Costs and Benefits of Machine Learning Software Defect Prediction: Industrial Case Study

Szymon Stradowski
Wroclaw University of Science and Technology & NOKIA
Wrocław, Poland
szymon.stradowski@pwr.edu.pl

Lech Madeyski
Wroclaw University of Science and Technology
Wrocław, Poland
lech.madeyski@pwr.edu.pl

ABSTRACT

Our research is set in the industrial context of Nokia 5G and the introduction of Machine Learning Software Defect Prediction (ML SDP) to the existing quality assurance process within the company. We aim to support or undermine the profitability of the proposed ML SDP solution designed to complement the system-level black-box testing at Nokia, as cost-effectiveness is the main success criterion for further feasibility studies leading to a potential commercial introduction. To evaluate the expected cost-effectiveness, we utilize one of the available cost models for software defect prediction formulated by previous studies on the subject. Second, we calculate the standard Return on Investment (ROI) and Benefit-Cost Ratio (BCR) financial ratios to demonstrate the profitability of the developed approach based on real-world, business-driven examples. Third, we build an MS Excel-based tool to automate the evaluation of similar scenarios that other researchers and practitioners can use. We considered different periods of operation and varying efficiency of predictions, depending on which of the two proposed scenarios were selected (lightweight or advanced). Performed ROI and BCR calculations have shown that the implemented ML SDP can have a positive monetary impact and be cost-effective in both scenarios. The cost of adopting new technology is rarely analyzed and discussed in the existing scientific literature, while it is vital for many software companies worldwide. Accordingly, we bridge emerging technology (machine learning software defect prediction) with a software engineering domain (5G system-level testing) and business considerations (cost efficiency) in an industrial environment of one of the leaders in 5G wireless technology.

CCS CONCEPTS

• **Software and its engineering → Software testing and debugging**; *Empirical software validation.*

KEYWORDS

software testing, machine learning, software defect prediction, industry, case study, cost-benefit analysis

FSE Companion '24, July 15–19, 2024, Porto de Galinhas, Brazil

ACM ISBN 979-8-4007-0658-5/24/07
https://doi.org/10.1145/3663529.3663831

ACM Reference Format:
Szymon Stradowski and Lech Madeyski. 2024. Costs and Benefits of Machine Learning Software Defect Prediction: Industrial Case Study. In *Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering (FSE Companion '24), July 15–19, 2024, Porto de Galinhas, Brazil.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3663529.3663831

1 INTRODUCTION

Machine learning software defect prediction (ML SDP) is a promising field of software engineering with the potential to improve quality and lower the costs of the overall software development life cycle (SDLC). However, its industrial application has lagged behind academic research in the field [37, 52]. Studies show several reasons behind limiting in vivo applications [10, 11, 29, 51]. One of the most important causes is the insufficient understanding of the cost-effectiveness of ML SDP, as integrating value analysis into software engineering principles and practices is critical from the business perspective (adhering to value-based software engineering (VBSE) [4]). Therefore, our research utilizes a general cost model proposed by Herbold [13] for preliminary evaluation of cost and profit considerations necessary for a favorable business-driven decision to apply such solutions in the commercial context of Nokia 5G system-level testing [50]. Second, we present two real-world examples in order to compare the cost of poor quality (COPQ) resulting from post-release defects. Specifically, we calculate the difference between two COPQs, with and without using ML SDP, complementing the standard quality assurance process within the company. Third, we offer a simple Excel-based calculation sheet to automate building and evaluating similar models that other researchers and practitioners can use to build their own scenarios (see Appendix A) and increase the chances of stakeholder support [48].

We follow the case study approach as guided by Runeson et al. [41, 42]. In Section 1, we describe the rationale, related work, and highlight the main contributions. Second, Section 2 describes the business need and selected evaluation method. Next, Section 3 explains the proposed cost framework, followed by Section 4 with our case study examples and results. Finally, Section 5 and Section 6 offer the discussion and conclusions.

1.1 Related Work

Our work is grounded on papers exploring the costing of the ML SDP applications [20, 37, 57] published in recent years and selected based on systematic mapping study [52], as well as business-driven literature review [51]:

- Zhang and Cheung investigated the cost-effectiveness of applying defect prediction models in software quality assurance

processes [60]. Their study proposed a cost-effectiveness criterion based on the assumption that a prediction-based strategy should at least be more profitable than inspecting all or randomly sampled modules. Even though such deliberation can be much more complex in a commercial context, the implication is valid. Benefits must outlay the costs and need to be treated as a performance measurement for the solution to be practical. Nonetheless, we also add the presumption that the cost-effectiveness of SDP not only needs to outperform random sampling but also complement the already existing quality assurance effectiveness and outweigh the cost of commercial introduction.

- A paper by Herbold [13] on the cost and profit of software defect prediction offers deep insight into the problem of evaluating the related monetary considerations. The proposed model includes aspects such as quality assurance, post-release defects, the possibility that quality assurance fails to reveal predicted defects, and the relationship between software artifacts and detected failures. Moreover, the author discusses essential topics such as standard performance metrics being insufficient to evaluate the success of SDP efforts and conditions for cost-saving in terms of defined boundaries that must be fulfilled to allow positive profit gain. Most importantly, the research shows how the cost model can be used holistically to calculate the cost and profit of an ML SDP solution in vivo, which we use in our study.
- Next, Tunkel and Herbold [56] published a paper exploring the relationship between performance metrics and the cost-saving potential of defect prediction models. In this work, authors measure several performance metrics and their cost-saving potential in defect prediction. However, no stable relationship between cost savings and performance metrics was found, which is attributed to the inability of performance metrics to account for the fact that a small proportion of large software artifacts are the main driver of the overall costs. Therefore, any ML SDP effort to find the most effective prediction model must consider cost savings directly. We apply the same principle to our research by calculating popular financial ratios for the ML SDP implementation effort within the company.
- Lastly, we were able to find only a handful of publications referencing the costs and benefits of ML SDP in industrial contexts: Monden et al. [33], Rana et al. [39], Hryszko and Madeyski [15, 16], and Kang et al. [25]. Also, a systematic literature review focusing on the real-world application of ML SDP solutions by Stradowski and Madeyski [51] highlights this deficiency in real-world research in greater detail. Nonetheless, the mentioned publications helped us to benchmark our results and learn from the available observations, recommendations, and discussions.

1.2 Contributions

Our business-driven case studies [30] in Nokia facilitate the decision-making process for implementing ML SDP in an industrial context by introducing a step-wise framework to build cost-benefit effectiveness. Specifically, as monetary consideration [28] is a critical aspect of any company's operation, evaluating needed spending and comparing it to the profits obtained is necessary to prove a positive business case. Thus, apart from providing a guideline and discussion, we analyze two real-world examples of system-level testing of the Nokia 5G solution based on the research done by Stradowski and Madeyski [30]. Lastly, the performed calculations using our framework will also be used within Nokia to make a value-based decision on committing to further research and potential future commercial deployment of the solution [48]. Thus, our contributions in this publication can be summarized as below:

- Short description of the related literature on the existing approaches to cost evaluation of ML SDP.
- Illustration of the context for the provided examples (further baseline research details can be found in [30]).
- Cost and benefit calculation using a general cost model proposed by Herbold [13], based on assumptions and estimates provided by Nokia practitioners.
- Return on investment (ROI) and Benefit-cost ratios (BCR) for lightweight and advanced use cases.
- Excel-based framework for reproduction and building custom scenarios (see Appendix A).
- Discussion and recommendations on good practices to evaluate the cost-effectiveness of ML SDP.

2 PROJECT CONTEXT

As explained in Section 1.2, this paper is a subsequent step to an already executed research in Nokia [30] concerning developing a lightweight ML-based solution for SDP in the commercial context of the Nokia 5G quality assurance process (which we will call baseline research). It is based directly on its outcomes, expands the results by a cost model, and evaluates its performance in monetary terms as one of the main success criteria for the company. Moreover, all provided examples and calculations are derived from real-world settings within Nokia; however, the original estimates were modified for confidentiality reasons (we explain all such cases in Section 4).

Also, Figure 1 illustrates particular phases of testing in Nokia. The company uses continuous delivery, integration, and test (CDIT) process to build its products (including continuous integration (CIT), continuous regression (CRT), and continuous delivery regression tests (CDRT)). CDIT adheres to state-of-the-art test principles emphasizing shift-left[1] and strict phase containment[2]. Notably, if a fault slips through the entire process and is found during customer acceptance and deployment, increased COPQ is incurred.

2.1 Project Results

Nokia is a multi-million dollar telecommunication software and hardware powerhouse developing wireless telecommunication solutions. Specifically, our efforts focus on system-level testing of the 5G gNB base station [53]. The ML SDP implemented solution has been designed to be a lightweight framework developed as a feasibility study for the company's management to enable a data-driven decision on full-scale commercial implementation and deployment. Furthermore, a clear understanding of which attributes are essential for adoption decisions helps to prioritize the features during the

[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 [19].
[2] Defect phase containment measures how many defects were caught before they escape into later phases [18].

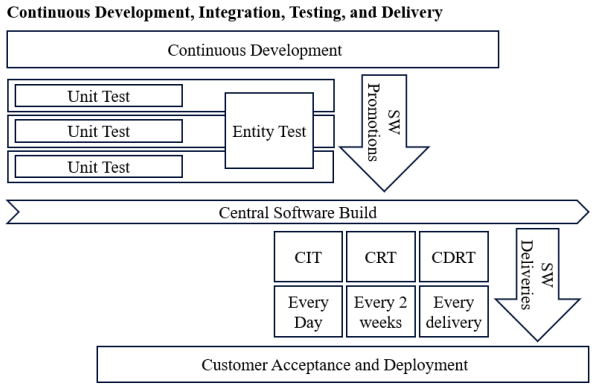

Figure 1: Nokia test process overview [50].

tool design and implementation [39]. The expectations set for our solution are as follows:

- Cost-effective (positive ROI and BCR values are the primary expectations).
- Utilizing existing data and features (a secondary expectation to support cost-effectiveness).
- Effective at predicting defects (a secondary expectation of MCC>0.5 to support cost-effectiveness).

So far, the latter two criteria have been satisfied by the baseline research [30]. Specifically, five classic learners (Naïve Bayes, Classification Tree, Random Forest, Light Gradient-Boosting Machine, and CatBoost Gradient Boosting) were implemented using the `MLR3` package[3]. Also, five times repeated 10-fold cross-validation was used to estimate the performance of the models on unseen data, where the analyzed data consisted of six sets from the existing test case repository in Nokia and containing test execution-related information. Consequently, approximately 800,000 test results from 100,000 test cases over a period of five and a half months were analyzed. For comparison, Matthew's Correlation Coefficient (MCC) was used as a reliable performance measure to benchmark the model's effectiveness [8, 45, 59] (also five other measures — Area under the Curve (AUC), Classification Accuracy (ACC), Recall, Precision, and F-beta score (Fbeta) with $\beta = 1$, were calculated to provide a comprehensive view of the results and to allow easier comparisons with other studies). Nemenyi-Wilcoxon-Wilcox tests have been performed to validate the results. Lastly, we provided a simple feature importance analysis to enable interpretability.

The results show that Random Forest and CatBoost had the highest MCC performance and achieved MCC between 0.674 and 0.874, while AUC was between 0.986 and 0.995. Importantly, we used the default versions of the classic algorithms without feature optimization, hyperparameter tuning, or other techniques to boost the performance further. We are confident that even better results can be obtained and invite other researchers to do so.

Furthermore, the research highlights several conclusions on the study's implementation process, feature importance, and operational aspects. The participating test practitioners welcomed the solution and saw considerable value in utilizing the obtained results. Also, during the feedback collection, it was raised by involved management participants that a value mindset is essential and that a cost-benefit analysis should be done before any full-scale commercial application of ML SDP solutions is considered.

[3]A description of the framework, together with details on the implantation of used learners and measures, is available under the link: https://mlr3.mlr-org.com/

3 METHODOLOGY

As mentioned, our research aims to evaluate the ROI and BCR results of ML SDP introduction into the Nokia 5G system-level test process. The company wants to know if investing in having the existing quality assurance mechanisms complemented by ML SDP has a positive business impact. To address this question, we have created an easy-to-understand framework based on the publication by Herbold [13], briefly introduced in Section 1.1 and applied to our real-world examples in Section 4.

Consequently, we define one research question (RQ1) during this phase of our implementation:

> **RQ1:** Is introducing ML SDP for defect prediction in an industrial Nokia 5G system-level testing process cost-effective for the company in terms of ROI and BCR?

Answering our **RQ1** allows the company to make a data-driven decision about further investment into a wide-scale introduction of ML SDP solutions in a real commercial setting. Furthermore, it can also encourage other companies and practitioners to do alike.

3.1 Cost-effectiveness of SDP

First of all, costing is a critical element of the operation of almost all commercial companies (see Economics 101 [32]). The main goal of the costing process is not only to understand how much resources are spent on particular activities within the company but also to ensure that new spending leads to building more value over time. Thus, the gained value should be higher than the incurred spending to provide a positive business case. The same principle should apply to ML SDP introduction in a real-world setting. Specifically, the purpose of SDP is to enable the allocation of quality assurance resources more efficiently than without it. Furthermore, said efficiency gain also needs to cover the required spending needed for the development and maintenance of the new solution itself. Especially as machine learning can incur massive ongoing technical debt and maintenance efforts [44].

Additionally, a vital aspect to consider is change management to carefully assess any substantial process improvements implemented on a broader scale [3]. Many interim steps are necessary for the change to be successfully implemented, from defining the purpose and success criteria, obtaining ambassadors, running pilots, and organizing feedback sessions and training programs before roll-out, to securing maintenance and phase-out of the new product. Such additional effort does not come for free within any organization, and decision-makers need to increase the chances the spending will bring the money back in terms of expected benefit.

There are several methods to evaluate the profitability of a business activity, which can be used depending on the context. Popular examples include [32]:

- Return on investment (ROI) - a simple ratio dividing the net profit (or loss) from an investment by its total cost.
- Internal rate of return (IRR) - the expected rate bringing the cost of a project and its cash inflows into equality.

- Benefit-cost ratio (BCR) - summarizing the relationship between a proposed project's relative costs and benefits.
- Net present value (NPV) - the difference between the present value of benefits and the present value of costs.
- Companies may use custom and more sophisticated investment appreciation methods depending on risk aversion and long-term strategy.

We decided on ROI and BCR as they are simple to calculate and understand and, at the same time, allow accurate comparison of different scenarios. Notably, all methods require an educated approximation of the cost and expected benefit of executing the project. However, for technology companies that create cutting-edge solutions, precise forecasting of software development work effort [6, 22] and evaluation of the expected returns is usually very difficult. Uncertainty and intrinsic risk leave managers with challenging decisions on future investment opportunities [26]. Our attempt to evaluate ML SDP introduction in a high-tech company provides an example of how such rough estimates can be done. That said, for some high-growth companies, the aim could be only increased revenue growth thanks to employed ML SDP instead of cost saving as in our case.

3.2 Estimation Assumptions

First of all, we need to highlight the main similarities and differences in the presumptions made by our study and the ones described in the surveyed papers (Section 1.1) All conjectures need to be clearly described and justified before cost-benefit estimations [47] and can serve as grounds for comparing results for future ML SDP studies. The main assumptions are described below:

- Cost projections are not exact and will not provide a straightforward answer. The following calculations are a range of predictions based on several assumptions and approximations that must be thoroughly analyzed and clearly explained. Furthermore, the outcome of a cost-benefit study is a broad spectrum of possibilities that helps understand the risk and level of uncertainty, increasing the probability of making correct business decisions [28]. Hence, the cost-benefit prediction is as good as the input data.
- Contrary to Herbold [13], we attempt to estimate the one-time cost of building a new application. In our case, we chose the most superficial way — all needed tools can be built in-house with only person-hour operational costs as both capacity and competence are available within the company [35]. However, we do not consider missed-opportunity costs or similar factors that may occur in vivo. Accordingly, there are several ways in which new software spending can be estimated [23], and each circumstance may require a specific approach. That said, evaluating the cost of any software project is complex and can not be accurately done context-free.
- We also do not neglect the continuous cost of execution (contrary to Herbold [13]) and provide an approximation in terms of person-hours and resulting monetary spending needed to run a cycle of SDP simulation on new data in once-every-two-weeks cadence.
- Nokia estimates escaped defect costs on all levels of testing, including the containment between internal phases as well as the customer escapes (see also Figure 1). We use an averaged subset of available values to evaluate potential gains without disclosing confidential details. Internally, the calculation can be done for specific components and test phases for more precise output.
- Importantly, we attempt to complement an already existing testing process within the company. In such case, the ML SDP model effectiveness can be compared with the actual results and not with the random sampling method [26]. The delta between the actual test results and ML SDP simulation results can be juxtaposed; a matching result confirms the actual test outcome was correct. Consequently, a contradicting result triggers a post-analysis investigation to check for a false negative. Each element of the confusion matrix has different implications from the cost perspective, as can be seen in Table 1:
  - **TP** True positive: post-analysis finds a new defect, preventing an escaped defect value is gained, and the cost of post-analysis is incurred.
  - **TN** True negative: post-analysis not done, no impact.
  - **FP** False positive: post-analysis confirms there is no defect, no value is gained, and the cost of post-analysis is incurred.
  - **FN** False negative: post-analysis not done, defect escapes as it would without ML SDP.

Table 1: Confusion matrix.

	Predicted faulty	**Predicted not faulty**
Actually faulty	True positives (TP)	False negatives (FN)
Actually not faulty	False positives (FP)	True negatives (TN)

  There might be other scenarios related to quality assurance imperfection and comparison with ML SDP simulation results; we provide the above examples as appropriate to our context.
- Following the mentioned concept proposed by Khoshgoftaar and Allen [26], we consider all elements of the confusion matrix (Table 1), as each has a different associated consequence from the quality assurance perspective (for example, false positive - post-analysis investigation only; false negative - no investigation, but defect escaped to the customer). However, we assume each positive prediction has the same cost related to the post-analysis following the models' predictions.
- Furthermore, similarly to Khoshgoftaar and Allen [26], we include the costs of misclassification as an essential factor to consider. The cost of acting on each type of prediction will depend on the mechanism that utilizes the model results. However, at this point, we do not account for any missed opportunity cost and see the cost of post-analysis to be the same for all predictions. This is a possibility for improvement in the accuracy of the cost model in the future. Similarly, building a statistical framework that would optimize the prediction performance to minimize the cost of misclassification could be done in the future.
- We purposefully split the discussion on cost-effectiveness from the predictive performance metrics, as there is no confirmed and stable relationship between cost savings and performance metrics [56]. Therefore, we perform separate calculations for

costs and benefits and do not consider original performance metrics results directly.
- Also, we consider the n-to-m relationship between defects and software modules in evaluating the historical data set against real escaped defect results, which is included in our mean-cost-per-escaped-defect approximation approach (see Table 5).
- Lastly, the issue where we deviate the most from the original proposal by Herbold [13] is the calculation of defect cost and, consequently, how the benefit is measured. Specifically, we possess detailed calculations of the historical numbers, containment efficiency, and costs of customer-escaped defects. Therefore, we can analyze different scenarios and derive a conclusion based on comparisons of the achieved results with and without using ML SDP. Thus, we do not account for the cost evaluation for the defects that escape while using the new process but rather treat the prevented defects as achieved benefit in operational savings.

Naturally, the above assumptions and limitations have a critical impact on the applicability of our solution. Both examples (Section 4) and provided Excel-based form (Appendix A) bridge the existing literature and available methods to the complex commercial reality of an emerging ML SDP technology from the financial perspective. Hence, the provided approach is influenced by the environment it originates from, and other contexts require different assumptions.

3.3 General Cost Model

We account for the below factors of the general cost model proposed by Herbold [13]:

- $cost_{INIT}$ is a one-time cost for the development and introduction of ML SDP into the existing development process.
- $cost_{EXEC}$ is the continuous cost of using the defect prediction model in the development process.
- $cost_{QA}$ is the cost of the quality assurance post-analysis due to the predictions made by the model (which also can be called the cost of intervention [31]).
- $cost_{DEF}$ is the COPQ of a software release enhanced by ML SDP. However, we do not account for this factor directly in calculating the cost of the solution. Thus, in our case, it represents the cost of defects that were saved due to ML SDP.
- $benefit$ in our case, the difference between COPQ with ML SDP and without ML SDP can be seen as the monetary benefit of the solution [2].

$$cost = cost_{SDP} = cost_{INIT} + cost_{EXEC} + cost_{QA} \tag{1}$$

$$benefit = cost_{NOSDP} - cost_{SDP} = cost_{DEF} \tag{2}$$

$$ROI = \frac{benefit - cost}{cost} \qquad BCR = \frac{benefit}{cost} \tag{3}$$

We will use the above equations to calculate the costs and benefits of several case study scenarios described in Section 4.

4 BUSINESS-DRIVEN REAL-WORLD EXAMPLES FROM NOKIA

In this section, we provide an extension of the past research detailed in Section 2.1 by describing and applying a cost model we used to evaluate the success of the developed solution (on top of the standard performance metrics — MCC (primary metric), AUC, ACC, Precision, Recall, and Fbeta). As reported by Tunkel and Herbold [56], so far, no relationship between cost savings and the most widely used performance metrics for ML SDP could be established. Therefore, cost considerations must be handled separately. We evaluate two scenarios resulting from our baseline research [30] with the methods proposed in the surveyed literature (Section 1.1):

Case 1: Lightweight solution - quick and easy implementation, utilizing available data, employing default learners with an acceptable level of performance.

Case 2: Advanced solution - a customized solution striving for the best possible performance of predictions.

As mentioned, we cannot publish the actual average costs of work invested in the project due to confidentiality concerns. However, for the purpose of our study, we used the person-hour cost of work by a software developer that is roughly based on the median salary in IT in Poland[4]. We decided on a value of 50 EUR per hour, accurately approximating the expense for the Nokia organization. Second, we publish a rough order of magnitude (ROM) of hours needed to develop both discussed solutions estimated together with responsible software architects and product owners.

4.1 Case 1: Lightweight Solution

The first scenario results directly from our baseline research and emphasizes a lightweight approach to bring immediate value without a substantial commitment to additional tool development and maintenance efforts. Therefore, the development and installation values assumed as in Table 2 are relatively low (similarly as suggested by Zhang and Cheung [60]) and cause recurring manual work for execution, but no additional licensing, servers, etc.

Table 2: $cost_{INIT}$ is the one-time cost of ML SDP introduction.

	Time required [h]	Cost [EUR]
Develop & Install	200	10,000
Training	20	1,000

$$cost_{INIT} = 11,000\,EUR \tag{4}$$

Second, one of the main requirements for the quick and easy solution was that it uses already available data, and its gathering and pre-processing is fully automated [17]. The cost relates to a practitioner observing and ensuring the process goes smoothly (Table 3). Here, we assume only one simulation run per software release cycle (the number can be increased in other scenarios with several rounds of predictions per release cycle, represented by Z).

$$cost_{EXEC} = Z \times 100\,EUR \tag{5}$$

[4] https://dou.eu/en/salaries?position=Software_Engineer&city=Wroclaw

Table 3: $cost_{EXEC}$ is the cost of the run simulations.

	Time required [h]	Cost [EUR]
Data collection	1	50
Modeling	1	50

To estimate the cost of post-analysis, we use internal Nokia evaluations based on JIRA[5] tooling and its effort management functionality. We use the exemplary mean value of hours logged as analyses of a failed test case leading to opening a fault report (in our example, it is 30 hours) and multiply it by Y, representing the number of predicted defects by the model and post-analysis (Table 4). Notably, we must accommodate the more realistic requirement where the test resources are not infinite. This can be done by allocating a predetermined and fixed Y (as we did) or having the cost grow with each additional post-analysis performed. Second, we do not account for additional test environment utilization and only consider the post-analysis effort to be spent as triggered code review. Also, we do not account for any further imperfection of quality assurance, causing the additional code review to miss the predicted defect, nor account for effort awareness, which is planned to be done in a subsequent step [31].

Table 4: $cost_{QA}$ is the cost of post-analysis for predictions.

	Time required [h]	Cost [EUR]
Post-analysis	30	1,500

$$cost_{QA} = Y \times 1,500\, EUR \tag{6}$$

In our example, we decided on $Y = 10$ — for each iteration of predictions, we select ten cases that ML SDP predicted as failed but standard testing passed. In such cases, we launch an additional intervention [31] (in our case, a code review). Significantly, this initial assumption for the cost-benefit analysis can be changed during regular operation and intermediate results so that the value can increase or decrease based on the observations.

Significantly, the baseline research does not account for the severity of the found defects. We add this distinction as an improvement and use historical data to assume that one in every ten faults are blockers, three out of ten are major, and the remaining six are minor defects (see Table 5). Nokia quality specialists regularly evaluate each category in terms of probability, cost, and effort; we use comparable values in our cost calculations. Also, it is worth noting that both the cost of the escaped defect and severity probability can frequently change, which can be easily reflected by modifying the values in our calculation sheet (Appendix A). To estimate the cost of escaped defects, we use internal Nokia evaluations based on the company's quality management analysis and IEEE standard definition [1]. For the below calculations, we use weighted mean value distinguished by severity, as visible in Table 5. Consequently, we simplify that the mean cost-per-escaped-defect is $8,000 EUR$.

The X_{SDP} is the number of defects escaping to the customer using the additional ML SDP solution, and X_{NOSDP} is the number of defects escaping to the customer not using the additional ML SDP solution. The delta between those two values represents the benefit of the introduced SDP mechanism on top of the traditional testing process. In such an approach, we do not need to estimate the number of escapes, just the potential savings.

[5] https://www.atlassian.com/software/jira

Table 5: $cost_{DEF}$ is the cost of the escaped defects.

	Minor	Major	Blocker
Escaped defect cost [EUR]	5,000	10,000	20,000
Relative probability	60%	30%	10%
Cost of escaped defect [EUR]			8,000

$$benefit = (X_{SDP} - X_{NOSDP}) \times cost_{DEF} \tag{7}$$

$$cost = cost_{INIT} + cost_{EXEC} + cost_{QA} \tag{8}$$

Namely, assuming 50% ML SDP efficiency, for every ten positives in the model that have passed the real test case execution (were false negatives during testing), five will be true positives, and code review can confirm an actual defect.

$$benefit = (10 - 5) \times 8000 = 40,000\ [EUR] \tag{9}$$

$$cost = 11,000 + 100 + 15000 = 26,100\ [EUR] \tag{10}$$

$$ROI = \frac{benefit - cost}{cost} = 0.53 \qquad BCR = \frac{benefit}{cost} = 1.53 \tag{11}$$

Even in such a pessimistic scenario, with only one-time execution of the ML SDP framework, testing resources limited to only 10, and 50% ML SDP efficiency, we get positive ROI and BCR results. Moreover, one of the ways to approximate the above predictive effectiveness more realistically is to use the Precision metric [60]. The average precision in our baseline research is 0.9, and assuming such performance to continue, we get much better profitability shown below (see Table 6 for more details on the calculation).

$$benefit = (10 - 1) \times 8000 = 72,000\ [EUR] \tag{12}$$

$$cost = 11,000 + 100 + 15000 = 26,100\ [EUR] \tag{13}$$

$$ROI = \frac{benefit - cost}{cost} = 1.76 \qquad BCR = \frac{benefit}{cost} = 2.76 \tag{14}$$

4.2 Case 2: Advanced Solution

The advanced scenario assumes a final commercial solution with a much higher initial investment needed (see Table 7) than the lightweight described in Case 1. Secondly, as the framework is fully automated, there is no cost directly resulting from simulations ($cost_{INIT} = 0$). Last, we also assume it offers a much higher Precision of 0.99 (for every 100 positives, 99 are true positives). The cost of execution, post-analysis, and escaped defects do not change (see Table 8 for details on the calculation).

Table 6: Exemplary calculation sheet for the lightweight scenario.

Input data		Calculations						Results	
Man-hour cost [EUR]	50	Cost INIT	[h]	[EUR]	Cost QA	[h]	[EUR]		
Simulations per release	1	Develop & install	200	10,000	Post-analysis	30	1,500	Cost	26,100
No. of post-analyses	10	Training	20	1,000	No. of post-analyses		10		
Precision	0.90			11,000	No. of releases		1	Benefit	72,000
No. of releases	1						15,000		
								ROI	1.76
Cost of escaped defect	[EUR]	Cost EXEC	[h]	[EUR]	Cost DEF		[EUR]		
Critical	20,000	Data gathering	1	50	Cost of escaped defect		8000	BCR	2.76
Critical %	10%	Modeling	1	50	No. of releases		1		
Major	10,000	No. of releases		1	No. of true positives		9		
Major %	30%			100			72,000		
Minor	5000								
Minor %	60%								
	8,000								

Table 7: $cost_{INIT}$ is the one-time cost of ML SDP introduction.

	Time required [h]	Cost [EUR]
Develop & Install	5000	250,000
Training	100	5,000
TOTAL	5100	255,000

$$cost_{INIT} = 255,000\,EUR \tag{15}$$

With ten post-analyses during each of the ten release cycles we get:

$$cost = 255,000 + 15000 = 270,000\ [EUR] \tag{16}$$

$$benefit = 9.9 \times 8000 = 79,200\ [EUR] \tag{17}$$

$$ROI = \frac{benefit - cost}{cost} = -0.71 \qquad BCR = \frac{benefit}{cost} = 0.29 \tag{18}$$

Interestingly, in a span of ten post-analyses during one release cycle, the lightweight solution proved considerably more profitable than the advanced one (ROI 1.76 to -0.71 and BCR 2.76 to 0.29, respectively). However, as we do not assume an end date of the project (product life cycle does not decline or phase-out [43]), we should include more software release iterations in the calculation. Nevertheless, after 100 positives were detected by the ML SDP solution over ten cycles of predictions, the lightweight scenario is still more lucrative than the advanced one (ROI 3.44 to 0.96 and BCR 4.44 to 1.96, respectively). After 100 cycles (in our bi-weekly cadence, this is almost four years), the profitability of the lightweight and advanced solutions becomes more comparable (ROI 3.73 to 3.51 and BCR 4.73 to 4.51, respectively). The advanced solution becomes more profitable after 151 cycles (5.8 years). Mentioned scenarios are available in Table 9 and Appendix A to enable further analysis.

In summary, the resulting ROI and BCR mainly rely on:

- Initial investment - the most considerable portion of the solution cost is the initial spending needed to build a framework. Depending on how much we want to spend, we might aim to obtain more or less effective predictions; however, from an economic perspective, fighting for marginal gains may be futile in non-safety-critical systems. Thus, building a custom, advanced solution may not be the best choice in vivo, and practitioners must carefully evaluate such a decision.
- Lifespan of the SDP solution - the longer the solution is used, the better profitability can be expected at the end (this is an intrinsic characteristic of the financial ratios [32]. Thus, the ML SDP mechanism that we discuss in his work does not include product decline and is treated as a long-term investment with consistent returns over each cycle.
- Cost of the escaped defect - the ratio between the cost of post-analysis and the cost of the escaped defect determines the pace at which the returns will outgrow the initial spending. In our case, the ratio is relatively positive; nevertheless, when drawing conclusions, it is necessary to remember that obtained results are based on assumptions and averages, showing only possible future projections.
- Prediction performance - we used the Precision metric, representing how many positives will turn out to be true positives after code review. Importantly, our results confirm the observation by Herbold [13] that even low-precision models can be cost-effective. Our lightweight scenario (Section 4.1, Table 9) shows that even if 50% of selected test cases will allow detecting escaped defects, the solution may be effective from the cost perspective. Moreover, other performance metrics can be used to support alternative means of cost modeling (minding that no direct relationships have been found [56]).

Conversely, as we possess the approximated costs for quality assurance and costs for defects, we can also calculate the ratio between those two values (C), which reveals if the prediction model would be cost-saving [13]. The ROI will be positive if the cost of developing the solution is lower than C multiplied by the number of successful iterations. Also, a range of values can be used if used approximations have low confidence.

That said, it is impossible to calculate the actual values of both costs and benefits precisely [6]. Therefore, a calculation sheet based on ROM estimations such as ours (Appendix A) is necessary to draw a spectrum of possible scenarios based on different approximations and assumptions. Consequently, resulting ROI and BCR values must be calculated and compared to make data-driven business decisions with the highest obtainable confidence. Naturally, the

Table 8: Exemplary calculation sheet for the advanced scenario.

Input data		Calculations						Results	
Man-hour cost [EUR]	50	Cost INIT	[h]	[EUR]	Cost QA	[h]	[EUR]		
Simulations per release	1	Develop & install	5,000	250,000	Post-analysis	30	1,500	Cost	270,000
No. of post-analyses	10	Training	100	5,000	No. of post-analyses		10		
Precision	0.99			255,000	No. of releases		1	Benefit	79,200
No. of releases	1						15,000		
								ROI	-0.71
Cost of escaped defect	[EUR]	Cost EXEC	[h]	[EUR]	Cost DEF		[EUR]		
Critical	20,000	Data gathering	0	0	Cost of escaped defect		8000	BCR	0.29
Critical %	10%	Modeling	0	0	No. of releases		10		
Major	10,000	No. of releases		1	No. of true positives		9.9		
Major %	30%			0			79,200		
Minor	5000								
Minor %	60%								
	8,000								

Table 9: Results per scenario.

Scenario	Precision	Analyses	Releases	ROI	BCR
Lightweight 0	0.50	10	1	0.53	1.53
Lightweight 1	0.90	10	1	1.76	2.76
Lightweight 2	0.90	10	10	3.44	4.44
Lightweight 3	0.90	10	100	3.73	4.74
Advanced 1	0.99	10	1	-0.71	0.29
Advanced 2	0.99	10	10	0.96	1.96
Advanced 3	0.99	10	100	3.51	4.51

more assurance can be gathered for the made assumptions, the more accurate and realistic the resulting scenarios will be.

5 DISCUSSION

In our baseline research [30], we used MCC as the main metric [59], AUC as supporting, and Recall, Precision, and Fbeta as additional data points not used for comparison. As argued by Tunkel and Herbold [56] there is no stable relationship between cost savings and any of the most widely used performance metrics for ML SDP. Therefore, as we did in this study, cost considerations must be handled separately and treated as an independent research question that needs to be investigated.

Second, according to VBSE [4], integrating value considerations into existing and emerging software engineering principles, practices, and projects is critical. Even more so, costs and benefits need to be meticulously evaluated and understood for a new technology introduction within any company. Therefore, the main success criterion for introducing ML SDP in a commercial setting should be the capability to direct quality assurance resources in a way that is more cost-effective than without it. Our examples in Section 4 show that ML SDP within Nokia 5G can be hugely profitable. Thus, we can positively answer our RQ1.

Answer to RQ1 (Is introducing ML SDP for defect prediction in an industrial Nokia 5G system-level testing process cost-effective for the company?): ML SDP can be cost-effective in complementing Nokia's existing 5G system-level test process.

Both ROI and BCR ratios are straightforward to interpret. A positive ROI value means that net returns are favorable and total returns are greater than the associated costs. A negative ROI indicates that the total expenses outgrow the possible returns. Similarly, if a project has a BCR greater than 1.0, it is expected to deliver a positive net present value to the investor; if less than 1.0, it will not be profitable. In our case, all results show high profitability of the investment. Even more so, both ratios can be used to compare the efficiency of different investment opportunities [9], and our results show that the lightweight scenario, requiring a less considerable initial investment but offering worse predictive performance, has a better profitability outlook in the longer term. Also, it is essential to note that the presented scenarios are very conservative to ensure the conclusions drawn on the potential benefits are sound. More realistic scenarios could increase the number of post-analyses and simulations per release to increase profitability further.

We obtained comparable outcomes to those of Hryszko and Madeyski [16]. Their in vivo case study in Volvo Group achieved an impressive ROI of 73 and a BCR of 74. Such results are overwhelmingly positive and show tremendous potential for the profitability of ML SDP solutions in vivo. Also, despite using different metrics, the few publications discussing the costs and benefits of ML SDP show similarly positive results [25, 33, 39].

One metric so far outstanding from the discussion is the time consumed by learners to compute the simulations. However, it is enough to point out that the cost associated with the time of calculations is negligible in our context. In the baseline research, the shortest calculation with the lightweight solution took one second, and the longest took 20 seconds for all six data sets combined. Notably, the longer-lasting calculations were usually more precise in predictions. Considering the low cost of execution, without additional external considerations and requirements, we suggest always prioritizing more extended computation time, increasing predictive performance. Secondly, in industrial environments, it is best to use several ML algorithms to build the models on each data set and choose the best-performing one [51], not prioritizing computation time as an essential factor.

As mentioned in this study, we focus on the monetary benefits of ML SDP predictions as input for challenging false positives after

the testing is done (or by highlighting areas of increased risk), consequently enabling quality assurance to limit the escaped defect ratio. The second benefit, which we do not explore at this point, is described in detail by Elsner et al. [9] in their industrial study on regression test optimization in the continuous integration process, discussing predictive effectiveness as omitting test cases as an unnecessary cost. This is a promising area to explore in further research in Nokia company. Specifically, the profitability of the ML SDP mechanism can be improved by predicting not only the areas of increased risk but also identifying areas with the lowest probability of containing defects. The low-risk areas could be deprioritized when defining the test scopes to substantially decrease the quality assurance costs. In Nokia, executing the test cases can be hugely expensive, as some more sophisticated 5G test environments cost millions of euros to build and maintain [50]. Furthermore, some test cases are very time-consuming, as specific stability scenarios take several hours or even days to execute. Thus, an additional verification mechanism for defect prediction can bring considerable operational savings, as shown in Section 4. Furthermore, based on the model results, decisions to omit low-risk areas not containing any defects, detect false positives limiting wasted effort, or offer confirmation that discovered faults require a software defect correction [30] further driving down the operation costs.

Importantly, increasing complexity and shortening the time-to-market schedules make avoiding more defects difficult [5, 14], and all companies worldwide seek new improvement methods for their quality assurance processes. Consequently, incorporating value considerations in software development needs to be treated as a dedicated and thought-through target, ensuring products adhere to requirements and meet financial goals [38]. Our ML SDP solution for augmenting the system-level testing will help achieve both aspirations. Specifically, SDP modeling supports decisions on software quality assurance resource allocation to the areas most likely to contain the highest number of defects. Thus, it helps to reduce the cost of the entire software development process. Namely, considering more future-oriented solutions applying ML SDP for the entire SDLC as described by Stradowski and Madeyski [49], $cost_{DEF}$ does not necessarily have to be the cost of customer escapes. Similar models can be built on particular test phases of the quality assurance process, where the benefit would result from the delta between the cost of executing and finding defects on different stages (where the later stages are more expensive [34]).

Notably, a commercial project lifespan can take many years in the wireless telecommunication industry, and the investment can go into hundreds of millions of EUR. From this viewpoint, the cost of developing and installing an ML SDP solution can become negligible in the long term. Even more so, if the delta between the benefit of a singular saved defect and its analysis cost is favorable, it is a question of how many times a new defect needs to be contained for the investment to be covered. As the answer to the posed research question RQ1 was positive, a more detailed question would be RQ1.1: How do the numbers of releases and post-analyses affect ROI in an industrial Nokia 5G system-level testing process? To answer the question, we have performed a sensitivity analysis to measure the degree to which the ROI result is sensitive to the variables representing the number of releases and post-analyses. A graphical representation of the calculations is shown in Figure 2. Analysis shows that increasing the numbers above a certain threshold starts to bring diminishing results, and balancing the frequency of running ML SDP is a critical aspect for the company to decide. However, we do not account here for the feasibility of post-analysis execution nor the maximum number of defects remaining to be discovered, which should also be considered while making the final decisions.

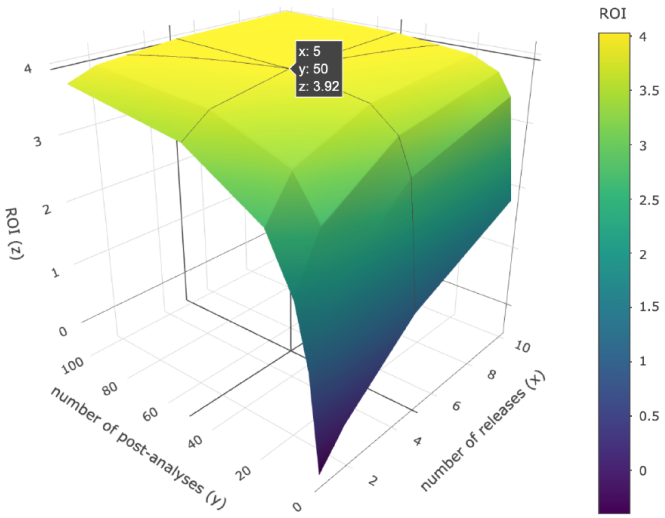

Figure 2: Sensitivity analysis of how the number of releases and number of post-analyses (interventions) affect ROI.

Last, it is worth noting that the telecommunication industry seems to be a high-yielding environment for introducing ML SDP in general. The most important factors that make it so welcoming are readily available vast amounts of data, high revenues, high cost of testing, high COPQ, availability of in-house expertise to create and maintain software solutions using ML, and finally, high-failure rate of new projects [54] which increases practitioners interest in new solutions. Consequently, there has been ample research applying ML SDP to this particular industrial environment:

- Khoshgoftaar et al. [27] - a joint telecommunication project of Nortel and Bell Canada.
- Ostrand et al. [36] - AT&T, one of the largest telecommunication providers in the world.
- Tosun et al. [55] - Turkcell, a large telecommunication company in Turkey.
- Monden et al. [33] - NTT, Japan's largest incumbent telecommunication operator.
- Jonsson et al. [21] and Rana et al. [39] - Ericsson, major telecommunications vendor based in Sweden.
- Bowes et al. [7] and Shippey et al. [46] - unnamed UK-based telecommunication companies.
- Wang and Khoshgoftaar [58] - unnamed large telecommunications software system.
- Hanmer and Mendiratta [12], as well as our work [30] - Nokia, major telecommunications vendor in Finland.
- Kang and Do [24] - telecommunication systems of Samsung Electronics.

5.1 Threats to Validity

We have identified several threats to the validity of our study and undertaken the following actions to mitigate their impact:

Construct validity: to answer our research question, we use an approach previously validated only in simulations rather than actual defect prediction models [13]; therefore, it is far from being established practice. Even more so, industrial contexts can vastly differ, and only case-by-case application of the same cost model across many projects can lead to comparable and reliable conclusions. Our research is one of the first steps toward this goal. For the cost metrics, we used standard economic ratios popularly used to measure cost-effectiveness across the world [32]. Also, as our study relies on the results of the baseline research, our construct poses the same limitations and threats to validity as the original — Stradowski and Madeyski [30]). Importantly, in our case, the models are built on already available and existing data, with no need for additional effort to prepare or pre-process. External validity: we based our research on a proprietary industrial data set and process; therefore, the generalizability of the results is limited. We can not claim any future efforts will achieve identical outcomes. However, we compared our results to similar studies, and the conclusions are complementary — ML SDP can be very profitable in industrial systems. Nonetheless, any profitability projection needs to account for different circumstances and factors. For example, suppose a company does not possess the internal resources to build an ML SDP framework. In that case, the cost will increase substantially, and if the predictive performance is not sufficient or the cost of poor quality savings is modest, the introduction of a similar solution may not be as profitable as in our case. Nevertheless, the framework and methodology are good benchmarks for future profitability analyses.

Internal validity: our cost-benefit evaluation is based on research results that could be more optimal from the predictive performance perspective. There is considerable potential for obtaining better results by employing techniques like feature analysis, hyperparameter optimization, more sophisticated algorithms, or ensembles. In such circumstances, the profitability could increase even further if the benefit of increased predictive performance surpasses the added cost needed to implement the upgrades. Naturally, this notion should still be validated with dedicated calculations similar to the ones described in this article. Importantly, assumptions described in Section 4 also constitute internal limitations and threats to validity. All are based on actual evaluations done within responsible functions in Nokia and are the best available approximations within our complex business reality. However, for the sake of confidentiality, the publication contains slightly modified values of similar magnitude (we explain each of such cases in Section 4). Hence, the actual values used are different but comparable in magnitude.

Conclusion validity: naturally, calculations based on approximations are not an exact science and do not give 100% confidence. Nevertheless, they offer an opportunity to make data-driven decisions on the profitability of investing in ML SDP in a commercial context. Our incurred cost and potential benefit approximations are based on actual projections from dedicated functions within the company, were validated by subject matter experts, and were purposefully undervalued to increase confidence even further. Secondly, with the proposed framework, such evaluation can be done quickly, considering different scenarios based on different input data. Therefore, to decide on future investment of ML SDP within the company, the conclusions are valid and sufficient for making business decisions on the following next steps. Lastly, as explained in Section 1.1, the cost-effectiveness of applying defect prediction models needs to be better explored in academia, so our comparison opportunities were relatively low.

6 CONCLUSIONS

The general cost model for software defect prediction proposed by Herbold [13] for preliminary cost-effectiveness evaluation has proved applicable to our scenarios. The calculated ROI was between 0.53 and 3.73 for the lightweight and between -0.71 and 3.51 for the advanced approach. Consequently, we have shown that a lightweight software defect prediction [30] is commercially feasible and, furthermore, may offer a higher return on investment than heavier but more prediction-effective solutions. Pursuing high performance is necessary; however, it must be measured against the required investment and decided on economic merit (value gained to outweigh the incurred cost [28, 32]). Thus, the minimum viable product [40] approach to validate the feasibility, make inroads, and gain confidence in future steps is an advantageous way to implement new technology in complex commercial environments.

Therefore, from the business perspective, demonstrating that the planned investment will bring monetary gain is critical. On the other hand, the cost-effectiveness of machine learning software defect prediction is a rarely explored subject in scientific research [51, 60]. Furthermore, even fewer studies utilized the proposed methods in case studies executed in vivo. Compared to the abundance of technical publications and new solution proposals, analyzing the profitability aspect of this emerging technology needs to be much more frequent to increase its commercial applicability [48]. Therefore, our work expands the existing knowledge of industry-validated solutions and provides the costs and benefits analysis that substantiates ML SDP's commercial potential.

Our real-world, business-driven examples show the meaningful benefits of introducing ML SDP on a large scale to complement the company's system-level test process. Consequently, ML SDP constitutes a positive business case [4] for Nokia to support further work towards commercial introduction within the quality assurance processes for its 5G products. Moreover, resulting outcomes and observations can also serve as a tutorial for other researchers and practitioners to overcome one of the critical factors inhibiting the transition of ML SDP from academic studies to standard practice.

ACKNOWLEDGMENTS

This research was conducted in partnership with Nokia (with Szymon Stradowski as an employee) and was financed by the Polish Ministry of Education and Science 'Implementation Doctorate' program (ID: DWD/5/0178/2021).

A SUPPLEMENTARY MATERIAL

The calculation sheet with examples and the source code of the visualization of how the number of releases and number of post-analyses affect ROI, see Figure 2, are available below: https://zenodo.org/doi/10.5281/zenodo.10995901.

REFERENCES

[1] 2010. IEEE Standard Classification for Software Anomalies. *IEEE Std 1044-2009 (Revision of IEEE Std 1044-1993)* (2010), 1–23. https://doi.org/10.1109/IEEESTD.2010.5399061

[2] Ishani Arora, Vivek Tetarwal, and Anju Saha. 2015. Open Issues in Software Defect Prediction. *Procedia Computer Science* 46 (2015), 906–912. https://doi.org/10.1016/j.procs.2015.02.161

[3] Gerardus Blokdyk. 2021. *Technology Change Management A Complete Guide - 2019 Edition.* 5STARCooks, Toronto, Canada.

[4] Barry Boehm. 2003. Value-Based Software Engineering: Reinventing. *SIGSOFT Software Engineering Notes* 28, 2 (2003), 3. https://doi.org/10.1145/638750.638775

[5] Barry Boehm and Victor R. Basili. 2001. Top 10 list [software development]. *Computer* 34, 1 (2001), 135–137. https://doi.org/10.1109/2.962984

[6] Jyoti Borade and Vikas R. Khalkar. 2013. Software Project Effort and Cost Estimation Techniques. *International Journal of Advanced Research in Computer Science and Software Engineering* 3 (2013), 730–739. https://www.ijtes.net/index.php/ijtes

[7] David Bowes, Steve Counsell, Tracy Hall, Jean Petrić, and Thomas Shippey. 2017. Getting Defect Prediction Into Industrial Practice: the ELFF Tool. In *(ISSREW)*. IEEE Computer Society, New York, NY, USA, 44–47. https://doi.org/10.1109/ISSREW.2017.11

[8] Davide Chicco and Giuseppe Jurman. 2023. The Matthews correlation coefficient (MCC) should replace the ROC AUC as the standard metric for assessing binary classification. *BioData Mining* 16, 1 (2023), 4. https://doi.org/10.1186/s13040-023-00322-4

[9] Daniel Elsner, Florian Hauer, Alexander Pretschner, and Silke Reimer. 2021. Empirically Evaluating Readily Available Information for Regression Test Optimization in Continuous Integration. In *ISSTA 2021: Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis* (Virtual, Denmark). ACM, New York, 491–504. https://doi.org/10.1145/3460319.3464834

[10] Norman Fenton and Martin Neil. 1999. A critique of software defect prediction models. *IEEE Transactions on Software Engineering* 25 (1999), 675 – 689. https://doi.org/10.1109/32.815326

[11] Vahid Garousi and Michael Felderer. 2017. Worlds Apart - Industrial and Academic Focus Areas in Software Testing. *IEEE Software* 34, 5 (2017), 38–45. https://doi.org/10.1109/MS.2017.3641116

[12] Robert Hanmer and Veena Mendiratta. 2023. Data Analytics: Predicting Software Bugs in Industrial Products. In *Data Analytics: Predicting Software Bugs in Industrial Products*. Springer, New York, 39–53. https://doi.org/10.1007/978-3-031-02063-6_3

[13] Steffen Herbold. 2019. On the costs and profit of software defect prediction. *IEEE Transactions on Software Engineering* 47, 11 (2019), 2617–2631. https://doi.org/10.1109/TSE.2019.2957794

[14] Kim Herzig, Michaela Greiler, Jacek Czerwonka, and Brendan Murphy. 2015. The art of testing less without sacrificing quality. In *37th IEEE/ACM International Conference on Software Engineering - Volume 1 (ICSE)* (Florence, Italy). IEEE Press, 483–493.

[15] Jarosław Hryszko and Lech Madeyski. 2017. Assessment of the Software Defect Prediction Cost Effectiveness in an Industrial Project. In *Software Engineering: Challenges and Solutions*. Advances in Intelligent Systems and Computing, Vol. 504. Springer, Poland, 77–90. https://doi.org/10.1007/978-3-319-43606-7_6

[16] Jaroslaw Hryszko and Lech Madeyski. 2018. Cost Effectiveness of Software Defect Prediction in an Industrial Project. *Foundations of Computing and Decision Sciences* 43, 1 (2018), 7–35. https://doi.org/10.1515/fcds-2018-0002

[17] Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren. 2019. *Automated machine learning: methods, systems, challenges*. Springer Nature, Berlin/Heidelberg, Germany. https://doi.org/10.1007/978-3-030-05318-5

[18] International Software Testing Qualifications Board. 2023. Advanced Level Test Manager (CTAL-TM). , 1-82 pages. https://www.istqb.org/certifications/test-manager Accessed: 02.12.2023.

[19] International Software Testing Qualifications Board. 2023. Foundation Level Syllabus v4.0. , 1-74 pages. https://www.istqb.org/certifications/certified-tester-foundation-level Accessed: 02.12.2023.

[20] Xiao-Yuan Jing, Haowen Chen, and Baowen Xu. 2024. *Intelligent Software Defect Prediction*. Springer Nature Singapore. https://doi.org/10.1007/978-981-99-2842-2

[21] Leif Jonsson, Markus Borg, David Broman, Kristian Sandahl, Sigrid Eldh, and Per Runeson. 2015. Automated Bug Assignment: Ensemble-based Machine Learning in Large Scale Industrial Contexts. *Empirical Software Engineering* 21 (2015). https://doi.org/10.1007/s10664-015-9401-9

[22] Magne Jørgensen. 2007. Forecasting of software development work effort: Evidence on expert judgement and formal models. *International Journal of Forecasting* 23, 3 (2007), 449–462. https://doi.org/10.1016/j.ijforecast.2007.05.008

[23] Magne Jørgensen and Martin Shepperd. 2007. A Systematic Review of Software Development Cost Estimation Studies. *IEEE Transactions on Software Engineering* 33, 1 (2007), 33–53. https://doi.org/10.1109/TSE.2007.256943

[24] Hongkoo Kang and Sungryong Do. 2024. ML-Based Software Defect Prediction in Embedded Software for Telecommunication Systems (Focusing on the Case of SAMSUNG ELECTRONICS). *Electronics* 13 (04 2024), 1690. https://doi.org/10.3390/electronics13091690

[25] Jonggu Kang, Duksan Ryu, and Jongmoon Baik. 2021. Predicting just-in-time software defects to reduce post-release quality costs in the maritime industry. *Software: Practice and Experience* 51, 4 (2021), 748–771. https://doi.org/10.1002/spe.2927

[26] Taghi M. Khoshgoftaar and Edward B. Allen. 1998. Classification of Fault-Prone Software Modules: Prior Probabilities, Costs, and Model Evaluation. *Empirical Software Engineering* 3 (1998), 275–298. https://doi.org/10.1023/A:1009736205722

[27] Taghri M. Khoshgoftaar, Edward B. Allen, John P. Hudepohl, and S.J. Aud. 1997. Application of neural networks to software quality modeling of a very large telecommunications system. *IEEE Transactions on Neural Networks* 8, 4 (1997), 902–909. https://doi.org/10.1109/72.595888

[28] John Leslie King and Edward L. Schrems. 1978. Cost-Benefit Analysis in Information Systems Development and Operation. *ACM Comput. Surv.* 10, 1 (1978), 19–34. https://doi.org/10.1145/356715.356718

[29] Michele Lanza, Andrea Mocci, and Luca Ponzanelli. 2016. The Tragedy of Defect Prediction, Prince of Empirical Software Engineering Research. *IEEE Software* 33 (2016), 102–105. https://doi.org/10.1109/MS.2016.156

[30] Lech Madeyski and Szymon Stradowski. 2024. Predicting Test Failures Induced by Software Defects: A Lightweight Alternative to Software Defect Prediction and its Industrial Application. (2024). https://madeyski.e-informatyka.pl/download/MadeyskiStradowski24.pdf (in reviews).

[31] Thilo Mende and Rainer Koschke. 2010. Effort-Aware Defect Prediction Models. In *2010 14th European Conference on Software Maintenance and Reengineering*. 107–116. https://doi.org/10.1109/CSMR.2010.18

[32] Alfred. Mill. 2016. *Economics 101: From Consumer Behavior to Competitive Markets–Everything You Need to Know About Economics*. Adams Media, New York, NY, USA.

[33] Akito Monden, Takuma Hayashi, Shoji Shinoda, Kumiko Shirai, Junichi Yoshida, Mike Barker, and Ken ichi Matsumoto. 2013. Assessing the Cost Effectiveness of Fault Prediction in Acceptance Testing. *IEEE TSE* 39 (2013), 1345–1357. https://doi.org/10.1109/TSE.2013.21

[34] Glenford J. Myers, Corey Sandler, and Tom Badgett. 2011. *The art of software testing*. John Wiley & Sons, Hoboken, New Jersey, U.S.

[35] Nokia Corporation. 2023. Nokia Annual Report 2022. https://www.nokia.com/about-us/investors/results-reports/ Accessed: 07.12.2023.

[36] Thomas J. Ostrand, Elaine J. Weyuker, and Robert M. Bell. 2005. Predicting the location and number of faults in large software systems. *IEEE Transactions on Software Engineering* 31, 4 (2005), 340–355. https://doi.org/10.1109/TSE.2005.49

[37] Jalaj Pachouly, Swati Ahirrao, Ketan Kotecha, Ganeshsree Selvachandran, and Ajith Abraham. 2022. A systematic literature review on software defect prediction using artificial intelligence. *Engineering Applications of Artificial Intelligence* 111 (2022). https://doi.org/10.1016/j.engappai.2022.104773

[38] Rudolf Ramler, Stefan Biffl, and Paul Grünbacher. 2006. *Value-Based Management of Software Testing*. Springer Berlin Heidelberg, Berlin, Heidelberg, 225–244. https://doi.org/10.1007/3-540-29263-2_11

[39] Rakesh Rana, Miroslaw Staron, Jörgen Hansson, Martin Nilsson, and Wilhelm Meding. 2014. A Framework for Adoption of Machine Learning in Industry for Software Defect Prediction. In *Proceedings of the 9th International Conference on Software Engineering and Applications (ICSOFT 2014)*. INSTICC, SciTePress, Vienna, Austria, 383–392. https://doi.org/10.5220/0005099303830392

[40] Eric Ries. 2009. Minimum viable product: a guide. http://www.startuplessonslearned.com/2009/08/minimum-viable-product-guide.html Accessed: 18.11.2023.

[41] Per Runeson and Martin Höst. 2009. Guidelines for conducting and reporting case study research in software engineering. *Empirical Software Engineering* 14 (2009), 131–164. https://doi.org/10.1007/s10664-008-9102-8

[42] Per Runeson, Martin Höst, Austen Rainer, and Björn Regnell. 2012. *Case Study Research in Software Engineering. Guidelines and Examples*. Wiley, Hoboken. https://doi.org/10.1002/9781118181034

[43] Antti Saaksvuori and Anselmi Immonen. 2008. *Product lifecycle management*. Springer Science & Business Media, Berlin/Heidelberg, Germany. https://doi.org/10.1007/978-3-540-78172-1

[44] D. Sculley, Gary Holt, Daniel Golovin, Eugene Davydov, Todd Phillips, Dietmar Ebner, Vinay Chaudhary, Michael Young, Jean-François Crespo, and Dan Dennison. 2015. Hidden Technical Debt in Machine Learning Systems. In *Advances in Neural Information Processing Systems*, C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett (Eds.), Vol. 28. Curran Associates, Inc., Norwich, England, 1–9. https://proceedings.neurips.cc/paper_files/paper/2015/file/86df7dcfd896fcaf2674f757a2463eba-Paper.pdf

[45] Martin Shepperd, David Bowes, and Tracy Hall. 2014. Researcher Bias: The Use of Machine Learning in Software Defect Prediction. *IEEE Transactions on Software Engineering* 40, 6 (2014), 603–616. https://doi.org/10.1109/TSE.2014.2322358

[46] Thomas Shippey, David Bowes, and Tracy Hall. 2019. Automatically identifying code features for software defect prediction: Using AST N-grams. *Information and Software Technology* 106 (2019), 142–160. https://doi.org/10.1016/j.infsof.2018.10.001

[47] Marilyn Simon. 2011. Assumptions, limitations and delimitations. https://studylib.net/doc/8312011/assumptions---limitations-and-delimitations Accessed: 01.12.2023.
[48] Szymon Stradowski and Lech Madeyski. 2023. Bridging the Gap between Academia and Industry in Machine Learning Software Defect Prediction: Thirteen Considerations. In *Proceedings of the 38th IEEE/ACM International Conference on Automated Software Engineering*. IEEE/ACM, Kirchberg, Luxemburg, 1098–1110. https://doi.org/10.1109/ASE56229.2023.00026
[49] Szymon Stradowski and Lech Madeyski. 2023. Can we Knapsack Software Defect Prediction? Nokia 5G Case. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE-Companion)*. IEEE/ACM, New York, 365–369. https://doi.org/10.1109/ICSE-Companion58688.2023.00104
[50] Szymon Stradowski and Lech Madeyski. 2023. Exploring the challenges in software testing of the 5G system at Nokia: A survey. *Information and Software Technology* 153 (2023), 107067. https://doi.org/10.1016/j.infsof.2022.107067
[51] Szymon Stradowski and Lech Madeyski. 2023. Industrial applications of software defect prediction using machine learning: A business-driven systematic literature review. *Information and Software Technology* 159 (2023), 107192. https://doi.org/10.1016/j.infsof.2023.107192
[52] Szymon Stradowski and Lech Madeyski. 2023. Machine learning in software defect prediction: A systematic mapping study. *Information and Software Technology* 155 (2023), 107128. https://doi.org/10.1016/j.infsof.2022.107128
[53] The 3rd Generation Partnership Project. 2021. 3GPP REL15. https://www.3gpp.org/release-15 Accessed: 19.04.2023.
[54] The Standish Group International, Inc. 2015. CHAOS REPORT 2015. https://standishgroup.com/ Accessed: 01.06.2023.
[55] Ayse Tosun, Ayse Bener, Burak Turhan, and Tim Menzies. 2009. Practical Considerations of Deploying AI in Defect Prediction: A Case Study within the Turkish Telecommunication Industry. In *PROMISE'09: Proceedings of the 5th International Conference on Predictor Models in Software Engineering*. ACM, New York, NY, USA, 1–9. https://doi.org/10.1145/1540438.1540453
[56] Steffen J. Tunkel and Steffen Herbold. 2022. Exploring the relationship between performance metrics and cost saving potential of defect prediction models. *Empirical Software Engineering* 27 (2022). https://doi.org/10.1007/s10664-022-10224-4
[57] Romi Wahono. 2015. A Systematic Literature Review of Software Defect Prediction: Research Trends, Datasets, Methods and Frameworks. *Journal of Software Engineering* 1 (05 2015), 1–16. https://doi.org/10.3923/JSE.2007.1.12
[58] Huanjing Wang and Taghi M. Khoshgoftaar. 2019. A Study on Software Metric Selection for Software Fault Prediction. In *2019 18th IEEE International Conference On Machine Learning And Applications (ICMLA)*. IEEE, New York, NY, USA, 1045–1050. https://doi.org/10.1109/ICMLA.2019.00176
[59] Jingxiu Yao and Martin Shepperd. 2021. The impact of using biased performance metrics on software defect prediction research. *Information and Software Technology* 139 (2021), 106664. https://doi.org/10.1016/j.infsof.2021.106664
[60] Hongyu Zhang and Shing-Chi Cheung. 2013. A Cost-Effectiveness Criterion for Applying Software Defect Prediction Models. In *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*. ACM, New York, NY, USA, 643–646. https://doi.org/10.1145/2491411.2494581

Received 2024-02-08; accepted 2024-04-18; revised 2024-02-08; accepted 2024-04-18