Noname manuscript No.
(will be inserted by the editor)

Finding the right regression testing method: a taxonomy-based approach

Maria Laura Brzezinski Meyer · Hélène Waeselynck · Fernand Cuesta

Received: date / Accepted: date

Abstract With numerous regression testing (RT) methods available in the literature, it is challenging to choose the right one for a specific context. Practitioners need support identifying suitable research. To this end, recent work has proposed a taxonomy. By mapping both the RT problem and existing solutions onto the taxonomy, practitioners should be able to determine which solutions are best aligned with their problem. Our work explores the practical relevance of this idea through an industrial case study. The context is the development of R&D projects at a major automotive company, in the domain of connected vehicles. We developed an RT problem solving approach based on the taxonomy. Following the approach, we characterized the RT problem, identified a set of 8 potentially relevant solutions from a set of 52 papers, and empirically evaluated their suitability. Our approach was successful, as we found effective RT methods among those selected using the taxonomy. One method, in particular, demonstrated remarkable robustness across various datasets, making it a strong recommendation for the industrial partner. However, this success came at the cost of difficulties due to unclear taxonomy elements, missing elements, and paper classification errors. We conclude that the taxonomy has practical value but would have to mature for easier applicability.

Keywords Regression testing · Taxonomy · Industrial case study · Test case prioritization · Test case selection

Maria Laura Brzezinski Meyer
Ampere Software Technology, Toulouse, France
LAAS-CNRS, Toulouse, France
E-mail: laurabrzmeyer@gmail.com

Hélène Waeselynck
LAAS-CNRS, Toulouse, France
E-mail: helene.waeselynck@laas.fr

Fernand Cuesta
Ampere Software Technology, Toulouse, France
E-mail: fernand.cuesta@ampere.cars

1 Introduction

Regression testing (RT) checks that the changes in the code did not introduce new issues, that is, the software did not regress. Given the impracticality of re-running all tests, there is a clear need for testing optimization. This is known as the RT problem and has been an active field of research for decades.

The RT methods can be divided into three categories (Yoo and Harman, 2012): Test Suite Minimization (TSM), Test Case Selection (TCS), and Regression Test Prioritization (RTP). TSM aims to reduce the number of tests by removing redundant ones. TCS addresses the dual problem of choosing which tests to run. This selection is often related to the code changes: tests that cover the modified parts of the code are selected. RTP neither removes nor selects tests, but rather assigns them an order of execution seeking to maximize early fault exposure. An RT strategy may hybridize TSM, TCS, and RTP. For example, a minimization technique can be applied after the selection to further reduce the number of tests. Alternatively, all tests can be prioritized but only the highest-priority ones are selected. Conceptually, the three categories of methods are closely related. All of them involve comparative characterization of the tests, based on which an optimal decision has to be made (which tests to remove, retain, or execute first).

In the article titled "*On the search for industry-relevant regression testing research*" (Ali et al., 2019), the authors noted that, despite the substantial body of work on the RT problem, dissemination in the industry is limited. They identified two main difficulties faced by practitioners. The first one stems from the discrepancies in terminology between academia and industry, which makes it harder for practitioners to search for methods in the literature. The second difficulty is assessing the relevance of the methods. Research is usually conducted in a controlled environment that differs from complex industrial scenarios.

In order to help practitioners to interpret, compare, and contrast the methods in the literature, Ali et al. (2019) created a taxonomy. When both the RT problem and existing solutions are mapped onto the taxonomy, practitioners may more easily determine which solutions are best aligned with their problem. The taxonomy has three parts to capture the practitioners' concerns: the industrial *context*, the desired *effects* of RT methods, and the input *information* they need (e.g., code changes, ...).

This paper builds on the work by Ali et al. (2019) to investigate a taxonomy-based approach that searches for suitable RT solutions. The approach starts by a characterization of the RT problem in terms of taxonomy elements. It then extracts the applicable and relevant RT research from a set of papers, based on their mapping onto the taxonomy. Finally, it evaluates the candidate solutions on industrial datasets and issue recommendations. We empirically study the application of the approach to a real RT problem. The context is the development of R&D projects at Renault for connected vehicles – specifically, in a Renault entity dedicated to intelligent electric vehicles: Ampere Software Technology, founded in November 2023. We report on both the beneficial aspects and difficulties of applying the approach in this context. Some difficulties stem from the taxonomy itself, due to unclear taxonomy elements, missing ones, and paper classification errors. Others are due to industrial constraints, which we had to accommodate in the evaluation of the RT methods.

The entire study spanned two years. After the first year, we published intermediate results (Brzezinski Meyer et al., 2023), which included an analysis of the industrial problem, the selection of potentially relevant RT methods, and preliminary considerations for their evaluation. At that point, we were unable to determine the overall usefulness of the taxonomy-based approach, as it was unknown whether any of the selected methods would successfully address the industrial problem. This paper extends our previous work by incorporating the evaluation of the methods conducted during the second year of the study. The process involved extracting multiple datasets for different types of tests (manual or automated, exercising different systems embedded in the connected vehicle), creating an evaluation plan that was compatible with industrial constraints, performing the evaluation (we evaluated a total of 40 variants of the eight selected methods), and analyzing the results. Through this process, we were able to confirm that the taxonomy-based approach did, in fact, lead to successful problem-solving in the given context. Among the methods selected by the approach, we found several that proved effective. Interestingly, in this industrial case, simple heuristics outperformed more sophisticated methods, including those based on machine learning.

The paper provides an integrated account of all results obtained during the two years. The main contributions of this work are:

- A taxonomy-based approach to identify which RT solutions from the literature are best suited for a given industrial context;
- An empirical study of this approach, through the complete analysis of an industrial RT problem, from its characterization to solution recommendation;
- As part of this empirical study, a thorough evaluation of the suitability of the candidate solutions for the industrial RT problem;
- Feedback on using the taxonomy to assist in a real RT problem;
- A GitHub repository that contains the implementation of all methods evaluated, in order to enable reuse with other datasets.

The remainder of this paper is organized as follows. Section 2 presents the background and context that motivated this work. Section 3 outlines the methodology we established to apply and assess the taxonomy. Section 4 characterizes the industrial RT problem, focusing on its context and desired effects. In Section 5, we analyze the feasibility of the existing methods from the literature, based on the information items they require. Section 6 explores how the desired effects can be measured within an industrial context. Section 7 details the candidate methods selected for the experimental evaluation. In Section 8, we evaluate the retained solutions in terms of their ability to achieve the desired effects. Section 9 discusses threats to validity, and Section 10 concludes the paper.

2 Background and Related Work

Since the early papers in the 90s, regression testing has been widely studied. Many surveys and systematic reviews have been published to keep pace with the evolving research in the area. In Ali et al. (2019), the authors identified eleven reviews of software RT literature between 2010 and 2017. New ones have been published since that time, e.g., Khatibsyarbini et al. (2018), Lima and Vergilio (2020), Pan et al. (2022), and Greca et al. (2023). The latter authors (Greca et al., 2023) noted that literature reviews may quickly become outdated as new research emerges. To remedy this problem, they have created an online *live* repository of RT studies, which they intend to maintain and update annually.

The growing number of surveys and systematic reviews, as well as the constant need for updates, reflects the substantial volume and continuous growth of the regression testing literature. In this rich body of research, the proposed RT methods are diverse. They differ in the (combination of) criteria to compare the tests: code coverage, code change coverage, criticality of tested requirements, similarity of tests, cost, historical effectiveness, ... The core RT algorithms are also diverse, including predefined strategies, strategies optimized by a metaheuristic search, as well as strategies based on machine learning.

With so many methods, it is difficult to know which ones to select and apply in a specific context. According to the principle of Evidence-Based Software Engineering (EBSE) (Dyba et al., 2005), decision-making and practices should be grounded in the best available scientific evidence. To this end, EBSE advocates the use of systematic literature reviews (SLRs), which synthesize findings from multiple empirical studies through a well-defined methodology. However, SLRs may not be easily digestible for practitioners, and most address abstract research questions rather than provide actionable guidance (Le Goues et al., 2018). To better support decision-making, some authors have explored taxonomies to structure and contextualize the presentation of research insights in a way that is meaningful to practitioners. For instance, Forward and Lethbridge (2008) proposed a structuration in terms of software types to which the findings are applicable. Şmite et al. (2014) developed a taxonomy for research in global software engineering, distinguishing various global sourcing situations that practitioners may encounter. Closer to our focus, Engström et al. (2017) and Ali et al. (2019) introduced taxonomies for industry-relevant research in software testing in general, and software regression testing in particular. These testing-related taxonomies integrate both problem- and solution-oriented aspects. The intent is to facilitate connections between the two and make it easier for practitioners to find relevant research that matches their specific needs.

In this work, we consider the taxonomy by Ali et al. (2019), which specifically targets problems and solutions in software regression testing. The taxonomy focuses on industrial relevance and applicability. It is divided into three parts: context, effect, and information. The purpose of the *context* part is to identify the characteristics of an industrial environment that make regression testing challenging. Three context factors are identified: system-related (size, complexity and type), process-related (testing process and test techniques), and people-related (organizational and cognitive factors). The second part of the taxonomy is the desired *effect* of a method, that is, the measurable improvement reported with its implementation. The effects are divided into three categories: test coverage related, test efficiency and effectiveness, and awareness. Together, the context and desired effects characterize an RT problem. Then, the applicability of candidate solutions is determined by the third part of the taxonomy. The solutions are characterized by the *information* they utilize, like: development artifacts (from requirements to binary code), information about test cases, test execution attributes, test report, and issues revealed by tests. If some information item is not available in the industrial context, then the method is not feasible in this context. The information part of the taxonomy is the most detailed one, representing 50 out of a total of 68 attributes for all parts (see these 50 attributes in Table 5).

The building of the taxonomy followed a principled approach, starting from a systematic literature review and involving practitioners at several steps. The authors also demonstrated the mapping of 38 RT papers to the taxonomy. However, they did not demonstrate the application of their taxonomy to address RT problems in real-world scenarios. This paper provides such a real-world scenario. The next section presents the methodology proposed to study the use of the taxonomy.

3 Methodology

This study adopts the point of view of an industrial company wishing to improve its RT process. Given the large number of options available in the literature, we consider that the company does not need to develop new RT methods. Rather, the key issue is how to discern which of the available methods can be adapted to the specific industrial context, taking into account current constraints and desired objectives. Accordingly, we assume that the company seeks solutions from a set of candidate research papers. Not all of them are relevant, and the company needs to identify the methods that are worth experimenting with. The general question of our study is whether the taxonomy is useful in helping to select a subset of promising RT methods, among which at least one would prove to be an appropriate solution to the industrial RT problem.

Our search for solutions considers a set of 52 candidate papers. We first took the 38 *RT* papers already mapped to the taxonomy. They are displayed in Table 1, where we retain the same ID as Ali et al. (2019) (from $S1$ to $S38$). These papers describe methods that were identified as industry-relevant by the authors of the taxonomy and their industrial partners.

Among the S_i papers, we identified techniques based on execution history, code coverage, requirement coverage, test case similarity and execution cost. But

Table 1: Selected papers on regression testing from Ali et al. (2019)

ID	Ref	ID	Ref
S1	Ekelund and Engström (2015)	S20	Rogstad et al. (2013)
S2	Saha et al. (2015)	S21	Krishnamoorthi and Mary (2009)
S3	Marijan et al. (2013)	S22	Tahvili et al. (2016)
S4	Marijan (2015)	S23	Janjua (2015)
S5	Buchgeher et al. (2013)	S24	Engström et al. (2010)
S6	Skoglund and Runeson (2005)	S25	Wikstrand et al. (2009)
S7	White et al. (2008)	S26	Engström et al. (2011)
S8	White and Robinson (2004)	S27	Vöst and Wagner (2016)
S9	Zheng et al. (2006b)	S28	Huang et al. (2009)
S10	Zheng et al. (2007)	S29	Srivastava and Thiagarajan (2002)
S11	Zheng (2005)	S30	Hirzel and Klaeren (2016)
S12	Zheng et al. (2006a)	S31	Pasala and Bhowmick (2005)
S13	Wang et al. (2013b)	S32	Herzig et al. (2015)
S14	Wang et al. (2017)	S33	Li and Boehm (2013)
S15	Wang et al. (2015)	S34	Anderson et al. (2014)
S16	Wang et al. (2016)	S35	Lochau et al. (2014)
S17	Wang et al. (2013a)	S36	Devaki et al. (2013)
S18	Wang et al. (2014)	S37	Carlson et al. (2011)
S19	Rogstad and Briand (2016)	S38	Gligoric et al. (2014)

Table 2: Added papers on ML-based methods

ID	Ref	ML	ID	Ref	ML
L1	Palma et al. (2018)	SL	L8	Shi et al. (2020)	RL
L2	Marijan et al. (2019)	SL	L9	Lima and Vergilio (2022)	RL
L3	Mahdieh et al. (2020)	SL	L10	Bertolino et al. (2020)	SL+RL
L4	Sharif et al. (2021)	SL	L11	Busjaeger and Xie (2016)	SL+NLP
L5	Yoo et al. (2009)	UL	L12	Lachmann et al. (2016)	SL+NLP
L6	Chen et al. (2011)	UL	L13	Medhat et al. (2020)	SL+NLP
L7	Spieker et al. (2017)	RL	L14	Kandil et al. (2016)	UL+NLP

none of these papers was published after 2016, and their set does not reflect the growing interest in machine learning (ML)-based methods. So we felt the need to include additional papers. We considered a recent literature review on ML-based *RT* research by Pan et al. (2022). The review classifies 29 papers depending on whether they use Supervised Learning (*SL*), Unsupervised Learning (*UL*), Reinforcement Learning (*RL*), Natural Language Processing (*NLP*), or a mix of them. We extracted a subset of 12 papers such that each learning technique is covered, as well as each mix of techniques. We also included two extra papers (Marijan et al., 2019; Sharif et al., 2021) not cited in the review, but which we had previously identified as ML-based work with an industrial focus. Table 2 lists the resulting set of added papers (from *L*1 to *L*14). We had to map them to the taxonomy, and did so with a systematic double check by different authors.

The real-world scenario is a major Renault project under development. We had full access to its data. We also interacted with the teams working on the project all along the two years of the study. In total, we engaged directly with approximately 20 engineers representing various roles, including test design, execution, and automation; debugging; build validation; test environment setup and maintenance; test bench design and configuration; testing platform management; and project data management.

The search for relevant methods followed four steps, shown in Figure 1. The first three steps leverage the taxonomy in accordance with the role of the context, effect and information described in Ali et al. (2019). Step 1 characterizes the industrial RT problem in terms of a context and desired effects. Step 2 identifies the feasible methods from the set of papers: the utilized information must correspond to information items available in the project. Step 3 keeps the feasible methods that address the desired effects, and determines how to measure these effects in the industrial context. The last and fourth step does not use the taxonomy but depends on the decisions taken at the previous steps. It consists in the empirical evaluation of the candidate RT methods that have been retained, using the metrics that have been retained.

Together, these four steps form a well-defined taxonomy-based approach to address RT problems. Application to Renault's real-life problem allows us to investigate whether the taxonomy fulfills the objectives stated by its authors, i.e., to help practitioners identify the RT methods best aligned with their needs. The general question about the practical utility of the taxonomy is refined into four research questions, one for each step of the approach:

– **RQ1**: *Does the taxonomy help in describing a specific industrial RT problem?*

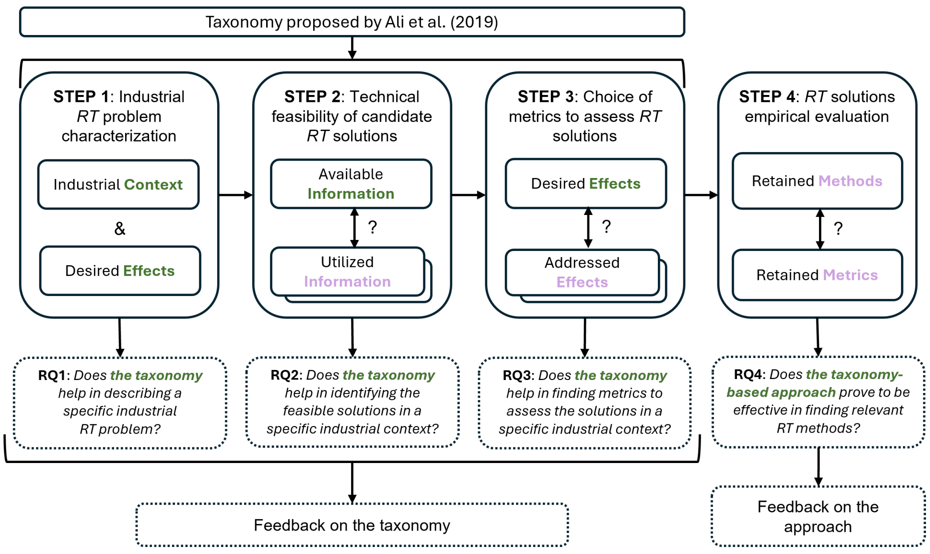

Taxonomy proposed by Ali et al. (2019)
STEP 1: Industrial RT problem characterization
Industrial Context
&
Desired Effects
STEP 2: Technical feasibility of candidate RT solutions
Available Information
?
Utilized Information
STEP 3: Choice of metrics to assess RT solutions
Desired Effects
?
Addressed Effects
STEP 4: RT solutions empirical evaluation
Retained Methods
?
Retained Metrics
RQ1: Does the taxonomy help in describing a specific industrial RT problem?
RQ2: Does the taxonomy help in identifying the feasible solutions in a specific industrial context?
RQ3: Does the taxonomy help in finding metrics to assess the solutions in a specific industrial context?
RQ4: Does the taxonomy-based approach prove to be effective in finding relevant RT methods?
Feedback on the taxonomy
Feedback on the approach

Fig. 1: Taxonomy-based methodology structure with related research questions

- **RQ2**: *Does the taxonomy help in identifying the feasible solutions in a specific industrial context?*
- **RQ3**: *Does the taxonomy help in finding metrics to assess the solutions in a specific industrial context?*
- **RQ4**: *Does the taxonomy-based approach prove to be effective in finding relevant RT methods?*

4 RT Problem Characterization

Step 1 of the approach characterizes the *RT* problem by an industrial context and a set of desired improvements. This section presents them. We then provide feedback about the use of the taxonomy for exposing the important characteristics of the problem.

4.1 Description of the Context

The case study involves an R&D automotive project called Project A in the rest of the paper. Figure 2 provides a high-level view of its architecture. It is multi-system and composed of several Electronic Control Units (ECUs) connected by a CAN bus (represented by purple lines in Figure 2). The two main components are the in-vehicle infotainment system (IVI) and the communication one (IVC). The infotainment constitutes the main point of contact with the driver and passenger thanks to a screen and connected equipment. This system is the vehicle interface for navigation, sound, image, communication, and user settings of the car. The other system is in charge of the communication between on-board systems and off-board services. A cloud platform is used to store vehicle information and to

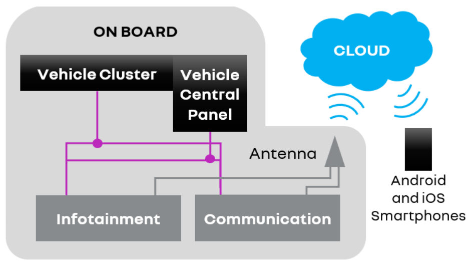


Fig. 2: Global view of Project A

Table 3: Context of Project A using taxonomy of Ali et al. (2019)

Context Factor		Project A
System-related	Size	Large-scale
	Complexity	Multi-language
	Type of system	Automotive embedded system
Process-related	Testing process	Weekly integration and issues allocation
	Test technique	Automated and manual
People-related	Cognitive factors	Not relevant
	Organizational factors	Multiple companies

communicate with other services. An example of an off-board service considered in Project A is a smartphone application for locking/unlocking the car's doors.

The case study focuses on the integration tests of the on-board software. Table 3 summarizes the context factors based on the taxonomy. Project A is an **automotive embedded system**, with a binary code **size** of around 6 GB. It comprises two **multi-language** systems. The infotainment is mainly in *C*, *C++*, and *Java*, while the communication is mainly in *C*. Regarding the **testing process**, a weekly integration is done and the issues are allocated according to their origin (due to the software under test, the test automation, or the environment). The **testing techniques** are: automated testing (executed during the night) and manual testing (executed during working hours). A relevant **organizational factor** is that the system development involves different companies. There are suppliers for infotainment, others for communication and Renault integrates both parts. Tests are carried out by Renault every week after the delivery of a new build.

The seven context factors of the taxonomy are high-level. The user is expected to put a short note about why a given factor is challenging (e.g., "multi-language" for the complexity factor in Table 3). We provide below a more extensive description of the context of Project A.

Tests are carried out on a test bench that reproduces the on-board architecture and manages access to external services. When a new version of the binary code is received from suppliers, it is flashed on the bench to update the *ECUs*. The integration team tests the new infotainment code using a stable version of the communication, and vice versa.

Both automated and manual tests have a priority ($P1$, $P2$, or $P3$) determined by the importance of the functional feature checked. When the test verdict is fail, the tester conducts an analysis to determine the origin of the problem: the software under test (*SUT*), the test automation, or a transient environment issue. Accordingly, the issue report is forwarded to product or test automation managers, who assign the issue a criticality (the impact it has on the customer) and a priority (to determine the order in which bugs should be fixed).

Test automation issues account for about 30% of the fail verdicts of automated tests. This is not an unusual situation, and reflects the complexity of the test environment for the embedded software. Other authors have studied automotive projects using hardware-in-the-loop test benches (Jordan et al., 2022). They reported from 74% up to 91% of failed executions due to automation issues (unreliable test infrastructures, incorrect tests). In Project A, the test automation calls for a dedicated development and evolution process. All automated tests are implemented using Robot Framework[1], following a keyword-driven approach. The main script of a test is concise, based on abstract keywords, but relies on many lower-level resources (keyword libraries, code libraries, configuration files). All in all, the automation code of a test typically represent tens of thousands lines of code ($>$ 10 KLOC), making it quite complex. A test is made available to the integration team only when it has demonstrated a stable correct behavior. If the testing of the SUT reveals an unstable behavior or any other automation issue, then the test becomes unavailable until it is repaired by the test automation team. To ensure a stable behavior, the test design puts emphasis on making each test a self-contained and standalone unit. Each test is responsible for its complete set-up and clean-up, including hardware reboots, the establishment of a connection to the cloud, and so on. In this way, a test starts and ends in a known state, and has no side effect on other tests.

Manual tests are also defined as standalone units. The tester is free to perform them in any order that matches the priority ($P1$ to $P3$, as previously explained). In practice, the tester may happen to merge two tests of the same priority that require a similar test bench configuration, in order to save the set-up time. However, this is not always good practice. A test may need a known initial state like a "fresh boot". If the test is appended to another one without the reboot, its behavior is less predictable. Spurious failures may occur. It is safer to systematically do the set-up specified for the test.

The set-up and cleaning actions are important to achieve stable tests, but are time-consuming. Due to them, the duration of each test – automated or manual – is typically in tens of minutes (from 20 to 50 minutes). One must also add the time to analyze the fail verdicts. On average, given the resources available to Project A, less than 25% of the tests can run each week. During a week, the practice is to have 2 nights and a half for running the automated tests, and about 2 days and a half for the manual ones. In the latter case, this is an estimate, because the execution time of manual tests cannot be precisely measured, nor clearly separated from the analysis time. When observing a fail verdict, the manual tester immediately performs the analysis before moving on to the next test. The number of executed tests thus depends on the fail verdicts of the week. The engineers estimate that, on average, 50% of the working time is spent on running the manual tests and 50% on analyzing them.

At the beginning of each week, the integration teams decides which tests will be executed. For manual tests, it is not uncommon that the low-priority ones end up not being executed, due to lack of time. Moreover, some tests tend to be rarely

[1] https://robotframework.org/

selected. As a result, some tests are not executed for months. It is, therefore, desirable to improve the regression testing process.

4.2 Desired Effects

The taxonomy proposes three broad categories of improvements: in test coverage, in efficiency and effectiveness, and in increasing awareness. For the engineers of Project A, the focus is on efficiency and effectiveness. For this, the taxonomy has several refined objectives, as shown in Table 4. We discuss below the ones that are the most desirable for Project A.

Following discussions with engineers, the two main objectives are (i) to decrease the time to find faults and (ii) to avoid tests that are never or rarely executed. The sooner the faults are found, the earlier the analysis of these bugs and the faster feedback to the system suppliers and test automation team. Moreover, tests should not be left behind for a long period of time. Otherwise, regression problems may remain unnoticed until late in the project. All in all, the test selection and prioritization could be more efficient than it is currently: it would ideally execute the failing tests first, at the very beginning of the week, and also ensure a circulation of the tests across weeks, none of them being forgotten. In terms of the taxonomy, the *decreased time for fault detection* clearly corresponds to the first objective, while the circulation of the tests is not explicitly mentioned.

Some supplementary objectives are desired but assigned less importance in the short term. For both manual and automated tests, a *reduced testing time* would make the bench available for other projects. Ideally, the usage time of the bench could go from two and a half days to just one day for manual testing, and from two and a half nights to one night for automated testing. However, the mere reduction in time would not be enough. It would be imperative to maintain the *fault detection capability.*

To sum up, the short-term objectives revolve around a more efficient use of the currently available testing time, while longer-term objectives would consider reducing this time. Note that the solutions might be different for manual and automated tests. For example, the decreased time for fault detection does not have the same meaning if the analysis is done on the fly (manual tests) or after the nightly execution (automated tests). For automated tests, a fine-grained ordering may not be needed, provided that the failing tests occur on the first night.

Table 4: Desired effects for Project A using the taxonomy

Test coverage	Feature coverage	
	Input (Pairwise)	
Efficiency effectiveness	Reduction of test suite	
	Reduced testing time	+
	Improved precision	
	Decreased time for fault detection	++
	Reduced need for ressources	
	Fault detection capability	+
	Severe fault detection	
	Reduced cost of failure	
Awareness	Transparency of testing decisions	

4.3 Feedback on the Taxonomy (RQ1)

The most helpful part of the taxonomy concerned the desired effects, which allowed the identification and prioritization of objectives. The context factors were deemed too high level to fully characterize an industrial problem. However, they were found relevant to guide the discussions with the engineers.

The review of desired effects revealed a missing one: the circulation of the tests. The taxonomy does not include it. Still, at least two papers analyzed by the authors of the taxonomy propose a method that ensures the circulation of the tests ($S26$, $S32$). One of the ML-based papers we analyzed ($L9$) also does. But none of them lets this appear as a *measured* effect. According to the methodology used to build the taxonomy, the circulation of the tests would then not qualify as a desired effect because *"an effect has to be demonstrated as a measurement"*. We could find another paper in the literature (Erik Strandberg et al., 2017) that mentions the circulation of tests, and that explicitly demonstrates this effect (along with other ones). Forgotten tests are a recurring problem in many industrial settings, thus we propose adding *forgotten test avoidance* to the taxonomy. Section 6 will further discuss the evaluation of this effect.

RQ1 (aid in describing the problem): The context part is too high-level but still relevant for guiding discussions with engineers. The effect part proved helpful in capturing the industrial objectives. A missing effect is the circulation of the tests.

5 Analysis of RT Methods Feasibility

Step 2 of the approach explores the feasibility of the methods proposed in the set of 52 papers. A method is considered as technically feasible if it is based solely on information items that are available in the industrial context. First, a taxonomy-based analysis is done, which uses the items mentioned in the taxonomy. Both the papers and Project A are mapped to the taxonomy, allowing a comparison of the required and available items. It yields a classification into feasible and infeasible methods. Then, we assess the outcomes of the previous analysis by browsing through the papers, and by manually determining whether each method is really (in)feasible in the context. The section ends with feedback on the use of the taxonomy to analyze feasibility.

5.1 Taxonomy-Based Analysis of Feasibility

Table 5 shows the mapping of Project A and papers to the taxonomy. The first two columns display the information items of the taxonomy. The third one indicates which items are available for Project A. It can be compared with the fourth column which lists the papers requiring an item. For example, looking at the first item: the changes in requirements are not tracked in Project A, so that the method described in $S21$ is not feasible. For Project A, a star is marked (X*) if the information is indirectly available: it may be derived from other available data, or replaced by an estimate.

Table 5: Information mapping for Project A and the set of 52 papers

	Information	Project A	State of the Art Papers
Requirements	No of changes in a requirement		S21
	Fault impact	X	S21
	Subjective implementation complexity	X	S21
	Perceived completeness	X	S21
	Perceived treaceability	X	S21
	Customer assigned priority	X	S21, S33, L14
Design artifacts	System models		S13-S18, S27, S35
	Code dependencies		S19, S20, S37, L10
Source code	Code change / Revision history		S1, S2, S5, S7, S8, S24, S25, S38, L1-L3, L10, L13
	Source file		S2, S7, S8, S30, S37, L11
	No of Contributors		S32
Intermediate code	Class dependencies		S6
	Code changes (method or class)		S2, S6, S28, L1, L10, L11
Binary code	Revision history	X	S6, S29
	Component changes		S9-S12, S31
	Binary files	X	S6, S9-S12, S23, S29
Test cases	Target variant		S26
	Type of test		S26, L14
	Model coverage		S13-S20
	Functionality coverage		S3, S4
	Static priority	X	S26
	Age	X	S26, L11
	Fault detection probability (estimated)	X*	S22, S29, S33, L13
	Execution time (estimated)	X	S22, S29, L4, L7, L8, L10, L12, L13
	Cost (estimated)	X*	S22, S33, L12
	Link to requirements	X	S21, S22
	Link to faults	X	S4, S21
	Link to source code		S6-S8, L2
Test executions	Execution time	X*	S29, S32
	Database-states		S36
	Invocation counts		S28
	Invocation chains		S28, S31
	Runtime component coverage		S31
	Method coverage		S28, L1, L6
	Code coverage		S5, S23, S29, S37, S38, L2, L3, L5, L11, L13
	Browser states		S36
	Class coverage		S6, L10
Test reports	Execution time	X*	S3, S4, S13-S18, S28
	Verdicts	X	S1-S4, S13-S18, S26, S32, S34, L1, L2, L4, L6-L11
	Severity	X	S28, S33
	Link to packages and their revisions		S1
	Link to branch		S32
	Build type		S32
	Link to failure		S13-S18
	Test session		S13-S18, S26
	Variant under test		S32
Issues	Link to fixed file / link to source code		S24, S25, L2, L3
	Fix-time	X*	S32
	Link to test case	X	S24, S25, S37, L3, L12- L14
	Failure severity	X	S3, S4, L12, L14

For papers Si, the table simply reproduces the mapping done by the taxonomy authors. For Project A and the papers Li, the mapping is ours. Obviously, it depends on our understanding of the taxonomy. This is worth mentioning since we experienced difficulties in interpreting some of the items. We report on these difficulties below.

1) Test execution time appearing at different places. Ali et al. (2019) provided the following explanation: as an attribute of a *test case* it is an estimation; as an attribute of a *test execution*, it is measured at runtime; and as an attribute of the *test reports*, it is further recorded and maintained. However, it was unclear to us why and how to distinguish these cases. We had a look at papers exemplifying them. For instance, the mapping of $S29$ has the execution time for both test cases and test executions. But the paper merely mentions an option to take time into consideration. There is no detail on time estimation, measurement, or recording. As another example, we could not understand why time in $S32$ (test executions) does not have the same mapping as in $S3$ and $S15$ (test reports).

We finally took the following mapping decisions. For Project A, we considered all three execution time items as available. Strictly speaking, the measured execution time (in test executions or reports) is available for automated tests only. We marked the information as indirectly available due to manual tests, for which we only have an estimate. For the mapping of papers Li, we decided not to delve into considerations of estimated, measured, or recorded values. If a paper used the duration of tests, we mapped it to *test case/execution time* and considered the information as available for Project A.

2) Items referring to faults, failures or issues. We feel that the terminology would have deserved an explanation. Is a failure the same as a fail verdict? If so, there is no added value of *test reports/link to failures* compared to *test reports/verdicts*. For instance, we could not explain the different mappings of papers $S4$ (verdict only) and $S18$ (verdict + link to failures): both compute failure rates in a test time window. Paper $S4$ is additionally mapped to *test cases/link to faults*, but seems more concerned with failures (their frequency, their severity) than faults (i.e., the investigated causes of failures). We observed that, in many papers, "faults" and "failures" are used interchangeably. In Project A, we have data for both. The investigation of faults is tracked by an issue management system. Not all failures yield the opening of an issue, and there may be a single issue opened for multiple failing tests.

We did the mapping as follows. We ignored the item *test reports/link to failures*. Rather, we used *test reports/verdicts* to indicate that a method needs the recording of the fail verdicts. Some methods do not consider the raw failure information, requiring a traceability between the tests and the revealed faults. For them, we decided not to distinguish whether the recording is in the direction from tests to faults (*test cases/link to faults*), or from faults to tests (*issues/link to test case*). Indeed, one item can be derived from the other by reversing the links. For papers Li, we chose *issues/link to test case* to represent any relation between tests and faults. This link direction is the most frequent one when using an issue management system. For Project A, we explicitly marked both directions as available. We also marked *test cases/fault detection probability* as indirectly available: estimates can be derived from historical data.

3) Test session (in test reports). It seems obvious that any RT process involves test sessions. However, only a few papers Si are mapped to this item, and we could not understand what makes them different. We decided to ignore this item.

Subject to our interpretation, the taxonomy-based analysis retains 8 out of the 52 papers. They are shown in the first row of Table 6. They use items like the execution time of tests, their cost, their historical effectiveness (fail verdicts, severity of the failures, issues found by the tests), the tested requirements, and their priority. All these items are available in Project A. Most of the eliminated papers require design or code artifacts that Project A does not have, as well as coverage information. A few papers are eliminated due to other causes: they consider variability data in multiple variants and code branches ($S26$, $S32$), or need to monitor the number of changes in requirements ($S21$).

5.2 Manual Feasibility Analysis by Reading the Papers

To assess the aid provided by the taxonomy, we need some ground truth. For this, we performed a manual labeling. We went through all the papers and labeled them as feasible or not in the context of Project A. Two authors systematically labeled each paper. First, one author read and mapped all the papers into the taxonomy. Then, a second author mapped the papers and compared their classification with that of the first author. Finally, the authors discussed any discrepancies and compared the three classifications to determine the final mapping.

As shown in Table 6, four papers were misclassified by the taxonomy. There are two false positives ($S33$, $L12$) and two false negatives ($S3$, $S26$). The classification errors are due to two causes: some papers are not correctly mapped to the taxonomy, and the taxonomy misses important items. Differences are highlighted in bold in Table 6.

Table 6: Technically feasible methods according to taxonomy and manual labeling

Labeling method	Selected papers
Taxonomy-based labeling	S22, **S33**, S34, L4, L7, L8, L9, **L12**
Manual labeling (ground truth)	**S3**, S22, **S26**, S34, L4, L7, L8, L9

While reading the papers, we had several disagreements with the mapping proposed for the papers Si by Ali et al. (2019). The disagreements are reported in Table 7. They concern as much as 50% of papers Si. Most of the time, the disagreement did not affect the final feasibility label. But it did in three cases: $S3$, $S26$, and $S33$. Paper $S3$ was considered as the same approach as $S4$. Both were eliminated due to the use of functionality coverage information. But only $S4$, which extends $S3$, uses this information: $S3$ is actually feasible in the context of Project A. Paper $S26$ was also wrongly eliminated. It is supposed to use *test cases/target variant.* While variability is mentioned in the paper, the authors explain that they could not retrieve the information and present a method that does not use it. The type of tests is also not used. For $S33$, the error is in the opposite direction. The paper was retained but uses data items that were forgotten in the mapping (functionality coverage and configuration variants), making it unfeasible.

Table 7: Disagreements with the mapping of papers by Ali et al. (2019)

ID	Disagreements
S3	Same mapping as S4 (same authors), but actually uses less information
S9-S12	Needs the source code of user functions and the links between tests and code
S13-S18	Focuses on product lines, but no variability-related item is marked. All papers have the same mapping while they consider different data subsets
S19-S20	Wrong mapping to design artifacts/code dependencies (uses a black-box model)
S26	Wrong mapping to test cases/target variant (does not use variability data)
S27	Needs the model coverage of test cases
S29	Wrong mapping to test cases/fault detection probability (the paper merely says that fault detection could be added)
S32	Needs the cost of test cases
S33	Needs the functionality coverage and target variant of test cases
S35	Needs the model coverage and target variant of test cases

The other cause of misclassification is when the taxonomy misses an item that is important for feasibility. Since the taxonomy was built from a literature review, it reflects the set of reviewed papers. As mentioned, this set had few methods based on machine learning. By adding 14 papers to represent this category of methods, we could get new information items. Such was the case for $L12$. It applies NLP techniques and requires a natural language description of test cases. There is no item for this in the taxonomy. The paper was labeled as feasible, but Project A lacks the required information. A similar missing element is the natural language description of issues, required by $L14$. Project A has this information, so the missing element did not cause a misclassification of $L14$. By the way, $L14$ was found unfeasible due to another reason (type of test). In other contexts where all information items required by $L14$ would be available, with the exception of a natural language description of issues, a misclassification would occur.

5.3 Feedback on the Taxonomy (RQ2)

The availability of information was found an effective criterion for determining which methods are applicable in a given context. For Project A, only 8 out of the 52 papers fulfill this criterion (ground truth). Having read the papers, we are quite confident that the methods are technically feasible in our context. However, their identification via the taxonomy was hindered by understanding issues and paper classification errors.

We had a hard time trying to understand the meaning of some information items. What is the difference between execution time for test executions and for reports? What do the many links between tests and failures (or faults, issues) mean? What is a test session? We ended up reading numerous papers just to determine how to interpret the taxonomy. We missed a user guide explaining the rationale and meaning of the items.

While browsing through the papers already aligned to the taxonomy, we had disagreements with as much as 50% of the mappings. They concern grouping papers from the same authors while there are differences in the approaches, forgetting to mark items, or marking items that the authors mention but actually do not use. For Project A, the inaccurate mappings caused three classification errors ($S3$, $S26$

misclassified as unfeasible, and $S33$ misclassified as feasible). The inaccuracies reported in Table 7 could be fixed, but there is no maintained repository to update the mappings for future usage. Other classification errors were due to missing elements in the taxonomy. They concern natural language artifacts used by some machine learning papers. The incompleteness of the taxonomy is unavoidable, as new approaches are continuously added by researchers. Keeping the taxonomy in line with the state of the art would again require a maintenance effort.

RQ2 (aid in selecting the feasible methods): The information items are helpful to classify the papers, but difficult to interpret. In addition, several papers are inaccurately mapped to the taxonomy. A documentation and maintenance effort would be needed to make the taxonomy more usable.

6 Metrics to Assess RT Solutions

Step 3 of the approach focuses on the choice of metrics to assess the methods retained after Step 2. Table 8 shows the alignment between the desired effects (from Step 1) and those addressed by the feasible methods. The most desired effects are in bold. The circulation of the tests is not in the taxonomy but added for completeness.

Table 8: Addressed Effects

Addressed effects	Methods
Reduced testing time	S3, L4, L7, L8, L9
Decreased time for fault detection	S3, S22, S26, L4,L7, L8, L9
Fault detection capability	S26, S34
Circulation of the tests	S26, L9

All the methods have at least one effect in common with Project A, and are thus potentially relevant. But, obviously, their relevance cannot be decided based on published results. They must be experimented in the industrial context. To prepare for the implementation of experiments, we must gain closer insights into the measurement of effects. For this purpose, we use the taxonomy mapping to identify the papers (feasible or not) willing to achieve the same desired effect as ours. Next, we examine their evaluation strategies to see if they are aligned with our context. We then decide the metrics associated with each desired effects. We also calculate the margin for improvement for each effect, by evaluating the industrial practice as a baseline.

In practice, the decisions on the measurements of effects had to accommodate industrial constraints. We first report on these constraints before presenting the choice of metrics.

6.1 Constraints on the evaluation of methods

The evaluation had to be done without interfering with the testing process of Project A. An online evaluation of the methods was therefore ruled out. The evaluation had to be offline, relying on historical data collected during the project.

Table 9: Datasets information

Dataset	Versions	Test Cases	Passed	Failed	Failed with Issues
Manu IVI	93	173	94%	6%	3%
Manu IVC	69	210	81%	19%	14%
Auto IVI	70	96	79%	21%	6%
Auto IVC	27	88	79%	20%	8%

We extracted four datasets of test executions from the project database, shown in Table 9. They are referred to as *Manu IVI*, *Manu IVC*, *Auto IVI*, and *Auto IVC*. They differ according to the type of execution (manual or automated), and whether they target the infotainment (IVI) or communication (IVC) systems.

These datasets are incomplete, in the sense that we do not have the outcome of the tests that were not run during a given validation cycle (i.e. during a given week, when a new SUT version is received). The lack of information concerns both quarantined tests and tests that were available but not selected for execution.

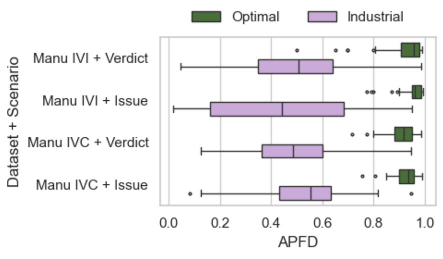
Data is also noisy. The measurement of execution time is imprecise for manual tests. For automated tests, the recorded value does not include the set-up time. The manual reporting of issues is imperfect. As can be seen in Table 9, many failed verdicts are not linked to issues. For example, for Manu IVI, there are 6% of test executions that fail, among which only half (3%) are linked with issues. The other failures may be due to an external cause in the environment of the test (e.g., a transient WiFi connection loss, a cloud service that is temporarily unavailable), in which case no issue needs to be linked. But there is an unknown proportion of cases in which the test operator should have opened an issue, or (perhaps, most often) should have introduced a link to an already existing issue. The reporting is worse for automated tests than for manual ones.

It was not possible to complete or repair the data. Indeed, this would be prohibitively expensive. It would require extensive access to the test bench to run the missing tests, as well as human resources to run the manual tests and analyze all the failed executions.

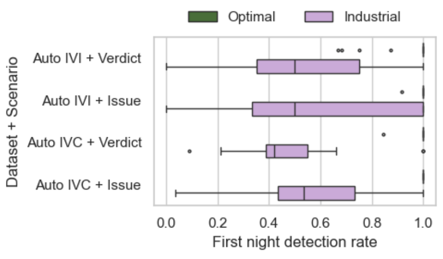
Having to cope with incomplete and noisy data, we had to be very careful about what could or could not be measured. We now explain the decisions taken for each desired effect.

6.2 Decreased Time for Fault Detection

The *decreased time for fault detection* is measured by 19 papers. Among the used metrics, one stands out and appears in 12 papers: *Average Percentage of Faults Detected* (APFD) and its variant *Normalized APFD* (NAPFD). *APFD* was first introduced by Rothermel et al. (1999) to measure how early an ordered test suite reveals the faults. It requires that the faults are known, the tests that reveal each fault are known, and the full suite is executed. If the faults and their revealing tests are unknown, the metric may use the raw information of the failing tests. The variant *NAPFD* (Qu et al., 2007) addresses cases where not all tests are run and thus only a certain percentage of faults (or failures) are detected compared to the full suite. However, it still requires that all faults (or failures) are known.



(a) *Manual execution*



(b) *Automated execution*

Fig. 3: Evaluation of optimal and industrial orders of each dataset

We only have partial information about the test executions, so we have to concentrate our evaluation on known outcomes. We consider that the full test suite for a given cycle (i.e., a week) is the set of tests that ran during that cycle. The methods under evaluation have to prioritize them, and we assess whether the resulting orders enable faster detection than the historical execution order. APFD is a suitable metric for comparing orders. We use it for manual tests. For automated tests, we do not need such a fine measure of orders. As the results are analyzed during working hours, the precise order of tests over the course of a night is not important. We simply retain the percentage of detections occurring on the first night. As the execution time of tests is imprecise, we take the approximation that one night corresponds to 40% of the tests (total duration is about two nights and a half).

Both metrics – APFD or First Night Detection Rate – can treat detection in terms of faults revealed or fail verdicts observed. Neither of these options is ideal for us. Dealing with faults means relying on the reporting of issues which, as we explained earlier, is highly noisy in our datasets. Dealing with fail verdicts does not take into account redundant detections (several tests fail due to the same issue) and also gives false positives (failed executions due to environmental events rather than to issues in the SUT or the tests). Since neither of these evaluation options is fully reliable, we decided to implement both, in terms of faults and failures. In the first case, we refer to the evaluation as being performed in an *Issue* scenario. In the second one, the scenario is called *Verdict*. We require candidate methods to be sufficiently robust to adapt to both scenarios.

Once the metrics have been chosen, it is worth determining whether the current industrial practice can be improved. We compared two baselines. The first, called *Optimal*, mimics a perfect prioritization method in which the detecting tests are put first. The second, called *Industrial*, reflects the actual order in which the tests were executed. Figure 3 shows the evaluation of these two baselines. The box plots visualize the spread of APFD or First Night Detection Rate across all validation cycles. As can be seen, the industrial order has a high variability in performance, and most measured values are far away from the optimum. This clearly confirms that there is room for improvement for the four datasets.

6.3 Circulation of the Tests

As regards the *circulation of the tests*, the taxonomy is of no help. Moreover, no paper from the set measures this effect, although some do address it ($S26$, $S32$, and $L9$). Outside the set of papers, we identified a relevant work (Erik Strandberg et al., 2017) in which the authors check that no test is forgotten in two months of execution. The evaluation of their RT method was done online, in a process where the software changes every day and regression tests are carried out every night. In $S32$, changes occur at a faster rate, and the RT method enforces each test to execute at least every third day.

The previous approaches suggest measuring the interval of time between two executions of a test, and checking whether it exceeds a given threshold. A global metric could then be the percentage of forgotten tests identified in this way.

We kept the idea of measuring the percentage of forgotten tests, but had to adapt it to Project A. First, a circulation threshold of only two months (not mentioning three days) would be too demanding. In Project A, a validation cycle lasts one week and has resources for less than 25% of the tests. Running all tests just once would then take more than one month (> 4 cycles). Second, a test can be quarantined for repair or even become obsolete. A long interval of time without any execution may reflect the unavailability of the test rather than a poor circulation. The determination of forgotten tests must, in some way, take into account how often each test was available during the period of observation.

The solution we adopted mixes a fixed threshold and a variable time tolerance to account for availability. We consider a fixed threshold N_{thres} of 16 cycles (approximately 4 months) below which a test does not need further examination. After this time, we count the number of cycles $n_a(tc)$ during which a test tc was available but still not run. If $n_a(tc)$ is greater than N_{all}, the number of cycles to run all tests, we report tc as forgotten. This means that, after N_{thres}, the test had enough opportunities to be executed but still was not, indicating that the test has likely been overlooked or neglected in the test scheduling process. The detection rule is formalized below, where $gap(tc)$ represents a number of cycles without execution of a test tc:

$$(gap(tc) > N_{thres}) \ \ and \ \ (n_a(tc) > N_{all}) \ \ \rightarrow \ \ forgotten(tc) = True \qquad (1)$$

If a test is always available, the detection occurs when the gap reaches $N_{thres} + N_{all}$ cycles. The tolerance is higher and variable if the test is unavailable in the cycles after N_{thres}. We checked whether, using this definition of forgotten tests, a target of 0% would be achievable for Project A. For this, we simulated an optimal method that selects a subset of available tests in descending order of last execution date. The method represents the best that can be done to ensure test circulation, given the allowed test size and the unavailability of some tests at the various cycles. As shown in Table 10, the evaluation of the optimal method does give 0% forgotten tests. We also evaluated the historical circulation of tests in the industrial datasets. From the results reported in Table 10, there is room for improvement: we observed $12 - 17\%$ of forgotten tests.

To measure the improvement supplied by candidate RT methods, we had to determine how to accommodate the constraint of an *offline* assessment with *incomplete* data. Indeed, the only way to improve the test circulation is to allow

Table 10: Percentage of forgotten tests measured according to Rule 1

Dataset	Optimal	Industrial
Manu IVI	0%	12%
Manu IVC	0%	16%
Auto IVI	0%	17%
Auto IVC	0%	14%

the selection of forgotten tests, for which we do not have execution results. We therefore considered augmenting the data with artificial results, which raised the problem of the values to choose. Although the evaluation focuses on a circulation objective rather than a detection one, the methods use past results as an input to select tests, so that the chosen values may indirectly affect the percentage of forgotten tests.

To address this problem, we took inspiration from Paper $S24$ (Engström et al., 2010). The authors considered various assumptions about the unknown results and performed a sensitivity analysis. The measured effect was not test circulation, but the same principle can be reused for it. As in $S24$, we considered both extreme and average cases: 1) all the unknown verdicts are pass; 2) they are all fail; 3) they are fail with a probability equal to the historical failure rate at this cycle. The evaluation then measures the percentage of forgotten tests in these three cases, and observes whether the methods perform consistently well in all of them. In this way, the test circulation can be judged as an inherent property of a method, rather than an artifact of the data augmentation.

6.4 Long-Term Objectives

The long-term objectives in Project A are to reduce the testing time while maintaining the fault detection capability. The testing time per week would ideally go down to just one night for automated tests and one working day for manual tests. Since we do not have a precise measurement of testing time, we approximate the objective by one in terms of test size reduction: the number of tests should be 40% of the historical number of tests run at a cycle.

The objective of *reducing testing time* is mentioned in numerous papers: $S5-S18$, $S23-S25$, $S27$, $S28$, $S30-S32$, $S35-S37$, $L2$, $L4$, $L8$ and $L9$. In most of the cases there is a comparison between the execution time with and without the selection, either in absolute numbers or in percentages. For Project A, we do not need to measure this effect, as the desired time reduction is fixed. The key is rather the measurement of the fault revealing power in the reduced time.

Fault detection capability is measured in $S7$, $S8$, $S13-S21$, $S24-S26$, $S28$, $S29$, $S34$, $L2$, $L6$, $L11$ and $L14$. The metrics used in these papers can be divided into four types: the absolute number of detected faults, the detection rate per test case, the percentage of detected faults – which is the recall metric – and the combination of recall and precision (i.e. F-Measure). We decided to use the recall.

Given our incomplete datasets, the evaluation must consider that the tests available at a given cycle are the ones for which we have outcomes, i.e., which were historically run at that cycle. Candidate methods are then judged on their

Table 11: Average detection rate at 40% of the original test size

Scenario	Dataset	Optimal	Industrial
Verdict	Manu IVI	96%	41%
	Manu IVC	98%	44%
	Auto IVI	98%	54%
	Auto IVC	99%	48%
Issue	Manu IVI	100%	35%
	Manu IVC	100%	45%
	Auto IVI	100%	56%
	Auto IVC	100%	61%

ability to select a subset of these tests, which will keep a high percentage of all historical detections.

Table 11 displays the results supplied by two baselines: the optimal method where all failed tests (or the ones linked to issues) are run first, and the industrial baseline corresponding to the historical execution order. The results of Optimal show that keeping 100% of detections is infeasible in the Verdict scenario. For a few cycles, the reduced test size is not sufficient to run all failed tests. Indeed, in these cycles, some major regression issues caused numerous failures. We also took a closer look at the other cycles to determine at which size it is feasible to reach 100% of detections in the Verdict scenario. In all datasets, a number of cycles have around 30% of failed tests. This shows just how tight the target size of 40% is. Missed detections will occur if only a small proportion of passing tests are mistakenly assigned a high priority. The results from Industrial (see Table 11) are clearly perfectible, but it seems unlikely that any of the candidate RT methods will approach 100% detections with only 40% of the tests. We therefore decided to evaluate the methods over a range of sizes from 40% to 90%.

It might appear that the metric at 40% is the same as the first night detection rate, which we previously introduced for the objective of *Decreased Time for Fault Detection*. Actually, their measurement processes differ. Previously, we wanted to measure the effect of *prioritization*, i.e., all tests are run and we would like the detecting ones to run first. Here, we are discussing the effect of a *selection* method, that first prioritizes the tests and then runs *only* the highest-priority ones. Since a subset of tests is run, some historical outcomes become unknown to the method. Its prioritization decisions are taken with less information. If a method is not robust to information loss, it can be very effective when used for prioritization, yet very poor when used for selection. To evaluate the impact of information loss, we decided to compare the percentage of detections at a size of 40% in the two cases: without information loss (i.e., method used for prioritization only, as measured for the previous objective), and with information loss (prioritization + selection of the top 40% tests, as measured for this objective).

6.5 Feedback on the Taxonomy (RQ3)

The taxonomy relevantly pointed to papers that measure the desired effects (with the exception of test circulation). We went through the papers to gain an overview of alternative metrics. We selected the ones we wanted to use and evaluated the margins for improvement in the industrial context.

In the taxonomy, the effect part is much less detailed than the information part, which contains a fine-grained list of items utilized by the methods. An additional level of detail would be useful for the effect part too. In particular, the effects could be explicitly refined into metrics. As an example, the live repository by Greca et al. (2023) does provide a mapping between papers and metrics, for a set of papers published after 2016. Having this information in the taxonomy would have saved us time in the identification of metrics, allowing us to concentrate on the design of experiments.

The most important design decisions concerned how to deal with the industrial constraints imposed on the evaluation process. A recap of these decisions is provided at the beginning of the section presenting the experimental results (see Table 13 in Section 8). To make decisions easier, it would have been relevant to know which papers also had to evaluate their method *offline* with *incomplete* test execution data. Unfortunately, the taxonomy does not provide elements to categorize the evaluation process used in each paper. We had to read all the papers to identify that $S24$ faced constraints similar to our own. The authors proposed an analysis of sensitivity to the unknown, which we reused when we augmented data to evaluate test circulation.

RQ3 (aid in identifying metric to assess the solutions): The taxonomy was only helpful in pointing to papers that measure the effects of interest. We lacked a list of metrics associated with the effects, and items to characterize the experimental settings in which the papers measured these metrics.

7 Description of RT Methods

This section presents the methods selected for evaluation, at the end of Steps 1-3 of the approach. We retained four S_i papers (from the papers studied by the authors of the taxonomy) and four L_i papers (added by us to represent ML-based methods): $S3$, $S22$, $S26$, $S34$, $L4$, $L7$, $L8$, and $L9$. For each paper, we considered variants of the proposed solution. Table 12 provides an overview of the 40 resulting candidate methods.

Some of the variants come from the papers. The authors of $S26$ and the ones of $S34$ studied alternative methods which we refer to as $S26/34.1$ and $S26/34.2$. Additionally, the authors of $S34$ had variants to limit the number of historical data: Win means a limited history (within a time window) and $Full$ is with the full history. The variants of methods $L7$, $L8$ and $L9$ correspond to the different configurations explored in the respective articles. They differ in the type of agents or policies employed and in the way the algorithms are rewarded.

We created the other variants when we adapted the methods to our constraints. Firstly, in our datasets, only a subset of available tests is executed at each cycle. For some methods that require the last k outcomes of a test ($S3$ and $L4$), we considered two cases: taking the outcomes of the last k executions of this test ($Case1$), or taking the outcomes of the last k cycles and explicitly accounting for unknown values ($Case2$). Secondly, our datasets lack precision in the duration of test executions. So, some methods that require execution time ($S3$ and $S22$) are also evaluated without this feature, resulting in the $withTime$ and $withoutTime$ variants. Finally, the original $S26$ method uses a full history, and we introduced

Table 12: List of methods and their variants

ID	Method	Variants (different configurations of the same method)		
		ID	Name	Description
S3	ROCKET	S3.1.1	Case1_withTime	Case1 = last executions
		S3.1.2	Case1_withoutTime	Case2 = last cycles
		S3.2.1	Case2_withTime	withTime = execution time
		S3.2.2	Case2_withoutTime	withoutTime = no time
S22	TOPSIS	S22.1	TOPSIS_withTime	withTime = execution time
		S22.2	TOPSIS_withoutTime	withoutTime = no time
S26	Faz/ExtFaz	S26.1.1	Faz_Full	Faz = Fazlalizadeh et al.
		S26.1.2	Faz_Win	ExtFaz = Engström et al.
		S26.2.1	ExtFaz_Full	Full = all past data
		S26.2.2	ExtFaz_Win	Win = recent past data
S34	MFF/ARM	S34.1.1	MFF_Full	Most Frequent Failed
		S34.1.2	MFF_Win	Association Rule Mining
		S34.2.1	ARM_Full	Full = all past data
		S34.2.2	ARM_Win	Win = recent past data
L4	DeepOrder	L4.1	Case1	Deep learning method
		L4.2	Case2	Case1/2 = last exec/cycles
L7	RETECS	L7.1.1	Tb + FCount	Reinforcement learning
		L7.1.2	Tb + TCFail	Agent + Reward
		L7.1.3	Tb + TRR	Agents:
		L7.2.1	NN + FCount	Tb = tableau
		L7.2.2	NN + TCFail	NN = neural network
		L7.2.3	NN + TRR	
L8	RL	L8.1.1	Tb + RHE_whole_all	Reinforcement learning
		L8.1.2	Tb + RHE_whole_four	Agent + Reward
		L8.1.3	Tb + RHE_part_all	Agents:
		L8.1.4	Tb + RHE_part_four	Tb = tableau
		L8.1.5	Tb + THE_whole_four	NN = neural network
		L8.1.6	Tb + THE_part_four	Weight functions:
		L8.2.1	NN + RHE_whole_all	RHE = *relu* | THE = *tanh*
		L8.2.2	NN + RHE_whole_four	Rewards:
		L8.2.3	NN + RHE_part_all	whole = for all tests
		L8.2.4	NN + RHE_part_four	part = for failed tests
		L8.2.5	NN + THE_whole_four	all = using full data
		L8.2.6	NN + THE_part_four	four = using 4 last cycles
L9	COLEMAN	L9.1.1	ε-Greedy + RNFail	Multi-Armed Bandit method
		L9.1.2	ε-Greedy + TRR	Policy + Reward
		L9.2.1	UCB + RNFail	Policies:
		L9.2.2	UCB + TRR	ε-Greedy: Epsilon-Greedy
		L9.3.1	FRRMAB + RNFail	UCB: Upper Confidence Bound
		L9.3.2	FRRMAB + TRR	FRR: Fitness-Rate-Rank

a Win variant inspired by the one of the authors of $S34$ (history within a time window). This variant limits the processing time and memory.

Implementing the methods required an effort because only 3 of the 8 articles have an online repository. In addition, we had to fix a bug in one case and eventually chose to redevelop the code in another. We also had to adapt the data preparation process to our datasets. The code for all variants, along with detailed descriptions, are made available for reuse in a GitHub repository[2]. A shorter description is given below. As there are 8 papers to introduce, the overall content is still long. Readers more interested in the evaluation results can jump directly to Section 8 and return to some of the methods later.

[2] https://github.com/laurabrzmeyer/papers-implementation

S3 and variants. The *ROCKET* method (**S3**) is implemented in an industrial case study by Marijan et al. (2013). The inputs required in this approach are: a set of test cases to be prioritized, the test verdicts (i.e., *pass* or *fail*) from previous executions, the test execution time, and the budget time allotted for testing. The priority is calculated based on the sum of past verdicts weighted according to how recent the execution was (most recent, second most recent and other). A failed execution increases the priority of a test, a passed execution decreases it. In addition, test execution time is taken into account to break the tie between tests with the same priority. Tests that take too long to run are penalized over faster ones.

S3.1.1 implements this method. Variants **S3.2.*** calculate the priority with higher weights on the two most recent *cycles* rather than the two most recent *executions*. Unknown verdicts are given the neutral value zero (vs. -1 and 1 in the known cases). Variants **S3.*.2** use a random tie breaker rather than one based on test execution time.

S22 and variants. The Technique for Order Preference by Similarity to Ideal Solution (*TOPSIS*) was first introduced by Hwang and Yoon (1981). It is a method for decision support when multiple conflicting criteria are present. TOPSIS searches for the alternative with the smallest geometric distance from the positive ideal solution (*PIS*) and the largest geometric distance from the negative ideal solution (*NIS*). The alternatives are thus ranked by their degree of similarity to *PIS* and dissimilarity to *NIS*. A combination of *TOPSIS* and fuzzy principles is done by Tahvili et al. (2016), resulting in the *FTOPSIS* method (**S22**). FTOPSIS is intended for cases where the input data are qualitative rather than quantitative, which justifies the fuzzy evaluation of criteria. The criteria are: fault detection probability, time efficiency, cost, and requirement coverage. In our case, it is possible to quantify each criterion directly. So we implemented *TOPSIS* rather than *FTOPSIS*. **S22.1** gives equal weights to all four criteria, while **S22.2** is a timeless variant giving a weight of zero to time efficiency.

S26 and variants. Engström et al. (2011) investigate a method developed by Fazlalizadeh et al. (2009), calling it *Faz* (**S26.1**). This method uses three features to prioritize a test: the test's frequency of failure, the number of times the test remained unexecuted, and the test's previous priority. Each element is assigned a weight. Engström et al. (2011) also extend *Faz*, creating the *ExtFaz* method (**S26.2**). Two features are added: the test's age (relative to the current cycle) and the test's static priority.

S34 and variants. A method based on the Most Frequent Failures is introduced by Anderson et al. (2014), which we refer to as *MFF* (**S34.1**). In MFF, every test case that has a failure rate greater than a threshold is tagged as a test that will potentially fail again. The authors also present another method, which we denote as *ARM*, i.e. Association Rule Mining (**S34.2**). This method runs a set of initial tests (called "smoke" tests by the authors), some of them being observed to fail. Then, ARM selects additional regression tests that are strongly associated with those, in the sense that whenever the smoke test failed in the past, then the other tests often failed as well. The selection is again threshold-based. Both MFF and ARM are implemented in two variants depending on the length of the history, which we note *Win* and *Full*.

We implemented the MFF and ARM variants without using thresholds. Hence, we turned these selection methods into prioritization ones, tests being ranked according to their failure rate (for *MFF*) or association level (for *ARM*). This facilitated the comparison with other candidate methods.

L4 and variants. Sharif et al. (2021) train a deep learning model to predict the priority of each test case. The code for this method is open source[3]. The following features are used in the prediction: the three last execution status of a test; the test execution duration; the absolute time when a test was last run; the distance between the two most recent executions of a test; and the number of times a test status has changed. During the training, the label is the $PriorityValue$ previously calculated using the $ROCKET$ method ($S3$). We used its two variants *Case1* and *Case2* with time.

L7 and variants. Spieker et al. (2017) introduce $RETECS$, a prioritization method based on reinforcement learning. The code is available online[4]. RETECS corresponds to a model-free method, where there is no initial concept of the dynamics of the environment. The authors propose two policy models: a tableau representation (denoted *Tb* in Table 12), which consists of two tables to store the number of times an action was chosen and the average reward received; and an *Artificial Neural Network* (ANN), where the input is the current state of the environment and the output is an action. Here, an action consists in assigning a priority to a test. The priority can then be used for selection of the top x% tests. Three reward functions can give feedback to the agents after their actions. The first one aims to maximize the number of failed test cases in the selection (*Failure Count Reward*, denoted *FCount*). The second one, *Test Case Failure Reward* (*TCFail*), returns the verdicts of the selected test cases as a reward. In both cases, we compute the reward under the assumption of a selection of the top 40% tests. The third reward, *Time-Ranked Reward* (*TRR*), considers the position of each test, penalizing passed tests that are ranked ahead of failed ones. The variants combine each policy model with each reward function.

L8 and variants. Shi et al. (2020) also used reinforcement learning. The authors propose new reward functions, aiming to improve the *Test Case Failure Reward* from $RETECS$. They use the same agents as *L7* but the reward functions are now the weighted sum of the previous scores of each test. The rewards differ according to the weight function (*relu* or *tanh*) and the quantity of past information used (*all* or last $four$ cycles). Additionally, the method can be configured to reward all tests (*whole*) or only those that fail (*part*). We implemented the same subset of combinations as the ones considered by the authors.

L9 and variants. Lima and Vergilio (2022) implemented a Multi-Armed Bandit (MAB) algorithm to prioritize tests (**L9**). To balance between exploration and exploitation, three policies are presented: *ε-greedy*, *Upper Confidence Bound* (UCB), and *Fitness-Rate-Rank Multi-Armed Bandit* (*FRRMAB*). Two reward functions are used. The first one – *Reward Based on Failures* ($RNFail$) – is based on the result of the test, that is, 1 if the test fails and 0 otherwise. The second is the *Time-Ranked Reward* (TRR, the same as described for *L7*)

[3] https://github.com/T3AS/DeepOrder-ICSME21
[4] https://bitbucket.org/HelgeS/retecs/src/master/

Table 13: Summary of the evaluation process

	Desired effect		
	Decreased time for fault detection	Circulation of the tests	Reducing testing time and keeping fault detection capability
Datasets	The 4 original datasets	The 4 datasets with data augmentation under 3 profiles	The 4 original datasets
Scenarios	Verdict and Issue	Verdict	Verdict and Issue
Technique	Prioritization	Selection based on priority	
Tests to be prioritized at each cycle	Tests executed in the cycle historically	Tests available in the cycle historically	Tests executed in the cycle historically
Number of tests after selection	—	Same as historical number of tests executed in the cycle	40-90% of tests executed in the cycle historically
Evaluation metric	APFD and % of detections during the 1^{st} night	% of forgotten tests	% of detections at a given test size, with/without information loss

and it verifies the rank of failing tests. The algorithm is available online[5]. The variants combine each policy with each reward function.

8 Evaluation of RT Methods

Step 4 of the approach evaluates the candidate methods using the metrics from Step 3. Table 13 summarizes the decisions taken for the measurement of each desired effect.

The evaluation procedure follows three loops: for 40 variants of methods (listed in Table 12), 4 datasets (cf. Table 9), all cycles in each dataset. We repeat this procedure 30 times to account for randomness. Besides, to evaluate the first and third objectives, we use two evaluation scenarios: Verdict and Issue. We ignore a few cycles that do not contain at least one fail verdict or one issue. The effectiveness of a method at each cycle is then measured by the mean of the metric over the 30 repeated runs. For the second objective (test circulation), the metric is a global measure for all cycles. We have data augmentation under three profiles and, for each profile, report the average global metric over 30 runs.

8.1 Decreased Time for Fault Detection

The main effect expected when applying a test prioritization method is a reduction in fault detection time. Let us recall that we measure this by *APFD* for manual tests and *% of detections during the first night* for automated tests (see Table 13).

[5] https://github.com/jacksonpradolima/coleman4hcs

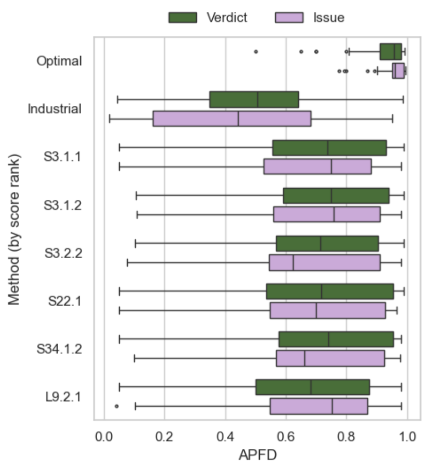
We might adopt different RT solutions for manual and automated tests, but would like the methods to be robust enough to adapt to both IVI and IVC tests, in both the Verdict and Issue scenarios.

For each method, we obtain eight boxplots showing the distribution of measures across cycles in each of the following settings: (manual/automated) x (Verdict/Issue) x (IVC/IVI). To identify the best and most robust methods, we derived an aggregated score that characterize the interquartile range in each setting. The score is the sum of the lower quartile ($Q1$), median ($Q2$), and upper quartile ($Q3$) observed in one setting. The higher the score, the better, since we seek for high values of the metrics in as many cycles as possible. We used this score to compare the methods with the original industrial prioritization and with each other. We extracted a short list of methods as follows. First, we required the candidate methods to improve the Industrial baseline. Out of 40 variants, 26 have scores that exceed the one of Industrial in all settings with manual tests, and 11 in all settings with automated tests. Then, we applied a second filter to retain the most effective from these subsets, based on their relative ranking by the score. For the prioritization of manual tests, we required methods to be in the top 5 for at least one of the four settings and in the top 10 for all of the four. This retained three variants of $S3$, and one of $S22$, $S34$ and $L9$. For automated tests, we used only the Top 5 requirement and retained four variants of $S3$, two of $S26$ and two of $L9$. Figure 4 displays their boxplots, as well as the ones of Optimal and Industrial baselines.

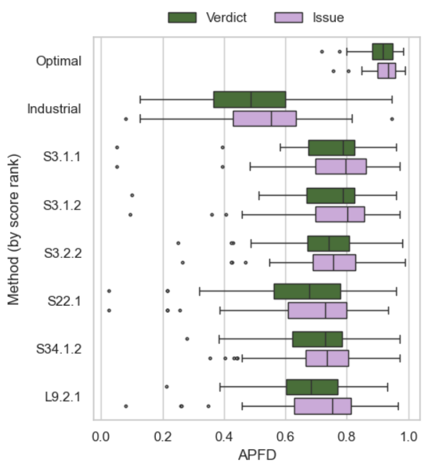
It is interesting to note that there are few machine learning-based variants remaining in the short list. The methods based on simple predefined heuristics, like the $S3$ variants, outperform them. As visualized by the boxplots, all the retained methods succeed in shifting the metric distributions towards higher values, thus improving the $Industrial$ baseline. An improvement is supplied in diverse settings, showing that the methods are versatile and can adapt to various types of data.

The methods that stand out are variants of $S3$ (ROCKET method). They are remarkably effective and robust, being in the short list for both manual and automated tests, while the other variants are more suited to one or the other type of tests. In particular, Variant $S3.1.2$ is in the top 5 of all *eight* settings, even if one considers the $full$ set of 40 methods and not just the ones that pass the first filter. $S3.1.1$ provides similar results, in the top 6 of all methods. They differ as follows. $S3.1.2$ corresponds to a ROCKET variant without time. When tests have the same priority based on their past results, there is a random tie breaker. In contrast, $S3.1.1$ (variant with time) favors the tests with the shortest execution time, in order to run as many tests as possible in a limited time. We observed very similar results but, to be fair, the constraints we had did not allow us to evaluate the prioritization in terms of time. Hence, we could not properly evaluate the effect of the different tie breakers. All what we can say is that, in ROCKET, the priority given by past results proves a very good criterion to order tests, without prejudging the improvement that a time-based refinement might (or might not) bring.

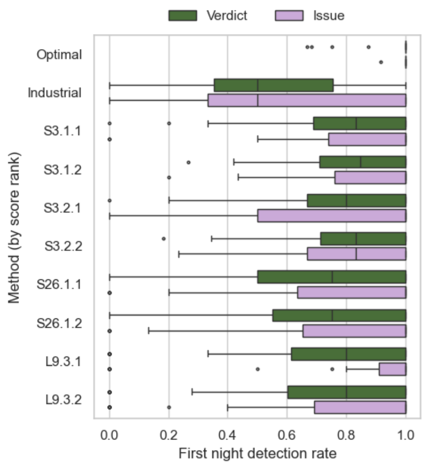
$S3$ was among the papers for which we reported a classification error at Step 2 of the approach. The authors of the taxonomy mapped $S3$ together with $S4$ because they are from the same group of researchers and $S4$ is just an extension of $S3$. However, $S4$ uses information items that are not available in Project A. If we had blindly accepted the mapping by Ali et al. (2019), we would have wrongly



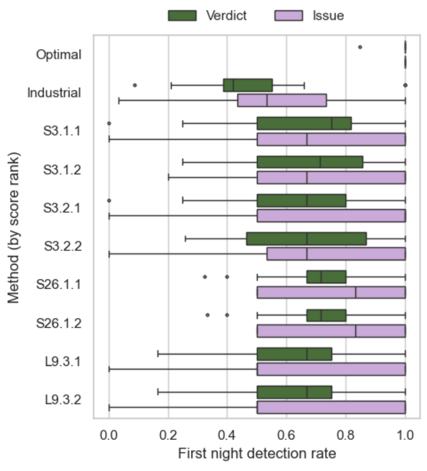
(a) *Manu IVI*



(b) *Manu IVC*



(c) *Auto IVI*



(d) *Auto IVC*

Fig. 4: Spread of the values of the metrics (*APFD* or *% of detections in the 1st night*) across all cycles

rejected $S3$. This would have deprived us of what turned out to be the most robust solution to our prioritization problem.

8.2 Circulation of the Tests

In addition to reducing fault detection time, the aim is also to avoid forgetting tests. A test is considered as *forgotten* if it is available but not executed for a long period of time (see the precise definition in Section 6.3). Table 14 displays the

Table 14: Average percentage of forgotten tests in 30 runs. Each method is evaluated with data augmentation for unknown verdicts: F = always fail, H = historical failure rate, P = always pass

		Manu IVI	Manu IVC	Auto IVI	Auto IVC
Optimal		0%			
Industrial		12.4%	16.6%	16.8%	13.9%
S3.1.1	F	**4.7%**	28.1%	**11.2%**	34.9%
	H	22.6%	17.7%	**14.2%**	**1.0%**
	P	22.5%	16.5%	19.1%	**1.2%**
S3.1.2	F	**3.9%**	29%	**10.4%**	34.4%
	H	**0.6%**	**3.3%**	**7.5%**	**0.7%**
	P	**0.4%**	**0.8%**	**2.8%**	**0.2%**
S3.2.1	F	**0% ★**	18.8%	**9.0%**	27.9%
	H, P	**0% ★**			
S3.2.2	F	**0% ★**	**12.8%**	**6.9%**	19.5%
	H, P	**0% ★**			
S22.1	F	29.6%	40.9%	36.0%	36.0%
	H	29.9%	37.9%	24.2%	36.5%
	P	30.2%	39.6%	25.8%	45.3%
S26.1.1	F, H, P	**0% ★**			
S26.1.2	F, H, P	**0% ★**			
S34.1.2	F	**0.7%**	**2.1%**	**0.4%**	**0.5%**
	H	**0.7%**	**1.8%**	**0.5%**	**0.1%**
	P	**0.6%**	**1.6%**	**0.6%**	**0.3%**
L9.2.1	F	29.2%	31.1%	21.0%	37.5%
	H	**7.4%**	17.6%	**12.2%**	17.5%
	P	**4.4%**	**8.4%**	**0.1%**	**11.9%**
L9.3.1	F	25.7%	24.0%	25.6%	27.9%
	H	23.0%	23.5%	24.0%	18.0%
	P	22.5%	25.1%	22.8%	17.4%
L9.3.2	F	26.7%	23.0%	25.8%	26.3%
	H	26.4%	24.0%	25.8%	21.8%
	P	26.5%	22.9%	25.8%	12.9%
★ No forgotten test	**In bold:** Better than *Industrial*				

percentage of forgotten tests supplied by the 11 methods that were in the short list for the previous objective. The evaluation involved data augmentation under three profiles.

Only two families of the selected methods deal explicitly with test circulation: $S26$ and $L9$. We expected them to be the most effective in achieving this objective. This was indeed the case for the $S26$ variants, but not for the $L9$ variants. In $S26$, the priority of a test takes into consideration the number of times it has remained unexecuted. This allowed the $S26$ variants not to forget any test, for any dataset and regardless of the data augment profile. $L9$ uses a *Multi-Armed Bandit* algorithm. The policies corresponding to the $L9.2.*$ and $L9.3.*$ variants have an exploration factor that should aid in selecting new tests. However, it seems that the balance between exploration and exploitation, while suitable for early detection, did not allow for the circulation of tests.

$S34.1.2$ is unexpectedly the second best choice for test circulation. Unlike the $S26$ variants, it does not reach zero forgotten tests, but their percentages remain consistently very low across all datasets and data augmentation profiles. The method ranks tests based on their failure rate within a time window. Its circulation results may be explained by the fact that, within the time window, most tests are either not run or passed. They end up having the same priority, and the random tie-breaker gives the forgotten tests a chance.

Tie-breakers also play a role in the circulation capability of $S3$ variants, with variants of the form $S3.*.2$ (random tie-breaker) tending to outperform their counterpart $S3.*.1$ (tie-breaker based on execution time). But the most impactful option concerns the interpretation of the "k last results" used to compute priority. These can be k last executions ($S3.1.*$) or k last cycles ($S3.2.*$), the cycle variant being better for test circulation. At each cycle, it distinguishes between tests that are not run and those that are passed: an unknown historical verdict is then penalized less than a pass verdict. All in all, combining the $S3$ options, the best variant for circulation is the one with cycles and a random tie-breaker, i.e., $S3.2.2$.

$S3.2.2$ is worth discussing because it ensures 0% of forgotten tests, except under the data augmentation profile where all unknown verdicts are treated as failures. The circulation capability of the method is thus sensitive to the presence of many failed tests. But, as also noted by Engström et al. (2010), this always–fail profile is the most unrealistic one. For Project A, it would imply that the testers are grossly mistaken in their selection of tests for the week. Since $S3.2.2$ can handle failure rates close to historical values, we believe it should not be discarded.

In our opinion, $S3.2.2$ actually offers the best trade-off between the early detection and circulation objectives. For early detection, it is a robust method with performance only slightly inferior to that of the top methods $S3.1.1$ and $S3.1.2$. For circulation, it makes all the difference: it ensures no forgotten tests if the failure rates are not too high. If this assumption is considered to be a major risk for circulation, then no $S3$ variant is suitable. One must accept having different solutions for different types of tests, rather than an all-in-one solution. Our recommendation would then be to use $S34.1.2$ for manual tests and $S26.1.1$ for automated tests.

8.3 Long-Term Objectives

The long term objective is to run fewer tests and still maintain a high percentage of detections. The target reduction would be to run only 40% of the historical number of tests per cycle, which approximates a reduction from 2.5 days/nights to 1. Our analysis of the margin for improvement showed that the target is challenging (see Section 6.4). Missing no detections would mean that the methods put few useless tests in their top 40% selection.

Figure 5 plots the average detection rate as a function of the percentage of tests selected. The evaluation focuses on methods that were in the short list for early detection, and that improved the circulation of tests under two of the data augmentation profiles (always–pass and historical failure rate). For manual tests, this leaves: $S3.1.2$, $S3.2.2$, and $S34.1.2$. For automated tests, the methods are: $S3.1.2$, $S3.2.1$, $S3.2.2$, $S26.1.1$ and $S26.1.2$.

The trends observed in Figure 5 confirm the difficulty of maintaining a high percentage of detections when only 40% of the tests are run. Even the best methods may provide less than 70% of detections in some settings. In the worst case, a method is not better than random selection. This is observed for $SS34.1.2$ in the Manu IVI/IVC Verdict settings: selecting 40% of the tests yields an average of 40% detections. Given these results, less ambitious targets, such as running 60% or even 80% of the tests would seem more realistic. An effective method such as

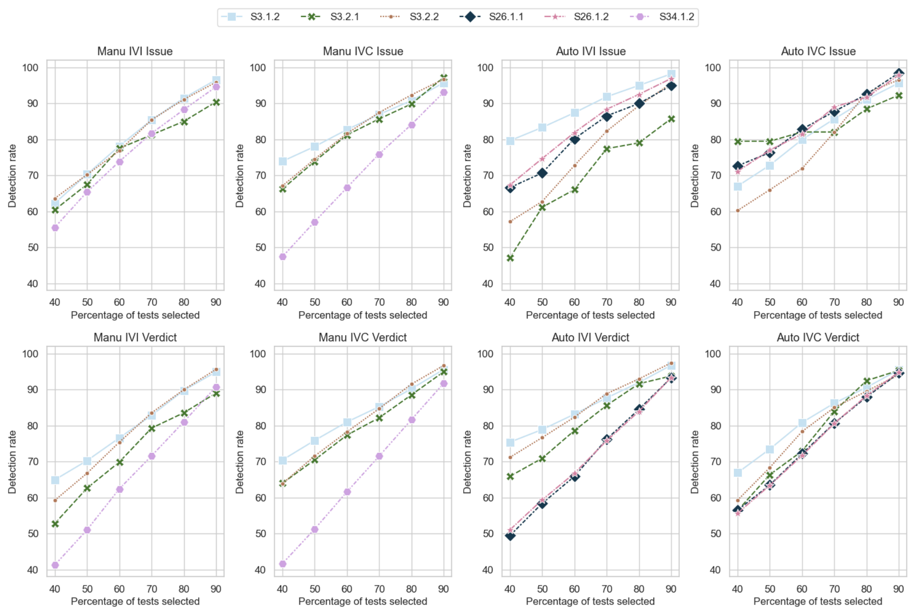
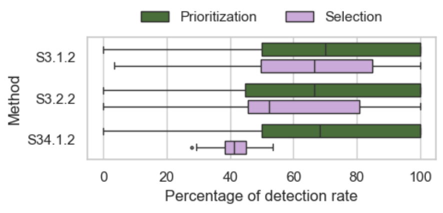


Fig. 5: Average detection rate as a function of the percentage of tests selected (across all cycles, 30 repeated runs)

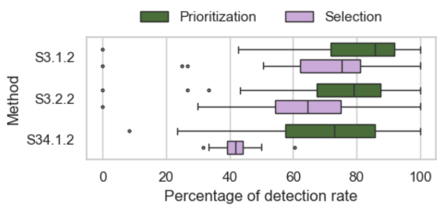
$S3.1.2$ would achieve 80% or 90% of detections at these sizes in all settings. Note, however, that these figures are for scenarios where the selection is from the set of tests that have been run historically. We do not know what the figures would be if the selection was from the set of all historically *available* tests. But it seems unlikely that the 40% target would be much less challenging in the latter case.

When the test size is reduced, RT methods have to deal with more unknowns. To study this effect, we zoom on the 40% size and compare the spread of the detection rates for two usages of the methods: for *prioritization* only or for priority-based *selection*. The first usage, for *prioritization*, is the baseline. The complete set of ordered tests is executed and information about their outcomes is made available for future decisions. The second one, prioritization followed by *selection*, corresponds to what we are studying here. Only a subset of tests is run, the top 40%, so the outcomes of the remaining 60% are unknown to the methods. By contrasting the detection rates at 40% for prioritization or selection, it is possible to study the robustness of the methods to information loss.

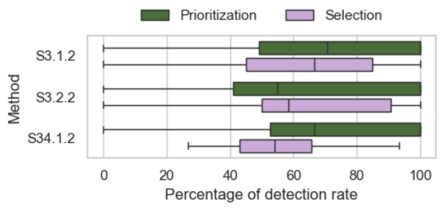
For manual tests (see Figure 6), the most severely impacted method is $S34.1.2$. Its prioritization algorithm simply ranks tests according to their failure rate over the last 10 cycles. When too many test outcomes are unknown, the method tends to become random, as already noted. The $S3$ variants are more robust, although also impacted. For automated tests (see Figure 7), we observe that the $S3$ and $S26$ variants are surprisingly unaffected in one of the settings (Auto IVC Issue), but can be severely affected in others (see, e.g., the results of $S3.2.1$ in Auto IVI Issue). $S3.1.2$ appears to be the most robust across the four settings.



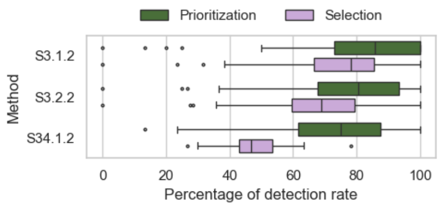
(a) *Manu IVI + Verdict*



(b) *Manu IVC + Verdict*

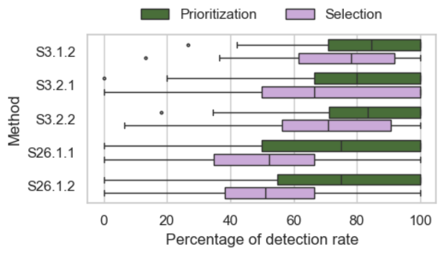


(c) *Manu IVI + Issue*

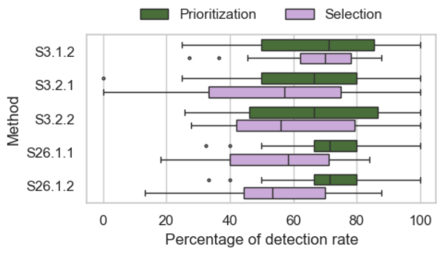


(d) *Manu IVC + Issue*

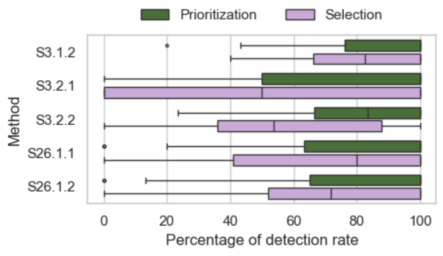
Fig. 6: Impact of information loss on the detection at a size of 40% and for manual tests. *Prioritization:* no information loss. *Selection:* the unexecuted tests have unknown outcomes



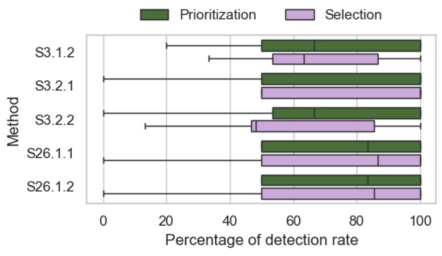
(a) *Auto IVI + Verdict*



(b) *Auto IVC + Verdict*



(c) *Auto IVI + Issue*



(d) *Auto IVC + Issue*

Fig. 7: Impact of information loss on the detection at a size of 40% and for automated tests. *Prioritization:* no information loss. *Selection:* the unexecuted tests have unknown outcomes

In conclusion, $S3.1.2$ was a very good choice for prioritization only (Objective 1), and remains a good choice for priority-based test selection. However, the reduction in test size should probably be much less drastic than originally considered if high detection power is to be maintained. An online evaluation of the method is strongly recommended before making a final decision.

8.4 Feedback on the Taxonomy-based Approach (RQ4)

In discussing feedback, we need to distinguish between the principle of the approach, and its implementation with the current version of the taxonomy.

The taxonomy captures relevant domain-specific knowledge about RT problems and solutions, which the approach uses in its steps to define the problem, identify the feasible solutions and develop an evaluation plan. For the case study presented in this paper, the approach delivered on its promise. It enabled us to identify a subset of RT methods (from 8 papers out of 52), among which some proved to be suitable for Project A. The ROCKET method ($S3$) was found to be particularly commendable, since it can adapt to the different systems developed in Project A (IVI, IVC), the different types of tests (manual, automated), and the different types of detections (in terms of verdicts or issues).

While the principle of the taxonomy-based approach proved to be successful, its implementation faced many difficulties. The identification of feasible solutions was hampered by errors in the mappings of papers, and by missing or unclear elements in the information part of the taxonomy. Most notably, the errors would have excluded $S3$ as infeasible if we had not re-checked all the mappings.

RQ4 (efficiency in finding solutions): The taxonomy-based approach was successful in finding solutions for Project A, but at the cost of fixing errors. The taxonomy would have to mature.

9 Threats to Validity

A threat to internal validity concerns the understanding of the industrial RT problem. It was addressed by several meetings as well as direct discussions with engineers. Besides, we could consolidate our understanding by consulting the data of Project A, which was made fully available to us. At Step 2 of the approach, there was the threat induced by our subjective interpretation of the taxonomy elements. We mapped 14 new papers to the taxonomy. To mitigate the introduction of a personal bias in the process, we systematically double-checked the mapping. Moreover, we explicitly reported the difficulties we encountered and explained the related mapping decisions. We also systematically double-checked the disagreements we had on the mapping of the original set of papers. The choice of articles to represent ML-based methods is another threat. We made sure that diverse categories of methods were included (supervised learning, unsupervised learning, reinforcement learning, and natural language processing). But, like in the original work to build the taxonomy, there is no claim for completeness. In particular, the information part of the taxonomy may have more missing elements than the ones we identified.

Implementing the methods proposed in the literature also poses a threat to internal validity. Since most of the papers did not provide source codes, the development of the algorithms relied on our interpretation. Furthermore, the three codes provided by the authors had to be changed, either to fix them or to adapt them to our data and our context. Whenever we had doubts about the supplied code or about how to implement a method, we contacted the authors. However, we did not always get an answer. Our code is made openly available for inspection by others.

The evaluation had to be performed on incomplete historical data. Obviously, this is a major threat to internal validity, since the selection capabilities of the methods could not be evaluated for tests that had not been run historically. There was no means to remove this threat. It arises in any industrial context where running the full test suite for every software release is impractical.

Moreover, we had to perform data augmentation in order to evaluate the objective of test circulation. This introduces another threat to validity. To mitigate this threat, we included three data augmentation profiles. In this way, consistent performance across profiles can support that test circulation is a property of the method itself, not an artifact of augmentation.

Given the threats posed by the evaluation on historical data, the conclusions delivered to Renault included a strong warning that the methods distinguished by our study ($S3.1.2$ for detection purposes, $S3.2.2$ for a trade-off between early detection and circulation) should now be evaluated online.

Finally, applying the taxonomy to a specific industrial context could threaten the external validity of this study. While the taxonomy-based approach successfully identified effective methods in the context under study, this outcome might not be achieved in some other industrial contexts. This threat is common to any in-depth industrial study. As regards the difficulties reported in the use of the taxonomy, they do not appear to be specific to Project A. Additionally, the four-step methodology we proposed is generic and broadly applicable.

10 Conclusion

This empirical study contributes to a line of research on how to assess industrial relevance and applicability of RT methods. It presents a real-world scenario in which practitioners search for solutions to their specific problems. The premise of our work is that, given the large body of research proposed so far, effective methods likely exist for many industrial RT problems, but practitioners need help to find them. The taxonomy proposed by Ali et al. (2019) serves as an alignment between the numerous published methods and the industrial problem.

The taxonomy introduces a decomposition into context, effects, and information factors. They capture different aspects to consider, which we have integrated into a four-step approach to tackle a real-world scenario. Step 1 is the problem characterization, independently from any solution. Step 2 analyzes the feasibility of the solutions based on the alignment between the required and available information items. Step 3 checks the alignment between the desired effects of solutions and the effects addressed by the methods. In this step, we also develop an evaluation plan for each desired effect and study the room for improvement. Step 4 involves the implementation and evaluation of the candidate methods according to the plan.

Applied to Project A, the approach was successful in identifying effective RT methods. This positive outcome provides support to the premise of our work (and of the taxonomy) that industry-relevant challenges may currently lie more in the search for already existing methods than in the development of new solutions. Interestingly, the method that stands out in this case study, ROCKET, corresponds to a simple heuristic introduced more than ten years ago (Marijan et al., 2013). It demonstrated remarkable robustness across various datasets, making it a strong recommendation for the industrial partner. For Project A, it outperforms more recent and more sophisticated methods based on machine learning.

While the principle of considering context, effects and information factors proved successful in our approach, the low-level content of the taxonomy was not as useful as it could have been.

Regarding the information part, the use of the taxonomy was hindered by hard-to-interpret elements, missing elements, and errors in the classification of candidate methods. The classification errors would have eliminated the method that we eventually recommend for Project A. Our experience suggests several potential improvements to the taxonomy, such as simplifying items related to test *execution time*, merging two attributes of *test reports – verdicts* and *link to failure*) – that appear to be essentially the same, and adding attributes for methods using project artifacts in natural language.

The effect part was helpful in characterizing the problem at a high level, but not so helpful in deciding how to measure the desired effects. We had to go through all the papers mapped to an effect just to get an overview of the metrics used in practice. The taxonomy also did not aid in identifying the evaluation constraints occurring in each paper, e.g., whether the evaluation was done online or offline, or whether all test outcomes were known or not. We found that such constraints are very important when preparing an evaluation plan. This suggests that the effect part of the taxonomy should be refined to a level of detail comparable to the information part, incorporating attributes that capture both evaluation metrics and relevant constraints.

Our experience thus shows that the taxonomy has practical value but needs improvement. The version published in Ali et al. (2019) is a relevant contribution to RT problem solving but may not be considered as a finished product. A means to manage its improvement would be to deploy and maintain a public repository, in the spirit of the live literature review created by Greca et al. (2023). Users would find updated versions of the taxonomy, a documentation, a database of papers mapped to the taxonomy, and could contribute by providing feedback.

Acknowledgements The authors would like to thank the Ampere Software Technology teams working on Project A. Their help was crucial throughout this study. Several researchers kindly answered questions about the RT methods they have developed: Jeffrey Anderson (S34), Aizaz Sharif (L4), Helge Spieker (L7), Jackson Antonio do Prado Lima and Silvia Regina Vergilio (L9). Any remaining misunderstandings would be our fault, not theirs.

Declarations

Funding: This work has been supported in part by the French National Association of Research and Technology (ANRT) with a CIFRE fellowship granted to Maria Laura Brzezinski Meyer.

Ethical Approval: not applicable.

Informed Consent: not applicable.

Author Contributions:

– *Maria Laura Brzezinski Meyer*: Conceptualization, Data Curation, Formal analysis, Investigation, Methodology, Software, Visualization, Writing Original Draft, Writing Review & Editing;
– *Hélène Waeselynck*: Conceptualization, Formal Analysis, Funding Acquisition, Investigation, Methodology, Supervision, Writing Original Draft, Writing Review & Editing;
– *Fernand Cuesta*: Conceptualization, Funding Acquisition, Investigation, Resources, Validation, Writing Review & Editing.

Data Availability Statements: The datasets used in this study are from Ampere Software Technology. The authors are not allowed to make them publicly available. However, the code of the methods analyzed is provided in a repository at: *https://github.com/laurabrzmeyer/papers-implementation*. The code for all experiments and assessments reported in this study is available upon request.

Conflict of Interest: The authors declare that they have no conflict of interest.

Clinical Trial Number: not applicable.

References

Ali NB, Engström E, Taromirad M, Mousavi MR, Minhas NM, Helgesson D, Kunze S, Varshosaz M (2019) On the search for industry-relevant regression testing research. Empirical Software Engineering 24(4):2020–2055, DOI 10.1007/s10664-018-9670-1

Anderson J, Salem S, Do H (2014) Improving the effectiveness of test suite through mining historical data. In: 11th Working Conf. on Mining Software Repositories (MSR), p 142–151, DOI 10.1145/2597073.2597084

Bertolino A, Guerriero A, Miranda B, Pietrantuono R, Russo S (2020) Learning-to-rank vs ranking-to-learn: Strategies for regression testing in continuous integration. In: ACM/IEEE 42nd Int. Conf. on Software Engineering (ICSE), pp 1–12, DOI 10.1145/3377811.3380369

Brzezinski Meyer ML, Waeselynck H, Cuesta F (2023) A case study on the “jungle” search for industry-relevant regression testing. In: 2023 IEEE 23rd International Conference on Software Quality, Reliability, and Security (QRS), pp 382–393, DOI 10.1109/QRS60937.2023.00045

Buchgeher G, Ernstbrunner C, Ramler R, Lusser M (2013) Towards tool-support for test case selection in manual regression testing. In: 2013 IEEE 6th Int. Conf. on Software Testing, Verification and Validation Workshops (ICSTW), pp 74–79, DOI 10.1109/ICSTW.2013.16

Busjaeger B, Xie T (2016) Learning for test prioritization: an industrial case study. In: 24th ACM SIGSOFT Int. Symp. on Foundations of Software Engineering (FSE), pp 975–980, DOI 10.1145/2950290.2983954

Carlson R, Do H, Denton A (2011) A clustering approach to improving test case prioritization: An industrial case study. In: 27th IEEE Int. Conf. on Software Maintenance (ICSM), pp 382–391, DOI 10.1109/ICSM.2011.6080805

Chen S, Chen Z, Zhao Z, Xu B, Feng Y (2011) Using semi-supervised clustering to improve regression test selection techniques. In: 4th IEEE Int. Conf. on Software Testing, Verification and Validation (ICST), pp 1–10, DOI 10.1109/ICST.2011.38

Şmite D, Wohlin C, Galviundefineda Z, Prikladnicki R (2014) An empirically based terminology and taxonomy for global software engineering. Empirical Software Engineering 19(1):105–153, DOI 10.1007/s10664-012-9217-9, URL https://doi.org/10.1007/s10664-012-9217-9

Devaki P, Thummalapenta S, Singhania N, Sinha S (2013) Efficient and flexible gui test execution via test merging. In: Int. Symp. on Software Testing and Analysis (ISSTA), pp 34–44, DOI 10.1145/2483760.2483781

Dyba T, Kitchenham BA, Jorgensen M (2005) Evidence-based software engineering for practitioners. IEEE Software 22(1):58–65, DOI 10.1109/MS.2005.6, URL https://doi.org/10.1109/MS.2005.6

Ekelund ED, Engström E (2015) Efficient regression testing based on test history: An industrial evaluation. In: 2015 IEEE Int. Conf. on Software Maintenance and Evolution (ICSME), pp 449–457, DOI 10.1109/ICSM.2015.7332496

Engström E, Runeson P, Wikstrand G (2010) An empirical evaluation of regression testing based on fix-cache recommendations. In: 3rd Int. Conf. on Software Testing, Verification and Validation (ICST), pp 75–78, DOI 10.1109/ICST.2010.40

Engström E, Runeson P, Ljung A (2011) Improving regression testing transparency and efficiency with history-based prioritization - an industrial case study. 4th IEEE Int Conf on Software Testing, Verification and Validation (ICST) pp 367–376

Engström E, Petersen K, Ali NB, Bjarnason E (2017) Serp-test: a taxonomy for supporting industry—academia communication. Software Quality Journal 25(4):1269–1305, DOI 10.1007/s11219-016-9322-x, URL https://doi.org/10.1007/s11219-016-9322-x

Erik Strandberg P, Afzal W, Ostrand TJ, Weyuker EJ, Sundmark D (2017) Automated system-level regression test prioritization in a nutshell. IEEE Software 34(4):30–37, DOI 10.1109/MS.2017.92

Fazlalizadeh Y, Khalilian A, Abdollahi Azgomi M, Parsa S (2009) Incorporating historical test case performance data and resource constraints into test case prioritization. International conference on tests and proofs 5668:43–57, DOI 10.1007/978-3-642-02949-3_5

Forward A, Lethbridge TC (2008) A taxonomy of software types to facilitate search and evidence-based software engineering. In: Proceedings of the 2008 Conference of the Center for Advanced Studies on Collaborative Research: Meeting of Minds (CASCON '08), Association for Computing Machinery, New York, NY, USA, DOI 10.1145/1463788.1463807, URL https://doi.org/10.1145/1463788.1463807

Gligoric M, Negara S, Legunsen O, Marinov D (2014) An empirical evaluation and comparison of manual and automated test selection. 29th ACM/IEEE Int Conf on Automated Software Engineering (ASE) pp 361–371, DOI 10.1145/2642937.2643019

Greca R, Miranda B, Bertolino A (2023) State of practical applicability of regression testing research: A live systematic literature review. ACM Comput Surv 55(13s), DOI 10.1145/3579851, URL https://doi.org/10.1145/3579851

Herzig K, Greiler M, Czerwonka J, Murphy B (2015) The art of testing less without sacrificing quality. In: 37th IEEE/ACM International Conference on Software Engineering (ICSE 2015), pp 483–493

Hirzel M, Klaeren H (2016) Graph-walk-based selective regression testing of web applications created with google web toolkit. In: Software Engineering (workshops)

Huang S, Chen Y, Zhu J, Li ZJ, Tan HF (2009) An optimized change-driven regression testing selection strategy for binary java applications. In: 2009 ACM Symp. on Applied Computing (SAC), pp 558–565, DOI 10.1145/1529282.1529403

Hwang CL, Yoon K (1981) Multiple Attribute Decision Making, 1st edn. Springer Berlin, Heidelberg

Janjua MU (2015) Onspot system: Test impact visibility during code edits in real software. In: 2015 10th Joint Meeting on Foundations of Software Engineering (ESEC/FSE), pp 994–997, DOI 10.1145/2786805.2804430

Jordan C, Foth P, Pretschner A, Fruth M (2022) Unreliable test infrastructures in automotive testing setups. In: 2022 IEEE/ACM 44th Int. Conf. on Software Engineering: Software Engineering in Practice (ICSE-SEIP), pp 307–308, DOI 10.1145/3510457.3513069

Kandil P, Moussa S, Badr N (2016) Cluster-based test cases prioritization and selection technique for agile regression testing. Journal of Software: Evolution and Process 29(6), DOI 10.1002/smr.1794

Khatibsyarbini M, Isa MA, Jawawi DN, Tumeng R (2018) Test case prioritization approaches in regression testing: A systematic literature review. Information and Software Technology 93:74–93, DOI https://doi.org/10.1016/j.infsof.2017.08.014

Krishnamoorthi R, Mary SASA (2009) Factor oriented requirement coverage based system test case prioritization of new and regression test cases. Inf Softw Technol 51(4):799–808, DOI https://doi.org/10.1016/j.infsof.2008.08.007

Lachmann R, Schulze S, Nieke M, Seidl C, Schaefer I (2016) System-level test case prioritization using machine learning. In: 15th IEEE Int. Conf. on Machine Learning and Applications (ICMLA), pp 361–368, DOI 10.1109/ICMLA.2016.0065

Le Goues C, Jaspan C, Ozkaya I, Shaw M, Stolee KT (2018) Bridging the gap: From research to practical advice. IEEE Software 35(5):50–57, DOI 10.1109/MS.2018.3571235

Li Q, Boehm B (2013) Improving scenario testing process by adding value-based prioritization: An industrial case study. In: Int. Conf. on Software and System Process (ICSSP), pp 78–87

Lima JAP, Vergilio SR (2020) Test case prioritization in continuous integration environments: A systematic mapping study. Information and Software Technology 121:106268, DOI https://doi.org/10.1016/j.infsof.2020.106268

Lima JAP, Vergilio SR (2022) A multi-armed bandit approach for test case prioritization in continuous integration environments. IEEE Transactions on Software Engineering 48(2):453–465, DOI 10.1109/TSE.2020.2992428

Lochau M, Lity S, Lachmann R, Schaefer I, Goltz U (2014) Delta-oriented model-based integration testing of large-scale systems. Journal of Systems and Software 91:63–84, DOI https://doi.org/10.1016/j.jss.2013.11.1096

Mahdieh M, Mirian-Hosseinabadi SH, Etemadi K, Nosrati A, Jalali S (2020) Incorporating fault-proneness estimations into coverage-based test case prioritization methods. Information and Software Technology 121:106269, DOI https://doi.org/10.1016/j.infsof.2020.106269

Marijan D (2015) Multi-perspective regression test prioritization for time-constrained environments. In: 2015 IEEE Int. Conf. on Software Quality, Reliability and Security (QRS), pp 157–162, DOI 10.1109/QRS.2015.31

Marijan D, Gotlieb A, Sen S (2013) Test case prioritization for continuous regression testing: An industrial case study. In: 2013 IEEE Int. Conf. on Software Maintenance (ICSM), pp 540–543, DOI 10.1109/ICSM.2013.91

Marijan D, Gotlieb A, Liaaen M (2019) A learning algorithm for optimizing continuous integration development and testing practice. Softw Pract Exp 49(2):192–213, DOI 10.1002/spe.2661

Medhat N, Moussa SM, Badr NL, Tolba MF (2020) A framework for continuous regression and integration testing in iot systems based on deep learning and search-based techniques. IEEE Access 8:215716–215726, DOI 10.1109/ACCESS.2020.3039931

Palma F, Abdou T, Bener A, Maidens J, Liu S (2018) An improvement to test case failure prediction in the context of test case prioritization. In: 14th Int. Conf. on Predictive Models and Data Analytics in Software Engineering (PROMISE), pp 80–89, DOI 10.1145/3273934.3273944

Pan R, Bagherzadeh M, Ghaleb TA, Briand L (2022) Test case selection and prioritization using machine learning: a systematic literature review. Empirical Software Engineering 27(2), DOI 10.1007/s10664-021-10066-6

Pasala A, Bhowmick A (2005) An approach for test suite selection to validate applications on deployment of cots upgrades. In: 12th Asia-Pacific Software Engineering Conference (APSEC'05), pp 361–364, DOI 10.1109/APSEC.2005.31

Qu X, Cohen MB, Woolf KM (2007) Combinatorial interaction regression testing: A study of test case generation and prioritization. In: IEEE Int. Conf. on Software Maintenance (ICSM), pp 255–264, DOI 10.1109/ICSM.2007.4362638

Rogstad E, Briand L (2016) Cost-effective strategies for the regression testing of database applications: Case study and lessons learned. Journal of Systems and Software 113:257–274, DOI https://doi.org/10.1016/j.jss.2015.12.003

Rogstad E, Briand L, Torkar R (2013) Test case selection for black-box regression testing of database applications. Information and Software Technology 55(10):1781–1795, DOI https://doi.org/10.1016/j.infsof.2013.04.004

Rothermel G, Untch R, Chu C, Harrold M (1999) Test case prioritization: an empirical study. In: IEEE Int. Conf. on Software Maintenance (ICSM), pp 179–188, DOI 10.1109/ICSM.1999.792604

Saha RK, Zhang L, Khurshid S, Perry DE (2015) An information retrieval approach for regression test prioritization based on program changes. In: 37th IEEE/ACM Int. Conf. on Software Engineering (ICSE), vol 1, pp 268–279, DOI 10.1109/ICSE.2015.47

Sharif A, Marijan D, Liaaen M (2021) Deeporder: Deep learning for test case prioritization in continuous integration testing. In: 2021 IEEE Int. Conf. on Software Maintenance and Evolution (ICSME), pp 525–534

Shi T, Xiao L, Wu K (2020) Reinforcement learning based test case prioritization for enhancing the security of software. In: 2020 IEEE 7th Int. Conf. on Data Science and Advanced Analytics (DSAA), pp 663–672, DOI 10.1109/DSAA49011.2020.00076

Skoglund M, Runeson P (2005) A case study of the class firewall regression test selection technique on a large scale distributed software system. In: Int. Symp. on Empirical Software Engineering (ISESE), pp 74–83, DOI 10.1109/ISESE.2005.1541816

Spieker H, Gotlieb A, Marijan D, Mossige M (2017) Reinforcement learning for automatic test case prioritization and selection in continuous integration. 26th ACM SIGSOFT Int Symp on Software Testing and Analysis (ISSTA) pp 12–22, DOI 10.1145/3092703.3092709

Srivastava A, Thiagarajan J (2002) Effectively prioritizing tests in development environment. In: 2002 ACM SIGSOFT Int. Symp. on Software Testing and Analysis (ISSTA), pp 97–106, DOI 10.1145/566172.566187

Tahvili S, Afzal W, Saadatmand M, Bohlin M, Sundmark D, Larsson S (2016) Towards earlier fault detection by value-driven prioritization of test cases using fuzzy topsis. In: 13th Int. Conf. on Information Technology: New Generations (ITNG 2016), pp 745–759, DOI 10.1007/978-3-319-32467-8_65

Vöst S, Wagner S (2016) Trace-based test selection to support continuous integration in the automotive industry. In: Int. Workshop on Continuous Software Evolution and Delivery (CSED@ICSE), pp 34–40, DOI 10.1145/2896941.2896951

Wang S, Ali S, Gotlieb A (2013a) Minimizing test suites in software product lines using weight-based genetic algorithms. In: 15th Annual Conf. on Genetic and Evolutionary Computation (GECCO), pp 1493–1500, DOI 10.1145/2463372.2463545

Wang S, Gotlieb A, Ali S, Liaaen M (2013b) Automated test case selection using feature model: An industrial case study. In: 16th Int. Conf. on Model-Driven Engineering Languages and Systems (MODELS), pp 237–253, DOI 10.1007/978-3-642-41533-3_15

Wang S, Buchmann D, Ali S, Gotlieb A, Pradhan D, Liaaen M (2014) Multi-objective test prioritization in software product line testing: An industrial case study. In: 18th Int. Software Product Line Conference (SPLC), pp 32–41, DOI 10.1145/2648511.2648515

Wang S, Ali S, Gotlieb A (2015) Cost-effective test suite minimization in product lines using search techniques. J Syst Softw 103:370–391, DOI https://doi.org/10.1016/j.jss.2014.08.024

Wang S, Ali S, Yue T, Bakkeli O, Liaaen M (2016) Enhancing test case prioritization in an industrial setting with resource awareness and multi-objective search. In: 38th IEEE/ACM Int. Conf. on Software Engineering Companion (ICSE-C), pp 182–191

Wang S, Ali S, Gotlieb A, Liaaen M (2017) Automated product line test case selection: industrial case study and controlled experiment. Softw Syst Model 16(2):417–441, DOI 10.1007/s10270-015-0462-4

White L, Robinson B (2004) Industrial real-time regression testing and analysis using firewalls. In: 20th IEEE Int. Conf. on Software Maintenance (ICSM), pp

18–27, DOI 10.1109/ICSM.2004.1357786

White L, Jaber K, Robinson B, Rajlich V (2008) Extended firewall for regression testing: an experience report. Journal of Software Maintenance and Evolution: Research and Practice 20(6):419–433, DOI https://doi.org/10.1002/smr.371

Wikstrand G, Feldt R, Gorantla JK, Zhe W, White C (2009) Dynamic regression test selection based on a file cache an industrial evaluation. In: 2nd Int. Conf. on Software Testing Verification and Validation (ICST), pp 299–302, DOI 10.1109/ICST.2009.42

Yoo S, Harman M (2012) Regression testing minimization, selection and prioritization: A survey. Softw Test Verif Reliab 22(2):67–120, DOI 10.1002/stv.430

Yoo S, Harman M, Tonella P, Susi A (2009) Clustering test cases to achieve effective & scalable prioritisation incorporating expert knowledge. In: 8th Int. Symp. on Software Testing and Analysis (ISSTA), pp 201–212, DOI 10.1145/1572272.1572296

Zheng J (2005) In regression testing selection when source code is not available. In: 20th IEEE/ACM Int. Conf. on Automated Software Engineering (ASE'05), pp 752–755

Zheng J, Robinson B, Williams L, Smiley K (2006a) Applying regression test selection for cots-based applications. In: 28th Int. Conf. on Software Engineering (ICSE '06), pp 512–522, DOI 10.1145/1134285.1134357

Zheng J, Robinson B, Williams L, Smiley K (2006b) A lightweight process for change identification and regression test selection in using cots components. In: 5th Int. Conf. on Commercial-off-the-Shelf (COTS)-Based Software Systems (ICCBSS), pp 137–143, DOI 10.1109/ICCBSS.2006.1

Zheng J, Williams L, Robinson B (2007) Pallino: Automation to support regression test selection for cots-based applications. In: 22nd IEEE/ACM Int. Conf. on Automated Software Engineering (ASE), pp 224–233, DOI 10.1145/1321631.1321665