The Journal of Systems and Software 223 (2025) 112360

Contents lists available at ScienceDirect

The Journal of Systems & Software

journal homepage: www.elsevier.com/locate/jss

Predicting test failures induced by software defects: A lightweight alternative to software defect prediction and its industrial application ☆

Lech Madeyski [a,*], Szymon Stradowski [a,b]

[a] Wrocław University of Science and Technology, Wyb. Wyspianskiego 27, Wrocław, 50-370, Dolnoslaskie, Poland
[b] Nokia, Szybowcowa 2, Wrocław, 54-206, Dolnoslaskie, Poland

ARTICLE INFO

Dataset link: https://doi.org/10.6084/m9.figshare.28263290

Keywords:
Software engineering
Software testing
Machine learning
Software defect prediction
Industry application
Reproducible research
Open science

ABSTRACT

Context: Machine Learning Software Defect Prediction (ML SDP) is a promising method to improve the quality and minimise the cost of software development.
Objective: We aim to: (1) apropose and develop a Lightweight Alternative to SDP (LA2SDP) that predicts test failures induced by software defects to allow pinpointing defective software modules thanks to available mapping of predicted test failures to past defects and corrected modules, (2) preliminary evaluate the proposed method in a real-world Nokia 5G scenario.
Method: We train machine learning models using test failures that come from confirmed software defects already available in the Nokia 5G environment. We implement LA2SDP using five supervised ML algorithms, together with their tuned versions, and use eXplainable AI (XAI) to provide feedback to stakeholders and initiate quality improvement actions.
Results: We have shown that LA2SDP is feasible in vivo using test failure-to-defect report mapping readily available within the Nokia 5G system-level test process, achieving good predictive performance. Specifically, CatBoost Gradient Boosting turned out to perform the best and achieved satisfactory Matthew's Correlation Coefficient (MCC) results for our feasibility study.
Conclusions: Our efforts have successfully defined, developed, and validated LA2SDP, using the sliding and expanding window approaches on an industrial data set.

1. Introduction

Machine learning software defect prediction (ML SDP or SDP for short) is a highly prospective field of software engineering (SE). Its goal is to employ specific algorithms to analyse the product (e.g., a portion of software product metrics) or process (e.g., via process metrics) to estimate the number of defects in particular areas of the code (regression) or whether defects exist in those areas (classification) (Madeyski and Jureczko, 2015). The business potential of such solutions is very high (see, e.g., Hryszko and Madeyski, 2017, 2018). Still, the pace of industrial application lags behind academic research in the field, and there are very few publications on results obtained in vivo (Stradowski and Madeyski, 2023e,d).

Industrial applications of SDP are rarely reported due to a variety of reasons. For example, there is a lack of incentives for the industry to share real-world SDP experience, know-how or intellectual property. Furthermore, several researchers reported serious weaknesses in the SZZ (Śliwerski, Zimmermann, Zeller) algorithm (Śliwerski et al., 2005) that is typically used in the SDP implementations (Herbold et al., 2022; Rosa et al., 2023). Namely, Herbold et al. (2022) reported that only half of the bug-fixing commits determined by SZZ are actually bug-fixing (see further details in Section 2.2). This is a consequential problem when the SZZ algorithm is the core component of an in vivo SDP application. Last but not least, there is a need for extensive mining of software repositories to collect data just for the sake of SDP, which can be uneconomical from the company's perspective.

Thus, in this paper, we present and evaluate in vivo (in Nokia's system-level 5G test process environment) LA2SDP, a lightweight alternative to ML SDP. Notably, this work was inspired by the conclusions of a survey conducted by Stradowski and Madeyski (2023c), highlighting considerable opportunities to improve the quality and minimise the cost of software testing within the company. Furthermore, a study by Paterson et al. (2019) states that defect prediction can accurately identify the modules that are most likely to be buggy.

☆ Editor: Daniel Mendez.

* Corresponding author.
E-mail address: lech.madeyski@pwr.edu.pl (L. Madeyski).

https://doi.org/10.1016/j.jss.2025.112360
Received 30 April 2024; Received in revised form 29 December 2024; Accepted 22 January 2025
Available online 3 February 2025

In contrast, we aim to predict test cases that will detect software defects in particular modules to trigger post-analysis and potentially omit the costs related to retesting in expensive environments. Importantly, we have not encountered a similar approach that would study the possibility of analysing high-level test results from a test repository to be used for SDP instead of prioritising test cases (Catal and Mishra, 2013; Pan et al., 2022). Therefore, we are merging the aspects of test case selection and prioritisation (TSP) with software defect prediction (SDP) to open new avenues in software engineering research, address direct company expectations, as well as extend industry applications.

The presented research is part of a larger, business-driven effort to gather the challenges (Stradowski and Madeyski, 2023c,b), analyse existing methods (Stradowski and Madeyski, 2023e,d), and now develop a dedicated solution that satisfies the business requirements of Nokia. The most important contributions of this industrial study are highlighted below:

- Use case design of a software defect prediction solution that can work specifically with system-level test process data to complement other defect prediction and test case selection and prioritisation mechanisms within the company.
- Proposal of a lightweight alternative to Software Defect Prediction (LA2SDP) to satisfy the expectations (see Section 2.1).
- Industrial data suited for LA2SDP and benchmarked prediction models with code included in the reproduction package (Madeyski and Stradowski, 2024).
- Feature importance analysis to support interpreting and communicating the models to stakeholders and initiating improvement actions.
- Evaluation and discussion of the obtained results, practitioners' feedback, and key learnings.

In Section 6, we describe the background of our research and highlight its main contributions. In Section 2, we set the business context and describe the Nokia 5G test process. Next, Section 3 explains the methods used and the models that were built, followed by Section 4 that contains the analysis of the obtained results. Finally, in Section 5 to Section 8, we present a discussion, feedback, related work, and derived conclusions.

2. Project context

The main motivation for proposing a new approach is the opportunity to use the available test repository data to implement an additional software defect prediction mechanism in the Nokia 5G quality assurance process. Our research on the profitability of ML SDP shows that our solution can bring significant operational savings to the company (return on investment (Stradowski and Madeyski, 2024)). Furthermore, Nokia practitioners see it as one of the less-explored possibilities within the 5G development process (Stradowski and Madeyski, 2023c). Hence, we proposed LA2SDP, which is relatively easy to build, uses data already existing within the company, and can have a significant positive business impact.

The challenges faced in the system-level testing of the 5G base station (called gNB or gNodeB (The 3rd Generation Partnership Project, 2021)) at Nokia have been explored in the survey conducted by Stradowski and Madeyski (2023c). Specifically, developing the 5G technology carries a considerable challenge. The difficulty arises from several factors, such as complex propagation characteristics needed for performance optimisation, wide frequency spectrum, hugely expensive over-the-air (OTA) test environments, complex verification of multiple-input multiple-output (MIMO) performance, defined by strict 3GPP specification requirements (The 3rd Generation Partnership Project, 2021).

Consequently, the Nokia 5G gNB system is a grand and complex project, with over 60 million lines of code (LOC) written in C/C++ programming language. The gNB consists of tens of components and hundreds of modules developed by several major development organisations, each with hundreds of thousands of dedicated unit and integration tests. In later phases, the central software build with all integrated deliveries is undergoing continuous testing in several globally distributed teams. Each team has its own set of requirements to validate the software against, as well as a dedicated laboratory infrastructure of varying complexity and associated maintenance costs. The test infrastructure can consist of servers with simulators and stubs, real Nokia gNBs in isolated setups, or massive anechoic chambers with multiple gNBs to measure signal interference and mobility scenarios. In our implementation project, LA2SDP is expected to support the existing effort in the system-level phase, being the last one of many test phases within the company (Stradowski and Madeyski, 2023b), as it is the most expensive to operate and thus provides the most extensive opportunities for savings.

The Nokia test process follows the state-of-the-art and adheres to accepted standards, briefly introduced below. The company uses continuous delivery (CD) and continuous integration (CI) to build its products, emphasising left-shift principles[1] and strict phase containment[2] criteria. However, due to the complexity and size of the product, current phase containment rates are not satisfactory. Therefore, further quality improvement initiatives are necessary, and any advancement upon the current baseline brings consequential benefits as each escaped defect that was not detected in internal testing costs the company tens of thousands of dollars (Stradowski and Madeyski, 2023c, 2024). The main three phases of the Nokia test process are described below and depicted in Fig. 1.

- First, unit-level tests (UT) are under the responsibility of the development units (DU). Execution within the development units mainly happens in simulated environments and isolated hardware elements that integrate and validate basic functionalities before merging to the central software build.
- Second, entity testing (ET) is executed, after which an automated mechanism of software promotion builds central build packages where elements from all development units are integrated, and the whole system is made operational. This level is executed on a complete and real gNB system, allowing validation of its core functionalities.
- Last, a wide plethora of system-level testing (ST) such as continuous integration (CIT), continuous regression (CRT), and continuous delivery regression tests (CDRT) are executed. They contain a wide scope from new feature verification, stability, configurability, through security, to performance and capacity benchmarking.

For the purpose of our case study, together with a group of five experienced test architects, we have run a feasibility study on the entire 5G gNB system-level environment (see Fig. 1) in which the designed LA2SDP solution can be validated. System-level testing includes regression, new features, and benchmark test case categories.

- Regression: By far the most numerous group of test cases, the purpose of which is to confirm that the legacy functionality works as it was designed and has not been broken by new code. As in many software companies (Garousi et al., 2018), the ever-increasing scope of maintenance testing in Nokia needs to be carefully managed.

[1] An approach where software testing is performed to find defects as early as possible in the life cycle, where they are cheaper to find and fix than the later stages (International Organization for Standardization, 2022; Damm et al., 2006).

[2] Defect phase containment measures how many defects were caught before they escape into later phases (ISTQB, 2023a).

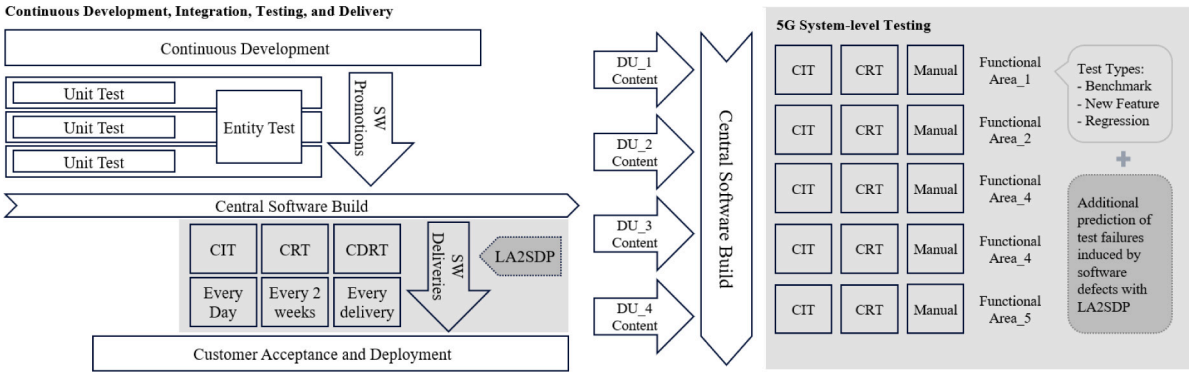


Fig. 1. Nokia test process (Stradowski and Madeyski, 2023c), with visualisation of LA2SDP addition.

- New Feature: All new content and resulting new functionalities must be tested to ensure they work as designed by the requirements. Such cases may range from a simple system reconfiguration to complex field verification scenarios requiring a close-to-real 5G network environment.
- Benchmark: As system performance is business critical for commercial wireless networks, extensive benchmarking needs to be executed to ensure that the added content increases the performance of the whole system. Such tests are challenging to execute, evaluate, and troubleshoot. Also, they usually take extended periods of time, lasting from one hour to several days, to thoroughly verify system stability. Hence, huge gains can be obtained by avoiding unnecessary tests with LA2SDP.

2.1. Expectations

For a successful implementation of a business-driven project, there needs to be a clear definition of the expectations that need to be satisfied (IIBA, 2015; Stradowski and Madeyski, 2023a). Hence, together with a group of five experienced test architects and quality managers from Nokia who participated in the project (for further information see Section 5.5), we defined the following set of high-level expectations for our feasibility study:

E1: The company wants to have a solution that is comparable to the performance already reported by other researchers. Shepperd et al. (2014) found in their thorough analysis of the ML SDP literature that the mean and median values of Matthew's correlation coefficient (MCC) are 0.305 and 0.308, respectively. Hence, we set our target for MCC > 0.3 for this phase of our feasibility study. Auxiliary performance measures are also expected to be reported for convenience, transparency, and further literature comparison.

E2: The company expects the LA2SDP solution to be as cost-efficient as possible and thus use already existing data and generally follow the KISS principle (Stellman and Greene, 2014), which helps to keep the code, design, tests, running times, and documentation fast and lean.

E3: We want the ML models behind LA2SDP to be taken from the pool of models that support interpretability, as it is necessary for further steps of the ML SDP project (Stradowski and Madeyski, 2023a, 2025).

Importantly, we do not aspire to fully satisfy all expectations in this phase of research. Current efforts are aimed mainly at feasibility and verification of **E1** (for prediction effectiveness study see Section 3). Second, this research also provides baseline data for a dedicated study related to expectation **E2** (analysing cost-efficiency Stradowski and Madeyski, 2024) and enables possibilities for **E3** (interpretability of created models). However, at this project phase, it is considered enough to prefer the ML models supporting **E3** (e.g., Aria et al., 2021; Konstantinov and Utkin, 2021; Molnar, 2023).

2.2. LA2SDP use case

The primary purpose of software testing is to discover defects and evaluate the quality of software artefacts. Testing can trigger failures that are caused by defects in the software and directly find defects in the test object but also result from environmental and process issues (ISTQB, 2023b). Accordingly, we propose an alternative to established software defect prediction approaches, such as based on software product or change metrics (Stradowski and Madeyski, 2023d; Madeyski and Jureczko, 2015), by directly predicting test failures induced by software defects using historical test repository data (see Fig. 2).

Our system-level use case in Nokia complements an existing testing process within the company (Fig. 1) where there are vast amounts of historical data that can be used in different ways. Specifically, we can filter the test run results to steer the modelling process towards different outcomes. For example, we can teach the ML algorithms exclusively on failures with a confirmed cause identified, being software defects or environmental issues. Teaching the models on such subsets allows for predicting only test failures induced by software defects, which allows us to formulate LA2SDP. Alternatively, test run failures confirmed as environmental issues can be used to improve the test process itself, which allows us to formulate a complementary (to LA2SDP) method that we named the Lightweight Alternative to Test Process Improvement (LA2TPI).

Also, we can use the entire set to predict test case failure in general, i.e. Test Failure Prediction (TFP). That said, our current research effort focuses on the first LA2SDP option, predicting test case failures due to software defects, as this is the main requirement for the improvement proposal (see Section 2.1). Moreover, it also opens other research paths towards more effort-aware defect prediction (Mende and Koschke, 2010; Jing et al., 2024), as the number of predicted failures (and corresponding requirements) can help distinguish between modules with high and low defect density.

Obtained predictions that can be compared with the actual test execution results and detected discrepancies should cause additional post-analysis of a test case, leading to a discovery of a software defect in the associated code module. A matching result confirms the actual test outcome was correct. Consequently, an opposing result can trigger a test re-execution and post-analysis investigation to check for a false positive or false negative. Specifically, each discrepancy in the results has different implications:

- Real test passed, and the prediction shows the test case failed: real test should be repeated as a possible false negative, where a pass should be indicative of a false positive in the prediction, while if the repeated real test does not pass, additional post-analysis should lead to a discovery of a software defect in the associated code module.

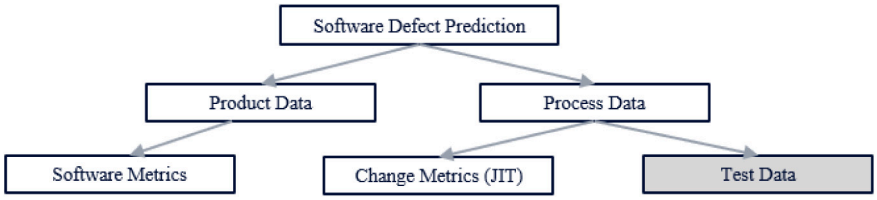


Fig. 2. A simplified view to the software defect prediction approaches based on input types.

- Real test failed, and the prediction shows the test case failed: this situation increases the probability of a software defect in the associated code module.
- Real test passed, and the prediction shows the test case passed: aligned results increase the probability of software defects not existing in the associated code module and lower the possibility of false positives.
- Real test failed, and the prediction shows the test case passed: real test should be repeated as a possible false positive, a pass should be indicative of possible issues with the environment or defective test automation procedure, indicating a lower chance of a software defect existing in the associated code module, while if the repeated real test does not pass, it increases the probability of a false negative in the prediction.

Analysing test history, which considers test cases that have recently failed, is closely related to defect prediction (Paterson et al., 2019). However, in our approach, the model is trained only on confirmed software defects and does not include environmental issues to make the gap even smaller. In our database, the failed test instances have been confirmed as defects by testers, who, after initial analysis, opened a defect report. Thus, there is considerable added value in using test case results leading to software defect predictions. Executing the test cases is hugely expensive, as some more sophisticated 5G test environments cost millions of EUR to build and maintain (Stradowski and Madeyski, 2023c).

Test cases can also be time-consuming as specific stability scenarios take several hours to execute. Therefore, having an additional verification mechanism for defect prediction can bring considerable operational savings. Furthermore, based on the model results, we can temporarily omit low-risk areas without defects, detect false positives that limit waste, or offer confirmation that discovered faults require a defect correction.

Also, we have compared our results with the defects that escaped to the customer in order to evaluate the coverage of the proposed solution. In our case, the failed test cases due to confirmed software defects at the studied system level included more than 90% of all software defects discovered after integration. This way, we effectively (in more than 90%) address the SDP problem by solving the test failure prediction problem, which is easier to use as we avoid all of the widely discussed issues related to the SZZ algorithm (Śliwerski et al., 2005) and its various implementations. The SZZ algorithm is the de facto standard for labelling bug-fixing commits and finding inducing changes for software defect prediction (Herbold et al., 2022). However, recent research uncovered many problems in different parts of this algorithm. For example, Herbold et al. (2022) found that about one-quarter of the links to defects detected by SZZ are wrong due to missed and false positive links. Furthermore, they reported that due to the combination of wrong links and mislabelled issues, only about half of the commits SZZ identifies are actually bug-fixing, and it misses about one-fifth of all bug-fixing commits. Herbold et al. (2022) also confirmed the earlier results (Herzig et al., 2013) that for every issue that is correctly labelled as a bug, there are 0.74 mislabelled bug issues. By employing LA2SDP, we escape from many problems related to the classic approach to SDP that require the SZZ algorithm and additional overhead related to calculating the necessary software metrics.

The tracking mechanism in Nokia was depicted in Table 1. A unique execution of a test case can pass or fail ('Test Run' and respective 'Test Result,' as in Table 2). After analysis and debugging, the failed test cases that end up as corrections in the software are marked with a resulting 'Defect Report'. Next, defect reports are tracked to specific areas of the code that have been corrected ('SW Module'), the team responsible for the modified code area ('Responsible Team'), the requirement of the product that this functionality satisfies ('Requirement'), and information about the failed 'Tested Build' (encoded as 'Build N'), as well as the build that this test has passed on last time it was tested 'Last Passing Build'. Furthermore, we can track all code changes ('SW changes') between the builds done in the area by the team ('Responsible Team') in a version control repository, which narrows down the changes that have been defective (changes between 'Build N' and 'Build N-y'). Consequently, putting all this information together allows tracking of a particular failed test run of a test case to defective changes (x' and y') in the code that has caused this failure. Such a mechanism can be applied to both real test case execution and machine learning predictions based on confirmed software defects. Furthermore, such a method can complement existing ML SDP approaches, making predictions on code and code change characteristics by addressing the issue from a black-box test perspective.

Identifying a failed test case does not equal finding a defect, but models trained on specific observations containing only confirmed software defects add considerable defect predictive value to the quality assurance process by offering a secondary verification mechanism to confirm or challenge test results. Since the test cases and open defect reports are tracked back to a specific product requirement, and each team owns a particular software area, we may assume that a predicted failed test case identifies a software defect in the given requirement.

This approach leads us to a new way, different from Madeyski and Kawalerowicz (2017), Kawalerowicz and Madeyski (2021) and Kawalerowicz and Madeyski (2023) that rely on build outcomes, to instantiate a lightweight alternative to SDP in industrial settings. Moreover, we are merging the aspects of test case selection and prioritisation (TSP) (Rothermel et al., 2001; Prado Lima and Vergilio, 2020) with ML SDP. According to Rothermel et al. (2001), there are four objectives of TSP: increasing the detection rate at the beginning of the regression test execution, increasing the system code coverage under test, increasing high-risk fault detection rate, and increasing the probability of revealing faults related to specific code changes. Our efforts constitute a deep dive into the fourth aspect by translating aspects of TSP to direct software defect prediction outcomes. Hence, the LA2SDP process can be visualised as in Fig. 3.

3. Methodology

We ask three research questions (**RQs**) in the initial phase of solution implementation to enable concluding our feasibility study within the company:

RQ1: Can LA2SDP achieve the expected performance of MCC > 0.3 (**E1**) with system-level test process data in Nokia 5G, assuming that we are allowed to use only already existing data (**E2**) and models that (according to literature) support interpretability (**E3**)?

Table 1
Tracking mechanism used to connect failed test cases to software defects.

Test Case	Requirement	Test run	Test result	Defect report	Resp. team	SW module	Tested build	Last passing build	SW changes
TC1	RQ1	Run1	PASS						
	RQ1	Run2	PASS						
	RQ1	Run3	FAIL	Report1	Team1	SWmodule1	Build N	Build N-x	x' changes
	RQ1	Run4	PASS						
TC2	RQ1	Run1	FAIL	Report2	Team2	SWmodule2	Build M	Build M-y	y' changes
	RQ1	Run2	PASS						
TC3	RQ2	Run1	PASS						
	RQ2	Run2	PASS						
TC4	RQ2	Run1	PASS						
TC5	RQ3	Run1	FAIL	Env. Issue	Filter out				

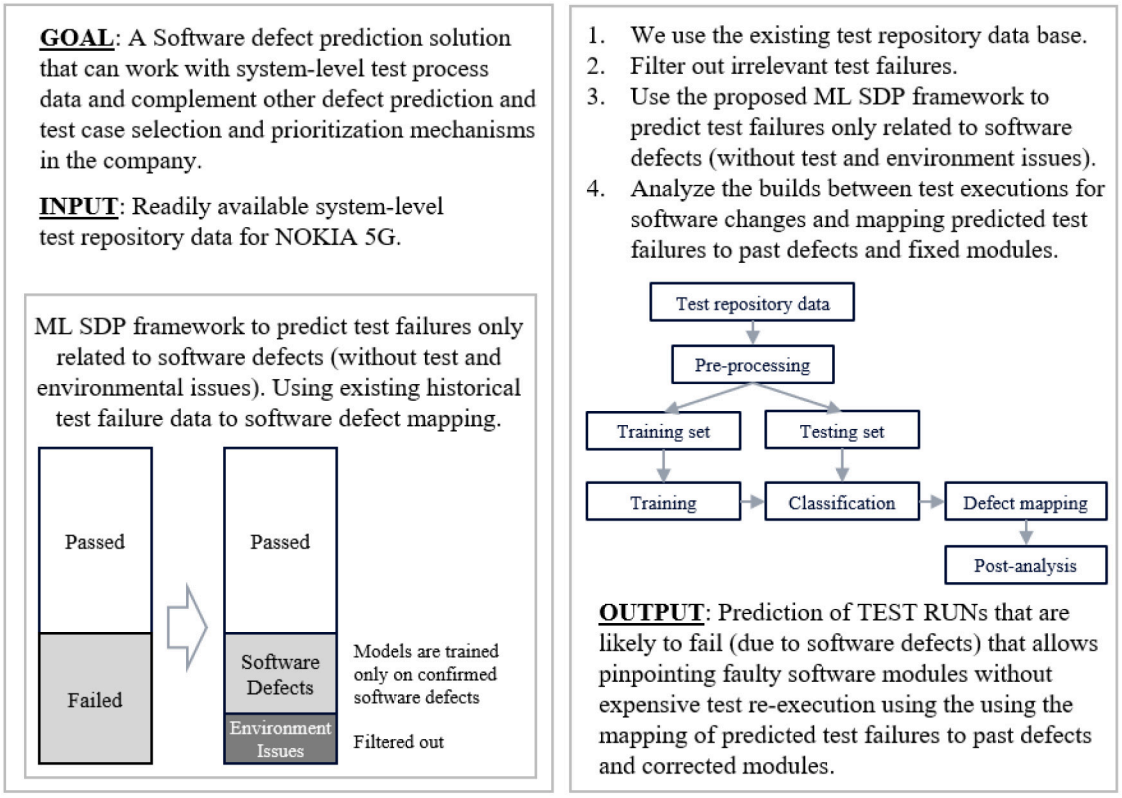


Fig. 3. LA2DSP process visualisation.

RQ2: Which learners employed in LA2SDP offer the highest predictive performance?

RQ3: What are the most important features in already existing data that can be used, and how can they be interpreted?

Also, we define success as satisfying the predefined requirements (see Section 2.1). Moreover, we collect feedback on the solution from subject matter experts in the company as an additional input source for validation and further improvement possibilities (see Section 5.5).

We started the data definition process by finding a suitable data set to satisfy our research questions. We gathered the data maintained in a dedicated test repository (called QC). We did not use the fault information stored in the fault report repository, as the QC data set was easier to obtain and analyse (which adheres to our second expectation **E2**). Also, as we investigated black-box system-level testing, we did not consider using software metrics (Li et al., 2018b). The aforementioned decisions were made together with subject matter experts within Nokia (four experienced software engineers and two quality managers) and had a critical impact on the research and its outcomes. When conducting research in a commercial environment with potentially important influence on many stakeholders, we wanted to make some initial inroads and show the value of ML SDP first, as then we would be better positioned to enhance the solution further based on the feedback and already the recognised expectation **E3**.

When considering the 5G software development life cycle (SDLC), we decided to focus on one of the interim stages of testing on the central build (see Fig. 1). Therefore, we only predict defects (that stem from commits) that can be detected at the system level, and at this point, we do not directly account for the escaped defects to the customer. Notably, the testing under scrutiny is black-box, meaning that the tester does not know the code implementation and focuses only on verification if the desired functionality works as specified in the requirements, analysing only the input and output states.

3.1. Data set

The data set we use is a collection of historical test process metrics from the main test case repository for the Nokia 5G quality assurance process (see Table 2). Overall, it contains almost 800,000 unique results for more than 100,000 test cases over a period of five and a half months from the beginning of January 2021 until the middle of June 2021, and executed into two-week feature builds. This constitutes a considerable database split by month into six separate files (with names starting with

Table 2
Data set statistics — count of elements and variable factors.

Data set	TEST. RUN.ID[a]	TEST. INSTANCE.ID[b]	EXECUTION. DATE[c]	TEST. PHASE[d]	RELEASE[e]
QC1	74305	17871	31	61	10
QC2	144051	23137	28	62	10
QC3	145506	23722	31	65	12
QC4	149182	28321	30	72	11
QC5	138262	24957	31	70	11
QC6	64475	16281	30	62	10
Data set	AUTOMATION. LEVEL[f]	TEST. OBJECT[g]	TEST. ENTITY[h]	ORGANISATION[i]	TEST. STATUS[j]
QC1	2	3	3	127	2
QC2	2	3	3	138	2
QC3	2	3	3	135	2
QC4	2	3	3	157	2
QC5	2	3	3	161	2
QC6	2	3	3	157	2

[a] TEST.RUN.ID — Identification number of the test run.
[b] TEST.INSTANCE.ID — Identification of the test case.
[c] EXECUTION.DATE — date of TEST.RUN execution.
[d] PROGRAM.PHASE — name of the milestone for which the testing contributes.
[e] RELEASE — name of the software system release for which the testing contributes.
[f] AUTOMATION.LEVEL — TEST.AUTOMATION.LEVEL is information about whether the test run was automated or manual, and AUTOMATION.LEVEL.FINAL is the target state of the test automation.
[g] TEST.OBJECT — information if the test run was part of Benchmark, New Feature, or Regression.
[h] TEST.ENTITY — information if the test run was part of CIT (continuous integration testing), CRT (continuous regression testing), or Manual (see also Fig. 1).
[i] ORGANISATION — name of the responsible organisation.
[j] TEST.STATUS — final state of the test run, and subject of our prediction (based on confirmed software defects).

QC). Thus, we can compare the performance of particular learners and analyse the differences between the performance of each learner on different data subsets.

The data was collected automatically from the central Nokia test repository and contains default features gathered for each test run entry by scripts and testers alike. The simple defect-code linking technique (Mausa et al., 2016) is based on historical defect reports that are mapped in the test repository to previous failed test runs, and the relationship between bugs and modules is one-to-one. Here, a crucial consideration is the granularity of predictions on the test case, test instance, or test run level:

- A test case has the lowest granularity and can run on different builds and environments and by different teams. A test case is characterised by a requirement it satisfies and the author. Hence, in our context, providing test case-level predictions does not bring sufficient defect prediction value.
- A test instance is a test case assigned to a particular feature build, characterised by the tester, test line, and environment. Test instance, as an aggregation of test runs and traced to test cases assigned to a software build, is much easier for a human to understand with XAI, but it does not yet enable precise identification of the software module containing the defect.
- Last, the test run represents the highest granularity, as it reflects multiple repetitions of the same instance. Hence, it allows the detection of low-occurrence and utilisation of a wider range of features to predict which runs will fail.

Each granularity level can be tracked to past software defects and used for SDP; however, we have chosen the test run as it carries the largest amount of information in our context, as well as is already mapped to defect reports and software modules in the repository, avoiding any additional pre-processing actions.

There are differences between data sets, e.g., wrt. missing data in the 'TEST.ENTITY' field. Thus, we have attempted to understand the root cause, but such occurrences were occasional throughout all time frames (not only the six we can provide) and were not related to data-gathering errors. Such a situation reflects reality and needs to be considered in any industry research. Second, there are no duplicate entries in the data sets, and each 'TEST.RUN.ID' represents a unique execution of a test case with unique characteristics (see Table 2). Last, each of our six data set files is heavily imbalanced, where number of passed test cases is much greater than the number of failed ones (less than 5%).

We train the models on data sets only containing failed test cases confirmed by opened defect reports (see Section 2.2). Naturally, variables associated with failures (software defects and environmental issues) were removed from the data sets prior to the training phase. For data definition, each point consists of three parts (Pradhan et al., 2020), each described and adjusted to our context:

- Data element or sample: a particular execution of a test instance (test case assigned to a particular release), as the same test case can be run over different releases multiple times with varying outcomes.
- Variables or features: multiple characteristics describing each execution of the test run: ID, Instance, Date, Phase, Release, Automation, Type, Entity, Organisation, and Result, as presented in Table 2.
- Observation or label: in our case, the observation is a single test run execution result — pass or fail state. In reality, there are more states that have blocked, postponed, ongoing, and similar inconclusive outcomes. However, we chose to focus only on the absolutes, as this is the primary goal of our solution.

Finally, we use and compare two approaches in our study, the expanding window and sliding window approaches, as they help analyse data in chunks when dealing with large datasets and continuous data streams. In the expanding window approach, the size of windows grows while sliding windows remain fixed. Consequently, the expanding windows cover more data as they progress, while sliding windows focus on the newest segments. For large datasets, as in our case, sliding windows can be more cost-efficient as fixing the length allows for limiting the computation time, assuming that the training data are sufficient for meaningful predictions. Finally, expanding windows are more suitable for reflecting long-term trends, while sliding windows are suitable for short-term patterns.

Note: Software companies can possess immense amounts of information on their process efficiency. Although the mechanisms to obtain this information and their structure were not designed with machine

learning tasks in mind, minimal pre-processing was needed, and the data presented in the following sections remained the same as the original values. However, to maintain confidentiality, we show the results obtained for historical test records (2021), not including the whole project repository but its selected subsets (it is worth noting that the predictive modelling on larger data sets generally tends to lead to better performance measured by MCC). Also, we had to remove potentially sensitive variables such as test case titles or author names that have predictive power.

Importantly, correct labels are the most critical information in the data set from the quality assurance standpoint, so there is high scrutiny of their correctness, from which we benefit in our approach. Moreover, a lot of additional features are generated automatically for each row describing each test instance, creating a vast amount of data that opens many analytical possibilities suitable for ML techniques from the get-go. That said, we designed our approach to be lightweight, and better results can be obtained by focusing more on pre-processing and better preparation of the data, which we plan to do in the next phases of our research.

3.2. Tools

We used the `mlr3` package in R language as a framework for predictive modelling (including also pre-processing) (Lang et al., 2019) and `DALEX` to support interpretability (Biecek, 2018).[3] All code and models, as well as software version information about the OS, R, and loaded R packages, are available in the Supplementary Material (Madeyski and Stradowski, 2024).

Note: All calculations were run on MacBook Pro Late 2023 (M3 Max, 48 GB RAM, Mac OS 15.2) and independently verified on Cluster Bem 2 provided by Wrocław Centre for Networking and Supercomputing.

3.3. Learners

We have used the following five supervised learners,[4] together with their tuned versions, supported by the `mlr3` framework for our ML SDP modelling:

- Classification Tree (`ct`) (Grochtmann and Grimm, 1993),
- Light Gradient-Boosting Machine (`lgbm`) (Ke et al., 2017),
- CatBoost Gradient Boosting (`cb`) (Prokhorenkova et al., 2018),
- Random Forest (`rf`) (Breiman, 2001),
- Naïve Bayes (`nb`) (Rish, 2001).

The learners were chosen to fulfil the expectation **E3**, i.e., all used models support interpretability. Decision trees are very interpretable as shown by Molnar (2023). Naïve Bayes is also considered to be an interpretable model because of the independence assumption, as it is very clear for each feature how much it contributes towards a certain class prediction as we can interpret the conditional probability (Molnar, 2023). Likewise, the rest of the models also fulfil the expectation of interpretability according to the literature (Aria et al., 2021; Konstantinov and Utkin, 2021).

Additionally, the selected models suit our data without sophisticated pre-processing steps (apart from simple ones like dealing with missing data) or creating additional collection mechanisms not already available within the company (**E3**). The set of classifiers selected for benchmarking is considered enough to get prompt and early feedback on the achievable performance.

[3] See also https://dalex.drwhy.ai/.

[4] We use the full names of the models, i.e., Classification Tree, Light Gradient-Boosting Machine, CatBoost Gradient Boosting, Random Forest, and Naïve Bayes in the text of the paper and respective short names, i.e., `ct`, `lgbm`, `cb`, `rf`, and `nb` as they are named in the `mlr3` package, in figures and tables generated by the tooling.

3.4. Performance measures

Our data, as often is the case with real-world applications, are severely imbalanced. Hence, we need to use proper performance measure(s). There is ample research on the benefits and risks related to particular performance metrics in software defect prediction research (Sokolova and Lapalme, 2009; Rizwan et al., 2019; Shepperd et al., 2014; Yao and Shepperd, 2021). Taking into account the arguments and recommendations of Shepperd et al. (2014) and Yao and Shepperd (2021), as well as Chicco and Jurman (2023a), we decided to depend on the main comparisons and conclusions using Matthew's correlation coefficient (MCC). Chicco and Jurman (2023a) concluded that MCC (that accounts for the whole confusion matrix, i.e., generates a high score in its $[-1;+1]$ interval only if the classifier scored high for all four quadrants of the confusion matrix) should replace the widely used ROC AUC as a standard statistic in all the scientific studies involving a binary classification. We report the performance of our models with five additional auxiliary metrics (accuracy, recall, precision, F-beta, and AUC), as well as a breakdown of particular results (TP — true positives, TN — true negatives, FP — false positives, FN — false negatives), to enable comparison with other studies and give the readers and Nokia practitioners an even broader understanding of the obtained results. Importantly, as the outcome of the feedback sessions, the precision metric evolved from the auxiliary metric suite to the second most important evaluation criterion (see Section 5.5).

4. Results

In this section, we analyse our results using two time-based split approaches, the expanding and the sliding window. Importantly, both approaches can constitute a realistic scenario for the company, and the use of unlimited access to the entire database opens more possibilities to achieve better predictive performance.

- The expanding window (Table 3) uses the time-based split of the data set (QC) where all older sets of data are used for training, while the latest one serves for testing. The training window length is growing with each new iteration. We start from window length two (Task1_3: QC1 to QC2 used for training and QC3 for testing), and finish with window length five (Task1_6: QC1 to QC5 used for training and QC6 for testing).
- The sliding window (Table 4) uses a predefined number of sets as training data and is tested on the latest one. The training set moves further with each new iteration. We provide the results for a window length of three (for example, Task2_5: QC2 to QC4 used for training and QC5 for testing), four (for example, Task2_6: QC2 to QC5 used for training and QC6 for testing), and five (Task1_6: QC1 to QC5 used for training and QC6 for testing).

We present the full results that were obtained for each learner with all measures across all data sets that we could disclose. Although all six measures are visible, we use the highlighted MCC and precision (with a ranking in Table 5), as well as focus on the window lengths of three, four, and six to evaluate which model is best in each scenario. A practical interpretation of our results per learner is presented below.

- CatBoost (`cb`) demonstrated the best overall performance among the learners, providing strong performance for both MCC and precision across multiple tasks, with highest MCC of the entire study, 0.370 for Task1_5, followed by and 0.342 for Task2_5, and precision of 0.742 for Task3_6 (it showed high precision across all larger tasks). However, tuning CatBoost led to a decrease in both MCC and precision, resulting in the untuned version performing better (except for MCC for Task1_4 increase from 0.254 to 0.215). On the other hand, in Task1_5, Tuned CatBoost significantly outperformed other models with precision of 0.934.

Table 3
Performance measures for walk-forward validation with expanding window approach (window length from two to five).

Task	Model	MCC	ACC	Recall	Prec.	Fbeta	AUC	TP	TN	FP	FN
1_3	ct	0.088	0.989	0.134	0.064	0.087	0.559	76	144057	1110	491
1_3	ct.tuned	0.086	0.989	0.134	0.063	0.085	0.571	76	144027	1140	491
1_3	lgbm	0.014	0.994	0.012	0.023	0.016	0.868	7	144869	298	560
1_3	lgbm.tuned	0.015	0.994	0.014	0.024	0.018	0.867	8	144835	332	559
1_3	cb	0.153	0.994	0.134	0.182	0.154	0.860	76	144826	341	491
1_3	cb.tuned	0.202	0.996	0.101	0.413	0.162	0.820	57	145086	81	510
1_3	rf	0.039	0.996	0.005	0.300	0.010	0.829	3	145160	7	564
1_3	rf.tuned	0.105	0.993	0.102	0.115	0.108	0.729	58	144720	447	509
1_3	nb	0.021	0.994	0.016	0.035	0.022	0.557	9	144917	250	558
1_3	nb.tuned	0.103	0.993	0.115	0.100	0.107	0.682	65	144582	585	502
1_4	ct	0.114	0.994	0.101	0.136	0.116	0.616	60	149024	381	535
1_4	ct.tuned	0.134	0.991	0.176	0.109	0.135	0.591	105	148545	860	490
1_4	lgbm	0.186	0.995	0.158	0.226	0.186	0.923	94	149083	322	501
1_4	lgbm.tuned	0.108	0.991	0.145	0.087	0.109	0.929	86	148506	899	509
1_4	cb	0.215	0.994	0.213	0.224	0.218	0.910	127	148964	441	468
1_4	cb.tuned	**0.254**	0.996	0.175	0.375	0.239	0.915	104	149232	173	491
1_4	rf	0.172	0.996	0.094	0.322	0.146	0.900	56	149287	118	539
1_4	rf.tuned	0.138	0.994	0.118	0.169	0.139	0.849	70	149060	345	525
1_4	nb	0.004	0.996	0.002	0.018	0.003	0.689	1	149351	54	594
1_4	nb.tuned	0.147	0.994	0.128	0.176	0.148	0.844	76	149049	356	519
1_5	ct	0.196	0.996	0.128	0.307	0.180	0.595	61	139245	138	416
1_5	ct.tuned	0.203	0.996	0.136	0.308	0.189	0.590	65	139237	146	412
1_5	lgbm	0.000	0.997	0.000	0.000	0.000	0.949	0	139381	2	477
1_5	lgbm.tuned	0.094	0.996	0.046	0.198	0.075	0.949	22	139294	89	455
1_5	cb	**0.370**	0.997	0.195	0.705	0.305	0.943	93	139344	39	384
1_5	cb.tuned	**0.334**	0.997	0.119	**0.934**	0.212	0.945	57	139379	4	420
1_5	rf	0.279	0.997	0.117	0.667	0.200	0.953	56	139355	28	421
1_5	rf.tuned	0.071	0.988	0.124	0.046	0.067	0.758	59	138167	1216	418
1_5	nb	0.048	0.993	0.055	0.048	0.051	0.711	26	138869	514	451
1_5	nb.tuned	0.235	0.993	0.291	0.195	0.234	0.912	139	138809	574	338
1_6	ct	0.094	0.956	0.022	0.457	0.043	0.511	64	61600	76	2783
1_6	ct.tuned	0.092	0.956	0.023	0.439	0.043	0.512	65	61593	83	2782
1_6	lgbm	0.138	0.957	0.032	0.662	0.060	0.740	90	61630	46	2757
1_6	lgbm.tuned	0.089	0.956	0.017	0.521	0.033	0.720	49	61631	45	2798
1_6	cb	0.123	0.957	0.017	**0.959**	0.032	0.812	47	61674	2	2800
1_6	cb.tuned	0.107	0.956	0.013	**0.947**	0.025	0.816	36	61674	2	2811
1_6	rf	0.090	0.956	0.012	0.739	0.024	0.753	34	61664	12	2813
1_6	rf.tuned	0.100	0.956	0.015	0.717	0.030	0.708	43	61659	17	2804
1_6	nb	0.146	0.952	0.082	0.330	0.131	0.712	233	61203	473	2614
1_6	nb.tuned	**0.280**	0.945	0.276	0.345	0.307	0.828	785	60187	1489	2062

- Random Forest (`rf`) and its tuned variant also rank highly in performance. RF achieves one of the highest precision values, such as 0.981 in Task2_5, alongside an MCC of 0.330 for the same task. Tuned RF maintains similarly high precision, such as 0.966 in Task2_5, but does not show significant improvements in MCC compared to its untuned counterpart. While not as consistently strong as CatBoost in MCC, Random Forest stands out as one of the most precise learners in this analysis.
- Naïve Bayes (`nb`) was a reliable performer, with noticeable improvements in both MCC and precision when tuned (e.g., Task2_6 where MCC increases from 0.249 to 0.295, outperforming all other learners on this test set). Nevertheless, its precision values were generally lower than those of other learners, rarely exceeding 0.3. Nonetheless, it consistently showed good MCC compared to weaker learners and showed improvement potential (the more training data, the better MCC). Notably, we tuned Naive Bayes despite it being a simple model that often does not benefit from iterative tuning. However, we found the Laplace smoothing parameter (`laplace`) offered by the model really helpful as a parameter to tune, making the tuned model a reliable performer overall.
- Light GBM (`lgbm`) provided moderate MCC and demonstrated some improvement in precision after tuning, particularly in certain tasks (e.g., MCC of 0.339 and a precision of 0.886 in Task2_5). However, its precision remained inconsistent, and its overall performance was outclassed by more robust models.
- Classification Tree (`ct`) showed moderately low MCC, with slight improvements in precision after tuning. However, its overall performance was weak, with both MCC and precision trailing behind other learners, indicating that it is not working well on our data sets.

We used the Hyperband multi-fidelity hyperparameter optimisation algorithm that dynamically allocates increasingly more resources to promising configurations and terminates low-performing ones (Li et al., 2018). However, sometimes, the tuned versions of the base models do not deliver better results. The plausible explanation is that the time limitation imposed on the search for the best combination of hyperparameters prevented finding a more effective combination. Due to the computation time limitations and the lightweight expectation towards the solution, we did not spend too much time finding the best combination of hyperparameters for each learner for each task. However, the LA2SDP software code we developed includes the used tuning mechanisms and the N_SECS parameter. Hence, adjusting the time budget should be very easy, and the provided software will be beneficial for wider-scoped studies within the company, as well as made available in the Supplementary Material (Madeyski and Stradowski, 2024).

We can observe that only the predictions made for QC5 satisfy our expectation of MCC > 0.3 (with the highest of 0.370), and for QC6 are close (0.295). CatBoost and Random Forest achieve MCC values above 0.3, showing that this expectation is achievable. CatBoost exceeds this threshold in tasks like Task1_5 (0.370 for untuned and 0.334 for

Table 4
Performance measures for walk-forward validation with sliding window approach (window lengths of three and four), without repeating results from Table 3 (Task1_4, Task1_5, and Task1_6).

Task	Model	MCC	ACC	Recall	Prec.	Fbeta	AUC	TP	TN	FP	FN
2_5	ct	0.179	0.996	0.126	0.260	0.169	0.569	60	139212	171	417
2_5	ct.tuned	0.158	0.995	0.136	0.190	0.159	0.600	65	139105	278	412
2_5	lgbm	0.092	0.996	0.029	0.292	0.053	0.942	14	139349	34	463
2_5	lgbm.tuned	**0.339**	0.997	0.130	0.886	0.227	0.930	62	139375	8	415
2_5	cb	**0.342**	0.997	0.145	0.812	0.246	0.955	69	139367	16	408
2_5	cb.tuned	**0.328**	0.997	0.117	0.918	0.208	0.928	56	139378	5	421
2_5	rf	**0.330**	0.997	0.111	**0.981**	0.200	0.949	53	139382	1	424
2_5	rf.tuned	**0.336**	0.997	0.117	0.966	0.209	0.792	56	139381	2	421
2_5	nb	0.127	0.942	0.568	0.033	0.063	0.834	271	131510	7873	206
2_5	nb.tuned	0.189	0.958	0.700	0.055	0.103	0.934	334	133696	5687	143
3_6	ct	0.087	0.956	0.021	0.429	0.040	0.510	60	61596	80	2787
3_6	ct.tuned	0.086	0.955	0.021	0.415	0.041	0.511	61	61590	86	2786
3_6	lgbm	0.116	0.956	0.029	0.529	0.055	0.734	82	61603	73	2765
3_6	lgbm.tuned	0.115	0.956	0.028	0.529	0.054	0.701	81	61604	72	2766
3_6	cb	0.105	0.956	0.016	**0.742**	0.032	0.800	46	61660	16	2801
3_6	cb.tuned	0.055	0.956	0.006	0.593	0.011	0.810	16	61665	11	2831
3_6	rf	−0.001	0.956	0.000	0.000	0.000	0.762	0	61675	1	2847
3_6	rf.tuned	0.053	0.956	0.005	0.636	0.010	0.701	14	61668	8	2833
3_6	nb	0.259	0.916	0.389	0.233	0.291	0.728	1108	58025	3651	1739
3_6	nb.tuned	**0.276**	0.896	0.497	0.210	0.296	0.832	1415	56367	5309	1432
2_6	ct	0.088	0.956	0.021	0.440	0.040	0.510	59	61602	75	2788
2_6	ct.tuned	0.087	0.956	0.021	0.434	0.040	0.510	59	61600	77	2788
2_6	lgbm	0.108	0.955	0.037	0.387	0.067	0.719	104	61512	165	2743
2_6	lgbm.tuned	0.165	0.957	0.041	0.715	0.078	0.683	118	61630	47	2729
2_6	cb	0.150	0.957	0.040	0.620	0.075	0.812	114	61607	70	2733
2_6	cb.tuned	0.139	0.957	0.026	0.802	0.050	0.802	73	61659	18	2774
2_6	rf	0.105	0.956	0.012	**0.971**	0.024	0.761	34	61676	1	2813
2_6	rf.tuned	0.102	0.956	0.012	**0.919**	0.024	0.708	34	61674	3	2813
2_6	nb	0.249	0.913	0.386	0.222	0.282	0.712	1100	57812	3865	1747
2_6	nb.tuned	**0.295**	0.904	0.499	0.230	0.315	0.831	1421	56924	4753	1426

Table 5
Ranking of learners according to their MCC and precision performance on the window length of three, four, and five.

MCC	Task1_4	Task2_5	Task3_6	Task1_5	Task2_6	Task1_6	Median	Rank
ct	8	7	6	6	9	7	7	6
ct.tuned	7	8	7	5	10	8	7.5	7
lgbm	3	10	3	10	6	3	4.5	3
lgbm.tuned	9	2	4	7	3	10	5.5	5
cb	2	1	5	1	4	4	3	2
cb.tuned	1	5	8	2	5	5	5	4
rf	4	4	10	3	7	9	5.5	5
rf.tuned	6	3	9	8	8	6	7	6
nb	10	9	2	9	2	2	5.5	5
nb.tuned	5	6	1	4	1	1	2.5	1
Precision	**Task1_4**	**Task2_5**	**Task3_6**	**Task1_5**	**Task2_6**	**Task1_6**	**Median**	**Rank**
ct	7	7	6	5	6	7	6.5	5
ct.tuned	8	8	7	4	7	8	7.5	6
lgbm	3	6	5	10	8	5	5.5	4
lgbm.tuned	9	4	4	6	4	6	5	3
cb	4	5	1	2	5	1	3	2
cb.tuned	1	3	3	1	3	2	2.5	1
rf	2	1	10	3	1	3	2.5	1
rf.tuned	6	2	2	9	2	4	3	2
nb	10	10	8	8	9	10	9.5	8
nb.tuned	5	9	9	7	10	9	9	7

tuned) and Task2_5 (0.342 for untuned and 0.328 for tuned). Random Forest also performs well in Task2_5, with MCC values of 0.330 for untuned and 0.336 for tuned. Naïve Bayes also performs well in certain cases, achieving MCC values close to the top learners in some tasks (particularly on QC6). These results highlight that ensemble models often excel at capturing complex patterns, and tuning can improve their performance even further.

However, it is critical to note that, in the commercial introduction, we are not limited to the six mentioned sets and we use longer windows, which allows much better MCC results. Furthermore, what drives practitioners' adoption of ML-based prediction models is precision. If a tool or model points out a problem, there should be a high certainty that it indeed needs to be fixed. Several researchers investigated this challenge (Kharkar et al., 2022; Ismail et al., 2024; Tosun and Bener, 2009). For example, Kharkar et al. (2022) demonstrated that their models could improve the precision of static analysis by 17.5%, thus achieving precision 0.787–0.854. In our case, Random Forest and CatBoost trained on sufficiently large data sets are approaching the highest possible precision (on Task1_6, CatBoost achieved a precision of 0.959, while on Task2_5 and Task2_6, Random Forest achieved a precision of 0.981 and 0.971, respectively).

Conversely, a situation with mediocre MCC but very high precision indicates that the models fail to detect many true positives (also visible by low recall). In a large-scale industry setting like ours, it is not feasible to do a post-analysis of all results. Hence, we argue that it is more important to be almost always correct in indicating a failure (high precision) rather than to have practitioners do excessive work

in filtering out a vast amount of false positives. However, different business contexts may require different decisions.

The main implication for our case study, after including in our analysis also the precision metric, is that the best learner we identified is:

- CatBoost with consistently high MCC and precision across multiple tasks, particularly in Task1_5 (MCC of 0.370, precision of 0.705) and Task1_6 (precision of 0.959).

Honourable mentions:

- Random Forest with exceptional precision in Task2_5 and Task2_6 (0.981 and 0.971, respectively), with acceptable MCC values as well (MCC of 0.330 on Task2_2), but quite unstable.
- Tuned Naïve Bayes with highest MCC performance (specifically on Task1_6 with MCC of 0.295 (QC6 was particularly difficult for all models)); constantly improving MCC with larger training data, however, with poor precision across all data sets.

In summary, after considering the relative performance of analysed learners (Table 5), we selected CatBoost for further recommendation for the company as the most consistent predictor, achieving the best results in terms of MCC and precision. We also recommend the sliding window approach as the most suitable for the defined use case implementation scenario. It is important to note that our expectation of MCC > 0.3 was not achieved in all instances. However, in the actual implementation, we will be able to use the entire database without disclosure limitations (in size and features), resulting in more consistent and favourable outcomes.

4.1. Calculation time

We have also performed simple measurements of the time used for computation (on MacBook 2023, see Section 3.2), including both training and prediction duration, for each learner. Table 6 shows the differences between the computational efficiency of the machine learning models (not using their tuned versions, as they are dependant on parametrisation, especially fixed maximum run times in subsequent resamplings determined by the built-in mlr3 terminator `trm(''run_time'')`), helping to select appropriate models based on the specific requirements of training and testing times for specific business contexts.

LightGBM, Classification Tree and Naïve Bayes are the fastest in terms of training time, making them suitable for scenarios where quick model updates are needed. Due to the characteristics of Naïve Bayes, it had the shortest training time but the longest testing time, making it the third fastest. Then, Random Forest has shorter test times but much longer training times than the three mentioned learners. However, it is still much faster than CatBoost, as our best-performing learner needed, by far, the longest computation time.

The computation time of a learner can be an advantage in time-critical industry use cases and where real-time data processing is used to optimise quality assurance. In such cases, there needs to be a balance between predictive performance and the computation time, depending on the context and purpose. If the prediction takes a very long, it can be a significant hindrance in its practical application and have a negative impact on operational efficiency, cost, and decision-making speed. Moreover, it can frustrate practitioners if many training iterations have to take place (especially with shorter windows) or if significant human intervention is necessary (even more so when low precision results in many false positives). In our case, with such relatively short timespans, the differences are negligible, and the learner with the best predictive performance was chosen.

Table 6
The average computation times for each model on all tasks gathered automatically by the `mlr3` framework.

Model	Training time [s]	Testing time [s]	Aggregated time [s]
cb	570.43	0.64	571.07
ct	2.55	0.22	2.77
lgbm	2.06	0.37	2.43
nb	0.85	7.26	8.11
rf	102.08	4.92	107.00

4.2. Interpretability

To satisfy expectation **E3**, we exercise the opportunity to interpret/explain predictions on our real-world quality assurance process data from the very beginning of our adoption project (Stradowski and Madeyski, 2023a). Hence, we deployed an external, posthoc model-agnostic explainable AI framework `DALEX` that xrays a model and helps to explore and explain its behaviour. It was proposed by Biecek (2018) as a dedicated `R` package to offer consistent methodology and tools for model-agnostic explanations, which create numerical and visual summaries. Also, it allows for comparing multiple models to help understand their relative performance, which we consider doing in subsequent phases of our research. Consequently, `DALEX` can explain the classifier for a specific single instance of an already created model, enabling global and local understanding and is, therefore, suitable for our solution and sufficiently satisfies our goals (see Section 2.1).

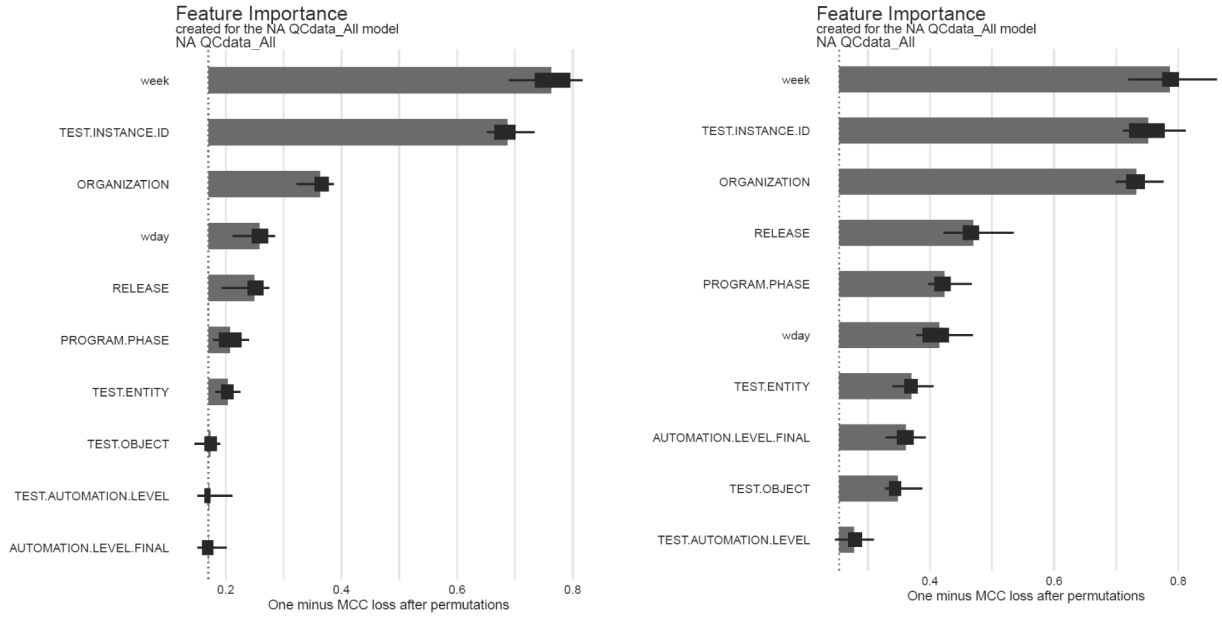
Global model explanations (also called global feature importance) help us understand which features impact the predictions of the CatBoost Gradient Boosting (as the best performing) and Random Forest (for comparison purposes) models over the aggregated data sets the most. A convenient way to compute variable importance is to permute the features (Breiman, 2001). Accordingly, a feature is important if permuting the feature causes a large degradation in model performance. This approach can be applied to any kind of model (i.e., it is model agnostic), and results are simple to understand.

We illustrate the use of the permutation-based variable-importance evaluation by applying it to the CatBoost Gradient Boosting and the Random Forest models on the data set that is composed of the longest window of Task1_6 (see Fig. 4). The dashed line in each panel of Fig. 4 shows the loss[5] for the full model, being either CatBoost Gradient Boosting or Random Forest. Features farther to the right are more important as permuting them produces a higher loss (measured with $1-MCC$). WEEK, TEST.INSTANCE.ID and ORGANISATION are the most important features for both best models and the order of importance for the three most important features does not differ.[6]

The interpretations obtained through `DALEX` show that while feature importance varies between models (Fig. 4), the week of execution, test instance, and organisation are the most impacting predictors.

[5] For the purpose of analysis of feature importance in the scenario where MCC is the performance measure of choice, it was necessary to implement a new loss function ($1 - MCC$) that is compatible with the chosen primary performance measure as `DALEX` does not provide the loss function based on MCC (if the performance measure of choice was AUC then `DALEX` offers the loss function $1 - AUC$). The implementation of the loss function (`loss_one_minus_mcc`) is included in the reproduction package in the `R` script `LA2SDP.R`.

[6] WEEK (as well as WDAY) is a result of EXECUTION.DATE decomposition into more granular features.



(a) The CatBoost Gradient Boosting model (b) The Random Forest model

Fig. 4. Feature Importance for CatBoost Gradient Boosting and the Random Forest model.

Meanwhile, the release, program phase, and automation levels are negligible. A detailed discussion of particular feature interpretations is presented in Section 5. Moreover, it is worth emphasising that the obtained results, backed by domain knowledge, can lead to interesting improvement opportunities and new quality-oriented projects started in Nokia (see Section 5 for more details on the improvement actions **A1–A3**, and a dedicated article on XAI by Stradowski and Madeyski, 2025).

5. Discussion

The results show that the new approach is feasible, and selected interpretable learners can achieve satisfying results with MCC (see **E1**). Hence, the proposed lightweight solution (LA2SDP) fulfils the expectations posed in Section 2.1. However, due to the expectation **E2** to keep the solution as simple as possible, LA2SDP also has an improvement potential that can be explored in subsequent phases of our research, namely:

- We used a limited number of ML algorithms, and possibly better results can be obtained by employing even more sophisticated models or pre-processing steps, as well as training models on larger data sets.
- Similarly, hyperparameter tuning was performed; however, further optimisation with a more extensive computation budget can be used to improve performance further. We invite the academic community to achieve better results on our data sets as we consider doing ourselves in the future.
- We did not conduct an extensive study of feature extraction or selection (Laradji et al., 2015; Agrawal and Menzies, 2018) as after making some preliminary inroads, we concluded that time and memory consumption would be too high and not in line with the expectations imposed by the company. The features we could use due to confidentiality requirements were limited, and thus selecting the most contributing ones was beyond the scope of this paper. If the solution is to be commercialised and accepted as part of the standard process, feature selection (if not already supported internally by employed models) must be considered as available resources will be much less constrained.
- Commercial data often suffer from missing samples. Our case is no different, mainly because the data set contains manually and automatically entered fields in the test repository. Most of the models we use can deal with missing data internally. In the case of Naïve Bayes, some pre-processing steps (feature imputation for missing data) are required. If the solution is accepted as part of the standard mode of operation, more sophisticated data imputation and pre-processing steps would be employed.
- Commercial data are frequently imbalanced. In each of our six data set files, the number of passed test cases is much greater than the number of failed ones (usually less than 5% of tests detect defects). Including mechanisms dealing with class imbalance is an absolute necessity. Class weight mechanisms were employed when supported by the models, but more sophisticated sampling algorithms can be considered if the LA2SDP solution is accepted as part of the standard mode of operation. For now, due to the class imbalance, our LA2SDP solution employs the MCC performance measure that considers all four quadrants of the confusion matrix, which is recommended in such an imbalanced scenario (Shepperd et al., 2014; Yao and Shepperd, 2021; Chicco and Jurman, 2023a). Also, MCC can later be used for further hyperparameter optimisation.

The software development life cycle in Nokia 5G is a very complex process (see Section 2), and in consequence, the SDP life cycle is similarly challenging to manage. Thus, our project impacts a limited area within a larger ecosystem with numerous relationships, stakeholders, and requirements that must be thoroughly considered. That said, we have proposed an idea of how to manage such process complexity and how to embed SDP in multiple test phases by utilising the multidimensional knapsack problem (Stradowski and Madeyski, 2023b), adhering to the company expectation that our solution works on system-level data complementing other already existing SDP and TSP mechanisms in the quality assurance process.

5.1. Answers to research questions

Below, we provide a detailed interpretation of our results and answers to the research questions we designed together with Nokia practitioners (see Section 3).

5.2. Answer to RQ1

As suggested by Tosun et al. (2011), it is important to decide how and when a defect prediction model will be used within the development life cycle. We aimed to understand if built models can be used with acceptable performance on one of the last phases of the SDLC. Therefore, our research sought a simple solution (**E2**) to gain value by proposing a lightweight alternative to SDP usable on industrial, real-world system-level test data sets. The proposed LA2SDP and obtained results showed that even relatively simple ML models can give adequate prediction results reliably pinpointing where to focus quality assurance activities thanks to high precision achieved by the best models.

First, despite using various search engines and digital libraries, as well as conducting systematic mapping study (Stradowski and Madeyski, 2023e) and systematic review (Stradowski and Madeyski, 2023d), we did not find similar studies using test process information for comparison. However, we are able to discuss our results in the context of studies that used similar prediction methods but were set in different contexts and used different inputs. First, we briefly compared our results with research by Arrieta et al. (2021), who used machine learning techniques to build test oracles in an industrial case study on elevator dispatching systems. The regression learning algorithms of this oracle are trained by using data from previously tested versions. The results of the five validated regression learning algorithms show that Regression Tree and SVM algorithms performed the best with slightly less precision than in our case (best scenarios achieved results of 0.89). We also compare our work to the research done by dos Santos and Figueiredo (2020), who studied the software feature impact on ML SDP performance using seven different classification algorithms. Apart from a custom-built Unbiased Search XGBoost algorithm, which turned out to be the best in the comparative study, Random Forest also proved very effective using both AUC and F1-score, similar to our case. Also, both research results demonstrate how a limited set of features can contribute to acceptable results. Finally, Shepperd et al. (2014) found that the mean and median MCC values achieved in ML SDP studies are 0.305 and 0.308, respectively.

After analysing five learners (Classification Tree, Light Gradient-Boosting Machine, CatBoost Gradient Boosting, Random Forest, and Naïve Bayes), together with their tuned versions, our lightweight predictive modelling achieved results that can be considered a solid starting point. The best scenarios are aligned with the literature averages; however, with the advantage of using an approach that does not require any code metrics or additional software measurement initiatives. Instead, we utilise only the already existing historical test repository data. The best models achieved satisfactory results of MCC > 0.3 (as well as very high precision) on some of the evaluated data sets, indicating that we can positively answer our **RQ1**.

> **Answer to RQ1** (Can LA2SDP achieve the expected performance of MCC > 0.3 (**E1**) with system-level test process data in Nokia 5G, assuming that we are allowed to use only already existing data (**E2**) and models that support interpretability (**E3**)?): **LA2SDP can be applied to the system-level testing of Nokia 5G** (with all the imposed expectations **E1, E2, E3 fulfilled** — achieving MCC > 0.3, using existing data, and enabling interpretability).

5.3. Answer to RQ2

To answer our **RQ2**, we wanted to understand which learner would offer the best practical performance.

- CatBoost turned out to be the best in terms of the overall MCC and precision results. CatBoost is a relatively new Gradient Boosting algorithm developed at Yandex that offers an innovative approach for processing categorical features (Prokhorenkova et al., 2018) (which appeared helpful due to the categorical features we operated with). On the other hand, it is essential to note that CatBoost was also one of the slowest in our computation time comparison.

Also, the honourable mentions go to:

- Random Forest, a learner widely known for its flexibility and providing acceptable results most of the time, even without hyperparameter tuning (Breiman, 2001; Fu et al., 2016). Random Forest combines the decision tree algorithm with bootstrap aggregation and constructs a large number of decision trees on random subsets of the training data. It allows the algorithm to learn complex relationships between the features and the dependent variable (Breiman, 2001).
- Naïve Bayes, a probabilistic classification algorithm that predicts the probability of a data point belonging to a particular class (Rish, 2001). It assumes that the features of a data point are independent of each other, hence the term "naive". We have utilised the 'impute features by sampling from non-missing training data' and 'impute features by their mode' preprocessing techniques to mitigate the missing values; however, we presented only the latter, which gave slightly better results.

Notably, as we found the answer to our **RQ1** using MCC, during the research, we added precision as a secondary metric to choose which learners performed the best, as it was a direct request from our participating practitioners (see Section 5.5). Hence, while still using MCC to compare and benchmark against the set expectations (**E1**), we have focused on evaluating both MCC and precision measures to obtain as useful results as possible. Consequently, some of the models achieved very high levels of precision (more than 0.9), showing they are well suited for industry adoption. Furthermore, it is important to reiterate that we can only publish the results based on a limited set of data, whereas, in reality, we can expand the windows to find much better results.

In the current phase of implementation, we ensured, based on literature (see, e.g., Molnar, 2023; Aria et al., 2021; Konstantinov and Utkin, 2021; Biecek, 2018), that the models we used should support the interpretability of the results (**E3**). We successfully employed the `DALEX` package to demonstrate this ability in practice. That said, we focus more on this aspect in the subsequent phases of the adoption project (Stradowski and Madeyski, 2025).

Moreover, our approach is consistent with the conclusions of systematic literature reviews conducted by Durelli et al. (2019), Pandey et al. (2021), and Pachouly et al. (2022) in terms of established state-of-the-art practices, and confirms the business potential of ML SDP. Importantly, our study also shows how relatively straightforward such implementation can be, even in a complex industrial environment.

> **Answer to RQ2** (Which learners employed in LA2SDP offer the highest performance?): **CatBoost Gradient Boosting was the best-performing learner on the analysed data sets.** Thus, considering the consistent MCC and precision metric results, it can be recommended for commercial implementation. Furthermore, Random Forest and Tuned Naïve Bayes are also providing promising outcomes.

5.4. Answer to RQ3

Finally, we have found answers to **RQ3** by performing a feature importance study (Section 4.2). Next, we discussed the findings with our panel of Nokia experts to evaluate their significance, open opportunities for new domain expertise and knowledge discovery, and create options for starting dedicated improvement projects to increase product and process quality.

There are many benefits to be gained from building-in interpretability to the ML SDP solutions, such as transparency and trust, accountability and compliance, debugging and improvement, domain expertise, and new knowledge discovery (Barredo Arrieta et al., 2020). In our case, the main objective for enabling interpretability for the solution is the understanding requirement (Carvalho et al., 2019), as we want to use the results and explanations as an additional output for our users' benefit. Notably, one of the remaining reasons for the interpretability problem to remain unsolved is that interpretability is subjective and, therefore, challenging to measure and compare (Carvalho et al., 2019; Mohammadkhani et al., 2023). Consequently, the usefulness of understanding predictions is domain- and context-specific, and it is necessary to consider the benefit of the use cases and the added value of each distinct feature importance insight.

Specific features and their interpretations are provided below (see also Fig. 4) and have been discussed with the involved practitioners to design improvement actions (**A1–A3**):

- EXECUTION.DATE we have split into calendar weeks and days of the week, creating more predictive information and avoiding the ones that do not carry any predictive value. Consequently, the calendar week when the test run was executed emerged as the most important predictor, significantly more impactful than RELEASE and PROGRAM PHASE. This shows that continuous delivery and continuous integration (see also Section 2) work as designed, and the flow of defects is consistent for each release happening in cycles rather than big bursts. This also reflects the shorter cadence of new content being delivered to the central build based on two-week feature builds. Specifically, we observed that WEEK indicates a pattern of peaking defects at the beginning of each feature build entry when new functionalities are starting to be tested on the system level. This observation opened up a considerable opportunity for additional action within the company to steer the test schedules to be more optimal in terms of the tests for the riskiest areas to be executed first (ISTQB, 2023a) (**A1**). Moreover, this finding is especially important when we can utilise the entire database without any limitations in the final implementation.
- TEST.INSTANCE.ID is a feature our engineering practitioners perceived as the most crucial in making software defect predictions. The test instance is a test (procedure description) prepared for actual execution, with a test assigned to a particular test environment and set required execution parameters (i.e., SW build, tester, external device, customer). It contains all the most critical information for the engineer to identify the faulty SW module and gives meaningful inferences on the defect's location. Therefore, from a technical point of view, it is an essential feature for practitioners to enable software defect prediction mechanisms in our process, especially with the combination of organisation, test line, and granularity information (see also Section 4.2 for more information on the granularity of predictions). Importantly, further analysis of the interpretability information led to uncovering previously unknown relationships between defect-finding test instances and software module correction patterns. A dedicated project was started to investigate the usefulness of this information to further quality assurance improvements (**A2**).
- ORGANISATION turned to be one of the strongest characteristics, which our practitioners also expected as specific organisations are responsible for different code areas, and some modules are much more defect-prone than others (Fenton and Ohlsson, 2000). Hence, test teams responsible for the more defect-prone SW modules fail more cases that are induced by software defects; however, failures due to test incorrectness of environmental issues (which we filtered out, Section 2.2) are comparable. Second, this is an essential consideration for the high-management stakeholders. ML SDP and XAI results should steer investments in the organisations' capacity to maximise defect finding and help prioritise the limited test resources on the most risky SW modules (Jiarpakdee et al., 2021). Furthermore, in our process, test engineers are usually responsible for dedicated test lines, and crosschecking of organisation and test instances can help understand why specific test lines cause defect reports much more frequently than others. Also, this is an important consideration for the particular organisation's management as it can steer investments in test infrastructure to maximise phase containment and efficient hardware utilisation. Namely, we found that a larger-than-expected number of software modules are executed for specific configurations, which led to starting another project to investigate the ramifications (**A3**).
- TEST.AUTOMATION.LEVEL and AUTOMATION.LEVEL.FINAL of the analysed test runs had no meaningful influence on the defect prediction effectiveness. Therefore, a high ratio of automated test cases to manual test cases did not directly influence defect discovery in particular software modules; however, it adds essential benefits of test automation on the quality assurance processes in terms of reliability, cost, and speed (Garousi and Mäntylä, 2016).

Answer to RQ3 (What are the most important features in already existing data that can be used, and how can they be interpreted?): The most important features are related to the **week of execution, test instance, and responsible organisation.** Furthermore, the discoveries brought **new domain knowledge and process improvement opportunities** to the organisation.

5.5. Feedback

Feedback from experts plays a crucial role in establishing the validity of the approach within the company. After implementation and obtaining initial results, we organised feedback sessions with the involved technical staff and management to elicit additional perceptions, expectations, and challenges for the created solution (Stradowski and Madeyski, 2023d; Wan et al., 2020).

Consequently, we held three feedback sessions with six participants located in Poland in the form of a face-to-face focus group. Two test architects (each with more than six years of experience) and three test managers (each with more than four years of experience) were interviewed by a facilitator to guide the discussion and capture opinions on the solution and process retrospectives. The most important points raised were the following:

- During the retrospective and post-validation, all involved practitioners unanimously saw the obtained predictions as useful (confirming the observations by Wan et al., 2020) about the willingness to adopt defect prediction techniques).
- During the discussion on preliminary results, it was observed that from a practical standpoint, the precision metric is imperative as

it shows the proportion of units correctly predicted as defective. False positives are waste from a cost perspective and are seen as a detriment to ML tools by software engineers who are asked to act upon the results (Wan et al., 2020). Hence, we decided to elevate precision from an auxiliary metric to an important deciding factor in the selection of the models (on top of MCC).
- As we trained our model only on test cases that fail due to confirmed software faults (including true negatives and false negatives, where a tester created a defect report that only after deeper analysis turned out not to be a real defect), experts saw value on comparison between predictions and actual test results for both cases (Section 2.2).
- Also, exploring the second group of failed test cases that were not confirmed as software defects to predict defects in the test environment (see Section 2.2) was discussed.
- A business case evaluation from a monetary perspective was done, showing a positive return on investment (ROI) and satisfying our expectation **E2**. Details have been described in a dedicated publication: Stradowski and Madeyski (2024).
- The specific feature importance values were valuable to practitioners and led to further evaluation under the company's improvement framework. Therefore, not only did the obtained new insight lead to knowledge discovery, but it also triggered specific follow-up actions (Stradowski and Madeyski, 2025).
- The management perspective on how to stabilise and treat ML SDP as a standard practice is important to facilitate the adoption. Aspects such as competence development, communication, and innovation, but also effort estimation, maintenance costs, and technical debt (Sculley et al., 2015), need to be studied further to accelerate the process.
- From the process perspective, making sure the predictions are accurate for incoming new data sets is imperative. Hence, the iterative nature of new data availability defines the need for time-based data set split and evaluation, being the go-to approach in the industry (new process visualisation in Fig. 1).

Consequently, after the commercialisation of the solution, the 5G quality assurance in Nokia is planned to include the ML SDP mechanism that is a time-based split to run every two weeks on new data after each feature build (Fig. 1). The model predicts failed runs within a test instance, which are tracked to respective requirements and software modules for additional analysis. Furthermore, the model can be compared with the actual test results of said test cases to detect false negatives and further tighten phase containment. Second, based on the confidence of the obtained predictions, test architects can make decisions on omitting specific test cases, which, due to the fact that they can be very expensive to run (Stradowski and Madeyski, 2024), can lead to meaningful operational savings in each feature build. In the next steps, dedicated product and process quality improvements will be added to increase the confidence and trust in the predictions as well as a decision to run the modelling more frequently based on a cost–benefit analysis (Stradowski and Madeyski, 2025).

5.6. Key lessons and takeaways

During the entire research and implementation effort, we have carefully documented the lessons learned, takeaway messages, and practical advice based on our experience. Below, we summarise the most important ones, focusing on increasing knowledge transfer from industry to research.

- A clear understanding of the expectations towards the implemented solutions is critical to a project's success (IIBA, 2015). In our ML SDP undertaking, we launched a survey among Nokia test practitioners to elicit opinions on the challenges within the current processes and uncover improvement opportunities (Stradowski and Madeyski, 2023c). Second, together with our project team from the company, we defined and reviewed the set of clearly defined requirements to enable everyone involved to work towards common objectives.
- One of the most critical enablers for the introduction of new technology is profitability (IIBA, 2015). To evaluate the cost-effectiveness of our solution, we used an abbreviation of the general cost model (Herbold, 2019) and calculated the return on investment (ROI) and benefit-cost ratio (BCR) financial ratios to confirm the profitability of our approach. We considered different periods of operation, varying efficiency of predictions, and two scenarios (lightweight and heavyweight). As a result, calculations have shown that the implemented ML SDP can have a positive monetary impact and be cost-effective in both scenarios, supplying management support (Stradowski and Madeyski, 2024).
- Many ML approaches have been developed and can be used in vivo (Stradowski and Madeyski, 2023d); however, reliable performance metrics need to be used to interpret the results correctly. Shepperd et al. (2014) and Yao and Shepperd (2021), as well as Chicco and Jurman (2023a), argue that the main comparisons and conclusions should rely on MCC and using incorrect metrics can lead to incorrect decisions. For example, if we would base the performance assessment of the models on the widely used accuracy metric, then all of the models could be assessed as excellent, which is not the case.
- Precision is a key metric to evaluate if the model is good at avoiding instances wrongly classified as positive, which is an important factor for practitioners (Wan et al., 2020). On the other hand, high recall indicates successful identification of most of the positive instances, and minimising false negatives can be crucial for tasks where defect leakage is very costly. Hence, comparing models with the reliable MCC metric but also taking into consideration other metrics based on the context is imperative for in vivo adoption.
- Advanced learners and techniques such as boosting, feature selection, or hyperparameter tuning should be considered depending on the context (Pachouly et al., 2022). However, committing to highly effective but heavy solutions needs to take root in the defined requirements, and corresponding investments need to be justified from the perspective of business needs as computational efficiency is examined. It is even more crucial to pay attention to the time and memory efficiency from the beginning as the data sets used by the company will only be bigger and bigger.
- ML studies are often performed with a predefined amount of data. However, software defect prediction is an iterative process of appending new data to create new predictions continuously. Therefore, a time-based data set split is the appropriate approach to evaluate the predictive performance and is a critical takeaway from our case study. Unfortunately, the iterative nature of ML SDP is rarely explored in the current literature, and long-term predictive performance still needs more research.
- The most formidable aspect of the project was related to gathering a suitable data set that allows actionable outcomes. Commercial companies can possess vast amounts of data that can be used for ML SDP; however, choosing and obtaining a practical set is not a trivial task. Domain knowledge and ML expertise need to be integrated and leveraged to select data that is easily obtainable, allows accurate predictions of the future, and enables meaningful use cases that are valuable for practitioners. After many considerations, we decided to use the test repository, where the format was suitable and minimal pre-processing was needed; however, downloading data from the repository was very long and required IT support, as it was not adjusted to handle the big files we needed for calculations. Nevertheless, we did not need to create and maintain a database specifically for the sake of ML tasks, which was very well received by the company stakeholders.

- We included the interpretability consideration from the beginning of the study in the set expectations. Consequently, our study reached a level of interpretability/explainability suitable to the company's needs and enabled a specific understanding of domain expertise and new knowledge discovery. Moreover, we aim to assess the effectiveness of the explanations in facilitating stakeholder engagement and designing specific engagement strategies to further increase the chances of final success and limit the possible pushback during large-scale deployment.
- Last, close cooperation with technical experts and management stakeholders was critical to the success of our project. We consulted and reviewed the progress of the implementation at each major milestone of the work, from requirements through implementation to final cost evaluation. Consequently, close cooperation with practitioners enabled us to effectively overcome challenges, manage risks, and increase technical and organisational impact by synergising domain knowledge with technical expertise through effective academic collaboration.

6. Related work

There are several valuable primary studies that had a significant impact on our research efforts. Below, we summarise the main contributions relevant to our work and how they helped our efforts.

- Melo et al. (2019) wrote a practical guide to support finding change-prone classes, which can help software professionals improve their product quality and steer future code changes. Furthermore, the authors apply the guideline to a case study based on a commercial data set. The approach consists of two phases: designing the data set and applying the prediction. In the first phase, it is necessary to choose the independent and dependent variables and collect the needed metrics. The application of the prediction step includes statistical analysis, normalisation, outlier detection, feature selection, resampling, cross-validation, tuning, selection of performance metrics, and ensuring reproducibility. In our study, we have followed a selected subset of these steps, described in detail in Section 3.
- Rana et al. (2014) created a framework to support the adoption of ML SDP in the industry. Research highlights factors that need to be considered during in vivo introduction, such as general usefulness, reliability, and cost-effectiveness. The publication provides a comprehensive analysis of aspects rarely explored in academia, such as perceived barriers and benefits, availability of tool support, organisational characteristics, or needed competence ramp-up. The proposed framework influenced our research, especially in terms of new technology adoption challenges, building organisational readiness, and definition of requirements.
- Furthermore, we internalised the experience report by Tantithamthavorn and Hassan (2018) on defect modelling in practice, as it discusses several valuable recommendations, common pitfalls, and main challenges that were observed as practitioners attempted to develop SDP models in vivo. We have faced similar issues in our work, such as the risk of employing class rebalancing techniques when models are used to guide decisions, different learners providing greatly varied effectiveness, or replication difficulties due to confidentiality concerns.
- We briefly compared our results with those of Arrieta et al. (2021). Their study uses five learners on an industrial data set for elevator dispatching algorithms. The authors propose a custom-built test oracle to evaluate the overall quality of the system by analysing the previous versions of the system. As a result, the accuracy of the proposed test oracle when predicting test results ranged between 0.79 and 0.87. Although our study is based on test metrics and we analyse a different set of learners, the overall methodology is similar and allows for indirect comparison.
- A second study to which we compare our results is the research by dos Santos and Figueiredo (2020). We use a similar methodology, however, in an industrial environment. The authors aimed to explore software features of ML SDP in a frequently used data set with five large open-source Java projects (Eclipse JDT and PDE, Equinox, Lucene, and Mylyn). Specifically, seven classification algorithms are evaluated using AUC and F1-score measures to select the best-performing learners. The overall research methodology is similar and allows comparing predictive performance.

Finally, to our knowledge, no secondary ML SDP studies focus on business applicability other than the work done by Stradowski and Madeyski (2023d). This systematic literature review analyses publications on machine learning software defect prediction validated in vivo, where the authors identified 32 publications and documented relevant evidence of methods, features, frameworks, and data sets used in the industry. However, the analysis also showed a minimal emphasis on feedback, practical lessons learned, and cost-consciousness within the reviewed publications, which are vital from a business perspective and which we have emphasised throughout our study.

7. Threats to validity

Construct validity reflects to what extent the studied operational measures represent what the researcher has in mind and what is investigated according to the research questions. While the general approach for evaluating the models we use has been validated in many contexts, academic and real-world (Stradowski and Madeyski, 2023d), there is a range of performance measures to choose from. We decided to report a range of them to provide a comprehensive view of the results and to allow easier comparisons with other studies. However, to make the answer to our RQs more robust, we based our conclusions on the MCC metric recommended to use in imbalanced scenarios (Shepperd et al., 2014; Yao and Shepperd, 2021; Chicco and Jurman, 2023a). Secondly, as the predictors we use are generated automatically as well as manually by test engineers, they may not reflect the results perfectly. Also, as there are more test metrics gathered in the Nokia system-level test process than we considered in this study, other features, including dynamic ones (e.g., test site metrics), could be used in the next steps. We have also recognised a relevant construct improvement possibility; we do not consider test case or defect priorities, which is an attractive prospect from a business standpoint.

Internal validity concerns examining whether the true causalities of the outcomes observed in the study are independent variables or other factors. We applied a time-wise data split to exactly reflect the real-world scenario. However, we used a limited data set and five learners; therefore, other data sets and learners can lead to different results, which need to be further verified in later stages of implementation. Other threats to the internal validity of this study are possible faults in the implementation of our approach and in the tools and libraries we used. Also, we chose to use popular and established tools (e.g., `R` language and platform including several `R` packages like `mlr3`) to implement our approach and reviewed our code several times.

External validity concerns the extent to which it is possible to generalise the findings. As the main focus of the study is a proprietary industrial data set and process, the generalisability of the results is limited. Second, we applied our method to only one project within the data set; we did not draw any conclusions about cross-project defect prediction potential (Zimmermann et al., 2009), which still should be further verified. However, our proposed approach, as well as the key lessons and takeaways, should be relevant and valuable for any large-scale industrial system where similar test repository data can be gathered. Finally, we ensure that our results are reproducible by other researchers (Madeyski and Kitchenham, 2017; Kitchenham et al., 2020; Lewowski and Madeyski, 2022) by providing all the details (data sets, code, and results) in the Supplementary Material (Madeyski and Stradowski, 2024).

For quantitative analysis, the counterpart to reliability is conclusion validity (Wohlin et al., 2012), which, in our case, is concerned with whether the results and outcomes of an experiment support our conclusions. First, we have chosen a reliable performance measure (MCC) that accounts for the entire confusion matrix and gives meaningful results on imbalanced data. That said, the real-world data set we could disclose and base this research on is not exhaustive and is an arbitrarily selected subset by company representatives. Nevertheless, it was sufficient to build a working solution that allows for an initial comparison of models. Unfortunately, we have not found any studies using black-box test metrics for SDP; therefore, comparing our results with other research conducted in a similar context is challenging. We analysed and published a data set that was slightly modified from the original for confidentiality's sake and ensured the modifications were random to limit the impact on the derived conclusions. Last, we paid much attention to keeping a high standard for communication and documentation (Shepperd et al., 2014; Madeyski and Kitchenham, 2017).

8. Conclusions and future work

Our paper addresses two appealing application prospects of ML SDP. First, we show how test repository data can be used for the detection of software defects by filtering the test results to teach the learners only on test cases that directly lead to defect discovery, besides the established approaches typically relying on software and process metrics. The proposed approach limits the need for expensive retesting of test cases and triggers a post-analysis directly based on the prediction results and past-defects mapping. Second, we utilise a lightweight application of ML SDP instead of a classic but more complex solution using an SZZ algorithm implementation and code mining tools to gather software product or process metrics (Madeyski and Jureczko, 2015). Thus, we exercise simplicity in obtaining meaningful industry adoption inroads.

The proposed lightweight alternative to SDP is feasible, utilising the existing test process data to predict test failures induced by software defects (named LA2SDP). Not only have we obtained satisfying results on our test repository data sets from the black-box system-level testing of Nokia 5G gNB, but we have also found the process relatively straightforward to implement with the `mlr3` framework. Specifically, we have used twelve supervised learners (including tuned versions) and a time-based data set split with expanding and sliding windows to build our models. As a result, the CatBoost Gradient Boosting ranked highest in our evaluations, considering the MCC metric. Moreover, by incorporating feedback from our participating company experts, we highlight a very high precision, which has significant practical consequences in limiting false positives. Consequently, Random Forest and Tuned Naïve Bayes were also among the best-performing learners on our data sets. Last, we analysed the variable importance for two of our models, where the calendar week, test instance, and the responsible organisation proved to be beneficial predictors from a business point of view and triggered dedicated quality improvement actions.

In the subsequent steps, we plan to conduct more empirical studies on larger real-world time-split data sets from Nokia 5G system-level testing, including defining control limits for predictive performance sustainability. Also, we consider employing sampling techniques, additional learners, further hyperparameter optimisation, and feature selection/extraction to achieve better predictive performance. Lastly, we continue to work on the defined improvement actions, as well as propose new ones based on the following analyses done on company data. As the results we obtained were sufficient to decide that further research efforts will be executed within the company to adopt a similar approach as standard practice and use it commercially. Hence, we have shown that LA2SDP (Lightweight Alternative to ML SDP) has the potential to improve the quality assurance processes within Nokia as well as other companies and large software projects that employ precise tracking of executed test cases to found software defects, as well as impacted software modules.

CRediT authorship contribution statement

Lech Madeyski: Writing – review & editing, Writing – original draft, Visualization, Validation, Supervision, Software, Resources, Methodology, Investigation, Funding acquisition, Formal analysis, Data curation, Conceptualization. **Szymon Stradowski:** Writing – review & editing, Writing – original draft, Visualization, Methodology, Software, Investigation, Data curation, Conceptualization.

Declaration of competing interest

This research was financed by the Polish Ministry of Education and Science 'Implementation Doctorate' program (ID: DWD/5/0178/2021). Szymon Stradowski reports a relationship with Nokia Corporation that includes employment and non-financial support. Lech Madeyski reports non-financial support was provided Wroclaw Centre of Networking and Supercomputing that includes computational resources.

Acknowledgements

This research was financed by the Polish Ministry of Education and Science 'Implementation Doctorate' program (ID: DWD/5/0178/2021). Calculations have been partly carried out using resources provided by Wrocław Centre for Networking and Supercomputing (http://wcss.pl), grant No. 578.

Appendix A. Supplementary data

All research artefacts required to reproduce the results (code, results, and data sets from Nokia) are available in the Supplementary Material (Madeyski and Stradowski, 2024). At the start of the R script, `LA2SDP.R`, we run `set.seed(123)` and use the R package `renv 1.0.11` to manage package versions. In the Supplementary Material, we also include our lockfile (`renv.lock`) recording metadata about every R package so that the computational environment can be reinstalled on a new machine. Reproducibility is often problematic when parallelisation of computations is used, and we heavily use parallelisation to speed up computations. To overcome this issue and support reproducibility, we employed the `future` package that ensures that all workers receive the same pseudo-random number generator streams, independent of the number of workers.[7]

Supplementary material related to this article can be found online at https://doi.org/10.1016/j.jss.2025.112360.

Data availability

The manuscript has data and code included as electronic supplementary material available on Figshare: https://doi.org/10.6084/m9.figshare.28263290.

References

Agrawal, A., Menzies, T., 2018. Is "better data" better than "better data miners"?: On the benefits of tuning SMOTE for defect prediction. In: 40th International Conference on Software Engineering. pp. 1050–1061. http://dx.doi.org/10.1145/3180155.3180197.

Aria, M., Cuccurullo, C., Gnasso, A., 2021. A comparison among interpretative proposals for random forests. Mach. Learn. Appl. 6, 100094. http://dx.doi.org/10.1016/j.mlwa.2021.100094.

Arrieta, A., Ayerdi, J., Illarramendi, M., Agirre, A., Sagardui, G., Arratibel, M., 2021. Using machine learning to build test oracles: An industrial case study on elevators dispatching algorithms. In: 2021 IEEE/ACM International Conference on Automation of Software Test. AST, pp. 30–39. http://dx.doi.org/10.1109/AST52587.2021.00012.

[7] https://www.jottr.org/2020/09/22/push-for-statistical-sound-rng/

Barredo Arrieta, A., Díaz-Rodríguez, N., Del Ser, J., Bennetot, A., Tabik, S., Barbado, A., Garcia, S., Gil-Lopez, S., Molina, D., Benjamins, R., Chatila, R., Herrera, F., 2020. Explainable artificial intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI. Inf. Fusion 58, 82–115.

Biecek, P., 2018. DALEX: Explainers for complex predictive models in R. J. Mach. Learn. Res. 19, 1–5.

Breiman, L., 2001. Random forests. Mach. Learn. 45, 5–32. http://dx.doi.org/10.1023/A:1010950718922.

Carvalho, D.V., Pereira, E.M., Cardoso, J.S., 2019. Machine learning interpretability: A survey on methods and metrics. Electronics 8, 832. http://dx.doi.org/10.3390/electronics8080832.

Catal, C., Mishra, D., 2013. Test case prioritization: A systematic mapping study. Softw. Qual. J. 21, 445–478. http://dx.doi.org/10.1007/s11219-012-9181-z.

Chicco, D., Jurman, G., 2023a. The Matthews correlation coefficient (MCC) should replace the ROC AUC as the standard metric for assessing binary classification. BioData Min. 16, 4. http://dx.doi.org/10.1186/s13040-023-00322-4.

Damm, L.-O., Lundberg, L., Wohlin, C., 2006. Faults-slip-through—a concept for measuring the efficiency of the test process. Softw. Process Improv. Pr. 11, 47–59. http://dx.doi.org/10.1002/spip.253.

Durelli, V.H.S., Durelli, R.S., Borges, S.S., Endo, A.T., Eler, M.M., Dias, D.R.C., Guimarães, M.P., 2019. Machine learning applied to software testing: A systematic mapping study. IEEE Trans. Reliab. 68, 1189–1212. http://dx.doi.org/10.1109/TR.2019.2892517.

Fenton, N., Ohlsson, N., 2000. Quantitative analysis of faults and failures in a complex software system. IEEE Trans. Softw. Eng. 26, 797–814. http://dx.doi.org/10.1109/32.879815.

Fu, W., Menzies, T., Shen, X., 2016. Tuning for software analytics: Is it really necessary? Inf. Softw. Technol. 76, 135–146. http://dx.doi.org/10.1016/j.infsof.2016.04.017.

Garousi, V., Mäntylä, M.V., 2016. When and what to automate in software testing? A multi-vocal literature review. Inf. Softw. Technol. 76, 92–117. http://dx.doi.org/10.1016/j.infsof.2016.04.015.

Garousi, V., Özkan, R., Can, A.B., 2018. Multi-objective regression test selection in practice: An empirical study in the defense software industry. Inf. Softw. Technol. 103, 40–54. http://dx.doi.org/10.1016/j.infsof.2018.06.007.

Grochtmann, M., Grimm, K., 1993. Classification trees for partition testing. Softw. Test. Verif. Reliab. 3, 63–82. http://dx.doi.org/10.1002/stvr.4370030203.

Herbold, S., 2019. On the costs and profit of software defect prediction. IEEE Trans. Softw. Eng. 47, 2617–2631. http://dx.doi.org/10.1109/TSE.2019.2957794.

Herbold, S., Trautsch, A., Trautsch, F., Ledel, B., 2022. Problems with SZZ and features: An empirical study of the state of practice of defect prediction data collection. Empir. Softw. Eng. 27, 42. http://dx.doi.org/10.1007/s10664-021-10092-4.

Herzig, K., Just, S., Zeller, A., 2013. It's not a bug, it's a feature: How misclassification impacts bug prediction. In: 2013 International Conference on Software Engineering. ICSE '13, IEEE Press, pp. 392–401. http://dx.doi.org/10.5555/2486788.2486840.

Hryszko, J., Madeyski, L., 2017. Assessment of the software defect prediction cost effectiveness in an industrial project. In: Madeyski, L., Śmiałek, M., Hnatkowska, B., Huzar, Z. (Eds.), Software Engineering: Challenges and Solutions. In: Advances in Intelligent Systems and Computing, vol. 504, Springer, pp. 77–90. http://dx.doi.org/10.1007/978-3-319-43606-7_6.

Hryszko, J., Madeyski, L., 2018. Cost effectiveness of software defect prediction in an industrial project. Found. Comput. Decision Sci. 43, 7–35. http://dx.doi.org/10.1515/fcds-2018-0002.

IIBA, 2015. Babok: A Guide to the Business Analysis Body of Knowledge. International Institute of Business Analysis.

International Organization for Standardization, 2022. ISO/IEC/IEEE 29119-1:2022 software and systems engineering — software testing. URL https://www.iso.org/standard/81291.html. (Accessed 20 December 2022).

Ismail, A.M., Hamid, S.H.A., Sani, A.A., Daud, N.N.M., 2024. Toward reduction in false positives just-in-time software defect prediction using deep reinforcement learning. IEEE Access 12, 47568–47580.

ISTQB, 2023a. Advanced level test manager (CTAL-TM). pp. 1–82, (Accessed 2 September 2023).

ISTQB, 2023b. Foundation level syllabus v4.0. pp. 1–74, (Accessed 2 September 2023).

Jiarpakdee, J., Tantithamthavorn, C.K., Grundy, J., 2021. Practitioners' perceptions of the goals and visual explanations of defect prediction models. In: 2021 IEEE/ACM 18th International Conference on Mining Software Repositories. MSR, IEEE/ACM, pp. 432–443. http://dx.doi.org/10.1109/MSR52588.2021.00055.

Jing, X.-Y., Chen, H., Xu, B., 2024. Intelligent Software Defect Prediction. Springer Nature Singapore, http://dx.doi.org/10.1007/978-981-99-2842-2.

Kawalerowicz, M., Madeyski, L., 2021. Continuous build outcome prediction: A small-N experiment in settings of a real software project. In: Fujita, H., Selamat, A., Lin, J.C.-W., Ali, M. (Eds.), IEA/AIE 2021. Advances and Trends in Artificial Intelligence. from Theory to Practice. In: LNCS, vol. 12799, Springer, Cham, pp. 412–425. http://dx.doi.org/10.1007/978-3-030-79463-7_35.

Kawalerowicz, M., Madeyski, L., 2023. Continuous build outcome prediction: An experimental evaluation and acceptance modelling. Appl. Intell. 53, 8673–8692. http://dx.doi.org/10.1007/s10489-023-04523-6.

Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., Ye, Q., Liu, T.-Y., 2017. LightGBM: A highly efficient gradient boosting decision tree. In: NIPS. Curran Associates Inc., pp. 3149–3157.

Kharkar, A., Moghaddam, R.Z., Jin, M., Liu, X., Shi, X., Clement, C., Sundaresan, N., 2022. Learning to reduce false positives in analytic bug detectors. In: Proceedings of the 44th International Conference on Software Engineering. ICSE '22, Association for Computing Machinery, New York, NY, USA, pp. 1307–1316.

Kitchenham, B., Madeyski, L., Brereton, P., 2020. Meta-analysis for families of experiments in software engineering: A systematic review and reproducibility and validity assessment. Empir. Softw. Eng. 25, 353–401. http://dx.doi.org/10.1007/s10664-019-09747-0.

Konstantinov, A.V., Utkin, L.V., 2021. Interpretable machine learning with an ensemble of gradient boosting machines. Knowl.-Based Syst. 222, 106993. http://dx.doi.org/10.1016/j.knosys.2021.106993.

Lang, M., Binder, M., Richter, J., Schratz, P., Pfisterer, F., Coors, S., Au, Q., Casalicchio, G., Kotthoff, L., Bischl, B., 2019. mlr3: A modern object-oriented machine learning framework in R. J. Open Source Softw. http://dx.doi.org/10.21105/joss.01903.

Laradji, I.H., Alshayeb, M., Ghouti, L., 2015. Software defect prediction using ensemble learning on selected features. Inf. Softw. Technol. 58, 388–402. http://dx.doi.org/10.1016/j.infsof.2014.07.005.

Lewowski, T., Madeyski, L., 2022. How far are we from reproducible research on code smell detection? A systematic literature review. Inf. Softw. Technol. 144, 106783. http://dx.doi.org/10.1016/j.infsof.2021.106783.

Li, L., Jamieson, K., DeSalvo, G., Rostamizadeh, A., Talwalkar, A., 2018. Hyperband: A novel bandit-based approach to hyperparameter optimization. J. Mach. Learn. Res. 18, 1–52, URL https://www.jmlr.org/papers/volume18/16-558/16-558.pdf.

Li, Z., Jing, X.-Y., Zhu, X., 2018b. Progress on approaches to software defect prediction. IET Softw. 12, 161–175. http://dx.doi.org/10.1049/iet-sen.2017.0148.

Madeyski, L., Jureczko, M., 2015. Which process metrics can significantly improve defect prediction models? An empirical study. Softw. Qual. J. 23, 393–422. http://dx.doi.org/10.1007/s11219-014-9241-7.

Madeyski, L., Kawalerowicz, M., 2017. Continuous defect prediction: The idea and a related dataset. In: 14th International Conference on Mining Software Repositories (May 20-21, 2017. Buenos Aires, Argentina). pp. 515–518. http://dx.doi.org/10.1109/MSR.2017.46.

Madeyski, L., Kitchenham, B., 2017. Would wider adoption of reproducible research be beneficial for empirical software engineering research? J. Intell. Fuzzy Systems 32, 1509–1521. http://dx.doi.org/10.3233/JIFS-169146.

Madeyski, L., Stradowski, S., 2024. Supplementary material for "predicting test failures induced by software defects: A lightweight alternative to software defect prediction and its industrial application". URL https://madeyski.e-informatyka.pl/download/MadeyskiStradowski24Supplement.pdf.

Mausa, G., Galinac Grbac, T., Dalbelo Bašić, B., 2016. A systematic data collection procedure for software defect prediction. Comput. Sci. Inf. Syst. 13, 61. http://dx.doi.org/10.2298/CSIS141228061M.

Melo, C.S., Cruz, M., Martins, A.D.F., Matos, T., da Silva Monteiro Filho, J.M., Machado, J.C., 2019. A practical guide to support change-proneness prediction. In: International Conference on Enterprise Information Systems. ICEIS, pp. 269–276.

Mende, T., Koschke, R., 2010. Effort-aware defect prediction models. In: 2010 14th European Conference on Software Maintenance and Reengineering. pp. 107–116. http://dx.doi.org/10.1109/CSMR.2010.18.

Mohammadkhani, A.H., Bommi, N.S., Daboussi, M., Sabnis, O., Tantithamthavorn, C., Hemmati, H., 2023. A systematic literature review of explainable AI for software engineering. arXiv:2302.06065.

Molnar, C., 2023. Interpretable Machine Learning: A Guide for Making Black Box Models Explainable, second ed. Lulu.

Pachouly, J., Ahirrao, S., Kotecha, K., Selvachandran, G., Abraham, A., 2022. A systematic literature review on software defect prediction using artificial intelligence: Datasets, data validation methods, approaches, and tools. Eng. Appl. Artif. Intell. 111, 104773. http://dx.doi.org/10.1016/j.engappai.2022.104773.

Pan, R., Bagherzadeh, M., Ghaleb, T.A., Briand, L., 2022. Test case selection and prioritization using machine learning: A systematic literature review. Empir. Softw. Eng. 27, http://dx.doi.org/10.1007/s10664-021-10066-6.

Pandey, S.K., Mishra, R.B., Tripathi, A.K., 2021. Machine learning based methods for software fault prediction: A survey. Expert Syst. Appl. 172, 114595. http://dx.doi.org/10.1016/j.eswa.2021.114595.

Paterson, D., Campos, J., Abreu, R., Kapfhammer, G.M., Fraser, G., McMinn, P., 2019. An empirical study on the use of defect prediction for test case prioritization. In: 2019 12th IEEE Conference on Software Testing, Validation and Verification. ICST, pp. 346–357. http://dx.doi.org/10.1109/ICST.2019.00041.

Pradhan, S., Nanniyur, V., Vissapragada, P.K., 2020. On the defect prediction for large scale software systems — from defect density to machine learning. In: 2020 IEEE 20th International Conference on Software Quality, Reliability and Security. QRS, pp. 374–381. http://dx.doi.org/10.1109/QRS51102.2020.00056.

Prado Lima, J.A., Vergilio, S.R., 2020. Test case prioritization in continuous integration environments: A systematic mapping study. Inf. Softw. Technol. 121, 106268. http://dx.doi.org/10.1016/j.infsof.2020.106268.

Prokhorenkova, L., Gusev, G., Vorobev, A., Dorogush, A.V., Gulin, A., 2018. CatBoost: Unbiased boosting with categorical features. In: 32nd International Conference on Neural Information Processing Systems. NIPS '18, 31, Curran Associates Inc., Red Hook, NY, USA, pp. 6639–6649. http://dx.doi.org/10.5555/3327757.3327770.

Rana, R., Staron, M., Hansson, J., Nilsson, M., Meding, W., 2014. A framework for adoption of machine learning in industry for software defect prediction. In: 9th International Conference on Software Engineering and Applications. ICSOFT 2014, SciTePress, pp. 383–392. http://dx.doi.org/10.5220/0005099303830392.

Rish, I., 2001. An empirical study of the Naïve Bayes classifier. In: IJCAI 2001 Work Empir Methods Artif Intell, vol. 3.

Rizwan, M., Nadeem, A., Sindhu, M.A., 2019. Analyses of classifier's performance measures used in software fault prediction studies. IEEE Access 7, 82764–82775. http://dx.doi.org/10.1109/ACCESS.2019.2923821.

Rosa, G., Pascarella, L., Scalabrino, S., Tufano, R., Bavota, G., Lanza, M., Oliveto, R., 2023. A comprehensive evaluation of SZZ variants through a developer-informed oracle. J. Syst. Softw. 202, 111729. http://dx.doi.org/10.1016/j.jss.2023.111729.

Rothermel, G., Untch, R., Chu, C., Harrold, M., 2001. Prioritizing test cases for regression testing. IEEE Trans. Softw. Eng. 27, 929–948. http://dx.doi.org/10.1109/32.962562.

dos Santos, G.E., Figueiredo, E., 2020. Failure of one, fall of many: An exploratory study of software features for defect prediction. In: 2020 IEEE 20th International Working Conference on Source Code Analysis and Manipulation. SCAM, pp. 98–109. http://dx.doi.org/10.1109/SCAM51674.2020.00016.

Sculley, D., Holt, G., Golovin, D., Davydov, E., Phillips, T., Ebner, D., Chaudhary, V., Young, M., Crespo, J.-F., Dennison, D., 2015. Hidden technical debt in machine learning systems. In: Cortes, C., Lawrence, N., Lee, D., Sugiyama, M., Garnett, R. (Eds.), Advances in Neural Information Processing Systems, vol. 28. Curran Associates, Inc., Norwich, England, pp. 1–9.

Shepperd, M., Bowes, D., Hall, T., 2014. Researcher bias: The use of machine learning in software defect prediction. IEEE Trans. Softw. Eng. 40, 603–616. http://dx.doi.org/10.1109/TSE.2014.2322358.

Śliwerski, J., Zimmermann, T., Zeller, A., 2005. When do changes induce fixes? SIGSOFT Softw. Eng. Notes 30, 1–5. http://dx.doi.org/10.1145/1082983.1083147.

Sokolova, M., Lapalme, G., 2009. A systematic analysis of performance measures for classification tasks. Inf. Process. Manage. 45, 427–437. http://dx.doi.org/10.1016/j.ipm.2009.03.002.

Stellman, A., Greene, J., 2014. Learning Agile: Understanding Scrum, XP, Lean, and Kanban. O'Reilly.

Stradowski, S., Madeyski, 2023a. Bridging the gap between academia and industry in machine learning software defect prediction: Thirteen considerations. In: 38th IEEE/ACM International Conference on Automated Software Engineering. pp. 1098–1110. http://dx.doi.org/10.1109/ASE56229.2023.00026.

Stradowski, S., Madeyski, L., 2023b. Can we knapsack software defect prediction? Nokia 5G case. In: IEEE/ACM 45th International Conference on Software Engineering: Companion Proceedings. pp. 365–369. http://dx.doi.org/10.1109/ICSE-Companion58688.2023.00104.

Stradowski, S., Madeyski, L., 2023c. Exploring the challenges in software testing of the 5G system at Nokia: A survey. Inf. Softw. Technol. 153, 107067. http://dx.doi.org/10.1016/j.infsof.2022.107067.

Stradowski, S., Madeyski, L., 2023d. Industrial applications of software defect prediction using machine learning: A business-driven systematic literature review. Inf. Softw. Technol. 159, 107192. http://dx.doi.org/10.1016/j.infsof.2023.107192.

Stradowski, S., Madeyski, L., 2023e. Machine learning in software defect prediction: A business-driven systematic mapping study. Inf. Softw. Technol. 155, 107128. http://dx.doi.org/10.1016/j.infsof.2022.107128.

Stradowski, S., Madeyski, L., 2024. Costs and benefits of machine learning software defect prediction: Industrial case study. In: ESEC/FSE Companion '24: 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering Proceedings. pp. 92–103. http://dx.doi.org/10.1145/3663529.3663831.

Stradowski, S., Madeyski, 2025. Interpretability/explainability applied to machine learning software defect prediction: An industrial perspective. IEEE Softw. http://dx.doi.org/10.1109/MS.2024.3505544.

Tantithamthavorn, C., Hassan, A.E., 2018. An experience report on defect modelling in practice: Pitfalls and challenges. In: 40th International Conference on Software Engineering: Software Engineering in Practice. In: ICSE-SEIP'18, Association for Computing Machinery, New York, NY, USA, pp. 286–295. http://dx.doi.org/10.1145/3183519.3183547.

The 3rd Generation Partnership Project, 2021. 3GPP REL15. URL https://www.3gpp.org/release-15. (Accessed 19 August 2023).

Tosun, A., Bener, A., 2009. Reducing false alarms in software defect prediction by decision threshold optimization. In: 2009 3rd International Symposium on Empirical Software Engineering and Measurement. pp. 477–480. http://dx.doi.org/10.1109/ESEM.2009.5316006.

Tosun, A., Bener, A., Kale, R., 2011. AI-based software defect predictors: Applications and benefits in a case study. AI Mag. 32, 57–68. http://dx.doi.org/10.1609/aimag.v32i2.2348.

Wan, Z., Xia, X., Hassan, A.E., Lo, D., Yin, J., Yang, X., 2020. Perceptions, expectations, and challenges in defect prediction. IEEE Trans. Softw. Eng. 46, 1241–1266. http://dx.doi.org/10.1109/TSE.2018.2877678.

Wohlin, C., Runeson, P., Höst, M., Ohlsson, M.C., Regnell, B., Wesslén, A., 2012. Experimentation in software engineering: An introduction, second ed. Springer, Berlin Heidelberg.

Yao, J., Shepperd, M., 2021. The impact of using biased performance metrics on software defect prediction research. Inf. Softw. Technol. 139, 106664. http://dx.doi.org/10.1016/j.infsof.2021.106664.

Zimmermann, T., Nagappan, N., Gall, H., Giger, E., Murphy, B., 2009. Cross-project defect prediction: A large scale experiment on data vs. domain vs. Process. In: ESEC/FSE'09. ACM, New York, NY, USA, pp. 91–100. http://dx.doi.org/10.1145/1595696.1595713.