A Case Study on the "Jungle" Search for Industry-Relevant Regression Testing

Maria Laura Brzezinski Meyer, Hélène Waeselynck, Fernand Cuesta

A Case Study on the "Jungle" Search for Industry-Relevant Regression Testing

Maria Laura Brzezinski Meyer[1,2,*], Hélène Waeselynck[2], and Fernand Cuesta[1]
[1]Renault Software Factory, Toulouse, France
[2]LAAS-CNRS, Université de Toulouse, France
mlbrzezins@laas.fr, helene.waeselynck@laas.fr, fernand.cuesta@renault.com
*corresponding author

Abstract—The optimization of regression testing (RT) has been widely studied in the literature, and numerous methods exist. However, each context is unique. Therefore, how to tell which method is appropriate for a specific industrial context? Recent work has proposed a taxonomy to aid in answering this question. The approach is to map both the RT problem and existing solutions onto the taxonomy, aiming to determine which solutions are best aligned with the problem. This paper presents a case study that evaluates the approach in a real setting. The context is the development of R&D projects at a major automotive company, in the domain of connected vehicles. We used the taxonomy to characterize the RT problem in terms of measurable effects, and to identify the technically feasible solutions from a set of 52 papers. We report on the beneficial aspects but also the difficulties of the approach, due to unclear taxonomy elements, missing ones and paper classification errors.

Keywords–Regression testing; taxonomy; industrial case study

1. Introduction

Regression testing (RT) verifies that the changes in the code did not introduce new issues, that is, the software did not regress. Since re-runing all tests is usually not practical, there is a need for some test optimization. This is called the RT problem and has been an active field of research for decades. The methods that have emerged can be divided into three categories [1]: 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. The 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 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. Or all tests are prioritized but only the highest-priority ones are selected. Conceptually, the three categories of methods are closely related. All of them involve some comparative characterization of the tests, based on which an optimal decision has to be taken (which tests to remove, retain or execute first).

In the article titled "*On the search for industry-relevant regression testing research*" [2], Ali et al. noted that, despite the large body of work on the RT problem, there is limited dissemination in industry. They identified two main difficulties faced by practitioners. The first one comes from the discrepancies in terminology between academia and industry: it makes it harder for practitioners to search for methods in the literature. The second one is the difficulty to assess the relevance of the methods. The research is typically done in a controlled environment, which differs from the complexity of an industrial scenario.

In order to help practitioners to interpret, compare and contrast the methods in the literature, Ali et al. [2] 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 presents an application and an evaluation of the taxonomy to assist in a real RT problem. The context is the development of R&D projects at Renault for connected vehicles. We used the taxonomy to extract a set of 8 promising methods out of 52 papers. The methods are currently being experimented at Renault. We report on the beneficial aspects but also the difficulties of using the taxonomy to identify the applicable and relevant RT research.

The paper is structured as follows. Section 2 introduces the background of this work. Section 3 describes the methodology we used to apply and assess the taxonomy. The industrial RT problem is first characterized by its context and desired effects (Section 4). Then, Section 5 analyzes the feasibility of the methods from the literature, based on the information items they require. Section 6 analyzes how the desired effects can be measured in the industrial context. Section 7 discusses the threats to validity and Section 8 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 [2], the authors identified eleven reviews of software RT literature between 2010 and 2017. New ones have been published since that time, e.g., [3] [4] [5]. The existing 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. This is why Ali et al. [2] introduced a taxonomy, allowing an alignment between methods proposed in the literature and real industrial contexts. 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 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 reports 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 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 to study the use of the taxonomy.

3. Methodology

The study takes the perspective of an industrial company wishing to improve its RT process. From a set of research papers, the company would like to know which methods to select for an experimentation in its context. The general research question is whether the taxonomy is useful to aid in this selection and in the preparation of experiments.

The study 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 in [2] (from $S1$ to $S38$). These papers describe methods that were identified as industry-relevant by the authors of the taxonomy and their industrial partners. But none of these papers was published after 2016, and their set does not reflect the growing interest on 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 [5]. The review classifies 29 papers depending on whether they use

Table 1. Selected papers on regression testing from [2]

ID	Ref
S1	[8]
S2	[12]
S3	[16]
S4	[20]
S5	[24]
S6	[28]
S7	[32]
S8	[36]
S9	[40]
S10	[43]

ID	Ref
S11	[9]
S12	[13]
S13	[17]
S14	[21]
S15	[25]
S16	[29]
S17	[33]
S18	[37]
S19	[41]
S20	[44]

ID	Ref
S21	[10]
S22	[14]
S23	[18]
S24	[22]
S25	[26]
S26	[30]
S27	[34]
S28	[38]
S29	[42]
S30	[45]

ID	Ref
S31	[11]
S32	[15]
S33	[19]
S34	[23]
S35	[27]
S36	[31]
S37	[35]
S38	[39]

Table 2. Added papers on ML-based methods

ID	Ref	Method
L1	[46]	SL
L2	[6]	SL
L3	[49]	SL
L4	[7]	SL
L5	[52]	UL
L6	[54]	UL
L7	[56]	RL

ID	Ref	Method
L8	[47]	RL
L9	[48]	RL
L10	[50]	SL + RL
L11	[51]	SL + NLP
L12	[53]	SL + NLP
L13	[55]	SL + NLP
L14	[57]	UL + NLP

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 [6] [7], 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 ($L1$, ... $L14$). We had to map them to the taxonomy, and did so with a double check by different authors.

The real-world scenario is a Renault project under development. We had full access to its data. We also interacted with the engineers working on the project all along the study. The search for relevant methods followed three steps, shown in Figure 1. Each step leverages the taxonomy for a different purpose, in accordance with the role of the context, effect and information described in [2]. 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. Each step provides feedback on the taxonomy. Hence, the general question about the utility of the taxonomy is actually refined into three research questions, one for each step.

- **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 ways to assess the solutions in a specific industrial context?*

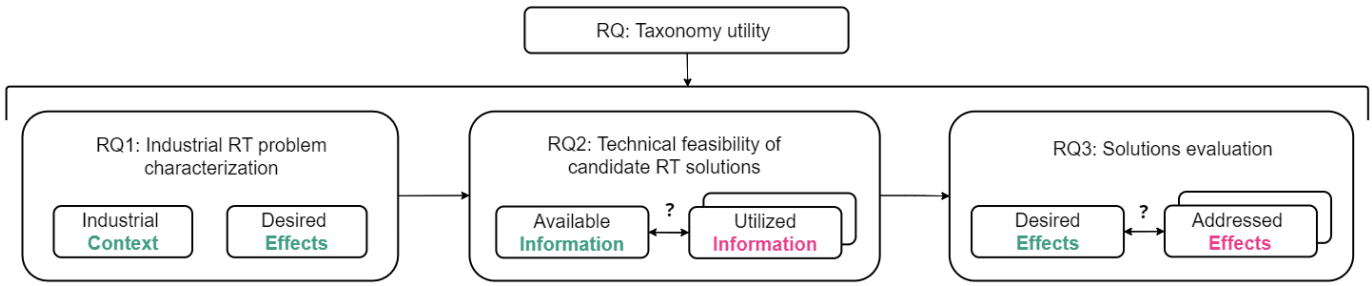

Figure 1: Research structure.

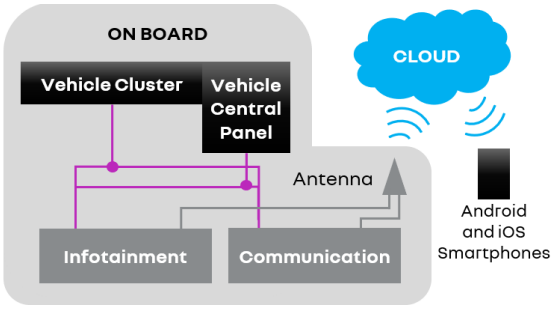

Figure 2: Global view of Project A.

Table 3. Context of Project A using taxonomy of [2]

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

4. RT Problem Characterization

Step 1 of the study 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 infotainment system and the communication one. 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 external services. A cloud platform is used to store vehicle information and to communicate with other external systems. An example of external system considered in Project A is a smartphone application to lock/unlock the doors of the car.

The case study focuses on the integration tests of the on-board software. Table 3 summarizes the context factors based on the taxonomy. Those factors are high-level. The user is expected to put a short note why a given factor is challenging (e.g., "multi-language" put for the complexity factor). We provide below a more extensive description.

Each system is multi-language. The infotainment is mainly in *C*, *C++* and *Java*, while the communication is mainly in *C*. Their development involves different companies. There is one supplier for infotainment, one for communication and Renault integrates both parts. We consider the tests done at Renault. They run on a test bench that reproduces the on-board architecture and manages the access to the external services. The binary code of the software under test (SUT) is flashed on the bench to update the ECUs. The size of the flashed code is in the scale of 6 GB. The integration team first tests the infotainment and communication separately, and then their integration. A new binary code version is received weekly from the suppliers.

There are two types of tests: automatic and manual. Automatic tests run during the night and manual ones during the working hours. All tests have a priority ($P1$, $P2$, or $P3$) determined by the importance of the functional feature they check. When a Fail is found, the tester conducts an analysis to determine the origin of the problem: infotainment, communication or test automation issue. Accordingly, the issue report is passed to product or test automation managers, who assign a criticality and priority to the issue. The criticality is defined by the impact that the bug has on the client, with $K1$ corresponding to the strongest impact, and $K4$ to the weakest one. The priority determines the order in which the suppliers or test automation team should fix the issues, using four levels of priority.

Test automation issues represent about 30% of the Fails of automatic 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 [58]. They reported from 74% up to 91% of Fails due to automation issues (unreliable

test infrastructures, incorrect tests). In Project A, the test automation calls for a dedicated development and evolution process. All automatic 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 a 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 – automatic or manual – is typically in tens of minutes (from 20 to 50 minutes). Running all tests would take 7 nights and 9 working days, which would not fit into a week. This time does not account for the analysis of the tests, which must also be done during the week. The practice is to have 2 nights and a half for running the automatic 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. The manual tester immediately performs the analysis when observing a Fail, before going to the next test. The number of executed tests thus depends on the Fails 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 selected. As a result, some tests are not executed for months.

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 shown in Table 4. We discuss below the ones that are the most desirable for Project A.

Table 4. Desired effects for Project A using the taxonomy

Desired effects		
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	

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 automatic 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 tests, and from two and a half nights to one night for automatic tests. However, the mere reduction in time would not be enough. It would be imperative to maintain the *fault detection capability*. If such is not the case, the reduced testing time should at least maintain, if not improve, the *severe fault detection* capability. As mentioned in the description of the context, each issue has a criticality. So, it may be acceptable to find less faults if the most critical ones are found.

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 automatic 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 (automatic tests). For automatic tests, a fine-grained ordering may not be needed, provided that the failing tests occur in the first night.

[1] https://robotframework.org/

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 discussion of desired effects revealed a missing one: the circulation of the tests. The taxonomy does not include it. Still, at least one paper analyzed by the authors of the taxonomy proposes a method that ensures the circulation of the tests ($S26$). One of the ML-based papers we analyzed ($L9$) also do. 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 that mentions the circulation of tests, and that explicitly demonstrates this effect (along with other ones) [59]. We believe that forgotten tests are a recurring problem in many industrial settings, and propose to add their avoidance in 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 the effect part proved very helpful. A missing effect is the circulation of the tests.

5. Analysis of RT Methods Feasibility

Step 2 of the study 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 by 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 that 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.

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.

1) Test execution time appearing at different places. In [2], the authors 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 automatic 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 different from a Fail verdict? Is 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 Fails 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 fails. Some methods do not consider the raw fails, and need 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. The direction of the link 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.

Table 5. Information for Project A using the taxonomy of [2] and mapping of the set of papers

Information		Project A	State of the Art Papers
Requeriments	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, L2, 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, L9, 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, L7, L8, L9, L10, 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, L13, L14
	Failure severity	X	S3, S4, L12, L14

Table 6. Technically feasible methods according to the taxonomy or a manual labeling (differences in bold)

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

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 fails, issues found by the tests), the tested requirements and their priority. All these items are available in Project A. Most of the eliminated paper require design or code artefacts 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 papers, and labeled them as feasible or not in the context of Project A. 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.

While reading the papers, we had several disagreements with the mapping proposed for the papers Si in [2]. 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.

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 a bunch of papers to represent these methods, we could get new information items. Such was the case for $L12$. It applies NLP techniques and require a natural language description of test cases. There is no item for this in the taxonomy. The paper was labeled as feasible, but actually 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, however $L14$ is unfeasible due to another reason (type of test). Although it does not cause any problem for our classification, it may yield a misclassification in other contexts where the information is not available.

Table 7. Disagreements with the mapping of papers in [2]

ID	Disagreements
S3	Same mapping as S4 (same authors), but actually uses less information
S9-S12	Needs the source code of user functions, as well as links from test cases to the 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 artefacts/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

5.3. Feedback on the taxonomy (RQ2)

The availability of information was found an effective criterion to separate 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 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,

Table 8. Addressed Effects

Addressed effects	Method
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
Severe fault detection	S3
Circulation of the tests	S26, L9

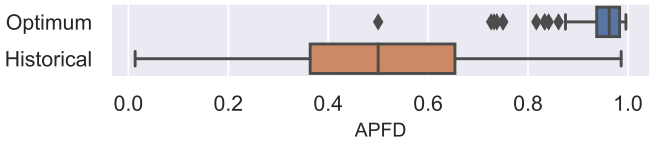

Figure 3: APFD for optimal and historical orders of manual tests.

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 artefacts 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 difficult to interpret, and several papers are inaccurately mapped to the taxonomy. A documentation and maintenance effort would be needed to make the taxonomy more usable.

6. Evaluation of RT Methods

Step 3 of the study focuses on the evaluation of the candidate methods. Table 8 shows the alignment between the desired effects (from Step 1) and the ones 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. All the methods have at least one effect in common with Project A, and are thus potentially relevant. However, their relevance cannot be decided based on published results. They must be experimented in the industrial context. To prepare the implementation of experiments, we must gain closer insights into the measurement of effects. For this, we had a look at all papers (feasible or not) mapped to each desired effect.

6.1. 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: APFD (Average Percentage of Faults Detected) and its variant NAPFD (Normalized APFD). APFD was first introduced in [60]. It measures 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 [61] addresses cases where not all tests are run and thus only a certain percentage of faults (or fails) are detected compared to the full suite. Note that neither APFD nor NAPFD is well-suited when there are few faults. As an alternative, we found that some authors focus on the first detection only. The metric can be relative to the number of tests (as in $L1$) or to test execution time (as in $L9$).

In project A, the number of revealed faults is quite different for automatic and manual tests.
The nightly automatic tests check mature functionalities that rarely fail. Most of the weeks, all executed tests pass. When there are Fails, they are linked to few issues (max 3 in a week). Hence, an APFD-like metric is not adequate. Moreover, since the test results are analyzed during working hours, the precise ordering of the tests during the night is not so useful. From historical data, the first fault (if any) most often surfaces in the first night. In the cases with 2 or 3 faults, there are detections after the first night. Accordingly, appropriate metrics could be the times to first and last detection with the granularity of nights. A candidate method should not degrade the time to first detection, and should make the last detection occur earlier.
In contrast, the manual tests reveal more faults and a fine-grained prioritization is useful. A classical APFD metric is possible. Figure 3 shows its application to historical data. Although a subset of tests were executed each week, NAPFD would not be appropriate to analyse the history of Project A. Indeed, we do not know the percentage of faults compared to the full test suite. Rather, we have to consider that the full test suite of a week is exactly the set of tests that ran during this week, and the faults are only the ones revealed by these tests. Using this interpretation, the historical APFD measures the adequacy of the execution order chosen by the tester, given the selection of the week. In Figure 3, we also give the APFD for an (hypothetical) optimal order of tests, where the revealing ones are always put first. The comparison between the historical and optimal order shows that there is room for improvement.
We thus retain APFD for manual tests, and the time to first and last detection for automatic tests (measured with the granularity of a night). The metrics will allow us to evaluate the 7 feasible methods that address the effect (see Table 8).

6.2. Circulation of the tests

For the *circulation of the tests*, the taxonomy does not help. No paper from the set measures this effect, although some address it ($S26$, $L9$). Outside the set of papers, we identified relevant work in [59]: the authors check that no test is forgotten in three months of execution. This suggests a measurement of the worst case time between two executions of a test.
We applied this metric to historical data over a period of one year. The worst case is more than 3 months for manual tests, and 6 months for automatic ones. If one considers the time to run the complete suite (respectively 9 working days and 7 nights), there is room for improvement. Each week uses 2.5 days and nights for running the tests, so a complete cycle

would last less that one month. This shows that it is feasible to decrease the worst case.

The candidate methods are obviously $S26$ and $L9$. We also consider augmenting the other methods with a circulation effect. A method will then fill only $x\%$ of the test time budget, the remaining $(100 - x)\%$ being filled by tests according to their seniority of execution. To allow an evaluation on historical data, we will consider that the tests ran during the first $x\%$ must be a subset of the tests that really ran during the project, possibly ordered differently. This will allow us to explore tradeoffs between the circulation of the tests and the decreased time for fault detection supplied by the original methods.

6.3. Long-term objectives

The long-term objectives in Project A are to reduce the testing time while maintaining the (severe) fault detection capability. The testing time per week would ideally go down to just one night for automatic tests and one working day for manual tests. We analyzed the historical data to know whether this time would be sufficient to run the tests that reveal the faults. For automatic tests, it is possible to catch all faults in one night. However, for manual tests, it is not always possible to reveal all faults in just one day, as there are more faults and thus more tests to run. If just the severe faults are considered -– the ones having the impact $K1$ or $K2$ (cf. Section 4.1) –, the execution time will be less then one day, so they all could be revealed in the first day. Hence, reaching the objectives may be hard but not impossible.

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). For manual tests, the recall is the most adequate metric, since we would like not to miss faults. For automatic tests, as there are few faults, the recall is less appropriate. We will rather use the absolute number of detected faults.

We also checked the papers for *severe fault detection capability*. In $S3$ and $S4$, the APFDc metric is used. Introduced in [62], this metric considers that test cases do not have the same cost in terms of running time or fault severity. Like APFD, it requires the full suite to be executed, and is thus not relevant to methods aiming at test time reduction. In $S21$, a new metric is introduced: it computes an average severity of faults detected (ASFD) for each requirement. This is not our objective, so it is not relevant. We eventually decided to use NAPFD and recall for manual tests, restricting the measurements to the set of severe faults. In this way, we evaluate whether the reduced suite (one day) misses any severe fault of the full suite (2.5 days), and whether the most critical revealing tests occur early. For automatic tests, we will simply use the absolute number of severe faults detected in the first night.

6.4. Feedback on the taxonomy (RQ3)

The taxonomy aided in finding metrics to assess the desired effects. The mapping relevantly pointed to papers that measure these effects. We could gain an overview of alternative metrics, choose the ones we will use and evaluate the margins for improvement in the industrial context. Still, there may be some reservation with respect to the granularity of the taxonomy. The effect part is much less detailed than the information one, which had a fine-grained list of items utilized by the methods. Similarly, the effects could be explicitly refined into metrics. This would have saved time in the identification of metrics, allowing us to concentrate on the preparation of experiments.

> **RQ3 (aid in the evaluation of solutions)**: The taxonomy was helpful to decide how to measure the desired improvements. A refined list of metrics would make the taxonomy even more helpful.

7. Threats to validity

Concerning the threats to internal validity, we can mention the 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 double-checked the mapping. Moreover, we explicitly reported the difficulties we encountered and explained the related mapping decisions. We also double checked the disagreements we had on the mapping of the original set of papers. Another threat is the choice of the articles to represent ML-based methods. 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. A last internal threat 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.

This case study applies the taxonomy to a specific industrial context, which may represent a threat to external validity. Each practitioner has a very unique RT process. However, the 3-step methodology we adopted is generic and intended to serve in any context. The difficulties we reported also do not seem specific to Project A. Hence, we believe that our experience may benefit other researchers and practitioners.

8. Conclusion

This study contributes to a line of research on how to asses industrial relevance and applicability of RT methods. It presents a real-world scenario in which practitioners search for solutions to their specific problem. The taxonomy proposed in [2] 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, forming a principled base from which we derived a three-step approach to tackle the real world scenario. The first step characterizes the problem independently from any solution. The second one studies the feasibility of solutions based on the alignment between the required and available information items. The third step studies the alignment between the desired effects of solutions and the effects addressed by the methods. It also prepares the experimental measurement of these effects in the industrial context. Applied to the Renault scenario, this approach successfully identified a set of eight promising RT methods, together with a set of metrics to evaluate them. Using the metrics, we also studied the margins for improvement in the current RT process.

While the principle of considering context, effects and information factors proved successful, the lower-level content of the taxonomy was not as useful as it could have been. The use of the taxonomy was hindered by hard-to-interpret elements, missing elements or insufficiently detailed ones, and errors in the classification of candidate methods. Our experience thus shows that the taxonomy needs improvement. A means to manage the improvement would be to deploy and maintain a public repository. Users would find updated versions of the taxonomy, a documentation, a database of papers mapped to the taxonomy, and could contribute by feedback. The version published in [2] may not be considered as a finished product. Regarding the real-world scenario, the study continues with the experimentation of the methods. Four methods have already been implemented. They are being applied to the historical project data, confirming the relevance of some.

Acknowledgment

The authors would like to thank the teams at Renault Software Factory that work on Project A for their help.

References

[1] S. Yoo and M. Harman, "Regression testing minimization, selection and prioritization: A survey," *Softw. Test. Verif. Reliab.*, vol. 22, no. 2, pp. 67–120, Mar. 2012.

[2] N. B. Ali, E. Engström, M. Taromirad, M. R. Mousavi, N. M. Minhas, D. Helgesson, S. Kunze, and M. Varshosaz, "On the search for industry-relevant regression testing research," *Empirical Software Engineering*, vol. 24, no. 4, pp. 2020–2055, 2019.

[3] M. Khatibsyarbini, M. A. Isa, D. N. Jawawi, and R. Tumeng, "Test case prioritization approaches in regression testing: A systematic literature review," *Information and Software Technology*, vol. 93, pp. 74–93, 2018.

[4] J. A. Prado Lima and S. R. Vergilio, "Test case prioritization in continuous integration environments: A systematic mapping study," *Information and Software Technology*, vol. 121, p. 106268, 2020.

[5] R. Pan, M. Bagherzadeh, T. A. Ghaleb, and L. Briand, "Test case selection and prioritization using machine learning: a systematic literature review," *Empirical Software Engineering*, vol. 27, no. 2, 2022.

[6] D. Marijan, A. Gotlieb, and M. Liaaen, "A learning algorithm for optimizing continuous integration development and testing practice," *Softw. Pract. Exp.*, vol. 49, no. 2, pp. 192–213, 2019.

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

[8] E. D. Ekelund and E. Engström, "Efficient regression testing based on test history: An industrial evaluation," in *2015 IEEE Int. Conf. on Software Maintenance and Evolution (ICSME)*, 2015, pp. 449–457.

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

[10] R. Krishnamoorthi and S. Sahaaya Arul Mary, "Factor oriented requirement coverage based system test case prioritization of new and regression test cases," *Inf. Softw. Technol.*, vol. 51, no. 4, pp. 799–808, 2009.

[11] A. Pasala and A. Bhowmick, "An approach for test suite selection to validate applications on deployment of cots upgrades," in *12th Asia-Pacific Software Engineering Conference (APSEC'05)*, 2005, pp. 361–364.

[12] R. K. Saha, L. Zhang, S. Khurshid, and D. E. Perry, "An information retrieval approach for regression test prioritization based on program changes," in *37th IEEE/ACM Int. Conf. on Software Engineering (ICSE)*, vol. 1, 2015, pp. 268–279.

[13] J. Zheng, B. Robinson, L. Williams, and K. Smiley, "Applying regression test selection for cots-based applications," in *28th Int. Conf. on Software Engineering (ICSE '06)*, 2006, pp. 512–522.

[14] S. Tahvili, W. Afzal, M. Saadatmand, M. Bohlin, D. Sundmark, and S. Larsson, "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)*, 2016, pp. 745–759.

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

[16] D. Marijan, A. Gotlieb, and S. Sen, "Test case prioritization for continuous regression testing: An industrial case study," in *2013 IEEE Int. Conf. on Software Maintenance (ICSM)*, 2013, pp. 540–543.

[17] S. Wang, A. Gotlieb, S. Ali, and M. Liaaen, "Automated test case selection using feature model: An industrial case study," in *16th Int. Conf. on Model-Driven Engineering Languages and Systems (MODELS)*, 2013, pp. 237–253.

[18] M. U. Janjua, "Onspot system: Test impact visibility during code edits in real software," in *2015 10th Joint Meeting on Foundations of Software Engineering (ESEC/FSE)*, 2015, pp. 994–997.

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

[20] D. Marijan, "Multi-perspective regression test prioritization for time-constrained environments," in *2015 IEEE Int. Conf. on Software Quality, Reliability and Security (QRS)*, 2015, pp. 157–162.

[21] S. Wang, S. Ali, A. Gotlieb, and M. Liaaen, "Automated product line test case selection: industrial case study and controlled experiment," *Softw. Syst. Model.*, vol. 16, no. 2, pp. 417–441, 2017.

[22] E. Engström, P. Runeson, and G. Wikstrand, "An empirical evaluation of regression testing based on fix-cache recommendations," in *3rd Int. Conf. on Software Testing, Verification and Validation (ICST)*, 2010, pp. 75–78.

[23] J. Anderson, S. Salem, and H. Do, "Improving the effectiveness of test suite through mining historical data," in *11th Working Conf. on Mining Software Repositories (MSR)*, 2014, p. 142–151.

[24] G. Buchgeher, C. Ernstbrunner, R. Ramler, and M. Lusser, "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)*, 2013, pp. 74–79.

[25] S. Wang, S. Ali, and A. Gotlieb, "Cost-effective test suite minimization in product lines using search techniques," *J. Syst. Softw.*, vol. 103, pp. 370–391, 2015.

[26] G. Wikstrand, R. Feldt, J. K. Gorantla, W. Zhe, and C. White, "Dynamic regression test selection based on a file cache an industrial evaluation," in *2nd Int. Conf. on Software Testing Verification and Validation (ICST)*, 2009, pp. 299–302.

[27] M. Lochau, S. Lity, R. Lachmann, I. Schaefer, and U. Goltz, "Delta-oriented model-based integration testing of large-scale systems," *Journal of Systems and Software*, vol. 91, pp. 63–84, 2014.

[28] M. Skoglund and P. Runeson, "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)*, 2005, pp. 74–83.

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

[30] E. Engström, P. Runeson, and A. Ljung, "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, 2011.

[31] P. Devaki, S. Thummalapenta, N. Singhania, and S. Sinha, "Efficient and flexible gui test execution via test merging," in *Int. Symp. on Software Testing and Analysis (ISSTA)*, 2013, pp. 34–44.

[32] L. White, K. Jaber, B. Robinson, and V. Rajlich, "Extended firewall for regression testing: an experience report," *Journal of Software Maintenance and Evolution: Research and Practice*, vol. 20, no. 6, pp. 419–433, 2008.

[33] S. Wang, S. Ali, and A. Gotlieb, "Minimizing test suites in software product lines using weight-based genetic algorithms," in *15th Annual Conf. on Genetic and Evolutionary Computation (GECCO)*, 2013, pp. 1493–1500.

[34] S. Vöst and S. Wagner, "Trace-based test selection to support continuous integration in the automotive industry," in *Int. Workshop on Continuous Software Evolution and Delivery (CSED@ICSE)*, 2016, pp. 34–40.

[35] R. Carlson, H. Do, and A. Denton, "A clustering approach to improving test case prioritization: An industrial case study," in *27th IEEE Int. Conf. on Software Maintenance (ICSM)*, 2011, pp. 382–391.

[36] L. White and B. Robinson, "Industrial real-time regression testing and analysis using firewalls," in *20th IEEE Int. Conf. on Software Maintenance (ICSM)*, 2004, pp. 18–27.

[37] S. Wang, D. Buchmann, S. Ali, A. Gotlieb, D. Pradhan, and M. Liaaen, "Multi-objective test prioritization in software product line testing: An industrial case study," in *18th Int. Software Product Line Conference (SPLC)*, 2014, pp. 32–41.

[38] S. Huang, Y. Chen, J. Zhu, Z. J. Li, and H. F. Tan, "An optimized change-driven regression testing selection strategy for binary java applications," in *2009 ACM Symp. on Applied Computing (SAC)*, 2009, pp. 558–565.

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

[40] J. Zheng, B. Robinson, L. Williams, and K. Smiley, "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)*, 2006, pp. 137–143.

[41] E. Rogstad and L. Briand, "Cost-effective strategies for the regression testing of database applications: Case study and lessons learned," *Journal of Systems and Software*, vol. 113, pp. 257–274, 2016.

[42] A. Srivastava and J. Thiagarajan, "Effectively prioritizing tests in development environment," in *2002 ACM SIGSOFT Int. Symp. on Software Testing and Analysis (ISSTA)*, 2002, pp. 97–106.

[43] J. Zheng, L. Williams, and B. Robinson, "Pallino: Au-

tomation to support regression test selection for cots-based applications," in *22nd IEEE/ACM Int. Conf. on Automated Software Engineering (ASE)*, 2007, pp. 224–233.

[44] E. Rogstad, L. Briand, and R. Torkar, "Test case selection for black-box regression testing of database applications," *Information and Software Technology*, vol. 55, no. 10, pp. 1781–1795, 2013.

[45] M. Hirzel and H. Klaeren, "Graph-walk-based selective regression testing of web applications created with google web toolkit," in *Software Engineering (workshops)*, 2016.

[46] F. Palma, T. Abdou, A. Bener, J. Maidens, and S. Liu, "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)*, 2018, pp. 80–89.

[47] T. Shi, L. Xiao, and K. Wu, "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)*, 2020, pp. 663–672.

[48] J. A. P. Lima and S. R. Vergilio, "A multi-armed bandit approach for test case prioritization in continuous integration environments," *IEEE Transactions on Software Engineering*, vol. 48, no. 2, pp. 453–465, 2022.

[49] M. Mahdieh, S.-H. Mirian-Hosseinabadi, K. Etemadi, A. Nosrati, and S. Jalali, "Incorporating fault-proneness estimations into coverage-based test case prioritization methods," *Information and Software Technology*, vol. 121, p. 106269, 2020.

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

[51] B. Busjaeger and T. Xie, "Learning for test prioritization: an industrial case study," in *24th ACM SIGSOFT Int. Symp. on Foundations of Software Engineering (FSE)*, 2016, pp. 975–980.

[52] S. Yoo, M. Harman, P. Tonella, and A. Susi, "Clustering test cases to achieve effective & scalable prioritisation incorporating expert knowledge," in *8th Int. Symp. on Software Testing and Analysis (ISSTA)*, 2009, pp. 201–212.

[53] R. Lachmann, S. Schulze, M. Nieke, C. Seidl, and I. Schaefer, "System-level test case prioritization using machine learning," in *15th IEEE Int. Conf. on Machine Learning and Applications (ICMLA)*, 2016, pp. 361–368.

[54] S. Chen, Z. Chen, Z. Zhao, B. Xu, and Y. Feng, "Using semi-supervised clustering to improve regression test selection techniques," in *4th IEEE Int. Conf. on Software Testing, Verification and Validation (ICST)*, 2011, pp. 1–10.

[55] N. Medhat, S. M. Moussa, N. L. Badr, and M. F. Tolba, "A framework for continuous regression and integration testing in iot systems based on deep learning and search-based techniques," *IEEE Access*, vol. 8, pp. 215 716–215 726, 2020.

[56] H. Spieker, A. Gotlieb, D. Marijan, and M. Mossige, "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, 2017.

[57] P. Kandil, S. Moussa, and N. Badr, "Cluster-based test cases prioritization and selection technique for agile regression testing," *Journal of Software: Evolution and Process*, vol. 29, no. 6, 2016.

[58] C. Jordan, P. Foth, A. Pretschner, and M. Fruth, "Unreliable test infrastructures in automotive testing setups," in *2022 IEEE/ACM 44th Int. Conf. on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, 2022, pp. 307–308.

[59] P. Erik Strandberg, W. Afzal, T. J. Ostrand, E. J. Weyuker, and D. Sundmark, "Automated system-level regression test prioritization in a nutshell," *IEEE Software*, vol. 34, no. 4, pp. 30–37, 2017.

[60] G. Rothermel, R. Untch, C. Chu, and M. Harrold, "Test case prioritization: an empirical study," in *IEEE Int. Conf. on Software Maintenance (ICSM)*, 1999, pp. 179–188.

[61] X. Qu, M. B. Cohen, and K. M. Woolf, "Combinatorial interaction regression testing: A study of test case generation and prioritization," in *IEEE Int. Conf. on Software Maintenance (ICSM)*, 2007, pp. 255–264.

[62] S. Elbaum, A. Malishevsky, and G. Rothermel, "Incorporating varying test costs and fault severities into test case prioritization," in *23rd Int. Conf. on Software Engineering (ICSE)*, 2001, pp. 329–338.