Хмельницький національний університет

Факультет інформаційних технологій

Кафедра інженерії програмного забезпечення

КВАЛІФІКАЦІЙНА РОБОТА

Удосконалений метод роботи з метриками тестового покриття коду для забезпечення ефективного оцінювання результатів тестування 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»

Шифр КвРІПЗ.180110.01.06.ПЗ

Виконав студент 2 курсу, група ІПЗм-22-1 _____ Підпис Павло ОЛІЙНИК Ім'я, ПРІЗВИЩЕ

Керівник д-р фіз.-мат. наук, професор Науковий ступінь, звання _____ Підпис Леонід БЕДРАТЮК Ім'я, ПРІЗВИЩЕ

Нормоконтролер канд. техн. наук, доцент. _____ Підпис Галина РАДЕЛЬЧУК Ім'я, ПРІЗВИЩЕ

До захисту допускаю:
Завідувач кафедри
інженерії програмного забезпечення _____ Підпис Леонід БЕДРАТЮК Ім'я, ПРІЗВИЩЕ

7 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 ІПЗ

Л. П. Бедратюк

01.09.2023 р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

Олійнику Павлу Анатолійовичу

Прізвище, ім'я, по батькові здобувача

1. Тема роботи Удосконалений метод роботи з метриками тестового покриття коду для забезпечення ефективного оцінювання результатів тестування програмного забезпечення

Керівник роботи Бедратюк Леонід Петрович, д-р фіз.-мат. наук, професор

Прізвище, ім'я, по батькові, науковий ступінь, вчене звання

Затверджена наказом ректора університету від 15.08.2023 р. № 30

2. Строк подання студентом роботи на кафедру 01.12.2023 р.

3. Вихідні дані до роботи Матеріали науково-дослідної практики

4. Зміст пояснювальної записки (перелік питань, які потрібно розробити)

1. Дослідження предметної області та постановка задачі

2. Аналіз сучасних технологій та утиліт

3. Розробка програмної системи

4. Оцінка результатів та підведення підсумків

5. Перелік графічного матеріалу (із зазначенням обов'язкових креслень)

Презентаційні матеріали (слайди)

6. Консультанти розділів кваліфікаційної роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Антиплагіат	Форкун Ю. В., доцент	23.11.2023р	01.12.2023р
Нормоконтроль	Радельчук Г. І, доцент	23.11.2023р	01.12.2023р

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

КАЛЕНДАРНИЙ ПЛАН

Назва етапів (розділів) кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1 Вивчення предметної області; формулювання мети та задач дослідження; визначення об'єкта та предмета дослідження; формування логістичної структури кваліфікаційної роботи	01.09 –09.09.2023	
2 Робота над розділом 1 кваліфікаційної роботи – вивчення літературних та Інтернет-джерел; аналіз відомих моделей, методів та засобів за темою роботи; визначення методологічних підходів до вирішення задачі; висновки до розділу та постановка задач дослідження	10.09 –20.09.2023	
3 Робота над розділом 2 кваліфікаційної роботи – загальний огляд та аналіз сучасних технологій та утиліт, що використовуються при розробці програмної системи; висновки до розділу	20.09 –01.10.2023	
4 Робота над науковими статтями	01.10 –24.10.2023	
5 Робота над розділом 3 кваліфікаційної роботи – розробка програмної системи; проектування та реалізація інформаційної технології вирішення задачі; висновки до розділу	24.10 – 05.11.2023	
6 Робота над розділом 4 кваліфікаційної роботи – аналіз розробленого програмного рішення та виділення рекомендацій стосовно майбутніх покращень; висновки до розділу	05.11 – 15.11.2023	
7 Попередній захист кваліфікаційної роботи	17.11.2023	
8 Узгодження постановки задачі, отриманих результатів та висновків; написання вступу, загальних висновків, оформлення джерел посилання та додатків; оформлення пояснювальної записки та графічних матеріалів згідно вимог чинних стандартів	18.11 – 30.11.2023	
9 Перевірка роботи на наявність плагіату; нормоконтроль; брошурування пояснювальної записки; підготовка супровідних документів	01.12 –04.12.2023	
10 Підготовка до захисту кваліфікаційної роботи	з 01.12.2023	

Студент _____ Павло Олійник
Підпис Ім'я, ПРІЗВИЩЕ

Керівник роботи _____ Леся Бедрак
Підпис Ім'я, ПРІЗВИЩЕ

РЕФЕРАТ

Тема кваліфікаційної роботи: «Удосконалений метод роботи з метриками тестового покриття коду для забезпечення ефективного оцінювання результатів тестування програмного забезпечення».

Автор роботи: Олійник Павло Анатолійович.

Керівник роботи: Бедратюк Леонід Петрович.

Пояснювальна записка: 121 с., 25 рис., 3 дод., 43 джерел.

ПРОГРАМНА СИСТЕМА, МЕТРИКИ ПОКРИТТЯ КОДУ, ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ, PHP, AWS, GitHub Actions, Docker, КОНТЕЙНЕРИ.

Об’єктом дослідження є метрики тестового покриття програмного коду.

Предмет дослідження – удосконалений метод для роботи з метриками тестового покриття коду для забезпечення ефективного оцінювання результатів програмного забезпечення.

Метою роботи є розробка удосконаленого методу для роботи з метриками покриття коду шляхом проектування та реалізації інноваційної програмної системи, яка автоматизовує та спрощує процес оцінювання метрик покриття коду на основі результатів тестування та використовує сучасні технології та підходи.

У роботі використані наступні методи дослідження та апаратура:

– спостереження, аналіз та синтез інформації, формалізація;

– засоби проектування та програмування;

– персональний комп’ютер.

Розроблено програмну систему для автоматизації оцінювання метрик покриття коду на основі результатів тестування програмного забезпечення. Забезпечуючи зручний і швидкий аналіз якості коду, вона позитивно впливатиме на процес розробки, що дозволить полегшити виявлення помилок та підвищити надійність програмних продуктів.

Розроблена система може знайти своє застосування в різних галузях програмування та розробки ПЗ, включаючи веброзробку, мобільні додатки, вбудоване програмне забезпечення та інші.

З точки зору соціально-економічної ефективності, впровадження системи призведе до зменшення часу, необхідного для тестування та аналізу коду, що в свою чергу призведе до економії ресурсів та підвищенні продуктивності розробників.

Розроблена система має велике значення для сучасної розробки програмного забезпечення, оскільки вона спрощує та поліпшує ключові етапи розробки, сприяючи створенню надійних та високоякісних програм.

Програмна система є підходом, який показує можливість інтеграції сучасних платформ та сервісів і тому вона може бути адаптована для будь-якої сучасної мови програмування та покращена у майбутньому.

Підпис

06.12.2023

Дата

ABSTRACT

Master's thesis: «An advanced method for working with code coverage metrics to ensure effective evaluation of software testing results».

Author: Oliinyk Pavlo.

Head of research: Bedratyk Leonid.

Master's thesis consists of: 121 p., 25 pc., 3 add., 43 src.

SOFTWARE, CODE COVERAGE METRICS, SOFTWARE TESTING, PHP, AWS, GitHub Actions, Docker, CONTAINERS.

The object of the research is the test coverage metric of the software code.

The subject of the study is an improved method for working with coverage test code metrics to ensure effective evaluation of software results.

The purpose of the work is to develop an improved method for working with code coverage metrics by designing and implementing an innovative software system that automates and simplifies the process of evaluating code coverage metrics based on test results and uses modern technologies and approaches.

The following research methods and equipment were used in the work:

– observation, analysis and synthesis of information, formalization;

– design and programming tools;

– personal computer.

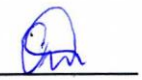
A software system was developed to automate the evaluation of code coverage metrics based on software testing results. By providing a convenient and quick code quality analysis, it will have a positive impact on the development process, which will facilitate the detection of errors and increase the reliability of software products.

The developed system can be used in different fields of programming and software development, including web development, mobile applications, embedded software etc.

From the point of view of socio-economic efficiency, the implementation of the system will lead to a reduction in the time required for testing and analysis of the code, which in turn will lead to saving resources and increasing the productivity of developers.

The developed system is of great importance to modern software development, as it simplifies and improves key development steps, contributing to the creation of reliable and high-quality applications.

The software system is an approach that shows the possibility of integrating modern platforms and services and therefore it can be adapted for any modern programming language and improved in the future.



Signature

06.12.2023

Date

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

AWS	–	Amazon Web Services
ECR	–	Elastic Container Registr
ECS	–	Elastic Container Service
EC2	–	Elastic Compute Cloud
EMR	–	Elastic MapReduce
PSR	–	Performance Summary Report
PHP	–	Hypertext Preprocesso
RDS	–	Relational Database Service
S3	–	Simple Storage Service
SMTP	–	Simple Mail Transfer Protocol
SSL	–	Secure Sockets Layer
TLS	–	Transport Layer Security
VPC	–	Virtual Private Cloud
YAML	–	Yet Another Markup Language

ВСТУП

Сучасний інформаційний світ вимагає від розробників програмного забезпечення не тільки створення функціональних і надійних додатків, але і постійного контролю за якістю коду. Забезпечення якості коду є важливою складовою розробки програмного забезпечення і відіграє ключову роль у забезпеченні надійності, безпеки та ефективності програмних продуктів. Одним з важливих інструментів для оцінки якості коду є метрики покриття коду.

Метрики покриття коду визначаються як спеціальні показники, які вказують на те, як багато рядків або частин коду були виконані під час тестування. Ці метрики допомагають виявляти недоліки в коді, які не можуть бути виявлені при звичайному тестуванні та вказують на те, які частини програми потребують більш ретельного та детального тестування.

Однак процес оцінки покриття коду може вимагати великої кількості ресурсі та часу, особливо у великих проектах. Саме тому автоматизація цього процесу стає дедалі більш важливою для розробників програмного забезпечення.

Актуальність роботи полягає в тому, що оцінка та аналіз якості покриття коду залишається популярною і важливою проблемою для розробників програмного забезпечення, тому існує потреба у створенні удосконаленого методу для роботи з метриками покриття коду, який забезпечить ефективну оцінку результатів тестування програмного забезпечення.

Метою роботи є розробка удосконаленого методу для роботи з метриками покриття коду шляхом проектування та реалізації інноваційної програмної системи, яка автоматизовує та спрощує процес оцінювання метрик покриття коду на основі результатів тестування та використовує сучасні технології та підходи.

Для досягнення поставленої мети слід вирішити ряд задач:

– провести аналіз предметної області та сформувати постановку задачі;

– проаналізувати існуючі програмні рішення, що використовуються для розрахунку метрик покриття коду та запропонувати удосконалений метод для роботи з метриками покриття коду;

– розробити детальну архітектуру запропонованої програмної системи;

– на основі плану архітектури провести розробку програмної системи;

– здійснити огляд отриманих результатів та впевнитись у правильності роботи програмної системи;

– запропонувати рекомендації для покращень розробленого програмного рішення у майбутньому.

Об’єктом дослідження є метрики тестового покриття програмного коду.

Предмет дослідження – удосконалений метод для роботи з метриками тестового покриття коду для забезпечення ефективного оцінювання результатів програмного забезпечення.

Вперше запропоновано удосконалений метод для роботи з метриками покриття коду, який полягає у забезпеченні повністю автоматизованого процесу оцінки метрик покриття коду та не потребує зовнішнього втручання команд розробників. Дана робота має важливий внесок у розвиток сучасних методів та інструментів для оцінки покриття коду та забезпечення ефективного оцінювання результатів тестування програмного забезпечення, що є актуальним завданням в інформаційній та комп'ютерній сфері.

Практична значимість отриманих результатів полягає в поліпшенні якості, надійності та ефективності розробки програмного забезпечення, що допомагає суттєво зекономити час та ресурси розробників, запобігти помилок та забезпечити більш швидкий та надійний процес виправлення помилок у програмному коді безпосередньо на стадії тестування.

Для досягнення поставленої мети були здійснені такі методи досліджень як детальний аналіз сучасної літератури і навчальних онлайн матеріалів стосовно основних положень розглянутих в кваліфікаційній роботі, а також огляд та вивчення існуючих програмних рішень для розрахунку метрик покриття коду.

1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Поняття тестування програмного забезпечення та метрик покриття коду

Тестування програмного забезпечення – це процес перевірки та оцінки програмного продукту з метою виявлення дефектів, помилок або невідповідностей вимогам та специфікаціям. Тестування відіграє важливу роль у забезпеченні якості програмного забезпечення та зниженні ризику виникнення проблем в роботі програми після її впровадження. Тестування програмного забезпечення також виконується, щоб перевірити чи виконується зазначена програмним забезпеченням бізнес-логіка та чи є прогалини у його вимогах, які потребують негайного вирішення зі сторони розробників[1,2,3].

Основні характеристики тестування програмного забезпечення включають:

– виявлення дефектів;

– перевірка відповідності вимогам;

– забезпечення надійності;

– валідація та верифікація;

– виявлення проблем з безпекою;

– оцінка продуктивності;

– підтримка якості.

Виявлення дефектів. Основна мета тестування полягає у знаходженні помилок та недоліків в програмному коді та його частинах.

Перевірка відповідності вимогам. Тестування допомагає впевнитися, що програмне забезпечення відповідає усім наведеним вимогам та специфікаціям визначеним для нього.

Забезпечення надійності. Тестування має на меті перевірити, наскільки надійно працює програма і виявити можливі проблеми, що можуть призвести до аварій або відмов.

Валідація та верифікація. Тестування дозволяє провести верифікацію правильності програмного коду та валідацію результатів його виконання.

Виявлення проблем з безпекою. Тестування може допомогти ідентифікувати потенційні проблеми з безпекою та захищеність програмного забезпечення від потенційних атак.

Оцінка продуктивності. Тестування може включати в себе оцінку продуктивності програми, вимірювання швидкості та завантаження системи під великим навантаженням.

Підтримка якості. Тестування сприяє забезпеченню високої якості програмного продукту та зменшенню ризику несправностей.

Тестування програмного забезпечення може виконуватися на різних етапах розробки, від ранніх етапів проекту до тестування в реальних умовах після випуску програми. Воно включає в себе створення тестових випадків, виконання тестів, аналіз результатів та виправлення виявлених дефектів[4].

Тестування програмного забезпечення є важливою частиною процесу розробки та забезпечення якості, яке допомагає покращити відповідність програмного продукту до вимог користувачів і зменшити ризик фінансових втрат та проблем у майбутньому[5,6].

Процес тестування програмного забезпечення може бути розділений на кілька основних етапів:

– планування тестування;

– створення тестових випадків;

– виконання тестів;

– збір результатів тестування;

– аналіз результатів і виправлення дефектів;

– завершення тестування та звітність;

– автоматизація тестування;

– тестування на різних етапах проекту.

Планування тестування. На початку проекту формується план тестування, в якому визначаються стратегія, методологія, обсяг та ресурси для проведення тестування. Визначаються також критерії завершення тестування.

Створення тестових випадків. На основі вимог та специфікацій розробляються тестові випадки, які включають в себе вхідні дані, дії та очікувані результати. Ці випадки можуть бути розроблені вручну або автоматизовано.

Виконання тестів. Тестові випадки виконуються відповідно до плану тестування. Це може включати в себе ручне виконання тестів або автоматизоване виконання за допомогою тестових скриптів.

Збір результатів тестування. Під час виконання тестів фіксуються результати, такі як пройдені та не пройдені тести, виявлені дефекти та інша додаткова інформація на основі результатів тестування.

Аналіз результатів і виправлення дефектів. Всі виявлені дефекти документуються, а розробники виправляють їх. Після виправлення дефектів тести можуть бути повторно виконані для перевірки.

Завершення тестування та звітність. Після завершення всіх тестів та виправлення дефектів генерується звіт про покриття коду, результати тестів і звіт про якість програми. Уся інформація ретельно документується для подальшого аналізу та аудиту.

Автоматизація тестування. Для покращення ефективності та швидкості тестування часто використовуються інструменти автоматизованого тестування. Це дозволяє автоматизовано виконувати тестові випадки та отримувати результати.

Тестування на різних етапах проекту. Тестування може виконуватися на різних етапах проекту, включаючи модульне тестування, інтеграційне тестування, системне тестування та приймальне тестування.

Загальна мета тестування програмного забезпечення полягає в тому, щоб переконатися, що програма відповідає вимогам, функціонує правильно, є надійною та безпечною для користувачів. Цей процес є невід'ємною частиною розробки програмного продукту та забезпечує високу якість та надійність програми перед її остаточним випуском[8,9,10].

В тестуванні програмного забезпечення виділяють два основних підходи такі як ручне та автоматичне тестування.

Ручне тестування — це підхід тестування у якому тестові випадки виконуються вручну без допомоги будь-яких автоматизованих інструментів. Це гарантує, що всі функції програми працюють, як визначено в документі вимог. Під час ручного тестування тестер перевіряє всі основні функції даної програми чи програмного забезпечення. У цьому процесі тестувальники програмного забезпечення власноруч виконують тестові випадки та генерують звіти про тестування без допомоги будь-яких сторонніх інструментів автоматизованого тестування програмного забезпечення. Оскільки для ручного тестування не використовуються програмні інструменти, воно зазвичай є повільним, дороговартісним та займає багато часу[12].

Даний підхід тестування використовується коли потрібне втручання людини та достовірне моделювання поведінки програми з точки зору користувача.

Ручне тестування є класичним метод для всіх типів тестування і допомагає якісно знаходити помилки в програмних системах. Зазвичай його проводить досвідчений тестер для завершення процесу тестування програмного забезпечення.

Виділяють наступні переваги ручного тестування:

– незалежність від технологій;

– гнучкість;

– тестування інтерфейсу користувача;

– рентабельність;

– швидкість.

Незалежність від технологій. Ручним тестувальникам не потрібні додаткові знання стороннього програмного забезпечення для тестування. Це робить ручне тестування зручним варіантом для команди тестувальників, що володіють меншими знаннями в програмуванні, оскільки їм не потрібно знати додаткові технології та інструменти тестування.

Гнучкість. Ручне тестування дозволяє тестувальникам змінювати тест-кейси та дії швидко відповідно до потреб.

Тестування інтерфейсу користувача. Можливість тестування графічного інтерфейсу з яким взаємодіють реальні користувачі, що є важливим аспектом для

тестування інтерактивних або вебдодатків, де потрібна пряма перевірка користувацького досвіду.

Рентабельність. Програмне забезпечення для автоматизованого тестування може бути дорогим, а також процес навчання команди по його користуванню. Ручне тестування ідеально підходить для команд з меншим бюджетом або з невеликим обсягом тестів.

Швидкість. Ручне тестування не вимагає етапу кодування для написання тестових сценаріїв, що суттєво пришвидшує виконання тестів.

Виділяють наступні недоліки ручного тестування:

– часозатратність;

– схильність до людських помилок;

– дороговартісні ресурси;

– непослідовність;

– нижче охоплення тестів.

Часозатратність. Ручне тестування не використовує жодних автоматизованих інструментів і залежить лише від зусиль людини. Це може зайняти багато часу. Ручний тестер не може обробляти великі обсяги даних так само ефективно, як автоматизоване програмне забезпечення.

Схильність до людських помилок. Ручні тестери більш схильні до помилок в порівнянні із програмними рішеннями.

Дороговартісні ресурси. Наймання та навчання ручних тестувальників може бути дорогим. Залежно від конкретної галузі існує потреба у тестерах з певним досвідом. Утримати високоякісних тестувальників на конкурентному ринку також може бути складно.

Непослідовність. Існує тенденція до отримання менш послідовних результатів в залежності від рівня концентрації тестерів. Будь-які мінімальні зміни в складі команди також можуть вплинути на продуктивність загальну якість виконання тестування.

Нижче охоплення тестів. Ручне тестування менш ідеальне для тестування системи великої складності та обсягу. Воно вимагає більших зусиль від команди

тестувальників. Це ускладнює досягнення максимального охоплення програмного забезпечення тестами.

Автоматизоване тестування – це процес використання інструментів автоматизованого тестування для контролю виконання тестів і порівняння результатів із очікуваними результатами, а також для пошуку дефектів. Тестувальники використовують відповідні засоби автоматизації для розробки тестових сценаріїв і перевірки програмного забезпечення. Мета полягає в тому, щоб завершити виконання тесту за менший проміжок часу.

Автоматизоване тестування повністю покладається на заздалегідь розроблений тест, що був написаний відповідною мовою програмування. Це допомагає тестувальнику визначити чи програма працює належним чином. Незважаючи на те, що всі процеси виконуються автоматично, автоматизація вимагає певних ручних зусиль для створення сценаріїв початкового тестування.

Переваги автоматизованого тестування:

– розширене охоплення тестування;

– швидкість та ефективність;

– надійність;

– повторюваність.

Розширене охоплення тестування. Автоматичне тестування дозволяє збільшити охоплення тестуванням. Воно краще обладнане для обробки великого обсягу тестових випадків, що допоможе перевірити

Швидкість та ефективність. За допомогою автоматизованих інструментів тести зазвичай виконуються швидше. Тести можуть проводитися практично без участі людини, що прискорює процес тестування.

Надійність. Автоматизоване тестування є більш точним, оскільки є менш вразливим до помилок людини в порівнянні із ручним тестуванням. Автоматизовані інструменти надійно виконуватимуть тестові сценарії відповідно до заздалегідь виділених вимог та попередньо написаних тестових сценаріїв, що пройшли ретельну перевірку та були одобрені.

Повторюваність. Автоматизовані тести виконуються однаково при будь-якій кількості запусків, що допомагає відкинути можливість непередбачуваної поведінки при виконанні тестів на відміну від ручного тестування.

Недоліки автоматизованого тестування:

– менша гнучкість;

– велика вартість для малих проектів;

– складність створення.

Менша гнучкість. Автоматизоване тестування тісно прив’язане до тестових сценаріїв. Якщо написані тести не враховують усі можливі варіанти поведінки програмного продукти, то присутня можливість отримання неповних результатів.

Велика вартість для малих проектів. Інструменти автоматизації та спеціалісти, що мають досвід роботи з ними можуть бути затратними в плані бюджету, особливо для проектів невеликого обсягу.

Складність створення. Написання та підтримка автоматизованих тестів може бути складним завданням та вимагати досвідчених спеціалістів.

На основі ручного та автоматизованого підходів до тестування програмного забезпечення виділяють ряд типів або методів тестування, кожен з яких спрямований на перевірку певних аспектів програми. Основні типи тестування включають наступні:

– модульне тестування (unit testing);

– інтеграційне тестування (integration testing);

– регресійсне тестування (regression testing);

– тестування зручності використання (usability testing);

– тестування сумісності (compatibility testing);

– тестування відновлення (recovery testing);

– функціональне тестування (functional testing).

Нижче буде приведено опис кожного із вищезгаданих типів тестування та подані їх переваги та недоліки.

Модульне тестування – це метод тестування окремих блоків або компонентів програмного додатку. Зазвичай це робиться розробниками та використовується для

забезпечення належної роботи окремих одиниць програмного забезпечення. Модульні тести зазвичай автоматизовані та призначені для перевірки певних частин коду, наприклад певної функції чи методу.

Модульне тестування проводиться на найнижчому рівні процесу розробки програмного забезпечення, де окремі одиниці коду тестуються окремо[7,13].

Переваги модульного тестування включають:

– виявлення помилок на ранніх етапах;

– ізоляція помилок;

– покращення якості програмного коду;

– автоматизація.

Виявлення помилок на ранніх етапах. Модульні тести дозволяють виявити та виправити помилки на ранніх стадіях розробки, що допомагає зменшити витрати на виправлення помилок з часом.

Ізоляція помилок. Модульне тестування допомагає виокремити та ізолювати помилки в конкретних модулях, що надзвичайно полегшує їх виявлення та виправлення розробниками.

Покращення якості програмного коду. Для написання модульних тестів зазвичай необхідно вести чистий і структурований код, що сприяє поліпшенню якості програмного забезпечення.

Автоматизація. Модульні тести можна автоматизувати, що дозволяє їх запускати під час кожного збирання коду або внесення нових змін в програмному продукті, спрощуючи процес тестування.

Недоліки модульного тестування включають:

– обмежена охопленість;

– витрати на написання тестів;

– нестабільність і зміни;

– необхідність спеціального навчання.

Обмежена охопленість. Модульне тестування детально перевіряє окремі компоненти, але не завжди відображає взаємодію між ними та усім функціоналом програмного продукту.

Витрати на написання тестів. Написання модульних тестів може бути часозатратним завданням, особливо на початкових етапах розробки.

Нестабільність і зміни. Модульні тести можуть стати не ефективними при частих змінах в коді, що вимагає постійного оновлення тестів.

Необхідність спеціального навчання. Написання ефективних модульних тестів вимагає від розробників навичок та розуміння принципів тестування.

Інтеграційне тестування – це тип тестування, що призначений для перевірки того, як різні модулі або компоненти програмного додатку взаємодіють один з одним. Він використовується для виявлення та вирішення будь-яких проблем, які можуть виникнути під час об’єднання різних модулів програмного забезпечення.

Інтеграційне тестування зазвичай виконується після модульного тестування і використовується для перевірки того, що різні модулі програмного забезпечення працюють разом належним чином. Наприклад, це може бути перевірка взаємодії з базою даних або перевірка сумісності усіх незалежних компонентів програми. Даний тип тестів є дорожчими для виконання, оскільки він вимагає запуску та роботи кількох частин програми[11]. Інтеграційне тестування є надзвичайно важливою складовою будь-якого процесу тестування програмного забезпечення.

Перевагами інтеграційного тестування є:

– виявлення проблем на ранніх етапах;

– перевірка відповідності специфікаціям;

– виявлення неправильних даних;

– покращення якості коду.

Виявлення проблем на ранніх етапах. Інтеграційне тестування дозволяє виявити проблеми взаємодії між компонентами програми на самих початкових стадіях розробки.

Перевірка відповідності специфікаціям. Даний тип тестування може допомогти переконатися, що програма відповідає усім функціональним та технічним вимогам.

Виявлення неправильних даних. Під час інтеграційного тестування можна виявити неправильну передачу даних між компонентами.

Покращення якості коду. Інтеграційне тестування може вимагати більшої уваги до якості коду та правильної взаємодії між компонентами.

Недоліками інтеграційного тестування є:

– постійні зміни;

– складність виявлення помилок;

– не завжди можливе повне покриття;

– залежність від інших компонентів;

– складність та витрати.

Постійні зміни. При зміні коду або додаванні нових функцій можуть змінюватися і інтеграційні тести, що вимагає їх постійного підтримування.

Складність виявлення помилок. Помилки, які виникають при інтеграції можуть бути важкими для виявлення та відладки.

Не завжди можливе повне покриття. Повне покриття всієї системи всіма можливими комбінаціями інтеграцій може бути недосяжним завданням.

Залежність від інших компонентів. Для проведення інтеграційних тестів може знадобитися доступ до інших компонентів, які можуть бути ще в розробці або недоступні для тестування.

Складність та витрати. Написання інтеграційних тестів може бути більш складним, а також надзвичайно часоємним завданням в порівнянні з іншими видами тестування.

Інтеграційне тестування є важливою частиною процесу тестування програмного забезпечення, оскільки воно допомагає забезпечити коректну взаємодію між компонентами та виявити проблеми на ранніх етапах розробки. Однак воно також вимагає обережності та витрат на його планування та виконання.

Регресійне тестування – це метод тестування, який використовується для того, щоб переконатися, що зміни внесені до програмного забезпечення не створюють нових помилок і не призводять до збою існуючих функцій. Зазвичай це робиться після внесення змін до коду, таких як виправлення помилок або нових функцій і використовується для перевірки того, що програмне забезпечення все ще працює належним чином.

Основні переваги регресійного тестування включають:

– виявлення впливу змін;

– забезпечення стабільності продукту;

– заощадження часу та ресурсів;

– підтримка якості коду;

– вдосконалення процесу розробки.

Виявлення впливу змін. Регресійне тестування допомагає виявити можливі проблеми та помилки, які можуть виникнути внаслідок внесених змін у програму.

Забезпечення стабільності продукту. Даний метод тестування допомагає забезпечити стабільну роботу продукту після кожної зміни або випуску нової версії.

Заощадження часу та ресурсів. Автоматизоване регресійне тестування може значно зекономити час і зусилля, оскільки дозволяє швидко та ефективно перевірити функціональність продукту після змін.

Підтримка якості коду. Регулярне проведення регресійних тестів спонукає розробників дотримуватися стандартів якості та уникати появи нових помилок.

Вдосконалення процесу розробки. Аналіз результатів регресійних тестів допомагає виявити проблеми в процесі розробки та виправити їх.

Основні недоліки регресійного тестування включають:

Час та ресурси. Проведення повних регресійних тестів може вимагати значних зусиль і часу, особливо в великих проектах.

Постійне оновлення тестів. Після кожної зміни в програмі необхідно оновлювати регресійні тести, що може бути трудомістким завданням.

Помилки в тестах. Якщо регресійні тести містять помилки або не точно відображають поточну функціональність, вони можуть призвести до неправильних результатів, що є суттєвим недоліком.

Необхідність автоматизації. Для швидкого та ефективного проведення регресійних тестів часто необхідна автоматизація, що може вимагати додаткових зусиль та навичок.

Загалом, регресійне тестування є важливою складовою процесу розробки програмного забезпечення, оскільки воно допомагає забезпечити стабільність та

якість продукту після кожної зміни. Однак воно також може бути витратним завданням і тому важливо збалансувати обсяг регресійних тестів з урахуванням ресурсів та потреб проекту.

Тестування зручності використання – це метод перевірки функціональності вебсайту, програми чи іншого цифрового продукту шляхом спостереження за реальними користувачами, які намагаються виконати на ньому завдання. Воно спрямоване на забезпечення задоволення та ефективності користувачів. Даний тип тестування прагне оцінити практичну функціональність продукту, зокрема, наскільки ефективно користувач досягає попередньо визначеної цілі.

Нижче наведено переваги тестування зручності використання:

– покращення користувацького досвіду;

– зменшення кількості помилок;

– виправлення проблем на ранніх етапах;

– підвищення конкурентоздатності.

Покращення користувацького досвіду. Тестування зручності дозволяє ідентифікувати проблеми та бар'єри, які можуть призвести до незадоволення користувачів і покращити користувацький досвід.

Зменшення кількості помилок. Виявлення проблем зручності дозволяє уникнути помилок та невірного розуміння вимог користувачів, що може призвести до невдач та витрат на виправлення.

Виправлення проблем на ранніх етапах. Тестування зручності допомагає виявити проблеми на ранніх етапах розробки, коли їх виправлення є менш витратним, що допомагає зекономити час команді розробників.

Підвищення конкурентоздатності. Забезпечення зручності використання дозволяє покращити продукт та зробити його більш конкурентоспроможним.

Нижче наведено недоліки тестування зручності використання:

– час та витрати;

– суб'єктивність;

– обмежене охоплення;

– потреба у великій кількості учасників.

Час та витрати. Проведення тестів зручності використання може вимагати часу та ресурсів, особливо на початкових етапах розробки.

Суб'єктивність. Оцінка зручності використання може бути суб'єктивною, оскільки вона ґрунтується на враженнях та відчуттях користувачів.

Обмежене охоплення. Тестування зручності використання не завжди може виявити всі можливі проблеми, які виникають під час реального використання продукту, що дає неповне представлення про наявність потенційних невиявлених проблем у ньому.

Потреба у великій кількості учасників. Для проведення ефективних тестів зручності використання є потреба у значній кількості ресурсів.

Незважаючи на недоліки, тестування зручності використання є важливою частиною процесу розробки програмного забезпечення. Воно допомагає перевірити чи продукт відповідає потребам та очікуванням користувачів і сприяє покращенню його якості та конкурентоздатності.

Тестування сумісності – це процес перевірки того, як програмне забезпечення або додаток працює на різних платформах, операційних системах, браузерах, пристроях та середовищах. Тестування сумісності дозволяє забезпечити те, що більша кількість користувачів зможуть користуватись програмним забезпеченням незважаючи на особливості платформ, які вони використовують.

Тестування сумісності характеризується наступними перевагами:

– забезпечення широкого охоплення користувачів;

– зменшення ризику втрат користувачів;

– покращення якості продукту;

– забезпечення працездатності на різних ринках.

Забезпечення широкого охоплення користувачів. Тестування сумісності допомагає переконатися, що продукт може працювати на різних пристроях та платформах, що розширює його аудиторію.

Зменшення ризику втрат користувачів. Якщо програмне забезпечення не сумісне з певними платформами або браузерами це може призвести до втрат

користувачів, таким чином тестування сумісності допомагає уникнути появи таких помилкових ситуацій.

Покращення якості продукту. Виявлення проблем сумісності та їх виправлення покращує якість продукту та сприяє задоволенню цільової аудиторії програмного продукту.

Забезпечення працездатності на різних ринках. Тестування сумісності дозволяє продукту працювати на різних глобальних ринках та міжнародних регіонах, що підвищує його популярність.

Тестування сумісності характеризується наступними недоліками:

Часові та ресурсні витрати. Тестування сумісності може вимагати значних часових та фінансових ресурсів, особливо коли необхідно випробувати багато різних програмних платформ.

Складність виявлення проблем. Деякі проблеми сумісності можуть бути надзвичайно складними для виявлення, оскільки вони можуть виникати в окремих умовах використання.

Необхідність постійного оновлення. Із швидким розвитком нових версій ОС, браузерів та пристроїв необхідно постійно оновлювати тестові сценарії та забезпечувати сумісність.

Специфічність для кожної платформи. Тестування сумісності вимагає розробки та виконання окремих тестових сценаріїв для кожної платформи, що може бути доволі затратним процесом.

Не завжди можливе повне покриття. З урахуванням різних конфігурацій платформ, повне покриття сумісності може бути важким завданням.

Не дивлячись на недоліки, тестування сумісності є важливою частиною процесу розробки програмного забезпечення, оскільки воно допомагає забезпечити те, що програмний продукт буде доступним і працюватиме на різних програмних платформах, що допоможе збільшити аудиторію користувачів.

Тестування відновлення – це вид тестування, що спрямований на перевірку того, як програмне забезпечення чи система відновлюється після виникнення непередбачуваних аварійних ситуацій чи відмов. Тестувальники виконують такі

тести шляхом імітації різних збоїв, таких як збої апаратного забезпечення або мережі, збої програмного забезпечення або відключення електроенергії. Після цього вони перевіряють чи може програма відновитися після певних збоїв і продовжити функціонувати належним чином. Це може допомогти мінімізувати час простою та втрату даних у разі збоїв або помилок.

Тестування відновлення має наступні переваги:

– забезпечення надійності;

– зменшення ризику втрати даних;

– забезпечення безпеки;

– покращення планів відновлення;

– мінімізація впливу відмов.

Забезпечення надійності. Тестування відновлення допомагає переконатися, що програмне забезпечення чи система можуть надійно відновити свою роботу після аварій та відмов.

Зменшення ризику втрати даних. Цей вид тестування допомагає визначити, як система реагує на відмови і чи забезпечує вона збереження даних та інформації.

Забезпечення безпеки. Перевірка процедур відновлення допомагає виявити можливі проблеми з безпекою та відновленням доступу після несподіваних аварій.

Покращення планів відновлення. Результати тестування можуть бути використані для вдосконалення планів відновлення.

Мінімізація впливу відмов. Виявлення та виправлення проблем відновлення перед їх виникненням допомагає зменшити вплив відмов на користувачів та різноманітні бізнес-процеси.

Тестування відновлення має наступні недоліки:

– час та ресурси;

– сценарії відмов;

– підготовка середовища;

– підготовка персоналу.

Час та ресурси. Проведення тестів відновлення може бути часозатратним та ресурсомістким завданням, особливо для великих та складних систем.

Сценарії відмов. Підготовка реалістичних сценаріїв відмов може бути важливою задачею, оскільки треба враховувати різні можливі ситуації.

Підготовка середовища. Для проведення тестів відновлення може бути потрібно підготувати відповідне тестове середовище, що може бути надзвичайно складним завданням.

Підготовка персоналу. Тестування відновлення може вимагати підготовки персоналу до роботи з аварійними ситуаціями та планами відновлення.

Можна сказати, що тестування відновлення є важливою складовою процесу забезпечення надійності програмного забезпечення та систем. Воно допомагає виявити та усунути можливі проблеми відновлення до того, як вони спричинять серйозні проблеми для користувачів та бізнесу.

Функціональне тестування – це вид тестування програмного забезпечення, який перевіряє, чи відповідає функціональність програми вимогам та очікуванням користувачів.

Це тестування не стосується вихідного коду програми. Кожна функціональність програмного додатку перевіряється шляхом надання відповідних тестових вхідних даних, очікування результату та порівняння фактичного результату з очікуваним результатом.

Функціональне тестування надзвичайно сильно зосереджено на перевірці інтерфейсу користувача, API, бази даних, безпеки, клієнтської або серверної програми та функціональності програми, що тестується. Функціональне тестування може бути ручним або автоматизованим. Метою функціонального тестування є перевірка функцій, можливостей і взаємодії системи з різними компонентами у цільовому проекті.

Нижче подані основні переваги функціонального тестування:

– відповідність вимогам;

– забезпечення коректної роботи;

– виявлення функціональних проблем;

– забезпечення стабільності;

– покращення користувацького досвіду.

Відповідність вимогам. Функціональне тестування спрямоване на перевірку відповідності програмного забезпечення вимогам та специфікаціям, що допомагає підтвердити його правильність та повноту.

Забезпечення коректної роботи. Цей вид тестування допомагає виявити та виправити помилки та дефекти, які можуть призвести до некоректної роботи програмного забезпечення.

Виявлення функціональних проблем. Функціональне тестування дозволяє виявити функціональні проблеми, такі як невідповідність вимогам, помилки в логіці програми та інші аномалії.

Забезпечення стабільності. Проведення функціональних тестів допомагає впевнитися в стабільності та надійності програмного забезпечення.

Покращення користувацького досвіду. Функціональне тестування спрямоване на забезпечення задоволення користувачів, перевіряючи, чи працює функціональність програми відповідно до їх очікувань.

Нижче подані основні недоліки функціонального тестування, такі як:

– обмежена орієнтованість;

– суб'єктивність;

– ігнорування змін в інтерфейсі;

– вимоги до ресурсів.

Обмежена орієнтованість. Функціональне тестування фокусується на перевірці конкретної функціональності і воно може не виявити проблеми в інших аспектах якості, таких як безпека, продуктивність та інші.

Суб'єктивність. Виявлення функціональних проблем може бути суб'єктивним процесом, особливо коли вимоги не є чіткими.

Ігнорування змін в інтерфейсі: Функціональне тестування зазвичай не враховує зміни в інтерфейсі користувача та в основному зосереджується на внутрішній функціональності.

Вимоги до ресурсів. Великі програмні системи можуть вимагати значних ресурсів для виконання функціональних тестів, що може бути витратним.

Щоб досягти найкращих результатів в тестуванні, командам слід використовувати різні види тестів, включаючи функціональне тестування, для максимального покриття всіх аспектів якості програмного забезпечення.

Метрики покриття коду є важливим інструментом в процесі розробки програмного забезпечення. Вони допомагають виміряти, наскільки код програми був використаний під час тестування та визначити, наскільки велика частина коду була охоплена тестами[14,15].

Використання метрик покриття коду має декілька переваг у процесі розробки програмного забезпечення:

– виявлення недостатньо протестованих областей;

– покращення якості коду;

– виявлення потенційних проблем;

– зменшення ризику внесення змін;

– підвищення довіри користувачів;

– оптимізація процесу тестування;

– покращення командної роботи.

Виявлення недостатньо протестованих областей. Метрики покриття коду допомагають ідентифікувати ті частини програми, які не були покриті тестами. Це може допомогти розробникам зосередитися на недостатньо протестованих областях і забезпечити більш широке покриття.

Покращення якості коду. Під час розробки, коли розробники відстежують покриття коду вони стають більш уважними до якості свого коду. Це може сприяти покращенню архітектури, розробці більш читабельного та підтримуваного коду.

Виявлення потенційних проблем. Метрики покриття коду можуть допомогти виявити потенційні проблеми та дефекти в програмі, особливо коли код не вдається виконати в результаті тестування.

Зменшення ризику внесення змін. При внесенні змін до існуючого коду, метрики покриття допомагають переконатися, що зміни не вплинули на функціональність і не порушили вже існуючий функціонал.

Підвищення довіри користувачів. Публічне використання метрик покриття коду може збільшити довіру користувачів та клієнтів до продукту, оскільки вони бачать, що програма протестована і покрита тестами.

Оптимізація процесу тестування. Метрики покриття коду допомагають розробникам і тестувальникам зорієнтуватися у тому, які частини програми потребують більш детального тестування і це може оптимізувати витрати часу і ресурсів на тестування.

Покращення командної роботи. Використання метрик покриття коду може покращити комунікацію та співпрацю між розробниками і тестувальниками, оскільки всі сторони більш точно розуміють, які частини програми були вже перевірені, а які потребують додаткової перевірки.Загалом, метрики покриття коду є корисним інструментом для покращення якості програмного забезпечення та зменшення ризику виникнення дефектів. Вони допомагають забезпечити більшу стабільність і надійність програми, що важливо для успішного впровадження та використання продукту.Існує велика кількість різноманітних метрик покриття коду, але основну увагу слід приділити трьом найбільш поширеним, які безпосередньо будуть використовуватись у даній роботі, а саме:

– метрика покриття рядків коду;

– метрика покриття функцій;

– метрика покриття класів.

Нижче поданий короткий опис кожної із вищенаведених метрик.

Метрика покриття рядків коду вимірює відсоток рядків програмного коду, які були виконані під час тестування. Ця метрика допомагає визначити, наскільки код був протестований і які рядки були пропущені під час складання тестів[16].

Метрика покриття функцій вимірює відсоток функцій або методів, які були викликані під час тестування. Вона допомагає переконатися, що всі функції програми були покриті тестовими сценаріями[17].

Метрика покриття класів вимірює відсоток класів програми, які були використані під час тестування. Вона особливо корисна для мов програмування, які використовують об'єктно-орієнтовану парадигму.

Використання цих метрик сприяє покращенню якості тестування та розробки, але важливо пам'ятати, що жодна метрика не є єдиним визначником якості програмного забезпечення і тому вони мають використовуватися разом з іншими методами тестування та аналізу коду.

У сучасному світі технологій розробникам і тестувальникам доводиться пришвидшувати життєві цикли розробки програмного забезпечення. Для того, щоб впоратися зі стислими термінами, розробники програмного забезпечення повинні створювати лише хороший код, тому хороша якість коду є доволі важливої частиною розробки. За допомогою звітів про аналіз метрик покриття коду, розробники можуть відстежувати частку коду, який добре працював у тестових сценаріях. Ця інформація діятиме як звіт із зворотним зв’язком, який допоможе розробникам написати хороший і чистий вихідний код. Зрештою це призведе до покращення якості коду, що позитивно вплине на якість програмного забезпечення, тому метрики покриття коду відіграють важливу роль у розробці якісних та надійних програмних продуктів.

1.2 Огляд існуючих методів та інструментів для збору метрик покриття коду

Тестувальникам і розробникам постійно доводиться мінімізувати життєві цикли розробки програмного забезпечення разом із створенням високоякісного програмного забезпечення для клієнта. Щоб впоратися з такими стислими термінами, розробники програмного забезпечення повинні прагнути створювати лише хороший та якісний код.

За допомогою утиліт, що мають на меті розрахувати метрики покриття коду та провести загальний аналіз покриття коду команда розробників може відстежувати частини коду, які були запущені тестовими сценаріями. Дана інформація діятиме як звіт із зворотним зв’язком, допомагаючи розробникам писати хороший і чистий вихідний код. Зрештою це призведе до покращення якості коду, що позитивно вплине на якість програмного забезпечення.

Слід привести приклади декількох популярних інструментів, що широко використовуються у сучасному світі розробниками програмного забезпечення для розрахунку метрик покриття коду, а також привести їх порівняння із розроблюваною програмною системою.

Перший інструмент має назву Istanbul. Istanbul – це популярний інструмент для збору метрик покриття коду в проектах, написаних на мові програмування JavaScript і TypeScript [18].

Istanbul допомагає розробникам та командам забезпечити ефективне тестування свого програмного коду та оцінювати рівень покриття коду тестами.

Istanbul підтримує різні види покриття коду, включаючи:

– лінійне покриття;

– функціональне покриття;

– гілкове покриття;

– інтеграція з тестовими фреймворками;

– генерація звітів;

– формати виведення;

– конфігурація;

– спільнота та підтримка.

Лінійне покриття. Визначає, які рядки коду були виконані або пропущені під час виконання тестів.

Функціональне покриття. Вказує, які саме функції були викликані, а також скільки разів.

Гілкове покриття. Відстежує відгалуження в коді, показуючи, які гілки були виконані або пропущені.

Інтеграція з тестовими фреймворками. Istanbul може бути інтегрований з популярними тестовими фреймворками, такими як Mocha, Jasmine, Jest та іншими. Він автоматично збирає дані про покриття під час виконання тестів.

Генерація звітів. Інструмент генерує докладні звіти про покриття коду. Ці звіти можуть бути відображені у вебінтерфейсі, що дозволяє розробникам аналізувати покриття на рівні файлів та функцій.

Формати виведення. Istanbul підтримує різні формати виведення, включаючи HTML-звіти для зручного перегляду та звіти у форматі JSON для інтеграції з іншими інструментами.

Конфігурація. Інструмент дозволяє налаштовувати різні аспекти збору метрик, такі як ігнорування певних файлів чи розширень.

Спільнота та підтримка. Istanbul є популярним інструментом у спільноті розробників JavaScript і він активно підтримується.

Приклади розрахунку метрик покриття коду з використанням утиліти Istanbul можна розглянути на рисунку 2.1 та рисунку 2.2.

```
----------------|----------|----------|----------|----------|----------------|
File            |  % Stmts | % Branch |  % Funcs |  % Lines |Uncovered Lines |
----------------|----------|----------|----------|----------|----------------|
All files       |    98.92 |    94.36 |    99.49 |      100 |                |
 yargs          |    99.17 |    93.95 |      100 |      100 |                |
  index.js      |      100 |      100 |      100 |      100 |                |
  yargs.js      |    99.15 |    93.86 |      100 |      100 |                |
 yargs/lib      |     98.7 |    94.72 |    99.07 |      100 |                |
  command.js    |     99.1 |    98.51 |      100 |      100 |                |
  completion.js |      100 |    95.83 |      100 |      100 |                |
  obj-filter.js |     87.5 |    83.33 |    66.67 |      100 |                |
  usage.js      |    97.89 |    92.59 |      100 |      100 |                |
  validation.js |      100 |    95.56 |      100 |      100 |                |
----------------|----------|----------|----------|----------|----------------|
```

Рисунок 2.1 – Консольний вивід результатів метрик покриття коду в Istanbul

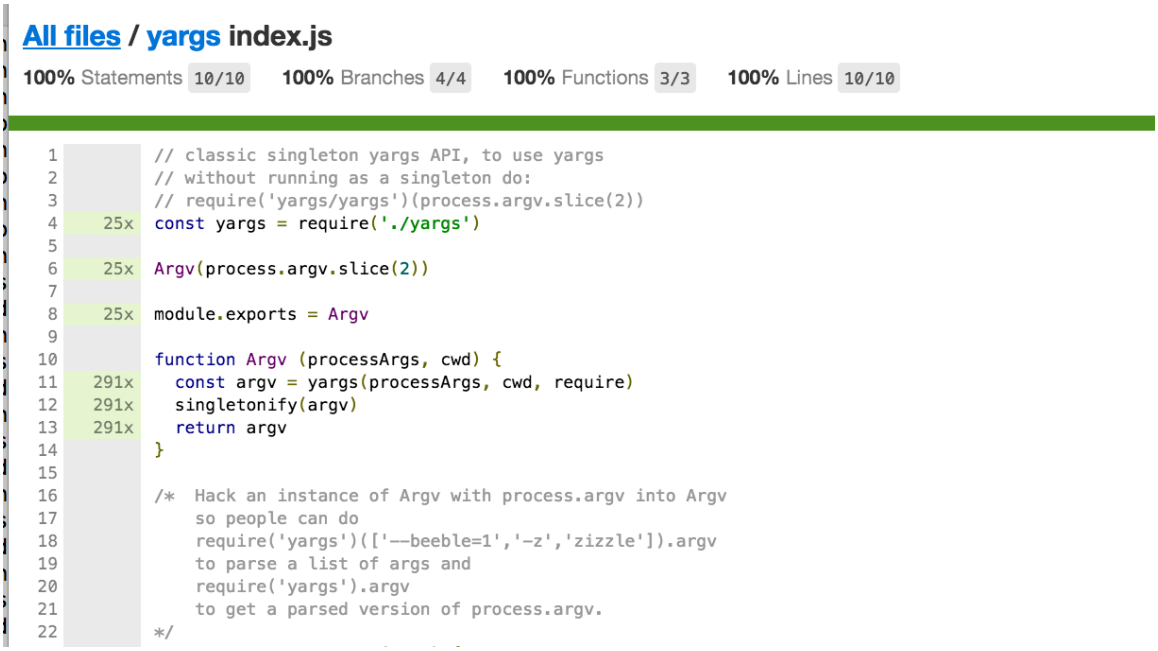


Рисунок 2.2 – Графічне подання результатів метрик покриття коду в Istanbul

Інструмент Istanbul допомагає забезпечити якість коду та забезпечує велику користь при розробці проектів, особливо великих та складних JavaScript-додатків.

Наступним інструментом, що допомагає розраховувати відсоток покритості програмного коду є JaCoCo. JaCoCo є популярним інструментом для збору метрик покриття коду в програмах, написаних на мові програмування Java. Він допомагає розробникам здійснювати оцінку тестового покриття код та визначати, які частини програмного коду були виконані під час виконання тестів, а які залишилися непокритими[19]. Ось основні характеристики інструменту JaCoCo:

– типи покриття коду;

– інтеграція з іншими інструментами;

– генерація звітів;

– конфігурація;

– підтримка мультиплатформеності;

– спільнота та підтримка.

Типи покриття коду. JaCoCo підтримує різні види покриття коду, включаючи лінійне покриття, вігвамне покриття та покриття методів та класів.

Інтеграція з іншими інструментами. JaCoCo може бути інтегрований з популярними інструментами для збору тестів та звітів, такими як Apache Maven, Gradle, та іншими. Це робить його надзвичайно хорошим вибором для проектів у Java екосистемі.

Генерація звітів. Інструмент генерує докладні звіти про покриття коду, які можуть бути представлені у різних форматах, включаючи HTML-звіти для відображення у веббраузері.

Конфігурація. JaCoCo дозволяє налаштовувати різні параметри, включаючи той код, який слід ігнорувати або включати у вимірювання покриття.

Підтримка мультиплатформеності. JaCoCo підтримує Java проекти на різних платформах і може бути використаний в різних інтегрованих середовищах.

Спільнота та підтримка. Інструмент має активну спільноту користувачів і регулярно оновлюється та підтримується.

Остаточні результати розрахунку метрик покриття коду з використанням JaCoCo подано на рисунку 2.3.

JaCoCo

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
org.jacoco.examples		58%		64%	24	53	97	193	19	38	6	12
org.jacoco.core		97%		93%	107	1,388	115	3,347	21	720	2	139
org.jacoco.agent.rt		77%		84%	31	121	62	310	21	74	7	20
jacoco-maven-plugin		90%		81%	35	183	44	407	8	110	0	19
org.jacoco.cli		97%		100%	4	109	10	275	4	74	0	20
org.jacoco.report		99%		99%	4	572	2	1,345	1	371	0	64
org.jacoco.ant		98%		99%	4	163	8	429	3	111	0	19
org.jacoco.agent		86%		75%	2	10	3	27	0	6	0	1
Total	1,355 of 27,352	95%	143 of 2,125	93%	211	2,599	341	6,333	77	1,504	15	294

Рисунок 2.3 – Результати метрик покриття коду в JaCoCo

JaCoCo допомагає розробникам визначати ефективність тестування та розуміти, наскільки компоненти їх програмного коду покриті тестами. Це робить його корисним інструментом для покращення якості і надійності Java додатків[20]. Даний програмний застосунок є доволі популярним.

Останній інструмент, що буде розглянутий, має назву Cobertura. Cobertura є інструментом для вимірювання метрик покриття коду в програмах написаних на мові програмування Java. Він надає можливість розробникам проводити аналіз тестового покриття, визначаючи, які частини програмного коду були виконані під час виконання тестів[21]. Ось деякі ключові характеристики інструменту Cobertura:

– інтеграція з іншими інструментами;

– генерація звітів;

– конфігурація;

– легкість використання;

– підтримка мультиплатформеності.

Інтеграція з іншими інструментами. Cobertura може бути інтегрований з різними інтегрованими розробних середовищами (IDE) та іншими засобами розробки, такими як Apache Maven та Ant.

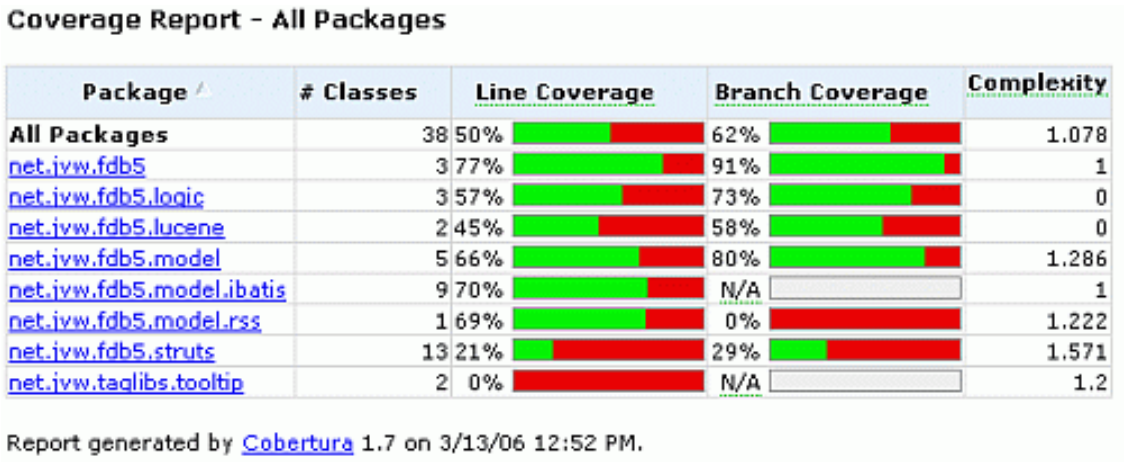
Генерація звітів. Інструмент генерує звіти про покриття коду у різних форматах, включаючи XML, HTML, та інші, що дозволяє аналізувати та відстежувати покриття коду у зручному форматі.

Конфігурація. Cobertura дозволяє налаштовувати параметри аналізу, включаючи вказування того коду, який слід ігнорувати або включати у вимірювання покриття додатково.

Легкість використання. Інструмент має простий інтерфейс, що дозволяє користувачам використовувати його без зайвих труднощів.

Підтримка мультиплатформеності. Cobertura підтримує Java-проекти на різних платформах і може бути використаний в різних розробних середовищах.

Приклад розрахованих показників метрик покриття коду з використанням інструменту Cobertura подано на рисунку 2.4.



Coverage Report - All Packages

Package	# Classes	Line Coverage	Branch Coverage	Complexity
All Packages	38	50%	62%	1.078
net.jvw.fdb5	3	77%	91%	1
net.jvw.fdb5.logic	3	57%	73%	0
net.jvw.fdb5.lucene	2	45%	58%	0
net.jvw.fdb5.model	5	66%	80%	1.286
net.jvw.fdb5.model.ibatis	9	70%	N/A	1
net.jvw.fdb5.model.rss	1	69%	0%	1.222
net.jvw.fdb5.struts	13	21%	29%	1.571
net.jvw.taglibs.tooltip	2	0%	N/A	1.2

Report generated by Cobertura 1.7 on 3/13/06 12:52 PM.

Рисунок 2.4 – Розраховані метрики покриття коду в Cobertura

Cobertura допомагає розробникам оцінювати, як ефективно їхні тестові сценарії покривають програмний код, що допомагає покращувати якість та надійність Java-додатків. Він є популярним інструментом для Java проектів та використовується багатьма розробниками для аналізу метрик покриття коду.

На основі розглянутих інструментів для вимірювання метрик покриття коду можна сказати, що кожен окремий інструмент націлений на розрахунок метрик покриття метрик для окремої мови програмування, а також розрахунок метрик відбувається в ручному режимі, тобто вимагає присутності розробника. Існує

потреба в розробці програмної системи, що буде надавати автоматизований спосіб для розрахунку метрик покриття коду, на основі складених Unit тестів, для якого не потрібна присутність розробників. Програмна система також повинна бути легко адаптованою для розрахунку метрик покриття коду для будь-якої мови програмування. Розроблювана програмна система не є готовим програмним продуктом, вона лише використовує інтеграцію сучасних технологій та платформ і тому є скоріше загальним планом того, як можна організувати якісний процес розрахунку метрик покриття коду. Результати метрик покриття коду повинні бути надісланими на цільову електронну пошту у вигляді прикріпленого текстового файлу, а сам лист повинен містити детальну інформацію про Git коміт та автора цього коміту, тобто змін що були внесені до віддаленого GitHub репозиторію.

1.3 Постановка задачі

Актуальність цієї роботи обумовлена необхідністю вдосконалення методів оцінки та аналізу якості покриття коду, що залишається актуальною та важливою задачею для розробників програмного забезпечення. В умовах постійного розвитку сфери IT виникає потреба у створенні спеціалізованого та ефективного методу роботи з метриками покриття коду, що забезпечить надійну оцінку результатів тестування програмного забезпечення.

Основною метою кваліфікаційної роботи є розробка вдосконаленого методу для розрахунку метрик покриття коду. Це досягається шляхом проектування та впровадження інноваційної програмної системи, яка автоматизує та спрощує процес оцінювання метрик покриття коду на основі результатів тестування.

Задля досягнення поставленої мети, у роботі є необхідність у вирішенні наступних задач:

– провести аналіз предметної області та сформувати постановку задачі;

– проаналізувати існуючі програмні рішення, що використовуються для розрахунку метрик покриття коду та запропонувати удосконалений метод для роботи з метриками покриття коду;;

– розробити детальну архітектуру запропонованої програмної системи;

– на основі плану архітектури провести розробку програмної системи;

– здійснити огляд отриманих результатів та впевнитись у правильності роботи програмної системи;

– запропонувати рекомендації для покращень розробленого програмного рішення у майбутньому.

Об’єктом дослідження є метрики тестового покриття програмного коду.

Предмет дослідження – удосконалений метод для роботи з метриками тестового покриття коду для забезпечення ефективного оцінювання результатів програмного забезпечення.

1.4 Висновки

В ході дослідження предметної області були розглянуто ряд основних понять та термінів, а також проведено аналіз існуючих програмних продуктів, що спеціалізуються на розрахунку метрик покриття коду. Аналіз існуючих інструментів допоміг виділити їх недоліки та сформувати переваги розроблюваної програмної системи, а також необхідність та корисність її розробки.

2 АНАЛІЗ СУЧАСНИХ ТЕХНОЛОГІЙ ТА УТИЛІТ

2.1 Загальний огляд сучасних технологій та утиліт, що будуть використовуватись у розробці програмної системи

У цьому розділі розглядаються сучасні технології та інструменти, які використовуються у розробці програмної системи для оцінювання метрик покриття коду та автоматизації тестування програмного забезпечення. Описані технології та утиліти надають загальне представлення для розуміння того, з яких частин буде складатись програмна система розроблена у даній роботі.

2.1.1 PHP та пакетний менеджер Composer

Розроблювана програмна система буде демонструвати приклад розрахунку метрик покриття коду на прикладі простого проекту написаного на мові програмування високого рівня PHP, що інтегрується із пакетним менеджером Composer з використанням якого інсталюються додаткова PHP бібліотека та фреймворк, що будуть описані далі. Слід більш детально описати мову програмування PHP та пакетний менеджер Composer.

PHP – це серверна сценарна мова з відкритим кодом, яку багато розробників використовують для веброзробки. Це також мова загального призначення, яку можна використовувати для створення багатьох проектів, у тому числі графічних інтерфейсів користувача. Завдяки простому синтаксису та простим у вивченні командним функціям PHP є мовою програмування зручною для початківців. Щоб почати використовувати PHP, важливо знати HTML, а також мови програмування на стороні сервера та клієнта. PHP доступний і простий у вивченні завдяки своїй природі з відкритим вихідним кодом, широкій підтримці баз даних і здатності працювати майже на будь-якому сервері разом з операційними системами, такими як Windows, Unix, Linux або Mac OS. Код PHP інтерпретується в режимі виконання,

а не компілюється до машинного коду, що робить його більш доступним та легким для розробки та підтримки [22,23,24,25].

Завдяки своїй популярності і широкому спектру функціональності, PHP знайшла застосування у багатьох галузях розробки:

– управління контентом;

– електронна комерція;

– вебсервіси;

– робота з базами даних;

– графічна обробка;

– аналітика та звітність;

– робота з API і сервісами.

Управління контентом. Багато популярних систем управління контентом таких як WordPress, Joomla, Drupal, побудовані на PHP. Вони дозволяють створювати та управляти вебсайтами з мінімальними навичками програмування.

Електронна комерція. PHP використовується для розробки інтернет-магазинів та електронних платформ для онлайн-торгівлі. Популярні системи, такі як WooCommerce для WordPress та Magento, базуються на PHP.

Вебсервіси. PHP може бути використаний для створення вебсервісів та API для взаємодії з іншими додатками та сервісами.

Робота з базами даних. PHP має вбудовану підтримку для багатьох систем управління базами даних, таких як MySQL, PostgreSQL, SQLite та інші. Це дозволяє легко взаємодіяти з даними в додатках.

Графічна обробка. За допомогою розширень, таких як ImageMagick, PHP може використовуватися для обробки графіки і створення зображень на льоту.

Аналітика та звітність. PHP дозволяє розробникам створювати інструменти для обробки та аналізу даних, створення звітів та візуалізації результатів.

Робота з API і сервісами. PHP може взаємодіяти з іншими сервісами та API, що робить його корисним для інтеграції зі сторонніми додатками та платформами.

PHP залишається однією з найпоширеніших мов програмування для веброзробки завдяки своїй простоті вивчення та ефективності в роботі з

вебтехнологіями. Багато великих вебсайтів і додатків, таких як Facebook, Wikipedia та WordPress.com, використовують PHP в якості основної мови програмування[26].

Ось декілька переваг, якими володіє PHP:

– простота вивчення та використання;

– широкі можливості;

– швидкість розробки;

– підтримка баз даних;

– велика спільнота розробників;

– безкоштовність;

– широкий вибір інструментів і фреймворків;

– висока продуктивність;

– безпека.

Простота вивчення та використання. PHP має дружній інтерфейс та легкий синтаксис, що робить його відмінним вибором для початківців у веброзробці. Розробники можуть вивчати PHP швидко та впроваджувати його в практиці без значних зусиль.

Широкі можливості. PHP використовується для створення різних типів вебзастосунків, від блогів до складних корпоративних порталів і електронних магазинів. Його функціональність дозволяє розробляти різноманітні додатки.

Швидкість розробки. PHP пропонує багато вбудованих функцій та бібліотек для швидкої розробки. Код може бути перероблений швидко завдяки легкому внесенню змін.

Підтримка баз даних. PHP інтегрується з різними системами управління базами даних, дозволяючи розробникам працювати з даними легко та ефективно.

Велика спільнота розробників. PHP має велику та активну спільноту розробників, яка надає безкоштовну підтримку та допомогу через форуми та блоги.

Безкоштовність. PHP є безкоштовною мовою програмування з відкритим вихідним кодом. Це робить її доступною для всіх розробників.

Широкий вибір інструментів і фреймворків. Для PHP існують численні фреймворки, такі як Laravel, Symfony, CodeIgniter, які спрощують розробку та покращують якість коду[27].

Висока продуктивність. PHP може справлятись із великими навантаженнями, що робить її популярною мовою для великих вебпроєктів.

Безпека. PHP надає інструменти для захисту від вразливостей, таких як SQL-ін'єкції та перехоплення сесій. Розробники можуть дотримуватися найкращих практик для забезпечення безпеки своїх додатків.

Загалом, PHP є потужним і зручним інструментом для веброзробки з багатьма перевагами, які роблять його популярним серед розробників та компаній, що створюють вебдодатки.

Composer – це інструмент для управління залежностями в PHP-проектах. Він дозволяє легко встановлювати, оновлювати та керувати бібліотеками та компонентами, які використовуються в PHP-проектах. Він кардинально змінив екосистему PHP, створивши основу для сучасної розробки PHP із компонентними програмами та фреймворками. Вимоги до пакетів від яких залежить проект оголошуються у файлі composer.json на рівні проекту, який потім використовує Composer, щоб оцінити, які версії пакетів найкраще відповідають залежностям програми. Packagist є загальнодоступним сховищем Composer, що містить велику кількість бібліотек PHP з відкритим вихідним кодом, які можуть бути встановлені через Composer. Розробники можуть публікувати свої пакети на Packagist або власних приватних репозиторіях. Composer автоматично генерує файли автозавантаження, що дозволяє підключати класи та файли залежностей без власного написання підключень.

Composer має численні переваги, які роблять його потужним інструментом для управління залежностями в PHP-проектах. Ось декілька основних переваг пакетного менеджера Composer:

– легкість установки та використання;

– ефективне управління залежностями;

– автоматичне вирішення конфліктів;

– підтримка версіонування;

– гнучкість та розширюваність;

– автоматичне завантаження класів;

– широкий вибір паке;

– підтримка скриптів та подій;

– активна спільнота та підтримка;

– безкоштовність та відкритий вихідний код.

Легкість установки та використання. Composer легко встановлюється на більшість серверів і локальних робочих станціях. Команди Composer дуже зрозумілі та прості для використання.

Ефективне управління залежностями. Composer дозволяє визначити всі залежності вашого проекту в файлі composer.json. Це включає бібліотеки, фреймворки, плагіни та інші компоненти.

Автоматичне вирішення конфліктів. Composer автоматично вирішує конфлікти між версіями залежностей та гарантує, що всі компоненти працюють разом без проблем.

Підтримка версіонування. Composer підтримує систему версіонування Semantic Versioning, що дозволяє точно визначати версії залежностей для встановлюваних PHP бібліотек та фреймворків.

Гнучкість та розширюваність. Існує можливість використовувати власні репозиторії, плагіни та скрипти в Composer, щоб додати додаткову кастомну функціональність та логіку.

Автоматичне завантаження класів. Composer генерує файли автозавантаження, що дозволяють на льоту підвантажувати класи та файли залежностей у програмному коді.

Широкий вибір пакетів. Composer використовує Packagist як основний репозиторій, де розміщені тисячі пакетів і бібліотек для різних цілей, що дає доступ до великої кількості різноманітних PHP фреймворків та бібліотек.

Підтримка скриптів та подій. Composer дозволяє виконувати скрипти перед та після встановлення або оновлення залежностей, що полегшує автоматизацію операцій в проекті.

Активна спільнота та підтримка. Composer має велику та активну спільноту розробників, яка надає підтримку, розробляє плагіни та інструменти.

Безкоштовність та відкритий вихідний код. Composer є безкоштовним програмним забезпеченням з відкритим вихідним кодом.

Composer є стандартом для управління залежностями в PHP розробці та використовується в багатьох проектах і фреймворках. Він значно спрощує процес розробки та підтримки PHP додатків, забезпечуючи легку та ефективну роботу з залежностями. Composer став необхідним і важливим інструментом для PHPрозробників, оскільки він робить процес управління залежностями простим, ефективним і безпечним.

2.1.2 Git, GitHub та GitHub Actions

Git – це розподілена система контролю версій, розроблена Лінусом Торвальдсом у 2005 році. Git став важливим інструментом для розробки програмного забезпечення та керування версіями коду.

У реальних проектах, як правило кілька розробників працюють паралельно, тому потрібна система контролю версій така як Git, щоб гарантувати відсутність конфліктів коду між розробниками. Git не є мовою програмування, але він став неймовірно важливим для великої кількості програмістів, які ведуть розробку будь-якою мовою програмування.

Сьогодні Git є загальним стандартом та зручним підходом для так званого контролю версій у проектах. Програмісти використовують контроль версій, щоб відстежувати оновлення та внесені зміни в код, мати змогу повертатися до попередніх версій проекту, якщо потрібноі бачити будь-які внесені зміни і отримувати інформацію про авторів внесених змін [28,29].

Git пропонує деякі служби керування сховищами, які можуть бути корисними. Загалом, онлайн серсіси з управління віддаленими репозиторіями є запорукою швидкої та ефективної розробки програмного забезпечення. Найпопулярнішими сервісами з онлайн хостингу Git репозиторіїв є GitHub, Bitbucket і GitLab.

У роботі Git використовується для зберігання та керування вихідним кодом програмного забезпечення.

GitHub є однією з найпопулярніших онлайн платформ для спільної роботи над програмними проектами та зберіганням Git репозиторіїв. У даній роботі GitHub використовується як віддалений репозиторій для зберігання програмного коду та для запуску GitHub Actions.

GitHub пропонує як платну так і безкоштовну підписку. За допомогою безкоштовної підписки користувачі можуть створювати загальнодоступні сховища. Загальнодоступні сховища доступні для перегляду будь-кому і будь-хто може їх клонувати або доповнювати. Для створення приватних сховищ потрібна платна підписка. Приватні репозиторії видимі лише для користувачів, яким власник сховища надав доступ. GitHub пропонує такі функції, як відстеження помилок, керування завданнями та керування проектами. Окрім репозиторіїв Git, GitHub також надає розробникам платформу для обміну фрагментами коду, проблемами та документацією. Розробники також можуть використовувати GitHub для відстеження прогресу своїх проектів [30].

Git є системою контролю версій, яка дозволяє розробникам відстежувати зміни у своєму коді. У свою чергу GitHub є вебсервісом для зберігання віддалених Git репозиторіїв. Можна використовувати Git без Github, але не можна використовувати GitHub без Git. Використовуючи Git і GitHub є можливість для отримання доступу до власного коду з будь-якого комп’ютера. Якщо локальну машину пошкодять або викрадуть, то є ризик втрати усього програмного коду та не можливості його відновлення. Однак якщо програмний код зберігається на GitHub, який є хмарною службою, то не потрібно турбуватися про його втрату. Git і GitHub

чудові інструменти, які варто використовувати розробникам під час роботи над будь-яким проектом.

Основними компонентами та поняттями в Git є наступні:

– репозиторій;

– коміти;

– гілки;

– віддалені репозиторії;

– операції злиття та перебазування;

– командний рядок і графічний інтерфейс.

Репозиторій. Репозиторій є одним з основних понять у Git та призначений для зберігання усього програмного коду та його історії змін. Репозиторії можуть бути локальними, віддаленими або розподіленими.

Коміти. Коміти є змінами в коді, які зберігаються в репозиторії. Кожен коміт має унікальний хеш ідентифікатор, автора, дату та повідомлення про зміни. Також можна часто зустрітити таке поняття як «закомічені зміни», що означає зміни, які були збережені в репозиторії.

Гілки. Гілки є окремими шляхами розвитку програмного коду. Вони дозволяють розробникам працювати над різними функціями чи завданнями окремо і пізніше об'єднувати їх. Зазвичай основна гілка в новоствореному репозиторії називається master але можуть бути створені гілки з іншими назвами за потребою.

Віддалені репозиторії. Віддалені репозиторії зберігаються на віддалених серверах. Вони дозволяють розробникам спільно працювати над проектами та обмінюватися змінами. Одним із найпопулярнішим сервісом для віддалених репозиторіїв є GitHub.

Операції злиття та перебазування. Операція злиття дозволяє об'єднувати зміни з однієї гілки в іншу. Перебазування дозволяє впорядковувати історію комітів, переносити коміти з однієї гілки на іншу та робити історію більш чистою.

Командний рядок і графічний інтерфейс. Git може бути використаний через командний рядок або графічний інтерфейс. Це дозволяє розробникам вибрати зручний спосіб взаємодії з Git.

Використання Git має безліч переваг, які роблять його найпопулярнішою системою контролю версій для розробки програмного забезпечення. Ось деякі з найважливіших переваг використання Git:

– розподілена система контролю версій;

– швидкість та ефективність;

– можливості гілкування та злиття;

– локальний резервне копіювання;

– спільна робота;

– GitHub та інші платформи;

– історія та коміти;

– легкість відміни змін.

Розподілена система контролю версій. Всі копії репозиторію Git мають повну історію та можуть функціонувати незалежно одна від одної. Це дозволяє розробникам працювати офлайн та спільно над проектами.

Швидкість та ефективність. Git славиться своєю швидкістю та ефективністю. Він допомагає ефективно керувати великими обсягами даних та забезпечує швидкий доступ до версій файлів та історії змін.

Можливості гілкування та злиття. Git спрощує створення та управління гілками. Розробники можуть створювати окремі гілки для роботи над конкретними функціями чи завданнями та легко зливати їх з основною гілкою.

Локальний резервне копіювання. Вся історія та версії файлів зберігаються локально на комп'ютері розробника. Це дозволяє робити коміти та вести розробку проектів навіть без доступу до Інтернету.

Спільна робота. Git полегшує спільну роботу над проектами з декількома розробниками. Він дозволяє об'єднувати та координувати зміни в коді.

GitHub та інші платформи. Існують платформи для спільної роботи та зберігання Git-репозиторіїв, такі як GitHub, GitLab, Bitbucket тощо. Вони надають інструменти для спільної роботи та відстеження змін, що надає широкий спектр вибору платформи в залежності від обсягу та потреб проекту.

Історія та коміти. Git зберігає історію змін у вигляді комітів, що включають автора, дату, час та коментарі. Це допомагає відстежувати кожну зміну та робити регулярні інформативні ревізії.

Легкість відміни змін. Git дозволяє легко повертатися до попередніх версій файлів чи відміняти зміни.

GitHub Actions – це сервіс, який надається платформою GitHub для автоматизації різних робочих процесів у репозиторіях GitHub. Він дозволяє створювати та налаштовувати автоматичні дії, які виконуються при специфічних подіях, таких як коміти, створення гілок, випуск нових версій тощо. GitHub Actions допомагає спростити і автоматизувати багато аспектів розробки програмного забезпечення, забезпечуючи надійну та повністю інтегровану інфраструктуру для тестування, розгортання та інших робочих процесів[31].

GitHub Actions містить ряд основних компонентів, які є важливими для розуміння, а саме події(events), робочі процеси (workflows), завдання (jobs), дії (actions), середовища запуску (runners). Необхідно привести короткий опис кожного із даних понять.

Робочий процес – це настроюваний автоматизований процес, який виконуватиме одне або кілька завдань. Робочі процеси записуються у файлі YAML, який зберігається у GitHub репозиторії. Він може бути запущеним внаслідок події у Git репозиторії, вручну або за встановленим розкладом. Робочі процеси визначаються в каталозі .github/workflows у сховищі. У цьому файлі будуть визначені події, які можуть ініціювати робочий процес, а також фактичні дії, які мають бути виконаними в робочому процесі. Репозиторій може мати кілька робочих процесів, кожен із яких може виконувати різний набір завдань.

Події – це стани або дії за яких запускаються робочі процеси. Коли хтось надсилає щойно закомічений код до віддаленого GitHub репозиторію, створює запит на відправку локальних файлів, створює запит на приєднання файлів до певної гілки в репозиторії, то спрацьовують деякі події, які в кінцевому рахунку ініціюють запуск завдань та дій у відповідних робочих процесах. Запуск робочих

процесів також можна розпочати вручну або за певним розкладом, а не лише у відповідь на певні події.

Завдання – це набір кроків у робочому процесі, який виконується на одному середовищі запуску. Кожне завдання складається із набору кроків, які виконуються в порядку, який визначає розробник. Кожен робочий процес може складатися з кількох завдань, які виконуються паралельно або слідують одне за одним у заздалегідь визначеній послідовності. Існує також можливість створення залежності між завданнями.

Дія – це спеціальна програма для платформи GitHub Actions, яка виконує складну, але регулярно повторювану роботу. Окремий крок у межах завдання може складатися з послідовності простих консольних команд, але якщо потрібно щось складніше є також можливість використання дій. Одним із ключових факторів дій є те, що вони багаторазові. У той час як виклики консольних команд можна просто включити у файл YAML, який визначає певний робочий процес, дії є автономними програмами на які є посилання у файлі YAML і які можуть використовуватися кількома незалежними робочими процесами. Такий тип багаторазового використання є одним із найпотужніших аспектів дій і робить їх надзвичайно важливими. Можна використовувати готові дії або створювати свої власні.

Середовища запуску – це сервери, які виконують створені розробниками робочі процеси. Кожне середовище запуску може виконувати одне завдання за раз. GitHub надає такі середовища запуску як Ubuntu Linux, Microsoft Windows і macOS для запуску робочих процесів. Кожне середовище запускається в окремій віртуальній машині.

2.1.3 Docker. Платформа AWS та її сервіси

Docker – це платформа для розробки, доставки і запуску застосунків в контейнерах. Контейнери є легковажними, портативними та автономними пакетами програмного забезпечення, які включають код, середовище виконання та всі

залежності, необхідні для запуску додатка. Docker зробив контейнеризацію додатків популярною і забезпечив простий спосіб створення, розгортання та управління контейнерами. Контейнери спрощують розробку та доставку розподілених програм. Вони стають дедалі популярнішими, оскільки організації переходять до хмарної розробки та гібридних мультихмарних середовищ. Розробники можуть створювати контейнери без Docker, працюючи безпосередньо з можливостями, вбудованими в Linux та інші операційні системи, але Docker робить контейнеризацію швидшою, легшою та безпечнішою. Відомо, що понад 13 мільйонів розробників використовують цю платформу [32,33].

Контейнери пропонують логічний механізм упаковки за допомогою якого програми можна абстрагувати від середовища в якому вони працюють. Це відокремлення дозволяє легко та узгоджено розгортати програми на основі контейнерів, незалежно від того чи є цільове середовище приватним центром обробки даних, загальнодоступною хмарою чи навіть персональним ноутбуком розробника. Це дає розробникам можливість створювати передбачувані середовища, ізольовані від решти програм і які можна запускати будь-де.

З точки зору операцій, окрім портативності, контейнери також дають більш точний контроль над ресурсами, підвищуючи ефективність цільової інфраструктури, що може призвести до значно кращого використання обчислювальних ресурсів [34].

Основні поняття та елементи Docker включають:

– контейнери;

– імеджі;

– Dockerfile;

– Docker Compose.

Контейнери. Контейнери Docker ізолюють додатки та їхні залежності від операційної системи та інших контейнерів. Вони дозволяють виконувати додатки в однаковому середовищі незалежно від віртуальної машини чи хост системи.

Імеджі. Імеджі Docker є шаблонами для створення контейнерів. Вони включають код додатка, середовище виконання та залежності. Імеджі можна

створювати самостійно або використовувати готові імеджі з Docker Hub - централізованого репозиторію імеджів.

Dockerfile. Dockerfile – це текстовий файл, що містить інструкції для автоматичного створення імеджа Docker. Він вказує, які команди потрібно виконати для побудови імеджа, включаючи копіювання файлів, встановлення програм та налаштування середовища.

Docker Compose. Docker Compose – це інструмент для опису та запуску багатоконтейнерних додатків. Він дозволяє визначити всі контейнери, що складають додаток, разом із їхніми параметрами та залежностями в одному файлі.

Використання Docker має численні переваги, які роблять його популярним інструментом для розробників та операторів. Основні переваги використання Docker включають:

– портативність;

– швидкість та легкість;

– масштабованість;

– ізоляція;

– спрощена розробка та тестування;

– зберігання та обмін імеджами;

– керування ресурсами;

– спільнота та екосистема;

– інтеграція з іншими інструментами.

Портативність. Docker контейнери є портативними і можуть бути розгорнуті на будь-якому хості, який підтримує Docker, без необхідності модифікації додатку або середовища виконання. Це робить їх ідеальними для розробки на локальному комп'ютері та розгортання в хмарному середовищі або на фізичних серверах.

Швидкість та легкість. Docker контейнери запускаються значно швидше, ніж віртуальні машини і мають менший обсяг. Вони використовують спільне ядро операційної системи з хостом, що робить їх легшими та ефективнішими.

Масштабованість. Docker дозволяє легко масштабувати додатки, додаючи або видаляючи контейнери за потреби. Це робить його ідеальним для вебдодатків і мікросервісної архітектури.

Ізоляція. Контейнери ізолюють додатки один від одного та від хост системи. Це підвищує безпеку та надійність, оскільки проблеми в одному контейнері не впливають на інші.

Спрощена розробка та тестування. Docker спрощує розробку, тестування та розгортання додатків завдяки стандартизації середовища. Розробники можуть працювати в однаковому середовищі, як і відділ тестування та виробництва.

Зберігання та обмін імеджами. Docker надає механізми для зберігання та обміну імеджами контейнерів, що спрощує дистрибуцію додатків та використання готових рішень.

Керування ресурсами. Docker може керувати ресурсами, доступними для контейнерів, такими як пам'ять і CPU, що дозволяє точно налаштовувати використання ресурсів для кожного контейнера.

Спільнота та екосистема. Docker має велику активну спільноту користувачів та багато готових імеджів та інструментів, які роблять його надзвичайно популярним серед розробників.

Інтеграція з іншими інструментами. Docker може легко інтегруватися з іншими інструментами та сервісами, такими як Kubernetes, Jenkins, Travis CI і багато інших.

Загалом, Docker спрощує загальний процес розробки, розгортання та управління додатками.

Платформа AWS – це провідний хмарний провайдер, який надає широкий спектр хмарних послуг та інфраструктури для розробки, тестування та розгортання програмних рішень. У роботі AWS разом із його окремими сервісами є одним із головних компонентів для автоматизації процесу тестування та оцінювання метрик покриття коду.

AWS працює по всьому світу в так званих регіонах, яких загалом є 25 на шести континентах. Кожен регіон складається з кількох зон доступності і це фізичні

центри обробки даних, де перебувають комп’ютери, які є географічно розділені, щоб зменшити ймовірність локальної катастрофи, яка можна пошкодити апаратуру цілого регіону [35].

AWS надає понад 200 сервісів, які можуть швидко та ефективно задовольнити всі потреби будь-якого бізнесу. Сервіси варіюються від базового зберігання та обчислень до більш спеціалізованих нішевих послуг, таких як потокове медіа, робототехніка та навіть квантові обчислення [36]. Окрім звичайного аварійного відновлення та використання віддаленого центру обробки даних, організації все більше використовують AWS як інвестицію в машинне навчання та аналітику даних. Багато організацій масово перемістили всі свої програмні рішення та проекти в AWS і отримали значні переваги в гнучкості, ефективності та надійності.

Основні характеристики та послуги AWS включають:

– обчислювальні послуги;

– зберігання та бази даних;

– мережеві послуги;

– аналітика та машинне навчання;

– інструменти для розробки та доставки;

– хмарні рішення для різних галузей;

– безпека та ідентифікація;

– глобальна присутність.

Обчислювальні послуги. AWS пропонує різні обчислювальні послуги, такі як Amazon EC2, які дозволяють створювати та керувати віртуальними серверами у хмарі. Користувачі можуть вибирати типи і розміри обчислювальних ресурсів відповідно до своїх потреб.

Зберігання та бази даних. AWS надає послуги зберігання даних, такі як Amazon S3 для зберігання об'єктів та Amazon RDS для керування реляційними базами даних. Ці послуги забезпечують надійність, масштабованість та безпеку даних у проектах.

Мережеві послуги. AWS має різноманітні мережеві послуги, включаючи Amazon VPC, яка дозволяє створювати ізольовані мережі та інші.

Аналітика та машинне навчання. AWS пропонує послуги для обробки та аналізу даних, такі як Amazon EMR і Amazon Redshift. Крім того, AWS має платформу для машинного навчання, яка включає Amazon SageMaker.

Інструменти для розробки та доставки. AWS надає інструменти для автоматизації розробки та доставки, включаючи Amazon CodeDeploy, Amazon CodePipeline та інші.

Хмарні рішення для різних галузей. AWS пропонує спеціалізовані хмарні рішення для різних галузей, включаючи хмарні послуги для медицини, фінансів, інтернету речей та інших секторів.

Безпека та ідентифікація. AWS надає різноманітні засоби та послуги для забезпечення безпеки даних та додатків, включаючи ідентифікацію, керування доступом та моніторинг.

Глобальна присутність. AWS має дата-центри та регіональні центри даних у різних куточках світу, що дозволяє користувачам розгортати додатки та послуги біля своїх користувачів.

AWS має численні переваги, які зробили його одним із провідних хмарних платформ у світі. Ось деякі з переваг AWS:

– широкий спектр послуг;

– масштабованість;

– глобальна присутність;

– безпека;

– надійність;

– економія коштів;

– інновації;

– підтримка та сервіс;

– екосистема партнерів.

Широкий спектр послуг. AWS пропонує велику кількість різноманітних хмарних послуг, включаючи обчислювальні, зберігання, бази даних, аналітику, штучний інтелект, машинне навчання і багато інших. Це дає можливість користувачам вибирати та налаштовувати послуги під свої потреби.

Масштабованість. AWS дозволяє користувачам гнучко масштабувати свої додатки та інфраструктуру в залежності від навантаження. Це допомагає ефективно використовувати ресурси та зменшує витрати.

Глобальна присутність. AWS має регіональні центри даних по всьому світу, що дозволяє користувачам розміщувати свої додатки та дані ближче до своїх користувачів, зменшуючи затримки та підвищуючи швидкість доступу.

Безпека. AWS забезпечує надзвичайно високий рівень безпеки та пропонує інструменти для захисту даних, включаючи мережеву захист, ідентифікацію, а також керування доступом.

Надійність. AWS гарантує високу доступність послуг завдяки резервному копіюванню та реплікації даних, а також автоматичному відновленню систем.

Економія коштів. AWS пропонує гнучкий модель ціноутворення, де користувачі платять лише за використані ресурси. Це дозволяє зменшити витрати на інфраструктуру.

Інновації. AWS постійно впроваджує нові технології та послуги, що дозволяє користувачам бути на передовій в галузі інформаційних технологій.

Підтримка та сервіс. AWS пропонує широкий спектр послуг підтримки, включаючи технічну цілодобову підтримку.

Екосистема партнерів. AWS має велику спільноту партнерів, включаючи незалежних розробників програмного забезпечення, консультантів та інтеграторів систем, які допомагають користувачам впроваджувати та оптимізовувати рішення на платформі AWS.

Ці переваги роблять хмарну платформу AWS популярним вибором для підприємств та розробників, які шукають надійну та масштабовану хмарну інфраструктуру для своїх проектів.

Як уже зазначалось, AWS надає доступ до великої кількості сервісів, але у даній роботі будуть використовуватись лише деякі з них, а саме Amazon ECS, Amazon ECR, Amazon EventBridge та Amazon S3. Необхідно подати опис кожного із зазначених AWS сервісів задля кращого розуміння їх використання.

Amazon ECS – це сервіс управління Docker контейнерами, який надає можливість розгортати та керувати контейнерами у хмарному середовищі AWS.

Основні характеристики та функціональність Amazon ECS включають:

– керування контейнерами;

– масштабування;

– інтеграція з іншими сервісами AWS;

– підтримка оркестраторів контейнерів;

– спеціалізовані рішення.

Керування контейнерами. Amazon ECS дозволяє легко розгортати, керувати та масштабувати контейнери на основі Docker. Присутня можливість використовувати власні контейнери або використовувати готові образи з Docker Hub або інших репозиторіїв.

Масштабування. Можна налаштувати автоматичне масштабування своїх контейнерних додатків на основі навантаження або правил. Amazon ECS дозволяє створювати групи контейнерів та керувати ними зручно та ефективно.

Інтеграція з іншими сервісами AWS. Amazon ECS інтегрується з іншими сервісами AWS, такими як Amazon VPC, AWS Fargate, AWS Identity and Access Management і багатьма іншими. Це дозволяє створювати складні рішення та використовувати різні послуги для покращення функціональності контейнерних додатків побудованих у платформі AWS.

Підтримка оркестраторів контейнерів. Amazon ECS підтримує інтеграцію з оркестраторами контейнерів, такими як Kubernetes, що дозволяє використовувати необхідні інструменти та підходи для керування контейнерами у хмарному середовищі AWS.

Спеціалізовані рішення. AWS пропонує спеціалізовані рішення на основі Amazon ECS для різних галузей, таких як медицина, фінанси та медіа.

Amazon ECS робить розгортання та керування контейнерами простим та ефективним, що дозволяє розробникам швидко створювати, масштабувати та управляти контейнерними додатками у хмарному середовищі AWS.

Amazon ECS має декілька основних компонентів, які дозволяють створювати та керувати контейнерами в хмарному середовищі AWS:

Кластер (cluster). Кластер – це група віртуальних або фізичних серверів, які призначені для розгортання та виконання Docker контейнерів. Кластери містять у собі сервіси із завданнями або лише завдання.

Визначення завдання (task definition). Визначення завдання – це опис, що визначає, які Docker контейнери повинні бути виконані як частина завдання. Воно включає в себе інформацію про контейнери, які повинні бути використані, а також різноманітні конфігураційні параметри.

Завдання (task). Завдання – це екземпляри визначення завдання, які запускаються на певному кластері. Кожне завдання може представляти лише один контейнер. Завдання можна представити у вигляді запущеного Docker контейнеру.

Сервіс (service). Сервіс визначає як кілька екземплярів завдань повинні бути розгорнуті та масштабовані в межах певного кластера. Сервіс дозволяє автоматично керувати завданнями та забезпечувати високу доступність додатків. Сервіси можуть містити один або декілька завдань або контейнерів та використовуються коли необхідно забезпечити довготривалу роботу одного або групи контейнерів.

У даній роботі Amazon ECS використовується для автоматичного розгортання та виконання Docker контейнера, який містить скрипти для обрахунку метрик покриття коду та відправки результатів.

Amazon ECR – це керований сервіс реєстру контейнерів, який дозволяє зберігати, керувати та масштабувати Docker образи в хмарному середовищі AWS. Amazon ECR надає інфраструктуру для зберігання Docker образів, а також забезпечує можливість інтегрувати ці образи з іншими AWS-сервісами та інструментами, такими як наприклад Amazon ECS. За допомогою Amazon ECR, розробники можуть ефективно керувати та зберігати свої контейнери, що дозволяє їм швидко та надійно розгортати свої додатки в хмарному середовищі AWS. Існує можливість створювати приватні та публічні реєстри образів Docker. У кваліфікаційній роботі Amazon ECR використовується для збереження одного, заздалегідь створеного, Docker образу та інтегрується із сервісом Amazon ECS.

Amazon EventBridge – це керований сервіс розподілених подій, який дозволяє легко інтегрувати та координувати реакцію на події в різних додатках та сервісах в хмарному середовищі AWS. Amazon EventBridge розроблений для створення архітектур із подій, які дозволяють різним сервісам AWS взаємодіяти між собою з меншими зусиллями та підвищеною ефективністю. Amazon EventBridge дозволяє будувати розподілені системи з подій, що реагують на зміни у різних сервісах та додатках. Він є потужним інструментом для створення багатокомпонентних додатків, які взаємодіють та реагують на реальні події, що відбуваються в хмарному середовищі AWS. У роботі використання Amazon EventBridge полягає у створенні події, яка при додаванні нових файлів до Amazon S3 буде запускати Docker контейнер або завдання в сервісі Amazon ECS.

Amazon S3 – це об'єктне сховище в хмарному середовищі AWS, яке надає масштабоване та довгострокове зберігання об'єктів, таких як файли, зображення, відео та інші дані. Amazon S3 є однією з основних послуг AWS і використовується для зберігання, управління та розподілу даних в інтернеті. Amazon S3 використовується для різних завдань, включаючи зберігання резервних копій даних, вебхостинг, розподілення медіа-контенту, обробку даних та багато інших. Він є важливою послугою для багатьох організацій та розробників, які працюють з об'єктами та даними в хмарному середовищі AWS. У кваліфікаційній роботі Amazon S3 буде використовуватись в якості сховища файлів для PHP проекту. Файли будуть надходити у Amazon S3 із GitHub репозиторію з використанням інструменту GitHub Actions [37].

2.1.4 Фреймворк PHPUnit

PHPUnit – це популярний фреймворк для тестування коду в мові програмування PHP. Він розроблений для автоматизації процесу створення і виконання юніт тестів програмного забезпечення та функціональних тестів для різноманітних PHP-додатків.

PHPUnit допомагає розробникам виявляти помилки та проблеми у своєму коді та переконуватися, що вони не виникають при внесенні змін в додаток [38]. Даний фреймворк доволі часто використовується у PHP проектах.

Основні характеристики та можливості PHPUnit включають:

– автоматизація тестування;

– робота з різними типами тестів;

– підтримка кодування згідно стандартів;

– підтримка розширень;

– інтеграція з іншими інструментами;

– підтримка тестового покриття коду.

Автоматизація тестування. PHPUnit дозволяє створювати автоматизовані тести, які виконуються без втручання користувача. Це дозволяє розробникам швидко перевіряти функціональність свого коду.

Робота з різними типами тестів. PHPUnit підтримує створення і запуск різноманітних видів тестів, включаючи юніт тести, інтеграційні тести, а також функціональні тести.

Підтримка кодування згідно стандартів. PHPUnit сприяє дотриманню стандартів написання коду і тестів, включаючи PSR.

Підтримка розширень. Є багато розширень для PHPUnit, які доповнюють його функціональність, наприклад, PHPUnit Selenium для автоматизації тестів вебдодатків та інші.

Інтеграція з іншими інструментами. PHPUnit може бути легко інтегрований з іншими інструментами для неперервної інтеграції, такими як Jenkins, Travis CI.

Підтримка тестового покриття коду. PHPUnit може генерувати звіти про метрики покриття коду, що допомагає визначити, які частини коду були виконані під час запуску юніт тестів, а які ні.

PHPUnit є незамінним інструментом для розробників PHP, який спрощує процес тестування та допомагає забезпечити якість коду. Він є частиною PHP розробки та активно використовується спільнотою програмістів для забезпечення надійності та функціональності PHP-додатків. Даний фреймворк використовується

у кваліфікаційній роботі з метою написання та виконання юніт тестів, а також генерації звітів із результатами образунку метрик покриття коду [39,40,41].

2.1.5 Бібліотека PHPMailer

PHPMailer – це потужна бібліотека для надсилання електронних листів з PHP-додатків. Вона надає зручний і простий спосіб надсилання електронних повідомлень через SMTP сервери або локальний агент пошти [42]. Основні характеристики та можливості бібліотеки PHPMailer включають:

– підтримка різних протоколів;

– підтримка HTML і текстових повідомлень;

– підтримка вкладень;

– підтримка розсилки;

– аутентифікація і безпека;

– підтримка кастомізації заголовків і повідомлень;

– обробка помилок та логування.

Підтримка різних протоколів. PHPMailer підтримує різні протоколи надсилання листів, включаючи SMTP, Sendmail, Qmail, та інші. Це дозволяє використовувати різні методи надсилання в залежності від конкретних вимог.

Підтримка HTML і текстових повідомлень. Присутня можливість легко створювати і надсилати як HTML, так і звичайні текстові повідомлення.

Підтримка вкладень. PHPMailer дозволяє прикріплювати файли до електронних листів, що робить його ідеальним для надсилання файлів через електронну пошту.

Підтримка розсилки. Можна використовувати PHPMailer для надсилання листів до багатьох адресатів одночасно, налаштовуючи список адрес.

Аутентифікація і безпека. Бібліотека підтримує аутентифікацію на SMTP-серверах, а також може працювати через SSL або TLS для забезпечення безпеки передачі даних.

Підтримка кастомізації заголовків і повідомлень. Існує можливість налаштовувати заголовки і вміст повідомлень, включаючи додавання кастомних заголовків і вказівку кодування.

Обробка помилок та логування. PHPMailer надає можливість обробляти помилки під час надсилання листів і вести журнал подій для їх відстеження.

PHPMailer є надійним і розширюваним інструментом для роботи з електронною поштою в PHP-додатках. Він дозволяє створювати та надсилати листи із сервера розробника або стороннього SMTP сервера, забезпечуючи при цьому контроль і безпеку цільового проекту. У даній роботі бібліотека PHPMailer використовується за своїм прямим призначенням, а саме задля відправки електронних листів, що містять інформацію про створений Git коміт та результати розрахунку метрик покриття коду [43].

2.2 Висновки

В результаті проведеного аналізу сучасних технологій та утиліт були приведені їх детальні описи та вказані способи їх використання у розроблюваній програмній системі. Розгляд кожної технології допоміг краще зрозуміти призначення та необхідність їх використання у кваліфікаційній роботі.

3 РОЗРОБКА ПРОГРАМНОЇ СИСТЕМИ

3.1 Проектування архітектури програмної системи

Важливим етапом кваліфікаційної роботи є проектування архітектури програмної системи, оскільки потрібно розуміти послідовність взаємодії сервісів, що будуть задіяні при безпосередній реалізації програмної системи. Правильний план архітектури є запорукою кращого розуміння взаємодії усіх компонентів програмної системи та її подальшої ефективної і чіткої реалізації. Етап розробки архітектури визначає майбутні можливості та ефективність програмної системи, тому його правильне виконання відіграє вирішальну роль у досягненні мети даної кваліфікаційної роботи. Загальна схема архітектури програмної системи зображена на рисунку 3.1.

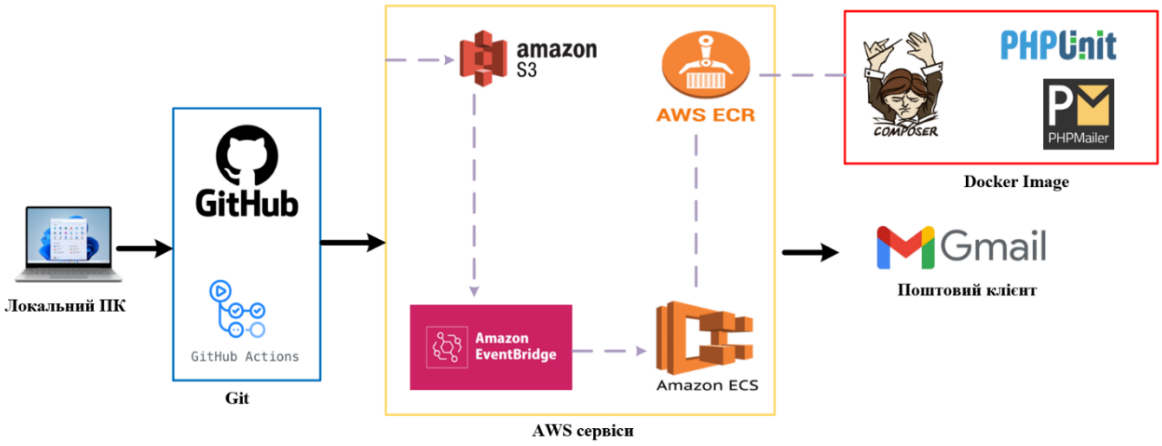


Рисунок 3.1 – Схема архітектури програмної системи

За розглянутою схемою вище можна виокремити загальні послідовні кроки, які відображають процес розробки та автоматизації оцінки метрик покриття коду.

Крок 1. Розробник створює програмний застосунок на мові високого рівня PHP. Після написання додатку, локально створюються юніт тести з використанням фреймворку PHPUnit.

Крок 2. Використовуючи систему контролю версій Git розробник відсилає свої зміни у віддалений репозиторій на GitHub.

Крок 3. З використанням GitHub Actions, у відповідь на подію відсилання змін в окрему наперед визначену гілку, відбувається запуск ряду команд, що ініціюють створення текстового файлу з інформацією про розробника, з'єднання із сервісом AWS S3 Bucket та відправку файлів із GitHub репозиторію у сховище S3 Bucket.

Крок 4. Після відправки файлів у S3 Bucket запускається наперед створена подія з використанням Amazon EventBridge, яка запускає у свою чергу Docker контейнер у сервісі Amazon ECS.

Крок 5. Контейнер запускається на основі наперед запакованого Docker імеджу, що зберігається у публічному репозиторії сервіса ECR. Даний імедж містить у собі наперед встановлений пакетний менеджер Composer, фреймворк PHPUnit та бібліотеку PHPMailer.

Крок 6. Запущений ECS контейнер, містить у собі консольний скрипт, який містить команди, що послідовно стягують усі файли із S3 Bucket, запускають юніт тести та створюють звіти метрик покриття коду з використанням фреймворку PHPUnit. Після створення репорту по метрикам покриття коду здійснюється запуск PHP скрипту, який виконує відправку листа з використанням бібліотеки PHPMailer. Електронний лист приходить на поштовий клієнт Gmail та містить детальні дані про автора Git коміту та текстовий файл із результатами розрахунку метрик покриття коду.

Вищенаведена послідовність кроків ілюструє процес автоматизованої оцінки покриття коду та інтеграцію розробки з послугами хмарних платформ, спрощуючи рутинні завдання розробників і покращуючи ефективність процесу розрахунку метрик покриття коду для цільового програмного забезпечення на основі написаних юніт тестів.

3.2 Реалізація логіки програмної системи

Опис реалізації програмної системи доцільно почати у послідовності, що була показана на схемі архітектури у попередньому підрозділі. Першим етапом

відбувається написання простого консольного застосунку на мові високого рівня PHP у інтегрованому середовищі розробки PHPStorm. Програма представляє собою калькулятор, що має можливість виконувати прості математичні операції, такі як ділення, множення, віднімання та додавання. Загальний вигляд структури файлів та папок проекту у PHPStorm можна побачити на рисунку 3.2.

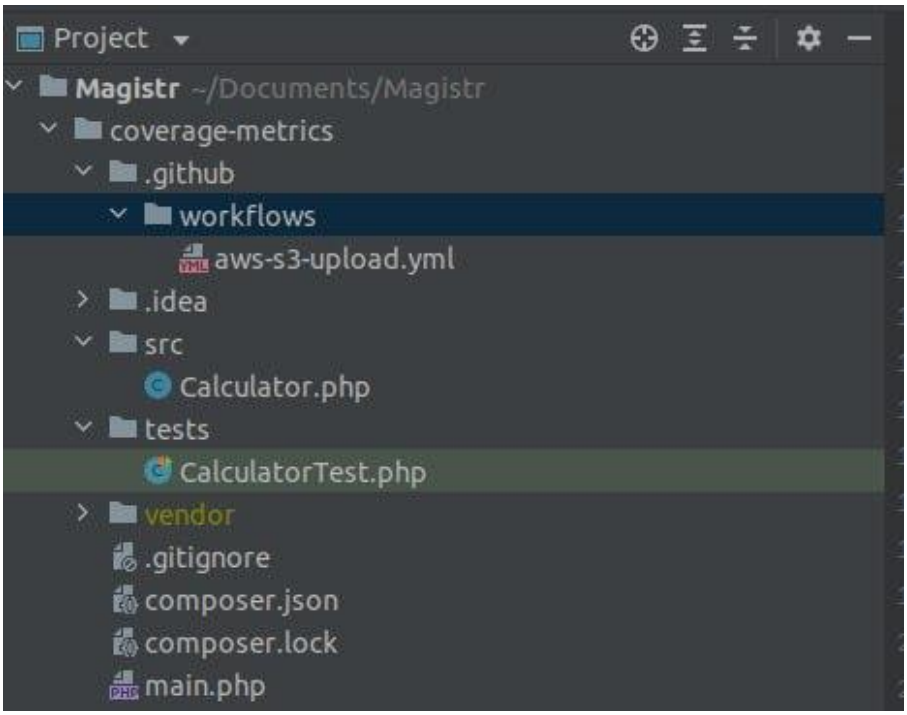


Рисунок 3.2 – Загальний вигляд файлів робочої директорії

Безпосередня реалізація раніше згаданих математичних операцій представлена у файлі Calculator.php, що розміщений у директорії src. Усі математичні операції представлені у вигляді окремих PHP методів та описані у класі Calculator. Кожен метод очікує на вході два аргументи, тобто два числа над якими буде виконуватись певна математична операція. Програмний код файлу Calculator.php зображено нижче:

```
<?php
declare(strict_types=1);

class Calculator
{
    public function addition(int $arg1, int $arg2)
    {
        return $arg1 + $arg2;
```

```
    }

    public function subtraction(int $arg1, int $arg2)
    {
        return $arg1 - $arg2;
    }

    public function division(int $arg1, int $arg2)
    {
        if ($arg2 === 0) {
            throw new Exception('Exception division by zero!');
        }
        return $arg1 / $arg2;
    }

    public function multiplication(int $arg1, int $arg2)
    {
        return $arg1 * $arg2;
    }
}
```

Головним файлом, що здійснює виклик вищенаведених операцій, є PHP скрипт під назвою main.php. Даний файл для прикладу містить виклик лише операції додавання. Він також містить рядок підключення попереднього скрипту через команду require_once. Його програмний код подано нижче:

```
<?php

require_once 'src/Calculator.php';

$calculator = new Calculator();

print $calculator->addition(6, 5);
```

Папка .idea була створена автоматично та містить файли, що стосуються середовища розробки PHPStorm і не використовується напряму.

Файл .gitignore містить папки, що не повинні бути включені до GitHub репозиторію, але які можуть існувати локально.

Файли composer.json та composer.lock містять налаштування пов’язані із пакетним менеджером Composer. Вони використовуються для встановлення фреймворку PHPUnit та усіх його залежностей, що надає можливість написання юніт тестів на локальному комп’ютері розробника. У проекті папка vendor також належить Composer та містить безпосередньо встановлений фреймворк PHPUnit

разом із його залежностями, а також інші файли Composer. Саме ця папка додана до файлу .gitignore, що допомагає виключити її із GitHub репозиторію.

У папці tests міститься PHP скрипт CalculatorTest.php, що безпосередньо призначений для написання юніт тестів для раніше розглянутого скрипту Calculator.php. Програмний код для скрипта CalculatorTest.php подано нижче:

```
<?php
declare(strict_types=1);

require_once __DIR__ . '/../src/Calculator.php';

class CalculatorTest extends \PHPUnit\Framework\TestCase
{
    public function testAddition()
    {
        $calculator = new Calculator();

        $result = $calculator->addition(3, 2);

        $this->assertSame(5, $result);
    }

    public function testSubtraction()
    {
        $calculator = new Calculator();

        $result = $calculator->subtraction(3, 2);

        $this->assertSame(1, $result);
    }

    public function testDivision()
    {
        $calculator = new Calculator();

        $result = $calculator->division(14, 2);

        $this->assertSame(7, $result);
    }
}
```

У скрипті CalculatorTest.php міститься клас CalculatorTest, що наслідується від головного класу фреймворку PHPUnit під назвою \PHPUnit\Framework\TestCase. У CalculatorTest.php міститься три методи, що призначені для перевірки операцій додавання, віднімання та ділення. У кожному методі відбувається виклик відповідних математичних методів та передача двох числових параметрів. Далі використовується метод класу PHPUnit під назвою assertSame, що перевіряє правильність роботи математичних методів скрипта Calculator.php. Метод

assertSame приймає два параметри, першим параметром є число, а другим змінна, що містить результат математичної операції.

Для того, щоб тести були успішними, потрібно щоб у всіх трьох методах результати операцій збігались із першим параметром методу assertSame.

У даному випадку усі три тестових сценарії, що представлені методами testAddition, testSubtraction та testDivision відпрацьовують справно та успішно при виконанні запуску юніт тестів.

Останнім файлом у робочій директорії, що слід розглянути є файл робочого процесу GitHub Actions під назвою aws-s3-upload.yml, що міститься у директорії .github/workflows, програмний код якого подано нижче:

```
name: coverage-metrics-aws-s3

on:
  push:
    branches: [ "develop" ]

jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout to repository
        uses: actions/checkout@v3

      - name: Create .txt file filled with committer details
        run: |
          echo AuthorEmail: ${{ github.event.head_commit.author.email }}  >
author.txt
          echo AuthorName: ${{ github.event.head_commit.author.name }} >>
author.txt
          echo CommitMessage: ${{ github.event.head_commit.message }}  >>
author.txt
          echo CommitTime: ${{ github.event.head_commit.timestamp }} >> author.txt
          echo CommitId: ${{ github.event.head_commit.id }}  >> author.txt



      - name: Configure AWS Credentials
        uses: aws-actions/configure-aws-credentials@v1
        with:
          aws-access-key-id: ${{ secrets.AWS_ACCESS_KEY }}
          aws-secret-access-key: ${{ secrets.AWS_SECRET_ACCESS_KEY }}
          aws-region: us-east-1

      - name: Deploy repository files to S3 bucket
        run: aws s3 sync . s3://${{ secrets.AWS_BUCKET }} -delete
```

Слід розглянути вищенаведений YML файл робочого процесу під назвою coverage-metrics-aws-s3 більш детально. Даний робочий процес є основним

елементом, що надає можливість автоматизованого запуску певних процесів у відповідь на подію, що відбувається у GitHub репозиторії та яка визначена командою on. Саме подія відправки змін у гілку develop, що містить усі раніше розглянуті файли робочої директорії, запускає робочий процес coverage-metrics-aws-s3. Дана подія описана у файлі aws-s3-upload.yml.

Наступною командою у робочому процесі є запуск одного завдання у секції jobs. Дане завдання буде запущене у окремому середовищі запуску, побудованого на основі операційної системи Ubuntu останньої версії. Середовище запуску описано командою runs-on. Далі команда steps визначає послідовні кроки, що будуть виконані у межах даного завдання. Усього кроків є чотири. Перший крок має назву Checkout to repository та використовує дію actions/checkout@v3. Дана дія виконує стягування усіх файлів поточного репозиторію всередину середовища запуску, що базується на Ubuntu, що надає можливість виконувати операції над файлами GitHub репозиторію безпосередньо на окремій віртуальній машині. Наступний крок має назву Create .txt file filled with committer details та виконує ряд консольних скриптів, що призначені для створення текстового файлу author.txt. У даний файл відбувається запис інформації про Git коміт, а саме ім’я та емейл автора коміту, повідомлення коміту, дата створення коміту та айді коміту. Дана інформація витягується з використанням вбудованих змінних, що витягуються через команду github.event.head_commit. Даний файл буде використаний на етапі відправки емейл листа на поштовий клієнт Gmail. Третій крок має назву Configure AWS Credentials та використовує дію aws-actions/configure-aws-credentials@v1, що призначена для підключення до сервісу AWS S3. У команді with вказується додаткова інформація, а саме публічний і приватний ключ та регіон. Публічний та приватний ключ були вказані явно у налаштуваннях даного GitHub репозиторію та викликаються через команди secrets.AWS_ACCESS_KEY та secrets.AWS_SECRET_ACCESS_KEY, що допомагає забезпечити їх конфіденційність. Останній крок має назву Deploy repository files to S3 bucket та використовується для відправки усіх файлів, що містяться у поточній директорії віртуальної машини до сервісу AWS S3. Даний крок використовує консольний скрипт у якому за допомогою команди aws s3 sync

відбувається відправка усіх файлів до вказаного сховища в AWS S3. Назва сховища підставляється через команду secrets.AWS_BUCKET у не явному вигляді.Загальний вигляд GitHub репозиторію під назвою coverage-metrics зображено на рисунку 3.3:

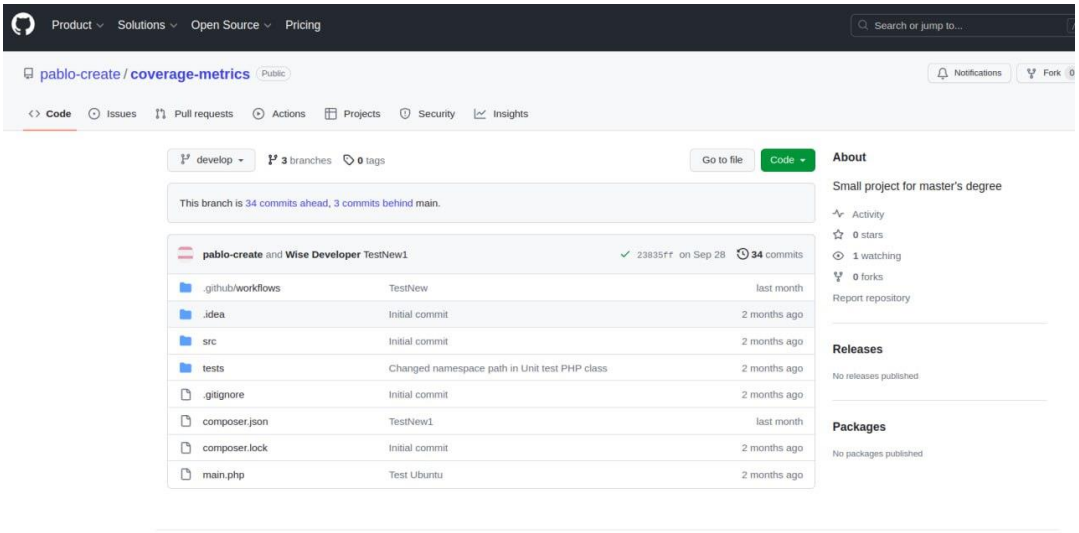


Рисунок 3.3 – GitHub репозиторій

Вигляд виконаних GitHub Actions можна глянути у репозиторії всередині секції Actions. Вони зображені на рисунку 3.4.

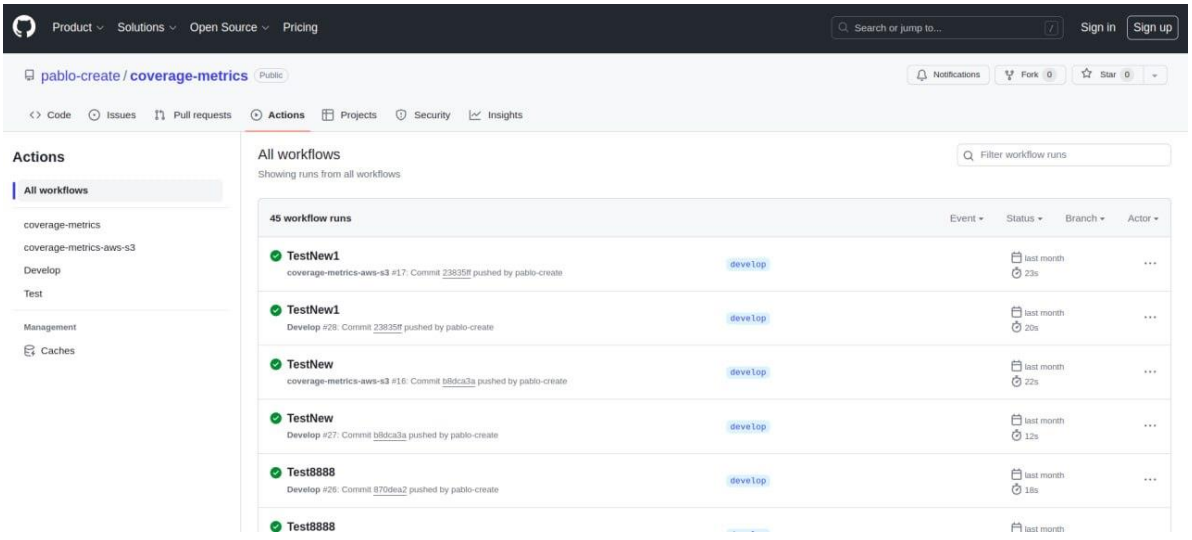


Рисунок 3.4 – Список виконаних GitHub Actions

Наступним кроком можна подати вигляд створеного сховища AWS S3 під назвою code-coverage-metrics-bucket, що містить усі файли GitHub репозиторію,

разом із текстовим файлом author.txt, що був створений на етапі відпрацювання GitHub Actions. AWS S3 сховище зображене на рисунку 3.5.

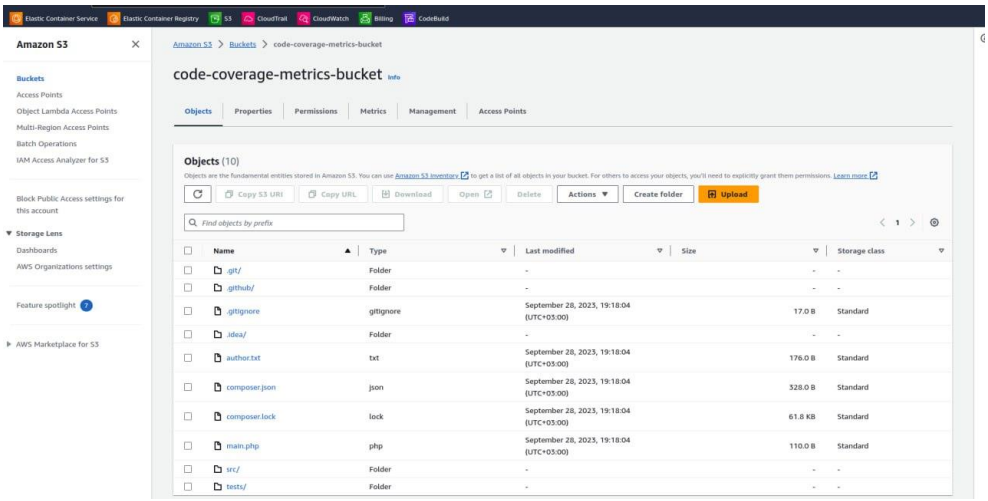


Рисунок 3.5 – Вигляд AWS S3 сховища

Наступний сервіс амазону, що використовується при розробці програмної системи має назву AWS EventBridge. Він використовується для створення правила події, що запускає докер контейнер у AWS ECS кожного разу як додаються нові файли до AWS S3 сховища. Загальний вигляд списку створених правил для AWS EventBridge зображено на рисунку 3.6.

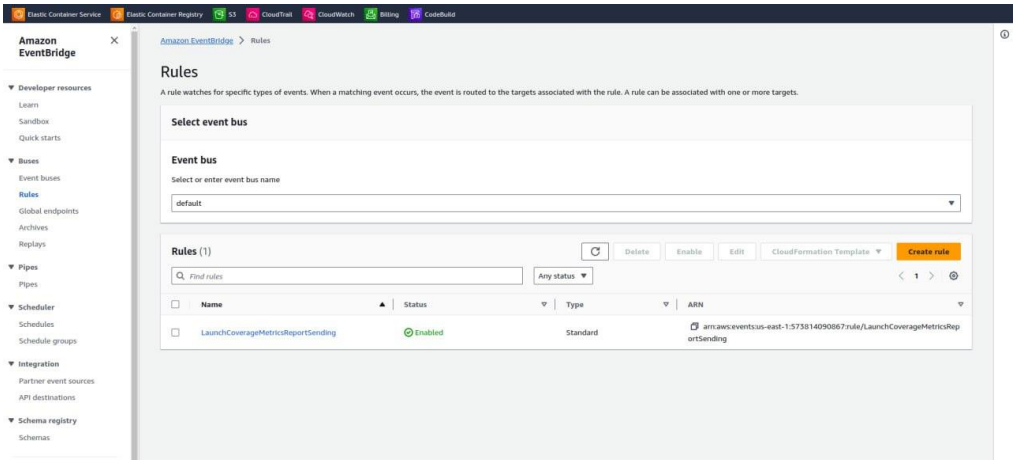


Рисунок 3.6 – Список правил у AWS EventBridge

При кліку по назві правила LaunchCoverageMetricsReportsSending можна отримати його детальну інформацію, а також доступ до його конфігурації. Дане

правило безпосередньо ініціює запуск докер контейнеру у відповідь на подію додавання файлів до сховища AWS S3. Вигляд вікна після вибору правила LaunchCoverageMetricsReportsSending подано на рисунку 3.7 нижче.

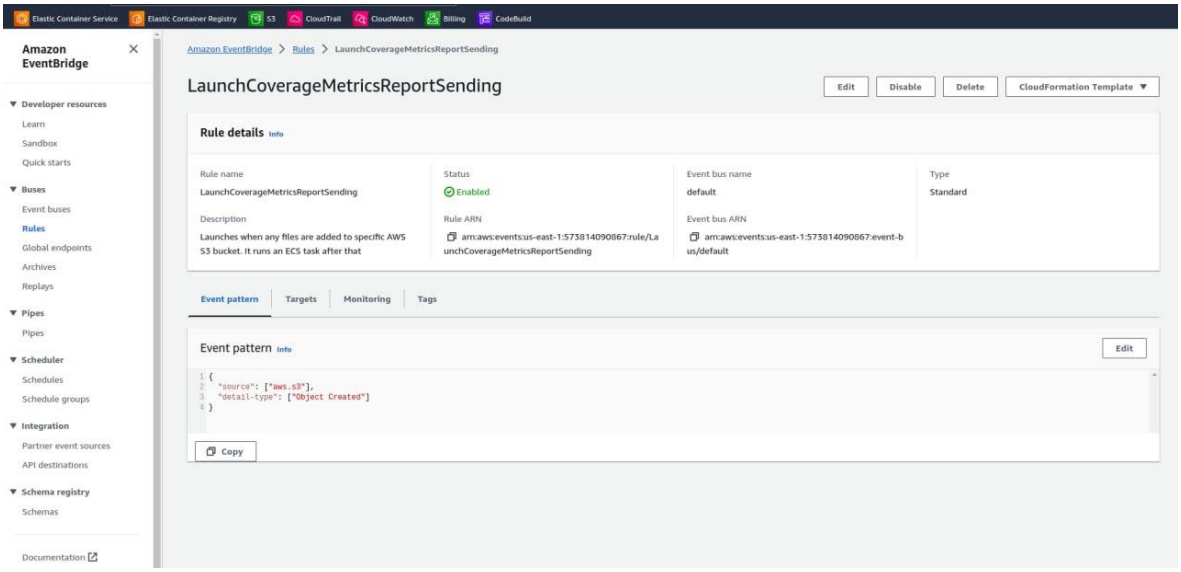


Рисунок 3.7 – Вікно обраного правила у AWS EventBridge

При кліку на кнопку Edit можна змінити конфігурацію існуючого правила. Після натискання на кнопку Edit відображається сторінка зображена на рисунку 3.8.

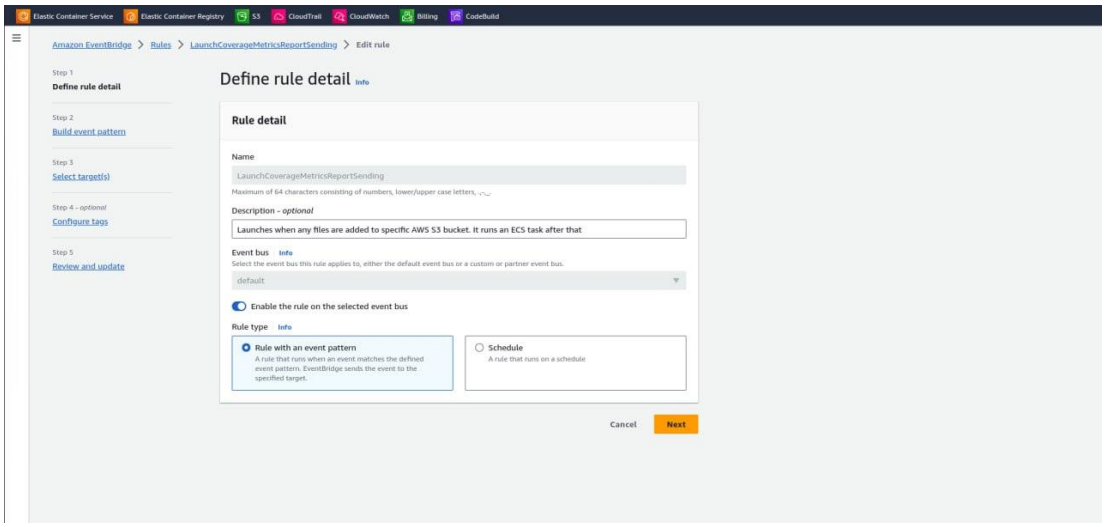


Рисунок 3.8 – Початкове вікно конфігурації обраного правила

На вищенаведеному скріншоті зображена конфігурація обраного правила AWS EventBridge. Конфігурація правила розділена на 5 секцій. Початкова секція,

що зображена вище має назву Define rule detail та використовується для вказування імені, опису та типу правила.

У наступній секції під назвою Build Event Pattern розміщуються налаштування пов’язані із створенням самої події. У даному випадку подія ініціюється у самій платформі AWS, тому у пункті Event Source обраний саме перший пункт. У пункті Event pattern містяться налаштування самої події. Тут вказується, що подія буде викликана сервісом AWS S3, категорія події, сама подія під назвою Object Created, а також вказується назва AWS S3 сховища до якого буде прикріплена дана подія.

У пункті Event pattern праворуч розміщене місце де можна вручну у форматі JSON вписати налаштування для даної події. У даному випадку, дана секція буде заповнюватись автоматично після внесення відповідних змін та вибору конкретних конфігураційних опцій. Це дає можливість гнучко налаштовувати поведінку конкретної події. Також, цей спосіб дозволяє зручно імпортувати або експортувати налаштування подій, що дозволяє ділитись налаштованою подією з іншими розробниками або зручно створювати нові події на основі існуючих конфігурацій.

Зовнішній вигляд секції Define rule detail подано на рисунках 3.9 та 3.10.

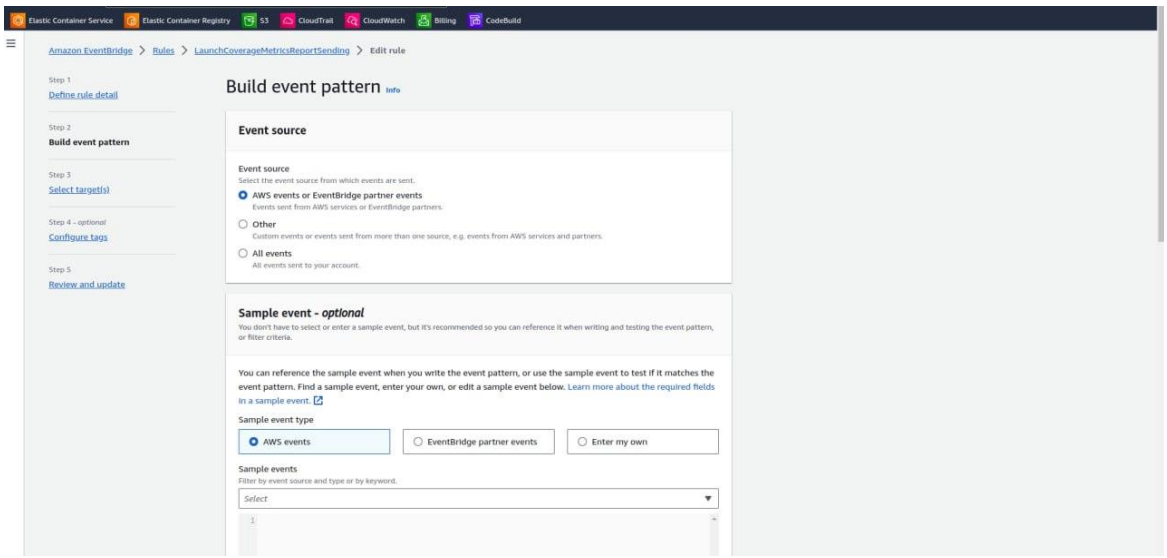


Рисунок 3.9 – Верхня частина вікна секції Build Event Pattern

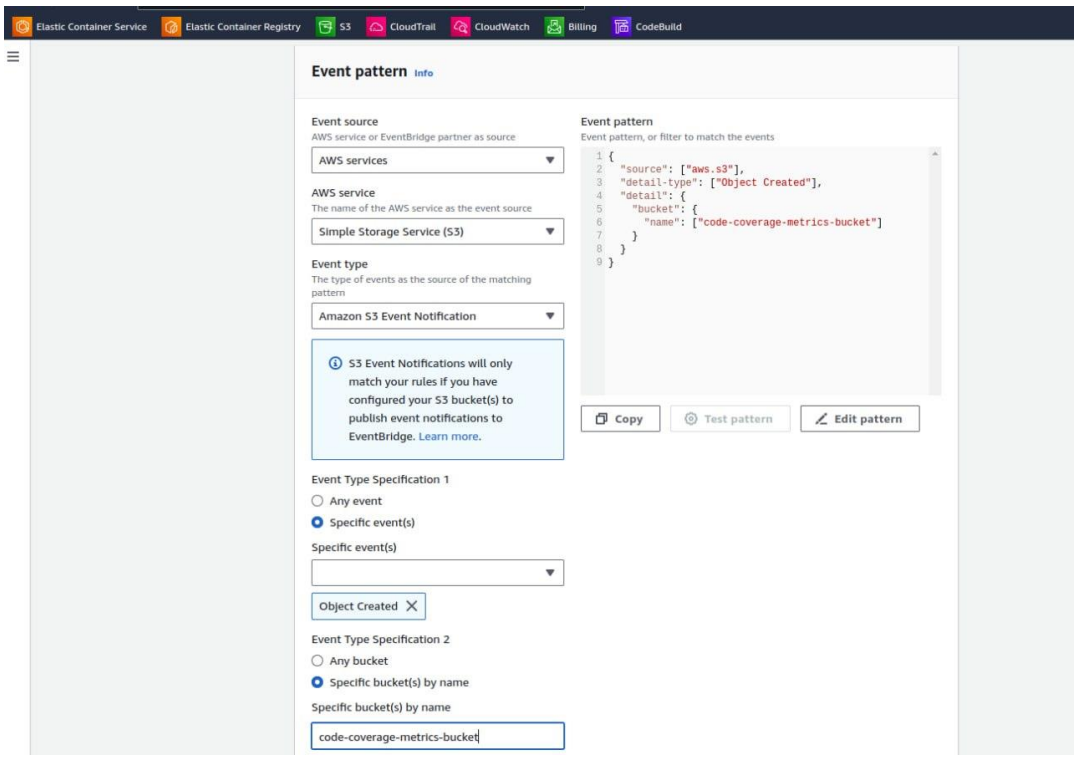


Рисунок 3.10 – Нижня частина вікна секції Build Event Pattern

Наступна секція Select target(s) містить налаштування пов’язані із тим, що має відбутись внаслідок конкретної події. В даному випадку, потрібно здійснити запуск завдання у сервісі AWS ECS, тобто запуск докер контейнеру. Тут вказується, що має бути запущене завдання сервісу ECS, назва існуючого кластеру, назва визначення завдання та його ревізія. Вікно секції Select target(s) подано на рисунку 3.11.

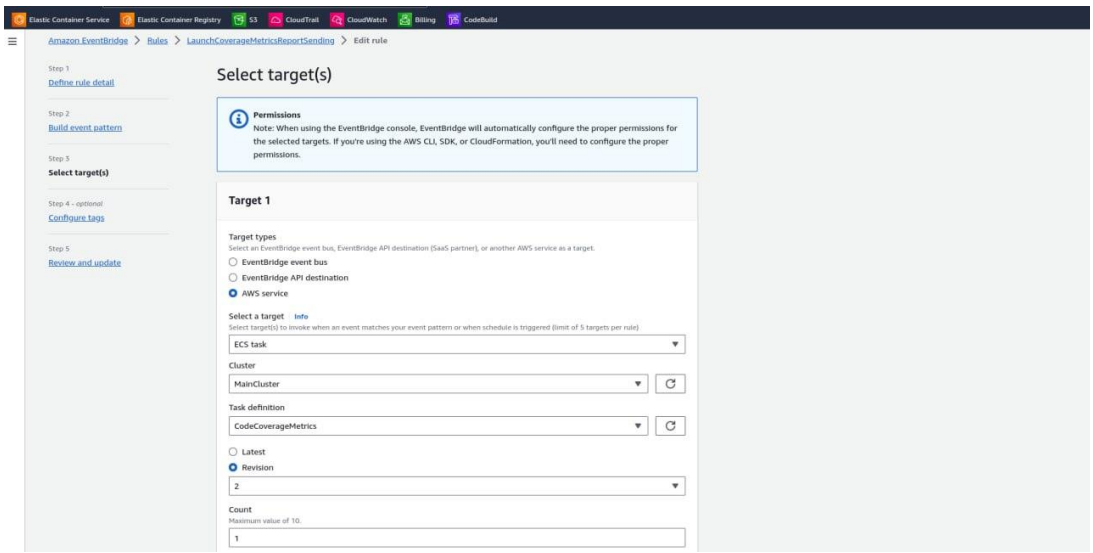


Рисунок 3.11 – Вікно секції Select target(s)

Останні дві секції, а саме Configure tags та Review and update не містять важливої інформації, тому їхній опис є не доцільним.

Далі необхідно розглянути сервіс AWS ECS, що використовується для запуску завдання, тобто докер контейнеру. Головне вікно сервісу AWS ECS містить список створених кластерів. У даному випадку був створений лише один кластер під назвою MainCluster. Вікно списку створених кластерів подано на рисунку 3.12.

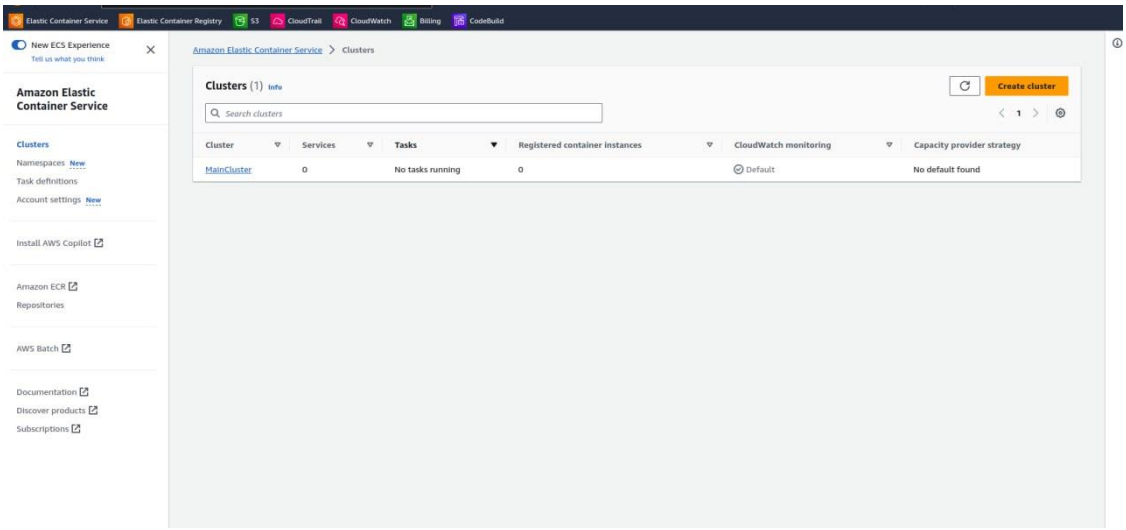


Рисунок 3.12 – Вигляд вікна списку створених кластерів у AWS ECS

Після натискання на імені кластера, буде відображене вікно з його детальною інформацією, що зображене на рисунку 3.13.

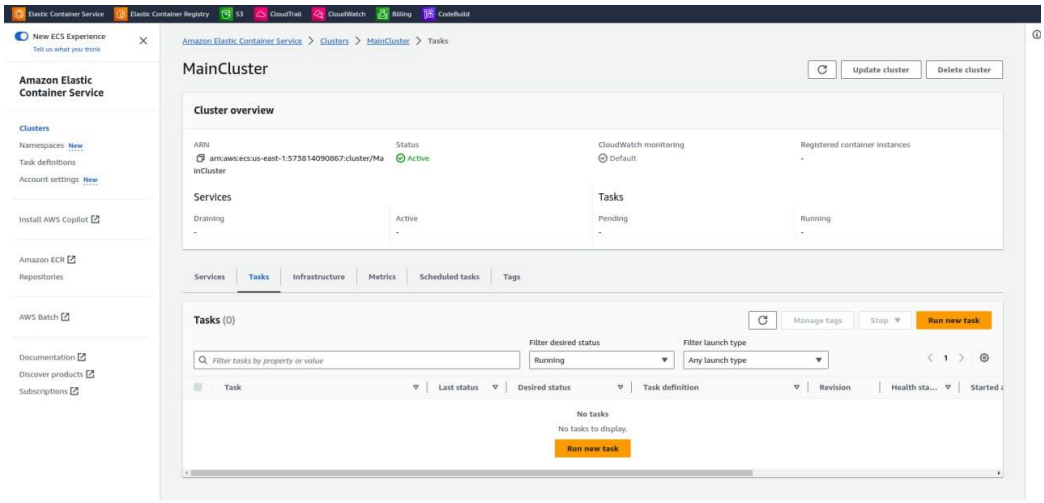


Рисунок 3.13 – Вигляд вікна списку створених кластерів у AWS ECS

З головного вікна сервісу AWS ECS ліворуч можна обрати елемент під назвою Task definitions, що призведе до переходу вікна із списком існуючих визначень завдань. Визначення завдання є своєрідним шаблоном на основі якого створюються самі завдання в AWS ECS. В даному випадку існує лише одне визначення завдання під назвою CodeCoverageMetrics, що зображене на рисунку 3.14.

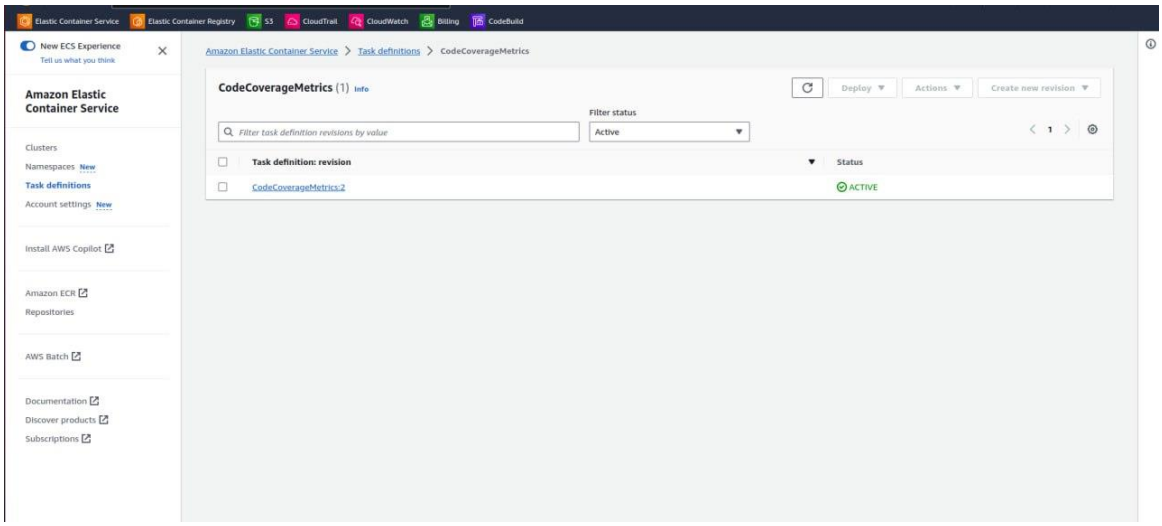


Рисунок 3.14 – Вікно списку створених визначень завдань у AWS ECS

Після натискання по назві визначення завдання відкривається вікно зі списком ревізій, тобто різних налаштувань, конкретного визначення завдання. Вікно списку ревізій даного визначення завдання показано на рисунку 3.15.

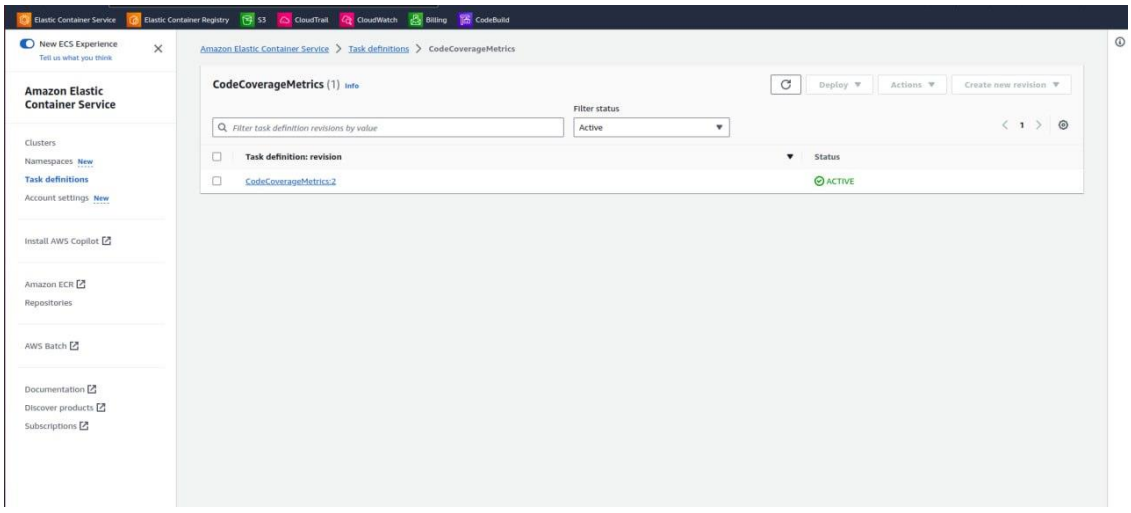


Рисунок 3.15 – Вікно списку ревізій визначення завдання

На вищенаведеному скріншоті можна побачити, що присутня лише одна ревізія під номером два.

Після вибору імені даної ревізії буде показане вікно із детальною інформацією про ревізію даного завдання. Тут розміщена інформація про обсяг процесорних ядер та оперативної пам’яті, що будуть виділені для ревізії даного завдання, що буде запущене.

Взагалі ревізія відноситься до конкретної версії або ітерації визначення завдання. Коли відбувається створення або оновлення визначення завдання, AWS призначає йому новий номер версії, починаючи з одиниці. Подальші версії того самого визначення завдання містять оновлення або зміни конфігурації. Наприклад, може бути одне визначення завдання з кількома ревізіями. Кожна ревізія може представляти іншу версію завдання, зміну вимог до ресурсів або будь-які інші налаштування конфігурації. Ревізії спрощують керування та відстеження змін у визначеннях завдань з часом. Це особливо корисно у випадках, коли існує потреба у розгортанні різних версій програми або зміні конфігурацій завдань для збереження історії цих змін.

Вигляд вікна з детальною інформацією про існуючу ревізію подано на рисунку 3.16 нижче.

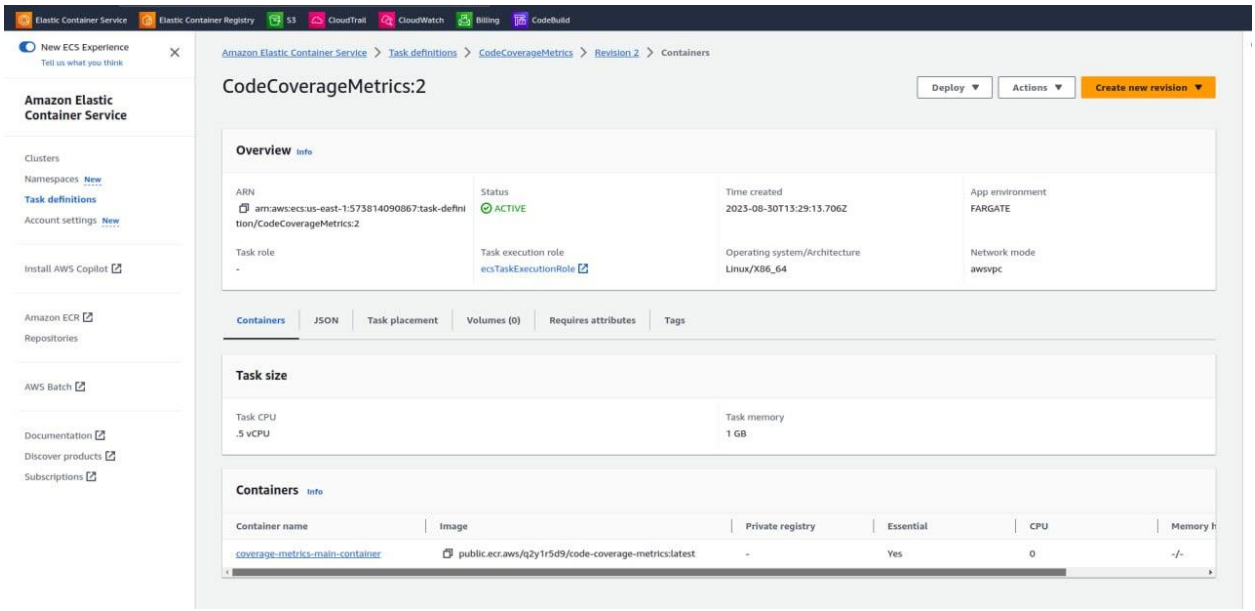


Рисунок 3.16 – Вікно ревізії визначення завдання

У вищенаведеному рисунку також присутня секція Containers, де можна побачити налаштування докер контейнеру, що буде запущений під час старту даної ревізії завдання. Після кліку по імені контейнеру coverage-metrics-main-container буде відображене спливаюче вікно зображене на рисунку 3.17.

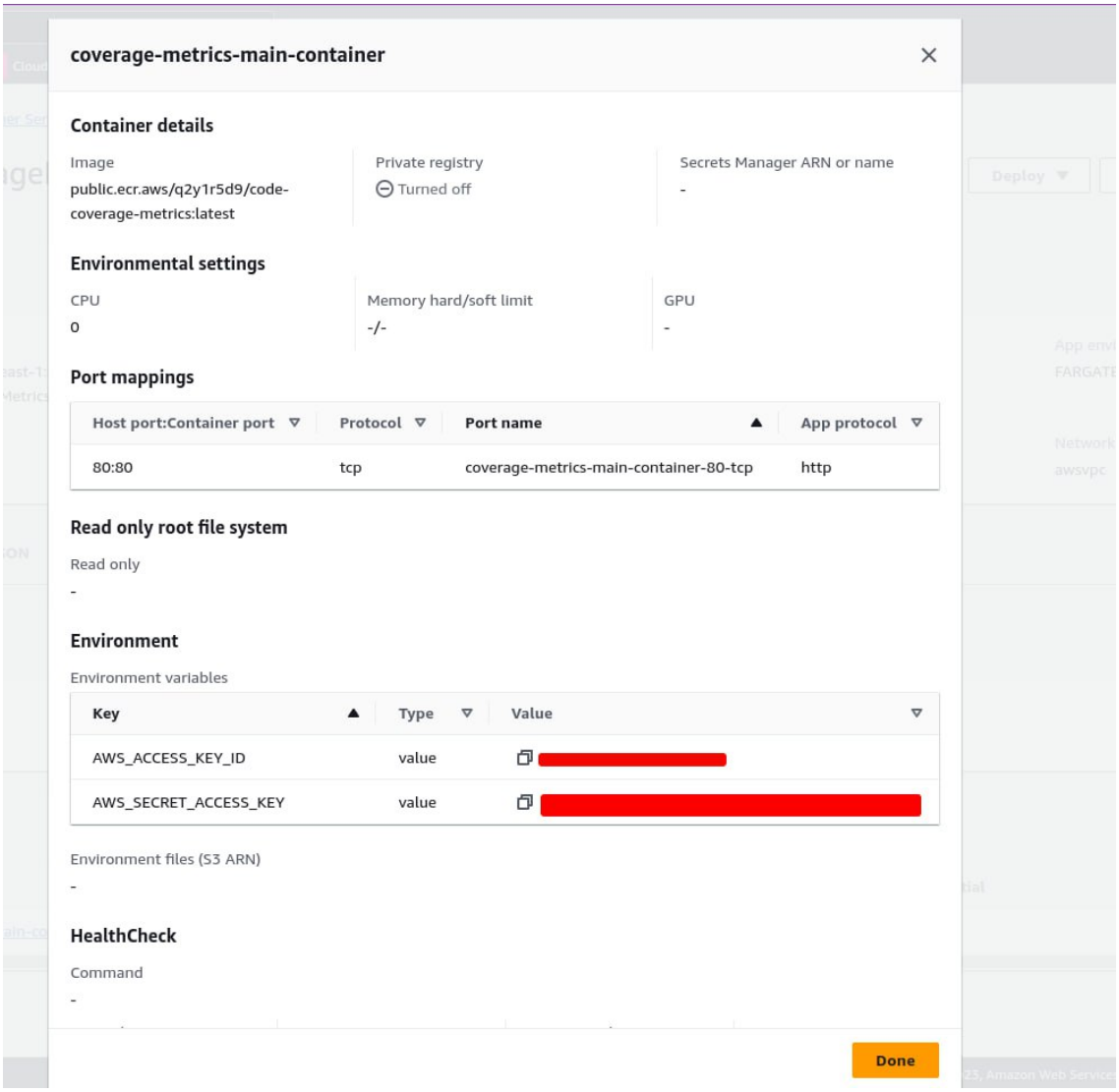


Рисунок 3.17 – Вікно налаштування докер контейнеру всередині обраної ревізії завдання в AWS ECS

Із вищенаведеного скріншоту можна побачити, що в контейнері налаштовані дві змінні середовища, а саме AWS_ACCESS_KEY_ID та AWS_SECRET_ACCESS_KEY, що будуть використані при з’єднанні із сервісом AWS S3 під час старту даного докер контейнеру. Також, можна побачити посилання

на репозиторій докер імеджу на основі якого буде запущений даний докер контейнер. Докер імедж міститься у реєстрі докер імеджів у сервісі AWS ECR.

Останній сервіс амазон, що буде розглянутий має назву AWS ECR. Він є сховищем репозиторіїв, що призначені для зберігання докер імеджів.

У даному випадку був створений публічний репозиторій під назвою code-coverage-metrics. Список створених репозиторіїв у сервісів AWS ECR зображено на рисунку 3.18.

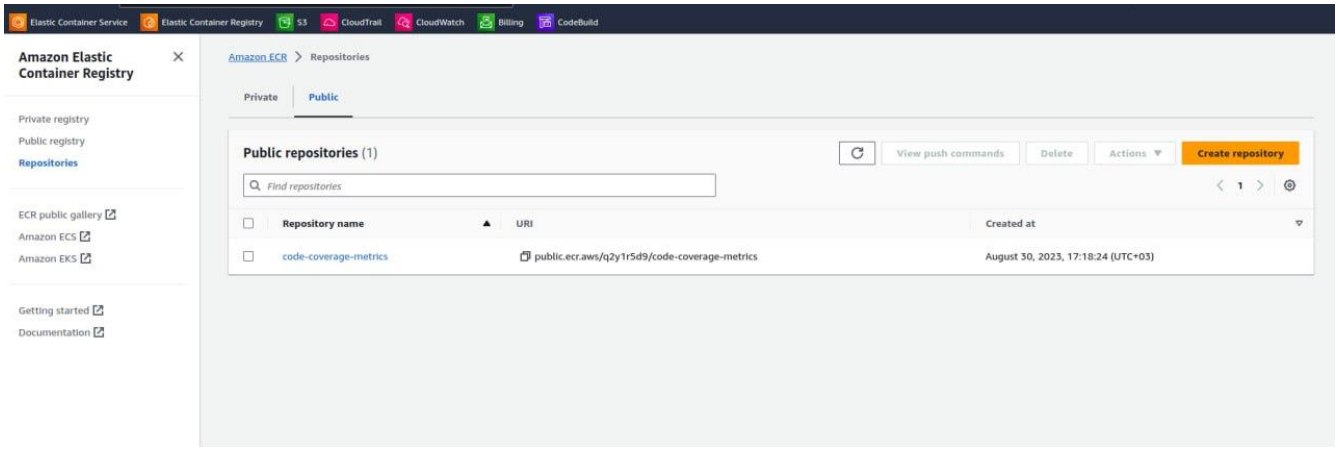


Рисунок 3.18 – Вікно списку репозиторіїв у сервісі AWS ECR

При кліку на імені репозиторію буде відкрите вікно зі списком існуючих докер імеджів. У даному випадку існує лише один докер імедж із тегом latest, який подано на рисунку 3.19.

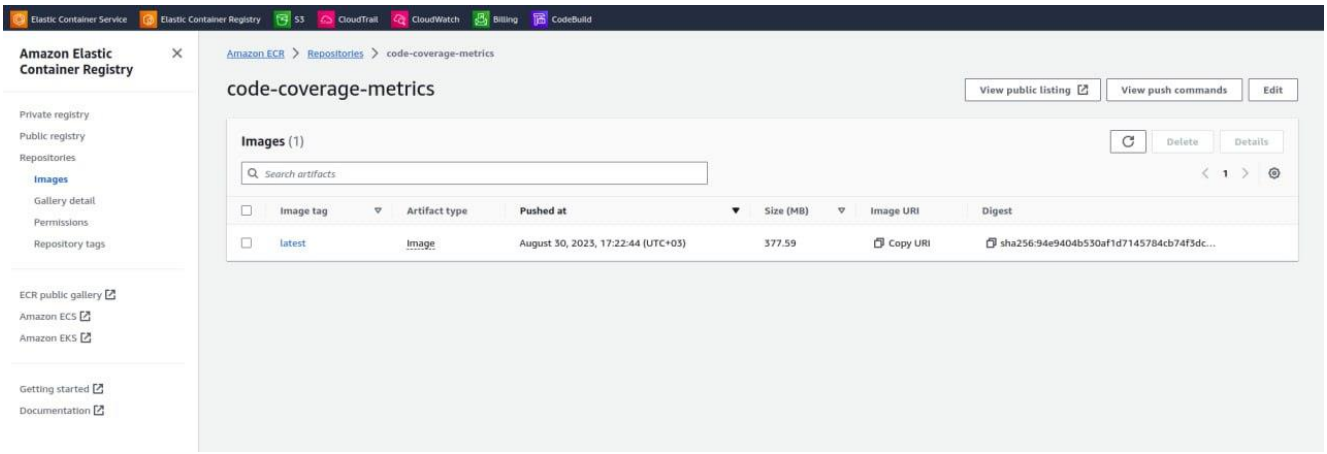


Рисунок 3.19 – Список докер імеджів всередині репозиторію AWS ECR

Перед публікацією даного докер імеджу в публічному репозиторії сервісу AWS ECR був проведений процес його створення на локальному комп'ютері. Слід більш детально розглянути основні компоненти докер імеджу, процес його роботи та створення.

Створення докер імеджу починається із файлу під назвою Dockerfile, що не має конкретного розширення. Вміст файлу Dockerfile подано нижче:

```
#-------------------------------------------------------------------------
# Base Image
#-------------------------------------------------------------------------
FROM php:8.0-fpm

#-------------------------------------------------------------------------
# Setting environment variable for PHPUnit test to generate metrics coverage
report
#-------------------------------------------------------------------------
ENV XDEBUG_MODE=coverage

#-------------------------------------------------------------------------
# Initializing working directory
#-------------------------------------------------------------------------
WORKDIR /app

#-------------------------------------------------------------------------
# Copying PHP script for sending emails inside docker container
#-------------------------------------------------------------------------
COPY email.php /app/
COPY main_script.sh /app/

#-------------------------------------------------------------------------
# Installing Composer
#-------------------------------------------------------------------------
RUN curl -sS https://getcomposer.org/installer | php -- --install-
dir=/usr/local/bin --filename=composer

#-------------------------------------------------------------------------
# Software's Installation
#-------------------------------------------------------------------------
RUN apt-get update && \
    apt-get install -y --no-install-recommends \
      curl \
      git \
      libz-dev \
      libpq-dev \
      libxml2-dev \
      unzip

#-------------------------------------------------------------------------
# Install the PHP extentions
#-------------------------------------------------------------------------
RUN docker-php-ext-install dom

#-------------------------------------------------------------------------
# Installing PHPUnit and PHPMailer libraries
#-------------------------------------------------------------------------
RUN composer require --dev phpunit/phpunit
```

```
RUN composer require phpmailer/phpmailer

#---------------------------------------------------------------------------
# Installing AWS CLI
#---------------------------------------------------------------------------
RUN curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o
"awscliv2.zip"
RUN unzip awscliv2.zip
RUN ./aws/install

#---------------------------------------------------------------------------
# Installing and enabling XDebug
#---------------------------------------------------------------------------
RUN pecl install xdebug-3.0.0
RUN docker-php-ext-enable xdebug

#---------------------------------------------------------------------------
# Launching PHP-FPM process
#---------------------------------------------------------------------------
CMD bash main_script.sh
```

Потрібно послідовно пройтись по командам у вищенаведеному Dockerfile. Даний докер файл базується на основі існуючого докер файлу під назвою php:8.0-fpm, що вказується у інструкції FROM. У наступній інструкції ENV задається змінна середовища під назвою XDEBUG_MODE із значенням coverage, яка є важливою для подальшої генерації звітів по метриках покриття коду. Інструкція WORKDIR ініціалізує робочу директорію /app всередині докер контейнеру. Директива COPY відповідно копіює два локальних файли, а саме email.php та main_script.sh у робочу директорію /app. З використанням послідовного виклику команд з використанням інструкції RUN відбувається запуск ряду команд в консолі, а саме встановлення пакетного менеджера Composer, різноманітних компонентів та розширень, встановлення фреймворку PHPUnit та бібліотеки PHPMailer з використанням Composer, також встановлення AWS консолі. Остання інструкція CMD визначає те, що буде запущене при старті докер контейнеру, що використовує даний докер імедж. У даному випадку буде запущений консольний скрипт під назвою main_script.sh, що був попередньо скопійований з локального комп’ютера всередину докер імеджу. Вміст консольного скрипту main_script.sh подано нижче:

```
#!/bin/sh
if [ -d "bucket" ]
then
  rm -r bucket
fi
```

```
aws s3 cp s3://code-coverage-metrics-bucket ./bucket --recursive
if [ -d "bucket/tests" -a -d "bucket/src" ]
then
  ./vendor/bin/phpunit ./bucket/tests/
  ./vendor/bin/phpunit ./bucket/tests/  --coverage-text=./bucket/coverage-metrics-
results.txt --coverage-filter ./bucket/src/
  php email.php
else
  echo "Error: Required directories don't exist. Email hasn't been sent!"
fi
```

У вищенаведеному консольному скрипті міститься ряд команд. Перша умовна конструкція перевіряє наявність папки bucket всередині запущеного докер контейнеру і якщо така директорія існує, то вона буде видалена. Наступна команда здійснює з'єднання із сервісом AWS S3 та стягування усіх файлів із вказаного сховища у директорію bucket даного запущеного докер контейнеру. З'єднання із сервісом AWS S3 можливе завдяки попередньо налаштованим змінним середовища AWS_ACCESS_KEY_ID та AWS_SECRET_ACCESS_KEY, що були згадані раніше. Далі слідує наступна умовна конструкція, що перевіряє наявність директорій tests та src всередині директорії bucket. Якщо ці дві директорії не існує, то буде виведене повідомлення про помилку і скрипт одразу завершить своє виконання. Якщо ж згадані директорії існують, то відбувається запуск юніт тестів та генерації звіту із результатами метрик покриття коду у окремому текстовому файлі coverage-metrics-results.txt з використанням фреймворку PHPUnit. Остання команда запускає PHP скрипт email.php, що ініціює процес формування та відправки емейл листа на поштовий клієнт Gmail. Програмний код скрипту email.php подано нижче:

```
<?php
use PHPMailer\PHPMailer\PHPMailer;
use PHPMailer\PHPMailer\SMTP;
use PHPMailer\PHPMailer\Exception;

require 'vendor/autoload.php';

$mail = new PHPMailer(true);

try {
    //Read and process .txt file filled with committer details
    $index = 0;
    $commitDetails = '';
    if (file_exists('bucket/author.txt')) {
        $commitRawData = explode("\n", file_get_contents('bucket/author.txt'));
```

```
        foreach ($commitRawData as $line) {
            $commitDetails .= $line . "<br>";
        }
    } else {
        $commitDetails = 'Committer data is empty';
    }

    //Server settings
    $mail->SMTPDebug = SMTP::DEBUG_SERVER;
    $mail->isSMTP();
    $mail->Host       = 'smtp.gmail.com';        //Set the SMTP server to send
through
    $mail->SMTPAuth   = true;
    $mail->Username   = 'your_username';
    $mail->Password   = 'your_password';
    $mail->SMTPSecure = PHPMailer::ENCRYPTION_SMTPS;
    $mail->Port       = 465;

    //Initializing recipient and sender
    $mail->setFrom('sender_email@gmail.com ', 'MailerLocal9');
    $mail->addAddress('receiver_email@gmail.com');

    //Adding attachments
    if (file_exists('bucket/coverage-metrics-results.txt')) {
        $mail->addAttachment('bucket/coverage-metrics-results.txt');
    }

    //Email content
    $mail->isHTML(true);
    $mail->Subject = 'New commit has been added with coverage metrics report';
    $mail->Body    = $commitDetails;

    $mail->send();
    echo 'Email message has been sent!';
    $mail->smtpClose();
} catch (Exception $e) {
    echo "Email message could not be sent. Mailer Error: {$mail->ErrorInfo}";
}
```

У вищенаведеному файлі на початку відбувається підключення ряду класів бібліотеки PHPMailer. Далі відбувається перевірка наявності текстового файлу author.txt всередині директорії bucket. Файл author.txt містить детальну інформацію про Git коміт. Якщо даний файл існує, то відбувається зчитування інформації даного файлу порядково та запис в окрему змінну. Далі відбувається ряд ініціалізації з використанням змінної $mail класу PHPMailer\PHPMailer\PHPMailer. При ініціалізації даної полів даної змінної вказується ряд параметрів, а саме ім’я SMTP серверу, його порт, його логін та пароль, вибір протоколу шифрування, назву та електронну пошту відправника емейл листа, електронні пошти отримувачів листа, заголовок листа, контент або тіла листа і прикріплені файли до листа за їх наявності. Відправка самого емейл листу відбувається за допомогою виклику

методу send. У разі виникнення помилки, вона буде оброблена та буде виведене повідомлення з інформацією про помилку.

Повний програмний код розробленої програмної системи подано в додатку А.

3.3 Висновки

В результаті роботи над даним розділом була спроектована чітка та детальна архітектура розроблюваної програмної системи та уточнений послідовний механізм взаємодії усіх сервісів та компонентів між собою. На основі розробленого плану архітектури був проведений опис реалізації усіх компонентів і сервісів та приведені лістинги програмних кодів ключових скриптів та файлів разом із їх детальними поясненнями. В результаті цей процес дозволив краще зрозуміти як працюють окремі компоненти та сама програмна система в цілому.

4 ОЦІНКА РЕЗУЛЬТАТІВ ТА ПІДВЕДЕННЯ ПІДСУМКІВ

4.1 Аналіз розробленого програмного рішення

В результаті роботи над кваліфікаційною робою було реалізоване робоча програмна система, що має на меті автоматизувати процес розрахунку метрик покриття коду. Програмна система використовує інтеграцію сучасних платформ та сервісів. Основними платформами виступають GitHub Actions та хмарна платформа AWS разом із деякими її окремими сервісами. GitHub Actions зв’язується із сервісом AWS S3 в результаті події додавання змін в окрему Git гілку, надсилаючи усі файли GitHub репозиторію у сховище AWS S3. У платформі AWS існує своя наперед визначена подія, що запускає завдання, тобто докер контейнер у межах сервісу AWS ECS. Запущений контейнер стягує усі існуючі файли із сховища AWS S3, запускає юніт тести та генерує звіти з метриками покриття коду з допомогою фреймворку PHPUnit та надсилає електронний лист з використанням бібліотеки PHPMailer на поштовий клієнт Gmail.

Усі сервіси працюють правильно та злагоджено, що забезпечує коректний алгоритм створення звітів метрик покриття коду та їх надсилання на адресу отримувача. Вигляд надісланого електронного листа зображено на рисунку 4.1.

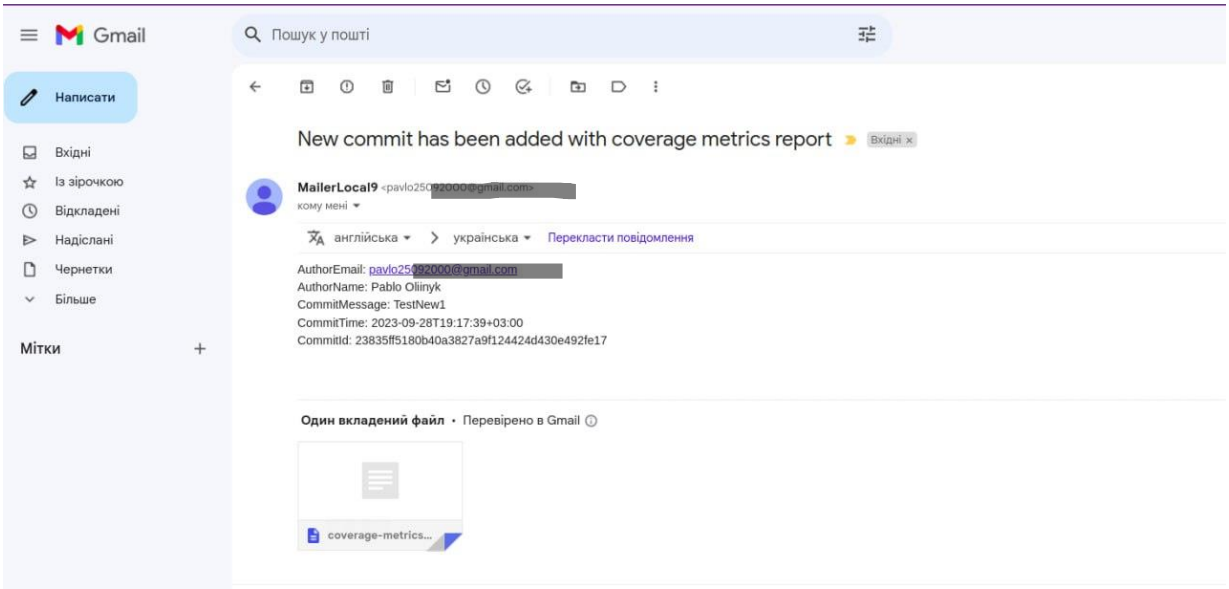


Рисунок 4.1 – Вигляд отриманого електронного листа

На рисунку можна побачити як виглядає електронний лист, що був надісланий запущеним завданням у AWS ECS. Іншими словами докер контейнер виконав формування та відправку даного листа. Лист містить заголовок, тіло з деякою інформацією, а також вкладений текстовий файл.

Тіло листа включає деяку важливу інформацію стосовно зробленого Git коміту, що був опублікований у GitHub репозиторій. Саме даний коміт запустив алгоритм програмної системи, в результаті чого був надісланий даний електронний лист. Тіло листа включає таку інформацію як:

– емейл автора коміту;

– ім’я автора коміту;

– назву коміту;

– дату створення коміту;

– унікальний ідентифікатор коміту.

Використовуючи попередньо згадані дані є можливість легко ідентифікувати, хто і коли саме створив нові зміни в GitHub репозиторії, що є доволі корисним.

Також, у листі можна побачити прикріплений файл, що безпосередньо вміщує інформацію про результаті обрахунку метрик покриття коду. Вигляд прикріпленого файлу можна побачити на рисунку 4.2 нижче.

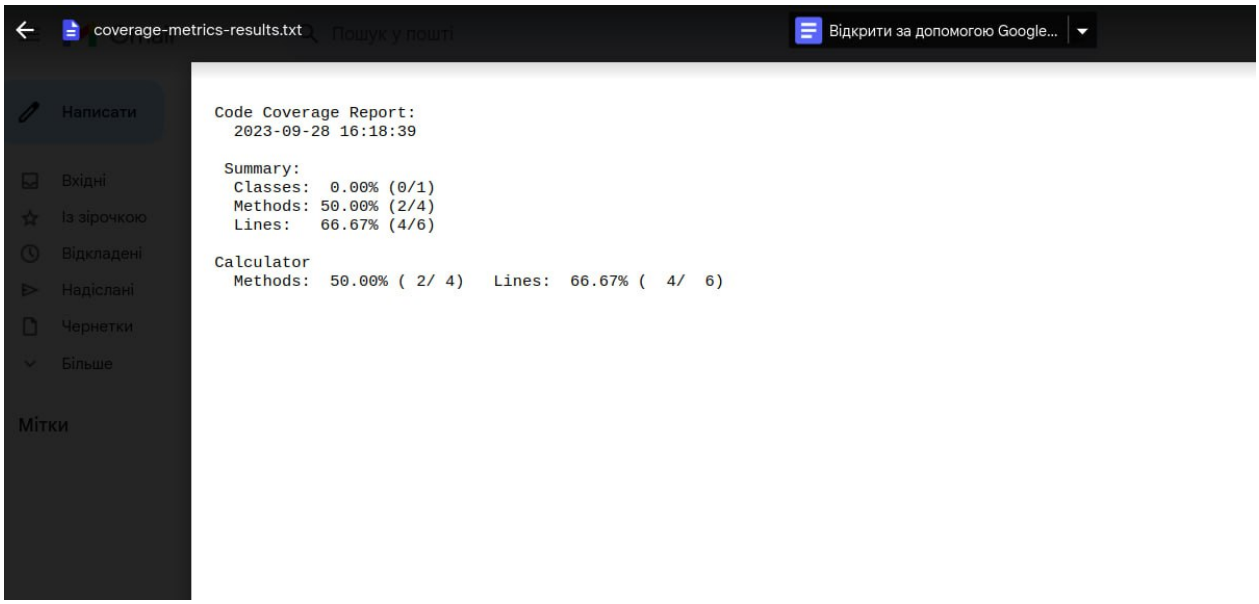


Рисунок 4.2 – Вигляд текстового файлу звіту метрик покриття коду

Даний звіт був сформований за допомогою PHP фреймворку PHPUnit. Звіт включає у собі три найбільш часто використовувані метрики покриття коду:

– метрика покриття класів;

– метрика покриття методів;

– метрика покриття рядків коду.

У даному випадку наша програма містить юніт тести написані лише на один клас під назвою Calculator, а саме на три з його чотирьох методів. Оскільки не усі методи, а отже не увесь клас був покритий тестами, то відсоткові результати метрики покриття класів та методів не є максимальними. Це можна також сказати про метрику покриття рядків коду, оскільки один метод не був покритий тестами, тому при виконанні тестів не усі рядки були пройдені.

Оскільки електронний лист був надісланий успішно разом із усією необхідною інформацією, можна стверджувати що розроблена програмна система на даний момент функціонує успішно.

Розроблювана програмна система, яка автоматизує оцінювання метрик покриття коду та результатів тестування програмного забезпечення, має потенціал стати надзвичайно корисною для розробників та команд розробки програмного забезпечення з наступних причин:

– підвищення ефективності тестування;

– зручність для розробників;

– відстеження прогресу;

– створення історії змін;

– забезпечення стабільності продукту.

Підвищення ефективності тестування. Система дозволяє автоматизувати процес виконання юніт-тестів та оцінювання покриття коду при кожній зміні в коді. Це сприяє виявленню помилок і проблем швидше, що в свою чергу значно зменшує витрати на ручне тестування.

Зручність для розробників. Керівництво проектів може легко і швидко перевіряти вплив змін розробників на покриття коду та якість коду без необхідності ручного запуску тестів. Це спрощує їхню роботу та підвищує продуктивність.

Відстеження прогресу. Присутність системи дозволяє відстежувати зміни у покритті коду та кількість юніт-тестів з часом. Це надає можливість аналізувати результати та вносити покращення.

Створення історії змін. Збереження результатів метрик та тестів дозволяє створювати історію змін якості коду, що корисно для аудиту та аналізу.

Забезпечення стабільності продукту. Автоматичне тестування та оцінювання покриття коду забезпечують стабільність та надійність розроблюваного програмного продукту, що є важливим аспектом його якості.

4.2 Виділення рекомендацій стосовно майбутніх покращень

Розроблена програмна система може бути вдосконалена різноманітним чином. Приклад програмної системи реалізований з використанням мови високого рівня PHP, але є можливість адаптації даного підходу для інших мов. Потрібно лише замінити фреймворк PHPUnit на відповідний інструмент, що дозволяє формувати юніт тести та створювати звіти із метриками покриття коду на їх основі.

Дана програмна система не є кінцевим програмним продуктом, а лише підходом, що показує можливість інтеграції сервісів між собою в результаті чого досягається автоматизація процесу генерації звітів по метрикам покриття коду, тому вона може бути надзвичайно гнучко адаптована для будь-якої існуючої мови програмування у майбутньому.

Дана програмна система потребує ряду покращень у майбутньому. Основні аспекти для подальшого вдосконалення включають такі:

– моніторинг та логування;

– забезпечення безпеки;

– тестування навантаження;

– зберігання результатів;

– резервне копіювання та відновлення;

– забезпечення оновлень і підтримки.

Моніторинг та логування. Важливо розглянути можливість внесення моніторингу та системи логування в дану систему. Це дозволить слідкувати за станом процесів та вчасно виявляти проблеми або помилки в робочому процесі.

Забезпечення безпеки. Необхідно переконатись, що всі дані, які передаються між компонентами системи, належним чином захищені та шифровані. Важливо також вивчити найкращі практики забезпечення безпеки для AWS ECS та інших інфраструктурних компонентів.

Тестування навантаження. Рекомендується провести тестування навантаження для впевненості, що програмна система здатна впоратися з великою кількістю запитів та обробляти їх ефективно.

Зберігання результатів. Потрібно переглянути можливість оптимізованого та ефективного зберігання історії результатів метрик покриття коду. Це допоможе зручно аналізувати та порівнювати поточні результати з попередніми версіями програмного забезпечення.

Резервне копіювання та відновлення. Важливо планувати резервне копіювання та можливості відновлення системи, щоб уникнути втрати даних або простою через непередбачені обставини.

Забезпечення оновлень і підтримки. Рекомендується розробити план оновлень для компонентів, бібліотек та інфраструктури системи, а також забезпечити систему підтримки та подальшого розвитку. Це допоможе забезпечити стабільну та ефективну роботу програмного забезпечення у майбутньому.

4.3 Висновки

В результаті роботи над даним розділом був приведений аналіз результатів роботи програмної системи та показане успішне надсилання електронного листа, що допомогло отримати розуміння того, що розроблена система працює успішно. Також, були приведені деякі переваги, що надає програмна система. На майбутнє

було виділено ряд рекомендацій стосовно покращень, що допоможуть покращити надійність та ефективність роботи програмної системи в цілому.

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи було вдосконалено метод для роботи з метриками покриття коду, шляхом розробки програмної системи, що має на меті автоматизувати процес генерації звітів, які містять результати обрахунку метрик покриття коду. Приклад програмної системи був реалізований із використанням мови програмування високого рівня PHP. Розроблена програмна система використовує сучасні сервіси та платформи такі як GitHub Actions та хмарну платформу AWS разом із деякими її сервісами, а також деякі інші додаткові компоненти. Також програмна система використовує основні ідеї і концепції сучасної програмної платформи контейнеризації під назвою докер.

Розроблена програмна система є доволі корисною, оскільки вона дозволяє командам розробників полегшити процес обрахунку метрик покриття коду, дозволяючи повністю автоматизувати даний процес та надає зручний спосіб сповіщення про сформовані результати цих метрик шляхом надсилання інформативних електронних листів при внесенні будь-яких змін у GitHub репозиторій. Вона забезпечує ефективне оцінювання результатів тестування програмного забезпечення, що допомагає спростити час та ресурси команд розробників, оскільки кожен програмний продукт вимагає написання великої кількості тестових сценаріїв, які повинні будуть проаналізовані шляхом формування звітів із результатами метрик покриття коду.

У першому розділі було виконане детальне дослідження предметної області в якому була наведена основна термінологія, що є ключовою для розуміння під час роботи над кваліфікаційною роботою. Проведено дослідження існуючих програмних рішень, що надають можливості обрахунку метрик покриття коду. Під час аналізу були виділені їх недоліки та запропонована необхідність у розробці власної універсальної програмної системи, що має на меті спростити процес обрахунку метрик покриття коду на основі результатів тестування.

У другому розділі був здійснений детальний аналіз сучасних технологій та приведено пояснення та способи використання кожної з них. Це дозволило

визначити роль та основні функції кожного сервісу у межах розроблюваної програмної системи.

У третьому розділі була приведена чітка схема архітектури програмної системи, що дозволило якісно оцінити та краще зрозуміти алгоритм взаємодії усіх сервісів та платформ, що використовуються при розробці програмної системи. В результаті отриманої архітектури була проведена реалізація програмної системи під час якої були надані детальні описи ключових скриптів та файлів із приведенням їх програмних кодів.

У четвертому розділі були підведені підсумки щодо отриманих результатів. Було наведено кінцеві результати роботи програмної системи, а саме показаний надісланий електронний лист, що містить усю необхідну інформацію, включаючи звіт із результатами обрахунку метрик покриття коду. Також, додатково було наведено ряд запропонованих покращень, які можуть бути внесені при продовженні роботи над програмною системою у майбутньому.

Оскільки в ході роботи над кваліфікаційною роботою була пояснена та розроблена працездатна програмна система, що дозволяє ефективно автоматизувати процес розрахунку метрик покриття коду на основі результатів тестування, то можна стверджувати, що основна мета, яка була поставлена у вступі є в повній мірі досягнута.

Розроблена програмна система не позиціонується як готовий програмний продукт. Вона виступає в ролі загального підходу, який показує можливість інтеграції існуючих сервісів та платформ. Одною з головних переваг програмної системи є те, що вона доволі гнучка та може бути адаптована для будь-якої мови програмування, а також легко масштабована для внесення різноманітних покращень та вдосконалень у майбутньому.

Наукові та практичні результати, які були отримані в ході кваліфікаційної роботи мають важливий потенціал для подальшого наукового та практичного використання. Результати даної роботи можуть бути використані в реальних проектах розробки програмного забезпечення. Розроблена система для автоматизації оцінювання метрик покриття коду та результатів тестування може

бути впроваджена в робочі процеси розробки програмних продуктів для покращення їх якості та надійності. Використовуючи розроблену систему, компанії та команди розробників можуть оптимізувати процес тестування, зменшити час, витрати та зусилля, що необхідні для перевірки якості програмного забезпечення.

Загалом, результати кваліфікаційної роботи відкривають широкі можливості для подальшого використання у наукових та практичних аспектах розробки програмного забезпечення, сприяючи покращенню якості та надійності програмних продуктів шляхом автоматизації розрахунку метрик покриття коду на основі готових тестових сценаріїв.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Ammann P., Offutt J. Introduction to Software Testing. Cambridgeshire: Cambridge University Press, 2013. 364 p.

2. What is Software Testing? URL: https://www.geeksforgeeks.org/software-testing-basics/ (дата звернення: 10.09.2023).

3. What is Software Testing: Definition, Types and Best Practices. URL: https://www.browserstack.com/guide/what-is-software-testing (дата звернення: 11.09.2023).

4. Myers G. J. The Art of Software Testing, 2nd edition. New Jersey: Wiley, 2004. 256 p.

5. 7 Principles of Software Testing. URL: https://www.interviewbit.com/blog/principles-of-software-testing/ (дата звернення: 13.09.2023).

6. Beizer B. Software testing techniques, 2nd edition. New York: Van Nostrand Reinhold, 2009. 550 p.

7. Khorikov V. Unit Testing: Principles, Practices, and Patterns. New York: Manning, 2020. 304 p.

8. Introduction To Software Testing. URL: https://www.test-institute.org/Introduction_To_Software_Testing.php (дата звернення: 13.09.2023).

9. What is Software Testing? All you need to know about Methods and Testing. URL: https://www.edureka.co/blog/what-is-software-testing/ (дата звернення: 14.09.2023).

10. Fundamentals of Software Testing: Concepts and Process.URL: https://www.simplilearn.com/tutorials/devops-tutorial/fundamentals-of-software-testing (дата звернення: 15.09.2023).

11. The different types of software testing. URL: https://www.atlassian.com/continuous-delivery/software-testing/types-of-software-testing (дата звернення: 15.09.2023).

12. Ultimate Guide to the Different Types of Software Testing in 2023. URL: https://hackr.io/blog/types-of-software-testing (дата звернення: 15.09.2023).

13. Types of Software Testing Explained. URL: https://www.alexhyett.com/types-of-testing/ (дата звернення: 16.09.2023).

14. Nicolette D. Software Development Metrics First Edition. New York: Manning, 2015. 192 p.

15. Everything you need to know about code coverage. URL: https://www.codegrip.tech/productivity/everything-you-need-to-know-about-code-coverage/ (дата звернення: 16.09.2023).

16. Understanding Code Coverage Metrics: A Simple Guide. URL: https://medium.com/@engrabdullahabdullah/understanding-code-coverage-metrics-a-simple-guide-dc0751102009 (дата звернення: 16.09.2023).

17. What are code coverage metrics? URL: https://buildpulse.io/blog/what-are-code-coverage-metrics (дата звернення: 17.09.2023).

18. Code coverage of Manual Testing using Istanbul. URL: https://vijayt.com/post/code-coverage-of-manual-testing-using-istanbul/ (дата звернення: 17.09.2023).

19. How to Generate Code Coverage Report with JaCoCo in Java Application? URL: https://www.geeksforgeeks.org/how-to-generate-code-coverage-report-with-jacoco-in-java-application/ (дата звернення: 18.09.2023).

20. What you need to know: Code Coverage with JaCoCo. URL: https://www.blueacornici.com/blog/code-coverage-with-jacoco (дата звернення: 19.09.2023).

21. Introduction to Cobertura. URL: https://www.baeldung.com/cobertura (дата звернення: 19.09.2023).

22. Vaswani V. Php: A Beginner's Guide. New York: McGraw Hill, 2008. 478 p.

23. Chan J. PHP: Learn PHP in One Day and Learn It Well. Chicago: Independently published, 2020. 246 p.

24. Bierer D. PHP 8 Programming Tips, Tricks and Best Practices: A practical guide to PHP 8 features, usage changes, and advanced programming techniques. Birmingham: Packt Publishing, 2021. 528 p.

25. Cybellium Ltd, Mastering PHP: A Comprehensive Guide To Learn PHP Programming. Chicago: Independently published, 2023. 276 p.

26. Lockhart J. Modern PHP: New Features and Good Practices. California: O'Reilly Media, 2015. 268 p.

27. Vanamala S. Learn PHP: Basics of PHP Language, 2020. 212 p.

28. Ponuthorai P., Loeliger J. Version Control with Git: Powerful Tools and Techniques for Collaborative Software Development 3rd Edition. California: O'Reilly Media, 2022. 546 p.

29. Jana A. GIT AND GITHUB : BEGINNER TO ADVANCE. Chicago: Independently published, 2023. 39 p.

30. Kaufmann M. Accelerate DevOps with GitHub: Enhance software delivery performance with GitHub Issues, Projects, Actions, and Advanced Security. Birmingham: Packt Publishing, 2022. 540 p.

31. Heller P. Automating Workflows with GitHub Actions: Automate software development workflows and seamlessly deploy your applications using GitHub Actions. Birmingham: Packt Publishing, 2021. 216 p.

32. Turnbull J. The Docker Book: Containerization is the new virtualization. Navi Mumbai: Shroff Publishers, 2014. 400 p.

33. Nickoloff J., Kuenzli S. Docker in Action, Second Edition 2nd Edition. New York: Manning, 2019. 350 p.

34. Chelladhurai J. S. Vinod Singh, Pethuru Raj, Learning Docker - Second Edition: Build, ship, and scale faster 2nd Revised edition. Birmingham: Packt Publishing, 2017. 300 p.

35. King T. H. AWS: The Ultimate Guide From Beginners To Advanced For The Amazon Web Services (2020 Edition). Chicago: Independently published, 2019. 197 p.

36. Ifrah. S. Deploy Containers on AWS: With EC2, ECS, and EKS 1st ed. Edition. New York: Apress, 2019. 384 p.

37. Wong G. Amazon S3 Programming Guide: Beginner’s guide book on how to get started with Amazon Simple Storage Service. California: CreateSpace Independent Publishing Platform, 2016. 90 p.

38. Bergmann S. PHPUnit Pocket Guide: Test-Driven Development in PHP. California: O'Reilly Media, 2005. 87 p.

39. How to Test PHP Code With PHPUnit. URL: https://www.freecodecamp.org/news/test-php-code-with-phpunit/ (дата звернення: 24.09.2023).

40. PHPUNIT Test Coverage Report. URL: https://engineering.teknasyon.com/phpunit-test-coverage-report-8863563f34bd (дата звернення: 24.09.2023).

41. Understanding PHPUnit and How to write Unit test cases. URL: https://www.valuebound.com/resources/blog/understanding-phpunit-and-how-to-write-unit-test-cases (дата звернення: 26.09.2023).

42. PHPMailer Guide. URL: https://mailtrap.io/blog/phpmailer/ (дата звернення: 27.09.2023).

43. How To Use PHPmailer In PHP. URL: https://robots.net/tech/how-to-use-phpmailer-in-php/#google_vignette (дата звернення: 28.09.2023).

ДОДАТОК А

(обов’язковий)

ПРОГРАМНИЙ КОД

А. 1 – Програмний код файлу main.php

```
<?php

require_once 'src/Calculator.php';

$calculator = new Calculator();

print $calculator->addition(6, 5);
```

А. 2 – Програмний код файлу Calculator.php

```
<?php
declare(strict_types=1);

class Calculator
{
    public function addition(int $arg1, int $arg2)
    {
        return $arg1 + $arg2;
    }

    public function subtraction(int $arg1, int $arg2)
    {
        return $arg1 - $arg2;
    }

    public function division(int $arg1, int $arg2)
    {
        if ($arg2 === 0) {
            throw new Exception('Exception division by zero!');
        }
        return $arg1 / $arg2;
    }
    public function multiplication(int $arg1, int $arg2)
    {
        return $arg1 * $arg2;
    }
}
```

А. 3 – Програмний код файлу CalculatorTest.php

```
<?php
```

```
declare(strict_types=1);

require_once __DIR__ . '/../src/Calculator.php';

class CalculatorTest extends \PHPUnit\Framework\TestCase
{
    public function testAddition()
    {
        $calculator = new Calculator();

        $result = $calculator->addition(3, 2);

        $this->assertSame(5, $result);
    }

    public function testSubtraction()
    {
        $calculator = new Calculator();

        $result = $calculator->subtraction(3, 2);

        $this->assertSame(1, $result);
    }

    public function testDivision()
    {
        $calculator = new Calculator();

        $result = $calculator->division(14, 2);

        $this->assertSame(7, $result);
    }
}
```

А. 4 – Програмний код файлу aws-s3-upload.yml

```
name: coverage-metrics-aws-s3

on:
  push:
    branches: [ "develop" ]

jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout to repository
        uses: actions/checkout@v3

      - name: Create .txt file filled with committer details
        run: |
          echo AuthorEmail: ${{ github.event.head_commit.author.email }}  >
author.txt
          echo AuthorName: ${{ github.event.head_commit.author.name }} >>
author.txt
          echo CommitMessage: ${{ github.event.head_commit.message }}  >>
author.txt
          echo CommitTime: ${{ github.event.head_commit.timestamp }} >> author.txt
          echo CommitId: ${{ github.event.head_commit.id }}  >> author.txt
```

```
      - name: Configure AWS Credentials
        uses: aws-actions/configure-aws-credentials@v1
        with:
          aws-access-key-id: ${{ secrets.AWS_ACCESS_KEY }}
          aws-secret-access-key: ${{ secrets.AWS_SECRET_ACCESS_KEY }}
          aws-region: us-east-1

      - name: Deploy repository files to S3 bucket
        run: aws s3 sync . s3://${{ secrets.AWS_BUCKET }} -delete
```

А. 5 – Програмний код файлу Dockerfile

```
#-------------------------------------------------------------------------
# Base Image
#-------------------------------------------------------------------------
FROM php:8.0-fpm

#-------------------------------------------------------------------------
# Setting environment variable for PHPUnit test to generate metrics coverage
report
#-------------------------------------------------------------------------
ENV XDEBUG_MODE=coverage

ENV AWS_ACCESS_KEY_ID=AKIAYLGP4FRZYFNKXZSV
ENV AWS_SECRET_ACCESS_KEY=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

#-------------------------------------------------------------------------
# Initializing working directory
#-------------------------------------------------------------------------
WORKDIR /app

#-------------------------------------------------------------------------
# Copying PHP script for sending emails inside docker container
#-------------------------------------------------------------------------
COPY email.php /app/
COPY main_script.sh /app/

#-------------------------------------------------------------------------
# Installing Composer
#-------------------------------------------------------------------------
RUN curl -sS https://getcomposer.org/installer | php -- --install-
dir=/usr/local/bin --filename=composer

#-------------------------------------------------------------------------
# Software's Installation
#-------------------------------------------------------------------------
RUN apt-get update && \
    apt-get install -y --no-install-recommends \
      curl \
      git \
      libz-dev \
      libpq-dev \
      libxml2-dev \
      unzip

#-------------------------------------------------------------------------
# Install the PHP extentions
#-------------------------------------------------------------------------
RUN docker-php-ext-install dom
```

```
#-----------------------------------------------------------------------------
# Installing PHPUnit and PHPMailer libraries
#-----------------------------------------------------------------------------
RUN composer require --dev phpunit/phpunit
RUN composer require phpmailer/phpmailer

#-----------------------------------------------------------------------------
# Installing AWS CLI
#-----------------------------------------------------------------------------
RUN curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o
"awscliv2.zip"
RUN unzip awscliv2.zip
RUN ./aws/install

#-----------------------------------------------------------------------------
# Installing and enabling XDebug
#-----------------------------------------------------------------------------
RUN pecl install xdebug-3.0.0
RUN docker-php-ext-enable xdebug

#-----------------------------------------------------------------------------
# Launching PHP-FPM process
#-----------------------------------------------------------------------------
CMD bash main_script.sh
```

А. 6 – Програмний код файлу email.php

```
<?php
use PHPMailer\PHPMailer\PHPMailer;
use PHPMailer\PHPMailer\SMTP;
use PHPMailer\PHPMailer\Exception;

require 'vendor/autoload.php';

$mail = new PHPMailer(true);

try {
    //Read and process .txt file filled with committer details
    $index = 0;
    $commitDetails = '';
    if (file_exists('bucket/author.txt')) {
        $commitRawData = explode("\n", file_get_contents('bucket/author.txt'));
        foreach ($commitRawData as $line) {
            $commitDetails .= $line . "<br>";
        }
    } else {
        $commitDetails = 'Committer data is empty';
    }

    //Server settings
    $mail->SMTPDebug = SMTP::DEBUG_SERVER;
    $mail->isSMTP();
    $mail->Host       = 'smtp.gmail.com';        //Set the SMTP server to send
through
    $mail->SMTPAuth   = true;
    $mail->Username   = 'pavlo25092000@gmail.com';
    $mail->Password   = 'kepeysccwfgwobeg';
    $mail->SMTPSecure = PHPMailer::ENCRYPTION_SMTPS;
    $mail->Port       = 465;
```

```
    //Initializing recipient and sender
    $mail->setFrom('fromTest34@example.com', 'MailerLocal9');
    $mail->addAddress('edifiei2000@gmail.com');

    //Adding attachments
    if (file_exists('bucket/coverage-metrics-results.txt')) {
        $mail->addAttachment('bucket/coverage-metrics-results.txt');
    }

    //Email content
    $mail->isHTML(true);
    $mail->Subject = 'New commit has been added with coverage metrics report';
    $mail->Body    = $commitDetails;

    $mail->send();
    echo 'Email message has been sent!';
    $mail->smtpClose();
} catch (Exception $e) {
    echo "Email message could not be sent. Mailer Error: {$mail->ErrorInfo}";
}
```

А. 7 – Програмний код файлу main_script.sh

```
#!/bin/sh
if [ -d "bucket" ]
then
  rm -r bucket
fi
aws s3 cp s3://code-coverage-metrics-bucket ./bucket --recursive
if [ -d "bucket/tests" -a -d "bucket/src" ]
then
  ./vendor/bin/phpunit ./bucket/tests/
  ./vendor/bin/phpunit ./bucket/tests/  --coverage-text=./bucket/coverage-metrics-
results.txt --coverage-filter ./bucket/src/
  php email.php
else
  echo "Error: Required directories don't exist. Email hasn't been sent!"
fi
```

ДОДАТОК Б

(обов’язковий)

КОПІЇ НАУКОВИХ ПУБЛІКАЦІЙ

Міжнародний науково-технічний журнал
«Вимірювальна та обчислювальна техніка в технологічних процесах»
ISSN 2219-9365

https://doi.org/10.31891/2219-9365-2023-75-16
УДК 004.054

ОЛІЙНИК Павло
Хмельницький національний університет
pavlo25092000@gmail.com
МАРТИНЮК Валерій
Хмельницький національний університет
https://orcid.org/0000-0001-5758-4244
e-mail: martynyuk.valeriy@gmail.com

УДОСКОНАЛЕНИЙ МЕТОД РОБОТИ З МЕТРИКАМИ ПОКРИТТЯ КОДУ ДЛЯ ЗАБЕЗПЕЧЕННЯ ЕФЕКТИВНОГО ОЦІНЮВАННЯ РЕЗУЛЬТАТІВ ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Стаття присвячена питанню вимірювання та аналізу тестового покриття коду в контексті оцінювання якості програмного забезпечення. У ній розглядається удосконалений метод роботи з метриками покриття коду, що забезпечує більш ефективне та автоматизоване оцінювання результатів тестування. Наукова стаття пропонує вдосконалений метод оцінювання метрик покриття коду як ключового аспекту процесу тестування програмного забезпечення. Розглянутій програмній системі доручено автоматизувати цей процес та підвищити ефективність оцінювання результатів тестування. Надано детальний опис методології, включаючи архітектуру програмної системи, інфраструктуру та процес збору даних про покриття коду та виконання тестів.

Ключові слова: метрики покриття коду, PHP, PHPUnit, PHPMailer.Git, GitHub, GitHub Actions, Docker, Composer, AWS, AWS ECS, AWS ECR.

OLIINYK Pavlo, MARTYNYUK Valeriy
Khmelnytskyi National University

AN ADVANCED METHOD FOR WORKING WITH CODE COVERAGE METRICS TO ENSURE EFFECTIVE EVALUATION OF SOFTWARE TESTING RESULTS

In the rapidly evolving landscape of software development, effective code coverage metrics and testing evaluation are paramount for ensuring the quality and reliability of software products. This research article introduces an innovative approach to code coverage assessment, designed to streamline the testing process and provide a robust framework for evaluating test results. The article is dedicated to the issue of measuring and analyzing code coverage in the context of software quality assessment. It considers an improved method of working with code coverage metrics that provides more effective and automated testing results evaluation. The article also discusses possible practical applications of this method and its potential advantages for software engineering. The proposed methodology is meticulously described, providing a detailed breakdown of the software system's architecture, the underlying infrastructure, and the data collection procedures for both code coverage and test execution. A central feature of the system is its seamless integration into the software development pipeline, ensuring that code coverage and testing are tightly woven into the fabric of the software development lifecycle. Notably, the system significantly reduces the time and effort required for testing, addressing a critical need in the context of agile and rapid software development environments. In conclusion, this article underscores the imperative of automating the code coverage assessment process and its transformative impact on software quality. It serves as a valuable resource for software engineers, developers, and researchers seeking to optimize their testing practices and elevate the quality of their software products. By presenting a comprehensive and innovative approach to code coverage evaluation, this research contributes to the ongoing quest for improved software reliability and resilience in the dynamic landscape of modern software development.

Keywords: code coverage metrics, PHP, PHPUnit, PHPMailer.Git, GitHub, GitHub Actions, Docker, Composer, AWS, AWS ECS, AWS ECR.

Постановка проблеми у загальному вигляді та її зв'язок із важливими науковими чи практичними завданнями

Фундаментальною проблемою є потреба в удосконаленому та більш ефективному методі оцінки покриття коду та результатів тестування для забезпечення якості та надійності програмного забезпечення. Забезпечення якості програмного забезпечення стає необхідним для бізнесу та організацій, щоб надавати надійні та стійкі рішення. Важливим аспектом забезпечення якості програмного забезпечення є тестування програмного забезпечення, а покриття тестами є важливою метрикою, яка оцінює якість та обсяг тестування.

Ця стаття намагається розглянути цю проблему та пропонує удосконалений метод для роботи з метриками покриття тестами, який автоматизує процес розрахунку метрик покриття коду на основі виконаних юніт тестів у програмному продукті.

Аналіз останніх досліджень та публікацій

Основою дослідження стали праці різноманітних дослідників, які включають в себе інформацію про тестування програмного забезпечення, метрик покриття коду, пояснення основних понять GitHub Actions та

Міжнародний науково-технічний журнал
«Вимірювальна та обчислювальна техніка в технологічних процесах»
ISSN 2219-9365

AWS ECS. Наприклад, в [1], вивчаються основні методи тестування та техніки, які використовуються в цій області. Книга містить вичерпний вступ до питання тестування ПЗ, що робить її незамінною для цього дослідження. Інший важливий ресурс, "The Art of Software Testing" [2], детально розглядає різні стратегії тестування та надає велику кількість інформації про різні методики та практики в області тестування. "Software Testing Techniques" [3] є ще одним важливим ресурсом, який досліджує різні техніки тестування програмного забезпечення та забезпечує глибокий аналіз цих методів. Метрики розробки програмного забезпечення висвітлені в роботі "Software Development Metrics" [4]. Ця книга розглядає метрики як інструмент для вимірювання та управління якістю ПЗ. У книзі "Learning GitHub Actions" [5] приводиться детальний опис головних компонентів, що мають на меті автоматизувати процеси розробки програмного забезпечення за допомогою GitHub Actions. Книга "Deploy Containers on AWS: With EC2, ECS, and EKS" [6] вивчає як правильно розгортати та керувати контейнерами за допомогою Docker на Amazon ECS. Ці джерела дали важливу базу для розуміння контексту та викликів, пов'язаних з використанням метрик покриття коду для оцінки якості ПЗ. Проте, у досліджуваній літературі відсутній детальний аналіз або універсальні методи використання метрик покриття коду для оцінки ефективності тестування ПЗ, тому дослідження цього питання є актуальним і має важливе значення для області тестування ПЗ.

Формулювання цілей статті

Цілями даної статті є:

1. Визначення основних понять, що будуть використовуватись у даній статті, з приведенням їх опису у стислій формі.
2. Приведення опису удосконаленого методу роботи з метриками тестового покриття коду, який дозволяє ефективно оцінювати результати тестування програмного забезпечення.
3. Визначення важливості та корисності впровадження цього методу в практику тестування ПЗ.
4. Виявлення усіх можливих покращень у майбутньому, які можуть бути зроблені задля вдосконалення цього методу.

Виклад основного матеріалу

Сьогодні, задача тестування програмного забезпечення є надзвичайно поширеною, оскільки кожного дня у нашому світі розробляється велика кількість програмного забезпечення, яке додатково проходить ряд тестувань. В загальному сенсі, тестування є методом перевірки того, чи фактичний програмний продукт відповідає очікуваним вимогам та чи він позбавлений різноманітних дефектів. У межах тестування програмного забезпечення є поширеним використання спеціальних числових показників, які отримали назву метрики. У метриках програмного забезпечення виділяють специфічну категорію метрик, що відносяться до тестування програмних продуктів, які отримали назву метрики покриття коду. Загалом, метрики покриття коду використовуються для вимірювання ступеня покриття тестами програмного коду. Вони дозволяють оцінити, яка частина коду була виконана під час виконання тестових сценаріїв. Це важливий аспект в процесі тестування, оскільки високий рівень покриття коду свідчить про те, що тести охоплюють більшість або всі гілки виконання програми. Дана категорія включає в собі багато різноманітних метрик. Серед цієї групи метрик можна виділити такі метрики як метрика покриття рядків коду

У даній роботі, основна увага надана вдосконаленню, що пов’язане з автоматизацією розрахунку метрик покриття коду на прикладі просто застосунку написаного на мові програмування високого рівня PHP, що дозволяє ефективно та динамічно робити висновки про якість тестування програмного забезпечення.

PHP є скриптовою мовою програмування з відкритим кодом, яку багато розробників використовують в основному для веб-розробки. Це також мова загального призначення, яку можна використовувати для створення багатьох проектів різної складності та масштабу, у тому числі графічних інтерфейсів користувача.

У даній роботі будуть використовуватись такі технології як Docker, AWS разом із його окремими сервісами, Git, GitHub, GitHub Actions, Composer, а також бібліотека PHP під назвою PHPMailer та фреймворк PHP під назвою PHPUnit. Слід подати короткий опис кожної із вищенаведених технологій задля кращого розуміння важливості та необхідності їх використання.

Docker є платформою для розробки, доставки та запуску додатків в контейнерах. Вона забезпечує стандартизоване середовище для розробки та виконання додатків, що дозволяє розробникам розгортати додатки разом з усіма їхніми залежностями та конфігурацією. Docker надає ізольоване середовище для додатків у вигляді контейнерів, що дозволяє їм працювати безпечно та незалежно від інших додатків на одному хості [7].

Image(імедж) – це шаблон для створення контейнера. Він містить інструкції для побудови контейнера та всі необхідні файли та залежності. Імеджі можна завантажувати з централізованого сховища Docker під назвою Docker Hub або створювати власні. Імеджі містять виконуваний вихідний код програми, а

Міжнародний науково-технічний журнал
«Вимірювальна та обчислювальна техніка в технологічних процесах»
ISSN 2219-9365

також усі інструменти, бібліотеки та залежності, необхідні для запуску коду програми як контейнера. Після запуску імеджу Docker, він стає одним екземпляром Docker контейнера.

Container(контейнер) – це ізольоване середовище в якому запускається додаток разом із всіма його залежностями. Контейнери дозволяють упаковувати додаток та його середовище в одну єдину одиницю, що забезпечує ізольованість та переносимість. Кожен контейнер створюється на основі деякого Docker імеджу. Контейнер можна уявити як запущену програму, тобто окремий активний процес в операційній системі, а імедж у вигляді програми в не активному стані.

Git – це розподілена система керування версіями, яка була створена Лінусом Торвальдсом. Вона призначена для відстеження змін у програмному коді та управління версіями проекту. Git відомий своєю швидкістю, простим дизайном, підтримкою нелінійної розробки, повною децентралізацією та можливістю ефективно працювати з великими проектами. Коміт можна уявити як окрему зміну в коді. На основі комітів можна дізнатись які зміни були зроблені в проекті, а також дізнатись додаткову інформацію, таку як інформацію про автора коміту, а також заголовок та повідомлення коміту [8].

GitHub – це онлайн платформа на якій розміщується велика кількість репозиторіїв, які дають можливості розробникам для спільної роботи над програмними проектами з використанням системи керування версіями Git. GitHub став найпопулярнішою платформою для роботи з Git і використовується для спільної роботи над проектами від великих корпорацій до відкритих джерел та особистих проектів. Він допомагає розробникам спільно працювати над кодом, відстежувати зміни та покращувати якість програмного забезпечення.

GitHub Actions – це автоматизована система для створення, налаштування та виконання різних робочих процесів у репозиторії на GitHub. Вона дозволяє створювати та налаштовувати різні автоматизовані завдання і дії, які виконуються при певних подіях у репозиторії. GitHub Actions дозволяє розробникам автоматизувати рутинні завдання, такі як тестування коду, розгортання додатків, створення звітів.

Amazon Web Services (AWS) – це набір хмарних обчислювальних, зберігальних, мережевих та інших послуг, які надаються компанією Amazon. AWS є однією з найбільших та найпопулярніших хмарних платформ у світі і використовується підприємствами, стартапами, розробниками та іншими організаціями для будівництва, розгортання та керування різноманітними послугами та програмами в Інтернеті. Платформа AWS надає понад 200 повнофункціональних послуг із центрів обробки даних, розташованих по всьому світу, і є найповнішою у світі хмарною платформою [9].

Amazon S3 Bucket – це сервіс AWS, що призначений для зберігання об'єктів(файлів) у хмарному сервісі Amazon Web Services (AWS). Даний сервіс виконує роль деякого контейнера, який містить об'єкти даних, що зберігаються у ньому. S3 Bucket дозволяє організаціям та розробникам зберігати, керувати та надійно забезпечувати доступ до своїх об'єктів даних у хмарному сервісі AWS.

Amazon Elastic Container Service (Amazon ECS) – це керована сервісами AWS платформа для оркестрації та управління контейнерами. Вона дозволяє розробникам легко запускати, масштабувати та керувати контейнерами з додатками, що працюють в середовищі AWS. Amazon ECS підтримує Docker контейнери і надає можливості для автоматизації розгортання та управління контейнерними додатками великих масштабів. Amazon ECS допомагає розробникам спростити розгортання, масштабування та керування контейнерними додатками, забезпечуючи високу доступність та надійність в середовищі хмарної інфраструктури AWS.

Amazon Elastic Container Registry (Amazon ECR) – це керований сервіс AWS, який надає можливість зберігати, керувати та використовувати Docker імеджи у хмарному середовищі AWS. Amazon ECR дозволяє розробникам легко створювати та управляти репозитаріями Docker імеджів, зберігати імеджі контейнерів та безпечно розповсюджувати їх для розгортання в Amazon ECS, Kubernetes, або на інших платформах контейнеризації. Amazon ECR робить процес розгортання контейнерних додатків більш ефективним і зручним, забезпечуючи централізоване зберігання та керування Docker імеджами. Цей сервіс особливо корисний для розробників, які використовують контейнеризацію для створення та доставки своїх додатків.

Amazon EventBridge – це керований сервіс AWS, який надає можливість легко створювати, керувати та інтегрувати події між різними службами та додатками в середовищі хмарної платформи AWS. EventBridge дозволяє розробникам створювати програмні системи, які відповідають на події та взаємодіють з різними компонентами без необхідності написання власного коду для обробки подій. Amazon EventBridge робить розробку та інтеграцію додатків більш ефективними та динамічними, дозволяючи реагувати на події в реальному часі та спрощуючи роботу з комплексними архітектурами. Він широко використовується для розробки мікросервісів, обробки журналів, моніторингу та багатьох інших сценаріїв додатків у середовищі AWS [10].

PHPMailer – це бібліотека для PHP, яка дозволяє надсилати електронну пошту з додатків написаних мовою PHP. Вона надає можливості для створення і відправки електронних листів через SMTP, Sendmail, або інші протоколи поштової доставки. PHPMailer дозволяє легко інтегрувати функціональність надсилання пошти у додатках написаних мовою PHP. Дана бібліотека є популярним інструментом для надсилання пошти дозволяє легко та зручно додавати функціональність електронної пошти до будь-якого PHP додатку.

Міжнародний науково-технічний журнал
«Вимірювальна та обчислювальна техніка в технологічних процесах»
ISSN 2219-9365

PHPUnit – це популярний фреймворк для тестування PHP додатків. Він надає зручні інструменти та середовище для створення, виконання та аналізу тестів, які допомагають впевнитися в якості та надійності вашого коду. PHPUnit спрощує автоматизацію процесу тестування, що дозволяє розробникам виявляти помилки та відстежувати зміни у коді під час розвитку проекту. PHPUnit є стандартним фреймворком для тестування PHP-додатків і використовується розробниками для створення надійних та стабільних програм.

Composer – це зручний і популярний менеджер залежностей для PHP, який допомагає розробникам управляти залежностями своїх проектів і ефективно керувати бібліотеками та компонентами, необхідними для їхніх додатків. Composer дозволяє визначити, які бібліотеки використовуються у проекті та автоматично завантажувати та встановлювати їх на вимогу розробників [11].

Головна суть роботи полягає у створенні автоматизованого підходу для розрахунку та представлення метрик покриття коду на основі створених юніт тестів на прикладі простого додатку написаного мовою PHP. Керівники, менеджери та інші відповідальні за створенні програмних продуктів особи повинні мати змогу динамічно відслідковувати зміни, що вносять розробники у своєму коді. Інформацію про дані зміни можна отримувати у вигляді простих звітів на електронну пошту, кожного разу як розробник публікує свої зміни у віддаленому Git репозиторії. Інформація повинна містити данні про розробника та деталі розрахунку метрик покриття коду на основі виконаних юніт тестів для цільового програмного продукту на основі внесених змін у програмну систему. Загальну схему взаємодії компонентів розроблюваної програмної системи можна подати у вигляді схеми нижче (рис. 1):

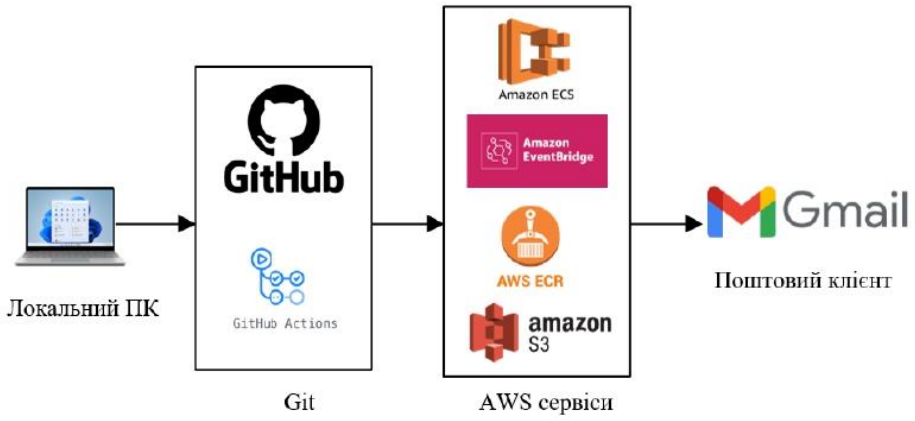


Рис. 1. Загальна схема взаємодії компонентів системи

Описати вищенаведену схему можна у вигляді загальних послідовних етапів без їх детального опису, які виконуються чітко один за одним, а саме:

1. Розробник створює програмний застосунок на мові високого рівня PHP. Після написання додатку, створюються юніт тести з використанням фреймворку PHPUnit.
2. Використовуючи систему контролю версій Git розробник відсилає свої зміни у віддалений репозиторій на GitHub.
3. З використанням GitHub Actions, у відповідь на подію відсилання змін в окрему наперед визначену гілку, відбувається запуск ряду команд, що ініціюють створення текстового файлу з інформацією про розробника, з'єднання із сервісом AWS S3 Bucket та відправку файлів із GitHub репозиторію у сховище S3 Bucket.
4. Після відправки файлів у S3 Bucket запускається наперед створена подія з використанням Amazon EventBridge, яка запускає у свою чергу Docker контейнер у сервісі Amazon ECS.
5. Запущений ECS контейнер, містить у собі консольний скрипт, який містить команди, що послідовно стягують усі файли із S3 Bucket, запускають юніт тести та створюють звіти метрик покриття коду з використанням фреймворку PHPUnit. Після створення репорту по метрикам покриття коду здійснюється запуск PHP скрипту, який виконує відправку листа з використанням бібліотеки PHPMailer. Електронний лист приходить на поштовий клієнт Gmail та містить детальні дані про автора Git коміту та текстовий файл із результатами розрахунку метрик покриття коду.

Вищеописаний підхід дозволяє використовувати усі переваги сучасних сервісів та хмарних технологій, що дозволяє отримувати детальні звіти із результатами метрик покриття коду. Процес повністю автоматизований та не потребує втручання зі сторони інших розробників, якщо не планується подальше масштабування та зміна даної програмної системи. Дана програмна система використовує ряд сучасних

Міжнародний науково-технічний журнал
«Вимірювальна та обчислювальна техніка в технологічних процесах»
ISSN 2219-9365

інструментів та сервісів для забезпечення ефективного тестування і збору метрик покриття коду. Розроблювана програмна система, яка автоматизує процес оцінювання метрик покриття коду та результатів тестування програмного забезпечення, може бути дуже корисною і доцільною для розробників і команд розробки програмного забезпечення з наступних причин:

1. Ефективність тестування. Дана система дозволяє автоматизувати процес виконання юніт тестів та оцінювання покриття коду при кожній зміні в коді. Це допомагає виявляти помилки і проблеми швидше та зменшує витрати на ручне тестування.

2. Зручність для розробників. Керівництво проектів може легко і швидко перевіряти вплив змін розробників на покриття коду та якість коду без необхідності ручного запуску тестів. Це спрощує їхню роботу та підвищує продуктивність.

3. Покращення якості коду. Завдяки автоматичному аналізу покриття коду, розробники можуть більше уваги приділяти написанню якісних тестів для критичних частин коду і вдосконалювати якість програмного забезпечення, оскільки вони будуть отримувати детальні відгуки про якість свого коду з боку керівництва.

4. Відстеження прогресу. Присутня можливість відстеження прогресу у покритті коду та створенні юніт тестів з часом, що допомагає керівництву та команді розробників аналізувати результати та вносити покращення.

5. Створення історії змін. Збереження результатів метрик та тестів дозволяє створювати історію змін якості коду, що може бути корисним для аудиту та аналізу.

6. Забезпечення стабільності продукту. Завдяки автоматичному тестуванню та оцінюванню покриття коду керівництво може бути впевненим в стабільності та надійності розроблюваного програмного продукту.

Дана програмна система потребує ряду покращень у майбутньому, а саме:

1. Моніторинг та логування. Можна додати моніторинг та систему логування до даної системи задля можливості слідкувати за станом процесів та виявляти проблеми або помилки в робочому процесі.

2. Безпека. Необхідно переконатись, що всі дані, які передаються між компонентами системи захищені та шифровані. Слід додатково дізнатись про найкращі практики забезпечення безпеки AWS ECS і інших інфраструктурних компонентів.

3. Тестування навантаження. Слід провести тестування навантаження, щоб переконатися, що дана програмна система зможе впоратися з великою кількістю запитів та буде здатна обробляти їх ефективно.

4. Зберігання результатів. Варто розглянути можливість зручно та ефективно зберігати історію результатів метрик покриття коду, щоб мати можливість їх аналізу та порівнювати поточні результати з попередніми версіями програмного забезпечення.

5. Резервне копіювання та відновлення. Важливо наперед спланувати резервне копіювання та можливості відновлення системи, щоб уникнути втрати даних або простою через непередбачені обставини.

6. Забезпечення оновлень і підтримки. Потрібно переконатись, що є план для оновлення компонентів, бібліотек і інфраструктури системи, а також для підтримки і покращення всієї системи у майбутньому.

Висновки з даного дослідження і перспективи подальших розвідок у даному напрямі

Отже, у даній статті був представлений та описаний інноваційний підхід до оцінювання покриття коду, спрямований на удосконалення процесу тестування та надання надійної структури для оцінки результатів тестування. Дана стаття робить вагомий внесок у сучасну практику розробки програмного забезпечення, підкреслюючи необхідність автоматизації оцінювання покриття коду та її позитивний вплив на надійність та якість програмних продуктів. Були описані та представлені сучасні технології з використанням яких була представлена розроблювана програмна система, що має на меті покращити процес забезпечення ефективного оцінювання результатів тестування програмних продуктів.

Література

1. P. Ammann, J. Offutt, “Introduction to Software Testing”, Cambridge University Press, 2013.
2. G. J. Myers, “The Art of Software Testing”, 2nd edition, 2004.
3. B. Beizer, “Software testing techniques”, 2nd edition, 2009.
4. D. Nicolette, “Software Development Metrics First Edition”, 2015.
5. Brent Laster, “Learning GitHub Actions”, 2023.
6. Shimon Ifrah, “Deploy Containers on AWS: With EC2, ECS, and EKS”, 2019.
7. James Turnbull, “The Docker Book: Containerization is the new virtualization”, 2014.
8. François Dupire, “Git Essentials: Developer's Guide to Git”, 2021.
9. Theo H. King, “AWS: The Ultimate Guide From Beginners To Advanced For The Amazon Web Services (2020 Edition)”, 2019.
10. David Boyne, “Amazon EventBridge”, 2019.

Міжнародний науково-технічний журнал
«Вимірювальна та обчислювальна техніка в технологічних процесах»
ISSN 2219-9365

11. W G T AVINDA, "Mastering PHP Dependency Management with Composer: Efficient Development of Modern PHP Applications (Web Development Book 2)", 2023.

References

1. P. Ammann, J. Offutt, "Introduction to Software Testing", Cambridge University Press, 2013.
2. G. J. Myers, "The Art of Software Testing", 2nd edition, 2004.
3. B. Beizer, "Software testing techniques", 2nd edition, 2009.
4. D. Nicolette, "Software Development Metrics First Edition", 2015.
5. Brent Laster, "Learning GitHub Actions", 2023.
6. Shimon Ifrah, "Deploy Containers on AWS: With EC2, ECS, and EKS", 2019.
7. James Turnbull, "The Docker Book: Containerization is the new virtualization", 2014.
8. François Dupire, "Git Essentials: Developer's Guide to Git", 2021.
9. Theo H. King, "AWS: The Ultimate Guide From Beginners To Advanced For The Amazon Web Services (2020 Edition)", 2019.
10. David Boyne, "Amazon EventBridge", 2019.
11. W G T AVINDA, "Mastering PHP Dependency Management with Composer: Efficient Development of Modern PHP Applications (Web Development Book 2)", 2023.

Міністерство освіти і науки України
Хмельницький національний університет

ЗБІРНИК НАУКОВИХ ПРАЦЬ

за матеріалами XV Всеукраїнської науково-практичної конференції
«Актуальні проблеми комп'ютерних наук АПКН-2023»

17-18 листопада 2023

Хмельницький 2023

АКТУАЛЬНІ ПРОБЛЕМИ КОМП'ЮТЕРНИХ НАУК - 2023

XV Всеукраїнська науково-практична конференція

Метою конференції є висвітлення актуальних проблем комп'ютерних наук, інформатики та інформаційних технологій.

СЕКЦІЇ КОНФЕРЕНЦІЇ:

1. Комп'ютерні науки та прикладні інформаційні технології.
2. Комп'ютерна інженерія та системи захисту інформації.
3. Математичне моделювання та інженерія програмного забезпечення
4. Телерадіокомунікації, медійні та комунікаційні системи.
5. Проблеми впровадження інформаційних технологій у виробництво та управління.

Робочі мови конференції: українська, англійська

ОРГКОМІТЕТ:

Олег СИНЮК – голова оргкомітету, проректор Хмельницького національного університету з наукової роботи, доктор технічних наук, професор

Олег САВЕНКО – заступник голови оргкомітету, декан факультету Інформаційних технологій ХНУ, доктор технічних наук, професор

Олександр БАРМАК – заступник голови оргкомітету, завідувач кафедри Комп'ютерних наук ХНУ, доктор технічних наук, професор

Тетяна ГОВОРУЩЕНКО – завідувач кафедри Комп'ютерної інженерії та інформаційних систем ХНУ, доктор технічних наук, професор

Олена ВИСОЦЬКА – доктор технічних наук, завідувач кафедри Радіоелектронних та біомедичних комп'ютеризованих засобів і технологій Національного аерокосмічного університету ім. М. Є. Жуковського «Харківський авіаційний інститут», професор

Євгеній ЛАВРОВ – доктор технічних наук, професор (Сумський державний університет)

Людмила ТІМОФЄЄВА – відповідальна за студентську науково-дослідну роботу ХНУ

Олександр МАЗУРЕЦЬ – секретар конференції, к.т.н., доцент кафедри Комп'ютерних наук ХНУ

Марина МОЛЧАНОВА – секретар конференції, викладач кафедри Комп'ютерних наук ХНУ

КОНТАКТНА ІНФОРМАЦІЯ:

e-mail для листування: *apkt.khnu@gmail.com*

УДК 004.054

Олійник П.О.

Хмельницький національний університет

УДОСКОНАЛЕНИЙ МЕТОД РОБОТИ З МЕТРИКАМИ ПОКРИТТЯ КОДУ ДЛЯ ЗАБЕЗПЕЧЕННЯ ЕФЕКТИВНОГО ОЦІНЮВАННЯ РЕЗУЛЬТАТІВ ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Розглянуто програмну систему, яка має на меті повністю автоматизувати процес розрахунку метрик покриття коду, що забезпечує ефективне оцінювання результатів тестування програмних продуктів. Був виконаний опис архітектури розроблюваної програмної системи. Представлена програмна система має на меті спростити завдання розробників, тестувальників та керуючого проектами персоналу, забезпечуючи підвищену ефективність та автоматизацію процесу оцінювання покриття коду та тестування.

The software system is considered, which aims at a fully automated process of calculating the code coverage metric, which provides an effective evaluation of the results of testing software products. The description of the developed software system' architecture was completed. The presented software system aims to simplify the tasks of developers, testers and staff project managers, ensuring high efficiency and automation of evaluating code coverage and testing processes.

У наш час стрімкої популярності розробки програмного забезпечення гостро постає питання у якості та ефективності проведення тестування програмного коду. Задля отримання детальних результатів стосовно правильності тестування використовуються метрики покриття коду, а саме їх процентні співвідношення, що показують ступінь покривання програмного коду тестами. Метрики покриття коду включають велику кількість метрик, серед яких виділяють метрику покриття рядків коду, метрику покриття функцій, метрику покриття класів [1]. Метою даного дослідження є вирішення наступних конкретних завдань:

1. Розробка програмної, яка автоматизовано збирає та аналізує дані щодо метрик покриття коду та результатів тестування в програмному забезпеченні.
2. Зниження витрат часу та ресурсів, які витрачаються на ручну перевірку покриття коду та виконання тестів, завдяки автоматизованій системі розрахунку метрик покриття коду.
3. Забезпечення підтримки DevOps та CI/CD, що допоможе автоматизувати та прискорити процес розробки та впровадження програмного забезпечення.

Задля реалізації програмної системи, що допоможе ефективно проводити розрахунок метрик покриття коду, було використано ряд технологій, серед яких можна виділити дві найбільш вагомих, а саме GitHub Actions та платформу AWS разом із її функціональними сервісами.

GitHub Actions є платформою безперервної інтеграції та безперервної доставки, що дозволяє автоматизувати розробку програмного забезпечення, його тестування та розгортання. GitHub містить багато різноманітних компонентів та може бути гнучко налаштованим в залежності від потреб розробників [2].

AWS є хмарною платформою, яка надає понад 200 повнофункціональних послуг із центрів обробки даних, розташованих по всьому світу, і є найповнішою у світі хмарною платформою [3]. У розробці програмної системи будуть використовуватись такі сервіси як Amazon S3 Bucket що виконує роль сховища файлів, Amazon ECS призначений для оркестрації Docker контейнерів, Amazon ECR який виконує роль регістру Docker імеджів та Amazon EventBridge який дозволяє створювати події та запускати відповідні операції у відповідь на події що відбуваються.

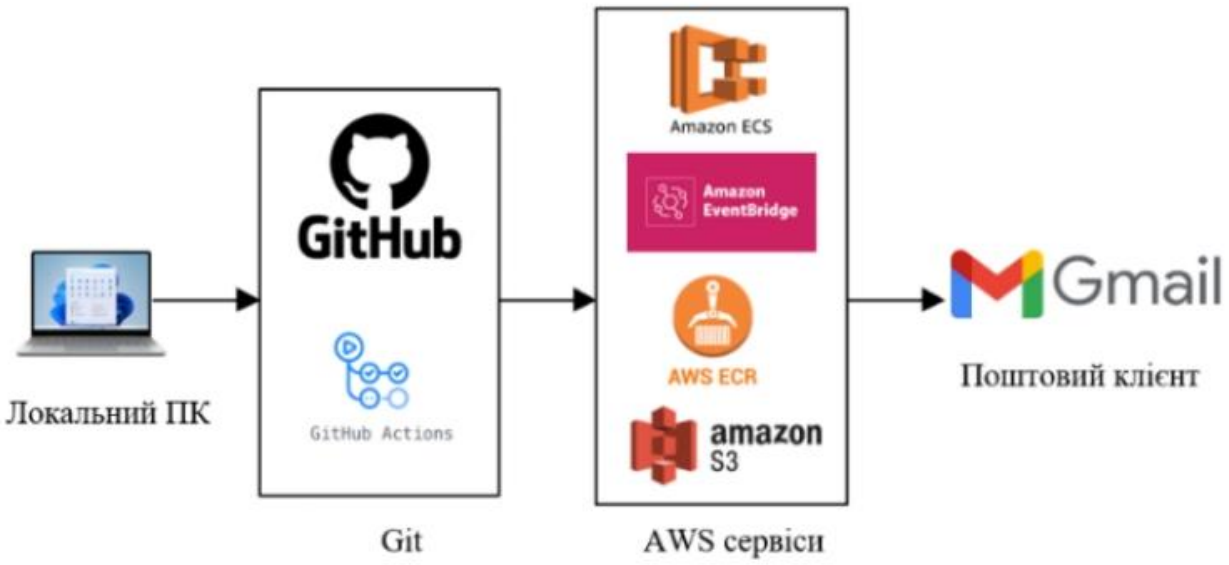


Рисунок 1 – Основних компоненти програмної системи

У відповідь на відправку змін розробником у віддалений GitHub репозиторій, GitHub Actions будуть здійснювати відсилання файлів репозиторію до сервісу Amazon S3 Bucket. У відповідь на подію додавання файлів у Amazon S3 Bucket буде відбуватись запуск контейнеру в сервісі Amazon ECS який буде відправляти електронний лист на вказану пошту одержувача. Лист буде містити

інформацію про автора Git коміту та результати розрахунку метрик покриття коду. Дана програмна система повинна автоматизувати процес розрахунку метрик покриття коду, що позитивно відобразиться на якості програмних продуктів. Загальний вигляд програмної системи та її основних компонентів можна представити у вигляді графічної схеми (рисунок 1).

Отже, була розглянута структура та поданий короткий опис запропонованої програмної системи, що допоможе розробникам, тестувальникам та іншим відповідальним особам у покращенні процесів тестування програмного забезпечення, сприятиме створенню високоякісних продуктів та дозволить підтримувати високий темп розробки у сучасній індустрії програмного забезпечення.

Перелік посилань

1. Code Coverage Metrics 2023: Guide & Explanation [Електронний ресурс] – Режим доступу до ресурсу: https://blog.gitnux.com/code-coverage-metrics/ .
2. What is GitHub Actions? Automated CI/CD for GitHub [Електронний ресурс] – Режим доступу до ресурсу: https://www.infoworld.com/article/3698188/what-is-github-actions-automated-cicd-for-github.html .
3. What is AWS: An Ultimate Guide to Amazon Web Services [Електронний ресурс] – Режим доступу до ресурсу: https://www.simplilearn.com/tutorials/aws-tutorial/what-is-aws .

ДОДАТОК В

(обов’язковий)

ПРЕЗЕНТАЦІЙНІ МАТЕРІАЛИ

Хмельницький Національний Університет
Кафедра інженерії програмного забезпечення

Кваліфікаційна робота магістра, на тему:

«Удосконалений метод роботи з метриками покриття коду для забезпечення ефективного оцінювання результатів тестування програмного забезпечення»

Розробив: Олійник Павло Анатолійович
Керівник: Бедратюк Леонід Петрович, д-р фіз.-мат. наук, професор

Об’єкт та предмет дослідження

- Об’єктом дослідження є метрики тестового покриття програмного коду.
- Предмет дослідження – удосконалений метод для роботи з метриками тестового покриття коду для забезпечення ефективного оцінювання результатів програмного забезпечення.

Актуальність та мета роботи

Актуальність роботи полягає в тому, що оцінка та аналіз якості покриття коду залишається популярною і важливою проблемою для розробників програмного забезпечення, тому існує потреба у створенні удосконаленого методу для роботи з метриками покриття коду, який забезпечить ефективну оцінку результатів тестування програмного забезпечення.

Метою роботи є розробка удосконаленого методу для роботи з метриками покриття коду шляхом проектування та реалізації інноваційної програмної системи, яка автоматизовує та спрощує процес оцінювання метрик покриття коду на основі результатів тестування та використовує сучасні технології та підходи.

Завдання роботи

- ❑ провести аналіз предметної області та сформувати постановку задачі;
- ❑ проаналізувати існуючі програмні рішення, що використовуються для розрахунку метрик покриття коду та запропонувати удосконалений метод для роботи з метриками покриття коду;
- ❑ розробити детальну архітектуру запропонованої програмної системи;
- ❑ на основі плану архітектури провести розробку програмної системи;
- ❑ здійснити огляд отриманих результатів та впевнитись у правильності роботи програмної системи;
- ❑ запропонувати рекомендації для покращень розробленого програмного рішення у майбутньому.

Наукова новизна

Вперше запропоновано удосконалений метод для роботи з метриками покриття коду, який полягає у забезпеченні повністю автоматизованого процесу оцінки метрик покриття коду та не потребує зовнішнього втручання команд розробників. Дана робота має важливий внесок у розвиток сучасних методів та інструментів для оцінки покриття коду та забезпечення ефективного оцінювання результатів тестування програмного забезпечення, що є актуальним завданням в інформаційній та комп'ютерній сфері.

Практична значимість

- поліпшення якості, надійності та ефективності розробки програмного забезпечення;
- автоматизація процесу розрахунку метрик покриття коду;
- значне зменшення часу та ресурсів розробників на етапі тестування програмного забезпечення.

Існуючі програмні рішення

- Istanbul;
- JaCoCo;
- Cobertura;

JaCoCo

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
org.jacoco.examples		58%		64%	24	53	97	193	19	38	6	12
org.jacoco.core		97%		93%	107	1,388	115	3,347	21	720	2	139
org.jacoco.agent.rt		77%		84%	31	121	62	310	21	74	7	20
jacoco-maven-plugin		90%		81%	35	183	44	407	8	110	0	19
org.jacoco.cli		97%		100%	4	109	10	275	4	74	0	20
org.jacoco.report		99%		99%	4	572	2	1,345	1	371	0	64
org.jacoco.ant		98%		99%	4	163	8	429	3	111	0	19
org.jacoco.agent		86%		75%	2	10	3	27	0	6	0	1
Total	1,355 of 27,352	95%	143 of 2,125	93%	211	2,599	341	6,333	77	1,504	15	294

```
----------------|----------|----------|----------|----------|-------------------|
File            | % Stmts  | % Branch | % Funcs  | % Lines  |Uncovered Lines    |
----------------|----------|----------|----------|----------|-------------------|
All files       |    98.92 |    94.36 |    99.49 |      100 |                   |
 yargs          |    99.17 |    93.95 |      100 |      100 |                   |
  index.js      |      100 |      100 |      100 |      100 |                   |
  yargs.js      |    99.15 |    93.86 |      100 |      100 |                   |
 yargs/lib      |     98.7 |    94.72 |    99.07 |      100 |                   |
  command.js    |     99.1 |    98.51 |      100 |      100 |                   |
  completion.js |      100 |    95.83 |      100 |      100 |                   |
  obj-filter.js |     87.5 |    83.33 |    66.67 |      100 |                   |
  usage.js      |    97.89 |    92.59 |      100 |      100 |                   |
  validation.js |      100 |    95.56 |      100 |      100 |                   |
----------------|----------|----------|----------|----------|-------------------|
```

Coverage Report - All Packages

Package	# Classes	Line Coverage	Branch Coverage	Complexity
All Packages	38	50%	62%	1.078
net.ivw.fdbS	3	77%	91%	1
net.ivw.fdbS.logic	3	57%	73%	0
net.ivw.fdbS.lucene	2	45%	58%	0
net.ivw.fdbS.model	5	66%	80%	1.286
net.ivw.fdbS.model.ibatis	9	70%	N/A	1
net.ivw.fdbS.model.rss	1	69%	0%	1.222
net.ivw.fdbS.struts	13	21%	29%	1.571
net.ivw.taglibs.tooltip	2	0%	N/A	1.2

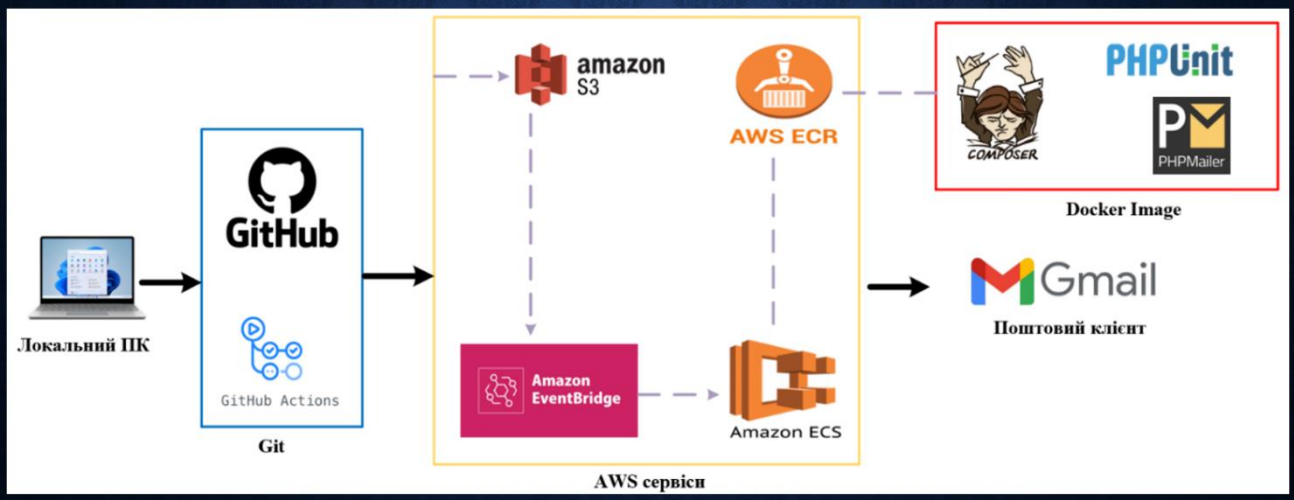
Report generated by Cobertura 1.7 on 3/13/06 12:52 PM.

Існуючі програмні рішення

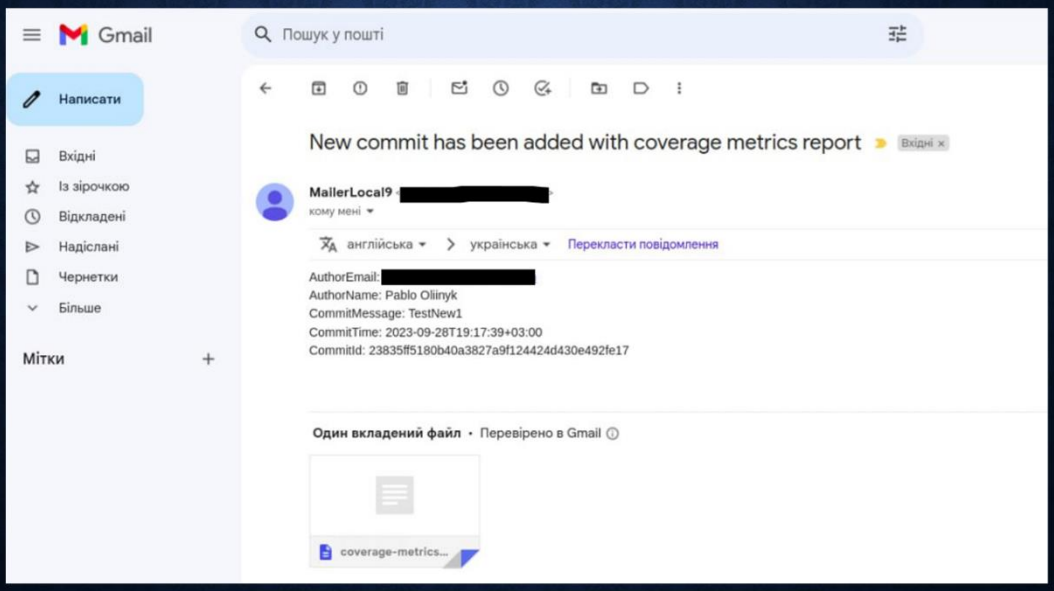
На основі розглянутих інструментів для вимірювання метрик покриття коду можна сказати, що кожен окремий інструмент націлений на розрахунок метрик покриття метрик для окремої мови програмування, а також розрахунок метрик відбувається в ручному режимі, тобто вимагає присутності розробника.

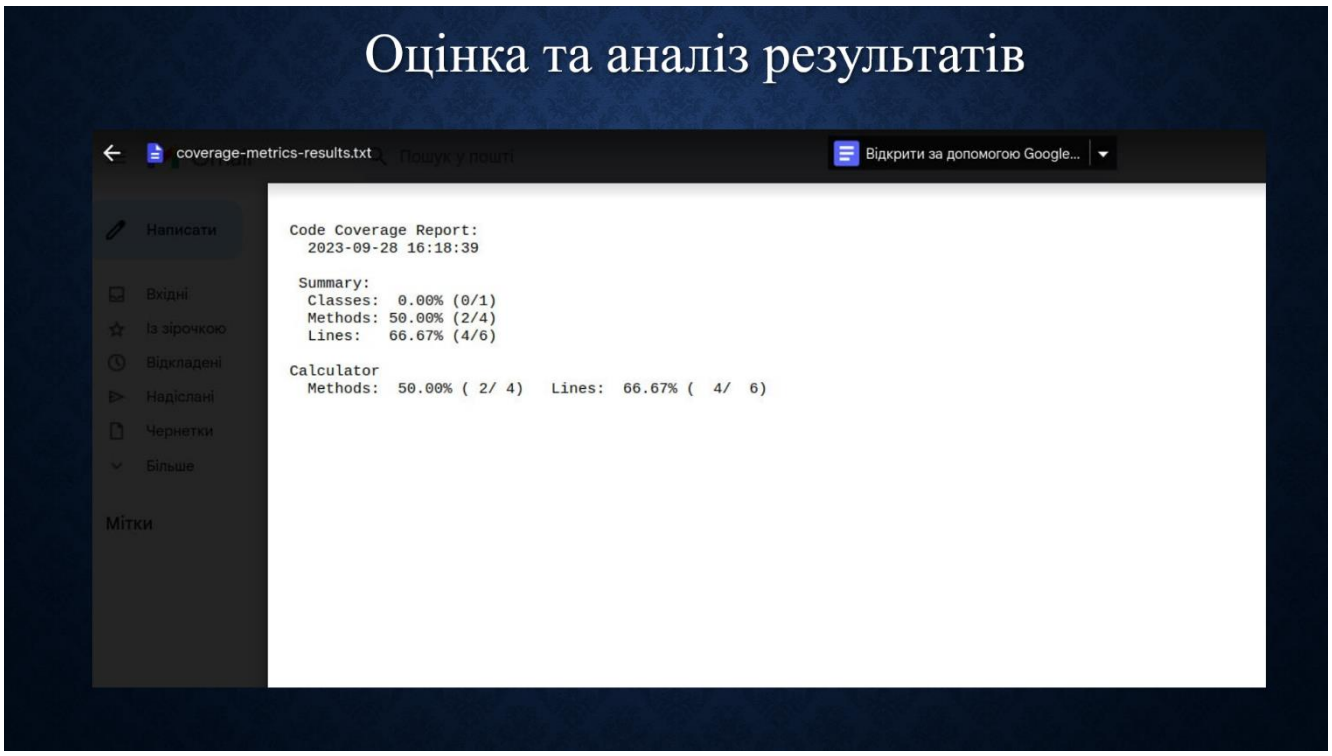
Існує потреба в розробці програмної системи, що буде надавати автоматизований спосіб для розрахунку метрик покриття коду, на основі складених Unit тестів, для якого не потрібна присутність розробників. Програмна система також повинна бути легко адаптованою для розрахунку метрик покриття коду для будь-якої мови програмування.

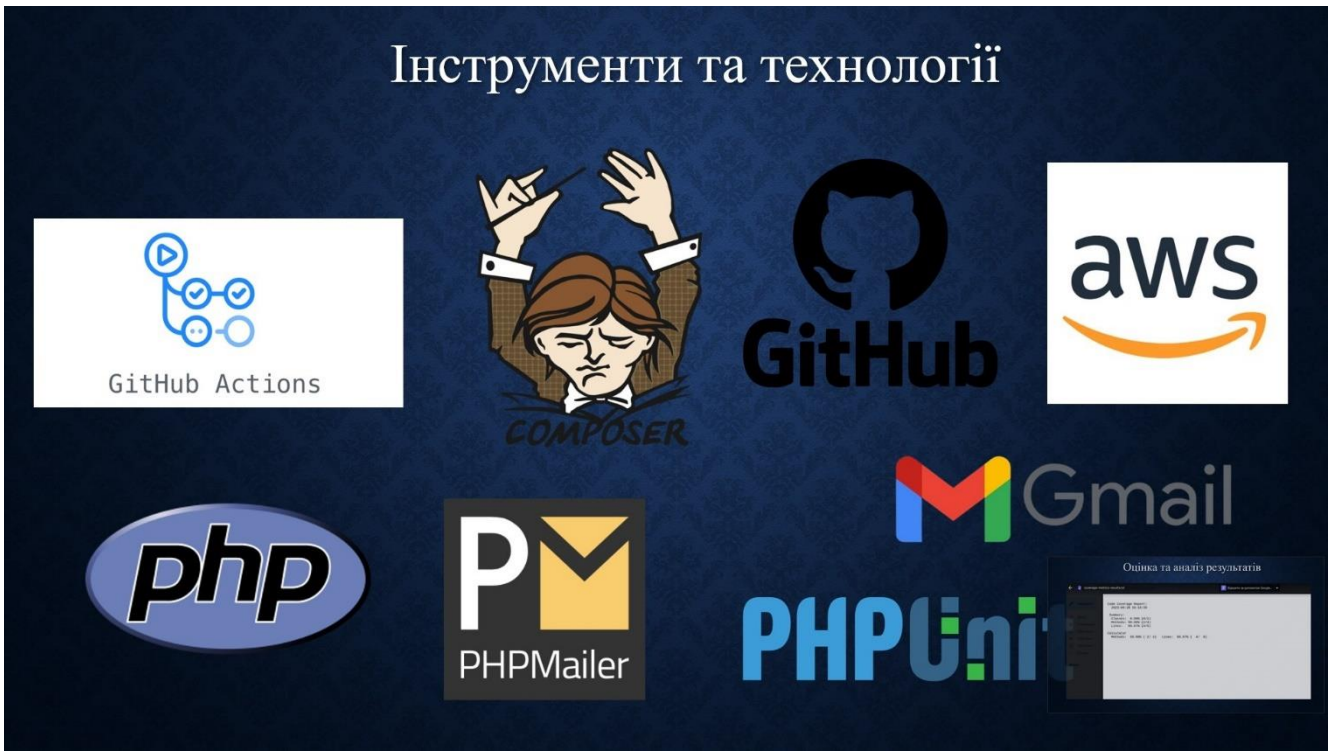
Архітектура програмної системи



Оцінка та аналіз результатів




Оцінка та аналіз результатів
coverage-metrics-results.txt
Відкрити за допомогою Google...
Написати
Вхідні
Із зірочкою
Відкладені
Надіслані
Чернетки
Більше
Мітки
Code Coverage Report:
2023-09-28 16:18:39
Summary:
Classes: 0.00% (0/1)
Methods: 50.00% (2/4)
Lines: 66.67% (4/6)
Calculator
Methods: 50.00% (2/ 4) Lines: 66.67% (4/ 6)


Інструменти та технології
GitHub Actions
COMPOSER
GitHub
aws
Gmail
php
PHPMailer
PHPUnit
Оцінка та аналіз результатів

Список публікацій

- Олійник П. А., Мартинюк Б. В., Удосконалений метод роботи з метриками покриття коду для забезпечення ефективного оцінювання результатів тестування програмного забезпечення. Вісник Хмельницького національного університету – 2023, №3 – 138-143 с.

- Олійник П. А., Мартинюк Б. В., Удосконалений метод роботи з метриками покриття коду для забезпечення ефективного оцінювання результатів тестування програмного забезпечення. Збірник наукових праць конференції АПКН-2023, №3.

Висновки

У результаті виконання кваліфікаційної роботи було вдосконалено метод для роботи з метриками покриття коду, шляхом розробки програмної системи, що має на меті автоматизувати процес генерації звітів, які містять результати обрахунку метрик покриття коду.

Було виконане детальне дослідження предметної області та наведена основна термінологія, що є ключовою для розуміння під час роботи над кваліфікаційною роботою. Під час дослідження існуючих програмних рішень, що надають можливості обрахунку метрик покриття коду, були виділені їх недоліки та запропонована необхідність у розробці власної універсальної програмної системи, що має на меті спростити процес обрахунку метрик покриття коду на основі результатів тестування. Був здійснений детальний аналіз сучасних технологій, що використовуються при розробці програмної системи.

Була приведена чітка схема архітектури програмної системи, що дозволило провести її реалізацію під час якої були надані детальні описи ключових скриптів та файлів із приведенням їх програмних кодів.

Були наведені та проаналізовані кінцеві результати роботи програмної системи, що дозволило впевнитись у її працездатності. Також, додатково було запропоновано ряд покращень, які можуть бути внесені у розроблену програмну систему у майбутньому.

Загалом, результати кваліфікаційної роботи відкривають широкі можливості для подальшого використання у наукових та практичних аспектах розробки програмного забезпечення, сприяючи покращенню якості та надійності програмних продуктів шляхом автоматизації розрахунку метрик покриття коду на основі готових тестових сценаріїв.

Завідувачу кафедри інженерії програмного
забезпечення проф. Леоніду БЕДРАТЮКУ
здобувача вищої освіти
Олійника Павла Анатолійовича
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.

06.12.2023
дата

підпис

Anti-Plagiarism v-15.257

Максимальне співпадіння з одним документом 3.0%

Словники перевірки: en_US, ru_RU, ua_UA. **Помилок в документах: 9%**

	Документ		Сумарний збіг по Базі Даних	
ID: 121915 Назва: Удосконалений метод роботи з метриками тестового покриття коду для забезпечення ефективного оцінювання результатів тестування програмного забезпечення Додано в БД: 2023-12-06 Автора: Олійни Павло Керівники: Бедратюк Леонід Консультанти: Опоненти:	Символи	Лексеми	Символи	Лексеми
	114427	994	4496 (4%)	79 (8%)

Джерело плагіату

ID	Опис	Наявність плагіату в документі	
		Символи	Лексеми

UNICHECK
by Turnitin

Ім'я користувача:
ІПЗ

ID перевірки:
1015975581

Дата перевірки:
06.12.2023 11:31:47 EET

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
06.12.2023 11:36:42 EET

ID користувача:
100012953

Назва документа: **Олійник**

Кількість сторінок: **94** Кількість слів: **17259** Кількість символів: **139064** Розмір файлу: **4.31 MB** ID файлу: **1015655173**

4.76%
Схожість

Найбільша схожість: **0.52%** з джерелом з Бібліотеки (ID файлу: **1015653883**)

4.53% Джерела з Інтернету	694	Сторінка 96
0.97% Джерела з Бібліотеки	72	Сторінка 99

0.09% Цитат

Цитати	1	Сторінка 100

Не знайдено жодних посилань

0%
Вилучень

Немає вилучених джерел

Модифікації

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

Замінені символи	3

РІШЕННЯ ЕКСПЕРНОЇ 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: «Удосконалений метод роботи з метриками тестового покриття коду для забезпечення ефективного оцінювання результатів тестування програмного забезпечення»
Автор: Олійник Павло Анатолійович
Спеціальність: 121 – Інженерія програмного забезпечення
Освітня програма: Освітньо-професійна програма «Інженерія програмного забезпечення»
Науковий керівник: Бедратюк Леонід Петрович, д-р фіз.-мат. наук, професор
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) в якості запозичень системою Unicheck було зафіксовано деякі послідовності вихідного коду і посилання на бібліотеки, які є стандартними мовними конструкціями програмування та не можуть розглядатися як об'єкт авторських прав і, відповідно, їх порушення;
3) запозичення, виявлені в тексті роботи, є фрагментарними або мають належним чином оформленні посилання;
4) виявлені модифікації тексту не впливають на відсоток схожості.
Максимальний обсяг запозичень, визначений системою Anti-Plagiarism, складає 3.0%. Обсяг запозичень, визначений системою Unicheck виявлення збігів ідентичності/схожості, складає 4.76% і адресується до 694 джерел з інтернету і 72 джерела з бібліотеки, що, з урахуванням наведених обґрунтувань, відповідає характеру теми і свідчить на користь кваліфікаційної роботи.

Дата 6.12.2023

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

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

РЕЦЕНЗІЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

Здобувач Олійник Павло Анатолійович

Тема Удосконалений метод роботи з метриками тестового покриття коду для забезпечення ефективного оцінювання результатів тестування програмного забезпечення

Спеціальність 121 «Інженерія програмного забезпечення»

Обсяг кваліфікаційної роботи:

Кількість листів креслень _____; кількість сторінок записки 121

1.Короткий зміст роботи та прийнятих рішень У кваліфікаційній роботі був здійснений аналіз предметної області та наведено існуючі програмні рішення, що призначені для обрахунку метрик покриття коду. В ході аналізу існуючих рішень були виділені їх недоліки, що дозволило зробити висновок про необхідність розробки програмної системи, яка має на меті повністю автоматизувати процес розрахунку метрик покриття коду. Були наведені описи сервісів, що використовуються для розробки програмної системи. Наведено схему архітектури програмної системи та проведений опис її реалізації. В результаті була здійснена перевірка працездатності програмного рішення та наведені рекомендації для його покращень у майбутньому.

2. Висновок про відповідність роботи поставленому завданню Кваліфікаційна робота освітнього ступеня «магістр» була виконана з дотриманням усіх вимог та стандартів стосовно оформлення, а також повністю відповідає поставленому завданню.

3. Характеристика виконання кожного розділу роботи, ступінь використання останніх досягнень науки і техніки і передових методів роботи: Вступ вміщує пояснювальну частину, що включає актуальність досліджуваної теми, мету та завдання роботи, наведення наукової новизни та практичної значимості отриманих результатів. У першому розділі був проведений аналіз предметної області, подане роз'яснення ключових термінів. Були розглянуті існуючі програмні рішення для розрахунку метрик покриття коду та виділені їх недоліки, що дозволило зробити висновок про необхідність розробки власної програмної системи. У другому розділі були наведені описи та детальний аналіз технологій та сервісів, що використовуються при розробці програмної системи. У третьому розділі була наведена детальна схема архітектури програмної системи на основі якої була здійснена її реалізація та приведені лістинги програмного коду. У четвертому розділі були проаналізовані отримані результати розрахованих метрик покриття коду, а також наведено ряд можливих покращень для розробленої програмної системи.

4. Позитивні сторони роботи Кваліфікаційна робота містить удосконалений метод роботи з метриками покриття коду, що допомагає забезпечити ефективне оцінювання результатів тестування програмного забезпечення. Цей метод досягається через використання інноваційної програмної системи, що автоматизує процес розрахунку метрик покриття коду, усуваючи пряме втручання розробників, що допомагає пришвидшити розробку програмного забезпечення. Програмна система використовує сучасні сервіси та платформи, а також може бути адаптованою під будь-яку мову програмування, що характеризує її універсальність та унікальність.

5. Негативні сторони роботи У роботі не було здійснене детальне тестування працездатності програмної системи під великими навантаженнями, коли більше одного розробника вносять багато змін до GitHub репозиторію одночасно, що може в перспективі допомогти виявити потенційні баги у роботі програмної системи. Також, було б добре забезпечити більш надійний та зручний спосіб зберігання звітів із метриками покриття коду. Тобто замість відправки звітів лише на електронну пошту, необхідно додатково реалізувати можливість їх зберігання у окремій базі даних та розробити зручний графічний інтерфейс, що дозволить переглядати звіти та здійснювати маніпуляції над ними.

6. Оцінка графічного оформлення та пояснювальної записки роботи Графічне оформлення було виконане згідно тематики кваліфікаційної роботи з дотриманням усіх поставлених вимог. Пояснювальна записка у повній мірі виконана з дотриманням вимог чинних стандартів.

7. Відгук про роботу в цілому Загалом, зміст кваліфікаційної роботи був поданий доволі структуровано та чітко. Усі розділи роботи є досить послідовними та інформативними, що допомагає легко орієнтуватись у роботі в цілому. Графічна частина в повній мірі відображає описані елементі, що згадуються в тексті роботи.

8. Інші зауваження

9. Оцінка кваліфікаційної роботи Детально проаналізувавши кваліфікаційну роботу можна зробити висновок, що вона заслуговує на оцінку «відмінно».

РЕЦЕНЗЕНТ (прізвище, ім'я, по-батькові, посада, місце роботи)

Говорущенко Т. О., д.т.н., проф., завідувач кафедри комп'ютерної інженерії та інформаційних систем

«6» грудня 2023 р. _____
(підпис)