Severity-Aware Prioritization of System-Level Regression Tests in Automotive Software

Roland Wuersching*
Technical University of Munich
Munich, Germany

Daniel Elsner*
Technical University of Munich
Munich, Germany

Fabian Leinen
Technical University of Munich
Munich, Germany

Alexander Pretschner
Technical University of Munich
Munich, Germany

Georg Grueneissl
MAN Truck & Bus SE
Munich, Germany

Thomas Neumeyr
MAN Truck & Bus SE
Munich, Germany

Tobias Vosseler
MAN Truck & Bus SE
Munich, Germany

***Abstract*—In automotive software engineering, system-level regression testing is crucial to ensure proper integration of often-times safety-critical components. Due to the inherent complexity of such systems and components, testing is commonly performed manually and in a black-box manner, which is particularly costly and leads to slow feedback cycles between testers and developers. Regression Test Prioritization (RTP) aims to reduce feedback time by ordering tests to reveal faults earlier during the testing process. However, most prior RTP research does not incorporate varying fault severity, which must be taken into account when evaluating and designing appropriate RTP approaches for safety-critical automotive software systems. In this work, we present a case study at our industry partner MAN, a leading international provider of commercial vehicles. We design and instantiate a domain-specific, severity-aware RTP assessment model and comparatively assess state-of-the-art RTP approaches. Our results indicate that simple and partly well-known heuristics based on test history and test costs have the best cost-effectiveness, achieving between 85% and 90% of the maximum possible feedback time reduction. On the other hand, search-based and machine-learning-based RTP approaches do not perform better, especially if available test history is sparse.**

***Index Terms*—Regression test prioritization, manual testing, automotive software, system-level testing**

I. Introduction

Regression testing is regularly performed on software systems to ensure that existing system behavior is not inadvertently affected by changes. However, with increasingly large test suites and shorter software (delivery) life-cycles, the need to reduce feedback time during development arises [1]. Regression Test Prioritization (RTP) aims to reduce feedback time for developers by running those tests earlier that are more likely to reveal faults. Yet, since testers cannot know in advance if a test case fails or not, *surrogates* are used to order tests.

RTP approaches using different kinds of surrogates have been proposed in the past two decades: existing approaches harness white-box information such as code coverage [2]–[6], (textual) test case similarity [7], [8], program changes and test code [9]–[12], or code quality metrics [12], [13]; or black-box information such as test input diversity [6], version control system metadata [12], [14]–[16], or failure history [16]–[19].

In the automotive domain, system-level regression testing is particularly complex, as test cases typically require the interplay of (embedded) hardware and software components in order to be successfully executed [20]. As a result, testing is commonly performed manually, making it especially costly. Often, system-level testing is further done in a black-box manner, since neither code artifacts nor information about components from external suppliers is available to testers [20], [21]. These characteristics naturally limit the applicability of RTP approaches that use information beyond black-box information.

Numerous RTP studies investigate the effectiveness of black-box RTP approaches [6], [16], [17], [19], [20], [22]–[24]. However, there are several limitations with these studies in the automotive domain. First, test failures are typically regarded as equally severe. Yet, in the given context, this assumption does not hold true: a failure of a test case that covers safety-critical requirements is more severe than of a test covering no such requirements. Second, availability or absence of historical data can pose significant challenges to the design and evaluation of RTP approaches [21]. For instance, if historical test plans are not available, defining a realistic baseline other than random test ordering is non-trivial. Last, several RTP approaches rely on Machine Learning (ML) algorithms requiring significant amounts of training data which cannot be guaranteed to be available. Also, these ML-based RTP approaches are rarely compared to search-based and heuristics-based approaches in an industrial context, even though the latter have been shown to work well, for instance, in the context of Continuous Integration (CI) [16], [17].

To address these limitations, this case study investigates the effectiveness of several RTP approaches in an industrial setting

*Both authors contributed equally

for system-level regression testing at MAN[1]. Particularly, we apply a set of purely black-box RTP approaches that are suitable even if only little historical test execution data are available. We further design and instantiate a domain-specific, severity-aware RTP assessment model, which allows us to account for varying failure severity. Also, we generate context-specific RTP baselines to realistically reflect the test prioritization performed manually by testers.

We then comparatively evaluate the baselines and RTP approaches on an industrial system-level regression test suite at MAN. The results of our study indicate that (1) well-known RTP surrogates based on historical failures work well and improve with increasing amount of historical data, (2) domain-specific failure severity models are required to produce adequate test orderings in the context of automotive software engineering, (3) neither multi-objective nor advanced ML-based RTP approaches provide significant benefits compared to simple, single-objective approaches.

In summary, our work makes the following contributions:

- **Severity-aware RTP assessment model** for system-level regression testing in the automotive domain
- **Empirical evaluation** of state-of-the-art black-box RTP approaches, both severity-unaware and severity-aware
- **Industrial case study** demonstrating how RTP techniques can reduce testing feedback time in automotive software engineering at MAN

II. System-level Testing in Automotive Software Engineering at MAN

At our industry partner MAN, system tests are performed with each new major release, which are planned for every three months. The system test cases are assigned to certain vehicle functions, e.g., a system test case might verify proper functioning of the illumination of the direction indicator lights. There are between 12 and 58 test cases for each of the vehicle functions examined in this work (see Sec. IV). Some of the test cases are partly automated, but the majority has to be executed manually.

Test cases and results from their execution are organized in proprietary tools. The test cases are described in natural language and contain preconditions, test execution steps with expected observable results, as well as setup or tear-down procedures. In addition, each test case can be linked to requirements. Some of these requirements are legally- or safety-relevant and therefore particularly important. Furthermore, several test cases are manually marked as smoke tests, i.e., they cover core functionality and their failure may block more in-depth testing. Test results in this context can be either *passed*, *failed*, or *inconclusive*. We expect *passed* and *failed* verdicts to be generally known. The *inconclusive* verdict signifies that there were problems with the test execution because the test case was not described in sufficient detail or due to problems with the test infrastructure. Sometimes faults can be inferred from this result, which is why *inconclusive* verdicts are more relevant for testers than *passed* verdicts, since they could indicate a bug in the system. Due to the technical complexity of instrumenting various (third-party) hardware and software components, execution information, such as coverage data, is not collected, which is common in the automotive domain and for manual (system-level) testing in general [21]. The execution times of the individual test cases are currently also not tracked.

As the regression testing process can take up to several weeks, receiving feedback about introduced regression bugs early is crucial to prevent last minute bug fixes. Despite these rather long testing periods, limited testing capacities still sometimes prohibit executing each test case for every release. Prioritization of test cases is thus essential for delivering high quality software given the time constraints. However, prioritization and selection of test cases is currently mainly based on expert knowledge and a few simple heuristics, which we explain in more detail in Sec. IV. Therefore, our goal for this work was to find more systematic RTP approaches which minimize the feedback time for system-level regression testing and operate on the readily available data at MAN.

III. Prioritizing System-level Regression Test Cases

Most RTP studies target unit-level testing whereas this study focuses on system-level regression testing. We have motivated why RTP approaches that merely use black-box metadata are often more suitable for system-level regression testing, given that static and dynamic program analysis results are either not available, limited to a single system or programming language, or may induce prohibitive overhead in production deployments [16], [21], [25]–[27]. This study aims to compare existing techniques and develop new approaches for RTP in system-level regression testing in an industrial context, more specifically for embedded automotive software at MAN.

In the following, we first outline single-objective black-box RTP techniques from prior research together with the widespread Average Percentage of Faults Detected (APFD) evaluation metric for RTP. Second, we introduce multiple, possibly competing objective functions for RTP in the given context and the idea behind the cost-cognizant Average Percentage of Faults Detected (APFD_C), an extension of the APFD metric by Elbaum et al. [28], that incorporates varying test costs and fault severities. Last, we describe solution approaches, such as genetic algorithms, for the resulting severity-aware multi-objective RTP problem.

The introduced RTP techniques are then comparatively evaluated with respect to APFD and APFD_C in our case study in Sec. IV.

A. Single-objective Black-box RTP

1) Evaluating RTP with the APFD Metric: Rothermel et al. [29] and Elbaum et al. [30] were among the first to use the APFD metric to compare and evaluate test suite orderings with respect to how early faults are detected in the testing process.

[1]MAN Truck & Bus is one of the leading international providers of commercial vehicles.

The intuition behind the APFD metric is explained by the illustration in Fig. 1, where two test suite orderings, T' and T, are compared. The area under these so-called *gain curves* reflects the APFD metric and can range from 0 to 1. A random ordering of test cases has an expected value of 0.5. Here, since T' has a larger area under the curve, it is preferred over T, as it reveals faults earlier in the testing process. The APFD metric thus measures the rate of fault detection, which is the single objective that is optimized for by numerous proposed white-box and black-box RTP approaches. We partially discuss these approaches in the next section and in Sec. V.

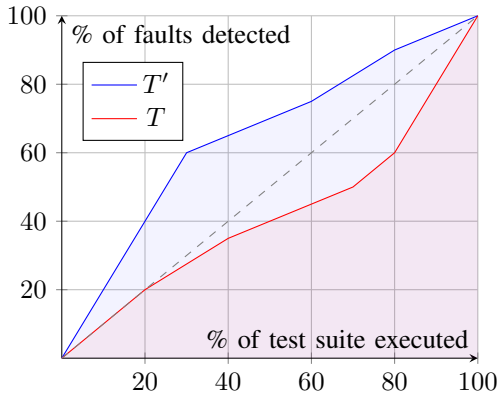


Fig. 1: Illustration of the intuition behind the APFD evaluation metric

Notably, the computation of the APFD metric is only possible, if at least one fault is detected by the test suite. There is another caveat in practice when information about detected *faults* is not available, but merely observed *failures* are reported—a so-called *failure-to-fault* mapping is missing. In these cases, prior RTP research makes the assumption of a one-to-one failure-to-fault mapping [10], [16], [31], [32], meaning that there is one distinct fault for every failure. Then, the APFD metric can be interpreted as the average percentage of detected *failures*. However, since multiple test failures can occur due to the same underlying fault, this assumption might distort RTP evaluation results [10], [16].

2) Single-objective RTP Surrogates from Black-box Meta-data: If system-level testing is performed *manually*, transferability of RTP approaches is often hindered by the fact that required analysis data, e.g., code coverage or static dependency graphs, are not available [21]. Similarly, for *automated* black-box system-level regression testing typically only metadata such as the failure history, the observed test execution costs, or textual information from the test case specification are available [20], [23]. While using such metadata for RTP has been shown to work well in automated CI testing [16], [17], there are only few studies that use them in optimizing black-box system-level testing.

At MAN, failure history data and test case specifications are readily available for system-level regression tests. Hence, the following surrogates can be computed, which have been used in prior studies to prioritize tests to optimize for early fault detection, i.e., obtain high values of the APFD metric:

- **Historical failure frequency:** Number of times a test failed in the past divided by how often it was executed [10], [16]
- **Test flip rate:** Number of times a test transitioned from one test result to another result divided by how often it was executed [33]
- **Test executions since last failure:** Number of test executions since the last time the test failed [17]
- **Test size:** Since data on test execution duration are missing [16], we take the number of test case steps to approximate test size

Each of these surrogates is based on a failure hypothesis (see Elsner et al. [16] for details), e.g., to execute tests first that have been more failure-prone in the past (*historical failure frequency*). The surrogates can either be directly used as heuristics to rank tests in ascending or descending order, or they can be fed into statistical models. These predict a value between 0 and 1 indicating the chance that a test fails in the next run. We follow the methodology proposed by Elsner et al. [16] and compare each surrogate individually as a heuristic and additionally apply the three ML models (1) logistic regression, (2) random forest and (3) Support Vector Machine (SVM). The models use all four surrogates as input, i.e., as predictors for test failures.

B. Multi-objective Black-box RTP

1) Evaluating RTP with the APFD$_C$ Metric: The execution costs of different regression tests can vary due to manifold reasons: A test may require certain (limited) infrastructure capabilities or take longer to execute due to a large number of assertions or a vast amount of tested functionality. Similarly, faults—or failures, if a one-to-one mapping is assumed—may have different severities. For instance, a fault may be safety-related and block the software release due to the implied risk for the users. Therefore, Elbaum et al. [28] extended the APFD metric by incorporating varying test case and fault costs (i.e., *severity*) into the APFD$_C$ metric. In contrast to the traditional APFD metric illustrated in Fig. 1, for the APFD$_C$ metric the y-axis depicts the *percentage of total fault severity* and the x-axis reflects the *percentage of total test cost incurred*.

Having a cost-aware (or severity-aware) metric for RTP evaluation raises concerns regarding the usefulness of the single-objective RTP approaches discussed in the previous section. If the objective of RTP is no longer limited to the rate of fault detection, adequate prioritization strategies must also take into account test costs and fault severity aspects. This demands optimization for multiple objectives at once [34], [35], which we describe next.

2) Competing Test Objectives at MAN: RTP can be performed with multiple objectives, instead of simply detecting *any* fault early. Below, we list objectives that are relevant in the context of MAN and for which data points are already available.

- **Failure revealing history:** This objective is based on the same hypothesis as the surrogates introduced in Sec. III-A2; it prioritizes tests that have failed before over tests that have never failed [20]
- **Requirements coverage:** Tests that cover more requirements are preferred, as more potentially erroneous functionality might be covered
- **Safety-critical/Legal coverage:** Tests that cover safety-critical or legal requirements are preferred, as a corresponding bug might block the release
- **Development/Smoke tests:** Tests that are labelled as smoke tests are particularly development-related and therefore should be executed earlier, since if they fail, they might block the execution of other tests
- **Test specification size:** Tests with fewer test steps are preferable, as they are often faster to execute and could give faster feedback than tests with more test steps

To properly account for these objectives when evaluating RTP approaches, we need to adjust the APFD_C metric accordingly.

In the context of MAN, we define the *costs* for the APFD_C metric as the number of test steps in a test case, due to the lack of other data to approximate test execution cost. For modeling the *severity* of a test failure (we assume a one-to-one failure-to-fault mapping), we define t to be a test case from the test suite T. R is defined as a subset of T containing all safety- or legally-relevant tests and S is defined as a subset of T containing all smoke tests:

$$t \in T$$
$$R \subset T$$
$$S \subset T$$

We define the properties s_t, r_t, and v_t of test case t in a given release as:

$$s_t = \begin{cases} 1, & \text{if } t \in S \\ 0, & \text{otherwise} \end{cases} \qquad r_t = \begin{cases} 1, & \text{if } t \in R \\ 0, & \text{otherwise} \end{cases}$$

$$v_t = \begin{cases} 0, & \text{if } \textit{passed} \\ 1, & \text{if } \textit{inconclusive} \\ 2, & \text{if } \textit{failed} \end{cases}$$

Using these specifications and the context-specific model parameters ρ and σ, we define our *severity assessment model* for the failure severity f_t of test case t as

$$f_t = (1 + (\rho \cdot s_t) + (\sigma \cdot r_t)) \cdot v_t$$

The base failure severity is 1, which is increased if the test case is a smoke test ($\rho \cdot s_t$) or if it is linked to a safety-critical or legal requirement ($\sigma \cdot r_t$). The exact amount of severity that is added on top of the base severity depends on the context-specific model parameters ρ and σ. We then multiply the severity with an integer value, v_t, representing the severity of the test verdict.

After thorough discussion with MAN engineers, we instantiate the model with $\rho = 2$ and $\sigma = 1$ in the case study described in Sec. IV. The choice of these values was based on the engineers' view that certain aspects of a test case are qualitatively more significant than others, and should therefore have a greater impact on the severity of a failure.

TABLE I: Summary of test verdicts across the six vehicle functions

Function	#Test Cases	#Test Executions			
		Total	Passed	Failed	Inconclusive
F_1	58	411	222	27	162
F_2	51	413	276	17	120
F_3	34	315	186	11	118
F_4	30	192	151	19	22
F_5	28	245	239	6	0
F_6	12	48	47	1	0
Total	213	1624	1121	81	422

3) Optimizing for Multi-objective RTP: To find suitable test orderings that optimize several of the outlined objectives to achieve better results in terms of APFD_C, prior research suggests the use of search-based optimization techniques, such as genetic algorithms [34], [36], ML models [23], or combined, multivariate heuristics [10].

To get a rough understanding of which group of techniques might be suitable in the given context, we apply two genetic algorithms (NSGA-II, TAEA), three ML models (multivariate linear regression, random forest regression, SVM regression), and several combined heuristics (see Sec. IV-B3).

IV. Evaluation & Case Study

To evaluate the RTP approaches presented in Sec. III, we analyze historical test data from different vehicle functions that use system-level regression testing as provided by our industrial partner MAN. Our research endeavour is steered by the following Research Questions (RQs):

- **RQ_1:** How well do RTP approaches solely based on failure history perform in automotive system-level regression testing?
- **RQ_2:** How does domain-specific failure severity assessment affect results compared to traditional RTP assessment?
- **RQ_3:** How do severity-aware, multi-objective approaches compare to single-objective RTP approaches?

A. Experimental Setup

1) Dataset: The dataset provided by MAN comprises 213 test cases covering six vehicle functions. As shown in Table I, the functions differ strongly both in terms of the existing test cases (from 58 to 12) and the actual test executions (from 413 to 48). The test cases are executed as the development of the vehicle function progresses for each release, although not all test cases are executed in each release due to time constraints. We use the results of 12 releases covering roughly three years of development as there is a new major release every three months.

The distribution of test verdicts for each release is shown in Fig. 2. It is noticeable that the number of failures and

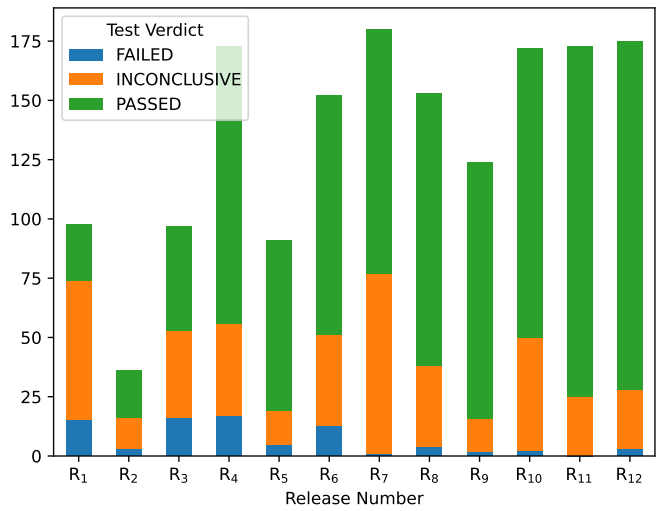


Fig. 2: Number of test executions and distribution of verdicts per release

inconclusive test results decreases as the development period progresses.

As outlined in Sec. III-B, some particularly important test cases are labelled as smoke tests. Our dataset contains 62 such smoke tests. Regarding the linked requirements, in total 18 test cases in our dataset cover at least one of the particularly important legal or safety-critical requirements.

2) Rolling Evaluation: To make the best use of the available data and to show the progression over time of the different RTP approaches, we use a *rolling evaluation* setup. Starting with the second release[2], we successively evaluate each RTP approach on each of the remaining releases. Approaches that use historical test verdicts only have access to test results from the previous releases. This allows us to simulate how such RTP approaches would have performed at a particular point in time [16]. Notably, for ML approaches, this implies that when evaluating a given release, we use all previous releases as the training set for the learning algorithm.

In addition to the evaluation across all vehicle functions, we report the results for a single function as well. We select F_2 for this purpose (see Table I), since it has the highest number of test executions and there is at least one failure for every release available.

Since several of our approaches are not entirely deterministic due to inherent randomness, we repeat the experiments multiple times and report the mean across 30 runs [37].

B. Compared RTP Approaches

An overview of the 18 applied RTP approaches is shown in Table II. All presented RTP approaches are black-box techniques, since they do not utilize any source code information. The approaches can be grouped into baselines, heuristics, ML models, and search-based algorithms.

[2]Since most RTP approaches need at least one historical test execution, the first release cannot be used here.

1) Baselines: The first group are baseline approaches. Since we have no information regarding the actual order in which the test cases were run, we employ the following baselines:

- $\mathbf{B}_0$ runs the test suite in the *optimal* order by sorting the tests in descending order of the failure severity score, as defined in Sec. III-B2. Note that this is not a realistic RTP approach in practice, since the severity formula contains the current test's verdict, which is unknown before executing the test. Nevertheless, this baseline returns the best prioritization that could have been achieved in theory, and therefore serves as a reference for the other approaches.
- $\mathbf{B}_1$ executes the tests in a *random* order. Any RTP strategy should outperform random ordering in order to be considered useful.
- $\mathbf{B}_2$ runs tests in *alphabetical* order of the test case names. This strategy is more common in practice than random ordering (and often the default in test management tools), since it is always deterministic and easy to implement.
- $\mathbf{B}_3$ is an *expert prioritization* baseline, derived from discussions with MAN experts about the crucial factors for test ordering decisions. With this strategy, tests that cover legal or safety-critical requirements are executed first, then tests labeled as smoke tests are executed, and then all others. Tests that cover legal or safety-critical requirements have the highest priority in this approach, since the effects of a failure can be enormously expensive, because they can delay legal approval, for example. Smoke tests find potential errors that are particularly relevant for developers and could impede further testing, which is why they are executed second.
- $\mathbf{B}_4$ executes tests in ascending order of the *test size* [10], [16]. With this approach, it is assumed that *small* and *cheap* are correlated variables and one achieves better cost-effectiveness by running the small tests (i.e., tests with the lowest number of steps) early.

2) Heuristics: The history-based surrogates presented in Sec. III-A2 can be used as prioritization heuristics, by ordering the tests in descending order of the respective value:

- $\mathbf{H}_1$ uses the historical failure frequency
- $\mathbf{H}_2$ uses the flip rate
- $\mathbf{H}_3$ uses the number of executions since the last failure

Additionally, we include two more heuristics:

- $\mathbf{H}_4$ is a modified version of H_1; it divides the failure frequency by the test cost (i.e., the number of test steps), and orders tests based on that *failure-frequency-to-cost* ratio.
- $\mathbf{H}_5$ is an *exponential smoothing* strategy proposed by Kim and Porter [38]; it also considers the historical failure frequency, but recent failures are weighted higher than older ones. P_k denotes the probability that a certain test will fail in release k. This value is recursively computed via the following scheme:

$$P_0 = h_1$$
$$P_k = \alpha h_k + (1-\alpha)P_{k-1}, \quad 0 \leq \alpha \leq 1, k \geq 1$$

TABLE II: Overview of prioritization approaches grouped by baselines, heuristics, ML models, and search-based algorithms

ID	Description	Type of Information		Multi-objective
		History-based	Cost-based	
B_0	Optimal order: descending order of failure severity		✓	
B_1	Random order			
B_2	Alphabetical order by test case name			
B_3	Expert order: (1) tests with critical requirements, (2) smoke tests, (3) others			
B_4	Ascending order of test cost (number of test steps)		✓	
H_1	Descending order of historical failure frequency	✓		
H_2	Descending order of flip rate	✓		
H_3	Ascending order of time (number of executions) since last failure	✓		
H_4	Descending order of historical failure frequency divided by test cost	✓	✓	✓
H_5	Descending order of failure history with exponential smoothing [38]	✓	✓	✓
M_1	Descending order of predicted failure probability (logistic regression)	✓	✓	
M_2	Descending order of predicted failure probability (random forest)	✓	✓	
M_3	Descending order of predicted failure probability (SVM)	✓	✓	
M_4	Descending order of predicted failure severity (linear regression)	✓	✓	✓
M_5	Descending order of predicted failure severity (random forest regression)	✓	✓	✓
M_6	Descending order of predicted failure severity (SVM regression)	✓	✓	✓
S_1	Pareto optimal order computed by NSGA-II	✓	✓	✓
S_2	Pareto optimal order computed by TAEA	✓	✓	✓

Here, h_k is 1 if the test failed in release k, and 0 otherwise. The constant α acts as an exponential decay factor; with higher values, recent failures are prioritized higher. Similar to H_4, the failure probability is also divided by the test cost.

If no historical data are available for a test case yet, these heuristics are initialized with their maximum value of 1.0 in order to prioritize tests executed for the first time.

3) Machine Learning Models: The surrogates which we use as heuristics can also be used as input features for (statistical) ML models. In general, there are two ways to formulate a supervised ML problem for RTP: first, we can formulate it as a binary classification problem, where a failure probability between 0 and 1 is predicted. Similar to prior work, we use three binary classifiers [15], [16], [39]; a logistic regression model ($\mathbf{M}_1$), a random forest ($\mathbf{M}_2$) and an SVM ($\mathbf{M}_3$). Second, we can also directly attempt to predict the failure severity, which can be formulated as a regression problem. By using a regression model targeting the failure severity, rather than a binary classification model, the model can learn to better discriminate between severe (high score) and uncritical (low score) test failures. We employ the three models linear regression ($\mathbf{M}_4$), random forest regression ($\mathbf{M}_5$) and SVM regression ($\mathbf{M}_6$).

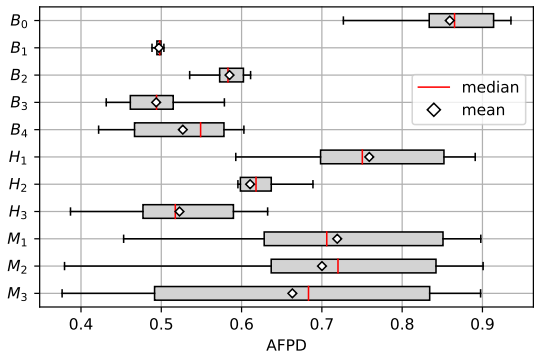
All models are trained on the same dataset containing four features, namely the surrogates we introduced in Sec. III-A2: *failure frequency*, *flip rate*, *time to last failure*, and *test size*.

4) Search-Based Optimization: Multiple objectives can be optimized together by finding *Pareto optimal* solutions [34], [35]. A solution is called Pareto optimal if no objective can be improved without deteriorating another objective. Genetic algorithms are a popular choice for this type of optimization problem [20]. We first employ a widely used, standard genetic algorithm called *NSGA-II* [40]. We also apply a second algorithm called *TAEA* [41], which addresses some of the limitations of NSGA-II and has been used for regression test optimization in the past [36], [42]. These algorithms ($\mathbf{S}_1$, $\mathbf{S}_2$) use the five objectives defined in Sec. III-B to generate Pareto optimal solutions:

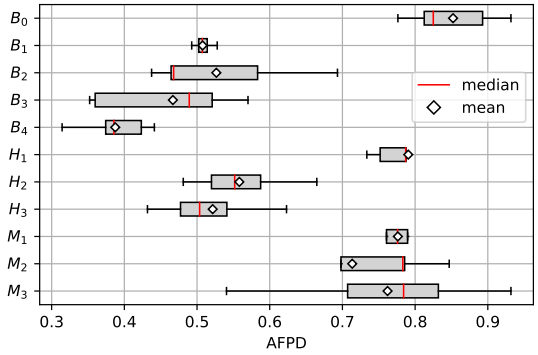
- `failures`: run tests with a high historical failure frequency first
- `all_reqs`: run tests with many linked requirements first
- `relevant_reqs`: run tests with safety-critical/legal requirements first
- `smokes`: run smoke tests first
- `size`: run tests with a low number of test steps first

C. Results

1) $\boldsymbol{RQ}_1$ – History-Based Prioritization: To answer RQ_1, we evaluate the performance of the five baselines (B_0 – B_4) against the single-objective approaches based on failure history (see Table II). These include the heuristics H_1 – H_3 and the ML models M_1 – M_3. This part of the evaluation uses the APFD metric, in which all tests are assumed to have the same execution costs, and the failure severity is treated equally for



(a) APFD across all functions



(b) APFD on function F_2

Fig. 3: Performance of the RTP approaches with the APFD over all releases

all failing tests. We thus consider *inconclusive* or *failed* tests as equally severe.

Fig. 3 shows the performance distribution of the investigated RTP approaches over all releases. Recall that B_0 is a reference baseline representing the best possible prioritization. The most noticeable observation is that the *failure frequency* heuristic (H_1) and the ML model (M_1) clearly outperform all other approaches, with an average APFD between 0.72 and 0.79. In comparison, the other two heuristics based on *flip rate* and *time to last failure* attain lower scores, between 0.52 and 0.61. With an APFD of 0.49 across all functions or 0.47 on the single function F_2, the baseline that replicates the experts' strategy (B_3) performs poorly and does not reach the score 0.50 of the random prioritization. The remaining baselines B_2 and B_4 are also not better than the random ordering. Overall, the simple and well studied heuristic (H_1) and the ML approaches (M_1–M_3) perform well, indicating that the historical failure frequency is a useful predictor for future failures.

In certain cases, the testing capacity can be limited and testers may not be able to execute all regression tests before a release. Therefore, we also report what fraction of the test suite needs to be executed to detect 90% of the failures (cf. Elsner et al. [16]). Fig. 4 shows the percentage of test cases needed to detect at least 90% of all failures across all functions; we compare only the two best history-based approaches (H_1 and

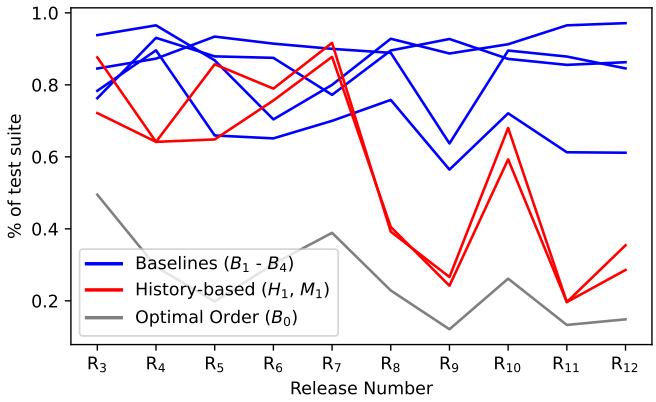


Fig. 4: Percentage of test suite required to detect at least 90% of all failures

M_1) to the baselines. The highest number of test cases is required by the baselines B_3 (expert order) and B_4 (test cost). The logistic regression model (M_1) and the failure frequency heuristic (H_1) achieve the best average percentage of 58%, meaning that on average 42% of the test suite can be skipped while still detecting 90% of all failures. As more historical data become available over time, the history-based approaches' performance improves, while the four realistic baselines show no such improvement. For the last five releases, the history-based algorithms can detect 90% of all failures with on average merely 38% of all test cases, resulting in a significant reduction in testing effort. Note that the remaining test cases are not permanently excluded from the test suite; they can still be executed in larger intervals. Fig. 4 shows a noticeable spike of the required testing effort in R_10. By inspecting the verdict distribution across releases (see Fig. 2) we notice a considerable increase in the number of *inconclusive* verdicts in R_{10}, which could explain this behavior.

> **RQ_1** *We find that RTP approaches based on test failure history outperform our baseline approaches. The baselines, even the one mimicking the experts' strategy, perform poorly, comparable to random ordering. For history-based approaches, both the failure frequency heuristic and the ML models perform similarly well. In a capacity-constrained scenario, on average only 58% of all test cases need to be executed to still detect 90% of all failures.*

*2) **RQ_2** – Domain-Specific Failure Severity:* To answer RQ_2, we assess how the evaluation results change if we use the severity-aware $APFD_C$ metric, based on the test execution costs (i.e., the sum of test steps) and the domain-specific failure severity model (see Sec. III-B). Table III shows the severity-aware ($APFD_C$) and severity-unaware (APFD) performance for all baselines, heuristics, and ML models considered in RQ_1. The reported numbers represent the average across all releases from the rolling evaluation.

TABLE III: Mean APFD/APFD$_C$ scores across all functions and for function F_2 (results for RQ_2 and RQ_3)

	All functions		Single function (2)	
Approach	**APFD**	**APFD$_C$**	**APFD**	**APFD$_C$**
B_0	*0.86*	*0.92*	*0.85*	*0.89*
B_1	0.50	0.50	0.50	0.51
B_2	0.58	0.54	0.53	0.44
B_3	0.49	0.56	0.47	0.56
B_4	0.53	0.68	0.39	0.55
H_1	**0.76**	0.70	**0.79**	0.73
H_2	0.61	0.61	0.56	0.53
H_3	0.52	0.49	0.52	0.48
H_4	0.72	0.76	0.76	0.77
H_5	0.73	**0.78**	0.78	**0.80**
M_1	0.72	0.65	0.78	0.70
M_2	0.70	0.66	0.71	0.67
M_3	0.66	0.71	0.76	0.70
M_4	0.66	0.68	0.73	0.73
M_5	0.61	0.67	0.58	0.62
M_6	0.63	0.69	0.71	0.71
S_1	0.66	0.72	0.69	0.72
S_2	0.63	0.70	0.58	0.69

Not surprisingly, the experts' prioritization baseline (B_3) performs better in the severity-aware setting compared to the APFD, since both the expert strategy and the evaluation metric take the same factors into account; smoke tests and tests with critical requirements contribute more to the failure severity than the other tests. Likewise, the result for the *test cost* baseline (B_4) improves, due to the fact that the APFD$_C$ metric takes execution costs into account. For the alphabetical order (B_2), the performance deteriorates slightly across all functions, and noticeably on the single function F_2. Alphabetical sorting can result in similar tests being grouped together, which can lead to a worse performance than random sorting.

The three single-objective heuristics (H_1 – H_3) perform worse in the severity-aware evaluation than in the severity-unaware evaluation. This is because all three heuristics take neither the cost nor the failure severity into account.

The three single-objective ML models (M_1 – M_3) almost always perform worse in the severity-aware setting; the exception is M_3, which shows a slight improvement across all functions. This lower performance with the APFD$_C$ metric is expected, as these RTP approaches do not incorporate any notion of test cost or failure severity.

RQ$_2$ *All investigated RTP approaches that do not take domain-specific knowledge into account perform worse in the severity-aware evaluation. Our two baselines making use of the test cost (B_4) and the failure severity (B_3) benefit from the severity-aware evaluation.*

TABLE IV: Mean APFD$_C$ scores obtained by NSGA-II for the top ten objective subsets and for all objectives (across all functions and releases)

Objectives	**APFD$_C$**
`size, failures`	0.72
`size, relevant_reqs`	0.67
`failures, relevant_reqs`	0.67
`size, failures, smokes`	0.66
`size, failures, relevant_reqs`	0.64
`size, failures, all_reqs`	0.64
`size, relevant_reqs, smokes`	0.61
`size, failures, relevant_reqs, smokes`	0.60
`failures, smokes`	0.60
`failures, relevant_reqs, smokes`	0.60
All Objectives	0.55

*3) **RQ$_3$** – Multi-Objective Optimization:* We also investigate the performance of approaches that target optimization of multiple domain-specific objectives as described in Sec. III-B. As indicated in Table II, these approaches are

- H_4 and H_5, two heuristics which seek to minimize the test cost
- M_4–M_6, three ML models which predict the failure severity instead of the failure probability
- S_1 and S_2, two genetic algorithms which aim to find a Pareto optimal prioritization satisfying all objectives

The performance of genetic algorithms can deteriorate as the number of objectives increases [43]. Since some of our objectives might be less relevant for a good prioritization, we first perform an exhaustive evaluation of all subsets of objectives for NSGA-II. Since we are considering five different objectives, there are 26 objective subsets with at least two elements. Table IV shows the APFD$_C$ scores of the best ten objective subsets obtained by NSGA-II. For comparison, the last row shows the performance of all five objectives together. The subset (`size, failures`) outperforms all others with an average score of 0.72, and is better than the set of all five objectives, which achieves only 0.55. This matches our expectation, since our APFD$_C$ metric evaluates the detected failure severity against the test cost: The most intuitive objectives would be to prioritize tests with high failure frequency and low test size.

Based on the results of the subset selection, we use only the best objective subset (*size, failures*) for the genetic algorithms. Table III shows the severity-aware performance comparison of the multi-objective approaches, for all functions and on F_2. Among the multi-objective approaches, the heuristics H_4 (*failure-frequency-to-cost ratio*) and H_5 (*weighted failure history*) achieve the best APFD$_C$ scores between 0.76 and 0.80, even outperforming all single-objective approaches. Compared to the optimal baseline B_0, which represents the theoretically

best prioritization, H_5 achieves 85% of the best possible $APFD_C$ across all functions on average, and 90% on F_2. The genetic algorithms perform comparably to the ML models, yet worse than the two heuristics. NSGA-II (S_1) was able to reach higher scores than TAEA (S_2). All multi-objective techniques show at least a slight improvement in the severity-aware setting ($APFD_C$) compared to the traditional APFD metric.

RQ$_3$ *Simple multi-objective heuristics, such as the failure-frequency-to-cost ratio (H_4) and the weighted failure history (H_5) are among the best approaches, achieving up to 90% of the maximum possible cost-effectiveness. Multi-objective RTP with genetic algorithms performs on par with the ML models and the baseline approaches, with NSGA-II (S_1) slightly outperforming TAEA (S_2). The genetic algorithms obtain the best results when using two objectives, (1) minimizing the test size and (2) maximizing the historical failure frequency.*

D. Discussion

1) Run Time of Prioritization: For large test suites, we need to consider the run time of the different RTP approaches. Since the goal of RTP is to optimize the regression testing process, computing the test ordering must not be prohibitively expensive. In our evaluation, the genetic algorithms have the longest execution time, followed by the ML models. The heuristics and baselines were the fastest. The good performance of the two heuristics *failure frequency* (H_1) and *weighted failure history* (H_5) suggests that more sophisticated techniques, such as ML models or search-based techniques, might not be beneficial in practice, given their comparatively high run time.

2) Failure-to-Fault Mapping: The goal of regression testing is to detect as many faults as possible as early as possible and not to produce as many test failures as possible. As discussed in Sec. III-A1, in most cases, no mapping of failures to faults is available and, therefore, a one-to-one failure-to-fault mapping is commonly assumed [10], [16]. In our context, there are incomplete data about the underlying faults: although a bug ticket should be created in case of a failure, this happens irregularly. In total, tickets have been created for 30 of the 81 *failed* test cases, and for 6 of the 422 *inconclusive* results. Due to these irregularities and missing consistency in the dataset, we refrain from using linked bug ticket information to assign faults to failures.

E. Threats to Validity

We identify several internal and external factors which could adversely affect the validity of our evaluation.

First, the cost of a test execution can consist of several factors, such as the execution time, the setup time, and technical resource costs. However, we currently only have access to the number of test steps, which we use to approximate the test costs. Since the durations of these steps are not equal, and can vary across different test runs, they might not be an accurate estimation of the true test costs.

Non-deterministic tests, also called flaky tests, are a well-known problem in automated testing. However, a recent developer survey has shown that flaky tests can also be present in manual test suites [21]. Since many of the RTP approaches used in this work utilize past test verdicts, the presence of flaky tests could diminish our evaluation results.

Another threat to validity is the randomness inherent in some of the approaches we apply, e.g., genetic algorithms, ML models, and random prioritization. To mitigate this threat and make our evaluation more robust, we repeated the experiments multiple times and reported the mean across 30 runs.

In our evaluation, we limit ourselves to reporting results from a few ML models, such as linear and logistic regression, SVMs, and random forest models. The rationale behind this limited set of algorithms is that prior research has found that differences between models are often insignificant [16] and we experienced similar trends during our experiments. We also did not yet incorporate reinforcement learning models in our evaluation, which might improve the results [12], [19].

Last, the failure severity model defined in Sec. III-B was instantiated with values derived from discussions with domain experts from MAN. We assume that these values are context-specific to MAN data and might not generalize to other automotive software systems. Nevertheless, our evaluation does not require a specific severity model and can easily be extended to support any other failure severity formula or instantiation of our severity model.

V. Related Work

Several RTP approaches have been proposed over the past two decades. In the following, we discuss related work and outline the research gaps with respect to automotive system-level regression testing.

Rothermel et al. [29] and Elbaum et al. [30] were among the first to study different RTP approaches and coined the APFD metric to evaluate these approaches. The traditional APFD metric was then extended to a cost-cognizant variant, $APFD_C$, which incorporates varying test costs and fault severities [28].

While these early studies discussed purely white-box, coverage-based RTP techniques on C programs, Kim and Porter [38] were the first to study history-based RTP, which uses statistical ranking models for tests based on past-fault coverage, i.e., test history, and function coverage. In 2014, Elbaum et al. [17] reported that even simpler history-based RTP techniques that merely rank tests based on how long ago they last failed, performed remarkably well in CI environments at Google. These results were, however, diminished later by Leong et al. [33], who discovered that many of the detected test failures were in fact caused by flaky tests. Elsner et al. [16] empirically evaluate the cost-effectiveness of heuristics and ML models based on readily available information from CI and Version Control System (VCS). They show that simple heuristics such as the ones proposed by Elbaum et al. [17] often outperform more sophisticated ML models. Yu et al. [44] evaluate multiple black-box techniques for prioritizing automated UI tests and find that history-based heuristics are among

the best approaches in terms of $APFD_C$. Najafi et al. [45] investigate how history-based techniques can be leveraged for both selection and prioritization; they identify the association between failures as the most valuable information in the test history. The compared baseline models in our work are also partially taken from Peng et al. [10], who empirically study RTP approaches based on information retrieval techniques, first proposed by Saha et al. [9].

Yoo and Harman [34], [35], [46], [47] propose a paradigm shift to formulate RTP as a multi-objective problem to respect different (competing) testing objectives (e.g., structural and functional testing) instead of a single-objective problem (e.g., code coverage). They further suggest to incorporate various kinds of costs, such as resource costs or test duration, into the optimization process to select tests or to find suitable test orderings [34]. Also, they propose to use search algorithms, such as genetic algorithms, to find Pareto optimal solutions, with respect to multiple objective functions. Several ideas in this paper are taken from their works, specifically the idea of prioritizing by historical fault revelation. Epitropakis et al. [36] apply multi-objective RTP techniques in an empirical study on six programs with mostly seeded faults. They use the three objective functions: statement coverage, fault history coverage, and execution cost (approximated by number of executed instructions). The comparison to traditional greedy coverage-based RTP indicates that multi-objective evolutionary algorithms significantly outperform these single-objective RTP approaches in 14 out of the 22 studied software versions across programs. In a case study at Cisco, Wang et al. [48] compare seven different search algorithms to approach their resource-aware multi-objective RTP problem, where they construct their fitness function from four cost-effectiveness measures. They find that the best performing genetic algorithm reduces the time for test resource allocation and test execution by on average 40.6% compared to the state-of-practice.

The research on RTP in system-level testing is relatively sparse. Haas et al. [21] report that especially if system-level testing is performed manually, few attempts have been started to transfer existing regression test optimization techniques to (manual) system-level testing [22], [49]. We consider Lachmann et al. [20] the closest to our work: although their ultimate goal is test selection rather than prioritization, their study, similar to ours, targets the automotive software engineering domain. They use seven different objective functions based on black-box metadata; we use several of them as well (see Sec. III). In contrast to our work, they permute the set of selected tests using genetic algorithms to find Pareto optimal subsets of the entire test suite, while we permute the orderings of tests. They further evaluate their test selection with respect to precision and recall in failure detection. We incorporate a failure severity model into the $APFD_C$ metric to evaluate test orderings, which allows more domain-specific investigation of trade-offs between multiple objectives.

In summary, we are not aware of any prior work that discusses multi-objective RTP in the context of automotive software engineering and evaluates RTP techniques using domain-specific failure severity models. Moreover, most existing work compares against random ordering baselines, whereas we use domain-specific baseline orderings, based on safety-critical requirements and development relevance of test cases.

VI. Conclusion

In this study, we discuss aspects of how RTP approaches need to be designed and evaluated in the context of system-level regression testing for automotive software systems. Contrary to most existing research, we incorporate varying fault severities into the evaluation of RTP approaches and attempt to derive realistic RTP baselines. We present a case study at our industry partner MAN, where we comparatively assess single- and multi-objective RTP approaches with respect to severity-aware and traditional evaluation metrics. We find that simple history-based RTP heuristics perform considerably well in both the severity-aware and the severity-unaware setting. ML-based and search-based multi-objective RTP approaches do not outperform simple heuristics and only achieve competitive performance once sufficient historical test verdicts are available. Hence, even if only relying on readily available black-box information, RTP reduces the feedback time at MAN compared to the current testing strategies while retaining failure detection.

VII. Acknowledgements

This work was partially funded by the German Federal Ministry of Education and Research (BMBF), grant "Q-SOFT, 01IS22001B". The responsibility for this article lies with the authors.

References

[1] S. Yoo and M. Harman, "Regression testing minimization, selection and prioritization: A survey," *Software Testing Verification and Reliability*, vol. 22, no. 2, pp. 67–120, 2012.

[2] G. Rothermel, R. H. Untch, C. Chu, and M. J. Harrold, "Prioritizing Test Cases for Regression Testing," *IEEE Transactions on Software Engineering*, vol. 27, no. 10, pp. 929–948, 2001.

[3] B. Jiang, Z. Zhang, W. K. Chan, and T. H. Tse, "Adaptive random test case prioritization," in *Proceedings of the International Conference on Automated Software Engineering*, 2009, pp. 233–244.

[4] D. Di Nardo, N. Alshahwan, L. Briand, and Y. Labiche, "Coverage-based test case prioritisation: An industrial case study," in *Proceedings of the International Conference on Software Testing, Verification and Validation*, 2013, pp. 302–311.

[5] ——, "Coverage-based regression test case selection, minimization and prioritization: A case study on an industrial system," *Software Testing, Verification and Reliability*, vol. 25, no. 4, pp. 371–396, 2015.

[6] C. Henard, M. Papadakis, M. Harman, Y. Jia, and Y. L. Traon, "Comparing white-box and black-box test prioritization," in *Proceedings of the International Conference on Software Engineering*, 2016, pp. 523–534.

[7] Y. Ledru, A. Petrenko, S. Boroday, and N. Mandran, "Prioritizing test cases with string distances," in *Automated Software Engineering*, vol. 19, no. 1, 2012, pp. 65–95.

[8] B. Miranda, E. Cruciani, R. Verdecchia, and A. Bertolino, "FAST approaches to scalable similarity-based test case prioritization," in *Proceedings of the International Conference on Software Engineering*, 2018, pp. 222–232.

[9] R. K. Saha, L. Zhang, S. Khurshid, and D. E. Perry, "An information retrieval approach for regression test prioritization based on program changes," in *Proceedings of the International Conference on Software Engineering*, 2015, pp. 268–279.

[10] Q. Peng, A. Shi, and L. Zhang, "Empirically Revisiting and Enhancing IR-Based Test-Case Prioritization," in *Proceedings of the International Symposium on Software Testing and Analysis*, 2020, pp. 324–336.

[11] T. Mattis and R. Hirschfeld, "Lightweight Lexical Test Prioritization for Immediate Feedback," *The Art, Science, and Engineering of Programming*, vol. 4, no. 3, pp. 12:1–12:32, 2020.

[12] A. Bertolino, A. Guerriero, R. Pietrantuono, S. Russo, B. Miranda, and R. Pietran-Tuono, "Learning-to-Rank vs Ranking-to-Learn: Strategies for Regression Testing in Continuous Integration," in *Proceedings of the International Conference on Software Engineering*, 2020, pp. 1–12.

[13] S. Wang, J. Nam, and L. Tan, "QTEP: Quality-aware test case prioritization," in *Proceedings of the Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2017, pp. 523–534.

[14] E. Knauss, M. Staron, W. Meding, O. Soder, A. Nilsson, and M. Castell, "Supporting Continuous Integration by Code-Churn Based Test Selection," in *Proceedings of the International Workshop on Rapid Continuous Software Engineering*, 2015, pp. 19–25.

[15] A. A. Philip, R. Bhagwan, R. Kumar, C. S. Maddila, and N. Nagppan, "FastLane: Test Minimization for Rapidly Deployed Large-Scale Online Services," in *Proceedings of the International Conference on Software Engineering*, 2019, pp. 408–418.

[16] D. Elsner, F. Hauer, A. Pretschner, and S. Reimer, "Empirically Evaluating Readily Available Information for Regression Test Optimization in Continuous Integration," in *Proceedings of the International Symposium on Software Testing and Analysis*, 2021, pp. 491–504.

[17] S. Elbaum, G. Rothermel, and J. Penix, "Techniques for improving regression testing in continuous integration development environments," in *Proceedings of the International Symposium on the Foundations of Software Engineering*, 2014, pp. 235–245.

[18] T. B. Noor and H. Hemmati, "A similarity-based approach for test case prioritization using historical failure data," in *Proceedings of the International Symposium on Software Reliability Engineering*, 2016, pp. 58–68.

[19] H. Spieker, A. Gotlieb, D. Marijan, and M. Mossige, "Reinforcement learning for automatic test case prioritization and selection in continuous integration," in *Proceedings of the International Symposium on Software Testing and Analysis*, 2017, pp. 12–22.

[20] R. Lachmann, S. Schulze, M. Felderer, C. Seidl, M. Nieke, and I. Schaefer, "Multi-objective black-box test case selection for system testing," in *Proceedings of the Genetic and Evolutionary Computation Conference*. Association for Computing Machinery, 2017, pp. 1311–1318.

[21] R. Haas, D. Elsner, E. Juergens, A. Pretschner, and S. Apel, "How can manual testing processes be optimized? Developer survey, optimization guidelines, and case studies," in *Proceedings of the Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2021, pp. 1281–1291.

[22] H. Hemmati, Z. Fang, and M. V. Mäntylä, "Prioritizing manual test cases in traditional and rapid release environments," in *Proceedings of the International Conference on Software Testing, Verification and Validation*, 2015, pp. 1–10.

[23] R. Lachmann, M. Nieke, C. Seidl, I. Schaefer, and S. Schulze, "System-level test case prioritization using machine learning," in *Proceedings of the International Conference on Machine Learning and Applications*, 2016, pp. 361–368.

[24] R. Lachmann, "Machine Learning-Driven Test Case Prioritization Approaches for Black-Box Software Testing," in *Proceedings of the European Test and Telemetry Conference*, may 2018, pp. 300–309.

[25] A. Celik, M. Vasic, A. Milicevic, and M. Gligoric, "Regression test selection across JVM boundaries," in *Proceedings of the Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2017, pp. 809–820.

[26] M. Machalica, A. Samylkin, M. Porth, and S. Chandra, "Predictive Test Selection," in *Proceedings of the International Conference on Software Engineering: Software Engineering in Practice*, 2019, pp. 91–100.

[27] D. Elsner, R. Wuersching, M. Schnappinger, A. Pretschner, M. Graber, R. Dammer, and S. Reimer, "Build System Aware Multi-language Regression Test Selection in Continuous Integration," in *Proceedings of the International Conference on Software Engineering: Software Engineering in Practice*, 2022, pp. 87–96.

[28] S. Elbaum, A. Malishevsky, and G. Rothermel, "Incorporating varying test costs and fault severities into test case prioritization," in *Proceedings of the International Conference on Software Engineering*, 2001, pp. 329–338.

[29] G. Rothermel, R. H. Untch, C. Chu, and M. J. Harrold, "Test case prioritization: an empirical study," in *Proceedings of the International Conference on Software Maintenance*. IEEE Computer Press, 1999, pp. 179–188.

[30] S. Elbaum, A. G. Malishevsky, and G. Rothermel, "Prioritizing test cases for regression testing," in *Proceedings of the International Symposium on Software Testing and Analysis*, 2000, pp. 101–112.

[31] T. Mattis, F. Dürsch, and R. Hirschfeld, "Faster feedback through lexical test prioritization," in *Companion of the International Conference on Art, Science, and Engineering of Programming*, 2019, pp. 1–10.

[32] A. Shi, A. Gyori, S. Mahmood, P. Zhao, and D. Marinov, "Evaluating Test-Suite Reduction in Real Software Evolution," in *Proceedings of the International Symposium on Software Testing and Analysis*, 2018, pp. 84–94.

[33] C. Leong, A. Singh, M. Papadakis, Y. Le Traon, and J. Micco, "Assessing Transition-Based Test Selection Algorithms at Google," in *Proceedings of the International Conference on Software Engineering: Software Engineering in Practice*, 2019, pp. 101–110.

[34] M. Harman, "Making the case for MORTO: Multi objective regression test optimization," in *Proceedings of the International Conference on Software Testing, Verification, and Validation Workshops*, 2011, pp. 111–114.

[35] S. Yoo and M. Harman, "Pareto efficient multi-objective test case selection," in *Proceedings of the International Symposium on Software Testing and Analysis*. ACM Press, 2007, pp. 140–150.

[36] 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 International Symposium on Software Testing and Analysis*, 2015, pp. 234–245.

[37] A. Arcuri and L. Briand, "A practical guide for using statistical tests to assess randomized algorithms in software engineering," in *Proceedings of the International Conference on Software Engineering*, Honolulu, HI, 2011, pp. 1–10.

[38] J. M. Kim and A. Porter, "A history-based test prioritization technique for regression testing in resource constrained environments," in *Proceedings of the International Conference on Software Engineering*, 2002, pp. 119–129.

[39] T. B. Noor and H. Hemmati, "Studying test case failure prediction for test case prioritization," in *Proceedings of the International Conference on Predictive Models and Data Analytics in Software Engineering*, 2017, pp. 2–11.

[40] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, 2002.

[41] K. Li, R. Chen, G. Fu, and X. Yao, "Two-archive evolutionary algorithm for constrained multiobjective optimization," *IEEE Transactions on Evolutionary Computation*, vol. 23, pp. 303–315, 4 2019.

[42] S. Yoo, M. Harman, and S. Ur, "Gpgpu test suite minimisation: search based software engineering performance improvement using graphics cards," *Empirical Software Engineering*, vol. 18, pp. 550–593, 6 2013.

[43] K. Praditwong and X. Yao, "A new multi-objective evolutionary optimisation algorithm: The two-archive algorithm," in *2006 International Conference on Computational Intelligence and Security*. IEEE, 11 2006, pp. 286–291.

[44] Z. Yu, F. Fahid, T. Menzies, G. Rothermel, K. Patrick, and S. Cherian, "TERMINATOR: Better automated UI test case prioritization," in *Proceedings of the Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2019, pp. 883–894.

[45] A. Najafi, W. Shang, and P. C. Rigby, "Improving Test Effectiveness Using Test Executions History: An Industrial Experience Report," in *Proceedings of the International Conference on Software Engineering: Software Engineering in Practice*, 2019, pp. 213–222.

[46] S. Yoo and M. Harman, "Using hybrid algorithm for Pareto efficient multi-objective test suite minimisation," *Journal of Systems and Software*, vol. 83, no. 4, pp. 689–701, 2010.

[47] S. Yoo, R. Nilsson, and M. Harman, "Faster Fault Finding at Google Using Multi Objective Regression Test Optimisation," in *Proceedings of the International Symposium on the Foundations of Software Engineering*, 2011.

[48] S. Wang, S. Ali, T. Yue, O. Bakkeli, and M. Liaaen, "Enhancing test case prioritization in an industrial setting with resource awareness and multi-objective search," in *Proceedings of the International Conference*

on Software Engineering. IEEE Computer Society, may 2016, pp. 182–191.

[49] D. Elsner, D. Bertagnolli, A. Pretschner, and R. Klaus, "Challenges in Regression Test Selection for End-to-End Testing of Microservice-based Software Systems," in *Proceedings of the International Conference on Automation of Software Test*, 2022, pp. 1–5.