 faults extracted from real-world software systems.

We utilize Pit [57] and hence our results are representative of a certain set of mutants. While Pit utilizes many of the same standard operators as other tools, this study could be expanded in the future to consider additional mutation testing tools.

VI. LESSONS LEARNED

In this section we summarize the pertinent findings of our study into discrete *learned lessons* and discuss their potential impact on future work in the TCP area.

Lesson 1: *Relative Performance of TCP techniques on mutants may not indicate similar relative performance on real faults, depending on the subject program.* Our study indicates that, for the subject programs studied, the relative performance of TCP techniques (following the popular methodology utilized in our experiments) is not similar between mutants and real faults. This indicates that a technique which outperformed competing techniques under the experimental setting of mutation analysis may not outperform competing techniques on real faults. This highlights a potential threat to validity for mutation-based assessments of TCP approaches, impacting the generalizability of TCP performance comparisons to real program faults. Future work should proceed in two directions. First, techniques for carefully selecting mutants should be pursued (see Lesson 3). Second, there is a clear need for comprehensive datasets of real faults, such as Defects4J, in order to more thoroughly evaluate TCP approaches. Therefore, researchers should focus on developing reliable automated or semi-automated techniques to extract and isolate real faults from existing open source software projects, which clearly calls for a community-wide effort.

Lesson 2: *The metrics utilized in mutation-based evaluations of TCP techniques impact the representativeness of performance on real faults.* We found that mutation-based APFD values generally exhibit only a *weak positive correlation* to APFD values calculated in terms of real fault discovery, whereas for APFDc this correlation was medium to strong. However, such results varied across subjects programs. This is important as it signals that when considering the extremely popular APFD metric, mutation-based performance of a particular TCP technique *may* be independent of its performance on real faults. This means that, depending on the subject program, one

may not be able to rely upon mutation-based APFD to predict the *practical* performance of TCP technique on real faults. While one could use the more strongly correlated APFDc metric, researchers may not always want to include execution cost in their performance evaluation.

Lesson 3: *The types of mutation operators utilized for TCP performance evaluations must be carefully selected or derived in order for the results to be representative of performance on real faults.* Our results indicate that correlations in TCP performance between mutants and real faults vary both across subject programs and across different types of mutation operators within a specific subject program. This suggests that different characteristics of subject programs most likely play a role in determining the representativeness of certain mutation operators for a particular subject (or domain). This is actually a positive outcome when considering the future applicability of mutation testing to TCP evaluations, as it shows that *under the right circumstances* mutation-based TCP can, in fact, be realistic. However, in order to properly achieve the "correct circumstances" for mutation operators to be applied, further research needs to be conducted. This specifically illustrates the need for the following interconnected research threads: (i) deriving, either manually or automatically, fault models for specific software systems or domains; (ii) developing tailored mutation operators based on such fault models; and (iii) seeding mutants using rigorous statistical methods according to observed distributions of faults. If thoroughly pursued, we believe that research efforts directed toward these goals will provide for future tools capable of generating mutants that are more representative of real faults, not only in for TCP, but also in other areas of software testing.

VII. CONCLUSION

In this work we conducted the first empirical study investigating the extent to which mutation-based evaluations of TCP approaches are *realistic*. We examined the performance, in terms of both the APFD and APFDc metrics, of eight different TCP approaches applied to a dataset of 357 real world faults from the Defects4J dataset and a set of over 35k mutants. Our results indicate that typical mutation-based evaluations of TCP techniques tend to *overestimate* absolute performance on real faults. Furthermore, for the APFD metric, relative performance on mutants may not be representative of relative performance on real faults, depending upon mutant and program characteristics. These findings highlight the need for future work on deriving mutation operators tailored toward specific subject programs or domains, driving mutation-based TCP evaluations toward being more realistic.

REFERENCES

[1] J. H. Andrews, L. C. Briand, and Y. Labiche. Is mutation an appropriate tool for testing experiments? In *27th International Conference on Software Engineering (ICSE 2005), 15-21 May 2005, St. Louis, Missouri, USA*, pages 402–411, 2005.

[2] J. H. Andrews, L. C. Briand, Y. Labiche, and A. S. Namin. Using mutation analysis for assessing and comparing testing coverage criteria. *IEEE Trans. Software Eng.*, 32(8):608–624, 2006.

[3] M. J. Arafeen and H. Do. Test case prioritization using requirements based clustering. In *Proc. IEEE International Conference on Software Testing, Verification, and Validation, ICST 2013*, pages 312–321, 2013.

[4] ASM. http://asm.ow2.org/.

[5] D. M. Blei, A. Y. Ng, and M. I. Jordan. Latent Dirichlet Allocation. *The Journal of Machine Learning Research*, 3:993–1022, 2003.

[6] B. Busjaeger and T. Xie. Learning for test prioritization: An industrial case study. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, FSE 2016, pages 975–980, New York, NY, USA, 2016. ACM.

[7] C. Catal and D. Mishra. Test case prioritization: a systematic mapping study. *Software Quality Journal*, 21(3):445–478, 2013.

[8] T. T. Chekam, M. Papadakis, Y. L. Traon, and M. Harman. An empirical study on mutation, statement and branch coverage fault revelation that avoids the unreliable clean program assumption. In *Proceedings of the 39th International Conference on Software Engineering*, ICSE '17, pages 597–608, Piscataway, NJ, USA, 2017. IEEE Press.

[9] T. H. Chen. *Studying Software Quality Using Topic Models*. PhD thesis, 2013.

[10] T. Y. Chen, H. Leung, and I. Mak. Adaptive random testing. In *Advances in Computer Science-ASIAN 2004. Higher-Level Decision Making*, pages 320–329. Springer, 2004.

[11] J. Czerwonka, R. Das, N. Nagappan, A. Tarvo, and A. Teterev. Crane: Failure prediction, change analysis and test prioritization in practice – experiences from windows. In *Proceedings of the 2011 Fourth IEEE International Conference on Software Testing, Verification and Validation*, ICST '11, pages 357–366, 2011.

[12] M. Daran and P. Thévenod-Fosse. Software error analysis: A real case study involving real faults and mutations. In *Proceedings of the 1996 International Symposium on Software Testing and Analysis, ISSTA 1996, San Diego, CA, USA, January 8-10, 1996*, pages 158–171, 1996.

[13] N. Dini, A. Sullivan, M. Gligoric, and G. Rothermel. The effect of test suite type on regression test selection. In *Software Reliability Engineering (ISSRE), 2016 IEEE 27th International Symposium on*, pages 47–58. IEEE, 2016.

[14] H. Do, S. Mirarab, L. Tahvildari, and G. Rothermel. An empirical study of the effect of time constraints on the cost-benefits of regression testing. In *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of Software Engineering, 2008, Atlanta, Georgia, USA, November 9-14, 2008*, pages 71–82, 2008.

[15] H. Do and G. Rothermel. A controlled experiment assessing test case prioritization techniques via mutation faults. In *21st IEEE International Conference on Software Maintenance (ICSM 2005), 25-30 September 2005, Budapest, Hungary*, pages 411–420, 2005.

[16] H. Do and G. Rothermel. An empirical study of regression testing techniques incorporating context and lifetime factors and improved cost-benefit models. In *Proceedings of the 14th ACM SIGSOFT International Symposium on Foundations of Software Engineering, FSE 2006, Portland, Oregon, USA, November 5-11, 2006*, pages 141–151, 2006.

[17] H. Do and G. Rothermel. On the use of mutation faults in empirical assessments of test case prioritization techniques. *IEEE Trans. Software Eng.*, 32(9):733–752, 2006.

[18] H. Do, G. Rothermel, and A. Kinneer. Empirical studies of test case prioritization in a junit testing environment. In *15th International Symposium on Software Reliability Engineering (ISSRE 2004), 2-5 November 2004, Saint-Malo, Bretagne, France*, pages 113–124, 2004.

[19] S. Elbaum, P. Kallakuri, A. Malishevsky, G. Rothermel, and S. Kanduri. Understanding the effects of changes on the cost-effectiveness of regression testing techniques. *Software testing, verification and reliability*, 13(2):65–83, 2003.

[20] S. Elbaum, A. Malishevsky, and G. Rothermel. Incorporating varying test costs and fault severities into test case prioritization. In *Proceedings of the 23rd International Conference on Software Engineering*, ICSE '01, pages 329–338, Washington, DC, USA, 2001. IEEE Computer Society.

[21] S. Elbaum, A. G. Malishevsky, and G. Rothermel. Test case prioritization: A family of empirical studies. *IEEE Trans. Software Eng.*, 28(2):159–182, 2002.

[22] S. Elbaum, G. Rothermel, S. Kanduri, and A. G. Malishevsky. Selecting a cost-effective test case prioritization technique. *Software Quality Journal*, 12(3):185–210, 2004.

[23] S. G. Elbaum, A. G. Malishevsky, and G. Rothermel. Prioritizing test cases for regression testing. In *Proceedings of the International Symposium on Software Testing and Analysis, ISSTA 2000, Portland, OR, USA, August 21-24, 2000*, pages 102–112, 2000.

[24] M. G. Epitropakis, S. Yoo, M. Harman, and E. K. Burke. Empirical evaluation of pareto efficient multi-objective regression test case prioritisation. In *Proceedings of the 2015 International Symposium on Software Testing and Analysis*, ISSTA 2015, pages 234–245, New York, NY, USA, 2015. ACM.

[25] E. S. Foundation. Eclipse jdt http://www.eclipse.org/jdt/, 2017.

[26] M. Gligoric, A. Groce, C. Zhang, R. Sharma, M. A. Alipour, and D. Marinov. Comparing non-adequate test suites using coverage criteria. In *Proceedings of the 2013 International Symposium on Software Testing and Analysis*, ISSTA 2013, pages 302–313, New York, NY, USA, 2013. ACM.

[27] R. Gopinath, C. Jensen, and A. Groce. Mutations: How close are they to real faults? In *Proceedings of the 2014 IEEE 25th International Symposium on Software Reliability Engineering*, ISSRE '14, pages 189–200, Washington, DC, USA, 2014. IEEE Computer Society.

[28] D. Grove, G. DeFouw, J. Dean, and C. Chambers. Call graph construction in object-oriented languages. In *Proceedings of the 12th ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications*, OOPSLA '97, pages 108–124, New York, NY, USA, 1997. ACM.

[29] D. Hao, L. Zhang, L. Zhang, G. Rothermel, and H. Mei. A unified test case prioritization approach. *ACM Trans. Softw. Eng. Methodol.*, 24(2):10:1–10:31, 2014.

[30] C. Henard, M. Papadakis, M. Harman, Y. Jia, and Y. L. Traon. Comparing white-box and black-box test prioritization. In *Proceedings of the 38th International Conference on Software Engineering*, ICSE 2016, New York, NY, USA, 2016. ACM.

[31] M. M. Islam, A. Marchetto, A. Susi, and G. Scanniello. A multi-objective technique to prioritize test cases based on latent semantic indexing. In *2012 16th European Conference on Software Maintenance and Reengineering (CSMR)*, pages 21–30, 2012.

[32] B. Jiang and W. Chan. Bypassing code coverage approximation limitations via effective input-based randomized test case prioritization. In *Proc. IEEE International Conference on Computers, Software and Applications, COMPSAC 2013*, pages 190–199, 2013.

[33] B. Jiang, Z. Zhang, W. K. Chan, and T. H. Tse. Adaptive random test case prioritization. In *ACM/IEEE International Conference on Automated Software Engineering, ASE 2009*, pages 257–266, 2009.

[34] R. Just, D. Jalali, and M. D. Ernst. Defects4j: A database of existing faults to enable controlled testing studies for java programs. In *Proceedings of the 2014 International Symposium on Software Testing and Analysis*, ISSTA 2014, pages 437–440, New York, NY, USA, 2014. ACM.

[35] R. Just, D. Jalali, L. Inozemtseva, M. D. Ernst, R. Holmes, and G. Fraser. Are mutants a valid substitute for real faults in software testing? In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, FSE 2014, pages 654–665, New York, NY, USA, 2014. ACM.

[36] G. M. Kapfhammer and M. L. Soffa. Using coverage effectiveness to evaluate test suite prioritizations. In *Proceedings of the 1st ACM international workshop on Empirical assessment of software engineering languages and technologies: held in conjunction with the 22nd IEEE/ACM International Conference on Automated Software Engineering (ASE) 2007*, pages 19–20. ACM, 2007.

[37] J. Kasurinen, O. Taipale, and K. Smolander. Test case selection and prioritization: Risk-based or design-based? In *Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, page 10. ACM, 2010.

[38] R. Kavitha and N. Sureshkumar. Test case prioritization for regression testing based on severity of fault. *International Journal on Computer Science and Engineering*, 2(5):1462–1466, 2010.

[39] M. Kendall. A new measure of rank correlation. 1938.

[40] M. G. Kendall. A new measure of rank correlation. *Biometrika*, 30(1/2):81–93, 1938.

[41] B. Korel, L. H. Tahat, and M. Harman. Test prioritization using system models. In *Software Maintenance, 2005. ICSM'05. Proceedings of the 21st IEEE International Conference on*, pages 559–568, Sept 2005.

[42] Y. Ledru, A. Petrenko, S. Boroday, and N. Mandran. Prioritizing test cases with string distances. *Automated Software Engineering*, 19(1):65–95, 2012.

[43] Z. Li, M. Harman, and R. Hierons. Search algorithms for regression test case prioritisation. *IEEE Trans. Software Eng.*, 33(4):225–237, 2007.

[44] J. Liang, S. Elbaum, and G. Rothermel. Redefining prioritization: Continuous prioritization for continuous integration. In *Proceedings of the 40th International Conference on Software Engineering*, page to appear, 2018.

[45] Y. Lu, Y. Lou, S. Cheng, L. Zhang, D. Hao, Y. Zhou, and L. Zhang. How does regression test prioritization perform in real-world software evolution? In *In Proc. of the ACM/IEEE International Conference on Software Engineering, ICSE'16*, 2016.

[46] Q. Luo, K. Moran, and D. Poshyvanyk. A large-scale empirical comparison of static and dynamic test case prioritization techniques. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, FSE 2016, pages 559–570, New York, NY, USA, 2016. ACM.

[47] Q. Luo, K. Moran, D. Poshyvanyk, and M. Di Penta. Online appendix. http://www.cs.wm.edu/semeru/data/ICSME18-TCP-Study/.

[48] Major. http://mutation-testing.org/.

[49] Mallet. http://mallet.cs.umass.edu/.

[50] H. Mei, D. Hao, L. Zhang, L. Zhang, J. Zhou, and G. Rothermel. A static approach to prioritizing junit test cases. *IEEE Trans. Softw. Eng.*, 38(6):1258–1275, Nov. 2012.

[51] B. Miranda, E. Cruciani, R. Verdecchia, and A. Bertolino. Fast approaches to scalable similarity-based test case prioritization. In *Proceedings of the 40th International Conference on Software Engineering*, page to appear, 2018.

[52] R. Nelson. Kendall tau metric. *Encyclopedia of Mathematics*, 2011.

[53] C. D. Nguyen, A. Marchetto, and P. Tonella. Test case prioritization for audit testing of evolving web services using information retrieval techniques. In *Proc. ICWS*, pages 636–643, 2011.

[54] M. Papadakis, S. Donghawn, S. Yoo, and B. Doo-Hwan. An empirical study on mutation, statement and branch coverage fault revelation that avoids the unreliable clean program assumption. In *Proceedings of the 40th International Conference on Software Engineering*, ICSE '18, page to appear, Piscataway, NJ, USA, 2018. IEEE Press.

[55] M. Papadakis, C. Henard, M. Harman, Y. Jia, and Y. Le Traon. Threats to the validity of mutation-based test assessment. In *Proceedings of the 25th International Symposium on Software Testing and Analysis*, ISSTA 2016, pages 354–365, New York, NY, USA, 2016. ACM.

[56] M. Papadakis, C. Henard, and Y. L. Traon. Sampling program inputs with mutation analysis: Going beyond combinatorial interaction testing. In *Proceedings of the 2014 IEEE International Conference on Software Testing, Verification, and Validation*, ICST '14, pages 1–10, Washington, DC, USA, 2014. IEEE Computer Society.

[57] PIT. http://pitest.org/.

[58] X. Qu, M. B. Cohen, and G. Rothermel. Configuration-aware regression testing: an empirical study of sampling and prioritization. In *Proceedings of the 2008 international symposium on Software testing and analysis*, pages 75–86. ACM, 2008.

[59] G. Report. http://google-engtools.blogspot.com/2011/06/testing-at-speed-and-scale-of-google.html.

[60] G. Rothermel, R. H. Untch, C. Chu, and M. J. Harrold. Test case prioritization: an empirical study. In *IEEE International Conference on Software Maintenance, ICSM 1999*, pages 179–188, 1999.

[61] G. Rothermel, R. J. Untch, C. Chu, and M. J. Harrold. Prioritizing test cases for regression testing. *IEEE Trans. Software Eng.*, 27(10):929–948, 2001.

[62] R. K. Saha, L. Zhang, S. Khurshid, and D. E. Perry. An information retrieval approach for regression test prioritization based on program changes. In *Proc. of the ACM/IEEE International Conference on Software Engineering, ICSE'15*, 2015.

[63] S. Shamshiri, R. Just, J. M. Rojas, G. Fraser, P. McMinn, and A. Arcuri. Do automatically generated unit tests find real faults? an empirical study of effectiveness and challenges (t). In *Automated Software Engineering (ASE), 2015 30th IEEE/ACM International Conference on*, pages 201–211. IEEE, 2015.

[64] A. M. Smith and G. M. Kapfhammer. An empirical study of incorporating cost into test suite reduction and prioritization. In *Proceedings of the 2009 ACM symposium on Applied Computing*, pages 461–467. ACM, 2009.

[65] A. Srivastava and J. Thiagarajan. Effectively prioritizing tests in development environment. In *ISSTA*, pages 97–106, 2002.

[66] S. W. Thomas, H. Hemmati, A. E. Hassan, and D. Blostein. Static test case prioritization using topic models. *Empirical Software Engineering Journal, EMSE*, 19(1):182–212, 2014.

[67] P. Tonella, P. Avesani, and A. Susi. Using the case-based ranking methodology for test case prioritization. In *IEEE International Conference on Software Maintenance, ICSM 2009*, pages 123–133, 2006.

[68] J. Tukey. Comparing individual means in the analysis of variance. *Biometrics*, 5(2):99–114, 1949.

[69] S. Varun Kumar and M. Kumar. Test case prioritization using fault severity. *IJCST*, 1(1), 2010.

[70] WALA. https://github.com/wala/WALA.

[71] K. R. Walcott, M. L. Soffa, G. M. Kapfhammer, and R. S. Roos. Time aware test suite prioritization. In *ISSTA*, pages 1–11, 2006.

[72] S. Wang, D. Buchmann, S. Ali, A. Gotlieb, D. Pradhan, and M. Liaaen. Multi-objective test prioritization in software product line testing: An industrial case study. In *Proceedings of the 18th International Software Product Line Conference - Volume 1*, SPLC '14, pages 32–41, New York, NY, USA, 2014. ACM.

[73] S. Wang, J. Nam, and L. Tan. QTEP: quality-aware test case prioritization. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2017, Paderborn, Germany, September 4-8, 2017*, pages 523–534, 2017.

[74] D. Xu and J. Ding. Prioritizing state-based aspect tests. In *Proc. IEEE International Conference on Software Testing, Verification and Validation, ICST 2010*, pages 265–274, 2010.

[75] S. Yoo and M. Harman. Regression testing minimization, selection and prioritization: a survey. *Softw. Test., Verif. Reliab.*, 22(2):67–120, 2012.

[76] D. You, Z. Chen, B. Xu, B. Luo, and C. Zhang. An empirical study on the effectiveness of time-aware test case prioritization techniques. In *Proceedings of the 2011 ACM Symposium on Applied Computing*, pages 1451–1456. ACM, 2011.

[77] L. Zhang, M. Gligoric, D. Marinov, and S. Khurshid. Operator-based and random mutant selection: Better together. In *Automated Software Engineering (ASE), 2013 IEEE/ACM 28th International Conference on*, pages 92–102. IEEE, 2013.

[78] L. Zhang, D. Hao, L. Zhang, G. Rothermel, and H. Mei. Bridging the gap between the total and additional test-case prioritization strategies. In *35th International Conference on Software Engineering, ICSE '13, San Francisco, CA, USA, May 18-26, 2013*, pages 192–201, 2013.

[79] L. Zhang, M. Kim, and S. Khurshid. Localizing failure-inducing program edits based on spectrum information. In *27th IEEE International Conference on Software Maintenance, ICSM 2011*, pages 23–32. IEEE, 2011.

[80] L. Zhang, J. Zhou, D. Hao, L. Zhang, and H. Mei. Prioritizing JUnit test cases in absence of coverage information. In *IEEE International Conference on Software Maintenance, ICSM 2009*, pages 19–28, 2009.