Хмельницький національний університет
Факультет інформаційних технологій
Кафедра інженерії програмного забезпечення

КВАЛІФІКАЦІЙНА РОБОТА

Метод підтримки якості програмного коду на основі автоматичного тестування
Назва теми

Рівень вищої освіти Другий (магістерський)

Галузь знань 12 «Інформаційні технології»

Спеціальність 121 «Інженерія програмного забезпечення»

Освітня програма Освітньо-професійна програма «Інженерія програмного забезпечення»

Шифр КвРІПЗ.190156.01.05.ПЗ

Виконав студент 2 курсу, група ІПЗм-22-1 _____ Олег НОВАЦЬКИЙ
Підпис Ініціали. прізвище

Керівник к. техн. наук, доцент _____ Оксана ЯШИНА
Науковий ступінь. звання Підпис Ініціали. прізвище

Нормоконтролер к. техн. наук, доцент _____ Наталія ПРАВОРСЬКА
Підпис Ініціали. прізвище

До захисту допускаю:
Завідувач кафедри інженерії програмного забезпечення _____ Леонід БЕДРАТЮК
Підпис Ініціали. прізвище

8 грудня 2023 р.

Хмельницький 2023

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет Інформаційних технологій
Кафедра Інженерії програмного забезпечення
Рівень вищої освіти Другий (магістерський)
Галузь знань 12 «Інформаційні технології»
Спеціальність 121 «Інженерія програмного забезпечення»
Освітня програма Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Л. П. Бедратюк
01 09 2023 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Новацькому Олегу Валерійовичу
Прізвище, ім'я, по батькові студента

1. Тема проєкту (роботи) Метод підтримки якості програмного коду на основі автоматичного тестування

Керівник проєкту (роботи) канд. техн. наук, доцент Яшина О.М.
Прізвище, ім'я, по батькові, науковий ступінь, вчене звання

Затверджена наказом ректора університету від 15.08.2023 р. № 30
2. Строк подання студентом проекту (роботи) на кафедру 01.12.2023 р.
3. Вихідні дані до проекту (роботи) Матеріали науково-дослідної практики
4. Зміст пояснювальної записки (перелік питань, які потрібно розробити)
1 Аналіз предметної області та рішень з програмного забезпечення.
2 Удосконалення методу підтримки якості програмного коду на основі автоматичного тестування
3 Архітектура програмної реалізації
4 Програмна реалізація
5. Перелік графічного матеріалу (із зазначенням обов'язкових креслень)
Презентаційні матеріали (слайди)

6. Консультанти розділів дипломного проєкту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Антиплагіат	Форкун Ю.В., доцент	07.11.23	07.11.23
Нормоконтроль	Праворська Н.І., доцент		

7. Дата видачі завдання «01» вересня 2023 р.

КАЛЕНДАРНИЙ ПЛАН

Назва етапів (розділів) дипломного проекту (роботи)	Строк виконання етапів проекту (роботи)	Примітка
1. Ознайомлення з предметною областю; формулювання мети та задач дослідження; визначення об`єкта та предмета дослідження; визначення структури дипломної роботи	01.09 - 07.09.2023	
2. Аналіз предметної області, актуальних технологій, моделей та методів	08.09 - 25.09.2023	
3 Розробка моделей та методів вирішення завдання	26.09 - 10.10.2023	
4. Робота над науковими публікаціями	11.10 - 20.10.2023	
5. Проектування архітектури системи для вирішення задачі, розробка вимог.	11.10 - 26.10.2023	
6 Детальний опис реалізації та оцінки системи	27.10 - 15.11.2023	
7 Оформлення пояснювальної записки згідно вимог чинних стандартів	16.11 - 30.11.2023	
8 Попередній захист дипломної роботи	17.11.2023	
9 Перевірка роботи на наявність плагіату; норм контроль; брошурування пояснювальної записки; підготовка супровідних документів	01.12 - 04.12.2023	
10 Підготовка до захисту дипломної роботи	05.12 - 08.12.2023	

Студент _____ Олег НОВАЦЬКИЙ
Підпис Ініціали, прізвище

Керівник проєкту (роботи) _____ Оксана ЯШИНА
Підпис Ініціали, прізвище

РЕФЕРАТ

Тема дипломної роботи: «Метод підтримки якості програмного коду на основі автоматичного тестування».

Автор роботи: Новацький Олег.

Керівник роботи: Яшина Оксана Миколаївна.

Пояснювальна записка: 96 с., 18 рис., 2 дод., 36 джерела.

ЯКІСТЬ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ, ЯКІСТЬ ПРОГРАМНИХ СИСТЕМ, УПРАВЛІННЯ ЯКІСТЮ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ, ПІДТРИМКА ЯКОСТІ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ, ПРОГРАМНИЙ ПРОДУКТ, ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ, АВТОМАТИЧНЕ ТЕСТУВАННЯ, НЕЙРОННІ МЕРЕЖІ, ШТУЧНИЙ ІНТЕЛЕКТ.

Мета роботи – удосконалення методу підтримки якості програмного коду на основі автоматичного тестування.

Предмет – якість програмного коду та його підтримка засобами автоматичного тестування.

Об’єкт – процес створення системи підтримки якості програмного коду засобами автоматичного тестування.

На основі мети, предмету, об’єкту кваліфікаційної роботи магістра можна виділити наступні завдання дослідження:

1. Провести якісний аналіз предметної області.
2. Здійснити аналіз існуючих методів підтримки якості програмного коду.
3. Провести удосконалення методу підтримки якості програмного коду на основі автоматичного тестування.
4. Розробити алгоритм роботи системи підтримки якості програмного коду на основі автоматичного тестування.

Наукова новизна:

1. Удосконалено метод підтримки якості програмного коду на основі автоматичного тестування.

2. Розроблено алгоритм роботи системи підтримки якості програмного коду на основі автоматичного тестування.

Практичне значення отриманих результатів. У даній кваліфікаційній роботі магістра здійснено спробу здійснити удосконалення методу підтримки якості програмного коду, шляхом використання автоматичного тестування. Результатом цієї магістерської роботи є алгоритмічне рішення роботи системи підтримки якості програмного коду. Результати даного дослідження можуть якісно вплинути на роботу розробників програмного забезпечення, шляхом використання методу підтримки якості програмного коду на основі системи автоматичного тестування. Доцільність та ефективність розробки системи теоретично обґрунтована у першому та другому розділах кваліфікаційної роботи, що випливають із результатів емпіричних досліджень.

В якості теоретичних методів дослідження взято: аналіз, синтез, абстрагування, порівняння.

Емпіричними методами дослідження виступають: опис та тестування.

Підпис

01.12.2023
Дата

ABSTRACT

Master’s thesis: «Method for maintaining code quality based on automated testing».

Author: Oleg Novatskyi.

Head of work: Oksana Yashyna.

Master’s thesis consists of: 96 pages of the general text, 18 graphics, 2 supplements, 36 literature sources.

SOFTWARE QUALITY, SOFTWARE QUALITY SOFTWARE SYSTEM, QUALITY CONTROL, QUALITY SUPPORT, SOFTWARE PRODUCT, SOFTWARE TESTING, AUTOMATED TESTING, NEURAL NETWORKS, ARTIFICIAL INTELLIGENCE.

The purpose of the work is to improve the method of maintaining the quality of software code based on automatic testing.

The subject is software code quality and its support by means of automatic testing.

The object is the process of creating a software code quality support system by means of automatic testing.

Based on the goal, subject, object of the master's qualification work, the following research tasks can be distinguished:

1. Conduct a qualitative analysis of the subject area.

2. Analyze the existing methods of maintaining the quality of software code.

3. To improve the method of maintaining the quality of software code based on automatic testing.

4. Develop an algorithm for the system of software code quality support based on automatic testing.

Scientific innovation:

1. The method of maintaining the quality of software code based on automatic testing has been improved.

2. The algorithm of operation of the software code quality support system based on automatic testing has been developed.

Practical significance of the obtained results. In this master's qualification work, an attempt was made to improve the method of maintaining the quality of software code by using automatic testing. The result of this master's thesis is an algorithmic solution to the operation of the software code quality support system. The results of this study can qualitatively affect the work of software developers by using the method of maintaining the quality of software code based on the automatic testing system. The expediency and effectiveness of the system development is theoretically substantiated in the first and second sections of the qualification work based on the results of empirical research.

The following theoretical research methods are used: analysis, synthesis, abstraction, comparison.

Empirical methods of research are: description and testing.

Signature

01.12.2023
Date

ЗМІСТ

ВСТУП

Розробка високоякісного програмного забезпечення становить ключове завдання для прогресу у будь-якій галузі людської діяльності, що особливо стає актуальним у сучасних умовах військової агресії на території України. Успішність програмної системи має прямий вплив на ефективність управління економічними об'єктами та процесом прийняття обґрунтованих рішень. Якість програмного забезпечення стає важливим критерієм в різних галузях застосування інформаційних технологій, засвідчуючи про його відповідність встановленим стандартам якості, в тому числі й на міжнародному рівні.

Загалом, під якістю програмного коду маються на увазі його характерні ознаки та властивості і вони можуть досить суттєво відрізнятися залежно від конкретних бізнес-потреб організації та завдань команди. І хоча немає точного переліку певних означених показників, однак існує декілька загальних критеріїв, що відрізняють якісний програмний код від неякісного.

Для підвищення, а також підтримки якості програмного коду використовують тестування. Загалом же тестування є важливим процесом у багатьох галузях техніки: новий телевізор чи смартфон, системи програмного забезпечення автомобіля, тестування може допомогти інженерам виявити дефекти та переконатися в тому, що продукт міцний у різних умовах використання.

Тестування програмного забезпечення має ту ж мету, що й тестування, яке застосовується в інших галузях техніки, а саме виявлення дефектів або недоліків товару. Тестування можна умовно розділити на дві категорії: ручне тестування та автоматичне тестування. У ручному тестуванні тестові випадки виконуються вручну людиною без будь-якої підтримки інструментів чи сценаріїв, тобто тестувальник вручну перевіряє програмний продукт на виявлення дефектів і складає звіт після цього. В автоматичному тестуванні тестові випадки виконуються за допомогою інструментів, сценаріїв і програмного забезпечення. Отже під автоматичним тестуванням розуміють використання різного роду програмних застосунків для пришвидшення то збільшення точності ручного

процесу перевірки та валідації програмного продукту, який виконується людиною.

Мета роботи – удосконалення методу підтримки якості програмного коду на основі автоматичного тестування.

Предмет – якість програмного коду та його підтримка засобами автоматичного тестування.

Об’єкт – процес створення системи підтримки якості програмного коду засобами автоматичного тестування.

На основі мети, предмету, об’єкту кваліфікаційної роботи магістра можна виділити наступні завдання дослідження:

5. Провести якісний аналіз предметної області.
6. Здійснити аналіз існуючих методів підтримки якості програмного коду.
7. Провести удосконалення методу підтримки якості програмного коду на основі автоматичного тестування.
8. Розробити алгоритм роботи системи підтримки якості програмного коду на основі автоматичного тестування.

Наукова новизна:

3. Удосконалено метод підтримки якості програмного коду на основі автоматичного тестування.
4. Розроблено алгоритм роботи системи підтримки якості програмного коду на основі автоматичного тестування.

Практичне значення отриманих результатів. У даній кваліфікаційній роботі магістра здійснено спробу здійснити удосконалення методу підтримки якості програмного коду, шляхом використання автоматичного тестування. Результатом цієї магістерської роботи є алгоритмічне рішення роботи системи підтримки якості програмного коду. Результати даного дослідження можуть якісно вплинути на роботу розробників програмного забезпечення, шляхом використання методу підтримки якості програмного коду на основі системи автоматичного тестування. Необхідність та актуальність такої системи аргументовано у першому та другому

розділах кваліфікаційної роботи із врахуванням результатів наявних емпіричних досліджень.

У даній кваліфікаційній роботі використовувались такі теоретичні методи, як аналіз, синтез, абстрагування, здійснення порівняння.

Використовуваними емпіричними методами дослідження виступають: опис та тестування.

Відповідно до теми кваліфікаційної роботи опубліковані тези «Метод підтримки якості програмного коду на основі автоматичного тестування» на конференції «Актуальні проблеми комп’ютерних наук АПКН-2023».

1 ТЕОРЕТИЧНИЙ ВИКЛАД ДОСЛІДЖУВАНОЇ ПРОБЛЕМИ

1.1 Аналіз предметної області

Якість є одним із важливих аспектів будь-якої програмної системи [10] і загалом програмного забезпечення незалежно від галузі чи технічної сфери. Більше того, забезпечення якості програмного забезпечення - це не одноразове завдання; починаючи з моменту впровадження і триваючи протягом усього терміну служби системи, розробники виконують технічне обслуговування для досягнення функціональних і нефункціональних цілей свого програмного забезпечення. Цей акцент на якості програмного забезпечення призвів до того, що організації витрачають від 60% до 80% ресурсів на технічне обслуговування програмного забезпечення [16], що є однією із найбільш витратних фаз життєвого циклу розробки програмного забезпечення.

Загалом же розробники програмного забезпечення перебувають в конкурентній гонитві за програмне забезпечення, де вони працюють в умовах дефіциту часу, а операційні директори - в умовах обмежених бюджетів. Але розробники, які є мозковими центрами технічної команди, відповідальної за впровадження нових функцій у програмне забезпечення, витрачають в середньому до 5 годин на тиждень лише на перегляд коду. Ці факти свідчать про те, що операційні директори ігнорують основний будівельний блок програмного забезпечення - якість коду. Якісний код так само важливий, як і функціональне програмне забезпечення.

Попри те, що якість коду є одним з найважливіших критеріїв при розробці програмного забезпечення, його часто ігнорують. Це призводить до розробки програмного забезпечення низької якості. Таке програмне забезпечення може функціонувати подібно до програмного забезпечення з належними стандартами кодування, але з часом воно застаріє і призведе до значного технічного боргу.

Основним аргументом, чому розробники ігнорують якість коду, є робоче навантаження. Понад 40% розробників вважають, що робоче навантаження та необхідність завершити проект у визначені терміни змушують їх ігнорувати

базові стандарти якості під час розробки програмного забезпечення. Ще однією причиною ігнорування будь-якої діяльності з перегляду коду є те, що перегляд коду забирає багато часу і є нудним процесом.

Інструменти спрощують процес покращення якості коду, але бракує таких інструментів, які були б комплексними, ефективними та бюджетно прийнятними. Керівники компаній несуть однакову відповідальність за ігнорування практики перегляду коду, оскільки більшість керівників не можуть забезпечити дотримання стандартів якості серед членів команди. Брак людських ресурсів та місцезнаходження членів команди є іншими основними перешкодами в процесі перегляду коду, які не вдається подолати операційним директорам. В результаті розробляється неякісно закодована програма, яка має невеликий термін служби і стикається з багатьма проблемами, щойно виникне потреба у внесенні змін.

Одним із важливих чинників підтримки якості програмного забезпечення є розуміння коду, оскільки для того аби зрозуміти програму необхідно прочитати вихідний код, визначити його призначення, вимоги, що пов'язані з їхньою діяльністю з обслуговування. З метою розвитку програмної системи, перед тим як змінювати щось у програмному коді, розробник повинен його прочитати у вихідному файлі коду для розуміння його поведінки.

Очевидним є той факт, що такі нюанси, як незрозумілість під час читання коду, низький рівень читабельності, впливають не тільки на час, що витрачають розробники виконуючи задачі, але й можуть опосередковано або безпосередньо впливати на якість програмної системи в цілому.

Тестування є найстарішим і найсуворішим типом підтримки якості програмного забезпечення. Так само, як в інших технологічних галузях, де використовується термін "тестування очима", для проведення тестових операцій вручну необхідний спеціаліст - тестувальник програмного забезпечення. Це вкрай важливо, оскільки незалежно від того, наскільки автоматизованим стає програмне забезпечення, автоматизація не може замінити людської інтуїції, умовиводу та індуктивного міркування. Наприклад, людина може змінити напрямок тестування посеред тестового запуску, щоб перевірити аспекти, які не були враховані раніше.

Крім того, автоматизоване тестування не може ефективно оцінити певні аспекти, такі як зручність для користувача чи взаємодія з клієнтом.

Однак, існують і недоліки, оскільки деякі види тестування, такі як тестування навантаження чи продуктивності, неможливо провести вручну, і, що найважливіше, ручне тестування вимагає значного часу. Саме через це багато компаній віддають перевагу автоматизованим тестам. У цьому підході тестувальники розробляють тестові сценарії для автоматизованої перевірки продукту. Іншими словами, вони не проводять "ручне" тестування програмного забезпечення, а використовують заздалегідь розроблені тести, які автоматично виконуються. Враховуючи це, автоматизоване тестування є швидшим та більш надійним способом виконання тестів і оцінки їх успішності, порівняно з ручним тестуванням. Загалом, автоматизоване тестування стало важливим процесом для покращення якості програмного забезпечення системи. Автоматизовані тести можуть допомогти переконатися, що виробничий код є надійним у багатьох умовах використання та що код відповідає вимогам продуктивності та безпеки.

Тестування є важливим процесом у багатьох галузях техніки: будь то новий телевізор, програмне забезпечення системи автомобіля, тощо тестування може допомогти інженерам виявити дефекти та переконатися в тому, що продукт міцний у різних умовах використання.

Тестування програмного забезпечення має ту ж мету, що й тестування, яке застосовується в інших галузях техніки, а саме виявлення дефектів або недоліків товару. Тестування можна умовно розділити на дві категорії: ручне тестування та автоматизоване тестування. У ручному тестуванні виконуються тестові випадки вручну людиною без будь-якої підтримки інструментів чи сценаріїв, тобто тестувальник вручну перевіряє виріб на виявлення дефектів і складає звіт після цього; в автоматизованому тестуванні, тестові випадки виконуються за допомогою інструментів, сценаріїв і програмного забезпечення.

Автоматизоване тестування вважається невід'ємною частиною будь-якого серйозного процесу розробки програмного забезпечення. Автоматизація дозволяє легко і швидко повторювати окремі тести або набори тестів під час розробки. Це

допомагає гарантувати, що випуски відповідають цілям якості та продуктивності. Автоматизація допомагає збільшити покриття і забезпечує швидший цикл зворотного зв'язку з розробниками. Автоматизація не тільки підвищує продуктивність окремих розробників, але й забезпечує виконання тестів на критичних етапах життєвого циклу розробки, таких як перевірка контролю вихідного коду, інтеграція функцій та випуск версії.

1.2 Аналіз існуючих рішень

Покращення та вдосконалення засобів та методів тестування складних програмних продуктів заслуговує на велику увагу фірм-розробників програмного забезпечення (ПЗ), оскільки незаперечним є факт важливості приведення до відповідного та затребуваного рівня якості програмного забезпечення, що випускається.

Тестування як основний спосіб підтримки, контролю та управління якістю різноманітних програмних рішень визначається як процес виконання програми з метою перевірки відповідності між реальною та очікуваною її поведінкою, а також у відповідності до вимог [33, 34]. Тестування здійснюють на кінцевому наборі тестових випадків, що обираються певним чином. В загальному процес тестування може бути представлений сукупністю таких основних етапів: планування робіт, розробка тестів та їх виконання, аналіз результатів перевірки ПЗ.

По всьому життєвому циклу розробки та супроводження програмного забезпечення тестування здійснюється на різних його рівнях. Рівень визначає, який об'єкт або елемент продукту, що тестується, розглядається в ході перевірки. Типові рівні тестування подано на рисунку 1.1.

Рисунок 1.1 – Головні рівні тестування програмного забезпечення

У модульному тестуванні помилки виявляються окремо з кожного компонента або блоку шляхом індивідуального тестування компонентів або блоків програмного забезпечення, щоб переконатися, що вони придатні для використання розробниками. Це найменша частина програмного забезпечення, що підлягає тестуванню.

У цьому тестуванні два або більше модулів, які тестуються помодульно, інтегруються для тестування, тобто технічно взаємодіючих компонентів, а потім перевіряється, чи працюють ці інтегровані модулі відповідно до очікувань чи ні, а також виявляються помилки інтерфейсу. Найбільш поширені такі покрокові підходи інтеграційного тестування як «згори донизу» (представлений на рисунок 1.2) і «Знизу вгору» (рисунок 1.3).

Якщо для валідації роботоздатності програмного забезпечення використовується метод низхідного тестування, спочатку акцентується увага на високорівневих модулях програми, а на подальших етапах увага переходить до

модулів, розташованих на більш низьких рівнях архітектурної структури програми.

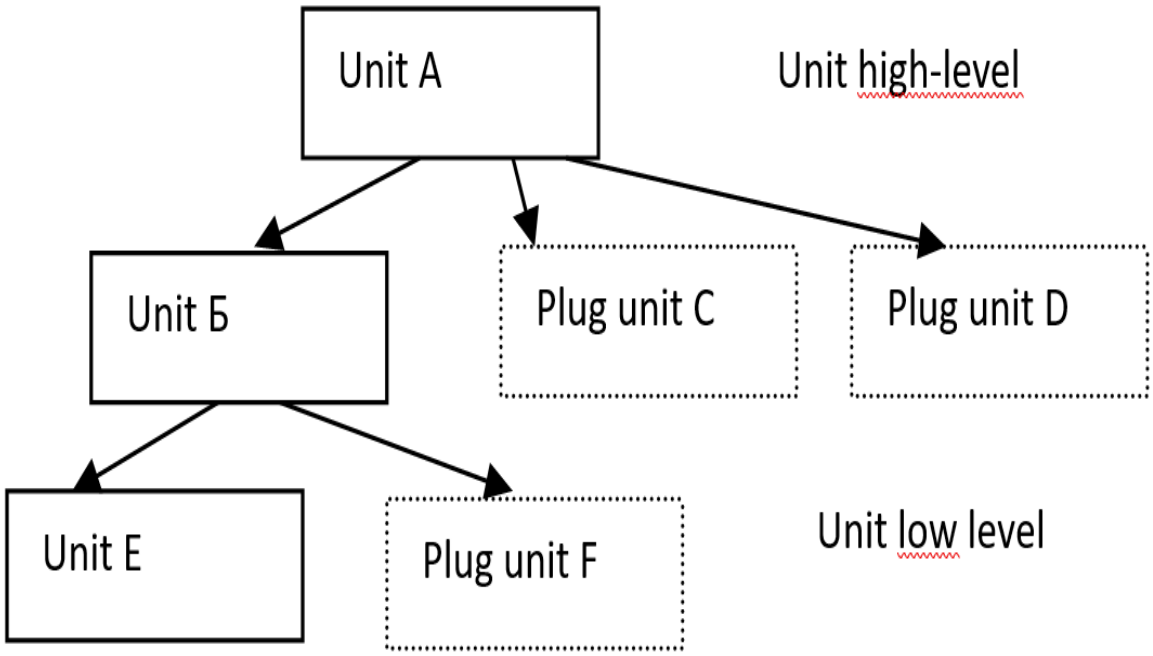


Рисунок 1.2 – Інтеграційне тестування згори-донизу

Для коректного тестування модуля, що перевіряється, підключаються допоміжні класи-заглушки, що виконують функцію заміщення реальних класів рівнем нижче за той, що розглядається. У міру готовності класи-заглушки замінюються реальними активними компонентами. Такий підхід дає змогу отримати загальне уявлення про структуру програми на проміжних етапах тестування, оцінити організацію взаємодії програмного продукту з користувачем.

Під час висхідного тестування першочерговою задачею є розгляд термінальних класів, тобто класів, які не використовують методи інших класів у своєму функціонуванні. Для подальшого етапу тестування вибираються модулі, які під час роботи використовують методи лише тих класів, які вже були протестовані. Ключову роль в цьому процесі відіграє драйвер для кожного модуля. Драйвер включає фіксовані тестові дані, ініціалізує тестовий модуль,

отримує вихідні результати тестування і порівнює їх із передбачуваним результатом роботи модуля.

Недолік використання висхідного підходу зводиться до відсутності структури програми на проміжних етапах тестування, при цьому цей підхід забезпечує більш просту перевірку результатів за рахунок відсутності необхідності використовувати заглушки для неготових модулів. Висхідний підхід найкращим чином підходить для ситуацій, коли помилки містяться в модулі нижнього рівня [20, 22, 33].

На етапі системного тестування весь інформаційний продукт, що розробляється, перевіряється на наявність помилок у функціоналі, коли програмні та апаратні компоненти системи об'єднані в єдине ціле. На цьому рівні перевіряється правильність використання системних ресурсів, сумісність з навколишнім середовищем, повнота і коректність функціональності та зручність використання. Для проведення системного тестування створюються тест-кейси на основі вимог до продукту, що розробляється (технічного завдання) та сценарію використання системи.

Приймальне тестування проводиться з метою визначення готовності програмного продукту до передачі замовнику. Приймальне тестування може проводитися фахівцями компанії-розробника програмного забезпечення або замовником, в останньому випадку це зовнішнє тестування.

На сьогоднішній день існує досить велика кількість систем тестування, з допомогою яких можна перевірити поведінку найменших та незначних елементів, будь-якої програми. Для тестування найрізноманітніших проектів, найчастіше застосовується той самий інструментарій з можливістю виконання типових завдань. Проте, варто враховувати, що тестування відбувається, ґрунтуючись на завданнях тестування та конкретної мети, у зв'язку з цим можна помітити, що не всі інструменти для тестування однаково застосовуються до різних методик тестування

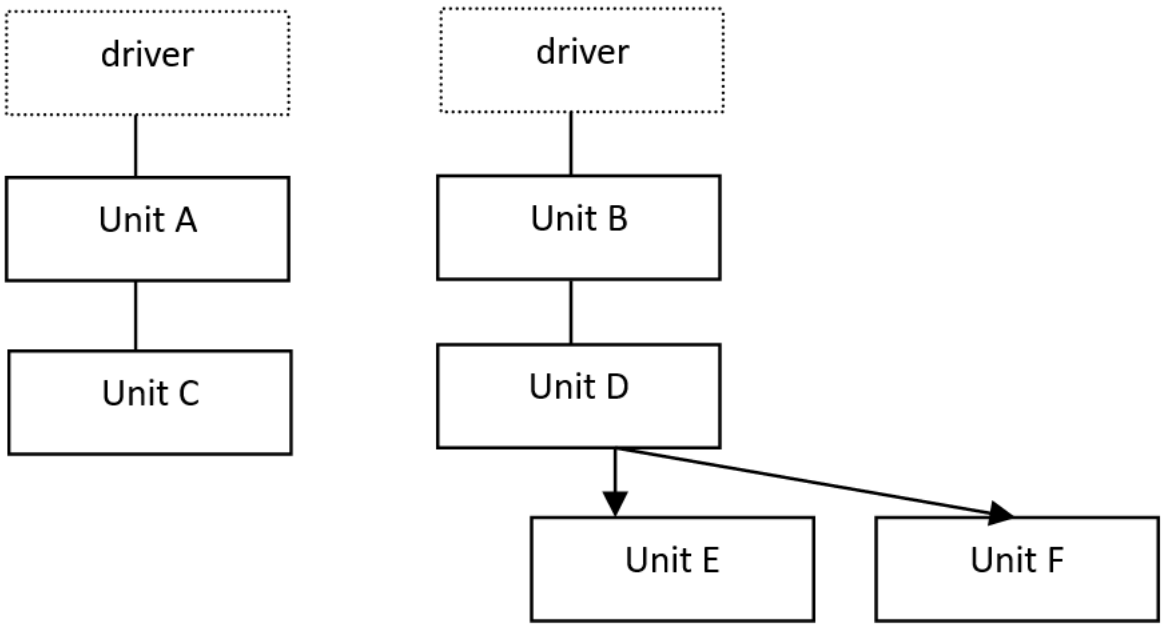


Рисунок 1.3 – Висхідне тестування

Наприклад, у тестуванні навантаження інструменти працюють на рівні протоколу, а для автоматизації регресійного тестування, інструменти працюють на рівні графічного використання інтерфейсів. Для виявлення та аналізу проблем у різних частинах системи є досить багато інструментів.

При проведенні тестування необхідно конкретно визначити цілі та завдання тестування: для яких цілей тестується додаток, які завдання будуть розглядатися при тестуванні та що надалі робити з отриманою інформацією. Тестування необхідно для виявлення невідповідностей у роботі програми, воно є одним із життєвих циклів при проектуванні та розробці будь-якого програмного продукту. Оскільки, допоможе знайти помилки на стадії проектування. Для цього були розглянуто деякі існуючі системи тестування з метою визначимо їх призначення та можливості застосування для тестування різноманітного програмного забезпечення, враховуючи їх застосування в різних видах та методах тестування

Автоматизоване тестування використовує програмні засоби для виконання тестів і перевірки результатів, що допомагає скоротити час тестування і спростити його процес.

При цьому існує два основних підходи: тестування на рівні коду, зокрема, модульне тестування, і тестування інтерфейсу користувача, коли здійснюється

імітація дій користувача за допомогою спеціальних тестових фреймворків. Сучасні інструменти тестування інтерфейсу користувача в загальному підтримуються не тільки виконанням сценарію тесту, але і його записом, коли дії користувача записуються і потім можуть відтворюватися в автоматичному режимі. Сценарії зберігаються у форматі, що підтримується конкретним фреймворком чи мовою програмування, і можуть змінюватися користувачем перед виконанням [12, 14].

Ключові переваги автоматизованого тестування:

– виключення «людського фактора» – автоматичне виконання сценарію не пропускає помилок;

– тест «через необережність» не помиляється в результатах, що підвищує якість результату;

– швидке виконання – автоматичне тестування не потрібує перевірки з інструкціями та документацією;

– невеликі витрати на підтримку – коли сценарії вже написані, на їх підтримку та аналіз результатів потрібно, як правило, менше часу, ніж на проведення того ж обсягу тестування вручну;

– автоматичне формування звітів про результати тестування;

– безперервне виконання тестів, в тому числі і в «неробочий час».

Ключові недоліки автоматизованого тестування:

– всі написані сценарії завжди будуть виконуватися однотипно, що не дозволить знайти помилку, яку би бачила людина, наприклад, на екранній формі відобразилися не ті дані, але сценарій передбачає перевірку інших елементів;

– чим частіше змінюється застосунок, тим вище витрати на підтримку сценаріїв;

– розробка автоматизованих тестів – це складний процес, так як фактично іде розробка додатків, які тестує інший застосунок, що підвищує вимоги до знань та вмінь тестувальника.

Таким чином, вибір підходу до тестування суттєво залежить від особливостей тестованого застосунку та процесу його розробки, а також кваліфікації тестувальника та вимог до якості.

Основні особливості програмних комплексів обробки вимірювальної інформації [19, 20, 22, 25]:

– обробка великого масиву вихідних даних;

– трудомісткі підходи з обробки даних;

– відмінності між програмними комплексами заключаються в основному в способі обробки даних, а не в способах представлення.

Для розробки програмних комплексів обробки вимірювальної інформації використовується модель інкрементної розробки, коли різні частини програмного коду розробляються в різний час і в загальному випадку не повинні виникнути зміни в уже завершених частинах.

Розроблена структура класів, що дозволяє створювати узагальнені сценарії. Загалом базовий клас, який забезпечує тестування. У рамках TestBase реалізується базова функціональність, що дозволяє виконувати узагальнені сценарії на користь всіх програмних комплексів обробки вимірювальної інформації, TestData надає унікальні дані [12, 17].

Використання узагальнених сценаріїв змінює класичну технологію автоматизованого тестування. При створенні узагальненого сценарію для тестування інтерфейсу користувача необхідно виконати такі кроки:

1) запис сценарію з використанням штатних інструментів тестування на одному із програмних комплексів;

2) визначення унікальних програмних комплексів послідовностей дій;

3) у згенерованих Unit-тестах для унікальних послідовностей дій винесення з використанням рефакторингу фрагментів коду у відповідному класі;

4) у разі відмінностей у поведінці серверної частини опис відповідної логіки на базі сервер-тесту;

5) виділення даних, необхідних для виконання сценарію, та поміщення їх у класи;

6) формування тестових сценаріїв для інших програмних комплексів з урахуванням їх особливостей.

Виконання четвертого кроку вимагає від тестувальника не тільки гарного знання інструментів автоматизованого тестування, програмування та вміння писати Unit-тести, але й знання логіки побудови серверної частини, що загалом відомо лише розробнику. Доцільним є цей крок виконувати авторам серверної частини програм.

Вказану послідовність кроків необхідно повторити стільки разів, скільки узагальнених сценаріїв необхідно створити для повноцінного тестування всіх програмних комплексів обробки вимірювальної інформації.

У процесі виконання тестів за узагальненими сценаріями на підставі зазначеної структури класів автоматично формуватимуться унікальні сценарії тестування для конкретного програмного комплексу, які враховують як особливості побудови клієнтської та серверної частини, так і вхідних та вихідних даних.

Переваги узагальнених сценаріїв:

– за наявності узагальненого сценарію, створеного для одного із програмних комплексів, формування сценаріїв інших програмних комплексів є нетрудомісткою завданням;

– у разі змін та виправлення помилок у загальної частини програмних комплексів змінюється загальна частина сценаріїв тестування, що скорочує час внесення змін;

– при появі нових аналогічних програмних комплексів складність створення сценаріїв істотно менша, ніж при написанні їх з самого початку;

– при появі нової функціональності потрібні незначні зміни узагальненого сценарію.

Недоліки узагальнених сценаріїв:

- Суттєвий час розробки першого узагальненого сценарію, який може окупитися лише за умови його повторного використання;

- Висока складність виділення загальних частин, яка потребує відповідної кваліфікації тестувальника;

- Об'єм узагальненого сценарію в загальному випадку більше звичайного сценарію.

В результаті переваги узагальнених сценаріїв полягають у можливості їх повторного використання для ідентичних програмних систем, що значно знижує трудомісткість розробки нових сценаріїв. Даний підхід з урахуванням зазначених особливостей застосовується для програмних комплексів обробки вимірювальної інформації, але може бути недостатньо ефективним при виконанні автоматизованого тестування в інших областях.

1.3 Огляд методів вирішення проблеми

Емпірична інженерія програмного забезпечення - це галузь розробки програмного забезпечення, зосереджена на збиранні даних за допомогою вимірювань і експериментів для побудови теорій про задіяні процеси в інженерії програмного забезпечення. Дана робота вбудована в контекст емпіричного дослідження інженерії програмного забезпечення. Доцільне використання змішаного підходу, щоб відповісти на дослідницькі запитання, використовуючи опитування, контрольовані експерименти, інтерв’ю та сховища програмного забезпечення для аналізу.

Нижче подано характеристики кожного методу дослідження, який доцільно використовувати під час вирішення поставлених завдань.

Репозиторії програмного забезпечення для майнінгу MSR - це галузь досліджень інженерії програмного забезпечення, у якій дослідники витягують дані зі сховищ програмного забезпечення. У даній роботі в основному розглядається два сховища програмного забезпечення: система керування версіями та сховище рецензій. Зокрема, доцільно використовувати репозиторії

контролю версій, щоб отримати інформацію про різні аспекти коду та те, як він розвивається з часом.

Для полегшення процесу дослідження доцільно використовувати фреймворк, із подальшим використанням його для вивчення проблем розробки тестів і того, як розробники це використовують. Доцільним також є використання репозиторіїв, щоб отримати інформацію про практики перевірки коду проекту, такі як обговорення серед рецензентів, і як вони виконують перевірку тестового коду.

Інтерв'ю та опитування. Хоча через MSR дослідники можуть зрозуміти, що відбувається в програмній системі, зрозуміти, чому це відбувається, часто неможливо лише спостерігаючи за даними. Щоб отримати повніші дані для відповідей на подібні питання можна використовувати інтерв’ю: є можливість дізнатися, чому щось сталося, спілкуючись із розробниками, можна запитати їхню думку щодо певної поведінки та зрозуміти процес прийняття рішень, тобто процес, що стоїть за зміною. Крім того, не зважаючи на те, що співбесіди є важливими джерелом інформації, результати можуть бути не узагальненими: отже, також є доцільність використовувати опитування, щоб оскаржити та розширити отримані результати.

Контрольований експеримент - це наукове випробування, проведене в контрольованих умовах, що означає, що одночасно змінюється лише один (або кілька) факторів, тоді як усі інші залишаються постійними. У цьому типі експерименту учасники поділяються на дві (або більше) груп: одна контрольна група (служить базовою лінією) і одна або більше груп, у яких усі змінні ідентичні контрольній групі, за винятком однієї, яка є фактором, що перевіряється. Також доцільним є вивчення ефекту TDR: в у цьому випадку змінна, що перевіряється, є прийняттям самого TDR. За допомогою даної методики можна вивчити вплив наявних коментарів рецензування на результат рецензування: у цьому випадку, змінна - наявність коментаря при запуску перегляду коду.

Поточні правила виявлення тестових проблем недостатньо точні, і, що більш важливо, не підтримують встановлення пріоритетів. Важливим моментом також є допомога розробникам краще та якісніше здійснювати перегляд тестового коду. По-перше, досліджуються потреби розробників, коли йдеться про перевірку коду, визначаючи інформаційні потреби високого рівня, які можна задовольнити за допомогою автоматизованих інструментів, заощаджуючи час для рецензентів. Потім йде зосередження на перевірці тестового коду зокрема. Спочатку вивчається, коли і як розробники переглядають тестовий код, визначаючи поточні практики, розкриваючи проблеми, з якими стикаються під час перегляду тестового коду, і розкриваючи потреби для інструментів, які можуть підтримувати перегляд тестового коду. Далі здійснюється дослідження впливу перевірки коду через тестування (TDR) на ефективність перевірки коду, показуючи, що вона може підвищити кількість знайдених проблем із тестовим кодом. Обговорюється, коли можна застосовувати TDR, а коли ні і чому не всі розробники бачать TDR як гідну практику.

1.4 Постановка задачі

Під час здійснення даного дослідження було поставлено таку мету - удосконалення методу підтримки якості програмного коду на основі автоматичного тестування.

Згідно із метою визначено такі завдання:

1. Провести аналіз предметної області.
2. Здійснити аналіз існуючих методів.
3. Провести удосконалення методу підтримки якості програмного коду на основі автоматичного тестування.
4. Розробити алгоритм роботи системи підтримки якості програмного коду на основі автоматичного тестування.

Отже, в даній кваліфікаційній роботі має бути здійснено удосконалення методу підтримки якості програмного коду за допомогою автоматичного тестування згідно алгоритму роботи системи.

Результати даного дослідження мають на меті допомагати розробникам використовувати ефективний та ретельно спроектований вибір інструментів для автоматизації та впровадження процесу тестування, що є ключовим етапом. Використання визнаних принципів і шаблонів проєктування, таких як тестування на основі даних або тестування на основі ключових слів, сприяє успішній реалізації цих завдань.

Створення надійних тестових сценаріїв, які взаємодіють з різноманітними вхідними даними, відкриває можливість їхнього подальшого перетворення в тести, які керуються даними. Це забезпечує гнучкість та адаптивність в процесі тестування, оскільки одні й ті ж сценарії можуть використовуватися з різними варіантами введених даних.

Наочна потужність та добре написані тестові сценарії роблять їх легкими у підтримці. Це важливо для ефективного відстеження та управління тестовими сценаріями на всіх етапах розробки програмного забезпечення.

1.5 Висновки до 1-го розділу

У першому розділі кваліфікаційної роботи проаналізовано предметну область, а також реальний та актуальний стан проблеми, що досліджується досліджуваної проблеми.

У підрозділі 1.1 було встановлено, що автоматичне тестування - це метод тестування програмного забезпечення, який передбачає використання інструментів та фреймворків автоматизації для виконання одного і того ж набору тест-кейсів велику кількість раз. Ключова різниця між ручним та автоматизованим тестуванням полягає в тому, що ручне тестування повністю

залежить від людини, яка сидить за комп'ютером. У той час, як автоматизовані тести можуть бути написані один раз і виконуватися багаторазово практично без участі людини.

Основні сфери застосування автоматизованого тестування: при виконанні тестів, що повторюються, при використанні тестування продуктивності або при тестуванні навантаження, коли є велика кількість тест-кейсів, коли потрібно виключити людський чинник, наявність великого обсягу даних.

У підрозділі 1.2. здійснено аналіз існуючих рішень та встановлено, існує безліч систем тестування, з допомогою яких можна перевірити поведінку найменших та незначних елементів, будь-якої програми. Для тестування найрізноманітніших проектів, найчастіше застосовується той самий інструментарій з можливістю виконання типових завдань.

У підрозділі 1.3 було здійснено аналіз методів вирішення проблеми, до яких належать емпірична інженерія програмного забезпечення, інтерв'ю та опитування, контрольований експеримент.

У підрозділі 1.4 було здійснено постановку задачі та окреслено шляхи її розв’язання.

Результатом написання першого розділу стало визначення мети даного дослідження, яка полягає у вдосконаленні методу підтримки якості програмного коду на основі автоматичного тестування та розробки відповідного алгоритму якості. Також було визначено об’єкт та предмет дослідження і виділено ті задачі, які необхідно вирішити для досягнення даної мети.

2 МЕТОД ПІДТРИМКИ ЯКОСТІ ПРОГРАМНОГО КОДУ

2.1 Концептуальна модель процесу підтримки якості програмного коду

Концептуальна модель – означає будь-яку модель, сформовану після процесу концептуалізації або узагальнення.

Модель - це графічний опис поведінки системи. Вона допомагає нам зрозуміти і передбачити поведінку системи. Моделі допомагають генерувати ефективні тестові кейси, використовуючи системні вимоги. Для тестування на основі моделі необхідно врахувати наступне:

- Побудувати модель;
- Визначити вхідні дані для моделі;
- Обчислити очікуваний вихід для моделі;
- Запустити тести;
- Порівняти фактичний результат з очікуваним;
- Прийняття рішення про подальші дії з моделлю.

Деякі з методів моделювання, з яких можуть бути отримані тестові кейси:

Діаграми - відображають стан системи і перевіряють стан після певних вхідних даних.

Таблиці рішень - таблиці, що використовуються для визначення результатів для кожного входу

Тестування на основі моделей - це техніка генерації тестових кейсів на основі вимог, що розвивається. Його основна перевага, полягає в тому, що він може визначати небажані стани, яких може досягти ваш графічний інтерфейс.

Вихідні дані, які були зібрані під час моделювання, та необхідність рівня деталізації та вхідних даних для моделювання визначаються відповідно до поставлених цілей моделювання [33, 34, 35]. Рівень деталізації моделі визначається за наступними факторами: метою проекту, критеріями оцінки ефективності, доступністю даних, достовірністю результатів, можливостями обчислювальної техніки, експертними думками щодо вирішуваної проблеми, а також обмеженнями у часі і фінансах. Здійснюють структурний аналіз

концептуальних моделей, розробляють опис припущень і обговорюють проблеми з клієнтами, керівниками проектів, аналітиками та експертами [33, 34, 35].

Створюють моделі вхідних даних і проводять їхній статистичний аналіз, результатом якого є визначення розподілу ймовірностей, регресійних, кореляційних та інших залежностей. На цьому етапі широко використовуються різні пакети для попереднього аналізу даних [33, 34, 35].

Для динамічних систем проводиться попередній аналіз функціонування, моделюються системи з детальним описом роботи елементів системи. За результатами такого аналізу можна визначити, чи можна вирішити проблему без використання моделювання. Детально розроблена концептуальна модель надає можливість замовнику визирнути на роботу системи та, наприклад, визначити вузькі місця системи, що призводять до зниження її пропускної здатності [33, 34, 35].

Однією з найбільш складних завдань, які стоять перед аналітиком з моделювання, є визначення адекватності моделі для системи. Якщо імітаційна модель є адекватною, її можна використовувати для прийняття рішень для системи, яку вона представляє, начебто вона базується на результатах експериментів з реальною системою. Модель складної системи може наближено відображати оригінал, незалежно від затрат, оскільки абсолютно адекватних моделей не існує.

Оскільки модель завжди розробляється для конкретного набору цілей, та модель, яка є адекватною для одного завдання, може не бути такою для іншого. Важливо відзначити, що адекватність моделі і достовірність не завжди збігаються. Модель може бути достовірною, але при цьому непридатною для прийняття рішень. Наприклад, достовірна модель може не враховуватися з політичних чи економічних міркувань.

На кожному етапі виконується перевірка достовірності моделі. Цей процес можна розділити на два етапи: валідація, що включає перевірку правильності створення концептуальної моделі, та верифікація, яка перевіряє правильність реалізації моделі. Зазвичай, у випадку моделювання, вимога ізоморфізму моделі

та об'єкта є зайвою, оскільки складність моделі повинна відповідати складності об'єкта.

Таким чином, створюють гомоморфні моделі, де виконується умова однозначної відповідності між моделлю та об'єктом. На етапі верифікації перевіряється, чи правильно концептуальна модель (модельні припущення) перетворюється на комп'ютерну програму, тобто налаштовується програма моделювання. Це завдання складне, оскільки може існувати безліч логічних шляхів. Етап перевірки правильності реалізації моделі включає перевірку еквівалентності перетворення моделі на кожному етапі її реалізації та порівняння станів. У цьому випадку модель проходить такі зміни: концептуальна модель - математична модель - алгоритм моделювання - програмна реалізація моделі.

Загалом валідація представляє собою процес, спрямований на визначення того, чи є модель (або комп'ютерна програма) точним відображенням системи для конкретних дослідницьких цілей. Розробляється план проведення експериментів з моделлю з метою досягнення визначеної мети. Основна мета цього планування експериментів полягає в вивченні поведінки модельованої системи за мінімальних витрат під час проведення досліджень.

Експерименти, що проводяться найчастіше подано нижче:

– порівняння середніх значень, а також дисперсій різних альтернатив;

– визначення важливості обліку впливу змінних та обмежень, які накладаються на встановлені змінні;

– визначення оптимальних значень з визначеної множини певних встановлених значень змінних [33, 34, 35].

Проведення експериментів зазвичай планується до пошуку незначних чинників. При оптимізації числового критерію утворюють гіпотезу щодо вибору оптимальних структур для модельованої системи або режимів її функціонування. Визначають діапазон значень параметрів (або режимів функціонування) моделі, в якому може бути знайдене оптимальне рішення. Також визначається кількість та час реалізації кожної моделі. Проводять екстремальний експеримент, під час якого визначається оптимальне значення критерію та відповідні значення

параметрів. Для оцінки точності стохастичних моделей будують довірчі інтервали, щоб отримати вихідні змінні.

Наступний крок передбачає аналіз і оцінку отриманих результатів. Результати комп'ютерних експериментів представляються у вигляді графіків, таблиць, роздруківок, а також піддаються оцінці якісних і кількісних характеристик моделювання. Для ілюстрації моделі використовується анімація. Вивчається процес створення моделі та її достовірність для підвищення рівня довіри до неї.

На підставі отриманих результатів формуються висновки щодо проведених досліджень, а також визначаються рекомендації з використання моделей та прийняття рішень. Усі етапи моделювання взаємопов'язані, а сам процес є ітераційним. Це обумовлено тим, що після виконання кожного етапу перевіряється вірність і достовірність моделі, і у випадку виявлення невідповідності здійснюється повернення до попередніх етапів для коригування та налаштування моделі. Залежно від характеру змін повертаються на попередній етап або до ранніх етапів.

На заключному етапі моделювання формулюються результати дослідження в документальній формі, і готується програмна документація для використання результатів у розробці поточних та майбутніх проектів.

Якщо моделювання складної системи створюється імітаційна модель, деякі етапи життєвого циклу моделей дуже схожі з етапами життєвого циклу програм.

Наприклад, якщо говорити про етап формулювання проблеми та змістовної постановки завдання, то одним із важливих завдань, яке необхідно вирішити на цьому етапі під час створення імітаційної моделі системи є визначення вимоги до моделі і до моделі, що моделюється. Таким чином, під час створення моделі системи можна виділити два види вимог: вимоги до системи, що моделюється, і вимоги до моделі. Вимоги до системи можна отримати з готової системи або з системи, що розробляється.

Одним з найскладніших питань в імітаційному моделюванні є визначення змісту імітаційної моделі. Завдання модельєра полягає в тому, щоб зрозуміти

реальну систему, яка є предметом імітаційного дослідження, і перетворити її на відповідну імітаційну модель.

Обрана модель може варіюватися від дуже простої моделі з одним сервером і чергою до моделі, яка намагається інкапсулювати кожен аспект системи. По суті, існує нескінченна кількість моделей, які можуть бути обрані в цьому діапазоні, кожна з яких має дещо інший зміст. Питання полягає в тому, яка модель має бути обраною.

На перший погляд може здатися, що відповідь полягає в тому, щоб побудувати модель, яка містить якомога більше деталей. Зрештою, така модель буде найбільш наближеною до реальної системи, а отже, і найбільш точною. Це могло б бути правдою, якби ми мали повне знання про реальну систему і дуже багато часу на розробку та запуск моделі.

Але що, якщо є лише обмежені знання про реальну систему і обмежений час, то потрібно розробити простішу модель, нам потрібно визначити рівень абстракції, на якому ми будемо працювати.

Якщо створюється модель програмної системи, то, швидше за все, вимоги до нею вже було висунуто на етапі аналізу вимог до системи. Після того, як вимоги до системи та моделі системи були висунуті, їх необхідно перевірити, щоб вони були несуперечливі. І для цього можна використовувати ті ж інструменти, що використовуються в життєвому циклі програми на етапі аналізу вимог до системи. Також на етапі формулювання проблеми вирішуються питання, пов'язані з визначенням цілей створення програми чи моделі, визначенням часових рамок розробки та ресурсів, які можна витратити на розробку. Виконується активна взаємодія із замовником для більш чіткого формування вимог.

Етап специфікації життєвого циклу розробки програми відповідає етапу розробки концептуальної моделі у життєвому циклі створення моделі. Специфікацію програми можна вважати її концептуальною моделлю, особливо якщо специфікація представлена у вигляді сутності, описаної деякою спеціалізованою мовою специфікацій. Завданням етапу специфікації є збирання всіх вимог та усунення суперечливості цих вимог. Етап розробки концептуальної

моделі також певною мірою призначений для усунення суперечностей, таких як створення максимально точної моделі при максимальному її спрощенні в порівнянні з реальним об'єктом, що дозволить заощадити ресурси, що витрачаються при створенні моделі, а також ресурси, що витрачаються на прогони моделі.

Найбільш схожими є етапи проектування та реалізації в життєвому циклі створення програм та етап розробки програмної реалізації моделі. Схожість проявляється особливо чітко в тому випадку, якщо для створення було обрано прикладну мову програмування. Етап розробки програмної реалізації в моделі тісно пов'язаний із розробкою концептуальної моделі. Хоча розробка імітаційної моделі значною мірою схожа на розробку звичайної програми, але розробка моделі відбувається на підставі зафіксованої архітектури, яка вибирається на основі розробки концептуальної моделі.

Етап розробки концептуальної моделі під час створення імітаційної моделі системи складається з трьох кроків:

1. Вибір ступеня деталізації опису об'єкта моделювання, на якому визначається, наскільки детально має бути описаний об'єкт моделювання. На цьому кроці необхідно вибрати між вартістю моделі та похибками моделювання. Чим вище буде ступінь деталізації об'єкта моделювання, тим довше відбуватиметься процес її розробки та калібрування і тим менше будуть відхилення від реальної системи одержаних результатів.

2. Опис змінних моделі, на якому визначаються вхідні, вихідні, внутрішні параметри та їх розподілу.

3. Формалізоване зображення концептуальної моделі, яке є самим важливим кроком на етапі розробки концептуальної моделі, яке тісно пов'язує цей етап із етапом розробки програмної реалізації моделі.

Тестування – це одна з технологій перевірки якості, воно здійснюється на стадії проектування та надалі, перед випуском програмного продукту, що розробляється. Основне завдання тестування виявити невідповідностей у роботі додатка, воно так само, є важливим етапом життєвого циклу будь-якого програми,

що розробляється. Тестування дозволить виявити невідповідності між очікуваною та реальною поведінкою програми, така перевірка здійснюється на наборі тестів, обраних певним чином. До тестування включені певні процеси такі як: проектування тестів, виконання тестування та аналіз одержаних результатів.

Всі ці процеси є невід'ємною частиною всього процесу тестування, де формуються тестові випадки чи сценарії, виконується тестування та аналізується поведінка програмного забезпечення. На рисунку 2.1 продемонстрована логічна модель виконання тестування.

Рисунок 2.1 – Логічна модель виконання тестування

Тестування програм на реальних пристроях дозволяє надати клієнтам по-справжньому якісні продукти та підвищити ефективність бізнесу у різних сферах діяльності: бізнесу, виробництва товарів та послуг тощо.

Варто зауважити, що процес тестування відбувається в рамках послідовності виконання поставлених задач, тобто здійснюється вибір програми чи застосунку, що проходитиме тестування, на відповідність вимогам встановлених при розробці та чинних стандартів якості. Після перевірки на критерії відповідності вимогам відбувається аналіз та збір інформації, в результаті

чого буде отримана інформація про невідповідності між очікуваним результатом та фактичним.

Процес тестування можна розглянути на рівні концептуальної моделі, з метою визначити перебіг процесу та всіх етапів проходження тестування Для цього було здійснено побудову спеціальної концептуальної моделі, яка є вже існуючою моделлю процесу чи функції.

Дослідження процесів є обов'язковою частиною під час проектування чи розвитку системи. Дана модель призначена для точного фіксування процесів, що здійснюються в системі, а також які об'єкти використовуються при виконанні будь-яких процесів або функцій різного рівня деталізації.

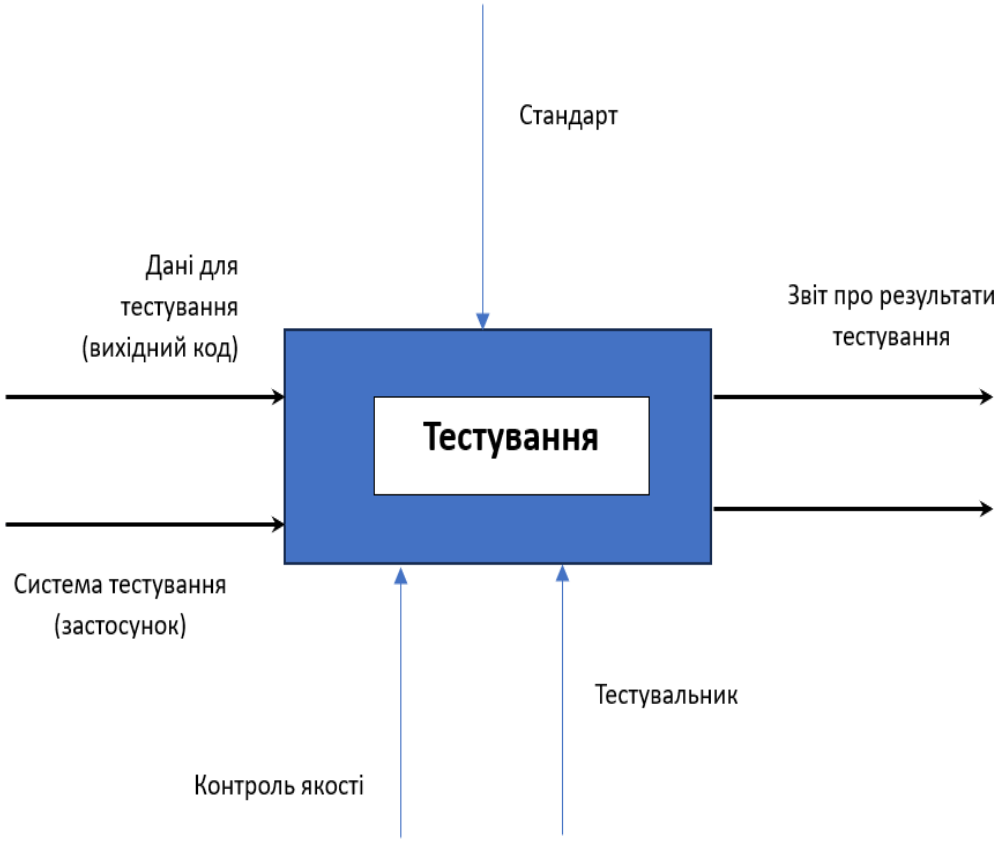


Рисунок 2.2 – Концептуальна модель роботи системи тестування

Під час побудови даної моделі, дуже важливо спроектувати модель, максимально наближену до дійсності, яка ґрунтується на природно присутніх процесах у системі.

В даному випадку, продемонстрований нижче процес, розглядається як процес тестування за допомогою готової системи тестування, з уже наявними даними, написаними сценаріями та готовими тест-кейсами. Концептуальну модель до розроблюваної системи на основі удосконаленого меду підтримки якості програмного коду продемонстровано на рисунку 2.2.

2.2 Метод підтримки якості програмного коду на основі автоматичного тестування

Тестування програмного забезпечення є одним із важливих етапів життєвого циклу програмних продуктів (QC/QA).

Виділяють також різноманітні підходи до виконання тестування, а не тільки рівні, зокрема підхід ручного тестування, підхід автоматичного виконання тестових сценаріїв. Завдання проведення ручного тестування визначає дії фахівця з тестування так, ніби він був користувачем розроблюваної системи. Під час ручного тестування відбувається моделювання різноманітних сценаріїв дій користувача. Процедура, що описує методику виконання тестування: набір і порядок виконання певних тестових випадків, діапазон значень вхідних параметрів, очікувані результати поведінки системи в кожному конкретному випадку, складається заздалегідь і документуються. Обов'язковою умовою успішного тестування є чітке розуміння з боку фахівця всіх пунктів зазначеної процедури. Детальність опису тестового випадку може бути різною, у зв'язку з чим від фахівця з тестування вимагається гнучкість у завданні вхідних даних і під час оцінювання нечітко сформульованих критеріїв результати роботи програми.

Автоматизоване тестування є аналогом ручного, однак для виконання тестів і оцінки їхніх результатів використовуються програмні засоби. За допомогою

спеціалізованих програмних інструментів здійснюється запуск тестів, ініціалізація, виконання тестового сценарію, подальший аналіз і видача результатів тестування. Автоматизація тестування дає змогу виключити вплив людського фактора на якість тестування, забезпечуючи сувору одноманітність виконання тестів, гарантоване виявлення помилок під час виконанні програми та чітке дотримання кроків алгоритму кожного тестового випадку. Разом із цим автоматизація процесу тестування значно скорочує часові витрати, виключає потреби у звірці з технічним завданням та інструкціями, характерні для ручного тестування. Слід зазначити, що звіти про результати тестування зберігаються і можуть бути надіслані всім зацікавленим особам в автоматичному режимі. Значним перевагою є і той факт, що виконання тестів відбувається без втручання фахівця з тестування і не залежить від графіка роботи фахівців.

Виокремлюють такі три рівні автоматичного тестування: рівень модульного тестування, рівень функціонального тестування та рівень тестування через користувацький інтерфейс.

На цьому початковому рівні, який є основою піраміди автоматизації тестування, розробники пишуть автоматизовані модульні тести або тести компонентів/модулів. При цьому тестувальникам дозволяється розробляти тести для перевірки коду за умови, що вони достатньо кваліфіковані для виконання такої роботи. Якщо ці тести доступні на ранніх стадіях проекту, а також якщо постійно додаються нові тести для перевірки оновлень і виправлень помилок, то ймовірність виникнення серйозних проблем в процесі розробки проекту є меншою.

Як правило, неможливо точно протестувати рівень бізнес-логіки архітектури додатку. Це може бути пов'язано з тим, що бізнес-логіка була реалізована таким чином, щоб не використовуватися користувачем. Саме тому, за погодженням з розробниками, команда тестувальників може отримати доступ безпосередньо до функціонального рівня, щоб протестувати бізнес-логіку додатку без залучення користувацького інтерфейсу.

На рівні тестування інтерфейсу є можливість протестувати як користувацький інтерфейс, так і функціональність, виконуючи операції, що стимулюють бізнес-логіку додатку. Вважається, що такі наскрізні тести є більш ефективними, ніж попередній рівень автоматизації, оскільки останній просто тестує функціональність, імітуючи поведінку кінцевого користувача із залученням інтерфейсу.

Автоматизація графічного інтерфейсу - це спосіб імітації дій користувача, таких як введення за допомогою миші та клавіатури, для автоматизації процесів, пов'язаних з кліками, введенням полів, розпізнаванням зображень та прокруткою. Хоча це не найелегантніший метод автоматизації, іноді єдиним способом введення та вилучення інформації з програми є її користувацький інтерфейс.

Досить часто для автоматизації графічного інтерфейсу пишуть так звані сценарії, тобто певний набір дій у системі користувач-програма.

Автоматизоване тестування відбувається швидше і обробляє кілька тестових кейсів одночасно. Крім того, фреймворки автоматизації підтримують запис результатів, скріншоти та деякі сучасні функції для тестування. Це дозволяє паралельне виконання тестів, що робить його хорошим варіантом для кросбраузерного тестування.

Очевидно, що автоматичне тестування має ряд переваг: підвищена точність, зменшення витрат, швидке виконання, зменшення витрат, більш достовірні результати, тощо. Однак, також варто зазначити, що автоматизація у деяких випадках поступається ручному методу тестування, а саме через складність написання тестових сценаріїв, відсутністю випадковості, нелінійності та гнучкості, що притаманно людині під час проведення тестування.

Вказані недоліки можна усунути, якщо спростити створення тестів із додаванням випадковості та гнучкості у поведінку скриптових сценаріїв, адже вони виконується за жорстким алгоритмом.

Для вирішення вказаних задач, можна використовувати нейронні мережі, що навчаються за допомогою алгоритму зворотного поширення на наборі тестових кейсів, застосованих до оригінальної версії системи. Навчання мережі

базується на підході "чорної скриньки", оскільки алгоритму надаються лише входи та виходи системи. Навчена мережа може бути використана як штучний приклад для оцінки правильності виводу нових і, можливо, помилкових версій програмного забезпечення.

Штучні нейронні мережі імітують роботу мозку людини, що являє собою дуже складну систему обробки інформації, що здійснюється нелінійно та паралельно. Для досягнення високопродуктивності потрібно, щоб нейронні мережі так само як і людський мозок використовували велику кількість різноманітних взаємозв’язків між найменшими елементами, тобто нейронами.

Підгрунтям для розробки усіх штучних нейронних мереж лежить модель нейрона, що зображено на рисунку 2.3.

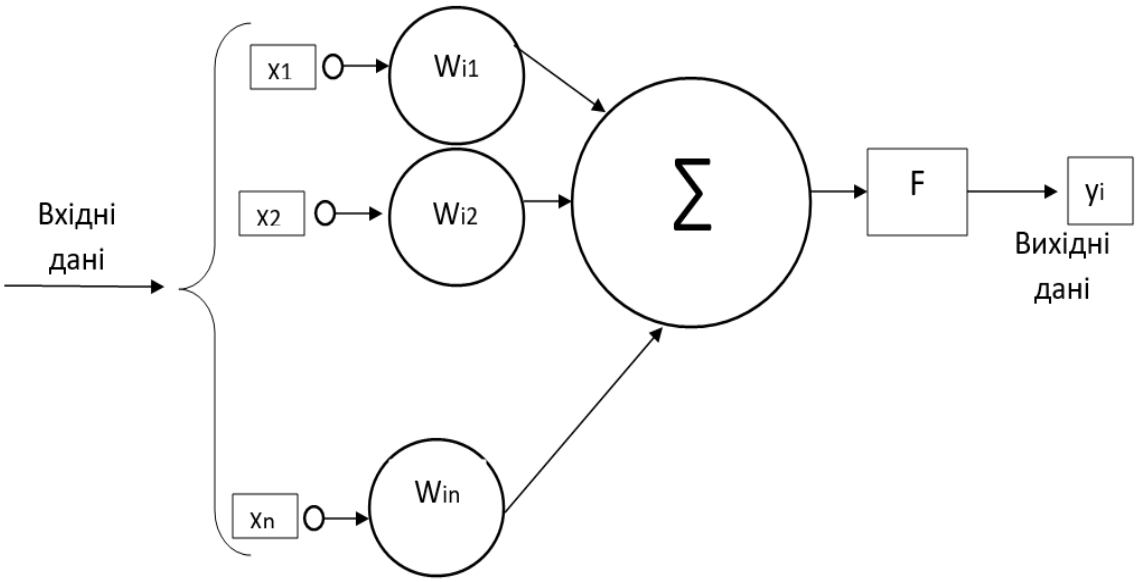


Рисунок 2.3 – Модель штучного нейрона

Нейронна мережа - це модель машинного навчання, розроблена для імітації функцій і структури людського мозку. Нейронні мережі - це складні мережі взаємопов'язаних вузлів, або нейронів, які співпрацюють для вирішення складних проблем.

Нейронні мережі, які також називають штучними нейронними мережами або глибокими нейронними мережами, являють собою тип технології глибокого навчання, що належить до більш широкої галузі штучного інтелекту (ШІ).

Штучна нейронна мережа зазвичай з багатьох процесорів, які працюють паралельно і розташовані ярусами або шарами. Перший рівень - аналог зорових нервів у зоровому аналізаторі людини - отримує необроблену вхідну інформацію. Кожен наступний рівень отримує вихідні дані від попереднього рівня, а необроблену вхідну інформацію - так само, як нейрони, віддалені від зорового нерва, отримують сигнали від тих, що розташовані ближче до нього. Останній рівень виробляє вихід системи.

Математично все вище описане подається так:

$$net = \sum_{j=1}^{n} x_j \, w_{ij} \tag{2.1}$$

$$y_{i=\frac{1}{1+e^{-net}}} \tag{2.2}$$

де $x_1, x_2, \ldots . x_n$ – вхідні дані,

$w_{i1}, w_{i2}, \ldots . w_{in}$ – ваги нейрона і;

y_i – вихідні дані нейрона.

Відповідно до властивостей існує поділ нейронів на такі групи:

- вхідні нейрони;
- вихідні нейрони;
- нейрони прихованого шару (обчислювальні вузли).

Багатошарова нейронна мережа прямого поширення - це взаємозв'язок перцептронів, в якому дані та обчислення протікають в одному напрямку, від вхідних даних до вихідних. Кількість шарів у нейронній мережі дорівнює кількості шарів перцептронів. Найпростіша нейронна мережа - це мережа з одним вхідним шаром і вихідним шаром перцептронів. Технічно вона називається одношаровою мережею прямого поширення з двома виходами, оскільки вихідний

шар є єдиним шаром з обчисленням активації. В якості методу навчання нейронної мережі прямого поширення може використовуватись зворотне поширення помилки. Даний тип алгоритму заключається у тому, що вхідні дані (сигнал) передається на початковому напрямку, відповідно сигнал помилки – передається у зворотному напрямку. На етапі прямого проходу вхідні дані (сигнали) подаються на вхідний шар нейронної мережі, а після цього він поступово поширюється мережею від одного шару до наступного. Результатом даних маніпуляцій стає набір вихідних сигналів, що є реакцією мережі на вхідні сигнали. Під час прямого проходу всі синаптичні ваги залишаються постійними, в той час як під час зворотного проходу ваги налаштовуються відповідно до принципу корекції помилок. Процес налаштування ваг відбувається наступним чином: фактичний вихід мережі віднімається від очікуваного, і це призводить до формування сигналу помилки. Даний сигнал розповсюджується по мережі у напрямку, протилежному синаптичним зв'язкам.

Структура описаної мережі показана на рисунку 2.4.

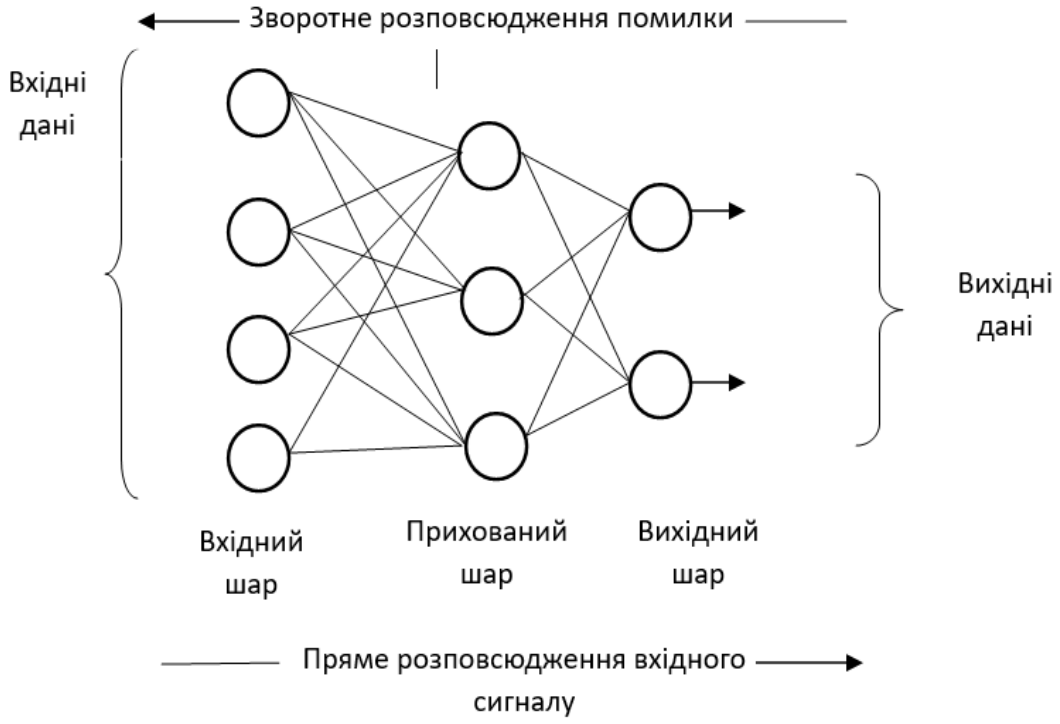


Рисунок 2.4 – Структура багаторівневої нейронної мережі

Фактично, нейронна мережа є специфічним методом задання функції. Для взаємодії з нею необхідно, передусім, провести її навчання. Процес навчання включає підготовку навчального набору, який представляє собою багато пар вхідних даних і відповідних їм правильних вихідних векторів. Елементи цього набору повинні бути незалежними для можливості розширення та подання їх у довільному порядку. Навчальний набір пройде через нейронну мережу, при цьому синаптичні ваги в нейронних зв'язках коригуються так, щоб задовольняти вимогам навчального набору. Якість навчання визначає ефективність роботи нейронної мережі. Другим етапом є перевірка результатів навчання мережі, для чого готується тестовий набір, аналогічний навчальному, але з іншими значеннями. Якщо результат тестування виявляється незадовільним, необхідно вдосконалити навчання мережі або змінити її структуру.

На рисунку 2.4 подано геометричну модель даного етапу, що має вигляд множини (D) [24].

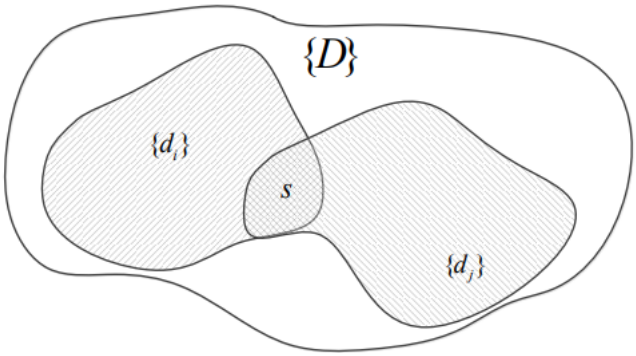


Рисунок 2.4 - Геометрична модель методу підтримки якості програмного коду на основі автоматичного тестування

Множина D у відтвореній моделі містить підмножини та може бути представлена у вигляді [24]:

$$D = \{D_i, D_j\}, \tag{2.3}$$

де, D_i – множина робіт із виконання тестів на функціональність, продуктивність, безпека тощо;

Dj – множина робіт із перевірки відповідності вимогам технічного завдання та виявлення критичних та другорядних помилок.

В множині D між підмножинами існують елементи, які належать їх перетину:

$$D_i \cap D_j, \tag{2.4}$$

Після виконання робіт множини D буде отримано програмну систему, яка протестована. Виконання тих робіт, які будуть належати перетину підмножин множини D призводять до отримання нового результату, а саме:

$$\{s\}=R(D_i \cap D_j), \tag{2.5}$$

де s – пройдені тест-кейси (test cases) або/та звіти про помилки (bug reports).

Після того, як здійснено завершення процесу тестування життєвий цикл програмного забезпечення (програмного продукту, програмної системи тощо) переходить до етапу інтеграції (впровадження) програмного продукту шляхом його розгортання на стороні клієнта передача клієнту (замовнику) [24].

Метод підтримки якості програмного коду полягає у тому, що на основі формування звіту із результатами автоматичного тестування здійснюється оцінка якості програмного продукту, шляхом виявлення серйозних та другорядних помилок в залежності від поставлених вимог.

2.3 Алгоритм процесу автоматичного тестування із визначенням якості програмного коду

Для демонстрування роботи був побудований алгоритм покрокового виконання роботи системи, який реалізується покроково, що показано на рисунках 2.1-2.3.

На рисунку 2.1 зображено етап, на якому необхідно зробити вибір методу тестування.

Формування результатів від самого початку визначається при виборі методики тестування, яка буде використовуватися. Відповідно вже заздалегідь відомо, який результат буде сформовано в кінцевому підсумку (може бути у вигляді графіка чи діаграми).

Безпосередня реалізація системи тестування може бути у вигляді, наприклад мобільного застосунку.

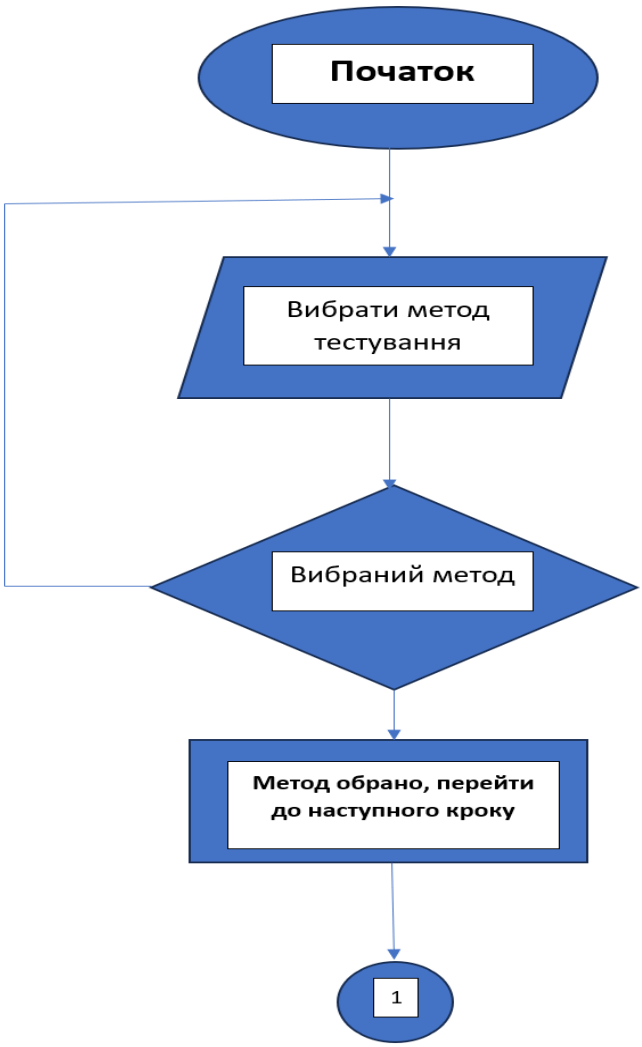


Рисунок 2.1 – Покроковий алгоритм виконання дій

На даному етапі розпочинається робота системи, при цьому буде необхідність здійснити вибір необхідного методу тестування. Після того, як тестувальник (користувач, розробник) вибрали необхідній метод тестування, здійснюється перехід до наступного кроку, а саме налаштування тестів 9тестової системи), що буде використовувати нейромережеві технології.

Рисунок 2.2 – Наступний етап алгоритму виконання дій

На наступному етапі (рисунок 2.2) необхідно вказати в налаштуваннях параметрів тестів дані для тестування, і якщо ці дані вказані вірно, то здійснити запуск тестування.

Рисунок 2.3 – Заключний етап процесу тестування

На рисунку 2.3. зображено заключний етап процесу тестування, після виконання якого можна побачити всі необхідні результати на виявлення помилок у програмному коді.

2.4. Висновки до 2-го розділу

Отже, у другому розділі визначено логічну та концептуальні моделі системи тестування, алгоритм роботи системи тестування програмного забезпечення із використанням методу підтримки якості програмного коду.

Концептуальна модель це певна абстрактна конструкція, яка визначає, вказує та пояснює причинно-наслідкові зв'язки та властивості об'єкта, що досліджується. Дані причинно-наслідкові зв’язки відповідають меті наукового аналізу. Дана модель - це формальний опис створюваного об'єкта моделювання та є відображенням концепції або визначеної точки зору для вирішення проблеми. Вона як правило може включати чітку логіку, визначені алгоритми, певні припущення або обмеження, що відносяться до певного дослідження.

Метод підтримки якості програмного коду полягає у тому, що на основі формування звіту із результатами автоматичного тестування здійснюється оцінка якості програмного продукту, шляхом виявлення серйозних та другорядних помилок в залежності від поставлених вимог.

Для визначення якості та помилок у програмному коду використано аналізатор, який заснований на роботі нейронних мереж - особливого способу задання функції. Для роботи із такою мережею перш за все потрібно навчити мережу. У процесі такого навчання здійснюється підготовка навчальної вибірки, що являє собою велику кількість пар вхідних і відповідних їм вірних вихідних векторів.

Елементи вказаної множини повинні бути незалежними для можливості розширення вибірки та подання її в довільному порядку. Навчальна вибірка

проходить через нейронну мережу, синаптичні ваги в нейронних зв'язках коригуються таким чином, щоб визначити функцію, яка задовольняє навчальну вибірку. Якість роботи нейронної мережі при цьому визначає якість навчання. Другим етапом є перевірка результату навчання мережі, для чого готується тестова вибірка, аналогічна до навчальної, але на інших значеннях. Якщо результат тестування буде визнано незадовільний, необхідно донавчити мережу або змінити її структуру.

Формування результатів від самого початку визначається при виборі методики тестування, яка буде використовуватися. Відповідно вже заздалегідь відомо, який результат буде сформовано в кінцевому підсумку (може бути у вигляді графіка чи діаграми).

3 АРХІТЕКТУРА СИСТЕМИ ТЕСТУВАННЯ

3.1 Формування та аналіз вимог програмної реалізації системи тестування

Визначення та формування вимог під час розробки будь-якого програмного забезпечення є одним із найголовніших етапів життєвого циклу програмного забезпечення.

Цьому призначенню добре відповідає так звана загальноприйнята модель життєвого циклу програмного забезпечення, згідно з якою програмні системи проходять у своєму розвитку дві фази: фазу розроблення і фазу експлуатації та супроводу [1].

Розробка починається з ідентифікації потреби в новому застосунку, програмній системі тощо, а закінчується передачею продукту розробки в експлуатацію.

Першим етапом фази розробки є постановка задачі та визначення вимог. Визначення вимог включає опис загального контексту завдання, очікуваних функцій системи та її обмежень. На цьому етапі замовник спільно з розробниками приймає рішення про створення системи. Найбільш критичний даний етап для складніших застосунків.

Якщо приймається позитивне рішення, то відбується оформлення технічного задання, де описуються вимоги до системи. У разі позитивного рішення починається етап специфікації системи відповідно до вимог, що оформляється відповідним чином у технічне завдання.

Важливо підкреслити, що призначення цих специфікацій - описувати зовнішню поведінку системи, яку системи, що розробляється, а не її внутрішню організацію, тобто відповідати на запитання, що вона має робити, а не як це буде реалізовано. Тут ідеться про призначення, а не про форму специфікацій, оскільки на практиці за відсутності відповідної мови специфікацій, на жаль, нерідко доводиться вдаватися до опису «що» за допомогою «як». Перш ніж приступати до створення проєкту за специфікаціями, вони мають бути ретельно перевірені на

відповідність вихідним цілям, повноту, сумісність (несуперечливість) і однозначність.

Оскільки технічне завдання або специфікації вимагають строго та чітко описувати очікування від реалізації певного проекту, то виникають певні проблеми із описом вимог. Більшою мірою вони пов'язані з необхідністю домагатися і підтримувати відповідність опису "що" нечітким, неточним і часто суперечливим вимогам з боку зовнішніх щодо проєкту людей. Немає підстав вважати, що ці люди будуть знайомі з "найкращою мовою специфікацій", що вони дбатимуть про коректність своїх вимог. Завдання етапу специфікацій у тому і полягає в тому, щоб опис програми вибудувати у вигляді логічно вивіреної системи, зрозумілою як для замовника цієї розробки, майбутніх користувачів, так і для виконавців проєкту.

Розроблення проєктних рішень, що відповідають на запитання, як має бути реалізовано систему, щоб вона могла задовольняти специфіковані вимоги, виконується на етапі проєктування. Загалом система може бути досить складною структурою, тому важливим етапом на даному етапі є її розбиття на певну кількість модулів чи процедур, тобто здійснення декомпозиції.

На наступному етапі реалізації або кодування кожен із цих модулів програмується мовою, яка найбільше підходить для цієї програми. З точки зору автоматичності даний етап є найбільш досконалим.

У розглянутій моделі фаза розроблення закінчується етапом тестування (автономного і комплексного) і передачею системи в експлуатацію.

Фаза експлуатації та супроводу охоплює всю діяльність із забезпечення нормального функціонування програмних систем, зокрема фіксування виявлених під час виконання програм помилок, пошук їхніх причин і виправлення, підвищення експлуатаційних характеристик системи, адаптування системи до потреб зовнішнього середовища і якщо є така потреба, то можливе покращення системи загалом.

Усе це дає право говорити про еволюцію системи. У зв'язку з цим фаза експлуатації та супроводу розбивається на два етапи: безпосередньо супровід і

розвиток. В низці випадків на цю фазу припадає більша частина коштів, що витрачаються в процесі життєвого циклу програмного забезпечення.

Очевидно, що загалом увага розробників до певних етапів розробки відповідає конкретно взятому проекту. Часто розробнику немає необхідності проходити через усі етапи, наприклад, якщо створюється невелика, добре зрозуміла, програма з ясно поставленою метою. Проблеми супроводу, які погано розуміють розробники невеликих програм для особистого користування, є водночас дуже важливими для великих систем.

Така коротка характеристика загальноприйнятої моделі. У літературі зустрічається багато варіантів, що розвивають її в бік деталізації та додавання проміжних фаз, етапів, стадій і окремих робіт (наприклад, із документування та технологічної підготовки проєктів) залежно від особливостей програмних проєктів або уподобань розробників.

Розглянемо етапи життєвого циклу моделі. Основою моделювання є методологія системного аналізу [2]. Це дає змогу досліджувати систему або процес з використанням технології операційного дослідження, що включає такі взаємопов'язані етапи .

На першому етапі замовник формулює проблему. Організовуються зустрічі керівника проєкту із замовником, аналітиками з моделювання та експертами з проблеми, що вивчається. Вивчаються цілі дослідження та спеціальні запитання, відповіді на які будуть отримані за результатами дослідження; встановлюються критерії оцінювання роботи, які використовуються для вивчення ефективності різних конфігурацій системи; розглядаються такі показники системи, як масштаб моделі, період досліджень і необхідні ресурси; визначаються конфігурації системи, а також необхідне програмне забезпечення.

На цьому ж етапі проводять цілеспрямоване дослідження модельованої системи або процесу, залучають експертів із проблеми, які володіють достовірною інформацією. Збирається інформація про конфігурацію системи, і розглядаються способи експлуатації для визначення параметрів моделі та вихідних розподілів імовірностей.

Тестування на основі вимог - це тип тестування, за якого оцінюють, чи відповідає система функціональним і нефункціональним вимогам, зазначеним замовником. Основна мета тестування на основі вимог - визначити, чи виконує розроблений програмний продукт усі поставлені завдання, чи ні, і є гарантією того, що не буде якихось протиставлень між очікуваним та отриманим. Такий варіант використання тестування пропонує виявляти певні неточності на початковій стадії розробки програмного забезпечення, що пов’язані із такими характеристиками як точність, зрозумілість, надійність, безпечність, масштабованість. Тести на основі вимог можна розділити на тести чорної скриньки (які перевіряють, наскільки добре додаток працює відповідно до очікувань користувача) і тести білої скриньки (які перевіряють кожен аспект коду). Тести, виконані з використанням цього підходу, повинні включати як позитивні, так і негативні тести. Тестувальникам важливо мати повне уявлення про вимоги, щоб розробляти ефективні тестові випадки та сценарії. Тестування на основі вимог є невід'ємною частиною процесу тестування програмного забезпечення, що допомагає розробникам створювати якісні продукти.

Використовуючи тестування на основі вимог, організації можуть гарантувати, що вони постачають високоякісні програмні додатки відповідно до очікувань клієнтів. Такий підхід підвищує довіру користувачів до продукту, а також знижує загальні витрати, пов'язані з розробкою та обслуговуванням. Результати цього типу тестування можна використовувати для виявлення будь-яких слабких ланок у системі та вжиття відповідних коригувальних заходів. Ба більше, гарантуючи, що розроблений продукт відповідає всім заданим вимогам, компанії можуть уникнути дорогих переробок або переробок у майбутньому.

Важливою вимогою для проведення адекватного тестування є розробка сценаріїв. Тестовий сценарій - це однорядковий оператор, який повідомляє, яку область у додатку буде перевірено. Сценарії тестування використовуються, щоб гарантувати, що всі технологічні процеси тестуються від початку до кінця. У конкретній ділянці застосунку може бути від одного тестового сценарію до кількох сотень сценаріїв залежно від масштабу і складності застосунку.

Загалом терміни «тестовий сценарій» і «тестові випадки» використовуються взаємозамінно, проте тестовий сценарій складається з декількох етапів, тоді як тестовий приклад складається з одного етапу. З цієї точки зору тестові сценарії є тестовими прикладами, але вони включають в себе кілька тестових випадків і послідовність їх виконання. Крім того, кожен тест залежить від результатів попереднього тесту.

Тестові випадки включають набір кроків, умов і вхідних даних, які можна використовувати під час виконання завдань тестування. Основна мета цієї діяльності полягає в тому, щоб переконатися, що програмне забезпечення пройшло або не пройшло з точки зору його функціональності та інших аспектів. Існує багато типів тестових прикладів, таких як функціональні, негативні, з помилками, логічні тестові приклади, фізичні тестові приклади, тестові приклади користувальницького інтерфейсу тощо.

Крім того, тестові випадки написані для відстеження охоплення тестування програмного забезпечення. Як правило, немає ніяких формальних шаблонів, які можна використовувати під час написання тестового прикладу. Тим не менш, такі компоненти завжди доступні та включені до кожного тестового прикладу:

- ідентифікатор тесту;
- модуль продукту;
- версія продукту;
- лист реєстрацій змін;
- мета;
- припущення;
- передумови;
- заходи;
- очікуваний результат;
- фактичний результат;
- постумови.

Багато тестових випадків можуть бути отримані з одного тестового сценарію. Крім того, іноді для одного програмного забезпечення написано кілька тестових випадків, які в сукупності відомі як набори тестів.

Для основних прецедентів системи тестування можна продемонструвати схематичну діаграму використання, де тестувальник (актор) має свою роль, взаємодіє із системою, а прецедентами є варіанти використання або сервісні запити, які забезпечуються системою. На діаграмі лініями вказано відношення асоціацій, які демонструють здатність застосування актором прецедента.

Рисунок 3.1 – Процес виконання тестування

На рисунку 3.1 продемонстровано дії, що повинен проробити тестувальник, для виконання тестування:

1. Здійснити запуск застосунку (мобільного чи десктопного).
2. Здійснити налаштування необхідних тестів.
3. Вказати необхідні дані для тестування.

4. Вибрати необхідний метод тестування та запустити систему на виконання тестування.
5. Отримати результати тестування та здійснити аналіз результатів.

3.2 Розробка архітектури системи тестування програмного коду

Для того, щоб здійснити перевірку моделі на якість, доцільність та правильність використовуються статистичні критерії, що є формальними. Однак така перевірка можлива за наявності надійних статистичних параметрів як оригіналу (об'єкта прогнозування), так і моделі. Якщо з якихось причин такі оцінки відсутні, то здійснюють порівняння окремих властивостей оригіналу і моделі. При цьому спочатку мають перевіряти істинність реалізованих функцій, що реалізуються, потім істинність структури і, нарешті, істинність значень параметрів, що досягаються при цьому. Для цього, крім моделі, необхідно мати функціонуючий оригінал, тобто проводити супровідне моделювання.

Верифікація моделі - оцінювання її функціональної повноти, точності та достовірності з використанням усієї доступної інформації в тих випадках, коли перевірка адекватності з тих чи інших причин неможлива.

Найчастіше використовують верифікацію моделей, оскільки в більшості випадків реальний об'єкт відсутній або розробляють нові (ще не існуючі) функції об'єкта моделювання.

Ступінь досконалості моделі виражають через різні вимірники точності.

Точність точкового результату експерименту в момент f визначається різницею між значенням точкового експерименту P і фактичним значенням Fh модельованого показника в певний момент часу. Окремий точковий експеримент не визначає точність процедури моделювання загалом, тобто потрібна деяка вибірка {(Pj, FJ)}, на основі якої розраховується значення деякого вимірника точності моделювання.

Наразі немає достатньо повного дослідження всіляких критеріїв точності, що ускладнює оцінювання адекватності різних моделей і досвіду їх застосування в моделюванні систем і процесів.

Для перевірки адекватності моделей можна використовувати будь-який коефіцієнт парної кореляції між послідовностями змодельованих і фактичних значень.

Класичний критерій перевірки адекватності моделі - коефіцієнт кореляції Пірсона.

Перевірка моделі на адекватність є одним із критеріїв завершення ітерації життєвого циклу моделі поряд із модульним та інтеграційним тестуванням програмної реалізації моделі. Перевірка моделі на адекватність виконується в тому випадку, якщо було проведено експериментування з реальним об'єктом або процесом.

У разі, якщо дані спостережень над реальним об'єктом або процесом відсутні, необхідно проводити верифікацію моделі. Перевірка моделі на адекватність (верифікація в разі відсутності даних спостереження) схожа за своєю природою на тестування.

Завданням тестування є порівняння вихідних даних з еталоном, завданням перевірки моделі на адекватність є перевірка адекватності відгуку за заданих вхідних значеннях фактора. У разі верифікації моделі, коли відсутні дані спостереження за реальним об'єктом або процесом, порівняння даних не відбувається, а проводиться дослідження набору відгуків моделі з використанням набору спеціальних методів.

Таким чином, відмінністю процесу тестування моделі від процесу перевірки її на адекватність є те, що під час перевірки моделі на адекватність проводиться перевірка наборів вхідних і вихідних даних, а не значень одиничного експерименту. Під час тестуванні найчастіше відбувається порівняння вихідного результату функції з еталоном. Таким чином, для того щоб забезпечити автоматизацію процесу перевірки моделі на адекватність (верифікацію), необхідно інструментально розширити засіб тестування додатковим

функціоналом для подачі на вхід тесту наборів даних, наприклад із бази даних, а також функціоналом для проведення кореляційного аналізу, на підставі якого буде застосовуватися рішення про адекватність моделі.

Так само як і під час розроблення програмного забезпечення, застосовується командний підхід: коли над проєктом працюють кілька розробників одночасно. Командний підхід вимагає набору спеціальних інструментальних засобів, які забезпечують ефективну роботу над проєктом. Важливими інструментами під час командній роботі над проєктом є система контролю версій, сервер безперервної інтеграції та фреймворки модульного і функціонального тестування. Зв'язок інструментальних засобів командної розробки програм представлено на рис. 3.1.

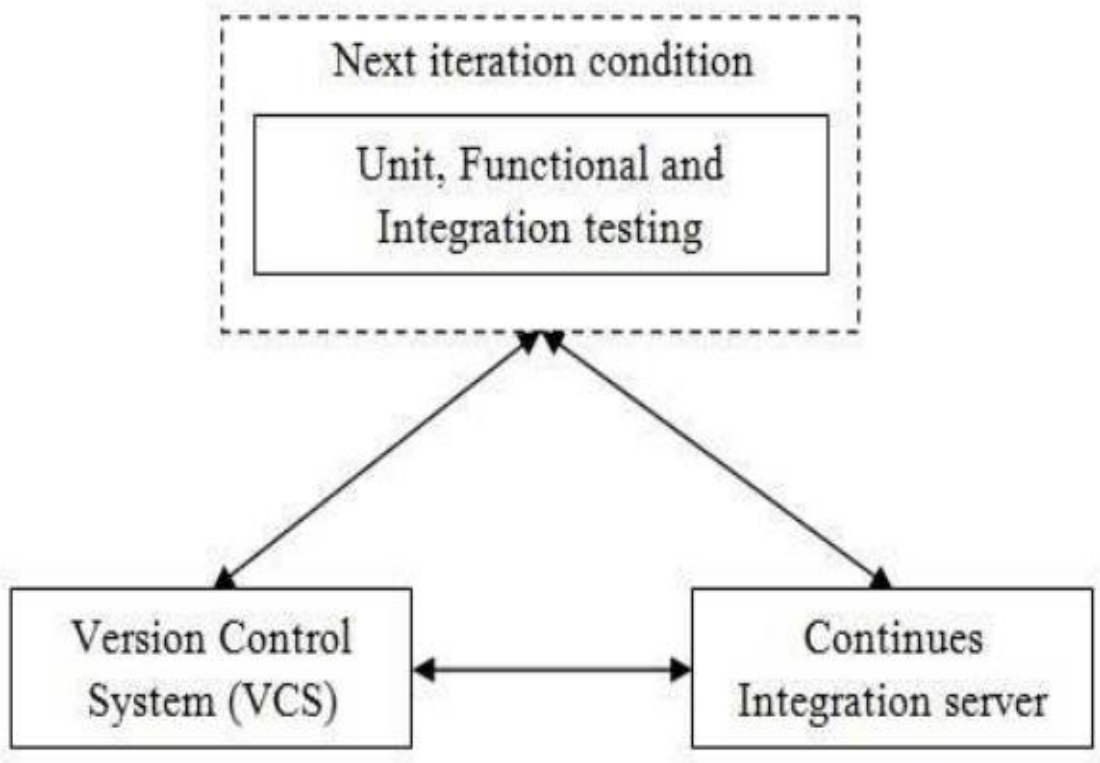


Рисунок 3.1 – Зв'язок інструментальних засобів командної розробки програмного забезпечення

Ці інструментальні засоби допомагають зберігати поточний стан файлів, пов'язаних із проєктом (систем контролю версій VCS), а також здійснювати прогін тестів (сервер безперервної інтеграції CI). Модульні та функціональні

тести використовуються для того, щоб визначити, чи не настав регрес у програмному коді, а також для визначення можливості завершення поточної ітерації.

З огляду на те, що для визначення завершення ітерації при створенні моделі необхідно не тільки провести тестування програмної реалізації моделі, а також провести її верифікацію, необхідно розширити набір інструментальних засобів командного розроблення інструментальними засобами автоматичної верифікації моделей. При цьому зв'язок інструментальних засобів командної розробки моделей матиме вигляд, що показано на рисунку 3.2

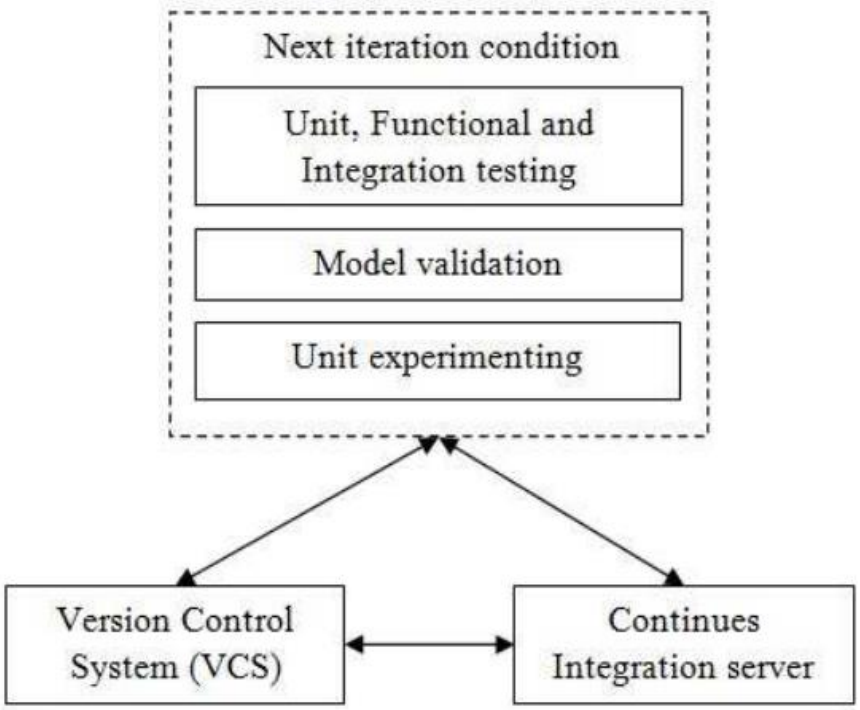


Рисунок 3.2 - Зв'язок інструментальних засобів командної розробки програм

Для проектування архітектури системи тестування буде використано напрацювання, що описані у попередньому розділі, а саме моделі нейронних мереж, оскільки штучні нейронні мережі можуть бути застосовані для вирішення завдання автоматизованого тестування програмного забезпечення. Було проведено експерименти, що доводять ефективність генерації тестових сценаріїв,

здатних виявляти несправності в роботі ПЗ, за допомогою штучних нейронних мереж.

Багаторівнева нейронна мережа може бути навчена відповідно до тестованого заастосунком на основі згенерованих у довільному порядку вхідних даних для тестування, які відповідають технічному завданню або специфікації згідно висунутих вимог. В результаті навчання нейронна мережа здатна показувати достатній ступінь точності в рамках програми, що тестується. Таким чином, навчена нейронна мережа стає симульованою моделлю програмного продукту.

Згодом, коли буде доопрацьовано функціонал розроблюваного програмного продукту і створено нові версії програми, знадобиться проведення так званого регресійного тестування. Цей вид тестування спрямований на перевірку змін, реалізованих у додатку або навколишньому середовищі. Як такі зміни може виступати новий функціонал, налагодження дефекту, міграція на нову базу даних, операційну систему або сервер додатків. У такому разі проводиться регресійне тестування з метою перевірки, що в новій версії додатку справно працює існуюча функціональність. Таким чином, у разі застосування в процесі тестування штучних нейронних мереж, під час регресійного тестування розглянутий програмний продукт перевіряється на тестових даних для отримання вихідних результатів, які потім порівнюються з результатами нейронної мережі.

Припустивши, що нові версії програмного продукту не змінюють наявний функціонал, можна очікувати, що за однакових вхідних даних нейронна мережа і реальний застосунок видадуть ідентичні результати. За допомогою інструменту для порівняння можна буде зробити висновок, чи коректний чи коректний висновок додатка, що тестується. Описана методика тестування представлена на рисунку 3.3.

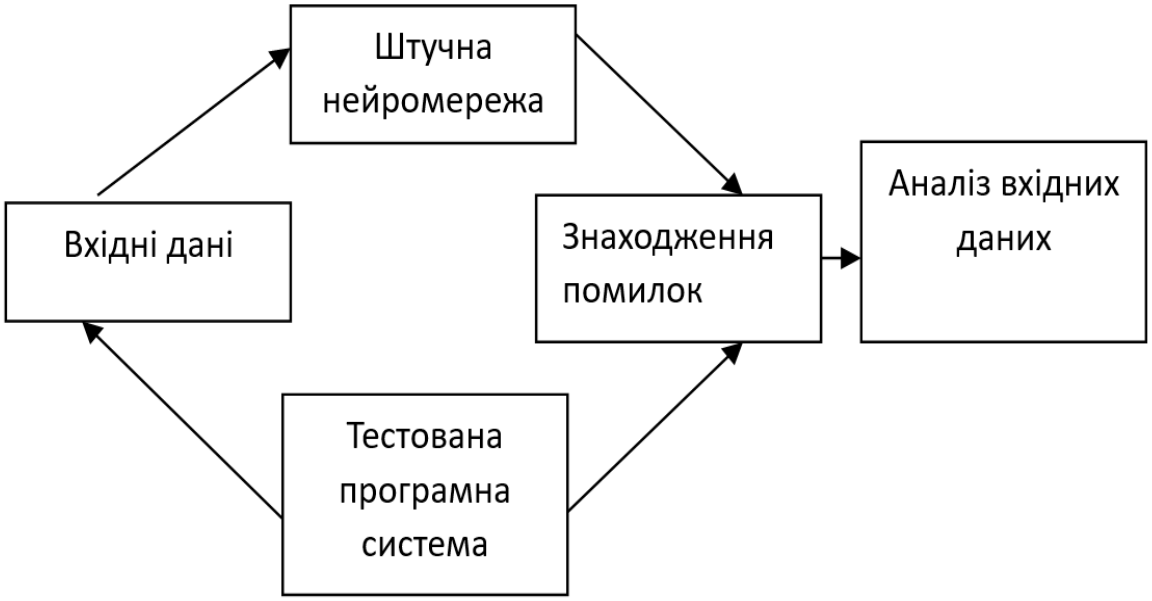


Рисунок 3.3 – Методика тестування

Використання заснованої на штучній нейронної мережі моделі програмного забезпечення замість оригінальної версії програмного продукту може бути ефективним з цілої низки причин. По-перше, оригінальна версія може стати непридатною для тестування через зміни в апаратній платформі або операційній системі.

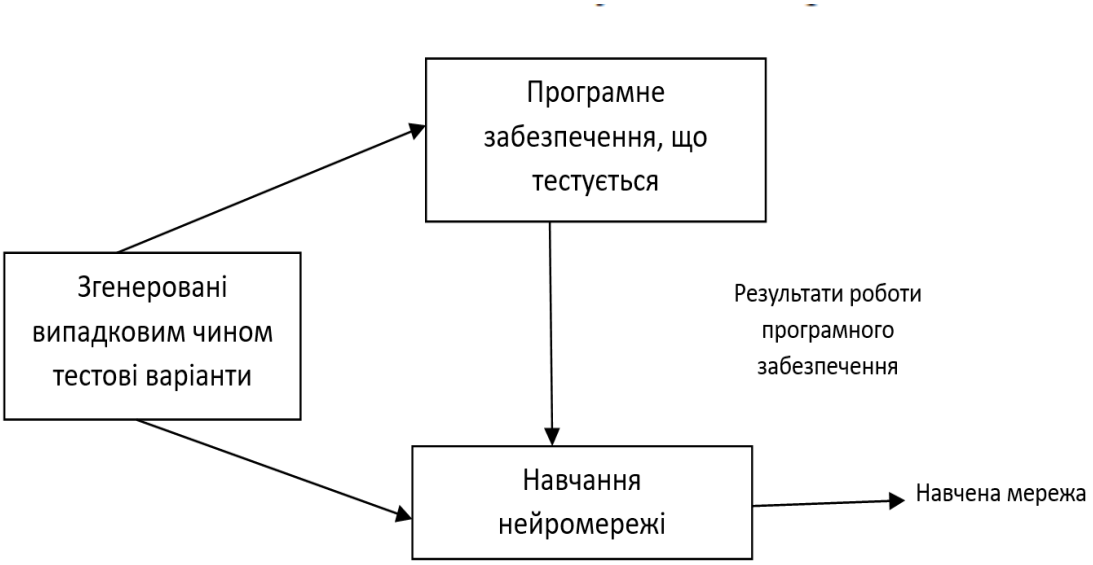


Рисунок 3.4 – Навчальна фаза нейронної мережі

По-друге, більшість пар вхідних і вихідних даних вихідного додатка може бути некритичною на даному етапі процесу тестування, і, таким чином, використання нейронної мережі для автоматизованого моделювання вихідного додатка може зберегти істотну кількість комп'ютерних ресурсів.

По-третє, збереження вичерпного набору тестових сценаріїв із вихідними значеннями початкового додатка може бути нездійсненним для реальних додатків.

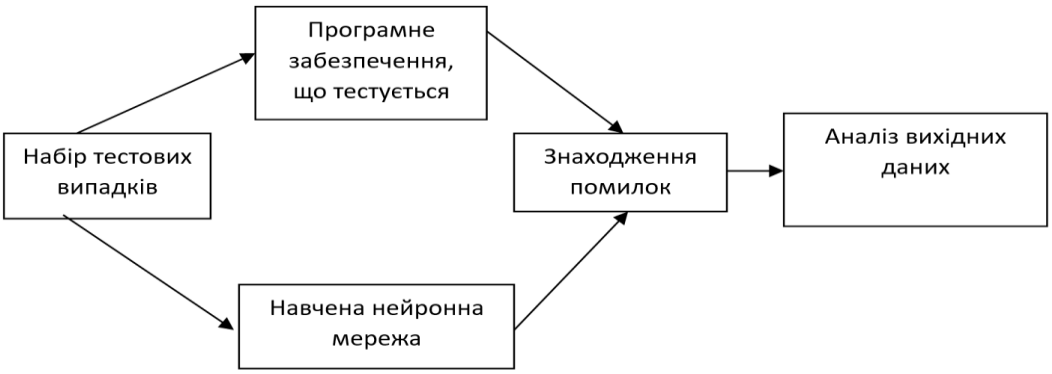


Рисунок 3.5 – Тестувальна фаза

Разом з тим, слід відмітити, що у будь-якому разі відсутня гарантія того, що початкова версія не містить помилок. Порівняння результатів роботи початкової та доопрацьованої версії може пропустити випадки, де обидві версії функціонують некоректно.

Нейронні мережі в цьому випадку забезпечують додатковий параметр для перевірки надійності тестування.

Оскільки програмне забезпечення, яке мережа навчена моделювати, було оновлено до нової доопрацьованої версії, аналогічним чином навчена нейронна мережа може вчитися класифікувати й нові дані.

Іншими словами, нейронна мережа здатна до навчання на нових версіях програмного продукту, що постійно розвивається програмного продукту, що постійно розвивається.

Крім описаного застосування штучної нейронної мережі в завданні автоматизації тестування, можуть бути розглянуто й інші аспекти використання штучних нейронних мереж.

Наприклад, важливим є завдання генерації вхідних даних для тестових випадків, які з найбільшим ступенем імовірності приведуть до виявлення помилки в роботі програмного продукту.

В результаті буде отримано архітектуру системи тестування програмного забезпечення із використанням аналізатора якості вихідного програмного коду, заснованого на нейромережевих технологіях (рисунок 3.6).

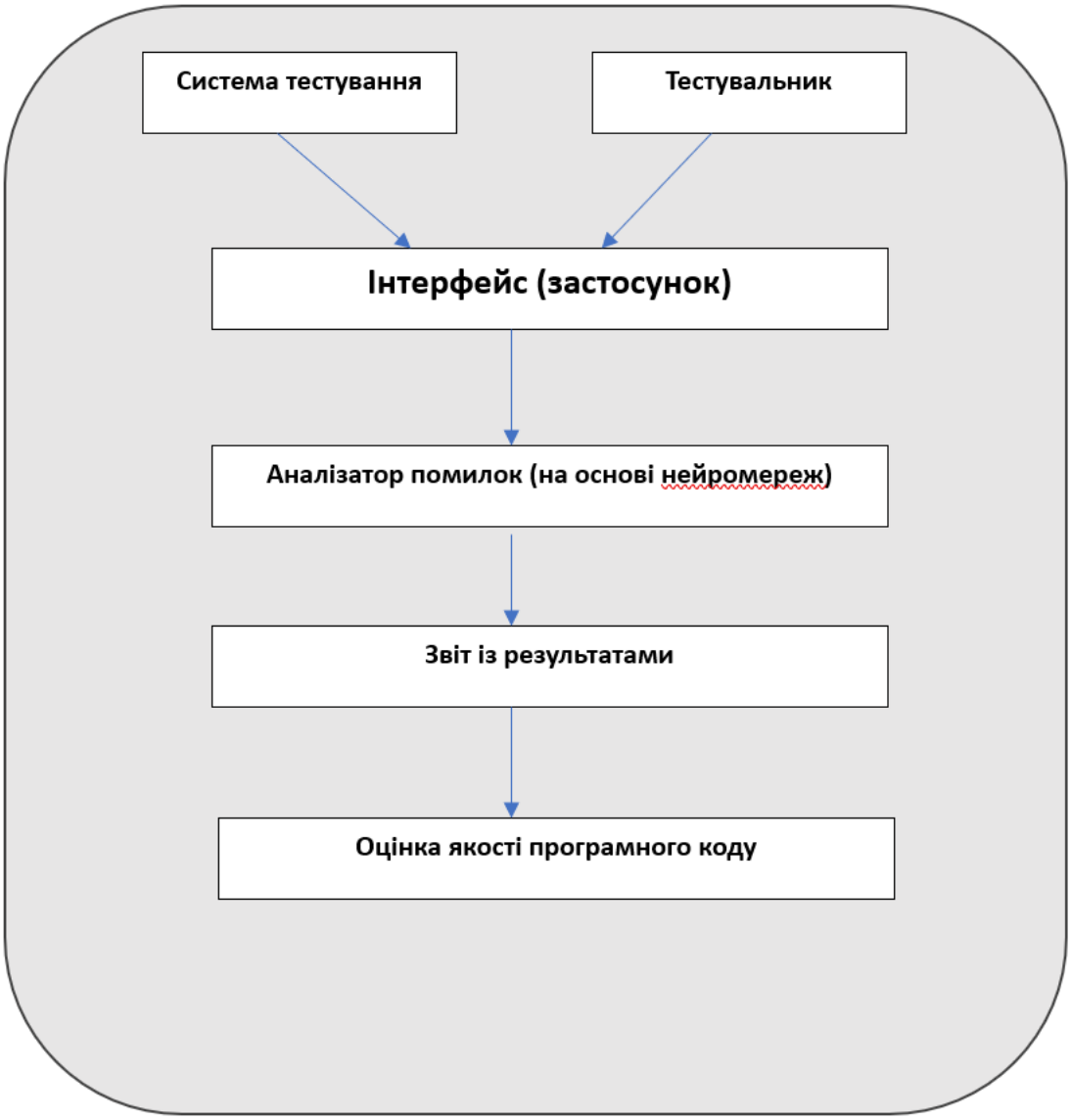


Рисунок 3.6 – Архітектура системи тестування

Таким чином для забезпечення стійкості та надійності, а також підтримки якості розроблюваного програмного коду можна використовувати різноманітні підходи, що мають свої переваги та недоліки.

Використання нейромережевих моделей у системі автоматизованого тестування дозволяє спростити хід тестування та визначити всі помилки та недоліки, що містяться у програмному коді будь-якого програмного забезпечення.

3.3. Висновки до 3-го розділу

У третьому розділі здійснено опис методу підтримки якості програмного коду, подано алгоритм роботи системи підтримки якості програмного коду, архітектура якої передбачає використання аналізатора, що заснований на нейронних мережах.

У висновку слід зазначити, що нейромережеві технології спільно з використовуваними методами автоматизації тестування здатні поліпшити результати автоматичного тестування в задачах тестування складних програмних систем.

4 ОЦІНКА СИСТЕМИ ПІДТРИМКИ ЯКОСТІ ПРОГРАМНОГО КОДУ

4.1 Оцінка системи тестування якості програмного коду

Впровадження автоматизації певних видів тестування завжди має свою очевидність [1, 5, 6]. Наприклад, обґрунтованість автоматизації може легко визначатися у випадках постійно повторюваних ідентичних тестів, а також у завданнях, пов'язаних із випробуванням навантаження, швидкодії системи та інших ситуаціях, де складно або неможливо забезпечити необхідні умови для ручного тестування. Проте, ситуація може викликати сумніви, коли мова йде про автоматизацію функціонального тестування.

Особливо гостро стоїть питання вибору тестів для автоматизації, коли немає можливості автоматизувати всю бібліотеку тестів: які випробування автоматизувати, який ступінь автоматизації доцільно використовувати (тобто варто домагатися повної відсутності втручання людини), який інструментарій автоматизації тестування вибрати.

Головними цілями впровадження автоматизованого тестування є підвищення якості процесу тестування та зменшення витрат часу, необхідного для проведення тестів. Під підвищенням якості розуміється можливість охоплення ширшого спектру тестів або тестування, яке неможливо виконати вручну. Крім того, автоматизація тестів дозволяє мінімізувати вплив людського фактору на результати тестування. Під економією часу розуміється той факт, що виконання автоматизованих тестів зазвичай вимагає значно менше часу, ніж аналогічне тестування вручну.

До переваг автоматизованого тестування можна віднести:

1. Скорочення часу на виконання тестів порівняно з ручним тестуванням;

2. Можливість проведення тестів, які не можна провести без використання засобів автоматизації;

3. Підвищення незалежності експертизи (виключається людський фактор);

4. Можливість проводити тестування поза робочими годинами тестувальника.

Необхідно також наголосити і на недоліках впровадження автоматизації тестування [3]:

1. Автоматизація тестування найчастіше вимагає значних трудовитрат;

2. Потрібен кваліфікований персонал;

3. Складний аналіз результатів;

4. Будь-яка зміна в роботі системи може вимагати трудомістких змін в автоматичних тестах;

5. Помилка реакції системи на один із тестів може призвести до помилкових результатів прогону наступних тестів;

6. Ризик виникнення помилок у самому автоматичному тесті;

7. У деяких випадках, не всі функціональні особливості системи можна покрити автоматизованими тестами за допомогою вибраного інструментарію.

4.2 Метрика впровадження автоматичного тестування

Точну економічну доцільність від автоматизації тестів не завжди можна прорахувати заздалегідь, тому що вона залежить від ряду факторів:

1. Запланованого життєвого циклу системи;

2. Вибраного методу розробки системи;

3. Об'єктів автоматизації;

4. Методи автоматизації.

Найпоширенішими є такі три способи підрахунку ефективності впровадження автоматичного тестування [2, 17, 22]:

1.Прямий розрахунок рентабельності інвестицій;

2.Розрахунок рентабельність інвестицій з погляду ефективності використання ресурсів;

3.Розрахунок рентабельності інвестицій з погляду мінімізації ризиків.

У кожного з цих методів є свої переваги та недоліки, тому слід використовувати відразу кілька методів для отримання найбільш зваженої та всебічної оцінки.

Пропонується використовувати наступну загальну формулу розрахунку рентабельності інвестицій (PPI):

$$PPI = \frac{(\text{Прибуток} - \text{Інвестиції})}{\text{Інвестиції}} * 100\% \quad (4.1)$$

У разі прямого розрахунку рентабельності інвестицій, під «Прибутком» (Gain) мається на увазі розрахункова вартість тестування без автоматизації, а під «Інвестиціями» (Investments) маються на увазі витрати на створення та виконання автоматичної бібліотеки тестів за той же період, що був використаний для обчислення розрахункової вартості тестування без автоматизації. Метод прямого розрахунку рентабельності інвестицій дозволяє провести розрахунок прямої вигоди щодо витрачених коштів.

При розрахунку витрат на автоматизацію тестування, необхідно враховувати такі фактори:

- вартість програмного забезпечення для автоматизації тестування;
- витрати на навчання персоналу чи залучення більш кваліфікованого персоналу;
- витрати на додаткові апаратні засоби (в окремих випадках);
- вартість розробки початкової бібліотеки автоматичних скриптів;
- вартість прогону бібліотеки автоматичних скриптів;
- витрати на аналіз результатів прогону автоматичних тестів;
- витрати підтримку автоматизованих скриптів у стані.

Наступна формула може бути використана для розрахунку необхідних інвестицій за виділений період (*In*), якщо передбачається, автоматизуватиметься існуюча бібліотека ручних скриптів та що автоматизацією тестування займатимуться ті самі працівники, що займалися ручним тестуванням (тобто без зміни вартості оплати) праці):

$$I_p = I_0 + C_0 + \sum_{m=1}^{n} (C_e + C_a + C_m) \quad (4.2)$$

де I_0 - стартові інвестиції, які вираховуються як сума витрат на ліцензію програмного забезпечення для автоматизації тестування, навчання персоналу та вартість додаткових апаратних засобів;

C_0 - вартість розробки та налагодження початкової бібліотеки автоматичних скриптів, яка вираховується як добуток часу, який вимагається на написання одного автоматизованого скрипта одним тестувальником (у годинах), помножене на загальну кількість скриптів та на вартість робочої години;

k - кількість запланованих циклів тестування (тобто передбачувана кількість прогонів скриптів);

C_e - вартість одноразового виконання набору автоматизованих скриптів, яка обчислюється як середній час, витрачений на підготовку до виконання і безпосереднє виконання одного скрипта одним тестувальником (у годинах), помножене на загальну кількість скриптів і на вартість робочої години. Ця змінна може приймати значення, наприклад, коли мається на увазі повністю автономне виконання тестів, що не потребує втручання людини ні на стадії підготовки до виконання тесту, ні під час виконання тесту;

C_a - вартість аналізу результатів одного прогону набору автоматизованих скриптів, яка обчислюється як передбачуваний відсоток невдало проведених тестів, збільшений на загальну кількість скриптів, середній час, необхідний на аналіз причин невдалого прогону одного скрипта одним тестувальником (у годинах), та на вартість робочої години тестувальника;

C_m - вартість підтримки автоматизованих скриптів в актуальному стані, яка обчислюється як добуток очікуваного коефіцієнта змін скриптів для кожного циклу виконання, часу, витраченого одним тестувальником на зміну одного скрипта (у годинах), загальної кількості скриптів та вартості робочої години тестувальника.

Ця змінна може приймати значення нуль, наприклад, якщо в даній функціональній області не планується жодних наступних змін.

У випадках, коли для автоматизації скриптів залучаються кваліфікованіші (тобто більш високооплачувані) фахівці, необхідно внести такі поправки при використанні формули розрахунку необхідних інвестицій за виділений період (*In*), наведеної вище:

1. Під час обчислень початкових інвестицій вартість навчання персоналу прирівнюється до нуля (I_0);

2. Слід використовувати вартість робочої години тестувальників, які залучаються до автоматизації, а не тестувальників, що виконують ручні тести, при розрахунку вартості розробки початкової бібліотеки автоматичних скриптів (C_0), вартості одного прого на набору автоматизованих скриптів (C_e), вартості аналізу результатів одного прогону набору автоматизованих скриптів (C_a) та вартості підтримки автоматизованих скриптів у актуальному стані C_m).

Вартість тестування за той же період, але без автоматизації (Vp) може бути подана в наступному вигляді:

$$V_p = V_0 + \sum_{m=1}^{n}(V_e + V_a + V_m) \quad (4.3)$$

де V_0 – вартість розробки початкової бібліотеки ручних скриптів. Це значення може дорівнювати 0, якщо мається на увазі наявність існуючої бібліотеки скриптів;

n - кількість запланованих циклів тестування (тобто передбачувана кількість прогонів скриптів);

V_e - вартість одноразового виконання набору ручних скриптів, яка обчислюється як середній час, витрачений на підготовку до виконання та на безпосереднє виконання одного скрипта одним тестувальником (у годинах), помножене на загальну кількість скриптів та на вартість робочої години тестувальника;

V_a – вартість аналізу результатів одного прогону набору ручних скриптів, яка обчислюється як передбачуваний відсоток невдало проведених тестів, помножений на загальну кількість скриптів, середній час аналізу причин невдалого виконання одного скрипта (у годинах), та на вартість робочої години тестувальника. Ця величина може бути прирівняна до нуля у більшості випадків, тому що розглядаються ручні скрипти, що представляють детально описану інструкцію для тестувальника;

V_m - вартість підтримки ручних скриптів в актуальному стані, яка обчислюється як очікуваний коефіцієнт змін, середнього значення кількості змінених скриптів за годину, помножена на загальну кількість скриптів, середній час, необхідний зміна одного скрипта (у годинах) та на вартість робочої години. Ця змінна може приймати значення 0, наприклад, якщо в даній функціональній області не планується жодних наступних змін.

Розрахувати очікувану вигоду від запровадження автоматичного тестування можна також і з погляду ефективності використання ресурсів. Даний метод заснований на порівнянні тимчасових витрат, необхідних на впровадження, виконання, аналіз результатів та підтримка автоматичних тестів (Інвестиції) із витратами на ручні тести (Прибуток).

Варто зазначити, що метод враховує лише тимчасові витрати персоналу, не беручи до уваги їхній грошовий вираз, тому він може бути особливо корисним у випадках, коли комерційні деталі невідомі.

Пропонується використовувати наступну формулу для розрахунку тимчасових інвестицій у автоматизацію тестування за виділений період (Q*вп*):

$$QI_p = QI_0 + QC_0 + \sum_{m=1}^{n}(QC_e + QC_a + QC_m) \quad (4.4)$$

де Q_{i0} - стартові тимчасові інвестиції, які в даному випадку складаються з часу витраченого на пошук відповідного програмного забезпечення для автоматизації тестування та навчання персоналу. Даний параметр може набувати нульового значення, в деяких випадках (наприклад, якщо залучалися вже навчені

фахівці, і не стояло проблеми вибору програмного забезпечення для автоматизації тестування);

QC_0 – тимчасові витрати на розробку початкової бібліотеки автоматичних скриптів, яка вираховується як середній час, необхідний написання одного скрипта (обраховується у годинах), множиться на загальну кількість скриптів;

n - кількість запланованих циклів тестування (тобто передбачувана кількість прогонів скриптів);

QCe - час, витрачений тестувальником на підготовку до виконання і безпосередньо на виконання одного скрипта (у годинах), помножений на загальну кількість скриптів.

Ця змінна може приймати значення 0, наприклад, коли мається на увазі повністю автономне виконання тестів, що не вимагає втручання людини ні на стадії підготовки до виконання тесту, ані під час виконання тесту;

QC_a – тимчасові витрати на аналіз результатів одноразового виконання набору автоматизованих скриптів, які обчислюються як передбачуваний відсоток невдало проведених тестів, помножений на загальну кількість скриптів і на середній час, вимагає моє аналіз причин невдалого виконання одного скрипта (у годинах);

QC_m – тимчасові витрати на підтримку автоматизованих скриптів в актуальному стані, які обчислюються як очікуваний коефіцієнт змін скриптів для кожного циклу виконання, помножений на середній час, який потрібний на зміну одного скрипта одним тестувальником (у годинах), та на загальну кількість скриптів. Ця змінна може приймати значення нуль, наприклад, якщо в даній функціональній області не планується ніяких наступних змін.

Тимчасові витрати на тестування за той же період, але без автоматизації (QVp) можуть бути представлені в наступному вигляді:

$$VQ_p = VQ_0 + \sum_{m=1}^{n}(VQ_e + VQ_a + VQ_m) \quad (4.5)$$

де VQ_0 – час, витрачений на розробку початкової бібліотеки ручних скриптів (у годиннику). Дане значення може дорівнювати 0, якщо мається на увазі наявність існуючої бібліотеки скриптів;

n - кількість запланованих циклів тестування (тобто передбачувана кількість прогонів скриптів);

VQ_e - тимчасові витрати на одноразове виконання набору ручних скриптів, які обчислюються як середній час, витрачений на підготовку до виконання і безпосередньо на виконання одного скрипта (у годинах), помножений на загальну кількість скриптів;

VQ_a - тимчасові витрати на аналіз результатів виконання одного циклу набору ручних скриптів, які обчислюються як передбачуваний відсоток невдало проведених тестів, помножений на середній час, потрібний на аналіз причин невдалого виконання одного скрипта одним тестувальником (у годинах), і на загальну кількість скриптів. Дана змінна дорівнює 0 у більшості випадків, оскільки розглядаються ручні скрипти, що представляють детально описану інструкцію для тестувальника;

VQ_m – тимчасові витрати на підтримку ручних скриптів в актуальному стані, які обчислюється як очікуваний коефіцієнт змін, помножений на середній час, необхідний на зміну одного скрипта одним тестувальником (у годинах), і на загальну кількість скриптів. Ця змінна може приймати значення нуль, наприклад, якщо в даній функціональної області не планується жодних наступних змін.

Ще одним способом оцінки рентабельності інвестицій в автоматизоване тестування є оцінка рентабельності інвестицій з погляду мінімізації ризиків.

Цей метод передбачає порівняння коштів, витрачених на тестування (Інвестиції) з збитками, які можуть виникнути внаслідок помилки функціонування готової системи на етапі експлуатації (Прибуток). Варто зазначити, що часто складно точно оцінити можливі збитки, тому цей метод має на увазі точний аналіз можливих ризиків.

Крім того, в даному методі мається на увазі, що були протестовані всі аспекти функціонування системи.

Приймаючи одну й ту саму величину можливих збитків як для автоматичного, так і ручного тестування, можна підрахувати значення ROI для кожного з видів тестування, використовуючи як величину інвестицій в автоматизоване тестування та як величину інвестицій у ручне тестування відповідно.

На основі аналізу наведених методів підрахунку доцільності впровадження автоматизації можна сформувати такі ключові фактори, присутні в кожному з методів:

– витрати розробку початкової бібліотеки автоматичних тестів;

– витрати на підготовку до виконання та безпосередньо на виконання одного скрипт;

– витрати на аналіз результатів одного прогону набору автоматизованих скриптів;

– витрати на підтримку автоматизованих тестів у актуальному стані;

– кількість запланованих циклів тестування.

Витрати розробки початкової бібліотеки автоматичних скриптів можуть бути зменшені за рахунок встановлення точної структури тестів, процесів і принципів розробки автоматичних тестів, єдиних для всіх тестувальників, що займаються автоматизацією на проекті. Крім того, для підвищення якості та ефективності розробки автоматизованих тестів, можна застосовувати техніки, які використовуються в аналогічних цілях у програмуванні.

Витрати на підготовку до виконання і безпосередньо на виконання одного скрипту можуть бути зменшені за рахунок оптимізації коду автоматизованого тесту, а також впровадження процедур ефективної поведінки автоматизованого скрипту у разі несправностей та невдалих тестів. Останнє особливо актуальне у випадках, коли виконання тестів передбачено в автоматичному режимі, без постійного контролю з боку тестувальника. У таких випадках велика ймовірність того, що один невдало виконаний тест може призвести до зупинки всіх наступних, що буде виявлено тільки на етапі аналізу результатів і, отже, призведе до неефективного використання часу.

Витрати на аналіз результатів одного прогону набору автоматизованих тестів можуть бути оптимізовані за рахунок впровадження більш докладних звітів про виконання тесту.

Це може бути досягнуто за рахунок додавання додаткових деталей у звіти, що дозволяють більш точно відобразити помилку та умови її прояву. Крім того, витрати на аналіз результатів виконання скриптів можна скоротити за рахунок максимально можливої незалежності проведення тестів, яка проявляється у використанні різних тестових (контрольних) даних для кожного з тестів, розпаралелювання тестів (виключення варіантів, коли результат виконання одного може потенційно вплинути на результат іншого та і т.д.).

Витрати на підтримку автоматизованих тестів в актуальному стані залежать від методів і принципів, використаних на етапі розробки тестів. Детальний документування змін, коментарі, а також використання засобів контролю версій може зменшити витрати на підтримку тестів.

Кількість запланованих циклів тестування є одним із ключових факторів щодо доцільності впровадження автоматичного тестування: що більше повторень планується, тим більше виправданим є використання автоматизованого тестування.

Разом з тим, слід зауважити, що автоматизація тестування не може розглядатися як універсальний засіб боротьби за якість кінцевого продукту, тому що крім явних переваг (таких, як щодо підвищення незалежності експертизи за рахунок виключення людського фактора, наприклад), у автоматизації тестування є й низка недоліків. Автоматизація тестування не завжди необхідна, раціонально обґрунтована чи економічно виправдана. Тільки детальний аналіз може допомогти зробити усвідомлений вибір, а також виявити плюси та мінуси обраного підходу. Методи, що розглянуті вище, можуть допомогти позбавитися хибних очікувань і скласти більш точне уявлення про результати автоматизації.

4.3. Висновки до 4-го розділу

У розділі описано доцільність впровадження автоматизації деяких видів тестування яка не завжди є очевидною, але необхідність автоматизації може бути легко доведена, якщо мова йде про автоматизацію постійно повторюваних ідентичних тестів, а також навантажувальному тестуванні, тестуванні швидкодії системи або інших випадках, коли невиправдано складно чи неможливо забезпечити необхідні умови ручного тестування.

Разом з тим, коли мова йде про автоматизацію функціонального тестування, виникає певна невизначеність стосовно необхідності проведення цього процесу. Особливо актуальним стає питання вибору конкретних тестів для автоматизації, особливо у випадках, коли неможливо автоматизувати всю бібліотеку тестів. Ключовими аспектами є визначення, які саме випробування слід автоматизувати, встановлення оптимального рівня автоматизації (тобто, чи варто прямувати до повної автоматизації без участі людини) та вибір найоптимальнішого інструментарію для автоматизації тестування.

ВИСНОВКИ

Отже, в результати проведеного дослідження можна зробити такі висновки.

Якість є важливим аспектом будь-якої програмної системи і загалом програмного забезпечення незалежно від галузі чи технічної сфери. Більше того, забезпечення якості програмного забезпечення - це не одноразове завдання; починаючи з моменту впровадження і триваючи протягом усього терміну служби системи, розробники виконують технічне обслуговування для досягнення функціональних і нефункціональних цілей свого програмного забезпечення. Цей акцент на якості програмного забезпечення призвів до того, що організації витрачають велику частину своїх ресурсів на технічне обслуговування програмного забезпечення, а це є найбільш витратною фазою життєвого циклу розробки програмного продукту.

Ключовим аспектом підтримки програмного забезпечення є розуміння коду. Розуміння програми - це дії розробників, що читають вихідний код, для того щоб зрозуміти призначення коду або визначити вимоги, пов'язані з їхньою діяльністю з обслуговування.

У першому розділі здійснено аналіз існуючих методів та практик щодо оцінки якості програмних систем в цілому, та програмного коду зокрема. Виявлено, що тестування програмного забезпечення (Software Testing) - перевірка відповідності реальних і очікуваних результатів поведінки програми, що проводиться на кінцевому наборі тестів, обраному певним чином.

Мета тестування - перевірка відповідності програмного забезпечення висунутим вимогам, забезпечення впевненості в якості програмного забезпечення, пошук очевидних помилок у програмному забезпеченні, які мають бути виявлені до того, як їх виявлять користувачі програми.

Загалом тестування програмного забезпечення проводиться:

– для перевірки відповідності вимогам;

– для виявлення проблем на більш ранніх етапах розробки та запобігання підвищенню вартості продукту;

– виявлення варіантів використання, які не були передбачені під час розробки, а також погляд на продукт з боку користувача;

– підвищення лояльності до компанії та продукту, оскільки будь-який виявлений дефект негативно впливає на довіру користувачів.

Загалом у тестуванні немає чітких визначень, як у фізиці, математиці, які при перефразуванні стають абсолютно невірними. Тому важливо розуміти процеси та підходи.

У другому розділі визначено концептуальні та логічні моделі для системи тестування програмного забезпечення, а також алгоритм функціонування цієї системи, використовуючи метод підтримки якості програмного коду.

Метод підтримки якості програмного коду полягає в оцінці якості програмного продукту шляхом формування звіту із результатами автоматичного тестування. Ця оцінка враховує виявлення серйозних та другорядних помилок відповідно до встановлених вимог.

Для аналізу якості та помилок у програмному коді використовується аналізатор, що ґрунтується на роботі нейронних мереж – спеціального методу визначення функцій. Процес роботи з такою мережею передбачає її навчання, що включає підготовку великої кількості пар вхідних та відповідних їм правильних вихідних векторів у навчальній вибірці.

Елементи цієї множини повинні бути незалежними для можливості розширення вибірки та її подання в довільному порядку.

У третьому розділі здійснено проектування системи тестування якості програмного коду із визначенням вимог. Результатом стало проектування архітектури системи тестування, що включає аналізатор виявлення помилок, заснований на нейроно-мережевих моделях.

У четвертому розділі здійснено теоретичне обгрунтування доцільності впровадження автоматичного тестування. Виявлено, що впровадження автоматизованого тестування спрямоване на досягнення двох основних цілей: підвищення якості проведеного тестування і ефективна економія часу, витраченого на цей процес. Під «підвищенням якості» розуміється розширення

охоплення тестів, включаючи ті, які є непрактичними для виконання вручну, а також зменшення впливу людського фактора на результати тестування через автоматичне їх виконання.

Практичне значення отриманих результатів. У даній кваліфікаційній роботі здійснено спробу здійснити удосконалення методу підтримки якості програмного коду, шляхом використання автоматичного тестування. Результатом цієї кваліфікаційної роботи є алгоритмічне рішення роботи системи підтримки якості програмного коду. Результати даного дослідження можуть якісно вплинути на роботу розробників програмного забезпечення, шляхом використання методу підтримки якості програмного коду на основі системи автоматичного тестування. Доцільність та ефективність розробки системи теоретично обґрунтована у першому та другому розділах кваліфікаційної роботи опираючись, випливають із результатів емпіричних досліджень.

Результати даного дослідження та їх впровадження можуть поліпшити продуктивність програмістів, включаючи їх у творчий процес розробки. Це, в свою чергу, сприятиме створенню та підтримці якості вихідного програмного коду програмних систем чи програмного забезпечення загалом.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Коцовський В.М. Супровід програмних систем: Методичний посібник для студентів спеціальності «Інженерія програмного забезпечення» / В. М. Коцовський. – Ужгород: Видавництво УжНУ «Говерла», 2016. – 52 с.

2.Якість програмного забезпечення та тестування: базовий курс. Навчальний посібник / За ред. Крепич С.Я., Співак І.Я. – Тернопіль: ФОП Паляниця В.А., 2020. – 478 с.

3.Benjamin Floyd, Tyler Santander, and Westley Weimer. 2017. Decoding the Representation of Code in the Brain: An fMRI Study of Code Review and Expertise. In Proceedings of the International Conference on Software Engineering (ICSE). 175–186.

4.Білас О.Є. Якість програмного забезпечення та тестування. Навчальний посібник / О.Є. Білас – Львів: Видавництво Львівської політехніки, 2011. - 216 с.

5.Моргун І.А. Метод експертної оцінки якості програмного забезпечення / І.А. Моргун // Матеріали міжнародної науково-практичної конференції аспірантів і студентів «Інженерія програмного забезпечення», №2(6), 2011. С. 33-37.

6.Malets I., Prydatko O., Popovych V., Dominik A. Interactive Computer Simulators in Rescuer Training and Research of their Optimal Use Indicator. 2018 IEEE Second Conference on Data Stream Mining & Processing. Lviv, 2018. – №2 – 558-562.

7.ISO/IEC TR 9126-2:2003 Software engineering – Product quality – Part2: External metrics.

8.Thomas Fritz, Andrew Begel, Sebastian C. Müller, Serap Yigit-Elliott, and Manuela Züger. 2014. Using Psycho-physiological Measures to Assess Task Difficulty in Software Development. In Proceedings of the International Conference on Software Engineering (ICSE). 402–413.

9.Придатко О. В., Придатко В. В., Борзов Ю. О., Дзень В. Є. Інтеграція новаційного методу мобільного навчання в освітні проекти підготовки розробників програмного забезпечення. Вісник ЛДУБЖД, Львів: ЛДУ БЖД, 2018. – №18. –С.70-80. 14. Barraood S. O., Mohd H., Baharom F. A Comparison Study of

Software Testing Activities in Agile Methods. Knowledge Management International Conference (KMICe) Virtual Conference. Malaysia, 2021. pp. 130-137

10. Загальносистемні принципи та етапи створення програм. Життєвий цикл програмного виробу. [Електронний ресурс] – URL: http://lib.mdpu.org.ua/e-book/vstup/L7.htm.

11. Моделі життєвого циклу, принципи і методології розробки програмного забезпечення [Електронний ресурс] – URL: https://evergreens.com.ua/ua/articles/software-development-metodologies.html.

12. Асєєва А. В., Кулаковська І. В. Аналіз проблем вибору технології для розробки програмного забезпечення. Комп'ютерно-інтегровані технології: освіта, наука, виробництво. Луцьк, 2019. №37. С. 10 – 18, https://doi.org/10.36910/6775-2524-0560-2019-37-2.

13. Test Deliverables in Software Testing – Detailed Explanation [Electronic resource] - URL: https://www.softwaretestingmaterial.com/testdeliverables/.

14. Авраменко А.С. Тестування програмного забезпечення. Навчальний посібник / А. С. Авраменко, В. С. Авраменко, Г. В. Косенюк. – Черкаси: ЧНУ імені Богдана Хмельницького, 2017. - 284 с

15. Kristien Ooms, Lien Dupont, Lieselot Lapon, and Stanislav Popelka. 2015. Accuracy and precision of fixation locations recorded with the low-cost Eye Tribe tracker in different experimental setups. Journal of eye movement research 8, 1 (2015).

16. Barbareschi M., Barone S., Carbone R. et al. Scrum for safety: an agile methodology for safetycritical software systems. Software Qual J. 2022. №30, pp. 1067–1088 https://doi.org/10.1007/s11219-022-09593-2.

17. J. Hofmeister, J. Siegmund, and D. V. Holt. Shorter identifier names take longer to comprehend. In 2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER), pages 217–227, 2017.

18. Motivation and Efficiency of Quality Management Systems Implementation: A Study of Lithuanian Organizations [Електронний ресурс]. - Режим доступу:

https://www.researchgate.net/publication/264857878_Motivation_and_Efficiency_of_Quality_Management_Systems_Implementation_A_Study_of_Lithuanian_Organizations.

19. Ponzanelli, Simone Scalabrino, Gabriele Bavota, Andrea Mocci, Massimiliano Di Penta, Rocco Oliveto, and Michele Lanza. Supporting Software Developers with a Holistic Recommender System. In Proceedings of ICSE 2017 (39th ACM/IEEE International Conference on Software Engineering). to be published, 2017.

20. Sarah Fakhoury, Yuzhan Ma, Venera Arnaoudova, and Olusola Adesope. The effect of poor source code lexicon and readability on developers' cognitive load. In 2018 IEEE/ACM 26th International Conference on Program Comprehension (ICPC), pages 286–28610, 2018.

21. Студентський репозитарій. [Електронний ресурс] URL: https://studfile.net/preview/6305127/page:2. https://studfile.net/preview/6305131/.

22. Моргун І.А. Метод експертної оцінки якості програмного забезпечення / І.А. Моргун // Матеріали міжнародної науково-практичної конференції аспірантів і студентів «Інженерія програмного забезпечення», №2(6), 2011. С. 33-37.

23. R. B. Scharff, G. Fang, Y. Tian, J. Wu, J. M. P. Geraedts, and C. C. Wang, Sensing and reconstruction of 3-d deformation on pneumatic soft robots, IEEE/ASME Transactions on Mechatronics 26, 1877 (2021).

24. Sacha Lity, Manuel Nieke, Thomas Thüm, Ina Schaefer : Retest test selection for product-line regression testing of variants and versions of variants. Journal of Systems and Software, No.147, January 2019. Pp 46-63.

25. Математичне моделювання процесу розробки спеціалізованих програмних систем безпеко-орієнтованого спрямування /Ю. С. Кордунова, М. Фелтіновскі, О. В. Придатко, О. О. Смотр //Вісник ЛДУБЖД Bulletin of Lviv State University of Life Safety https://journal.ldubgd.edu.ua/index.php/Visnuk ISSN 2078-4643.

26. M. S. Xavier, A. J. Fleming, and Y. K. Yong, Finite element modeling of soft fluidic actuators: Overview and recent developments, Advanced Intelligent Systems 3, 2000187 (2021).

27. S. S. Muthu and M. A. Gardetti, Sustainability in the Textile and Apparel Industries (Springer, 2020).

28. T. George Thuruthel, F. Renda, and F. Iida, First-order dynamic modeling and control of soft robots, Frontiers in Robotics and AI 7, 95 (2020).

29. D. De Barrie, M. Pandya, H. Pandya, M. Hanheide, and K. Elgeneidy, A deep learning method for vision based force prediction of a soft fin ray gripper using simulation data, Frontiers in Robotics and AI 8, 104 (2021).

30. Björn Evers. Unit test generation for common and uncommon behaviors. master thesis, Delft University of Technology, 2020.

31. Fitsum Kifetew, Xavier Devroey, and Urko Rueda. Java unit testing tool competition: seventh round. In Proceedings of the 12th International Workshop on Search-Based Software Testing, pages 15–20. IEEE Press, 2019.

32. Jos Winter, Maurício Aniche, Jürgen Cito, and Arie van Deursen. Monitoring-aware IDEs. In ESEC/FSE'19, pages 420–431. ACM, 2019.

33. Qatestlab Training Center [Електронний ресурс] URL: https://training.qatestlab.com/blog/course-materials/glossary-testing-lifecycle/ .

34. Моделювання систем : конспект лекцій / В. М. Задачин, І. Г. Конюшенко. – Харків : Вид. ХНЕУ, 2010. – 268 с. http://surl.li/exlpa

35. Антонюк А. О. Опорний конспект з курсу Моделювання систем для спеціальності Інтелектуальні системи прийняття рішень, НУДПСУ. – Ірпінь, 2009.

36. Найповніший посібник із керування вимогами та відстеження [Електронний ресурс] – URL: https://visuresolutions.com/uk.

ДОДАТОК А

(обов’язковий)

КОПІЇ НАУКОВИХ ПУБЛІКАЦІЙ

УДК 004.9

Новацький О.В., Сороколіт В.О., Яшина О.М.

Хмельницький національний університет

МЕТОД ПІДТРИМКИ ЯКОСТІ ПРОГРАМНОГО КОДУ НА ОСНОВІ АВТОМАТИЧНОГО ТЕСТУВАННЯ

Розглянуто задачу підтримки якості програмного коду на основі автоматичного тестування, що дозволяють здійснювати правильне оформлення програмного коду згідно правил та стандартів. Здійснено аналіз доцільності підтримки коду програмного забезпечення в сучасних реаліях роботи розробника.

The task of maintaining the quality of the software code based on automatic testing, which allows for the correct design of the software code according to rules and standards, is considered. An analysis of the expediency of software code support in the modern realities of the developer's work was carried out.

Тестування є найстарішим і найсуворішим типом підтримки якості програмного забезпечення. Загалом, тестування є важливим процесом у багатьох галузях техніки: будь то новий телевізор, програмне забезпечення системи автомобіля тощо, тестування може допомогти інженерам виявити дефекти та переконатися в тому, що продукт міцний та надійний в різних умовах використання.

Тестування програмного забезпечення має ту ж мету, що й тестування, яке застосовується в інших галузях техніки, а саме виявлення дефектів або недоліків товару. Тестування можна умовно розділити на дві категорії: ручне тестування та автоматизоване тестування. У ручному тестуванні виконуються тестові випадки вручну людиною без будь-якої підтримки інструментів чи сценаріїв, тобто тестувальник вручну перевіряє виріб на виявлення дефектів і складає звіт після цього; в автоматизованому тестуванні, тестові випадки виконуються за допомогою інструментів, сценаріїв і програмного забезпечення.

Для виконання тестових операцій вручну потрібен тестувальник програмного забезпечення: це дуже важливо, оскільки незалежно від того, наскільки воно покращується, автоматизація не може замінити людську інтуїцію, умовиводи та індуктивне міркування. Наприклад, людина може змінити курс посеред тестового запуску, щоб перевірити щось, чого не було розглянуто раніше. Крім того, автоматизоване тестування не може перевірити деякі аспекти коду, наприклад зручність для користувача або взаємодію з клієнтом. З іншого боку, є й такі великі недоліки: деякі типи тестування (наприклад, тестування навантаження та продуктивності) не можуть бути виконується вручну, і, що найголовніше, ручне тестування займає багато часу. Ось чому компанії покладаються (здебільшого) на автоматизовані тести. У цій практиці тестувальники розробляють тестові сценарії для автоматичної перевірки продукту. Іншими словами, вони не тестують «вручну» програмні продукти, але покладаються на попередньо розроблені тести, які

виконуватимуться автоматично. Враховуючи це, виконання тестів і підрахунок їх успішності автоматизоване тестування є швидшим і надійнішим, ніж ручне тестування.

Особливістю автоматичного тестування програмного забезпечення є те, що як продукт, так і тести зроблені з того самого «матеріалу» і розроблені таким же чином: продукт під тестування - це код (називається робочим кодом), і він перевіряється за допомогою іншого коду (називається тестовим кодом). Як можна собі уявити, тестування програмного забезпечення передбачає тестування різних аспектів робочого коду: у нас є нефункціональне тестування (спрямоване на тестування нефункціональних аспектів програмного забезпечення систем, таких як продуктивність, зручність використання, надійність, безпека), а також функціональне тестування перевіряє, чи кожна функція програми працює відповідно до специфікації вимог. Функціональні тести - це переважна більшість тестів, написаних розробниками. У цій галузі тестування можна знайти модульне тестування (тести, які перевіряють один блок програмного забезпечення), інтеграційне тестування (тести, які інтегруються у різні підрозділи та їх взаємодія), системне тестування, регресійне тестування тощо. Існують буквально сотні різних типів тестування, кожен окремо з них вимірює різні аспекти виробничого коду.

Подібно до інженерії в інших дисциплінах, використовується тестування програмних систем, щоб виявити помилки на ранніх етапах циклу розробки (наприклад, до того, як код буде створено, здійснюється відправлення клієнтам). Без належного тестування програмного забезпечення помилки можуть дійти до клієнтів, що може бути дорогим для виправлення або навіть небезпечним для життя та здоров'я.

Отже, автоматичне тестування стало важливим процесом для покращення якості програмного забезпечення системи. Автоматизовані тести можуть допомогти переконатися, що виробничий код є надійним у багатьох умовах використання та що код відповідає вимогам продуктивності та безпеки.

Перелік посилань

1. Коцовський В.М. Супровід програмних систем: Методичний посібник для студентів спеціальності «Інженерія програмного забезпечення» / В. М. Коцовський. – Ужгород: Видавництво УжНУ «Говерла», 2016. – 52 с.
2. Robert Brautigam. Why I Never Null-Check Parameters. URL - https://dzone.com/articles/why-i-never-null-check-parameters, 2018.
3. Benjamin Floyd, Tyler Santander, and Westley Weimer. 2017. Decoding the Representation of Code in the Brain: An fMRI Study of Code Review and Expertise. In Proceedings of the International Conference on Software Engineering (ICSE). 175– 186.
4. Загальносистемні принципи та етапи створення програм. Життєвий цикл програмного виробу. [Електронний ресурс] – URL: http://lib.mdpu.org.ua/e-book/vstup/L7.htm.
5. Simon Butler, Michel Wermelinger, Yijun Yu, and Helen Sharp. 2009. Relating Identifier Naming Flaws and Code Quality: An Empirical Study. In Proceedings of the Working Conference on Reverse Engineering (WCRE). 31–35.
6. ISO/IEC 9126-1:2001. Software engineering – Software product quality – Part 1: Quality model.

ДОДАТОК Б

(обов’язковий)

ПРЕЗЕНТАЦІЙНІ МАТЕРІАЛИ

Метод підтримки якості програмного коду на основі автоматичного тестування

Автор роботи:
студент групи ІПЗм-22-1 Новацький О.
Керівник роботи:
к.т.н., доцент Яшина О.М.

Актуальність

Підтримка якості програмного коду відіграє важливу роль під час розробки програмного забезпечення. Під якістю програмного коду розуміються його характерні ознаки та властивості. Вони можуть відрізнятися залежно від конкретних бізнес-завдань організації та потреб команди. І хоча немає точного переліку контрольних показників, проте існує кілька загальних критеріїв, що відрізняють якісний програмний код від неякісного. Для підвищення та підтримки якості програмного коду використовують тестування. Загалом тестування є важливим процесом у багатьох галузях техніки. Тестування програмного забезпечення має за мету виявлення дефектів або недоліків та поділяється на: ручне тестування та автоматизоване тестування. У ручному тестуванні тестові випадки виконуються вручну людиною без будь-якої підтримки інструментів чи сценаріїв, в автоматизованому тестування, тестові випадки виконуються за допомогою інструментів, сценаріїв і програмного забезпечення.

Таким чином, автоматичне тестування допомагає у підтримці якості програмного коду, та розробленого програмного забезпечення в цілому.

Провівши грунтовний аналіз останніх досліджень та публікацій було виявлено та вивчено підходи таких вітчизняних та зарубіжних науковців:

- Лавріщєва К.М., яка займалася питаннями програмної інженерії та якістю програмного забезпечення зокрема;
- Коцовський В.М. розглядав супровід програмних систем в контексті якості.
- Лю та інші пропонують автоматизований підхід, заснований на методі глибокого навчання для налагодження назви на основі узгодженості між назвою методу та його реалізацією.

Мета: дослідження є удосконалення методу підтримки якості програмного коду на основі автоматичного тестування.

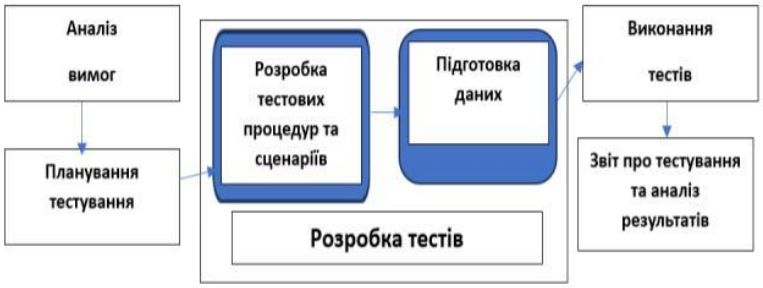
Об'єкт дослідження: процес створення системи підтримки якості програмного коду засобами автоматичного тестування.

Предмет дослідження: якість програмного коду та його підтримка засобами автоматичного тестування.

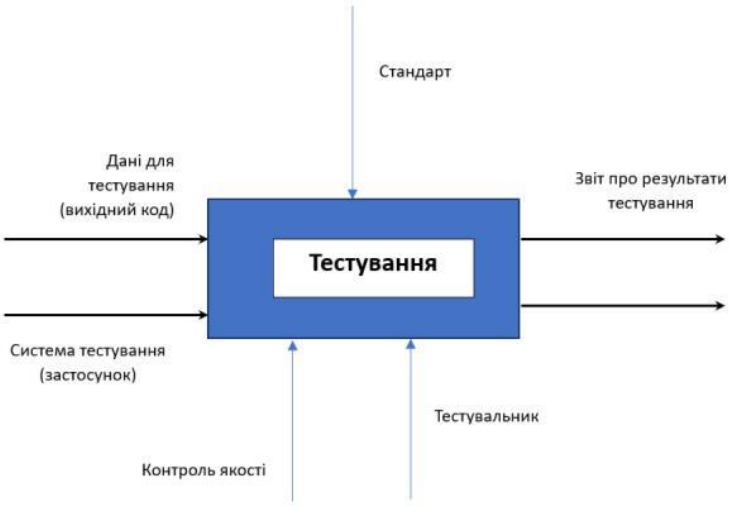
Задачі дослідження

1. Провести аналіз предметної області.
2. Здійснити аналіз існуючих методів.
3. Провести удосконалення методу підтримки якості програмного коду на основі автоматичного тестування.
4. Удосконалити алгоритм роботи системи підтримки якості програмного коду на основі автоматичного тестування.

Логічна модель

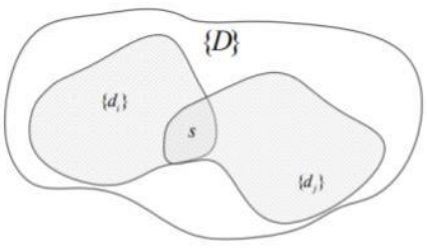


Концептуальна модель



Метод підтримки якості програмного коду на основі автоматичного тестування

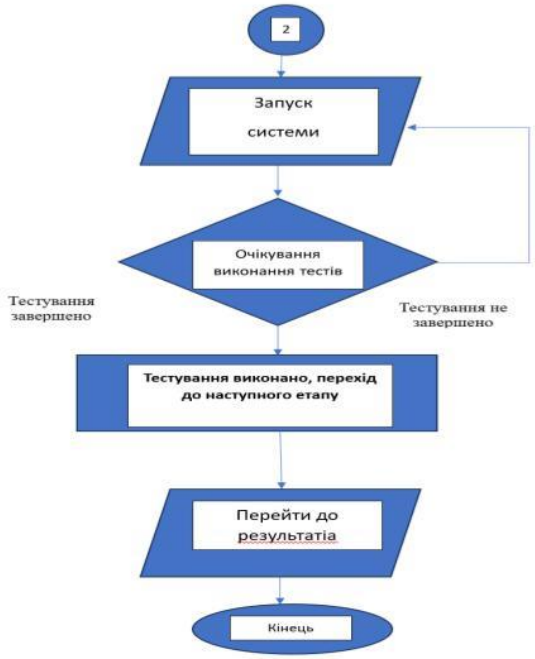
Метод підтримки якості програмного коду полягає у тому, що на основі формування звіту із результатами автоматичного тестування здійснюється оцінка якості програмного продукту, шляхом виявлення серйозних та другорядних помилок в залежності від поставлених вимог.



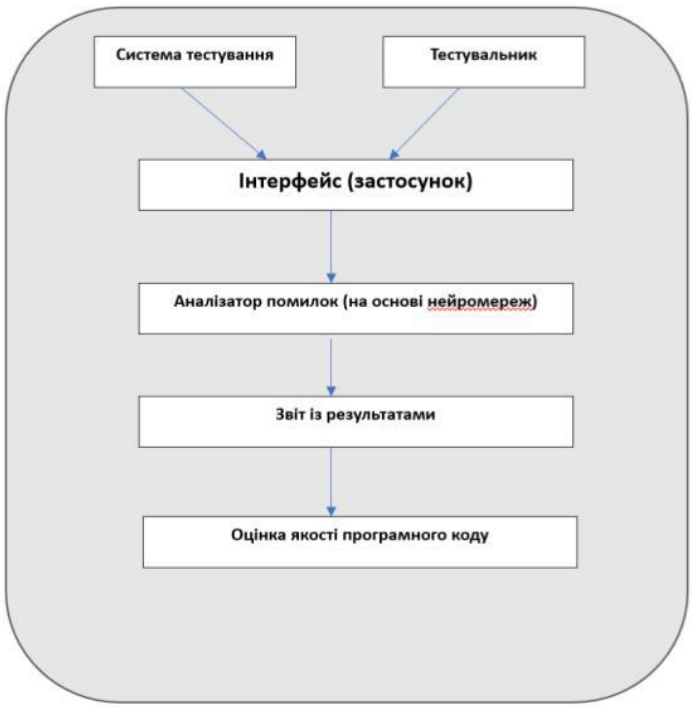
Алгоритм вибору необхідного методу тестування

Алгоритм вибору необхідного методу тестування

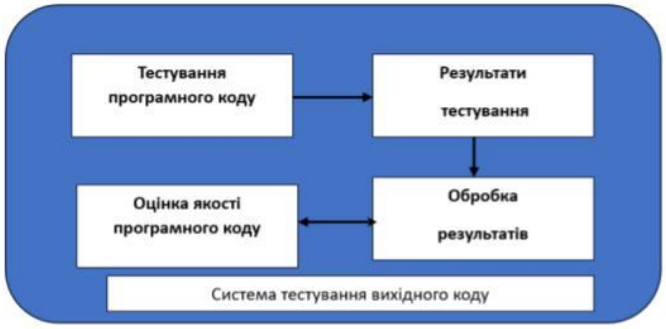
Алгоритм вибору необхідного методу тестування



Архітектура програмного забезпечення



Результати роботи



Наукова новизна

1. Удосконалено метод методу підтримки якості програмного коду на основі автоматичного тестування.
2. Удосконалено алгоритм роботи системи підтримки якості програмного коду на основі автоматичного тестування.

Практична значимість

Практична значимість отриманих результатів полягає у тому, що отримані результати кваліфікаційної роботи можуть:

- допомогти розробникам створювати якісний програмний код та тестувати його в автоматичному режимі із використанням удосконаленого методу;
- бути використаними для підвищення якості програмного коду в цілому.

Публікації за темою роботи

За темою кваліфікаційної роботи опубліковано тези у науково-практичній конференції Хмельницького національного університету «Актуальні проблеми комп'ютерних наук АПКН-2023»:

Новацький О.В., Сороколіт В.О., Яшина О.М. «Метод підтримки якості програмного коду на основі автоматичного тестування» // - Актуальні проблеми комп'ютерних наук АПКН-2023. - 2023 - с.225-226

Висновки

1. Здійснено якісний аналіз предметної області.
2. Проаналізовано існуючі методи підтримки якості програмного коду.
3. Проведено удосконалення методу підтримки якості програмного коду на основі автоматичного тестування.
4. Удосконалено алгоритм роботи системи підтримки якості програмного коду на основі автоматичного тестування.

Дякую за увагу!

Завідувачу кафедри інженерії програмного
забезпечення проф. Леоніду БЕДРАТЮКУ
здобувача вищої освіти
Новацького Олега Валерійовича
факультет ІТ, 2 курс, група ІПЗм-22-1

ЗАЯВА

З правилами чинного Положення про систему забезпечення академічної доброчесності в Хмельницькому національному університеті, згідно з яким виявлення плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту і застосування заходів дисциплінарної та академічної відповідальності, ознайомлений. Про використання програмно-технічних засобів для перевірки кваліфікаційних робіт здобувачів вищої освіти на плагіат оповіщений та надаю свою згоду на обробку й збереження університетом моєї роботи в інституційному репозитарії університету.

Також надаю університету право на передачу моєї роботи для обробки та збереження в базах даних програмно-технічних засобів (Unicheck та Anti-Plagiarism) та використання роботи для виявлення плагіату в інших роботах, які перевіряються програмно-технічними засобами та користувачами, що мають доступ до цих програмно-технічних засобів, виключно в обмежених цілях для виявлення плагіату в текстах робіт.

Робота для перевірки університетом надається в друкованому та електронному варіанті. Електронна версія моєї роботи збігається (ідентична) з друкованою.

07.12.2023
дата

підпис

Fri Dec 08 16:29:57 EET 2023, Форкун Юрій Вікторович, Хмельницький національний університет, ХНУ

Anti-Plagiarism v-15.257

Максимальне співпадіння з одним документом 13.0%

Словники перевірки: en_US, ru_RU, ua_UA. **Помилок в документах: 5%**

ID: 122215 Назва: Метод підтримки якості програмного коду на основі автоматичного тестування Додано в БД: 2023-12-08 Автора: Новацький Олег Керівники: Яшина Оксана Консультанти: Опоненти:	Документ		Сумарний збіг по Базі Даних	
	Символи	Лексеми	Символи	Лексеми
	95420	1400	13073 (14%)	191 (14%)

Джерело плагіату

ID	Опис	Наявність плагіату в документі	
		Символи	Лексеми
119414	Назва: Звіт з науково-дослідної практики Додано в БД: 2023-10-19 Автора: Новацький Олег Керівники: Яшина О.М. Консультанти: Опоненти:	11994 (13.0%)	184 (13.0%)

UNICHECK
by Turnitin

Ім'я користувача:
ІПЗ

ID перевірки:
1015985194

Дата перевірки:
08.12.2023 17:26:16 EET

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
08.12.2023 17:31:51 EET

ID користувача:
100012953

Назва документа: ДР_Новацький_плагіат

Кількість сторінок: 76 **Кількість слів:** 13699 **Кількість символів:** 112101 **Розмір файлу:** 1.34 MB **ID файлу:** 1015665944

10%
Схожість

Найбільша схожість: 2.74% з Інтернет-джерелом (https://core.ac.uk/download/pdf/323526244.pdf)

9.97% Джерела з Інтернету	499	Сторінка 78
1.04% Джерела з Бібліотеки	110	Сторінка 80

0% Цитат

Не знайдено жодних цитат

Не знайдено жодних посилань

0%
Вилучень

Немає вилучених джерел

Модифікації

Виявлено модифікації тексту. Детальна інформація доступна в онлайн-звіті.

Замінені символи	7

РІШЕННЯ ЕКСПЕРТНОЇ КОМІСІЇ
КАФЕДРИ ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ
ПРО ДОПУСК КВАЛІФІКАЦІЙНОЇ РОБОТИ ДО ЗАХИСТУ

Підтверджуємо ознайомлення з результатами звіту/звітів подібності щодо роботи, продукованими програмно-технічним засобом(ами) перевірки текстів на плагіат.
Назва: «Метод підтримки якості програмного коду на основі автоматичного тестування»
Автор: Новацький Олег Валерійович
Спеціальність: 121 – Інженерія програмного забезпечення
Освітня програма: Освітньо-професійна програма «Інженерія програмного забезпечення»
Науковий керівник: Яшина Оксана Миколаївна, кандидат технічних наук, доцент
Після аналізу звіту подібності зроблено такий висновок:

№	Висновок	Позначка про відповідність
1	Запозичення, виявлені в роботі, є законними і не є плагіатом. Робота приймається до захисту.	**відповідає**
2	Виявлені запозичення не є плагіатом, розміщені у розділах, які не описують безпосередньо авторське дослідження, але кількість цитат перевищує обсяг, виправданий поставленою метою роботи. Робота приймається до захисту, але має бути відкоригована. Відкоригований варіант має бути поданий на кафедру за два дні до захисту, разом із заявою щодо самостійності виконання письмової роботи та ідентичності друкованої й електронної версії роботи.	
3	Виявлені запозичення не є плагіатом, але частково розміщені у розділах, які описують безпосередньо авторське дослідження, а кількість цитат перевищує обсяг, виправданий поставленою метою роботи. В зв'язку з цим мета роботи та поставлені завдання не були досягнені. Робота може бути допущена до захисту (наступного року) після того, як буде відкоригована та доопрацьована і успішно пройде повторну перевірку на академічний плагіат.	
4	Робота містить навмисні текстові спотворення, передбачувані спроби укриття запозичень або інші прояви академічного плагіату. Робота містить фабрикацію або фальсифікацію даних. Робота не допускається до захисту.	
5	Інше:	

Підтвердження:
Запозичення, виявлені у роботі, є законними і не є плагіатом, оскільки:
1) у тексті кваліфікаційної роботи системою перевірки на плагіат Unicheck виявлено схожість з деякими документами у частині загальновживаних обов'язкових словосполучень у стандартних бланках (титулка, бланк завдання), у структурі змісту, у назвах розділів/підрозділів, у назвах публікацій переліку джерел посилання тощо;
2) запозичення, виявлені в тексті роботи, є фрагментарними або мають належним чином оформленні посилання;
3) виявлені модифікації тексту не впливають на відсоток схожості.
Максимальний обсяг запозичень, визначений системою Anti-Plagiarism, складає 13.0%. Обсяг запозичень, визначений системою Unicheck виявлення збігів ідентичності/схожості, складає 10% і адресується до 499 джерел з інтернету і 110 джерела з бібліотеки, що, з урахуванням наведених обгрунтувань, відповідає характеру теми і свідчить на користь кваліфікаційної роботи.

Дата 08.12.2023

Завідувач кафедри ІПЗ — Леонід БЕДРАТЮК
Гарант освітньої програми — Оксана ЯШИНА
Керівник кваліфікаційної роботи — Оксана ЯШИНА

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

РЕЦЕНЗІЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

Здобувач Новацький Олег Валерійович
Тема Метод підтримки якості програмного коду на основі автоматичного тестування

Спеціальність 121 «Інженерія програмного забезпечення»

Обсяг кваліфікаційної роботи:

Кількість сторінок кваліфікаційної роботи 94 .

1. Короткий зміст роботи та прийнятих рішень: у кваліфікаційній роботі здійснено системний аналіз предметної області у сфері підтримки якості коду програмних систем. На основі проведеного аналізу удосконалено метод підтримки якості програмного коду на основі автоматичного тестування. Реалізація удосконаленого методу покращить продуктивність програмістів та якість програмного забезпечення в цілому. Цей метод знизить витрати часу на розробку тестових випадків за допомогою нейронних мереж, що також знизить ризики людського фактору під час цієї роботи, а також витрати на цю роботу.

2. Висновок про відповідність роботи поставленому завданню: Кваліфікаційна робота освітнього рівня «магістр» відповідає всім пунктам поставленого завдання.

3. Характеристика виконання кожного розділу роботи, ступінь використання останніх досягнень науки і техніки і передових методів роботи: У вступі викладено тему кваліфікаційної роботи, визначається мета та завдання дослідження. Обгрунтовано актуальність цієї кваліфікаційної роботи, описується наукова новизна та практична цінність отриманих результатів. У першому розділі охарактеризовано предметну області та її актуальність. Також викладено інформацію про існуючі методи і засоби підтримки якості програмного коду, виконана розгорнута постановка задачі та можливі шляхи її вирішення. У другому розділі визначено логічну та концептуальну моделі тестування, алгоритм роботи системи тестування програмного забезпечення із використанням методу підтримки якості програмного коду з використанням нейронних мереж. У третьому розділі описано метод підтримки якості програмного коду, подано алгоритм роботи системи підтримки якості програмного коду, архітектура якої передбачає використання аналізатора, що заснований на нейронних мережах. У четвертому розділі описано доцільність впровадження методу автоматичного тестування. Використання нейронних мереж набуло трендів у світі інформаційних технологій, що сприяє їх розвитку. Це дозволяє ширше використовувати їхні можливості, в тому числі і в тестуванні програмних систем.

4. Позитивні сторони роботи: Кваліфікаційна робота доводить доцільність удосконалення методу підтримки якості програмного коду на основі автоматичного тестування. Запропоновано використовувати нейроні мережі, що мають значно більші ресурси та можливості аніж люди. Цей метод матиме позитивний вплив на продуктивність роботи розробників програмного забезпечення та його якість загалом.

5. Негативні сторони роботи: у роботі описано доцільність та структуру системи, що реалізовуватиме метод підтримки якості програмного коду, проте недостатньо інформації щодо інтеграції у проекти.

6. Оцінка графічного оформлення та пояснювальної записки роботи: Графічне оформлення виконане відповідно до теми кваліфікаційної роботи з дотриманням вимог стандартів. Пояснювальна записка відповідає вимогам стандартів до її оформлення.

7. Відгук про роботу в цілому: Кваліфікаційна робота заслуговує позитивної оцінки. Весь матеріал роботи структурований, чіткий та послідовний. Усі розділи роботи є послідовними та логічними, що дозволяє чітко розуміти викладений матеріал у рамках тематики кваліфікаційної роботи.

8. Інші зауваження:

9. Оцінка кваліфікаційної роботи: Розглянувши всі сторони представленої кваліфікаційної роботи, можна зробити висновок, що вона заслуговує оцінки «добре».

РЕЦЕНЗЕНТ (прізвище, ім'я, по-батькові, посада, місце роботи) Клець Юрій Павлович, к.т.н., доцент, зав. кафедри кібербезпеки, ХНУ

«8» 12 2023 р.

(підпис)