Preprint: 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 (ASE '23). Association for Computing Machinery, New York, NY, USA. DOI: soon. https://madeyski.e-informatyka.pl/download/StradowskiMadeyski23ASEpreprint.pdf

Bridging the Gap between Academia and Industry in Machine Learning Software Defect Prediction: Thirteen Considerations

1st Szymon Stradowski[1,2]
[1]*Mobile Networks, Radio Frequency*
Nokia
Wrocław, Poland
0000-0002-3532-3876

2nd Lech Madeyski[2]
[2]*Department of Applied Informatics*
Wrocław University of Science and Technology
Wrocław, Poland
0000-0003-3907-3357

***Abstract*—This experience paper describes thirteen considerations for implementing machine learning software defect prediction (ML SDP) in vivo. Specifically, we provide the following report on the ground of the most important observations and lessons learned gathered during a large-scale research effort and introduction of ML SDP to the system-level testing quality assurance process of one of the leading telecommunication vendors in the world — Nokia. We adhere to a holistic and logical progression based on the principles of the business analysis body of knowledge: from identifying the need and setting requirements, through designing and implementing the solution, to profitability analysis, stakeholder management, and handover. Conversely, for many years, industry adoption has not kept up the pace of academic achievements in the field, despite promising potential to improve quality and decrease the cost of software products for many companies worldwide. Therefore, discussed considerations hopefully help researchers and practitioners bridge the gaps between academia and industry.**

***Index Terms*—machine learning, software defect prediction, Nokia 5G, industry introduction, experience paper**

I. Introduction

Machine learning software defect prediction (ML SDP) has inspired academics and allured practitioners for over two decades [1], [2]. Despite having considerable commercial potential, in vivo applications have lagged behind academic research. The main reasons for such inadequacy have been identified as the divergent focus of academia and industry [3], practical futility of building defect prediction models using defect history [4], scarcity of available publications on costs and lessons learned [5], and many more [6]. Consequently, a sizable effort still needs to be put into developing ML SDP solutions to gain deserved recognition and prove their commercial value. Yet, with experience reports like this one, achieving the goal of widespread ML SDP solutions becomes considerably more feasible.

The recommendations presented below are based on our own experience and originate from an introduction of ML SDP to the context of system-level testing in the Nokia 5G product. The conducted survey initiating the project is described in a dedicated paper [7], followed by the implementation details [8] and a preceding industry challenge definition [9].

The practical significance of the discussed observations and conclusions was validated in a real industry context [10]. Particularly, our research example aims to complement the existing system-level test practices with additional ML SDP mechanisms within an extensive and complex software quality assurance process for cutting-edge wireless communication technology development in a software powerhouse. Importantly, Nokia practitioners are constantly looking for new opportunities to increase the quality and lower the costs of the software development life cycle (SDLC). We gathered our observations during one of such improvement projects aimed to enhance standard test practices in the company with an additional ML SDP mechanism to help direct test resources to the most defect-prone areas and consequently decrease the number of defects escaping to the customer.

Specifically, the 5G gNB system [11] is a grand and complex project with more than 60 million lines of code written in C/C++ language. Secondly, the developed product is challenging to test due to strict functional and non-functional requirements, thousands of potential software and hardware configurations, and the complexity of test environments and infrastructure. To overcome resulting difficulties, Nokia adheres to agile principles and state-of-the-art test processes, using the continuous development, integration, test, and delivery (CDIT) concept to develop its products [12] (see also Figure 1). The CDIT approach allows thousands of software engineers worldwide to add even the smallest increments to the main software line simultaneously. The entire test process is split into distinct phases based on the progress of component or system integration, similar to many other large-scale software products [13]. The following experience report focuses on the final phase - the entire, complete system and specific functionalities working end-to-end.

The underlying case study aimed to design, implement, and analyze the effectiveness of Machine Learning Software Defect Prediction in a real-world Nokia 5G system-level test environment. The analyzed data set is a collection of historical test process metrics from the test case repository for the Nokia 5G quality assurance process, containing almost 800000

unique results for more than 100000 test cases over five and a half months. Five relatively simple supervised machine learning algorithms were implemented to compare software defect prediction capability (using repeated 10-fold cross-validation) based on historical test process metrics. Due to the class imbalance problem, the Matthews Correlation Coefficient (MCC) was used for reliable performance measurements [14]–[16]. After completing a set of empirical evaluations using the R[1] language and mlr3 package[2], the best-performing solution was selected and compared with similar studies conducted in vivo. As a result, CatBoost and Random Forest performed the best in all tasks, followed by Light Gradient-Boosting Machine, Classification Tree, and Naïve Bayes. Even without tuning, CatBoost and Random Forest achieved satisfactory results, with differences that are not statistically significant; hence both models were recommended. Consequently, the study has successfully proven that software defects can be accurately predicted in vivo using limited data readily available within the Nokia 5G system-level test process, even using relatively simple learners and without resorting to sophisticated performance enhancement methods.

As the project progressed, we developed a report on made headways and encountered challenges. Furthermore, our observations are complemented by feedback from a selected group of Nokia experts and reflect the discussions observed during the planning, execution, and conclusion of the project. Thus, our experience report reflects practitioners' perspectives to help others achieve similar results in other commercial contexts.

Moreover, for the overall project progression, we have followed the global standard of the business analysis body of knowledge [17]. Lessons learned [18], guidelines, and instructions are essential for practitioners to establish confidence and help build initial inroads [19], [20]. Thus, we aim to bridge the identified gaps, make discussed mechanisms and practices widely available, and ease further adoption within the industry. Consequently, this publication provides a sequenced guideline regarding thirteen consecutive considerations each practitioner should account for when planning to introduce ML SDP in an industrial environment.

The paper is organized into four sections. Section I introduces the researched subject and context. Section II describes the thirteen considerations derived from our hands-on experience. Section III illustrates the importance of highlighted challenges and how they were prioritized in our specific context. Next, Section IV provides the identified threats to validity. Last, Section V offers the summary and conclusion.

II. Considerations

Below we describe the thirteen most impacting observations, lessons learned, and risks identified during our academic research efforts and industry introduction experience. Second, they constitute a comprehensive step-by-step progression and a holistic checklist that will increase the chances for success for similar ML SDP introductions. Third, the resulting discussion was reviewed together with practitioners participating in building the framework and developing a final commercial solution (to improve the quality and decrease the cost of system-level testing of Nokia's 5G gNB product), up to the point of a commercial implementation decision within the company.

The proposed checklist consists of thirteen steps provided below:

1) Collect requirements and set appropriate goals.
2) Build upon solid theoretical and practical foundations.
3) Consider the entire SDLC.
4) Conduct technology assessment and introduction.
5) Conduct risk analysis.
6) Choose appropriate data set.
7) Choose appropriate tooling.
8) Apply appropriate learners and performance metrics.
9) Build for interpretability.
10) Prepare a cost evaluation.
11) Manage stakeholders.
12) Plan for long-term evolution.
13) Plan project closure.

Consideration 1)

Collect requirements and set appropriate goals.

First and foremost, any business endeavor should start with gathering requirements [17]. A clear understanding of the stakeholder expectations the planned change must satisfy is critical to the project's success. If not done correctly, there is a high probability that the project deliverables will not meet the requirements and effectively end in failure. When the requirements are gathered and understood, only then the correct and precise goals can be defined by the project team. Goal setting is essential in any business project to provide a clear direction and purpose [17]. Moreover, clearly defined goals enable everyone involved in the project to work towards common objectives and allow progress to be tracked and measured against specified success criteria.

Preceding the ML SDP introduction, a widespread survey was launched among test practitioners within the company to elicit opinions on the current challenges within the system-level test process and to uncover further improvement opportunities [7]. As a result, 312 out of 2935 (10.63%) invited Nokia practitioners (representing management and software engineering functions from eight countries) have completed the questionnaire. Obtained results show that the same three challenges are seen as the most important and urgent: customer scenario testing, performance testing, and competence ramp-up. Accordingly, the challenges seen as the most difficult to solve were low occurrence failures, hidden feature dependencies, and hardware configuration-specific problems. The survey has also shown that defect prediction models are not widely used. Based on the analysis of the results by the

[1] https://www.r-project.org/
[2] https://mlr3.mlr-org.com/

company practitioners, the following high-level goals have been defined:

- The proposed solution must not disrupt the already existing quality assurance processes.
- Defect prediction efficiency must be on an acceptable level (not necessarily super high, but good enough).
- Built framework should utilize existing data and should be fully automated.
- There are no immediate timeline requirements for the project.
- Cost-effectiveness is important (positive return on investment (ROI)).
- Prediction modelling needs to allow interpretability.

Gathering requirements and setting proper goals is critical to the success of any business endeavor. By carefully defining and considering what is to be achieved, the project team can ensure that the project is correctly set up from the outset and does not diverge to unnecessary activities or change the business priorities.

Consideration 2)

Build upon solid theoretical and practical foundations.

Theoretical preparation before starting a project is important for several reasons: it helps to define the scope and goals of the project correctly, helps to ensure that the team has a thorough understanding of the problem they are trying to solve, enables effective planning by providing the information needed to create a detailed project plan, and improves overall decision-making. Consequently, exhaustive theoretical preparation helps ensure the project is well-planned and well-executed. Secondly, it helps the research community to validate shared results [21].

For our theoretical preparation, we have used the rapid review [22] to adapt the regular review process to fit software engineering practitioners' constraints. It helped to streamline knowledge transfer and provided decision-making support. Consequently, we have concluded several comprehensive literature reviews to understand the spectrum of possibilities and limitations for our implementation project (specifically, in the context of requirements and goals defined in Consideration 1)). Despite the overall scarcity of published research on ML SDP in vivo, there is a handful of valuable and practice-oriented papers having a tremendous impact on our research and underlying in vivo implementation. Second, reviewed studies allowed us to build the most considerable contribution of our work — a holistic end-to-end guideline describing the main aspects of ML SDP introduction in vivo. Lastly, appropriate theoretical preparation will also help hand over the final solution after integration into the standard quality assurance process within the company (Consideration 13)).

Specifically, the most impacting related works in terms of experience reports focusing on specific aspects of the process were highlighted below:

- Rana et al. [23] provide a framework for adopting machine learning for software defect prediction in the industry. The resulting paper focuses on the adoption process steps and lessons related to technology acceptance.
- Tantithamthavorn and Hassan [24] published an experience report on defect modeling during in vivo introduction. Their work describes encountered pitfalls and challenges based on their vast experience in the field.
- Melo et al. [25] developed a practical guide to support predicting change-prone classes based on a commercial software case study. The paper focuses mainly on the technical aspects of the introduction.

Second, it is worth underlining the importance of publications offering critiques of ML SDP practices that uncover concerns and research gaps to be closed. The works described below helped us make our considerations more complete and reliable:

- Fenton and Neil [26] offer a compelling critique of SDP models. The work raises awareness of model imperfections and risks related to the relationship between defects and failures. Importantly, the authors recommend using holistic, well-understood models instead of customized ones.
- Lanza et al. [4] describe considerable concerns towards SDP in terms of missing industrial practice impact of the academic achievements in the field, scalability difficulties, and cast doubt on evaluating approaches based on historical data.
- Garousi and Felderer [3] discuss different priorities and areas of focus between industry and academia. Depicted discrepancies cause difficulties in transitioning new solutions to commercial contexts and highlight relevant issues that must be mitigated to improve collaboration.

Last, there are several valuable secondary research efforts aggregating observations from numerous results in the field:

- Durelli et al. [27] published a systematic mapping study on machine learning applied to software testing. Authors identify the most frequently used test case generation, refinement, and evaluation algorithms.
- Pachouly et al. [28] published a systematic literature review on software defect prediction using artificial intelligence and analyzed the most commonly used data sets, validation methods, approaches, and tools.
- Stradowski and Madeyski [5] conducted a business-driven systematic literature review on industrial applications of software defect prediction using machine learning. They derived valuable conclusions on used methods, features, frameworks, data sets, costs, and learnings.

Consideration 3)

Consider the entire SDLC.

Considering the entire process perspective is another crucial aspect of introducing software quality assurance improvements. Large-scale software development requires scaling

methodologies to manage efficiently [13], and testing with a single-layered verification effort is rarely possible for grand products. However, most ML SDP studies are executed on singular test phases, and considering the whole software development life cycle (SDLC) is seldom explored [5], [9].

Suppose the implemented ML SDP solution proves to be successful and cost-effective. In that case, an adequately planned introduction can extend to all of the test phases within the life cycle, providing even more substantial saving potential to the company. When targeting lower-level phases, utilizing ML SDP with code metrics and code review can be sufficient [29], [30]. On higher levels, black-box test repository data can be used to enable test case selection and prioritization [31]. Consequently, each ML SDP instance will have its own characteristics, benefits, and inherent difficulties to plan for and overcome. On the other hand, each instance can benefit from the predictions of the previous phases. Moreover, the end goal can be a dedicated solution optimizing the overall predictions to accommodate the capacity and efficiency of particular phases in finding defects of different types.

Nokia uses continuous development, integration, and testing as the software development process in which applications are built continually throughout the entire life cycle [32]. Specifically, the goal is to continuously evaluate software quality on many levels of testing, providing feedback as quickly as possible, detecting more defects, and enabling faster deliveries at lower costs. Figure 1 shows the high-level visualization of the test process in the company. As a long-term aspiration, the ML SDP should work on all levels, provide individual predictions considering specific parameters and the interim capacity of particular test environments, and synergize in building general defect models for the entire product. Incorporating the holistic approach early in the project can save time and effort in future ML SDP implementations.

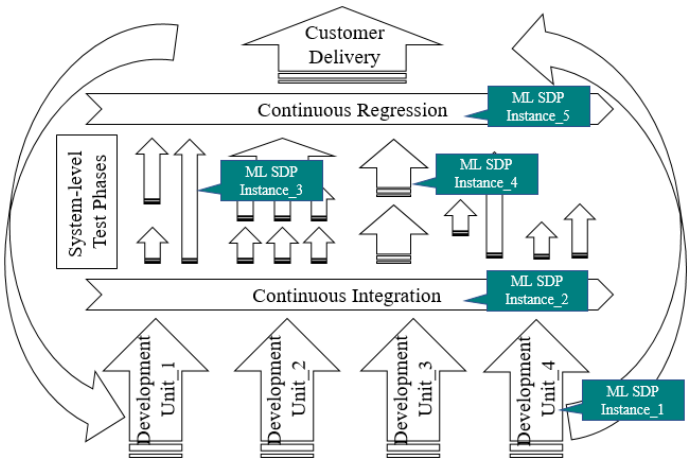

Fig. 1. High-level visualization of Nokia test process.

Consideration 4)

Conduct technology assessment and introduction.

The goal of the technology readiness level assessment (TRL)[3] is to optimize preparation for a new technology change within the company [33].

In the case of our system-level testing of 5G in Nokia, the new process that will be introduced aims to supplement the current quality assurance methods by introducing machine learning software defect prediction technology. Consequently, we have built an analysis and description of all nine TRL steps, provided evidence for each, and created a specific progression plan and a precise timeline for our ML SDP solution introduction. Noteworthy steps were the following:

- TRL 1: Basic principle observed - thorough theoretical preparation as described in Consideration 2).
- TRL 2: Technology concept formulated - consideration of the context and defining requirements and goals as in Consideration 1).
- TRL 5: Technology validated in a relevant environment - lightweight working solution to make initial inroads.
- TRL 6: Technology demonstrated in a relevant environment - a working pilot solution finalized by a demo showcase with main stakeholders.
- TRL 8: System complete and qualified - fully working and tested solution (currently ongoing).

Furthermore, our work in this aspect was greatly influenced by a publication by Rana et al. [23], providing a framework for adopting machine learning software defect prediction in the industry. First, the authors argue that an insufficient understanding of factors relevant to industrial practitioners is one of the main reasons for the low adoption of ML SDP in vivo. Second, Rana et al. develop a framework for explaining the adoption of ML for SDP in the industry based on the technology acceptance model and technology adoption frameworks. Third, they describe the characteristics of ML (perceived benefits, barriers, and tool availability), organizational characteristics (need and importance, satisfaction, familiarity with ML, and competence), and external environment (adoption in other industries and competition). Last, the paper also provides compelling instructions on how to use the proposed framework.

Naturally, a different method can be used to ease the new technology adoption within an existing ecosystem; however, executing appropriate project planning and change management tools that explicitly support introducing new methods, processes, and competencies is imperative [17].

Consideration 5)

Manage risks.

Risk assessment is the process of identifying and evaluating potential hazards that might endanger the success of a project, as well as analyzing what mitigations can be triggered when the hazard occurs. There are ample methods and techniques aimed to support the process [17]; however, we have built

[3]https://www.nasa.gov/directorates/heo/scan/engineering/technology/technology_readiness_level

our risk management framework in accordance with ISO 31000:2018[4] guidelines. We also depict an excerpt of our analysis in Table I.

TABLE I
EXAMPLE RISK ANALYSIS.

Risk#	Risk name	Impact (3-1)	Probability (4-1)	Risk value	Response
Risk 1	Lack of available resources	3	4	12	Mitigate
Risk 2	Lack of needed competence	2	2	4	Avoid
Risk 3	No visible containment gain	3	1	3	Avoid
Risk 4	Tooling and license unavailability	3	1	3	Mitigate
Risk 5	Organizational changes	2	2	4	Accept
Risk 6	Resistance among practitioners	3	3	9	Transfer
Risk 7	Loss of critical data	3	1	3	Avoid

We evaluated each risk in terms of impact and probability. The resulting scores can be multiplied to calculate the risk value used to prioritize the effort being committed to response actions (we planned four response types to the risks: avoid, mitigate, accept, or transfer). Secondly, after the analysis, items must be monitored throughout the project duration. As time progresses, new developments may shed light on unknown risks or change their probability or impact on the project's success.

Notably, the initial risk identification and assessment effort can result in a preemptive actions list - tasks that are worth executing as soon as possible to mitigate future hazards. In our case, four actions emerged for instant execution. First, a potential lack of resources needs to be immediately planned into the project by gaining commitment from management and sponsors. Second, utilizing transformation managers and submitting the project to official improvement frameworks to gain additional support and buy-in. Third, the launch of an awareness and competence ramp-up campaign on ML needs to be started as soon as possible. Last, research data needs to be regularly backed up on online servers.

One might consider a more lightweight or heavier approach based on the criticality of the business context for the implementation. Also, our risk examples are not exhaustive and are highly dependent on the project ecosystem. Nevertheless, the process enables better decisions and increases the project's chances of final success.

Consideration 6)

Choose appropriate data set.

Commercial companies can possess vast amounts of data that can be used for ML SDP. Regarding the entire SDLC (as in Consideration 3)), predictors for each test phase can be built on different data sets. Furthermore, each can be more effective when using specific features that need to be carefully selected [34].

The data we plan to gather and utilize will determine many aspects of the implementation. For example, conducting feature selection on the data set is important to optimize the process and increase prediction performance. Also, commercial data often suffer from missing samples, especially if the data set contains manually entered fields in the repository. Nonetheless, information carried by the observations with missing values can have predictive value [35] and needs to be planned for accordingly. Moreover, commercial data are often imbalanced [36] and noisy [37] and thus must be handled appropriately.

As mentioned, the data set we use is a collection of historical test process metrics gathered automatically from the main test case repository for the Nokia 5G quality assurance process, consisting of approximately 8000000 unique results of 100000 test cases executed over five and a half months from January 2021 until June 2021. Data were split by month into six separate files, allowing comparison of particular learners' effectiveness and the analysis of the differences between learners on different data subsets. However, as typical for industry data, there are considerable differences in the data quality between sets, which needs to be analyzed. Secondly, the data was a combination of both automatically and manually generated, creating unexpected inconsistencies. Therefore, a decision was made by the practitioners that all missing or incomplete entries have to be omitted during pre-processing. Last, as often is the case with real-world applications, the sets were severely imbalanced, which needs to be accounted for by how performance is measured (see Consideration 8)).

Hence, one of the most important aspects is planning for automation. Automated ML enables the process of gathering the data, pre-processing, simulation, and presenting the result to be acted upon to be fully automated [38]. Therefore, for online code metrics, change metrics, and software fault report repository to serve as real-time features, we need to plan accordingly and consider the limitations of interfaces towards the data repositories (for example, Jira[5] or Jenkins[6]).

Moreover, many additional concepts are expanding the opportunities to use ML SDP in different circumstances that can be considered — just-in-time (JIT) defect prediction [39], cross-project defect prediction (CPDP) [40], cross-company defect prediction (CCDP) [41] in homogeneous and heterogeneous defect prediction contexts (HDP) [42].

The most important breakthrough for each endeavor is to connect the data set with a use case. Naturally, we cannot freely select from all the possibilities in all circumstances, and project context, including stakeholders and confidentiality, can severely predetermine and limit options. Nevertheless, the outcome of the data set selection is very influential as it will determine tooling (Consideration 7)), learners (Consideration 8)), and interpretability (Consideration 9)).

[4]https://www.iso.org/standard/65694.html

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

[6]https://www.jenkins.io/

Consideration 7)

Choose appropriate tooling.

Choosing the framework to use for solution introduction in vivo needs to be based on the requirements and goals defined in Consideration 1). Identifying what we want to accomplish with the software tools we deploy will dictate the choice of the available software options. Tools that offer the required features need to be identified and compared to make a list of potential opportunities. Practitioners need to consider licensing options and available budget. Also, they need to evaluate the user interfaces and needs on how the data and models will be presented. Finally, we can compare the technology used in the tool with in-house competence availability. For example, if a company employs Python developers, using Python may be an obvious choice considering how widespread and helpful Python frameworks and libraries are.

Another essential factor to consider is the compatibility of the ML framework with existing data repositories within the company. Automated interfaces for data gathering to existing tools as described in Consideration 6)) need to be utilized or developed from scratch. Importantly, licensing and other legal aspects of the used framework and obtained results must be carefully verified.

Many options became available as machine learning gained in popularity over the last years, all with specific advantages and limitations, e.g.:

- R[7] is a programming language for data analysis and machine learning. Specifically, this framework was used (together with the mlr3 universe[8]) in the underlying research as it provided an object-oriented, unified interface to a wide range of ML models offered by a plethora of other useful R packages.
- TensorFlow[9] is an open-source machine learning library developed by Google, popular for its ease of use and flexibility.
- Keras[10] is a high-level neural networks API written in Python, designed to enable fast experimentation with deep neural networks.
- PyTorch[11] is a machine learning library developed by Facebook, popular for its dynamic computational graph and ease of use.
- Scikit-learn[12] is a popular machine-learning library for Python, designed to be simple and efficient, making it easy to use for both beginners and experts.
- Weka[13] is an open-source machine learning tool written in Java.
- Microsoft Azure ML Studio[14] is a cloud computing platform that provides many services for machine learning applications.
- Finally, there are several custom tools built by researchers to satisfy specific study requirements. A business-driven systematic literature review by Stradowski and Madeyski provides an analysis of such solutions [5].

Alternatively, a dedicated solution can be built (insourced or outsourced). However, considering the high quality of already available frameworks, good reasons need to exist for building something new. Creating customized solutions can cause high development costs, uncertainty of outcome, lack of dedicated support, or difficulties in scaling. On the other hand, custom ML SDP tooling may be specifically designed to meet the organization's current needs precisely and may give a better predictive performance [43]. Consequently, pros and cons need to be evaluated to make a data-driven decision.

In the underlying study, although an insourced solution was initially considered, the participating software developers decided on using the R language based on the results of a successful pilot and agreed project goals (Consideration 1)). Testers, as users of the solution, found the prediction models relatively easy to use and commented that the essential aspect is the ability to act upon the results. Last, management representatives appreciated the fact that no additional cost apart from man-hours was necessary to implement the solution, confirming that cost consideration remains crucial in industry adoption efforts [5], [44].

Consideration 8)

Apply appropriate learners and performance metrics.

Many ML techniques have been conceived [28], [45], each with different prediction effectiveness based on the circumstance. Notably, it is also claimed that no universal model could be applied to all data sets to develop effective solutions, as in the "no free lunch" (NFL) theorems [46]. Therefore, we advise employing various classifiers to select the best-performing ones for used data sets. Specifically, the most popular methods used in industry research are Linear Regression, Naive Bayes, Logistic Regression, Decision Trees, Support Vector Machine, K-Nearest Neighbor, and Random Forest [5]. Furthermore, different classifiers can find different defects that can be used to benefit the final test coverage [30].

Significantly, selection should be based on reliable performance metrics. There are many performance measures; however, considering the arguments and recommendations of Shepperd et al. [14], [15], as well as Chicco and Jurman [16], the main comparisons and conclusions should rely on Matthew's Correlation Coefficient (MCC). That said, assuming that the calculation of additional measures bears a minimal cost in most off-the-shelf frameworks, it is advisable to gather more metrics as it provides opportunities for further

[7] https://www.r-project.org/
[8] https://mlr3.mlr-org.com/
[9] https://www.tensorflow.org/
[10] https://keras.io/
[11] https://pytorch.org/
[12] https://scikit-learn.org/
[13] https://www.weka.io/
[14] https://ml.azure.com/

insight and understanding of the models. Specifically, a reliable dependency between any metric and costing is yet to be established [47].

Also, additional techniques such as normalization, outlier detection, feature selection, re-sampling and cross-validation, hyperparameter tuning, boosting, ensuring reproducibility, and conducting statistical analyses need to be considered depending on the context [28]. Before committing to highly effective but also very complex solutions such as deep learning [48], alignment with the solution requirements (Consideration 1)) is advised. Finally, chosen learners will impact the interpretability potential of built models (Consideration 9)).

In the underlying study, five relatively simple supervised machine learning algorithms were implemented and compared using repeated 10-fold cross-validation with the software defect prediction performance measured by the Matthews Correlation Coefficient (MCC) due to the data class imbalance. Results have shown that CatBoost and Random Forest performed the best in all executed tasks, followed by Light Gradient-Boosting Machine, Classification Tree, and Naïve Bayes. Importantly, even without tuning, CatBoost and Random Forest achieved satisfactory results, with differences that are not statistically significant. Consequently, both models were recommended, as they have successfully proven that software defects can be accurately predicted in vivo using limited data readily available within the Nokia 5G system-level test process.

Consideration 9)

Build for interpretability.

We advise the principle of limited trust towards any machine learning predictions, especially in business-critical contexts. Predictive technology can analyze vastly more extensive amounts of data than any human could, but it may also lack aspects like experience, ethics, or intuition. Therefore, it should be used cautiously and not relied upon completely in all circumstances. However, implementing explainable or interpretable AI/ML and human-in-the-loop (HITL) capabilities can considerably improve the usefulness or effectiveness of implemented solutions, as well as the chances of final project success [49], [50]. Also, several reasons can influence decisions behind building-in interpretability [51] potential to the ML SDP solutions:

- Transparency and trust - models that explain their predictions or decisions can help build trust in the model's effectiveness and decision-making process. This is especially important in safety-critical domains.
- Accountability and compliance - ensuring compliance with regulations and ethical standards is of critical importance. For example, the European Union's General Data Protection Regulation (GDPR)[15] includes a "right to explanation" that requires organizations to explain automated decisions that impact any individual.
- Debugging and improvement - interpretable models can help identify and correct errors or biases in the model. Consequently, it can also improve the effectiveness and reliability of the model, as well as help ensure that it is making decisions based on the intended criteria.
- Domain expertise - interpretable models can be easier to understand for domain experts who do not need to have an understanding of machine learning algorithms. Second, it can facilitate collaboration between data scientists and domain experts and enable better-informed decision-making.
- Knowledge discovery - interpretability can help uncover patterns and insights in data that may not be apparent through other methods. This can generate new hypotheses and insights that can drive innovation and improvement within the company.

Notably, high interpretability (high model transparency) typically comes at the cost of performance. If a company wants to achieve the highest performance but still wants to explain the ML model's behavior in human terms, model explainability may be the way to choose. In the case of complex back-box models, it could not be possible to fully understand how the inner mechanics impact the prediction. Fortunately, thanks to model-agnostic methods (partial or SHAP dependence plots, as well as surrogate models), it is possible to explain the model's behavior [52]. However, post hoc explanations may not be reliable and can be misleading; hence there is a trade-off in using both approaches [53].

During the implementation in Nokia, interpretability was approached as an additional value rather than a necessity. Involved practitioners wanted to know what factors drive ML algorithm decisions and compare them with their experience. After making some inroads using available R packages (e.g., iml, DALEX) supporting the interpretability or explainability of ML models, it was perceived as a valuable feature of the employed ML models. It helped to find inspiration for further improvement actions and opportunities to refine the software development process in a complementary way, i.e., not by further improvement of the ML models but by indicating the most important sources of the observed quality issues. Thus, the interpretability or explainability of ML models has the potential to deliver additional business value to the company from the perspective of management (global explanations), as well as developers or testers (global explanations can be used as guidelines, while local explanations can be used to determine which explanatory variables affect a model's prediction for a single observation). As a result, Nokia practitioners see this valuable opportunity created by an ability to interpret or explain models' predictions as a substantial added value.

Furthermore, the interpretability of machine learning is essential for ensuring the accuracy, fairness, and accountability of machine learning models, as well as gaining confidence among practitioners. Even if the goals for the implemented solution do not include interpretability, it is worth considering

[15] https://gdpr.eu/what-is-gdpr/

as a path for improvement as regulations and expectations might change in the future.

Consideration 10)

Prepare a cost evaluation.

Cost-effectiveness is a critical aspect of any project from the business perspective (adhering to the value-based software engineering (VBSE) [54] concept). Unfortunately, the cost and benefits of ML SDP are not yet well understood, and the scientific research on the topic is scarce [44]. Second, cost considerations must be handled separately from predictive performance [47]. Nevertheless, the publications that do exist provide several models for cost-benefit evaluations and show very promising results (e.g., [55]).

A general cost model we used to present the business case of ML SDP introduction in Nokia was based on a publication by Herbold [44]. The proposed model includes aspects such as the initial investment needed to set up the solution, expenses related to running the simulations, additional quality assurance effort, and escaped defects. For our purpose, we calculated the ROI[16] value, which showed a very positive outlook. Using popular economic ratios has helped to gain management support for the project.

Different scenarios can be evaluated based on the solution's expected efficiency and approximations of mentioned costs. In our case, Nokia possessed extensive analytics on software effort estimations, containment metrics, cost of poor quality, and similar values. Thus, it helped us tremendously to provide as accurate as possible evaluations. If such information is not readily available, approximations need to be estimated based on the best knowledge and subject matter expertise, resulting in a lower level of confidence. However, the facilitation of a data-driven decision process will substantially increase the understanding of challenges and risks that need to be considered.

In our example, the most important conclusions we derived were the following:

- Lightweight, easy, and fast solutions to gain initial inroads have shown a better cost-benefit ratio than more effective but heavier solutions.
- It was vital to include the solution's lifetime in the calculations. Naturally, planning to use the ML SDP solution for an extended time will be more attractive than short-term or one-time models.
- Cost of the escaped defect is a notable influencing factor on profitability. The more expensive one defect is, the more saving can be expected from ML SDP.
- The estimated profitability of the project ranged between ROI 3.5 and 4 depending on the projected lifetime and used assumptions, showing the implemented ML SDP solution has a positive monetary impact and can be cost-effective.

[16] https://www.investopedia.com/terms/r/returnoninvestment.asp

In summary, from the industry application perspective, demonstrating the investment needed by ML SDP will bring monetary return is critical. Without showing the expected gains, chances of success are diminished, as profitability is a vital success criterion in industry [54].

Consideration 11)

Manage stakeholders.

Stakeholder management is critical to any successful business endeavor, as stakeholders can considerably impact the project's outcomes [17]. The following steps were performed:

- The first step we took in stakeholder management was identifying all individuals or groups with an interest or stake in the project.
- Second, we analyzed the stakeholder needs, interests, and expectations regarding the project (also in the context of requirements gathering, as in Consideration 1)).
- In consequence, a management strategy was developed based on assessing stakeholder requirements and priorities, containing plans for engaging and communicating with relevant parties throughout the project.
- Finally, a resource plan with team roles and responsibilities has been created and approved.

Effective stakeholder management requires proactive engagement and communication, a deep understanding of stakeholder needs and interests, and a willingness to adjust strategies as the project progresses. By prioritizing stakeholder management, project teams can build stronger relationships with stakeholders and increase the likelihood of project success.

A few observations on managing stakeholders for ML SDP implementations are shared below:

- From the stakeholder management perspective, in vivo studies can be much more challenging than academic efforts as more people are involved. Researchers, practitioners, and management can have vastly different priorities and expectations for the project.
- Researcher bias should be carefully considered (as studied by Shepperd et al. [14]).
- Most participants were very optimistic about ML SDP and expressed satisfaction with the results and value provided by our lightweight solution.

In the end, project progress was much more straightforward, with efficient engagement, communication, and understanding of stakeholder needs and interests.

Consideration 12)

Plan for long-term evolution.

It is also worthwhile to plan for the long-term evolution of the ML SDP field in the coming years and evaluate possible scenarios and consider their feasibility in a specific context to seek further opportunities.

For example, the project in Nokia attempts to validate if the current approach to test planning done by test architects (TAs) can be supplemented by adding an ML SDP solution. Models can consider historical aspects, handle several programs simultaneously, and operate much quicker than humans; however, they can make many more mistakes and be less trustworthy in business-critical situations. Based on the achieved predictive performance and our cost-benefit analysis, there is solid justification for implementing such an ML SDP support mechanism for the test architects on system-level testing. Consequently, in the future, it may be feasible to substitute human interference completely and entirely rely on ML SDP to make the best test coverage decisions. Our results show this transition could be possible in the future after a full-scale verification of the hybrid model in vivo over several releases and a thorough evaluation from a cost perspective. In our example, the specific Nokia-based supposition for the described technology progression can be seen in Table II.

TABLE II
STEP-WISE EVOLUTION OF ML SDP INTRODUCTION.

	Step 1: Current state Test Architects	**Step 2: Our solution TAs + ML SDP**	**Step 3: Future Pure ML SDP**
Predictive accuracy	baseline	improved	improved
Auto data acquisition	limited	yes	yes
Historical data	limited	yes	yes
Report generation	manual	automatic	automatic
Multiple projects	limited	limited	yes
Running time	full-time	low	low
Installation cost	high	low	moderate
Maintenance cost	high	low	moderate

Continuously improved by the academic community, methods such as new algorithms, boosting, ensembles, hyperparameter tuning methods, cross-validation, outlier detection, and similar need to be combined with environment-focused mechanisms to tailor Nokia's needs after the solution is implemented and operational. Gradual effort towards better performance can, at some point, allow ML SDP to be a single test planning mechanism. Thus, acknowledging the long-term future and feasibility of pure ML SDP solutions has heavily impacted the projects' technology assessment (Consideration 4)) and cost evaluation (Consideration 10)) steps.

Consideration 13)

Plan project closure.

At the end of a project, several key tasks should be planned to ensure a successful conclusion and handover [17].

- First of all, review the project goals and requirements and measure the achievements against the success criteria (as in Consideration 1)).
- A handover of the responsibility towards designated practitioners needs to be done. It should be planned in advance, and the transfer of knowledge about ML and the used framework needs to be secured (Consideration 2)).
- Run a dedicated retrospective meeting to draw final conclusions on the process, gather lessons learned, and elicit stakeholder feedback (Consideration 11)).
- Reflect upon the next steps: conclude the technology acceptance model (Consideration 4)) and review the whole SDLC impact to identify further improvement opportunities (Consideration 3)).
- Finally, consider publishing the results and experience reports to benefit the wider community.

III. PRACTICAL IMPORTANCE

To provide further insight into our analysis, we have evaluated each of the presented considerations in two categories to create a control-impact matrix[17] — illustrated in Figure 2 and detailed scoring in Table III. We used this effective problem-solving and prioritization tool to organize the proposed solutions according to the level of impact on the project's success and the amount of influence the researchers can have on the identified factors. Each factor is evaluated on a scale from 1-low to 4-high, and the multiplication of those two values shows the relative priority.

- Impact - subjective measure reflecting how critical a particular consideration is to the end success of the ML SDP introduction.
- Control - subjective measure reflecting how much influence the project leaders can have over the outcome of the consideration.

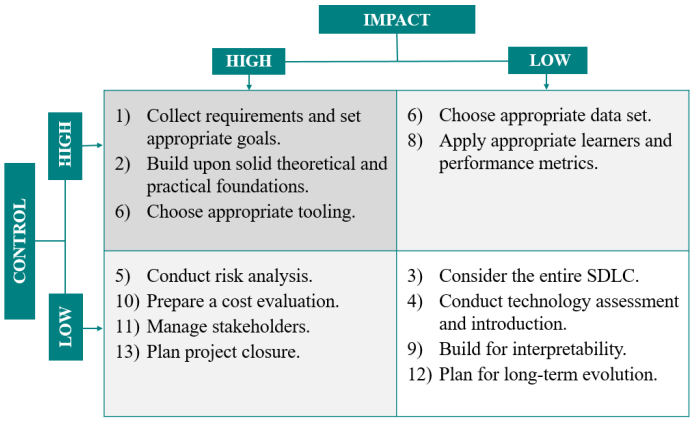

Fig. 2. Exemplary control-impact matrix.

The provided analysis illustrates which considerations have the highest impact and control, offering the most significant increase in the chances of final success at the lowest amount of time and effort spent:

[17]https://www.sixsigma-institute.org/Six_Sigma_DMAIC_Process_Analyze_Phase_Control_Impact_Matrix.php

- High Impact and High Control: 1) Collect requirements and goals, 2) Build upon solid theoretical and practical foundations, and 6) Choose appropriate data set.
- High Impact and Low Control: 5) Conduct risk analysis, 10) Prepare a cost evaluation, 11) Manage stakeholders, and 13) Plan project closure.
- Low Impact and High Control: 6) Choose appropriate data set, and 8) Apply appropriate learners and metrics.
- Low Impact and Low Control: 3) Consider the entire SDLC, 4) Conduct technology assessment, 9) Build for interpretability, and 12) Plan for long-term evolution.

TABLE III
CONTROL-IMPACT EVALUATION.

#	Consideration	Control	Impact	Priority
1)	Collect requirements and goals.	3	3	**9**
2)	Build upon solid foundations.	4	3	**12**
3)	Consider the entire SDLC.	2	1	2
4)	Conduct technology assessment.	2	1	2
5)	Conduct risk analysis.	2	3	6
6)	Choose appropriate data set.	3	2	6
7)	Choose appropriate tooling.	2	4	**8**
8)	Apply appropriate learners and metrics.	3	2	6
9)	Build for interpretability.	2	2	4
10)	Prepare a cost evaluation.	2	4	**8**
11)	Manage stakeholders.	2	3	6
12)	Plan for long-term evolution	2	2	4
13)	Plan project closure.	2	3	6

Note: above considerations are an example reflecting the particular context of our research in Nokia. Potential followers must prepare their analysis according to the specific circumstances they are working in, as the results may be very different. Also, there might be other external aspects inhibiting introduction progress — factors such as changing business priorities, economic slowdowns, or technology disruptions that have not been discussed in our report.

IV. THREATS TO VALIDITY

We have identified several threats to the validity [19], [20], [56] of our study:

- Construct validity: In the case of our experience report, several construct validity threats result from the underlying study conclusions [8]. Specifically, the technical aspect of the conducted underlying ML SDP research was based on approaches previously validated in several contexts and grounded state-of-the-art methods [8]. For example, our experience report relies on results using Matthews Correlation Coefficient (MCC) as a reliable performance measure [14]–[16]. However, to avoid the *mono-method bias* threat and to give an even broader understanding of the obtained results, several auxiliary measures like Area Under the Curve (AUC), Classification Accuracy (ACC), Recall, Precision, F-beta score (Fbeta) with $beta = 1$, were reported and analysed as well. Furthermore, repeated 10-fold cross-validation was applied to obtain even more reliable conclusions than classic 10-fold cross-validation. Finally, statistical tests and calculated robust non-parametric effect sizes [57] were conducted to grasp the statistical and practical significance of the presented results. *Mono-operation bias* does not seem to be a large threat since we study the outcome of the whole software development process on the system level of Nokia 5G; however, we acknowledge that other testing levels are beyond the scope of our study. *Interaction of different treatments* also does not seem to be a large threat as all change initiatives in the organization are monitored through a panel of experts. There is a threat of *interaction of testing and treatment*, as when we started our improvement project, we indicated that the quality is essential and needs to be improved. However, high quality is a widely recognized, long-term goal of Nokia; hence, we do not expect this threat to be large. There is also a threat of *restricted generalizability across constructs* as our intervention of introducing ML SDP impacted the test organization but also caused increased effort and extra resources. In consequence, we cannot fully distinguish whether the improvement was caused by the predictions or the allocation of dedicated resources. However, the increased effort and additional resources were kept as low as possible. The threat of *evaluation apprehension* seems to be limited as the experts from Nokia, we gathered the feedback from, were accustomed to providing feedback without exceptional stress, but we can not entirely rule out the human aspects influencing their stated opinions. Problems with inaccurate defect labels (due to the limitations of the SZZ algorithm and its implementations) are a severe threat to the validity of the state of the art of defect prediction [58]. Hence, the underlying project takes advantage of precise mapping of test cases to specific parts of the code, but this approach also has its own limitations, as not all failures become defects. As a result, the most important threat to construct validity seems to be an *inadequate preoperational explication of constructs*.
- Internal validity: The passage of time makes things better or worse, even without intervention. However, in a relatively short project period of a few months, we consider the *maturity* threat to be limited. There is also the *history* threat as some specific events may occur between measurements or collecting subsequent data sets as the software matures by itself, but again a relatively short span of time in our case limits the threat. The *instrumentation* threat was under control, as we confirmed that the instruments used to collect data did not change during the study.
  Importantly, assumptions in the underlying research 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 a complex business reality. However, for the sake of confidentiality, the publication contains slightly modified values of similar magnitude; hence, the actual values used in the company are different but comparable in magnitude. We did not find any other noteworthy

discrepancies regarding internal validity that needed to be addressed. We acknowledge that the identified considerations may be influenced by transient challenges faced in specific areas of the test process, particular capabilities or insights of the participating practitioners, and currently emphasized company priorities. However, such factors also reflect a standard industrial context and must be included in any ML SDP introduction.

- External validity: It is essential to understand the validity of the considerations beyond the case that was studied. There is a serious threat of *constructs, methods, and confounding* as the study takes advantage of the traceability of test cases to requirement and software components which is not always the case. The threat of *multiple treatment interference* is low as we did not introduce any other action simultaneously (we also monitored change actions within the company). The threat of *the real-world setting vs. experimental setting* is low as our setup is embedded within a typical company setting. Finally, there is a threat of *selection biases* as we were able to select only one company (Nokia) as a context for our research. We based our research on a proprietary industrial data set and process; therefore, the generalizability of the prediction performance results is limited, and we can not claim any similar efforts would achieve identical outcomes. However, the observations we have made and documented in the report, as well as the proposed generic approach to introducing ML SDP, should be generalizable to a large extent to any large-scale software project, including other telecommunication companies.
  Adapting particular considerations to specific contexts and using the control-impact matrix for prioritization should immensely improve the chances of the final success of any industry ML SDP implementation. Naturally, not all suggestions may be applicable to every situation (for example, '3) Consider the entire SDLC' may be superficial in a company that has a much simpler cycle than Nokia; however, it may still be worthwhile to deliberate how to incorporate a holistic approach early in the project). Last, the results of our exemplary control-impact matrix are not transferable without proper analysis and are specific to our project, as has been highlighted in Section III. Both control and impact evaluations need to be done by practitioners and subject-matter experts who understand the target environment's specifics enough to weigh their respective importance.
- Reliability: Threats to reliability are concerned with the extent to which the data and the analysis depend on the specific researchers. Specifically, the possibility of missing data and their treatment, as well as misinterpreting the observations pose meaningful threats to reliability. Although a significant effort was put into building the report, essential aspects still might not have been noticed. Therefore, the observations made as the project progressed were cross-checked with direct feedback from a group of Nokia practitioners responsible for the project to more accurately reflect the discussions that happened during the planning, execution, and analysis of the project. Furthermore, after the report's documentation and editing, the participating stakeholders' review was initiated, uncovering further misinterpretations and errors. Nevertheless, if different participating stakeholders and Nokia practitioners were involved, the results may not be exactly the same, which can be expected in industry studies. Last, to further reduce the reliability threat, a reproduction package for the underlying study [8] is available, following the reproducible research practice [59].

V. Conclusion

In summary, despite many challenges to overcome [6], [24], [28], defect prediction methods offer substantial profitability potential and attractive business cases [23], [29], [55]. Therefore, provided observations and highlighted risks can benefit other researchers and practitioners [18], [19] improve the chances of success for future introductions and help ML SDP become a standard procedure among software engineering practitioners.

Our experience report offers a holistic and sequenced approach to introducing a new ML SDP technology within the industry on a large software system. Consequently, the thirteen most important lessons learned we identified and discussed will help researchers and practitioners to benefit from our experience. We explain the insights and best practices that emerged, frameworks that can be used, risks to be mitigated, and further improvement possibilities for ML SDP industry adoption. Also, we used the control-impact matrix prioritization tool to decide where to put the primary focus during the introduction. In our case, the analysis revealed collecting requirements and setting appropriate goals, building upon solid theoretical and practical foundations, and choosing appropriate tooling to be the most impacting on the project outcome and in the project's team control [17]. Thus, they must be prioritized during the execution to increase the chances of the final success of the ML SDP solution we are implementing within the Nokia 5G system-level testing.

Likewise, we hope to see more practitioners publish their results and learnings on lowering the costs and improving the quality of developed software products using ML SDP. Increasing the number of successful implementation examples in published experience reports, with case studies and lessons learned [21], brings us closer to achieving widespread and effective defect prediction processes. Furthermore, available experience reports like the presented paper help the software engineering community to support building fully automated, reliable, and inexpensive ways to guide quality assurance resources more effectively for software companies worldwide.

Acknowledgment

This research was carried out in partnership with Nokia and was financed by the Polish Ministry of Education and Science 'Implementation Doctorate' program (ID: DWD/5/0178/2021). Both authors contributed equally to this work.

References

[1] C. Catal and B. Diri, "A systematic review of software fault prediction studies," *Expert Systems with Applications*, vol. 36, no. 4, pp. 7346–7354, 2009.

[2] S. Stradowski and L. Madeyski, "Machine learning in software defect prediction: A business-driven systematic mapping study," *Information and Software Technology*, p. 107128, 2023.

[3] V. Garousi and M. Felderer, "Worlds Apart - Industrial and Academic Focus Areas in Software Testing," *IEEE Software*, vol. 34, no. 5, pp. 38–45, 2017.

[4] M. Lanza, A. Mocci, and L. Ponzanelli, "The tragedy of defect prediction, prince of empirical software engineering research," *IEEE Software*, vol. 33, pp. 102–105, 11 2016.

[5] S. Stradowski and L. Madeyski, "Industrial applications of software defect prediction using machine learning: A business-driven systematic literature review," *Information and Software Technology*, vol. 159, p. 107192, 2023.

[6] I. Arora, V. Tetarwal, and A. Saha, "Open issues in software defect prediction," *Procedia Computer Science*, vol. 46, pp. 906–912, 2015.

[7] S. Stradowski and L. Madeyski, "Exploring the challenges in software testing of the 5G system at Nokia: A survey," *Information and Software Technology*, p. 107067, 2023.

[8] L. Madeyski and S. Stradowski, "A Lightweight Approach to Software Defect Prediction and its Industrial Application in Nokia 5G System-Level Testing," vol. 1, p. 1, 2023, (in reviews). [Online]. Available: https://madeyski.e-informatyka.pl/download/MadeyskiStradowskiPP.pdf

[9] S. Stradowski and L. Madeyski, "Can we Knapsack Software Defect Prediction? Nokia 5G Case," in *2023 IEEE/ACM 45th International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*. New York: IEEE/ACM, 2023, pp. 365–369.

[10] K. Petersen and C. Wohlin, "Context in industrial software engineering research," in *2009 3rd International Symposium on Empirical Software Engineering and Measurement*, 2009, pp. 401–404.

[11] The 3rd Generation Partnership Project, "3GPP REL17," 2021, accessed: 17.04.2023. [Online]. Available: https://www.3gpp.org/specifications-technologies/releases/release-17

[12] Nokia Corporation, "Nokia Annual Report 2022," 2023, accessed: 07.03.2023. [Online]. Available: https://www.nokia.com/about-us/

[13] H. Edison, X. Wang, and K. Conboy, "Comparing methods for large-scale agile software development: A systematic literature review," *IEEE Transactions on Software Engineering*, vol. 48, no. 08, 2022.

[14] M. Shepperd, D. Bowes, and T. Hall, "Researcher Bias: The Use of Machine Learning in Software Defect Prediction," *IEEE Transactions in Software Engineering*, vol. 40, no. 6, pp. 603–616, 2014.

[15] J. Yao and M. Shepperd, "The impact of using biased performance metrics on software defect prediction research," *Information and Software Technology*, vol. 139, p. 106664, 2021.

[16] D. Chicco and G. Jurman, "The matthews correlation coefficient (mcc) should replace the roc auc as the standard metric for assessing binary classification," *BioData Mining*, vol. 16, no. 1, p. 4, 2023.

[17] IIBA, *Babok: A Guide to the Business Analysis Body of Knowledge*. International Institute of Business Analysis, 2015, no. t.3, accessed: 17.04.2023. [Online]. Available: https://www.iiba.org/career-resources/a-business-analysis-professionals-foundation-for-success/babok/

[18] L. Rhodes and R. Dawson, "Lessons learned from lessons learned," *Knowledge and Process Management*, vol. 20, pp. 154–160, 07 2013.

[19] P. Runeson and M. Höst, "Guidelines for conducting and reporting case study research in software engineering," *Empirical Software Engineering*, vol. 14, pp. 131–164, 2009.

[20] P. Runeson, M. Höst, A. Rainer, and B. Regnell, *Case Study Research in Software Engineering. Guidelines and Examples*. Hoboken: Wiley, 2012.

[21] E. Barr, C. Bird, E. Hyatt, T. Menzies, and G. Robles, "On the shoulders of giants," in *Proceedings of the FSE/SDP Workshop on Future of Software Engineering Research*. ACM, 2010, pp. 23–28.

[22] B. Cartaxo, G. Pinto, and S. Soares, "The role of rapid reviews in supporting decision-making in software engineering practice," in *EASE '18: Proceedings of the 22nd International Conference on Evaluation and Assessment in Software Engineering 2018*. New York, NY, USA: ACM, 2018, pp. 24–34.

[23] R. Rana, M. Staron, J. Hansson, M. Nilsson, and W. Meding, "A framework for adoption of machine learning in industry for software defect prediction," in *9th International Conference on Software Engineering and Applications*, 2014, pp. 383–392.

[24] C. Tantithamthavorn and A. E. Hassan, "An experience report on defect modelling in practice: Pitfalls and challenges," in *Proceedings of the 40th International Conference on Software Engineering*, ser. ICSE-SEIP '18, 2018, p. 286–295.

[25] C. S. Melo, M. Cruz, A. D. F. Martins, T. Matos, J. M. da Silva Monteiro Filho, and J. C. Machado, "A practical guide to support change-proneness prediction," in *International Conference on Enterprise Information Systems (ICEIS)*, 2019, pp. 269–276.

[26] N. Fenton and M. Neil, "A critique of software defect prediction models," *IEEE Transactions on Software Engineering*, vol. 25, pp. 675 – 689, 10 1999.

[27] V. H. S. Durelli, R. S. Durelli, S. S. Borges, A. T. Endo, M. M. Eler, D. R. C. Dias, and M. P. Guimarães, "Machine learning applied to software testing: A systematic mapping study," *IEEE Transactions on Reliability*, vol. 68, no. 3, pp. 1189–1212, 2019.

[28] J. Pachouly, S. Ahirrao, K. Kotecha, G. Selvachandran, and A. Abraham, "A systematic literature review on software defect prediction using artificial intelligence: Datasets, data validation methods, approaches, and tools," *Engineering Applications of Artificial Intelligence*, vol. 111, p. 104773, 2022.

[29] A. Monden, T. Hayashi, S. Shinoda, K. Shirai, J. Yoshida, M. Barker, and K. ichi Matsumoto, "Assessing the cost effectiveness of fault prediction in acceptance testing," *IEEE Transactions on Software Engineering*, vol. 39, pp. 1345–1357, 2013.

[30] D. Bowes, T. Hall, and J. Petrić, "Software defect prediction: Do different classifiers find the same defects?" *Software Quality Journal*, vol. 26, no. 2, p. 525–552, 06 2018.

[31] S. Yoo and M. Harman, "Regression testing minimization, selection and prioritization: a survey," *Software testing, verification and reliability*, vol. 22, no. 2, pp. 67–120, 2012.

[32] International Software Testing Qualifications Board, *Foundation Level Syllabus*. ISTQB, 2018, accessed: 28.03.2023. [Online]. Available: https://www.istqb.org/certifications/certified-tester-foundation-level

[33] P. Raffaini and L. Manfredi, "Chapter 15 - project management," in *Endorobotics*, L. Manfredi, Ed. Academic Press, 2022, pp. 337–358.

[34] S. Khalid, T. Khalil, and S. Nasreen, "A survey of feature selection and feature extraction techniques in machine learning," in *2014 science and information conference*. IEEE, 2014, pp. 372–378.

[35] M. Kakkar, S. Jain, A. Bansal, and P. Grover, "Evaluating missing values for software defect prediction," in *2019 International Conference on Machine Learning, Big Data, Cloud and Parallel Computing (COMITCon)*, 2019, pp. 30–34.

[36] R. Longadge and S. Dongre, "Class imbalance problem in data mining review," *International Journal of Computer Science*, vol. 2, 05 2013.

[37] S. Kim, H. Zhang, R. Wu, and L. Gong, "Dealing with noise in defect prediction," in *ICSE '11: 33rd International Conference on Software Engineering*, 2011, pp. 481–490.

[38] F. Hutter, L. Kotthoff, and J. Vanschoren, *Automated machine learning: methods, systems, challenges*. Springer Nature, 2019.

[39] Z. Zeng, Y. Zhang, H. Zhang, and L. Zhang, "Deep just-in-time defect prediction: How far are we?" in *ISSTA 2021*. ACM, 2021, p. 427–438.

[40] S. Hosseini, B. Turhan, and D. Gunarathna, "A systematic literature review and meta-analysis on cross project defect prediction," *IEEE Transactions on Software Engineering*, vol. 45, no. 2, pp. 111–147, 2019.

[41] Y. Ma, G. Luo, X. Zeng, and A. Chen, "Transfer learning for cross-company software defect prediction," *Information and Software Technology*, vol. 54, no. 3, pp. 248–256, 2012.

[42] J. Nam and S. Kim, "Heterogeneous defect prediction," in *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*. New York, NY, USA: ACM, 2015, p. 508–519.

[43] T. Hall, S. Beecham, D. Bowes, D. Gray, and S. Counsell, "A systematic literature review on fault prediction performance in software engineering," *IEEE Transactions on Software Engineering*, vol. 38, no. 6, pp. 1276–1304, 2012.

[44] S. Herbold, "On the costs and profit of software defect prediction," *IEEE Transactions on Software Engineering*, vol. 47, pp. 2617–2631, 2019.

[45] G. Bonaccorso, *Machine Learning Algorithms: Popular algorithms for data science and machine learning*. Packt Publishing Ltd, 2018.

[46] D. H. Wolpert and W. Macready, "No free lunch theorems for optimization," *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 1, pp. 67–82, 1997.

[47] S. Tunkel and S. Herbold, "Exploring the relationship between performance metrics and cost saving potential of defect prediction models," *Empirical Software Engineering*, vol. 27, 09 2022.
[48] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, pp. 436–444, 2015.
[49] A. Kotriwala, B. Klöpper, M. Dix, G. Gopalakrishnan, D. Ziobro, and A. Potschka, "Xai for operations in the process industry-applications, theses, and research directions," in *AAAI Spring Symposium: Combining Machine Learning with Knowledge Engineering*, 2021, pp. 1–12. [Online]. Available: https://ceur-ws.org/Vol-2846/paper26.pdf
[50] L. von Rueden, S. Mayer, K. Beckh, B. Georgiev, S. Giesselbach, R. Heese, B. Kirsch, J. Pfrommer, A. Pick, R. Ramamurthy, M. Walczak, J. Garcke, C. Bauckhage, and J. Schuecker, "Informed machine learning – a taxonomy and survey of integrating prior knowledge into learning systems," *IEEE Transactions on Knowledge and Data Engineering*, vol. 35, no. 1, pp. 614–633, 2023.
[51] A. Barredo Arrieta, N. Díaz-Rodríguez, J. Del Ser, A. Bennetot, S. Tabik, A. Barbado, S. Garcia, S. Gil-Lopez, D. Molina, R. Benjamins, R. Chatila, and F. Herrera, "Explainable Artificial Intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI," *Information Fusion*, vol. 58, pp. 82–115, 2020.
[52] "Model explainability with aws artificial intelligence and machine learning solutions," Amazon Whitepaper, Tech. Rep., 2023. [Online]. Available: https://docs.aws.amazon.com/pdfs/whitepapers/latest/model-explainability-aws-ai-ml/model-explainability-aws-ai-ml.pdf
[53] C. Rudin, "Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead," p. 206–215, 2019.
[54] B. Boehm, "Value-based software engineering: Reinventing," *SIGSOFT Software Engineering Notes*, vol. 28, no. 2, p. 3, mar 2003.
[55] J. Hryszko and L. Madeyski, "Cost effectiveness of software defect prediction in an industrial project," *Foundations of Computing and Decision Sciences*, vol. 43, no. 1, pp. 7–35, 2018.
[56] M. Staron, *Action Research in Software Engineering - Theory and Applications*. Springer, 2020.
[57] B. Kitchenham, L. Madeyski, D. Budgen, J. Keung, P. Brereton, S. Charters, S. Gibbs, and A. Pohthong, "Robust Statistical Methods for Empirical Software Engineering," *Empirical Software Engineering*, vol. 22, no. 2, pp. 579–630, 2017.
[58] S. Herbold, A. Trautsch, F. Trautsch, and B. Ledel, "Problems with SZZ and features: An empirical study of the state of practice of defect prediction data collection," *Empirical Software Engineering*, vol. 27, no. 2, p. 42, 2022.
[59] L. Madeyski and B. Kitchenham, "Would wider adoption of reproducible research be beneficial for empirical software engineering research?" *Journal of Intelligent & Fuzzy Systems*, vol. 32, no. 2, pp. 1509–1521, 2017. [Online]. Available: https://doi.org/10.3233/JIFS-169146