EUROPEAN CONFERENCE

# Conference Proceedings

## VI International Science Conference «Scientific trends in the development of modern technologies and inventions»

October 06 - 08, 2025

Prague, Czech Republic

# SCIENTIFIC TRENDS IN THE DEVELOPMENT OF MODERN TECHNOLOGIES AND INVENTIONS

Abstracts of VI International Scientific and Practical Conference

Prague, Czech Republic
(October 06-08, 2025)

UDC 01.1

ISBN – 979-8-89814-229-2

The VI International scientific and practical conference «Scientific trends in the development of modern technologies and inventions», October 06-08, 2025, Prague, Czech Republic, 165 p.



The recommended citation for this publication is: Synytsia O., Shlapak H., Reus O. The influence of natural antioxidants on the active acidity of liver sausages. Abstracts of VI International Scientific and Practical Conference. Prague, Czech Republic. Pp. 11-13.

URL: https://eu-conf.com/en/events/scientific-trends-in-the-development-of-modern-technologies-and-inventions/

# ЗАСТОСУВАННЯ АЛГОРИТМІВ МАШИННОГО НАВЧАННЯ ДЛЯ ОПТИМІЗАЦІЇ ПРОЦЕСІВ ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

**Калашник Валерій Юрійович**
к.т.н., старший викладач
Київський Національний Університет Технологій та Дизайну

**Анотація**

У сучасному світі, де темпи розробки програмного забезпечення постійно зростають, критично важливим стає застосування ефективних та інноваційних методів забезпечення якості. Дана робота присвячена дослідженню можливостей інтеграції алгоритмів машинного навчання у процеси тестування програмних систем. Акцент зроблено на застосуванні алгоритмів для автоматичної генерації тестових даних, оптимізації тестових наборів та прогнозування дефектів, що дозволяє значно підвищити ефективність та точність тестування. Розглядається архітектура системи тестування, яка використовує моделі машинного навчання для аналізу історичних даних і поведінки системи, забезпечуючи більш глибоке розуміння потенційних "вузьких місць" і підвищуючи загальну надійність продукту. Представлена методика може бути інтегрована в освітні програми для підготовки майбутніх фахівців, здатних вирішувати складні завдання забезпечення якості на основі сучасних наукових досягнень.

**Вступ**

Процес тестування програмного забезпечення є невід'ємною частиною життєвого циклу розробки. Однак, у міру зростання складності програмних систем, традиційні методи тестування стають менш ефективними та вимагають значних ресурсів. Поява великих обсягів даних, пов'язаних з розробкою та експлуатацією ПЗ, відкриває нові можливості для застосування підходів, що базуються на машинному навчанні. Ці підходи дозволяють не лише автоматизувати рутинні операції, але й виявляти неочевидні закономірності, оптимізувати стратегії тестування та підвищувати загальну якість продукту. У цій статті ми досліджуємо потенціал машинного навчання як інструменту для трансформації процесу забезпечення якості ПЗ.

**Актуальність дослідження та постановка проблеми**

Згідно з сучасними тенденціями **DevOps** та **Continuous Delivery**, швидкість розгортання нових версій програмного забезпечення є ключовим показником. Це створює величезний тиск на процеси тестування, які часто стають "вузьким місцем". Традиційні підходи, такі як ручне або навіть скриптоване автоматизоване тестування, не можуть впоратися з динамічно змінними та високо навантаженими системами. Існують також проблеми з виявленням непередбачуваних дефектів, пов'язаних з комплексною взаємодією між мікросервісами, а також з неефективною витратою часу на дублюючі тести. Ця

проблема є актуальною для всіх галузей IT-індустрії, що вимагає розробки нових, більш інтелектуальних підходів. Метою даного дослідження є розробка та обґрунтування методики інтеграції алгоритмів машинного навчання у процес тестування для вирішення цих проблем.

Машинне навчання надає широкий спектр алгоритмів, які можуть бути адаптовані для вирішення завдань тестування. Наприклад, **алгоритми кластеризації**, такі як **K-Means** або **DBSCAN**, можуть бути використані для групування тестових сценаріїв за схожими характеристиками. Це дозволяє оптимізувати порядок їх виконання та виявляти надлишкові тести, які перевіряють одну й ту саму функціональність. **Прогностичні моделі**, наприклад, **Random Forest** або **Gradient Boosting**, можуть аналізувати історичні дані про дефекти та результати тестів, щоб передбачити ймовірність появи нових дефектів у певних модулях коду. Це дозволяє команді тестування зосередити свої зусилля на найбільш ризикованих областях, що значно економить час і ресурси.

Для реалізації інтелектуального тестування необхідно використовувати відповідні інструменти та фреймворки. Наприклад, у нашому курсі ми використовуємо **Docker** та **NestJS** для створення архітектури мікросервісів, що дозволяє легко інтегрувати різні інструменти тестування. Як тестовий об'єкт ми використовуємо `BookingService`, який був розроблений та модернізований у рамках лабораторних робіт. Це дозволяє нам наочно продемонструвати застосування алгоритмів на практиці.

**Основні положення та результати**

1. **Генерація тестових даних:** Використання **генеративно-змагальних мереж (GANs)** або інших моделей для створення реалістичних, але аномальних тестових сценаріїв. Наприклад, для `BookingService`, модель може генерувати запити з нетиповими датами, некоректними форматами даних, які можуть виявити логічні помилки у системі. Це дозволяє автоматично знаходити крайні випадки, які могли б бути пропущені людиною.

2. **Оптимізація тестових наборів:** Застосування **алгоритмів ранжування та відбору**, що базуються на аналізі покриття коду, історії дефектів та частоті змін. Наприклад, якщо модель виявить, що певний тестовий сценарій ніколи не виявляв дефектів протягом останніх 100 запусків, вона може запропонувати виключити його з щоденного набору регресійних тестів. Також можуть використовуватися **генетичні алгоритми** для пошуку найефективнішого набору тестів, що забезпечує максимальне покриття з мінімальною кількістю тест-кейсів.

3. **Прогнозування дефектів:** Розробка **класифікаційних моделей**, які на основі метрик коду (складність, зв'язаність) та даних про попередні помилки можуть передбачати, які компоненти системи найбільш схильні до збоїв. Це дає змогу сфокусувати увагу на певних модулях ще до початку тестування. Ми можемо застосувати **логарифмічну регресію** або **нейронні мережі**, навчені на історичних даних, щоб передбачити ймовірність помилки. Наприклад, якщо модель виявить високий рівень зв'язаності між модулями та частими змінами,

вона може позначити цей компонент як "високоризиковий".

Для успішного застосування машинного навчання в тестуванні необхідна відповідна архітектура, яка забезпечить збір даних, навчання моделей та їх інтеграцію в процес CI/CD. Ми пропонуємо трирівневу архітектуру:

1. **Рівень збору даних (Data Ingestion Layer):** Цей рівень відповідає за збір усіх релевантних даних, таких як результати unit-тестів, покриття коду, логи помилок, метрики продуктивності та дані з bug-трекінг систем (наприклад, Jira).
2. **Рівень аналітики та моделювання (Analytics & Modeling Layer):** На цьому рівні зібрані дані обробляються, очищаються та використовуються для навчання моделей машинного навчання. Застосовуються алгоритми, що були описані вище, для генерації, оптимізації та прогнозування.
3. **Рівень інтеграції (Integration Layer):** Моделі, що були навчені, експонуються через API і інтегруються в автоматизовані пайплайни CI/CD. Наприклад, перед запуском регресійного тесту, пайплайн може звернутися до прогностичної моделі, щоб отримати список найбільш ризикових модулів та сформувати оптимізований набір тестів. Це дозволяє не тільки автоматизувати, але й **інтелектуалізувати** процес тестування.

На прикладі нашої "Системи бронювання" ми можемо розробити модель, яка, аналізуючи вхідні дані для API `create-booking`, буде генерувати валідні та невалідні тестові запити, які виходять за межі очікуваних значень. Це дозволяє автоматично виявляти потенційні "діри" у валідації та логіці, які можуть бути пропущені при ручному або традиційному автоматизованому тестуванні. Застосування алгоритмів машинного навчання, зокрема, дозволяє підвищити точність тестування, зменшити час на розробку тестових сценаріїв і, як наслідок, покращити загальну якість програмного забезпечення.

**Висновки та практична цінність**

Представлена методика має значну практичну цінність. Вона дозволяє суттєво зменшити витрати на тестування, підвищити ефективність виявлення дефектів та прискорити цикл розробки. В рамках освітнього процесу, інтеграція таких підходів у навчальний план дозволяє готувати фахівців, які володіють не лише базовими навичками тестування, а й здатні застосовувати передові наукові методи.

Використання машинного навчання в тестуванні є перспективним напрямком, що дозволяє автоматизувати та інтелектуалізувати процеси забезпечення якості. Інтеграція таких підходів у навчальний процес дозволяє готувати фахівців нового покоління, здатних вирішувати складні інженерні завдання. Подальші дослідження можуть бути спрямовані на розробку більш складних моделей, які враховують не тільки статичний аналіз коду, але й динамічну поведінку системи під час навантаження. Перспективним напрямком є також розробка **"самовідновлюваних" тестів**, які б автоматично адаптувалися до змін у коді без втручання людини. Це дозволить ще більше зменшити людський фактор і забезпечити безперервне тестування в умовах швидкої розробки.

**Список літератури**

1. Калашник, В.Ю. (2024). Забезпечення якості програмного забезпечення в agile-середовищах. *Збірник наукових праць КНУТД*, 2024(1), 45-51.
2. Калашник, В.Ю. (2024). Інтеграція сучасних індустріальних практик проектування архітектури та забезпечення якості ПЗ у навчальні програми. *Scientific trends in the development of modern technologies and inventions*. Прага, Чехія.
3. Сарновський, М. А. (2021). Архітектура програмних систем. К.: КПІ ім. Ігоря Сікорського.
4. Кент, Б. (2020). Чистий код. Створення, аналіз і рефакторинг. К.: Наш Формат.
5. Григоренко, О. А. (2019). Основи тестування програмного забезпечення. К.: Центр учбової літератури.
6. Bass, L., Clements, P., & Kazman, R. (2012). Software Architecture in Practice (3rd ed.). Addison-Wesley.
7. Fowler, M. (2002). Patterns of Enterprise Application Architecture. Addison-Wesley.
8. Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley.
9. IEEE Std 1016-2009. (2009). IEEE Standard for Information Technology – Systems and Software Engineering – Recommended Practice for Software Design Descriptions. IEEE.

Scientific publications

MATERIALS

The VI International Scientific and Practical Conference
«Scientific trends in the development of modern technologies and inventions»

Prague, Czech Republic
(October 06-08, 2025)