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

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

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.

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):

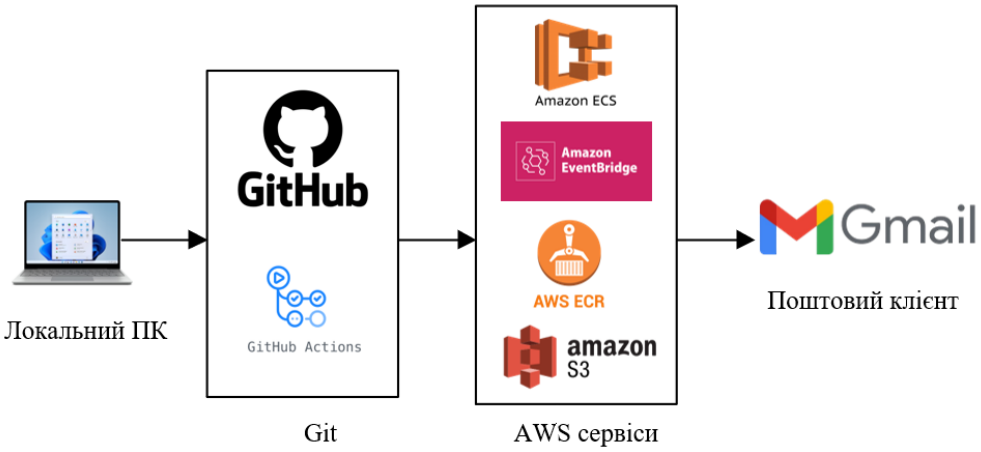

Рис. 1. Загальна схема взаємодії компонентів системи

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

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.

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.