https://doi.org/10.31891/csit-2025-2-9
UDC 004.052

PIKH Iryna, BILYK Oleksii
Lviv Polytechnic National University

WORKLOAD BALANCING IN THE TEST CASE SCHEDULING: A METHEMATICAL APPROACH

Efficient scheduling of test cases is a critical task in environments where execution resources, such as testers or test environments, are limited and subject to individual availability constraints. In this paper, we propose a flexible and extensible mathematical model for optimizing test scheduling based on discrete time blocks. Each test case has a fixed duration and must be assigned to exactly one compatible tester. Testers, in turn, may be unavailable at specific time blocks due to pre-scheduled meetings or fixed breaks, such as lunch. The scheduling objective is to minimize the makespan, defined as the latest finish time among all scheduled tests. The model is formulated as a mixed-integer linear programming (MILP) problem that integrates testers' compatibility and availability constraints with task assignments into a unified framework. In contrast to models that assume testers are always available or disregard personal schedules, our method incorporates individual availability constraints for more realistic planning. The model is assessed on a synthetic scenario involving multiple testers with defined break times and varying task compatibility, and the resulting schedule is visualized with Gantt charts. The proposed formulation serves as a foundation for more advanced scheduling systems in quality assurance and resource-constrained testing workflows.

Keywords: software testing, test case scheduling, resource allocation, mixed-integer linear programming, constrained optimization.

ПІХ Ірина, БІЛИК Олексій
Національний університет «Львівська політехніка»

БАЛАНСУВАННЯ НАВАНТАЖЕННЯ ПРИ ПЛАНУВАННІ СЦЕНАРІЇВ ТЕСТУВАННЯ: МАТЕМАТИЧНИЙ ПІДХІД

Ефективне створення розкладу сценаріїв тестування є надзвичайно важливим завданням у середовищах, де ресурси для виконання, такі як тестувальники або тестові середовища, є обмеженими та доступними за індиввдуальними графіками. У цій статті запропоновано гнучку та масштабовану математичну модель для оптимізації розкладу тестування на основі дискретних часових блоків. Кожен тест-сценарій має фіксовану тривалість і повинен бути призначений одному сумісному тестувальнику. Тестувальники, у свою чергу, можуть бути недоступними у певні проміжки часу через заплановані зустрічі або фіксовані перерви, наприклад, обід. Метою планування є мінімізація загального часу виконання тестування, що визначається як найпізніший час завершення призначених тест-сценаріїв. Модель сформульовано як задачу змішаного цілочисельного лінійного програмування (MILP), яка об'єднує в єдиній структурі обмеження на сумісність та доступність тестувальників для призначення завдань. На відміну від моделей, які припускають постійну доступність тестувальників або не враховують їхні особисті розклади, наш метод враховує індивідуальні обмеження доступності для більш реалістичного планування. Для оцінки ефективності моделі використано синтетичні сценарії із декількома тестувальниками, визначеними періодами перерв і різною сумісністю завдань, а отриманий розклад подано через діаграми Ганта. Запропонована модель може слугувати основою для побудови більш складних систем планування у сфері тестування та контролю якості в умовах нестачі ресурсів.

Ключові слова: тестування програмного забезпечення, планування тестових сценаріїв, розподіл ресурсів, змішане цілочисельне лінійне програмування, оптимізація з обмеженнями.

Introduction

In modern software development lifecycles, testing plays a central role in ensuring product quality, system stability, and timely releases. As development processes accelerate under Agile and continuous integration frameworks, the pressure on test teams to deliver fast and thorough feedback increases. Testing must not only be accurate but also efficiently organized — particularly when multiple test cases must be assigned to limited testing resources under strict time constraints. Real-world testing environments introduce a number of challenges: testers and test environments are not always interchangeable, availability may be fragmented due to meetings or shifts, and time windows for testing are often constrained by sprint boundaries or release deadlines. As a result, manually constructing optimal test schedules that satisfy all technical and organizational requirements is both time-consuming and error-prone.

Previous research has explored test planning and resource allocation from various perspectives, including multi-sprint scheduling [1], metaheuristic test assignment [2], and architecture-driven reliability models [3]. Scheduling problems based on Mixed-Integer Linear Programming (MILP) have also been proposed for testing environments with complex resource constraints [4], [5], demonstrating their flexibility in encoding assignment, timing, and compatibility rules. Broader studies on test resource allocation [6], [7] emphasize the importance of optimizing test execution in terms of both efficiency and quality, while dynamic sprint replanning strategies [8] and modular software setups with change-point analysis [9] extend this line of research to more adaptive contexts. AI-driven solutions have also been investigated for resource-aware scheduling under uncertainty [10], incorporating intelligent heuristics and learning-based strategies. Several recent works address challenges closely related to our test scheduling model. Time-aware scheduling in shared-resource environments has been explored in cyber-physical systems, highlighting the importance of precise execution under resource constraints [11]. Optimization-based sequencing using metaheuristics like particle swarm optimization targets similar goals of cost and time efficiency [12]. Dynamic test selection approaches, including just-in-time execution [13] and fuzzy prioritization [14], demonstrate the value of context-sensitive planning, though they often lack formal guarantees. Constraint-guided scheduling has shown strong industrial applicability in managing complex test environments [15], and large-scale case management efforts emphasize the need for robust tooling [16]. Complementing these efforts, systematic reviews provide a foundation for evaluating prioritization strategies [17]. Our work builds on these insights by offering a practical, extensible MILP-based model that unifies compatibility, availability, and non-overlap constraints to achieve balanced, time-efficient test schedules.

Despite these contributions, many existing models either assume continuous resource availability or do not explicitly incorporate structured calendars, shared breaks (such as lunch), or time-discretized execution windows. In practice, however, such factors are unavoidable. To address this, we propose a discrete-time MILP-based model that optimizes test case scheduling under personalized availability constraints. The model ensures that each test is assigned to a compatible tester in a way that respects all timing constraints while minimizing the overall makespan — the latest test finish time across all testers.

In this paper, we formally define the scheduling problem and present a MILP formulation that integrates resource-task compatibility, time discretization, and fixed unavailability periods. We validate our approach using a realistic synthetic scenario involving multiple testers, varied test durations, and non-overlapping availability schedules. The model is evaluated based on schedule compactness and total completion time, and results are visualized using block-based Gantt charts.

Problem Definition

We consider the problem of scheduling a set of software test cases over a working day, where each test must be assigned to a compatible tester within their available time. The objective is to minimize the makespan — the time at which the last test finishes — while ensuring no overlaps, respecting resource constraints, and accounting for structured unavailability such as meetings and lunch breaks.

Time is discretized into uniform blocks of size Δ (e.g., 15 minutes). The total number of blocks, denoted by T, depends on the duration of the working day. All tests are non-preemptive, meaning they must run to completion once started.

Let:

- I: Total number of test cases;
- K: Total number of testers;
- T: Total number of discrete time blocks in the scheduling horizon;
- d_i, $i = 1..I$: Duration of test case i, expressed in number of blocks;
- $c_{i,k} \in \{0,1\}$, $i = 1..I$, $k = 1..K$: Compatibility matrix — 1 if test case i can be executed by tester k, 0 otherwise;
- $A_{k,t} \in \{0,1\}$, $i = 1..I$, $t = 1..T$:Availability matrix — 1 if tester k is available at time block t, 0 otherwise (e.g., due to a fixed meeting or break);

We define binary decision variables $x_{i,k,t} \in \{0,1\}$, $i = 1..I$, $k = 1..K$, $t = 1..T$,: where $x_{i,k,t} = 1$ means that test case i is executed by tester k and starts at time block t. Additionally, we define $M \in R^{+}$ as the makespan, measured in minutes.

The model includes the following constraints:

- *Unique assignment*: Each test case must be scheduled exactly once
$$\sum_{k=1}^{K} \sum_{t=1}^{T} x_{i,k,t} = 1 \ \forall i = 1..I;$$
- *Compatibility*: Tests can only be assigned to compatible testers
$$x_{i,k,t} = 0 \ if \, c_{i,k} = 0 \ \forall i = 1..I, k = 1..K, t = 1..T;$$
- *Task duration feasibility*: No test may exceed the end of the working day
$$x_{i,k,t} = 0 \ \forall i = 1..I, k = 1..K, t \text{ such that } (t + d_i - 1) > T;$$
- *Tester availability*: No test case may be scheduled to start at a time that would cause it to exceed the scheduling horizon.
$$x_{i,k,t} = 0 \ if \, A_{k,\mathcal{T}} = 0, \mathcal{T} = t,..,t + d_i - 1, \forall i = 1..I, k = 1..K, t = 1..T;$$
- *No overlap*: A tester may run only one test at a time
$$\sum_{i=1}^{I} \sum_{t=1}^{T-d_i+1} \sum_{\mathcal{T}=t}^{t+d_i-1} x_{i,k,t} \leq 1 \ \forall k = 1..K;$$
- *Makespan definition*: the makespan must be greater than or equal to the end of any scheduled task
$$M \geq (t + d_i - 1) * \Delta * x_{i,k,t} \forall i = 1..I, k = 1..K, t = 1..T.$$

The objective is to minimize the overall makespan M, which corresponds to the completion time of the last scheduled test case. Thus, it is defined as
$$min M$$

By minimizing this value, the model encourages compact and efficient scheduling, improving test throughput and enabling faster integration or release cycles. This is especially valuable in time-sensitive development processes such as Agile sprints or continuous delivery pipelines.

Materials and Research Methods

This study applies a mathematical optimization framework to address the problem of scheduling test cases under resource constraints and individual availability limitations. The core methodology is based on a Mixed-Integer Linear Programming (MILP) formulation, which provides a rigorous and flexible means to encode scheduling decisions, compatibility constraints, and temporal limitations using a fully linear model.

To support the discrete execution of test cases, the working day is divided into a sequence of uniform time blocks of duration Δ (e.g., 15 minutes). All temporal aspects of the problem—such as test case duration, start and end times, and periods of unavailability due to meetings or breaks—are represented in terms of these blocks. This discretization allows the model to reflect real-world calendar constraints while maintaining computational efficiency.

The MILP model includes binary decision variables $x_{i,k,t}$, which indicate whether test case iii is assigned to tester k at time block t, and a continuous variable MMM representing the overall makespan, defined as the latest test completion time. The model enforces:

- unique assignment of each test case to one compatible tester;
- avoidance of overlaps in the schedule of any tester;
- exclusion of tasks from blocked periods (e.g., meetings, breaks);
- and minimization of the makespan.

All constraints and the objective function are expressed using linear relationships, enabling exact optimization with MILP solvers.

The model is implemented in Python using the PuLP optimization interface. We utilize the CBC solver (Coin-or branch and cut), an open-source MILP solver that integrates seamlessly with PuLP and supports both binary and continuous variables. Model construction and parameterization are fully automated. Feasible variable combinations are generated dynamically, based on test duration, tester compatibility, and availability. To interpret the resulting schedule, we generate visualizations using matplotlib and seaborn. The final output is rendered as a Gantt-style chart, with time plotted along the horizontal axis and testers along the vertical axis. Each test case is shown as a colored bar, labeled with its identifier, and fixed unavailability periods (e.g., lunch or meetings) are marked in gray. This provides an intuitive view of the scheduling outcome, workload distribution, and idle periods.

The proposed implementation supports configurable simulation parameters, enabling reproducibility and adaptation to various practical settings, such as full-day testing, sprint-bound scheduling, or load-balanced resource planning.

Experiments

To evaluate the effectiveness of the proposed scheduling model, we conducted two experiments with increasing complexity. In both cases, the goal was to assign test cases to testers in a way that respects fixed meeting times, avoids overlaps, and optimally utilizes the available workday. All tests were scheduled between 10:00 and 19:00 and split into 15-minute blocks. Each tester had a lunch break from 14:00 to 15:00, and additional fixed meetings that varied in number and timing.

Experiment 1: Two Testers with Long Tasks

In this scenario, 10 test cases of varying durations were distributed between two testers. Most tasks were compatible with both testers, although some were exclusive to one tester to introduce decision complexity. Each tester is also subject to availability constraints, including a mandatory lunch break and individual fixed meetings. The complete input data for this experiment is presented in Tables 1 and 2. Table 1 shows the test cases, their durations (in minutes), and the compatible testers. Table 2 presents the fixed meeting times for each tester, with all time values expressed in 24-hour format.

Table 1.

Description of the test cases for the first experiment: durations and compatibility with testers

Test Case	Duration (min)	Compatible Testers
Test A	135	Tester 1, Tester 2
Test B	60	Tester 1, Tester 2
Test C	90	Tester 1
Test D	120	Tester 1, Tester 2
Test E	90	Tester 2
Test F	75	Tester 1, Tester 2
Test G	75	Tester 1, Tester 2
Test H	75	Tester 2
Test I	60	Tester 1, Tester 2
Test J	45	Tester 1, Tester 2

Table 2.

Description of the testers fixed meetings for the first experiment

Tester	Event Start	Event End	Duration (min)
Tester 1	14:00	15:00	60
Tester 2	11:30	12:00	30
Tester 2	14:00	15:00	60
Tester 2	15:00	15:30	30

The resulting test schedule is illustrated in Figure 1. As shown, the optimizer respects all constraints and allocates tasks efficiently. Tester 1, with fewer interruptions, completes longer tasks and works until 19:00, while Tester 2, who has more frequent meetings, concludes by 18:15.

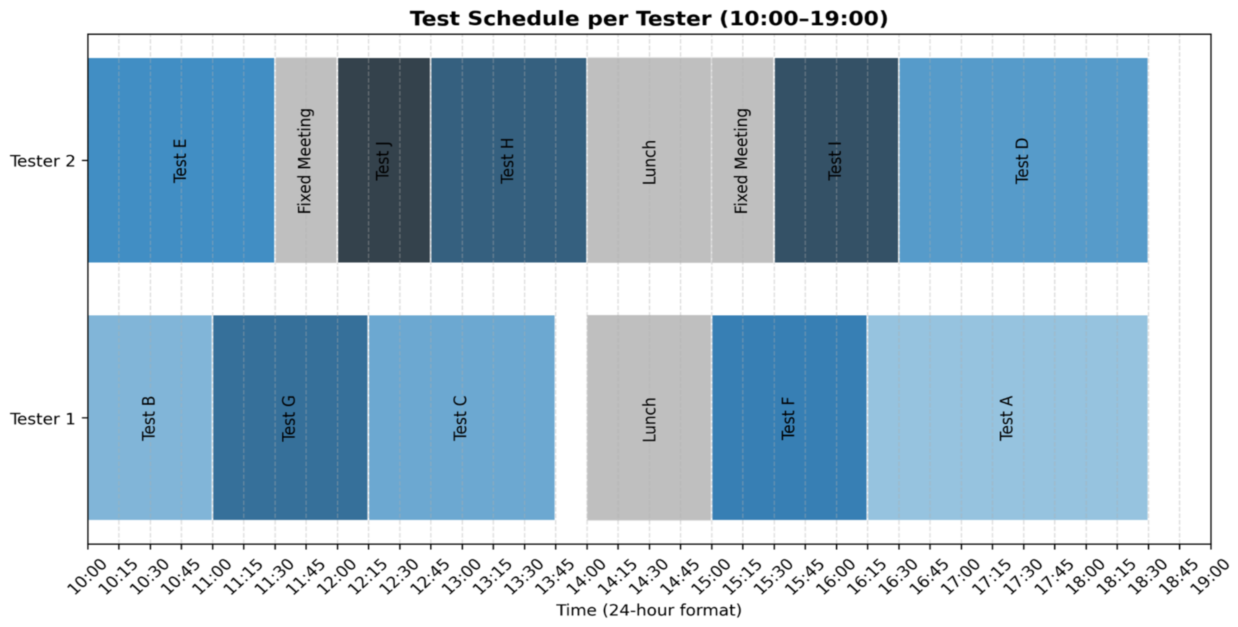

Figure 1. The first experiment. The optimized test schedule for two testers

Experiment 2: Three Testers with Denser Constraints

The second experiment increases the complexity to reflect more realistic, high-density team operations. It includes three testers and twenty-two test cases with diverse durations and a more intricate compatibility matrix. Many test cases are executable by multiple testers, though some remain exclusive. This setting mirrors real-world project scenarios where responsibility overlaps but expertise or permissions differ. Each tester has an individual schedule of fixed meetings throughout the day in addition to the shared lunch break from 14:00 to 15:00. The input configuration is summarized in Table 3 and Table 4.

Table 3.

Description of the test cases for the second experiment: durations and compatibility with testers

Test Case	**Duration (min)**	**Compatible Testers**
Test A	60	Tester 1, Tester 2
Test B	45	Tester 2, Tester 3
Test C	90	Tester 1
Test D	60	Tester 1, Tester 3
Test E	75	Tester 2
Test F	30	Tester 1, Tester 2, Tester 3
Test G	60	Tester 3
Test H	45	Tester 1, Tester 2
Test I	90	Tester 2, Tester 3
Test J	60	Tester 1, Tester 3
Test K	45	Tester 1
Test L	30	Tester 2, Tester 3
Test M	60	Tester 1, Tester 2, Tester 3
Test N	45	Tester 1
Test O	90	Tester 2
Test P	75	Tester 3
Test Q	30	Tester 1, Tester 2
Test R	60	Tester 2, Tester 3
Test S	45	Tester 2, Tester 3
Test T	30	Tester 1
Test U	60	Tester 2, Tester 3
Test V	45	Tester 1, Tester 2

Table 4.

Description of the testers fixed meetings for the second experiment

Tester	Event Start	Event End	Duration (min)
Tester 1	10:30	11:00	30
Tester 1	14:00	15:00	60
Tester 1	15:00	15:30	30
Tester 1	16:00	16:15	15
Tester 2	11:30	12:00	30
Tester 2	14:00	15:00	60
Tester 2	16:00	15:30	30
Tester 2	16:45	17:00	15
Tester 3	13:00	13:15	15
Tester 3	14:00	15:00	60
Tester 3	15:30	15:45	15

The optimized schedule is visualized in Figure 2, where the model demonstrates its ability to handle tightly packed constraints while maintaining balanced task allocation. All time constraints are satisfied, idle time is minimized, and resource usage is effectively distributed across testers without overlaps.

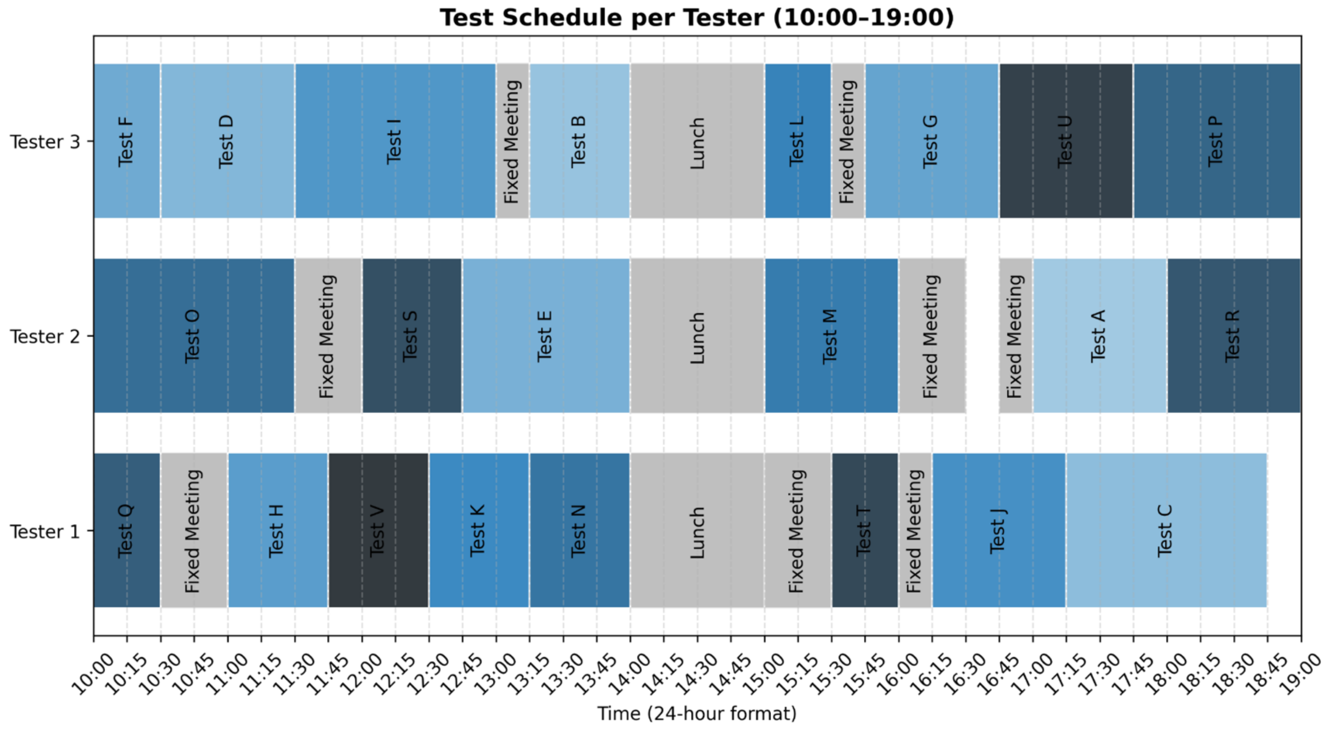

Figure 2. The second experiment. Optimized test schedule for three testers

Conclusions

This paper presented a mixed-integer linear programming approach for optimizing the scheduling of software test cases across multiple testers under realistic constraints. The model considers individual test durations, tester compatibility, and fixed availability windows, including meetings and lunch breaks. Time is discretized into fixed-length intervals, enabling precise formulation of scheduling rules and avoiding overlaps.

Two experiments were conducted to evaluate the model's effectiveness under different workload and availability scenarios. In both cases, the optimizer successfully generated feasible and compact schedules, respecting all compatibility and time constraints. The first experiment demonstrated the model’s ability to efficiently allocate longer test cases to two testers with minimal idle time. The second experiment scaled up the problem to include three testers and twenty-two test cases, along with denser meeting schedules. The model remained robust and produced a tightly packed schedule with balanced task distribution.

The results confirm that formal optimization techniques can significantly improve the efficiency of test planning in constrained environments. By automating task allocation while incorporating practical constraints, the approach offers a valuable tool for test managers seeking to minimize idle time and makespan in real-world agile or sprint-based workflows.

Future work may explore the integration of test case priorities, dynamic availability, and non-linear objectives such as cost or risk balancing. The model can also be extended to support adaptive re-planning in the presence of runtime changes or execution delays.

References

1. Kumar, R., & Sharma, D. (2019). Software Testing Resource Allocation and Release Time Problem: A Review. International Journal of Software Engineering and Its Applications, 13(2), 47–64. https://www.researchgate.net/publication/333016163
2. Xie, Y., & Zhang, H. (2020). Search-Based Optimization for the Testing Resource Allocation Problem. Proceedings of the ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM), 1–10. https://dl.acm.org/doi/10.1145/3383219.3383247
3. Pham, H., & Zhang, X. (2016). Optimizing Testing-Resource Allocation Using Architecture-Based Software Reliability Models. Software: Practice and Experience, 46(6), 751–768. https://doi.org/10.1002/spe.2356
4. Lopez, G., & Fernandez, J. (2018). Test Scheduling Using Mixed-Integer Linear Programming. International Journal of Advanced Computer Science and Applications, 9(11), 65–72. https://www.researchgate.net/publication/328614092
5. Cai, Y., & Yu, D. (2021). A Lagrangian Heuristic for Sprint Planning in Agile Software Development. Computers & Industrial Engineering, 157, 107291. https://doi.org/10.1016/j.cie.2021.107291
6. Garg, S., & Gupta, A. (2022). On the Testing Resource Allocation Problem: Research Trends and Perspectives. Journal of Systems and Software, 190, 111394. https://doi.org/10.1016/j.jss.2022.111394
7. Abelló, A., & Romero, O. (2020). Sprint Planning Optimization in Agile Data Warehouse Design. Journal of Intelligent Information Systems, 54(1), 65–84. https://doi.org/10.1007/s10844-019-00555-4
8. Pereira, C. R., & Figueiredo, A. A. (2022). Multi-Sprint Planning and Smooth Replanning: An Optimization Model. Journal of Systems and Software, 186, 111225. https://doi.org/10.1016/j.jss.2021.111225
9. Li, X., & Chen, J. (2020). A Study of Optimal Testing Resource Allocation Problem for Modular Software with Change Point. Journal of Software: Evolution and Process, 32(10), e2265. https://www.researchgate.net/publication/343675156
10. Zhou, Q., & Wei, Z. (2023). Software Testing Resource Scheduling Based on Artificial Intelligence. DiVA Portal – Uppsala University Publications. https://www.diva-portal.org/smash/record.jsf?pid=diva2%3A1741041
11. Mossige, M., Gotlieb, A., Spieker, H., Meling, H., & Carlsson, M. (2019). Time-aware Test Case Execution Scheduling for Cyber-Physical Systems. *arXiv preprint arXiv:1902.04627*. https://arxiv.org/abs/1902.04627
12. Iqbal, Z., Zafar, K., Iqbal, A., & Khan, A. (2024). On Test Sequence Generation using Multi-Objective Particle Swarm Optimization. *arXiv preprint arXiv:2404.06568*. https://arxiv.org/abs/2404.06568
13. Amoroso d'Aragona, D., Pecorelli, F., Romano, S., Scanniello, G., Baldassarre, M. T., Janes, A., & Lenarduzzi, V. (2022). *CATTO: Just-in-time Test Case Selection and Execution*. arXiv preprint arXiv:2206.08718. https://arxiv.org/abs/2206.08718arXiv
14. Karatayev, A., Ogorodova, A., & Shamoi, P. (2024). *Fuzzy Inference System for Test Case Prioritization in Software Testing*. arXiv preprint arXiv:2404.16395. https://arxiv.org/abs/2404.16395arXiv
15. Gotlieb, A., Mossige, M., & Spieker, H. (2023). *Constraint-Guided Test Execution Scheduling: An Experience Report at ABB Robotics*. arXiv preprint arXiv:2306.01529. https://arxiv.org/abs/2306.01529arXiv
16. Kok, T. (2025). *Optimizing Test Case Management for Large-Scale Projects*. TestMonitor Blog. https://www.testmonitor.com/blog/optimizing-test-case-management-for-large-scale-projects
17. Singhal, S., Jatana, N., Suri, B., Misra, S., & Fernandez-Sanz, L. (2021). Systematic Literature Review on Test Case Selection and Prioritization: A Tertiary Study. *Applied Sciences*, 11(24), 12121. https://doi.org/10.3390/app112412121

Iryna Pikh **Ірина Піх**	Doctor of Technical Sciences, Professor, Professor of Virtual Reality Systems Department, Lviv Polytechnic National University, Lviv, Ukraine, e-mail: iryna.v.pikh@lpnu.ua https://orcid.org/0000-0002-9909-8444 Scopus Author ID: 57208669246 https://www.scopus.com/authid/detail.uri?authorId=57208669246	Доктор технічних наук, професор, професор кафедри систем віртуальної реальності, Національний університет «Львівська політехніка», Львів, Україна.
Oleksii Bilyk **Олексій Білик**	Post-Graduate Student, Lviv Polytechnic National University, Lviv, Ukraine; e-mail: oleksii.z.bilyk@lpnu.ua https://orcid.org/0009-0002-1355-2333	Аспірант, Національний університет «Львівська політехніка», Львів, Україна;