*Preprint:* Szymon Stradowski and Lech Madeyski, "Can we Knapsack Software Defect Prediction? Nokia 5G Case" in IEEE/ACM 45th International Conference on Software Engineering: Companion Proceedings, pp. 365–369, 2023. DOI: 10.1109/ICSE-Companion58688.2023.00104 Preprint: https://madeyski.e-informatyka.pl/download/StradowskiMadeyski23d.pdf

# Can we Knapsack Software Defect Prediction? Nokia 5G Case

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

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

***Abstract*—As software products become larger and more complex, the test infrastructure needed for quality assurance grows similarly, causing a constant increase in operational and maintenance costs. Although rising in popularity, most Artificial Intelligence (AI) and Machine Learning (ML) Software Defect Prediction (SDP) solutions address singular test phases. In contrast, the need to address the whole Software Development Life Cycle (SDLC) is rarely explored. Therefore in this paper, we define the problem of extending the SDP concept to the entire SDLC, as this may be one of the significant next steps for the field. Furthermore, we explore the similarity between the defined challenge and the widely known Multidimensional Knapsack Problem (MKP). We use Nokia's 5G wireless technology test process to illustrate the proposed concept. Resulting comparison validates the applicability of MKP to optimize the overall test cycle, which can be similarly relevant to any large-scale industrial software development process.**

***Index Terms*—artificial intelligence, software defect prediction, software testing, continuous integration, software development life cycle, Nokia 5G.**

## I. Introduction

Software companies worldwide struggle to deliver high-quality products within the estimated time and budget. The success ratio seems to decrease with the size of the project and its complexity [1]. Furthermore, the telecommunications industry has the second lowest chance of a favorable outcome, only slightly higher than government initiatives. An excellent example of a grand, complex telecommunication system is the 5G technology developed by Nokia. The company employs approximately 90 thousand people in 130 countries [2]. Consequently, it faces considerable process challenges due to the tremendous scale and complexity resulting from the number of interfacing components, possible hardware combinations, used frequency spectrum, and the cooperation of several development units distributed worldwide.

Finding new opportunities to improve the quality and minimize the cost of the software development life cycle (SDLC) has been the goal of software engineering practitioners and researchers for decades [3]. One exceptionally promising concept is software defect prediction (SDP) using artificial intelligence (AI) and machine learning (ML) models that indicate the areas of the code where faults are most probable [4]. Unfortunately, no universal model can be applied for all data sets to develop accurate predictions due to the "no free lunch" theorems [5]. Furthermore, in vivo application of ML SDP lags academic research [6]. This paper aims to describe the challenge of scaling ML SDP for grand and complex software projects, using the example of Nokia 5G system-level testing. Second, we discuss the dream state of a general ML SDP solution that would address the whole SDLC in a real-world setting. Last, we invite researchers and practitioners to explore the described problem further.

Our contributions are the following:

1) Definition of Test Selection and Prioritization (TSP) Problem to be complemented by Software Defect Prediction (SDP) Problem (specific to any organizations that offer precise tracking of test cases to requirements and software modules). The outcome is $TSP_{SDP}$.
2) Formulation of the $TSP_{SDP}$ Problem as the Multidimensional Knapsack Problem (MKP). The outcome is $MKTSP_{SDP}$.

The presented ML SDP approach is complementary to the search-based software testing (SBST) [7]. SBST techniques are effective at generating tests with high code coverage [8], which may not be sufficient to create the best test strategy considering budget limitation without utilizing defect prediction [9]. Therefore, our proposal focuses on the synergy between various high-level testing phases by optimizing the results of several ML SDP models for the whole SDLC.

## II. 5G Test Challenge

The 5G gNB (or gNodeB) is a wireless base station responsible for establishing and maintaining the connection between the user equipment (UE) and the core network [10]. The whole 5G technology must adhere to strict 3rd Generation Partnership Project (3GPP) requirements [10] such as bandwidth, coverage, and latency, while at the same time offering complex mobility and carrier aggregation scenarios. Testing such functionalities at the early stages of product development focuses on the software and hardware configuration of the gNB or verifying the outgoing transmission characteristics using spectrum analyzers. On the system-level, test scenarios need to be verified end-to-end using real UEs, real core network, and a real over-the-air (OTA) interface. Many user stories like gNB reconfiguration, call set-up, max throughput, or stability can be sufficiently tested with simulators and

simple lab infrastructure. However, complex high-speed cell-edge scenarios can only be verified by flying a UE attached to a drone that circles a set of several gNBs or driving a van with multiple UEs through a dense urban environment [11]. Therefore, to effectively verify the overall system-level wireless telecommunication performance, testing must be done not only in simulators and conducted mode (by physically connecting the antenna to the user equipment) but also in real over-the-air conditions [12].

Second, there are thousands of potential software and hardware configurations with countless dedicated functional and non-functional requirements regarding robustness, operability, performance, power consumption, resilience, and similar. Such variation of possibilities poses a significant challenge in terms of planning and optimization of test scopes to be run during different test phases [13]. Furthermore, Nokia's customer base includes a multitude of customers with already live 5G networks [2]. Each brings specific needs and requirements, translated to thousands of features and software/hardware configurations. Consequently, the 5G system consists of a rough estimate of over 60 million lines of code in C/C++ language, with each new release introducing new and more advanced functionalities. Due to the complexity and size of the system, it is difficult to predict all possible interactions deterministically, and exhaustive testing is not feasible [14].

Nokia uses the Continuous Development, Integration, and Testing concept to build its products. Continuous Development allows thousands of developers to commit their code to a common software line (Trunk) as frequently as possible with the smallest possible increments. Continuous Integration merges new commits into functionalities on hardware as quickly as possible. Continuous Testing executes various automated test frameworks as part of the software delivery pipeline to obtain immediate feedback on the quality and allow most defects to be found quickly after the development and integration stages. Finally, Continuous Delivery prepares the final software package, including changes after commit, integration, and testing for production. Software builds are created in short cycles, ensuring that the product can be reliably released to the customer at any time.

Importantly, the CDIT adheres to the International Software Testing Qualifications Board (ISTQB) guidelines [14]. One of the main principles of test theory emphasizes the importance of testing early. The shorter the time between introducing and discovering the defect, the cheaper it is to find and correct. Therefore, each testing step needs to be efficient in finding the faults it has been designed to find and include all potential escapes from previous phases [15]. Each stage is also more expensive to execute as it executes more code, benchmarks over more extended periods of time, increases the number of repetitions, or replaces simulators with actual hardware to be more equivalent to the real life environment. Fig. 1 shows examples of test environments used in Nokia — from 1) server farms running automated tests on parts of the code and enabling continuous development, 2) real 5G gNBs tested in conducted mode, through 3) anechoic chambers used for testing the radio interface in OTA mode, to 4) massive walls with mounted antennas for advanced propagation and mobility scenarios. Planning the scope for each of the multiple phases is difficult and expensive.


Fig. 1. Examples of Nokia test infrastructure [2].

## III. Problem Definition

We treat SDP as it directly complements the concept of test selection and prioritization [16]. We apply this simplification as our case offers precise tracking of test cases to requirements and software modules. Therefore, we generalize that each failed test case pinpoints the defect precisely (in product or testware). Moreover, different test cases can fail the exact requirement when run in different environments (with different associated costs). Therefore, an important matter is deciding which test case to run to catch a predicted defect while reducing the cost of the whole process. For example, if the same defect may be caught on a simulator or full-scale 5G setup, it is imperative to catch it early [14]. However, for testing done in parallel, cost optimization for the same defect can be achieved by considering the real-time capacity of each phase's infrastructure.

Second, supporting the test case selection process can be achieved in many ways; manually by test architects, using search-based software engineering (SBSE) [17], ML-based solutions [18] including reinforcement learning [19], or future means yet to be discovered. In our example, we chose the ML solution as this is where the company's current interest lies. Each method has its pros and cons, and similarly, employing

ML-based solutions to the overall large-scale SDLC needs to be explored to design the best-performing approaches [20].

Currently, industrial ML SDP solutions focus on complementing the existing processes by employing a single oracle to optimize the containment in a singular test phase [21]. Despite the difficulties of in vivo validation [22]–[25], there is abundant research on the topic [18], [26]–[29], Moreover, with concepts like explainable artificial intelligence [30], just-in-time (continuous) defect or build outcome prediction [31]–[33], cross-project defect prediction [34], cross-company defect prediction [35] in homogeneous and heterogeneous defect prediction [36] settings, we have more tools to expand the problem to further dimensions. Therefore, attempts to define the big-picture challenge are important. Nevertheless, we have analyzed the existing primary research in the context of industry validation of SDP and have not found a similarly defined problem in the reviewed literature.

Most importantly, we wanted to build our problem definition on solid theoretical foundations to facilitate initial analysis. Therefore, we chose to compare it to an already well-known problem. From a big-picture perspective, testing such a grand system as the 5G gNB resembles the multidimensional knapsack problem (MKP) [37], [38]. The MKP is an NP-hard extension to the standard binary knapsack selection problem that has been a popular focus of study for decades. The goal is to find a subset of items (in our case, defects) that maximizes the total gain (or avoided cost). The main difference is that instead of having a single knapsack, there are multiple, each with distinct characteristics and constraints. Naturally, the subset of selected items can not violate capacity (lab infrastructure occupancy or the number of available testers) of each respective knapsack (test phase).

Fig. 2 shows a simplified representation of multiple layers of testing, each constituting an individual knapsack (test phase, described in Section II). Therefore, the upward arrows on the graph represent a group of testers responsible for maintaining, executing, and analyzing the results of a dedicated test scope reported in a common test repository and using a dedicated lab infrastructure. Second, the test scope reflects a set of requirements that must be validated at a specific time on a particular software build. Finally, defects found by each group are reported in a fault report repository to be corrected.

Naturally, reaching 100% phase containment in a grand and complex product is not feasible using any of the aforementioned techniques. Considering the ever-growing regression, the whole testing process aims to find a balance reflecting the desired quality and current business priorities [39]. At Nokia, this task is performed by a group of test architects analyzing the requirements and suggesting appropriate tests to be run in each phase. Therefore, our hypothesis is:

*Can ML-based SDP successfully complement test case assignment to particular test phases and provide sufficient explanation on made decisions?*

The MKP problem for ML SDP at the entire SDLC can be defined as follows.

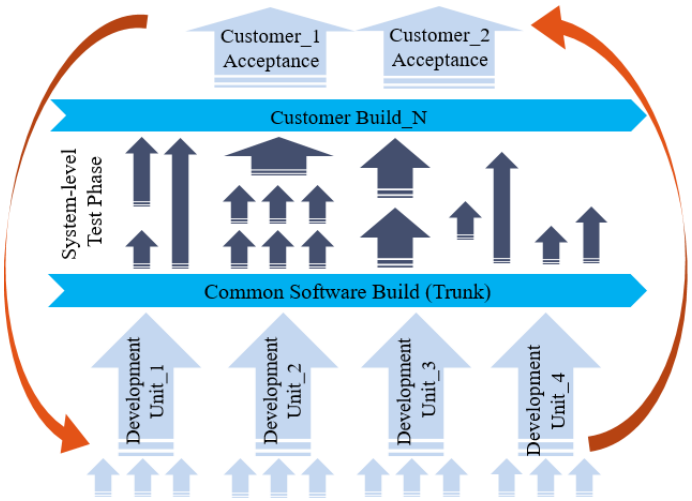


Fig. 2. Graphical representation of Nokia 5G test process.

(MKP) maximize:

$$z = \sum_{j=1}^{n} p_j x_j \tag{1}$$

Subject to:

$$\sum_{j=1}^{n} w_{ij} x_j \leqslant c_i, \quad i = 1, 2, ....., m \tag{2}$$

$$x_j \in \{0, 1\}, \quad j = 1, 2, ....., n \tag{3}$$

Where[1]:

| Variable | Explanation (see also Fig. 3) |
|---|---|
| z | Value of the number of items found in all knapsacks. → Value of the number of defects found during the whole SDLC. |
| n | Number of items. → Number of predicted defects. |
| p | Profits of each item. → Value gained by catching the defect or avoiding the cost of an escaped defect. In cases like Nokia, this can be tens of thousands of dollars; in safety-critical systems, it can be much more than what can be counted with money. |
| x | A vector of binary variables indicating whether an item → defect is selected. Based on this vector, a test suite for each knapsack is selected. |
| m | Resources, meaning the number of knapsacks → test phases or lab infrastructure elements. Depending on the development process, the value can reflect the whole cycle or a subset (for example, phases used inside of one of the Development Units, see Fig. 2). |
| c | The capacity of each knapsack. → The capacity of each test phase or infrastructure element (together with specific containment characteristics, cost of execution, etc.). |
| w | Resources consumed from each resource i. → Effort for each defect finding (specifically with different logging and troubleshooting capacity, of flakiness of results). Specifically, each defect consumes more resources depending on in which knapsack it is found. |
| i | Counter for resources. → Counter for knapsacks. |
| j | Counter for items. → Counter for defects. |

Additionally, for the solution to be feasible in vivo, there are several necessary preconditions. For example:

- Understanding the characteristics of the test phases, e.g., capacity, cost of execution, cost of escape.

[1]There are numerous possibilities to expand the concept with new variables and adapt it to the specifics of the tested product. For example, the constraints can change over time, reflecting the temporary capacity of each phase. Second, it can also be expanded to other variations of the MKP, like the multidimensional multiple-choice knapsack problem [40] to include more than one constraint.

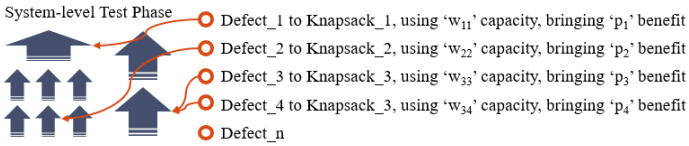


Fig. 3. Exemplary assignment of predicted faults to different test phases.

- Test repository, with automatic execution, reporting, and precise tracking of requirements to software modules.
- Online code metrics, change metrics, and software fault report repository to serve as real-time data sources.
- ML SDP framework integrated into company databases.
- Effective SDP oracles on each of the steps constituting the whole SDLC or its selected subset (see Fig. 1), combined into a general intelligence synergizing the entire process.
- Possibility to analyze, interpret, and act upon the results to modify in real time the test suite on affected levels.
- Technology and organizational readiness to implement AI-based solutions on a wide scale.
- A positive cost-benefit analysis [41].

## IV. Industrial Impact and Practical Importance

The complexity of the products we build can surpass our ability to test them efficiently. Consequently, companies release the software with less-than-desirable quality and cost (despite employing available scaling methodologies [42]).

Currently, the tedious task of planning test cases at each phase is usually done by groups of test architects that try to fit the scope into each test phase adhering to the purpose and characteristics of each test environment. Therefore, machine learning software defect prediction algorithms must aspire to achieve better results than a group of subject matter experts in terms of efficiency and cost. Moreover, in a commercial context, the solution should be able to explain its decisions to said experts with XAI [30] to attain sufficient confidence.

ML SDP offers plenty of learning algorithms, metrics, improvements, and configurations to choose from, and each satisfies its specific purpose better than others. Unfortunately, the academic effort is currently focused on singular test phases [21], despite such a scenario usually applying to only small-scale projects. Nokia is at the other end of the spectrum (followed by potentially more complex products like airplanes, autonomous cars, space shuttles, etc.). In our example, with hundreds of defects being found and corrected every day, the return on investment [41] of a solution addressing the whole SDLC and not a singular phase would be enormous.

Moreover, each defect escaped to the customer significantly increases the cost of poor quality. On the other hand, specific limited capacity testing, troubleshooting, or reproducing a particularly difficult fault can require a comparable or even more significant expenditure. Product quality management professionals estimate such costs based on expert knowledge, experience, and working assumptions. But no data-driven decisions can be made where particular defects can be caught in an automated and deterministic way for the whole process, as the overarching intelligence has yet to be created. High-level benefits of such a solution would be:

- Improved phase containment, resulting in increased predictability and more reliable capacity planning.
- Fewer defects escaping to the customer as different algorithms would detect different issues [43].
- Holistic approach to testing and gaining coverage synergy between various test phases.
- Less operational cost, faster time-to-market, and similar business-driven results of an efficient SDLC.
- Increased ability to balance the development process between prioritizing cost, time, and quality.

Treating the described issue as MKP is an imperfect simplification by modeling a complex and constrained commercial reality to a well-understood theoretical problem. Companies looking to adopt ML SDP techniques seek singular instances where algorithms are most effective and where the new methods are effective and cost-efficient [44]. In the following steps, maximizing the overall value of the defects predicted by ML SDP in each test phase by looking at the whole process can increase quality and minimize cost significantly, impacting its practical importance even further. Also, developed mechanisms will not only effectively predict the defects at each stage but should account for and utilize the intricacies of the whole machinery. For Nokia, it means considerable process improvements in the currently developed 5G and fast-approaching 6G — the next generation of wireless telecommunication network [45].

## V. Conclusions

We have defined a real-world problem for grand and complex software testing challenges in terms of a well-known MKP definition. Consequently, we can better understand the next step in scaling ML SDP, which will decrease the cost and increase the quality of software products (formulated as $MKTSP_{SDP}$). Moreover, the described approach has significant advantages over singular instances of ML SDP application. By employing MKP, we can utilize multiple theoretical and practical solutions like metaheuristics [38], domain solutions from the SBSE current [7], and in our case effective ML SDP algorithms to optimize defect detection among phases and benefit from varying capacity and cost of execution.

Defining the next steps for the field of ML SDP is of significant importance to practitioners and academics alike. Understanding where to concentrate future research and undertake the first attempts to govern several phases or even the entire SDLC can accelerate the pace of industry application, as it is more holistic and attractive from a business perspective [23], [46]. Thus, the first practical question to be raised is how the research community can create adequate data sets accurately reflecting the multi-stage test processes of large and complex software products to make further exploration possible.

## 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).

## References


[1] The Standish Group International, Inc., "Chaos report 2015," 2015.

[2] Nokia Corporation, "Nokia Annual Report 2021," 2021. [Online]. Available: https://www.nokia.com/about-us/investors/results-reports/

[3] M. Kramer, "Best practices in systems development lifecycle: An analyses based on the waterfall model," *Review of Business and Finance Studies*, vol. 9, pp. 77–84, 2018.

[4] N. Fenton and M. Neil, "A critique of software defect prediction models," *IEEE Transactions on Software Engineering*, vol. 25, no. 5, pp. 675–689, 1999.

[5] 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.

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

[7] M. Harman, Y. Jia, and Y. Zhang, "Achievements, open problems and challenges for search based software testing," in *Software Testing, Verification and Validation (ICST)*, 2015, pp. 1–12.

[8] S. Ali, L. C. Briand, H. Hemmati, and R. K. Panesar-Walawege, "A systematic review of the application and empirical investigation of search-based test case generation," *IEEE Transactions on Software Engineering*, vol. 36, no. 6, pp. 742–762, 2010.

[9] A. Perera, A. Aleti, M. Böhme, and B. Turhan, "Defect prediction guided search-based software testing," in *35th International Conference on Automated Software Engineering*. NY, USA: ACM, 2021, p. 448–460.

[10] The 3rd Generation Partnership Project, "3GPP REL15," 2021, accessed: 10.11.2022. [Online]. Available: https://www.3gpp.org/release-15

[11] 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.

[12] Y. Qi, G. Yang, L. Liu, J. Fan, A. Orlandi, H. Kong, W. Yu, and Z. Yang, "5g over-the-air measurement challenges: Overview," *IEEE Transactions on Electromagnetic Compatibility*, vol. 59, no. 6, pp. 1661–1670, 2017.

[13] S. Masuda, Y. Nishi, and K. Suzuki, "Complex software testing analysis using international standards," in *IEEE International Conference on Software Testing, Verification and Validation*, 2020, pp. 241–246.

[14] International Software Testing Qualifications Board, *Foundation Level Syllabus*. ISTQB, 2018, accessed: 28.12.2022.

[15] W. Afzal, R. Torkar, R. Feldt, and T. Gorschek, "Prediction of faults-slip-through in large software projects: An empirical evaluation," *Software Quality Journal*, vol. 22, 03 2014.

[16] R. Pan, M. Bagherzadeh, T. A. Ghaleb, and L. Briand, "Test case selection and prioritization using machine learning: a systematic literature review," *Empirical Software Engineering*, vol. 27, no. 2, 2021.

[17] M. Harman, Y. Jia, J. Krinke, W. B. Langdon, J. Petke, and Y. Zhang, "Search based software engineering for software product line engineering: a survey and directions for future work," *18th International Software Product Line Conference - Volume 1*, 2014.

[18] 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.

[19] M. Bagherzadeh, N. Kahani, and L. Briand, "Reinforcement learning for test case prioritization," pp. 1–1, 04 2021.

[20] S. Pradhan, V. Nanniyur, and P. K. Vissapragada, "On the defect prediction for large scale software systems – from defect density to machine learning," in *IEEE 20th International Conference on Software Quality, Reliability and Security (QRS)*, 2020, pp. 374–381.

[21] S. Stradowski and Madeyski, "Industrial Applications of Software Defect Prediction using Machine Learning: A Business-Driven Systematic Literature Review," *Information and Software Technology*, 2023.

[22] J. Hryszko and L. Madeyski, "Bottlenecks in Software Defect Prediction Implementation in Industrial Projects," *Foundations of Computing and Decision Sciences*, vol. 40, no. 1, pp. 17–33, 2015.

[23] M. Lanza, A. Mocci, and L. Ponzanelli, "The tragedy of defect prediction, prince of empirical software engineering research," *IEEE Software*, vol. 33, no. 6, pp. 102–105, Nov 2016.

[24] C. Tantithamthavorn and A. E. Hassan, "An experience report on defect modelling in practice: Pitfalls and challenges," in *40th International Conference on Software Engineering: Software Engineering in Practice*. NY, USA: ACM, 2018, p. 286–295.

[25] 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.

[26] 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.

[27] 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.

[28] 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.

[29] N. Li, M. Shepperd, and Y. Guo, "A systematic review of unsupervised learning techniques for software defect prediction," *Information and Software Technology*, vol. 122, p. 106287, 02 2020.

[30] 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.

[31] L. Madeyski and M. Kawalerowicz, "Continuous Defect Prediction: The Idea and a Related Dataset," in *14th International Conference on Mining Software Repositories (MSR)*, 2017, pp. 515–518.

[32] Z. Zeng, Y. Zhang, H. Zhang, and L. Zhang, "Deep just-in-time defect prediction: how far are we?" *Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2021.

[33] M. Kawalerowicz and L. Madeyski, "Continuous Build Outcome Prediction: A Small-N Experiment in Settings of a Real Software Project," in *IEA/AIE 2021*, ser. LNCS, vol. 12799. Springer, 2021, pp. 412–425.

[34] 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.

[35] 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.

[36] J. Nam and S. Kim, "Heterogeneous defect prediction," in *2015 10th Joint Meeting on Foundations of Software Engineering*, ser. ESEC/FSE. NY, USA: Association for Computing Machinery, 2015, p. 508–519.

[37] H. Kellerer, U. Pferschy, and D. Pisinger, *Knapsack Problems*. Springer Berlin Heidelberg, 2010.

[38] J. Puchinger, G. Raidl, and U. Pferschy, "The multidimensional knapsack problem: Structure and algorithms," *INFORMS Journal on Computing*, vol. 22, pp. 250–265, 05 2010.

[39] A. Bertolino, A. Guerriero, B. Miranda, R. Pietrantuono, and S. Russo, "Learning-to-rank vs ranking-to-learn: Strategies for regression testing in continuous integration," in *ACM/IEEE 42nd International Conference on Software Engineering*, ser. ICSE'20. NY, USA: ACM, 2020, p. 1–12.

[40] S. Laabadi, M. Naimi, H. El Amri, and A. Boujemâa, "The 0/1 multidimensional knapsack problem and its variants: A survey of practical models and heuristic approaches," *American Journal of Operations Research*, vol. 08, pp. 395–439, 09 2018.

[41] S. Herbold, "On the costs and profit of software defect prediction," *IEEE Transactions on Software Engineering*, vol. 47, no. 11, pp. 2617–2631, 2019.

[42] 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.

[43] 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, jun 2018.

[44] 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*. SciTePress, 2014, pp. 383–392.

[45] M. K. Shehzad, L. Rose, M. M. Butt, I. Z. Kovács, M. Assaad, and M. Guizani, "Artificial Intelligence for 6G Networks: Technology Advancement and Standardization," *IEEE Vehicular Technology Magazine*, vol. 17, no. 3, pp. 16–25, 2022.

[46] 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.