

KHOA KỸ THUẬT VÀ CÔNG NGHỆ
BỘ MÔN CÔNG NGHỆ THÔNG TIN



THỰC TẬP ĐỒ ÁN CƠ SỞ NGÀNH
HỌC KỲ 1, NĂM HỌC 2023 – 2024

NGHIÊN CỨU GITHUB ACTION ĐỂ KIỂM THỬ TỰ ĐỘNG.

Giáo viên hướng dẫn:
Họ tên: Nguyễn Bảo Ân

Sinh viên thực hiện:
Họ tên: Nguyễn Tấn Lộc
MSSV: 110121189
Lớp: DA21TTB

Trà Vinh, tháng 01 năm 2024

KHOA KỸ THUẬT VÀ CÔNG NGHỆ
BỘ MÔN CÔNG NGHỆ THÔNG TIN



THỰC TẬP ĐỒ ÁN CƠ SỞ NGÀNH
HỌC KỲ 1, NĂM HỌC 2023 – 2024

NGHIÊN CỨU GITHUB ACTION ĐỂ KIỂM THỬ TỰ ĐỘNG.

Giáo viên hướng dẫn:
Họ tên: Nguyễn Bảo Ân

Sinh viên thực hiện:
Họ tên: Nguyễn Tấn Lộc
MSSV: 110121189
Lớp: DA21TTB

Trà Vinh, tháng 01 năm 2024

NHẬN XÉT CỦA GIÁO VIÊN HƯỚNG DẪN

This image shows a full page of white paper with horizontal dotted lines. The lines are evenly spaced and run across the width of the page, providing a guide for writing. There are no margins, text, or other markings on the paper.

Trà Vinh, ngày tháng năm

Giáo viên hướng dẫn

(Ký tên và ghi rõ họ tên)

NHẬN XÉT CỦA THÀNH VIÊN HỘI ĐỒNG

[illegible]

Trà Vinh, ngày tháng năm

Thành viên hội đồng
(Ký tên và ghi rõ họ tên)

LỜI CẢM ƠN

Em xin chân thành cảm ơn thầy Nguyễn Bảo Ân đã dành thời gian hướng dẫn tạo điều kiện và giúp em có thêm kiến thức trong quá trình thực hiện đề tài.

Thông qua quá trình thực hiện đề tài, em đã phần nào tích lũy được những kiến thức về GitHub Action, công cụ kiểm thử và hiểu biết thêm về ngôn ngữ lập trình Java.

Trong quá trình thực hiện nghiên cứu đề án, dù đã cố gắng hết sức nhưng em vẫn không tránh khỏi những sai sót, mong được thầy, cô góp ý và thông cảm. Giúp em học hỏi và cải thiện các dự án trong tương lai.

MỤC LỤC

MỞ ĐẦU	10
CHƯƠNG 1: TỔNG QUAN.....	12
CHƯƠNG 2: NGHIÊN CỨU LÝ THUYẾT	13
2.1 Quản lý mã nguồn với GitHub.	13
2.1.1 Giới thiệu về GitHub.	13
2.1.2 Tính năng của GitHub.	13
2.1.3 Vai trò của GitHub.	14
2.2 Kiểm thử tự động với GitHub Action.	14
2.2.1 Giới thiệu GitHub Action.	14
2.2.2 Các thành phần của GitHub Action.....	15
2.2.3 Cách sử dụng GitHub Action để kiểm thử tự động.....	15
2.3 Tích hợp liên tục.	17
2.3.1 Khái niệm.	17
2.3.2 Vai trò.....	18
2.3.3 Cách thức hoạt động.....	18
2.4 Unit test.....	19
2.4.1 Khái niệm.	19
2.4.2 Các khái niệm có liên quan khi dùng Unit Test.	19
2.4.3 Vai trò của Unit test.....	20
2.4.4 Ứng dụng Unit test.	20
CHƯƠNG 3: HIỆN THỰC HÓA NGHIÊN CỨU.....	21
3.1 Xây dựng Java Maven.	21
3.1.1 Tổng quan.....	21
3.1.2 Sử dụng quy trình làm việc khởi động Maven.....	21
3.2 Mô tả thuật toán.	23

3.2.1 Tổng quan.....	23
3.2.2 Phân tích.	23
3.2.3 Viết Test Cases sử dụng Junit.	25
3.3 GitHub Workflows.	28
3.3.1 Workflows.	28
3.3.2 Xây dựng và sử dụng.....	28
3.3.3 Các phụ thuộc bộ nhớ đệm.	28
3.3.4 Đóng gói dữ liệu quy trình làm việc.....	29
3.3.5 Tạo Workflows.....	30
CHƯƠNG 4: KẾT QUẢ NGHIÊN CỨU.....	34
CHƯƠNG 5: KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN	41
DANH MỤC TÀI LIỆU THAM KHẢO.....	42
PHỤ LỤC.....	43

DANH MỤC HÌNH ẢNH – BẢNG BIỂU

Hình 2. 1	15
Hình 2. 2	16
Hình 3. 1	21
Hình 3. 2	22
Hình 3. 3	22
Hình 3. 4	23
Hình 3. 5	23
Hình 3. 6. Tạo file Calculator.....	24
Hình 3. 7	25
Hình 3. 8. Tạo file TestCalculator.....	26
Hình 3. 9	29
Hình 3. 10	30
Hình 3. 11	30
Hình 3. 12	31
Hình 3. 13. Tạo file maven.....	33
Hình 4. 1. File Java Maven (maven.yml).....	34
Hình 4. 2. File Class (Calculator.java).....	35
Hình 4. 3. File Test (TestCalculator.java).....	37
Hình 4. 4. Kết quả kiểm thử tất cả điều kiện đều đúng.....	38
Hình 4. 5. Kết quả kiểm thử các điều kiện đều đúng và một điều kiện được bỏ qua.	39
Hình 4. 6. Kết quả kiểm thử các điều kiện đều đúng và một điều kiện sai.....	40

TÓM TẮT ĐỒ ÁN CƠ SỞ NGÀNH

Tóm tắt vấn đề:

GitHub Actions cung cấp khả năng kiểm thử tự động các dự án trên nền tảng GitHub. Nghiên cứu GitHub Action để kiểm thử tự động xoay quanh việc tự động hóa quy trình kiểm thử, tăng cường chất lượng và giảm thiểu lỗi.

Các hướng tiếp cận:

- Sử dụng GitHub Action để tạo workflows chạy các test cases tự động khi có sự thay đổi vào mã nguồn. Điều này gồm việc cấu hình lên các bước kiểm thử như cài đặt môi trường, chạy test cases và báo cáo kết quả.
- Xác định được công cụ kiểm thử phù hợp với dự án là JUnit. Tùy vào nhu cầu có thể sử dụng các framework như Selenium, Cypress hoặc các công cụ kiểm thử tự động khác.
- Cấu hình workflows để tạo ra báo cáo chi tiết của kết quả kiểm thử, gồm hiển thị thông tin .

Cách giải quyết vấn đề:

- Sử dụng GitHub Actions để tự động hóa quá trình kiểm thử, giúp đảm bảo rằng mã nguồn của dự án luôn hoạt động đúng và ổn định.
- Tạo workflows linh hoạt để thực hiện các bước kiểm thử cần thiết và cung cấp báo cáo chi tiết về kết quả kiểm thử.

Một số kết quả đạt được:

- Kiểm thử tự động giúp phát hiện và giảm thiểu lỗi.
- Kiểm thử tự động giúp tiết kiệm thời gian so với kiểm thử thủ công.
- Cho ra được đầy đủ và chi tiết kết quả kiểm thử, giúp theo dõi được chất lượng và xác định các vấn đề cần phải sửa.

MỞ ĐẦU

1. Lý do chọn đề tài.

- GitHub Action là công cụ quan trọng cho việc tự động hóa quy trình phát triển phần mềm. Sử dụng GitHub Action để kiểm thử tự động là cần thiết để hiểu thêm những điều mới trong công nghệ phần mềm.
- Kiểm thử tự động thông qua GitHub Action giúp giảm thời gian và công sức cho nhà phát triển, giúp tập trung vào việc viết mã và cải thiện tính năng của phần mềm.
- Sử dụng GitHub Action để kiểm thử tự động giúp tối ưu hóa quy trình, giảm thời gian kiểm thử, tăng cường hiệu suất, chất lượng và độ tin cậy của sản phẩm.

2. Mục đích.

- Sử dụng GitHub Action để kiểm thử tự động giúp nhà phát triển tự động hóa quá trình kiểm thử mã nguồn. Điều này có thể giúp cải thiện chất lượng mã nguồn, giảm thời gian phát triển và cải thiện trải nghiệm của người dùng.
- Sử dụng các hành động sẵn có hoặc tự tạo các hành động để định nghĩa các quy trình kiểm thử tự động. Các hành động là các đơn vị nhỏ của mã có thể được kết hợp để tạo ra các quy trình kiểm thử phức tạp.
- GitHub Actions cung cấp một số lợi ích cho kiểm thử tự động, bao gồm:
 - + Linh hoạt: GitHub Actions cho phép các nhà phát triển tùy chỉnh quy trình kiểm thử để phù hợp với nhu cầu cụ thể.
 - + Tự động hóa: GitHub Actions có thể tự động chạy quy trình kiểm thử bất cứ khi nào có thay đổi đối với mã nguồn. Điều này giúp đảm bảo rằng mã luôn được kiểm tra đầy đủ.
 - + Tích hợp: GitHub Actions được tích hợp chặt chẽ với GitHub, giúp dễ dàng thiết lập và sử dụng.

3. Đối tượng nghiên cứu.

- Đối tượng nghiên cứu là các nhà phát triển phần mềm có thể là các nhà phát triển cá nhân, các nhóm phát triển hoặc các doanh nghiệp sử dụng để nâng cao kỹ năng về GitHub Action, tối ưu và cải thiện quá trình kiểm thử trong quy trình phát triển, tăng cường chất lượng sản phẩm.

4. Phạm vi nghiên cứu.

- Nghiên cứu lý thuyết: Nghiên cứu các tài liệu tham khảo, các thông tin về GitHub Action và kiểm thử tự động.

- Nghiên cứu cách sử dụng GitHub Action để xây dựng, thiết lập, cấu hình và triển khai các bộ kiểm thử tự động.

- Nghiên cứu cách tối ưu và cá nhân hóa quy trình kiểm thử tự động để đáp ứng nhu cầu cụ thể của từng dự án.

- Thực hiện các thử nghiệm để đánh giá hiệu quả GitHub Action trong kiểm thử tự động và thu thập, phân tích dữ liệu để đánh giá tác động của GitHub Action đối với chất lượng mã nguồn.

CHƯƠNG 1: TỔNG QUAN

Kiểm thử phần mềm là một phần quan trọng trong quy trình phát triển để đảm bảo chất lượng và độ tin cậy của sản phẩm. Kiểm thử phần mềm là bước không thể thiếu trong quá trình phát triển phần mềm, vì thông qua việc kiểm thử ta có thể phát hiện ra các lỗi phần mềm hoặc các lỗi ẩn sẵn có.

Công nghệ thông tin đang phát triển mạnh mẽ, việc đánh cắp dữ liệu là thường gặp đối với các phần mềm có lỗ hổng. Việc kiểm thử sẽ giúp ngăn chặn và hạn chế những cuộc tấn công độc hại.

Trước đây có thể sử dụng những công cụ khác để kiểm thử nhưng hiện giờ ta có sử dụng GitHub Action để xây dựng các workflow trong đó có các bước để kiểm thử tự động.

Nghiên cứu GitHub Action để kiểm thử tự động là đề tài nghiên cứu đóng vai trò trong việc tối ưu hóa các quy trình phát triển phần mềm.

GitHub cung cấp công cụ GitHub Action, là công cụ giúp ta vừa đưa mã nguồn mới từ kho lưu trữ từ xa lên thì mã nguồn mới đó phải được kiểm thử liên và khi kiểm thử ngay tại đó thì ta đòi hỏi phải có công cụ tự động và từ khóa đó là CI (Continuous Integration) tích hợp liên tục, trước khi tích hợp liên tục là phải kiểm thử và GitHub Action nó sẽ tự động thực hiện những yêu cầu của ta đưa lên.

Việc sử dụng GitHub Action để kiểm thử tự động đóng vai trò quan trọng đối với việc phát triển phần mềm, giúp tạo môi trường công tác giữa các thành viên trong cùng một dự án, cải thiện, xây dựng và chia sẻ các quy trình kiểm thử tăng cường sự tin cậy và ổn định của các dự án.

CHƯƠNG 2: NGHIÊN CỨU LÝ THUYẾT

2.1 Quản lý mã nguồn với GitHub.

2.1.1 Giới thiệu về GitHub.

GitHub là một dịch vụ cung cấp kho lưu trữ mã nguồn Git dựa trên nền web cho các dự án phát triển phần mềm. GitHub lưu trữ mã nguồn và quản lý các phiên bản phân tán. Nó cho phép người dùng phát triển phần mềm cộng tác, quản lý và theo dõi các thay đổi của dự án.

GitHub hiện đã trở thành yếu tố có sức ảnh hưởng trong cộng đồng phát triển mã nguồn. Nhiều nhà phát triển bắt đầu xem GitHub là sự thay thế cho sơ yếu lý lịch và một số nhà tuyển dụng đánh giá ứng viên qua việc yêu cầu các ứng viên cung cấp một liên kết đến tài khoản GitHub.

GitHub cung cấp nền tảng trực quan và dễ sử dụng để lưu trữ các dự án phần mềm, quản lý phiên bản và phối hợp công việc. Nó được cung cấp nhiều tính năng hữu ích như theo dõi lỗi, các công cụ liên quan đến kiểm tra, tích hợp liên tục và nhiều hơn nữa.

2.1.2 Tính năng của GitHub.

- **Lưu trữ mã nguồn:** GitHub cho phép lưu trữ mã nguồn của dự án trong các kho lưu trữ, ta có thể tạo kho lưu trữ và tải mã nguồn lên từ máy tính cá nhân.
- **Quản lý phiên bản:** GitHub được xây dựng dựa trên hệ thống thông tin quản lý phiên bản phân tán của Git. Có thể quản lý các phiên bản mã nguồn của dự án, tạo ra các nhánh, thực hiện các thay và hợp nhất các nhánh.
- **Hệ thống theo dõi lỗi:** GitHub cung cấp tính năng theo dõi lỗi để dễ dàng quản lý và giải quyết vấn đề liên quan đến dự án.
- **Wiki:** GitHub cho phép tạo các trang wiki để tài liệu hoặc dự án. Có thể tạo các trang wiki để hướng dẫn, chia sẻ thông tin và tài liệu liên quan của dự án.
- **Tích hợp liên tục:** GitHub tích hợp các công cụ CI/CD phổ biến như Travis CI, Jenkins và nhiều công cụ khác. Nó giúp tự động hóa quy trình kiểm tra và cải thiện mã nguồn.
- **Công cụ hỗ trợ kiểm tra:** GitHub cung cấp công cụ cho phép xem xét, phê duyệt và đề xuất các thay đổi vào mã nguồn. Có thể tạo các yêu cầu và mời người khác vào tham gia quá trình xem xét và thảo luận.

- **Nhận xét và phản hồi:** Ta có thể xem, đánh giá và bình luận về dự án, giúp chia sẻ ý kiến, giao tiếp và cải thiện.
- **Tìm kiếm và khám phá:** GitHub cung cấp các công cụ tìm kiếm, giúp tìm kiếm các dự án, cá nhân, tổ chức và các nội dung liên quan đến mã nguồn.

2.1.3 Vai trò của GitHub.

- **Quản lý phiên bản:** GitHub sử dụng hệ thống kiểm soát phiên bản Git, cho phép lưu trữ các mã nguồn, theo dõi lịch sử thay đổi và quản lý những phiên bản khác nhau của phần mềm. Giúp dễ dàng quản lý và theo dõi các thay đổi, cũng như khôi phục mã nguồn trở về trạng thái trước.
- **Đóng góp và hợp tác:** GitHub cung cấp các tính năng cho phép nhiều người dùng cùng làm việc trên cùng một dự án. Thông qua việc thực hiện thay đổi, xem xét mã nguồn và đưa ra ý kiến thì người dùng có thể đóng góp vào mã nguồn.
- **Quản lý dự án:** GitHub cung cấp các công cụ quản lý về quản lý dự án như theo dõi tiến độ, quản lý vấn đề, yêu cầu kéo và cấp quyền truy cập, giúp điều hành và tổ chức dự án hiệu quả.
- **Triển khai và lưu trữ:** GitHub cung cấp dịch vụ trả phí cho những dự án riêng tư và dịch vụ miễn phí cho các dự án mã nguồn mở. Ngoài ra GitHub còn giúp triển khai ứng dụng web thông qua GitHub Pages và các dịch vụ khác.
- **Khám phá và học hỏi:** GitHub là nơi có nhiều dự án mã nguồn mở cho phép người dùng tìm kiếm, chia sẻ và học hỏi các dự án.

2.2 Kiểm thử tự động với GitHub Action.

2.2.1 Giới thiệu GitHub Action.

- Github Action là công cụ do GitHub cung cấp để có thể tự động hóa quy trình CI/CD (Continuous Integration/Continuous Deployment), cho phép người sử dụng định nghĩa các workflow (quy trình làm việc) tự động hóa các hoạt động trong việc phát triển phần mềm.
- Mỗi workflow trong GitHub Action là một tập hợp các hành động được định nghĩa trong file YAML, các hành động có thể là các lệnh cụ thể như: test; triển khai, build ứng dụng. Có thể dùng các action được cung cấp sẵn bởi GitHub, các action được cộng đồng lập trình tạo ra hoặc tạo các action tùy chỉnh để phù hợp với nhu cầu dự án.

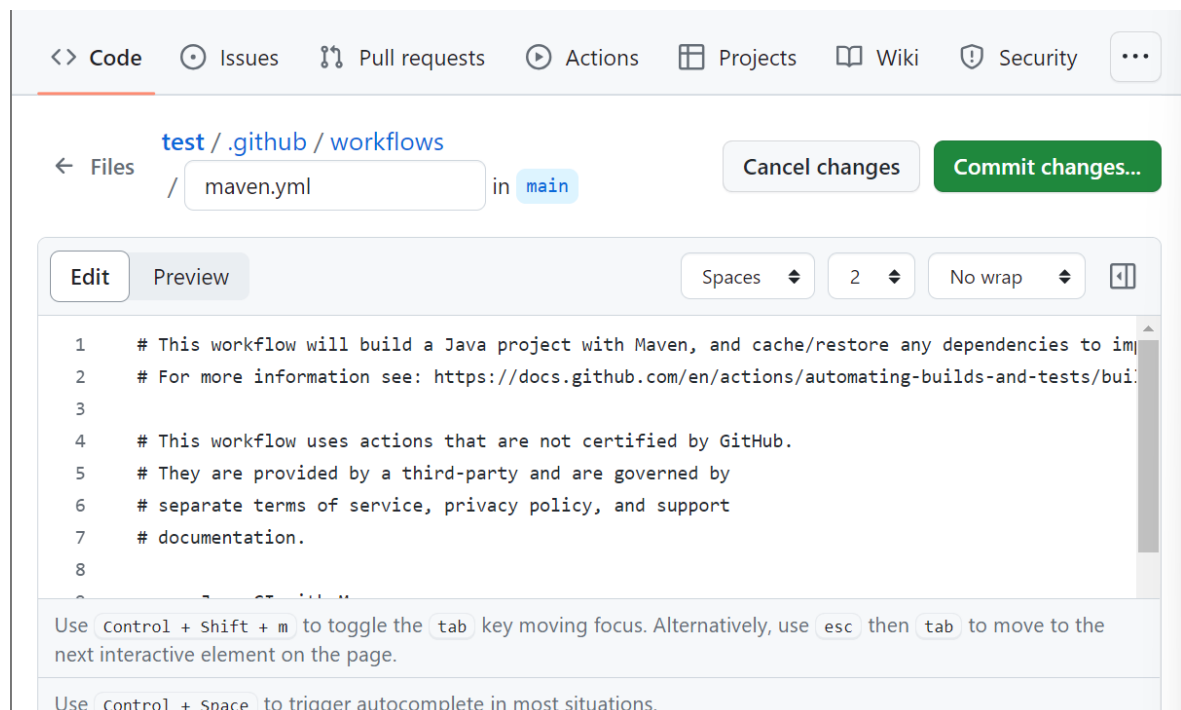
2.2.2 Các thành phần của GitHub Action.

- **Workflows:** Là một quy trình tự động một Workflow bao gồm một hoặc nhiều Job và nó sẽ kích hoạt khi nhận được một event hoặc sẽ kích hoạt trong khoảng thời gian nhất định nào đó.
- **Runner:** Là một server mà GitHub Actions cài đặt các ứng dụng trong đó để có thể thực thi các Workflow. Nó có thể do GitHub cung cấp hoặc ta có thể tự tạo cấu hình riêng.
- **Step:** Cho phép ta thực thi một câu lệnh và đồng thời cũng cho phép ta cấu hình các thông tin cần thiết cho việc thực thi câu lệnh.
- **Action:** Là một building block, có tính tự động được tích hợp sẵn, Action được tạo bởi cộng đồng người dùng hoặc có thể tự tạo riêng cho mình.
- **Job:** Là tập hợp các Step được thực hiện khi workflow được kích hoạt. Mặc định nếu có nhiều Job trong một Workflow thì các Job sẽ được chạy song song với nhau.

2.2.3 Cách sử dụng GitHub Action để kiểm thử tự động.

- Tạo workflow file.

- (1) Tạo thư mục .github/workflows trong thư mục gốc của repository trên GitHub.



Hình 2. 1

- (2) Trong thư mục tạo file với đuôi .yml, nó là nơi định nghĩa các bước cho workflow.

- **Cấu hình workflow cho kiểm thử:** Tùy vào yêu cầu mà chọn cấu hình workflow phù hợp. Ví dụ: cấu hình kiểm thử tự động sử dụng Maven trong Java.

```

1  name: Java CI with Maven
2
3  on:
4    push:
5      branches: [ "main" ]
6    pull_request:
7      branches: [ "main" ]
8
9  jobs:
10   build:
11
12     runs-on: ubuntu-latest
13
14     steps:
15       - uses: actions/checkout@v3
16
17       - name: Set up JDK 17
18         uses: actions/setup-java@v3
19         with:
20           java-version: '17'
21           distribution: 'temurin'
22           cache: maven
23
24       - name: Maven package and test
25         run: mvn -B package test
26
27       - name: Report
28         uses: dorny/test-reporter@v1
29         if: always()
30         with:
31           name: 'Test Report'
32           path: 'target/surefire-reports/TEST*.xml'
33           reporter: 'java-junit'
34           fail-on-error: 'true'
35           fail-on-empty: 'true'

```

Hình 2. 2

- **Cấu hình bộ test và thêm vào các công cụ để kiểm thử:** Cài đặt môi trường thực thi, thiết lập ngôn ngữ Java hoặc ngôn ngữ khác, trong file cấu hình và thực hiện các bước để kiểm thử.

- **Lưu và commit file cấu hình:** Lưu file cấu hình, sau đó thực hiện commit và push file lên repository trên GitHub.

2.2.4 Ưu điểm khi tích hợp Github Action vào kiểm thử tự động.

- **Tự động hóa quy trình kiểm thử:** Xác định và cấu hình các bước kiểm thử tự động mỗi khi có thay đổi trong mã nguồn, giúp tự động hóa quy trình và bỏ qua việc thực hiện kiểm thử thủ công.

- **Tích hợp dễ dàng:** GitHub Action tích hợp sâu vào GitHub, giúp quản lý việc kiểm thử và mã nguồn trong nền tảng, tạo điều kiện thuận lợi cho việc quản lý quy trình kiểm thử.

- **Hỗ trợ đa dạng nhiều ngôn ngữ và công cụ:** GitHub Action hỗ trợ nhiều ngôn ngữ và công cụ khác nhau, giúp chọn lựa ngôn ngữ và công cụ phù hợp với dự án.

- **Báo cáo kết quả chi tiết:** Cung cấp chi tiết kết quả kiểm thử, giúp theo dõi và phân tích kết quả kiểm thử dễ dàng.

- **Tăng hiệu suất và tiết kiệm thời gian:** Bằng việc phát hiện và sửa lỗi nhanh chóng giúp cải thiện hiệu suất và tiết kiệm thời gian.

- **Tính liên tục:** Kiểm thử tự động giúp xây dựng và tích hợp liên tục, đảm bảo tính ổn định và chất lượng.

2.3 Tích hợp liên tục.

2.3.1 Khái niệm.

- **Tích hợp liên tục** hay còn gọi là CI (viết tắt của Continuous Integration) là phương pháp phát triển phần mềm mà ở đó trong một nhóm các thành viên làm việc cùng nhau thường xuyên tích hợp các thay đổi vào mã nguồn chung. Mục tiêu là kiểm tra và tích hợp mã nguồn tự động, thường xuyên để đảm bảo những thay đổi được thêm vào không bị xung đột hoặc gây ra lỗi với phần mềm.

- **Một số điểm quan trọng:**

(1) Tự động hóa: Để tích hợp mã nguồn, kiểm thử và biên dịch các thay đổi mới một cách tự động, CI đã sử dụng công cụ tự động hóa.

(2) Thường xuyên tích hợp: Thay vì chỉ tích hợp một lần vào cuối quá trình phát triển, thì CI yêu cầu tích hợp liên tục, thường xuyên tích hợp các thay đổi vào mã nguồn chung.

(3) Kiểm thử tự động: Sau mỗi lần tích hợp, để đảm bảo mã nguồn vẫn hoạt động như mong đợi, CI thường thực hiện các bài kiểm thử tự động.

- (4) Xác định lỗi: Trong quá trình tích hợp khi có lỗi xuất hiện, CI sẽ giúp xác định và phát hiện lỗi.
- (5) Tăng tính ổn định và chất lượng: Tích hợp thường xuyên giúp mã nguồn ổn định và chất lượng.
- (6) Quản lý phiên bản: CI thường đi cùng với hệ thống quản lý phiên bản, giúp quản lý phiên bản và theo dõi các thay đổi hiệu quả.

2.3.2 Vai trò.

- **Giảm xung đột trong quản lý phiên bản:** Giảm xung đột và không tương thích giữa các phiên bản khác nhau, đồng thời tạo điều kiện cho việc quản lý phiên bản, bằng việc thường xuyên tích hợp mã nguồn.
- **Tăng tính ổn định và chất lượng:** Thường xuyên kiểm tra và kiểm thử tự động mã nguồn, giúp mã nguồn ổn định, chất lượng, phát hiện và sửa lỗi.
- **Hỗ trợ hợp tác nhóm:** Tạo điều kiện cho các thành viên trong nhóm hợp tác phát triển, giúp đảm bảo môi trường làm việc chung hiệu quả bằng cách cung cấp quy trình tích hợp và kiểm thử.
- **Tối ưu hóa quy trình phát triển:** Hỗ trợ quá trình phát triển, giúp triển khai ứng dụng và tăng tốc quá trình phát triển, bằng việc tạo điều kiện kiểm thử, tích hợp và phân phối.
- **Xây dựng tin cậy:** Quá trình kiểm thử tự động và tích hợp liên tục giúp xây dựng niềm tin, sự tin cậy trong việc phát triển, đặc biệt là các dự án lớn và phức tạp.
- **Tiết kiệm thời gian và chi phí:** Việc phát hiện và sửa lỗi sớm, giúp tiết kiệm thời gian, chi phí cho việc sửa chữa về sau, cũng như tối ưu quá trình kiểm thử và phát triển.

2.3.3 Cách thức hoạt động.

- **Tích hợp thường xuyên:** Các nhà phát triển thường xuyên tải lên (commit) các thay đổi vào hệ thống quản lý phiên bản.
- **Kích hoạt quá trình tích hợp liên tục:** Mỗi khi có thay đổi được commit lên repository của dự án, CI Server nhận diện sự thay đổi và kích hoạt quá trình tích hợp liên tục.

- **Tải và kiểm tra mã nguồn:** CI Server tự động tải mã nguồn mới nhất từ repository và thực hiện các bước xử lý như biên dịch, kiểm tra lỗi, và kiểm tra định dạng mã nguồn.
- **Kiểm thử tự động:** Sau khi mã nguồn thành công biên dịch, CI Server thực hiện kiểm thử tự động tùy thuộc vào cấu hình của dự án, gồm kiểm thử đơn vị, tích hợp hoặc giao diện.
- **Báo Cáo Kết Quả:** Kết quả của quá trình kiểm thử sẽ được tự động ghi lại và báo cáo. Nếu có lỗi xuất hiện, CI Server thông báo về các vấn đề cần sửa chữa.
- **Tích hợp vào nhánh chính:** Nếu quá trình kiểm thử thành công và không có lỗi, CI Server có thể tự động tích hợp (merge) thay đổi vào nhánh chính của dự án.

2.4 Unit test.

2.4.1 Khái niệm.

- Unit test là một loại kiểm thử phần mềm cho các thành phần riêng lẻ hay các đơn vị của phần mềm kiểm thử.
- Unit test kiểm thử mã nguồn như class, module, method,..kiểm thử được thực hiện trong quá trình phát triển với mục tiêu là xác minh và cô lập một phần mã nguồn.

2.4.2 Các khái niệm có liên quan khi dùng Unit Test.

- **Assertion:** Là phát biểu mô tả những công việc kiểm tra cần tiến hành. Xác định điều kiện mà kết quả đưa ra phải thỏa mãn, nếu điều kiện không đúng thì kết quả sẽ fail. Mỗi một Unit Test gồm nhiều assertion kiểm tra dữ liệu đầu ra, tính chính xác của các lỗi ngoại lệ và nhiều vấn đề khác.
- **Test Case:** Bộ sưu tập các điều kiện, trạng thái và dữ liệu đầu vào được thiết kế để kiểm tra tính đúng đắn.
- **Test Point:** Là đơn vị kiểm tra, chỉ chứa đơn giản một assertion nhằm khẳng định tính đúng đắn của mã.
- **Test Suite:** Là tập hợp các test case định nghĩa cho từng hệ thống con hoặc module.
- **Regression:** Là loại kiểm thử phần mềm để xác định những đổi mới, đảm bảo rằng việc thay đổi hay cập nhật phần mềm không làm gây hại các tính năng đã hoạt động trước đó.

- **Production code:** Là phần mã nguồn chính được viết để cung cấp chức năng và tính năng của phần mềm và ứng dụng được chuyển giao cho người dùng cuối cùng.
- **Unit Testing Code:** Là mã nguồn được viết để kiểm tra và đảm bảo tính chính xác của mã nguồn chính.

2.4.3 Vai trò của Unit test.

- **Đảm bảo tính chính xác:** Unit Test kiểm tra từng đơn vị của mã nguồn để xác định chúng hoạt động đúng hay không. Đảm bảo mọi thay đổi trong mã nguồn không làm ảnh hưởng các chức năng đã hoạt động trước đó.
- **Phát hiện sớm lỗi:** Thực hiện Unit Test sớm trong quá trình phát triển giúp phát hiện và sửa lỗi nhanh chóng, trước khi chúng trở thành vấn đề lớn hoặc ảnh hưởng đến các phần khác của mã nguồn.
- **Tạo độ tin cậy:** Unit Test cung cấp cách để đánh giá và đảm bảo rằng các phần của mã hoạt động đúng, từ đó tạo niềm tin về tính ổn định và chất lượng của mã nguồn.
- **Hỗ trợ thiết kế:** Giúp mã dễ dàng bảo trì và mở rộng sau này, Unit Test thường yêu cầu mã nguồn phải được chia nhỏ và thiết kế linh hoạt.
- **Tăng tính tự động hóa:** Unit Test có thể được thực hiện tự động hóa, giúp tăng hiệu quả trong quá trình kiểm thử và phát triển.

2.4.4 Ứng dụng Unit test.

- **Xác định phạm vi và chuẩn bị môi trường:** Xác định mã nguồn cần để kiểm tra và chuẩn bị môi trường cho việc thực hiện Unit Test.
- **Viết Test Case:** Viết Test case để kiểm tra các phần của mã nguồn.
- **Chuẩn bị dữ liệu và thực hiện:** Xác định và chuẩn bị dữ liệu đầu vào đảm bảo các trường hợp đa dạng và đặt kỳ vọng cho kết quả. Thực hiện chạy và kiểm tra kết quả thu được, xem kết quả có khớp với kỳ vọng không.
- **Bảo trì và cập nhật:** Bảo trì và cập nhật các bài Test case khi có sự thay đổi trong mã nguồn, đảm bảo phản ánh chính xác chức năng mã nguồn.
- **Tự động hóa:** Viết script để chạy các bài kiểm tra tự động, giúp tiết kiệm thời gian và tính nhất quán.
- **Tổng hợp và báo cáo:** Tổng hợp và báo cáo kết quả kiểm thử, gồm các thông tin chi tiết và lỗi phát hiện về hiệu suất của mã nguồn.

CHƯƠNG 3: HIỆN THỰC HÓA NGHIÊN CỨU

Chương này mô tả một quy trình kiểm thử tự động một dự án Java Maven với Github Action. Các unit test được lập trình để kiểm thử một chương trình đơn giản gọi là Calculator. Sau khi mã nguồn được commit lên Github, một workflow Github action sẽ được chạy để kiểm thử từng phương thức trong class Calculator bằng các unit test đã chuẩn bị trước.

3.1 Xây dựng Java Maven.

3.1.1 Tổng quan.

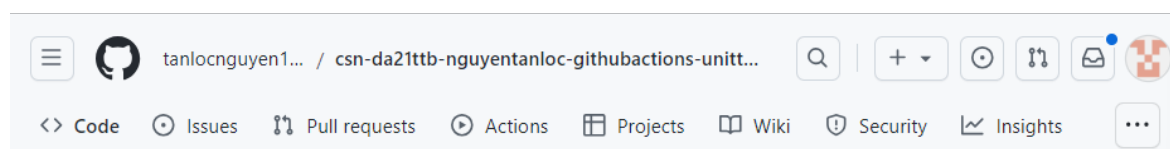
- Maven là một trong những công cụ để quản lý và xây dựng dự án khi dùng java. Nó giúp tạo cấu trúc dự án và kết nối các thư viện cần thiết để triển khai trên máy chủ.
- Maven được phát triển bằng ngôn ngữ Java cho phép chạy trên nhiều nền tảng khác nhau, nó hỗ trợ việc tự động hóa quá trình tạo dự án ban đầu, kiểm thử, đóng gói và triển khai sản phẩm.

3.1.2 Sử dụng quy trình làm việc khởi động Maven.

- Để bắt đầu, phải thêm quy trình làm việc ban đầu vào thư .github/workflows mục kho lưu trữ.
- GitHub cung cấp quy trình khởi đầu làm việc cho Maven, quy trình hoạt động với hầu hết các dự án Java Maven.
- Hướng dẫn cách tùy chỉnh quy trình làm việc ban đầu:

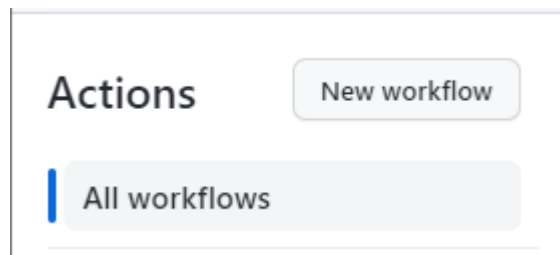
(1) Trên trang GitHub.com, mở trang chính của kho lưu trữ.

(2) Dưới tên kho lưu trữ nhấn vào Actions



Hình 3. 1

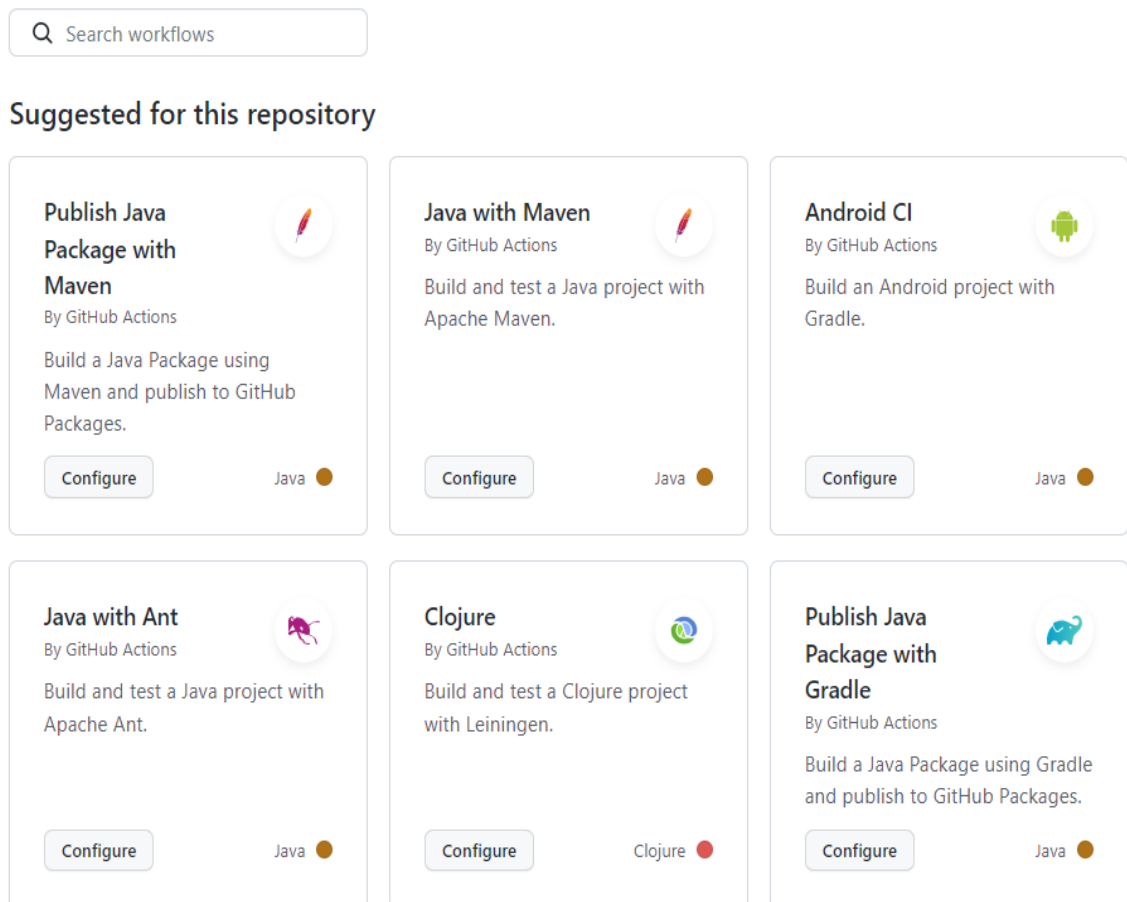
(3) Nếu đã có quy trình làm việc trong kho lưu trữ, thì nhấp vào New workflow.



Hình 3. 2

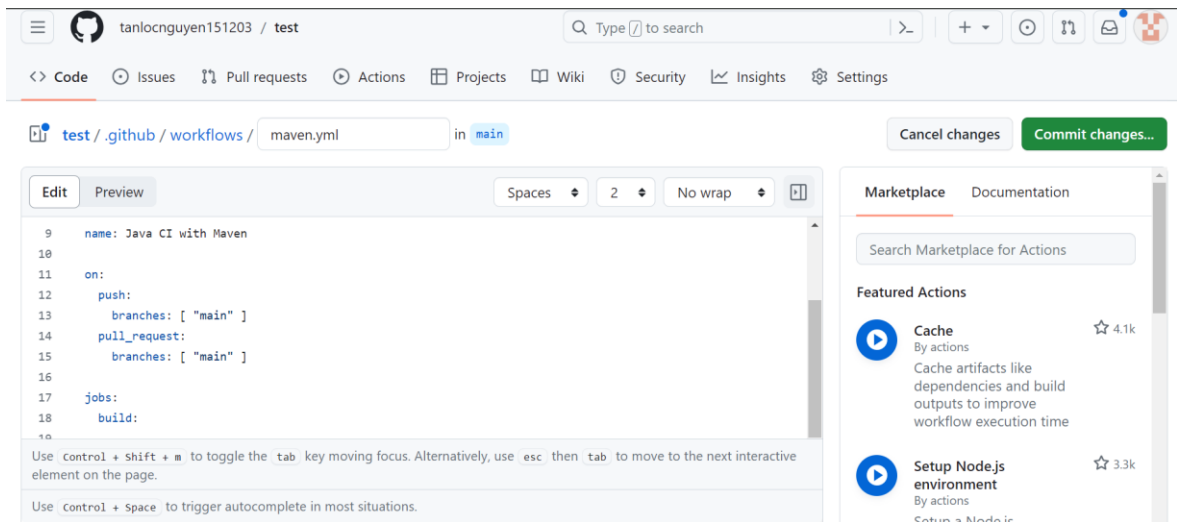
(4) Chọn quy trình công việc.

(5) Trên quy trình làm việc ‘Java with Maven’, nhấp vào để định dạng cấu hình.



Hình 3. 3

(6) Chỉnh sửa quy trình làm việc theo nhu cầu sử dụng.



Hình 3. 4

(7) Nhấp vào cam kết thay đổi (Commit changes...).



Hình 3. 5

3.2 Mô tả thuật toán.

3.2.1 Tổng quan.

- Thuật toán hay còn gọi là giải thuật, là tập hợp các hướng dẫn hoặc quy trình cụ thể để thực hiện tác vụ hoặc giải quyết vấn đề nhất định.
- Mô tả thuật toán là mô tả quy trình xử lý tác vụ hoặc quá trình giải quyết vấn đề nhất định, bằng cách sử dụng các bước logic cụ thể.

3.2.2 Phân tích.

```
package com.system;

public class Calculator {
    public int tong(int a, int b) {
        return a + b;
    }

    public int tru(int a, int b) {
        return a - b;
    }

    public int tich(int a, int b) {
        return a * b;
    }

    public int thuong(int a, int b) {
        if (b != 0)
            return a / b;
        else
            throw new IllegalArgumentException(s:"Số chia không được là số 0");
    }
}
```

Hình 3. 6. Tạo file Calculator

- **‘package com.system;’**: Khai báo package (gói) của lớp ‘Calculator’, nơi lưu trữ lớp này trong cấu trúc thư mục.
- **‘public class Calculator { ... }’**: Định nghĩa một lớp có tên là ‘Calculator’, là nơi chứa các phương thức để thực hiện các phép tính.
- **‘public int tong(int a, int b) { return a + b; }’**: Phương thức ‘tong’ nhận hai tham số ‘a’ và ‘b’, trả về tổng của chúng.
- **‘public int tru(int a, int b) { return a - b; }’**: Phương thức ‘tru’ nhận hai tham số ‘a’ và ‘b’, trả về hiệu của chúng.
- **‘public int tich(int a, int b) { return a * b; }’**: Phương thức ‘tich’ nhận hai tham số ‘a’ và ‘b’, trả về tích của chúng.
- **‘public int thuong(int a, int b) { ... }’**: Phương thức ‘thuong’ nhận hai tham số ‘a’ và ‘b’.
- **‘if (b != 0) return a / b;’**: Kiểm tra nếu ‘b’ khác 0, thực hiện phép chia ‘a’ cho ‘b’ và trả về kết quả.

- `'else throw new IllegalArgumentException("Số chia không được là số 0");'`:
Nếu 'b' bằng 0, ném ra ngoại lệ `'IllegalArgumentException'` với thông điệp "Số chia không được là số 0".

3.2.3 Viết Test Cases sử dụng Junit.

- Import các thư viện và lớp:

```
import org.junit.Assert;  
import org.junit.Ignore;  
import org.junit.Test;  
  
import com.system.Calculator;  
  
public class TestCalculator {
```

Hình 3. 7

'import': Đoạn này import các class và phần mềm thư viện cần thiết cho việc viết và thực thi test cases. Trong đoạn mã này, các class từ package `'org.junit'` và `'com.system.Calculator'` được import.

'Test', 'Ignore': Là các chú thích của JUnit để xác định các phương thức test. `'Test'` được sử dụng để đánh dấu phương thức test, trong khi đó `'Ignore'` được sử dụng để bỏ qua việc thực thi một test case cụ thể.

- Test Cases:

```
import org.junit.Assert;
import org.junit.Ignore;
import org.junit.Test;

import com.system.Calculator;

public class TestCalculator {

    @Test
    public void testTongThanhCong() {
        Calculator c = new Calculator();
        Assert.assertEquals(300, c.tong(100, 200));
    }

    // @Ignore
    @Test
    public void testTongThatBai() {
        Calculator c = new Calculator();
        Assert.assertThrows(AssertionError.class,
            () -> {
                Assert.assertEquals(400, c.tong(100, 200));
            });
    }

    @Test
    public void testTruThanhCong() {
        Calculator c = new Calculator();
        Assert.assertEquals(100, c.tru(200, 100));
    }

    @Test
    public void testTruThatBai() {
        Calculator c = new Calculator();
        Assert.assertThrows(AssertionError.class,
            () -> {
                Assert.assertEquals(400, c.tru(200, 100));
            });
    }

    @Test
    public void testTichThanhCong() {
        Calculator c = new Calculator();
        Assert.assertEquals(8, c.tich(2, 4));
    }

    @Test
    public void testTichThatBai() {
        Calculator c = new Calculator();
        Assert.assertThrows(AssertionError.class,
            () -> {
                Assert.assertEquals(6, c.tich(2, 4));
            });
    }
}
```

Hình 3. 8. Tạo file TestCalculator

- '@Test': Đánh dấu phương thức là một test case.
- '**public void testTongThanhCong()**': Một test case kiểm tra phương thức 'tong()' của lớp 'Calculator' khi gọi với hai giá trị đầu vào và so sánh với kết quả mong đợi.
- '**Calculator c = new Calculator();**': Khởi tạo một đối tượng 'Calculator'.
- '**Assert.assertEquals(300, c.tong(100, 200));**': So sánh kết quả trả về từ phương thức 'tong()' của 'Calculator' với giá trị mong đợi là '300'.
- '**Assert.assertThrows(AssertionError.class, () -> { Assert.assertEquals(400, c.tong(100, 200)); });**': Kiểm tra rằng khi gọi phương thức 'tong()' của 'Calculator' với đầu vào là '100' và '200', nó sẽ trả về '400' và phải ném ra một ngoại lệ kiểu 'AssertionError'. Điều này chỉ ra rằng kết quả trả về không khớp với giá trị mong đợi và test case sẽ fail.
- Tương tự, các phương thức kiểm thử khác nhau như 'testTruThanhCong', 'testTichThanhCong', và 'testThuongThanhCong' cũng kiểm tra các phương thức 'tru()', 'tich()', và 'thuong()' của lớp 'Calculator' khi đưa vào các giá trị đầu vào tương ứng và so sánh với kết quả mong đợi.
- Các phương thức kiểm thử như 'testTongThatBai', 'testTruThatBai', 'testTichThatBai', 'testThuongThatBai', 'testThuongThatBai2' kiểm tra các trường hợp thất bại mong đợi. Ví dụ, 'testThuongThatBai' kiểm tra việc ném ra một ngoại lệ loại 'IllegalArgumentException' khi chia một số cho 0, trong khi 'testThuongThatBai2' kiểm tra kết quả không khớp khi chia một số nguyên.

3.2.4 Đưa lên GitHub.

Sử dụng Git Command.

- Mở terminal/command prompt.
- Di chuyển đến thư mục chứa dự án bằng lệnh '**cd** đường dẫn đến thư mục'.
- Khởi tạo Git trong thư mục nếu chưa có '**git init**'
- Liên kết repository với repository trên GitHub: **git remote add origin** đường dẫn repository
- Thêm tất cả các file vào staging area '**git add**'
- Commit các thay đổi: **git commit -m** "thêm code của Calculator"

- Push code lên repository trên GitHub: **git push -u origin main**

Sử dụng GitHub Desktop.

- Tải và cài đặt GitHub Desktop. Link: <https://desktop.github.com/>
- Mở GitHub Desktop và đăng nhập tài khoản.
- Nhấp vào “Add” hoặc thêm repository và chọn thư mục muốn đưa lên.
- Đặt tên và mô tả cho repository.
- Nhấn vào Publish repository để đưa lên GitHub.

3.3 GitHub Workflows.

3.3.1 Workflows.

- Workflows là một quy trình tự động có thể định cấu hình để thực hiện một hoặc nhiều công việc. Quy trình làm việc được xác định bằng tệp YAML được đăng ký trong kho lưu trữ và sẽ chạy khi được kích hoạt bởi một sự kiện trong kho lưu trữ hoặc có thể được kích hoạt theo cách thủ công hoặc theo lịch trình xác định.
- Quy trình công việc được xác định trong `.github/workflows` thư mục trong kho lưu trữ và kho lưu trữ có thể có nhiều quy trình công việc, mỗi quy trình có thể thực hiện một nhóm tác vụ khác nhau.

3.3.2 Xây dựng và sử dụng.

- Có thể áp dụng các lệnh tương tự đã sử dụng trong quá trình xây dựng và kiểm tra mã trên máy cục bộ.
- Quy trình làm việc sẽ chạy theo mặc định. Cấu hình Maven mặc định lệnh sẽ tải xuống các phần phụ thuộc, chạy thử nghiệm, xây dựng các lớp và đóng gói các lớp thành định dạng có thể phân phối.
- Nếu sử dụng các lệnh khác nhau để xây dựng dự án hoặc sử dụng cho một mục tiêu khác, thì có thể định các lệnh đó.

3.3.3 Các phụ thuộc bộ nhớ đệm.

- Để tăng tốc độ chạy của quy trình công việc, có thể lưu vào bộ nhớ đệm các phần phụ thuộc.

- Khi chạy thành công kho lưu trữ của Maven cục bộ sẽ được lưu trữ trong bộ đệm. Như vậy, các lần chạy quy trình làm việc trong tương lai, bộ nhớ đệm sẽ được khôi phục để không cần tải xuống các phần phụ thuộc từ xa từ kho lưu trữ Maven.
- Có thể sử dụng các phần phụ thuộc bằng cách sử dụng cache action hoặc setup-java action cho cấu hình nâng cao và tùy chỉnh.

```

steps:
  - uses: actions/checkout@v3

  - name: Set up JDK 17
    uses: actions/setup-java@v3
    with:
      java-version: '17'
      distribution: 'temurin'
      cache: maven

  - name: Maven package and test
    run: |
      cd scr/ProjectJava
      mvn -B package test
    
```

Hình 3. 9

3.3.4 Đóng gói dữ liệu quy trình làm việc.

- Khi ứng dụng, thử nghiệm đã thành công và vượt qua, dưới dạng tạo sản phẩm bản dựng có thể tải lên các gói Java kết quả. Điều này giúp lưu trữ các gói đã được xây dựng như một phần quy trình làm việc và cho phép tải xuống.
- Các thành phần cấu thành phần mềm có thể giúp kiểm tra và gỡ lỗi các yêu cầu kéo trong môi trường cục bộ trước khi chúng được hợp nhất.
- Maven thường sẽ tạo các tệp đầu ra như JAR, EAR hoặc WAR trong target thư mục, để tải chúng lên dưới dạng tạo phẩm, có thể sao chép chúng vào thư mục mới chứa các tạo phẩm để tải lên.

```

steps:
- uses: actions/checkout@v3

- name: Set up JDK 17
  uses: actions/setup-java@v3
  with:
    java-version: '17'
    distribution: 'temurin'
    cache: maven

- name: Maven package and test
  run: |
    cd scr/ProjectJava
    mvn -B package test

- name: Report
  uses: dorny/test-reporter@v1
  if: always()
  with:
    name: 'Test Report'
    path: 'scr/ProjectJava/target/surefire-reports/TEST*.xml'
    reporter: 'java-junit'
    fail-on-error: 'true'
    fail-on-empty: 'true'

```

Hình 3. 10

3.3.5 Tạo Workflows.

```

.github > workflows > maven.yml
1  name: Java CI with Maven
2
3  on:
4    push:
5      branches: [ "main" ]
6    pull_request:
7      branches: [ "main" ]
8
9  jobs:
10   build:
11
12   runs-on: ubuntu-latest

```

Hình 3. 11

- **‘name: Java CI with Maven’** : ‘name’ là định dạng cho workflow, có thể sử dụng để xác định tên của quy trình CI/CD. Trong đoạn trên tên workflow là “Java CI with Maven”.

- **‘on:’** Dùng để khai báo sự kiện mà workflow sẽ được kích hoạt.

- **‘push:’** Dùng để định nghĩa một sự kiện push vào repository.
- **‘branches: [“main”]’** Xác định nhánh “main” khi có push sẽ kích hoạt workflow.
- **‘pull_request:’** Định nghĩa một sự kiện pull request mở.
- **‘branches: ["main"]’** Xác định những pull request từ nhánh "main" sẽ kích hoạt workflow.
- **‘jobs:’** Định nghĩa danh sách các công việc trong workflow.
- **‘build:’** Tên công việc trong workflow, ở đây là công việc để xây dựng mã nguồn.
- **‘runs-on: ubuntu-latest’** Cấu hình cho công việc, xác định công việc sẽ chạy trên một máy ảo với hệ điều hành Ubuntu phiên bản mới nhất.

```

steps:
- uses: actions/checkout@v3

- name: Set up JDK 17
  uses: actions/setup-java@v3
  with:
    java-version: '17'
    distribution: 'temurin'
    cache: maven

- name: Maven package and test
  run: mvn -B package test

- name: Report
  uses: dorny/test-reporter@v1
  if: always()
  with:
    name: 'Test Report'
    path: 'target/surefire-reports/TEST*.xml'
    reporter: 'java-junit'
    fail-on-error: 'true'
    fail-on-empty: 'true'
  
```

Hình 3. 12

- **‘steps:’** Khai báo danh sách các bước sẽ được thực hiện trong workflow.
- **‘uses:actions/checkout@v3’** Bước đầu tiên sử dụng action ‘actions/checkout@v3’ để checkout mã nguồn từ repository.
- **‘uses:’** Sử dụng action từ thư viện hành động của GitHub.

‘@v3’ Phiên bản cụ thể của action được sử dụng là v3.

- ‘**name: Set up JDK 17**’

‘**name:**’ Tên của bước kế tiếp trong quy trình.

‘**uses:**’ Sử dụng action ‘actions/setup-java@v3’ để cài đặt JDK.

‘**with:**’ Các tham số được cung cấp cho action.

‘**java-version: '17'**’ Xác định phiên bản Java là 17.

‘**distribution: 'temurin'**’: Sử dụng bản phân phối Temurin của Java.

‘**cache: maven**’: Lưu trữ cache của Maven.

- ‘**name: Maven package and test**’

‘**name:**’ Tên của bước để thực thi lệnh Maven package và test.

‘**run:**’ Thực thi lệnh Maven ‘mvn -B package test’ để đóng gói ứng dụng và chạy các bài kiểm tra.

- ‘**name: Report**’

‘**name:**’ Tên của bước để tạo báo cáo.

‘**uses:**’ Sử dụng action ‘dorny/test-reporter@v1’ để tạo báo cáo.

‘**if: always()**’: Điều kiện để chạy bước này luôn luôn.

‘**with:**’: Các tham số cấu hình cho action.

‘**name: 'Test Report'**’: Tên của báo cáo.

‘**path: 'target/surefire-reports/TEST*.xml'**’: Đường dẫn đến các tệp báo cáo trong thư mục ‘target/surefire-reports’.

‘**reporter: 'java-junit'**’: Loại báo cáo là ‘java-junit’.

‘**fail-on-error: 'true'**’: Báo lỗi nếu có lỗi xảy ra trong quá trình tạo báo cáo.

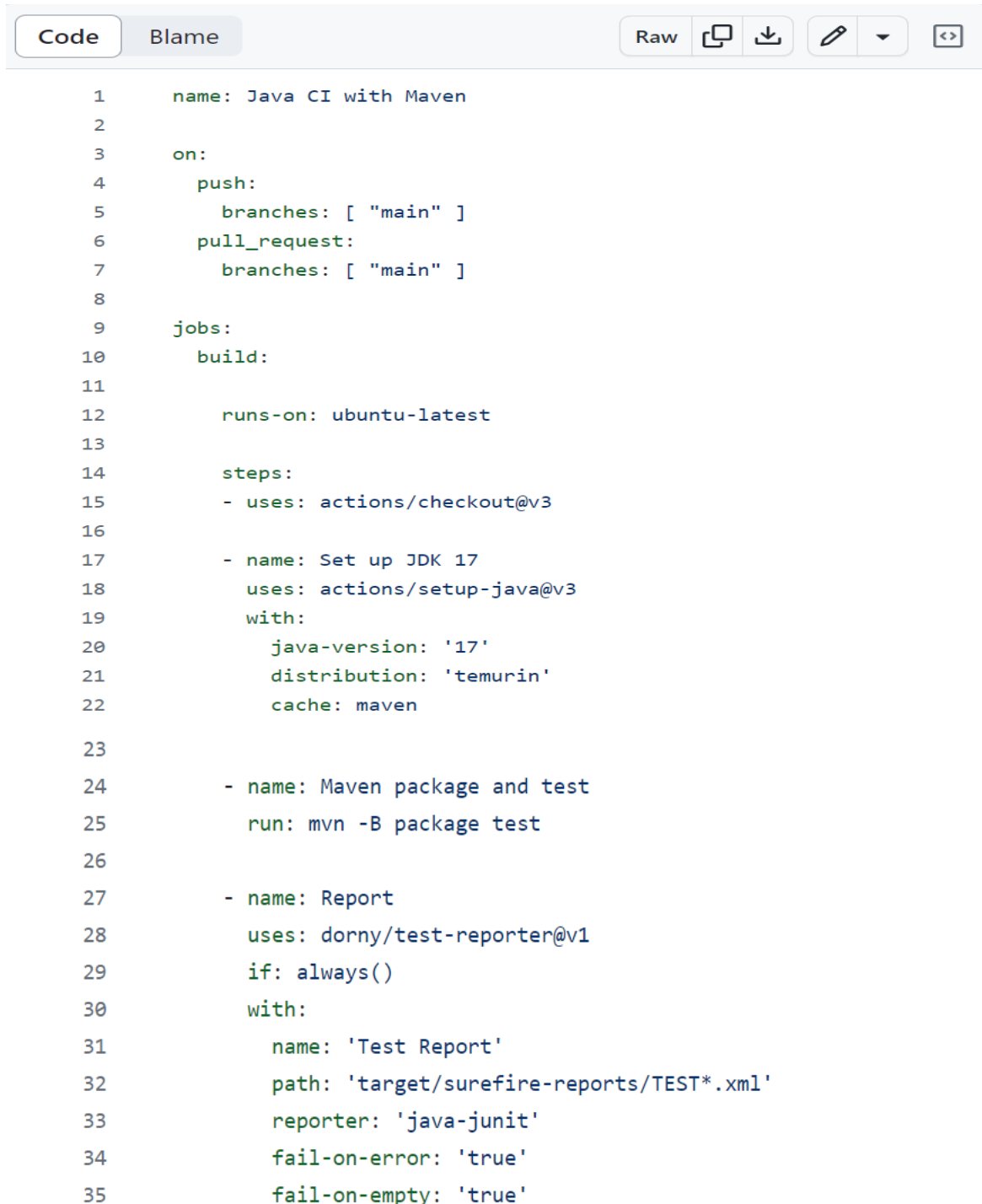
‘**fail-on-empty: 'true'**’: Báo lỗi nếu không có báo cáo nào được tạo.


```
.github > workflows >  maven.yml
1  name: Java CI with Maven
2
3  on:
4    push:
5      branches: [ "main" ]
6    pull_request:
7      branches: [ "main" ]
8
9  jobs:
10   build:
11
12     runs-on: ubuntu-latest
13
14     steps:
15       - uses: actions/checkout@v3
16
17       - name: Set up JDK 17
18         uses: actions/setup-java@v3
19         with:
20           java-version: '17'
21           distribution: 'temurin'
22           cache: maven
23
24       - name: Maven package and test
25         run: mvn -B package test
26
27       - name: Report
28         uses: dorny/test-reporter@v1
29         if: always()
30         with:
31           name: 'Test Report'
32           path: 'target/surefire-reports/TEST*.xml'
33           reporter: 'java-junit'
34           fail-on-error: 'true'
35           fail-on-empty: 'true'
36
```

Hình 3. 13. Tạo file maven.

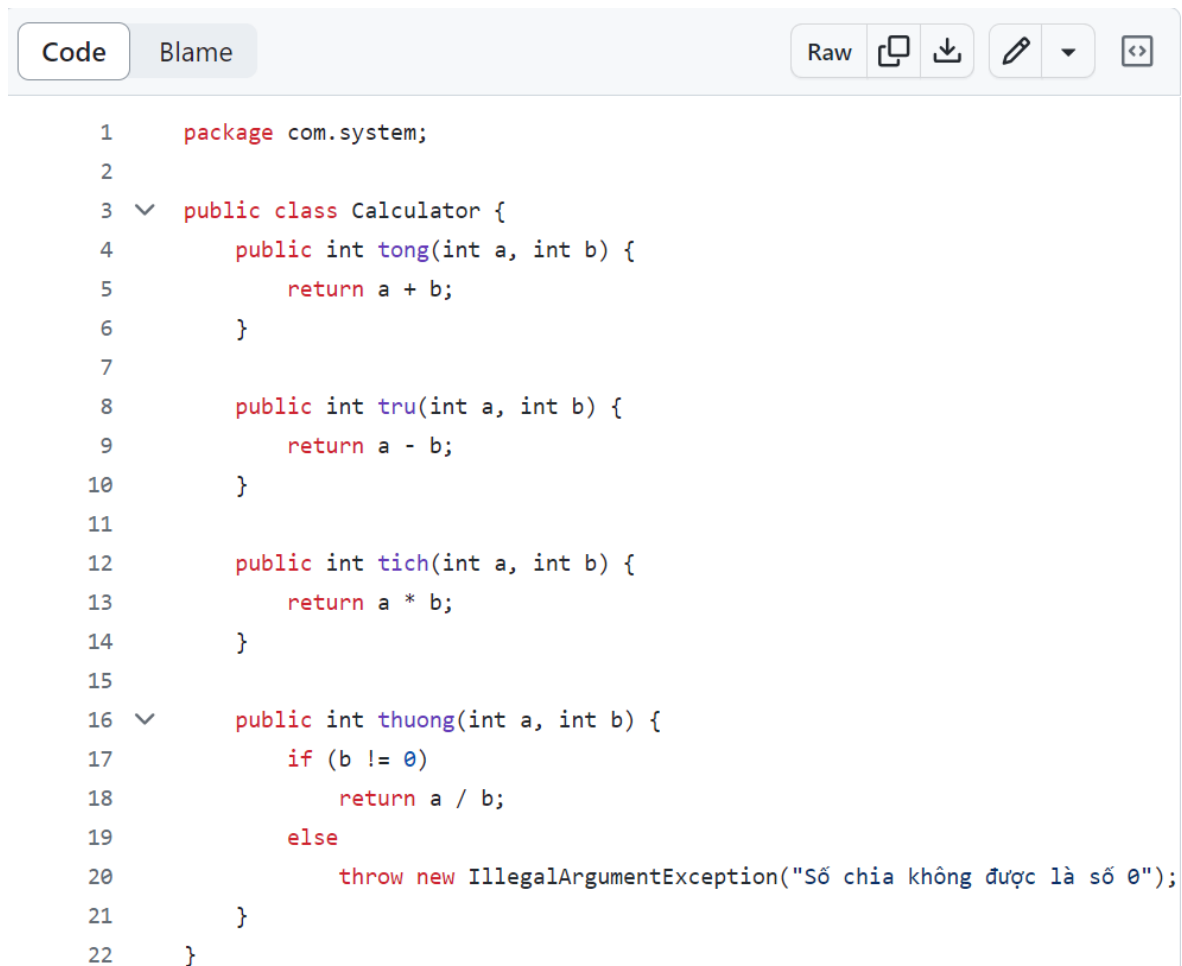
CHƯƠNG 4: KẾT QUẢ NGHIÊN CỨU

Hoàn tất việc nghiên cứu và sử dụng Github Action để kiểm thử tự động, cho ra thông báo kết quả các trường hợp kiểm thử là đúng hoặc sai.

The image shows a screenshot of a GitHub Actions workflow file named 'maven.yml'. The interface includes a header with 'Code' and 'Blame' tabs, and a toolbar with icons for 'Raw', copy, download, edit, and a dropdown menu. The workflow is displayed as a code block with line numbers from 1 to 35. It defines a job named 'build' that runs on 'ubuntu-latest'. The job includes three steps: 1) 'checkout@v3' to checkout the repository, 2) 'Set up JDK 17' using 'actions/setup-java@v3' with 'java-version: 17', 'distribution: temurin', and 'cache: maven', and 3) 'Maven package and test' using 'mvn -B package test'. Additionally, there is a 'Test Report' step using 'dorny/test-reporter@v1' that runs 'mvn -B test' and generates a report named 'Test Report' at the path 'target/surefire-reports/TEST*.xml' using the 'java-junit' reporter. The 'fail-on-error' and 'fail-on-empty' options are both set to 'true'.

```
1  name: Java CI with Maven
2
3  on:
4    push:
5      branches: [ "main" ]
6    pull_request:
7      branches: [ "main" ]
8
9  jobs:
10   build:
11
12     runs-on: ubuntu-latest
13
14     steps:
15     - uses: actions/checkout@v3
16
17     - name: Set up JDK 17
18       uses: actions/setup-java@v3
19       with:
20         java-version: '17'
21         distribution: 'temurin'
22         cache: maven
23
24     - name: Maven package and test
25       run: mvn -B package test
26
27     - name: Report
28       uses: dorny/test-reporter@v1
29       if: always()
30       with:
31         name: 'Test Report'
32         path: 'target/surefire-reports/TEST*.xml'
33         reporter: 'java-junit'
34         fail-on-error: 'true'
35         fail-on-empty: 'true'
```

Hình 4. 1. File Java Maven (maven.yml).



```

1      package com.system;
2
3      public class Calculator {
4          public int tong(int a, int b) {
5              return a + b;
6          }
7
8          public int tru(int a, int b) {
9              return a - b;
10         }
11
12         public int tich(int a, int b) {
13             return a * b;
14         }
15
16         public int thuong(int a, int b) {
17             if (b != 0)
18                 return a / b;
19             else
20                 throw new IllegalArgumentException("Số chia không được là số 0");
21         }
22     }

```

Hình 4. 2. File Class (Calculator.java).

Code Blame Raw Copy Download Edit Dropdown Expand

```
1  import org.junit.Assert;
2  import org.junit.Ignore;
3  import org.junit.Test;
4
5  import com.system.Calculator;
6
7  ✓ public class TestCalculator {
8
9      @Test
10     public void testTongThanhCong() {
11         Calculator c = new Calculator();
12         Assert.assertEquals(300, c.tong(100, 200));
13     }
14
15     // @Ignore
16     @Test
17     ✓ public void testTongThatBai() {
18         Calculator c = new Calculator();
19         Assert.assertThrows(AssertionError.class,
20             () -> {
21                 Assert.assertEquals(400, c.tong(100, 200));
22             });
23     }
24
25     @Test
26     public void testTruThanhCong() {
27         Calculator c = new Calculator();
28         Assert.assertEquals(100, c.tru(200, 100));
29     }
30
31     @Test
32     ✓ public void testTruThatBai() {
33         Calculator c = new Calculator();
34         // Assert.assertThrows(AssertionError.class,
35         // () -> {
36             Assert.assertEquals(400, c.tru(200, 100));
37         // });
38     }
39
40     @Test
41     public void testTichThanhCong() {
42         Calculator c = new Calculator();
43         Assert.assertEquals(8, c.tich(2, 4));
44     }
45
```

```
46     @Test
47     ✓ public void testTichThatBai() {
48         Calculator c = new Calculator();
49         Assert.assertThrows(AssertionError.class,
50             () -> {
51                 Assert.assertEquals(6, c.tich(2, 4));
52             });
53     }
54
55     @Test
56     public void testThuongThanhCong() {
57         Calculator c = new Calculator();
58         Assert.assertEquals(3, c.thuong(6, 2));
59     }
60
61     @Test
62     ✓ public void testThuongThatBai() {
63         int a = 4;
64         int b = 0;
65         Calculator c = new Calculator();
66         Assert.assertThrows(IllegalArgumentException.class,
67             () -> {
68                 c.thuong(a, b);
69             });
70     }
71
72     @Test
73     ✓ public void testThuongThatBai2() {
74         int a = 4;
75         int b = 4;
76
77         Calculator c = new Calculator();
78         Assert.assertThrows(AssertionError.class,
79             () -> {
80                 Assert.assertEquals(5, c.thuong(a, b));
81             });
82     }
83
84 }
```

Hình 4. 3. File Test (TestCalculator.java).

Kết quả kiểm thử:

- Đoạn mã đúng:

```

@Test
public void testTruThanhCong() {
    Calculator c = new Calculator();
    Assert.assertEquals(100, c.tru(200, 100));
}

@Test
public void testTruThatBai() {
    Calculator c = new Calculator();
    Assert.assertThrows(AssertionError.class,
        () -> {
            Assert.assertEquals(400, c.tru(200, 100));
        });
}

@Test
public void testTichThanhCong() {
    Calculator c = new Calculator();
    Assert.assertEquals(8, c.tich(2, 4));
}

@Test
public void testTichThatBai() {
    Calculator c = new Calculator();
    Assert.assertThrows(AssertionError.class,
        () -> {
            Assert.assertEquals(6, c.tich(2, 4));
        });
}

```

```

T E S T S
-----
Running TestCalculator
Tests run: 9, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.046 sec

Results :

Tests run: 9, Failures: 0, Errors: 0, Skipped: 0

```

Hình 4. 4. Kết quả kiểm thử tất cả điều kiện đều đúng.

- Đoạn mã đúng:

```
@Test
public void testTongThanhCong() {
    Calculator c = new Calculator();
    Assert.assertEquals(300, c.tong(100, 200));
}
```

- Đoạn mã được bỏ qua kiểm thử:

```
@Ignore
@Test
public void testTongThatBai() {
    Calculator c = new Calculator();
    Assert.assertThrows(AssertionError.class,
        () -> {
            Assert.assertEquals(400, c.tong(100, 200));
        });
}
```

```
-----
T E S T S
-----

Running TestCalculator
Tests run: 9, Failures: 0, Errors: 0, Skipped: 1, Time elapsed: 0.045 sec

Results :

Tests run: 9, Failures: 0, Errors: 0, Skipped: 1
```

Hình 4. 5. Kết quả kiểm thử các điều kiện đều đúng và một điều kiện được bỏ qua.

- Đoạn mã sai:

```
@Test
public void testTongThatBai() {
    Calculator c = new Calculator();

    Assert.assertEquals(400, c.tong(100, 200));
}
```

- Đoạn mã đúng:

```
@Test
public void testTongThanhCong() {
    Calculator c = new Calculator();
    Assert.assertEquals(300, c.tong(100, 200));
}
```

```

-----
T E S T S
-----

Running TestCalculator
Tests run: 8, Failures: 1, Errors: 0, Skipped: 0, Time elapsed: 0.055 sec <<<
FAILURE!
testTruThatBai(TestCalculator) Time elapsed: 0.004 sec <<< FAILURE!
java.lang.AssertionError: expected:<400> but was:<100>
    at org.junit.Assert.fail(Assert.java:89)
    at org.junit.Assert.failNotEquals(Assert.java:835)
    at org.junit.Assert.assertEquals(Assert.java:647)
    at org.junit.Assert.assertEquals(Assert.java:633)
    at TestCalculator.testTruThatBai(TestCalculator.java:36)

```

Hình 4. 6. Kết quả kiểm thử các điều kiện đều đúng và một điều kiện sai.

Kết quả nghiên cứu cho thấy việc kiểm thử tự động giúp tự động hóa quy trình kiểm thử, đảm bảo vẫn hoạt động đúng sau mỗi lần thay đổi, cải thiện chất lượng dự án và độ đáng tin cậy của mã nguồn, giúp tiết kiệm thời gian và công sức.

CHƯƠNG 5: KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

Kết luận:

- Kết quả đạt được: Thông qua nghiên cứu GitHub Action có thể tự động hóa quy trình kiểm thử mã nguồn, thực hiện kiểm thử tự động khi có thay đổi trong mã. Giảm thiểu lỗi và tăng cường chất lượng bằng cách phát hiện lỗi sớm thông qua kiểm thử tự động. Tiết kiệm thời gian và tăng hiệu suất khi sử dụng kiểm thử tự động.
- Đóng góp mới: Báo cáo nghiên cứu có thể được dùng làm tài liệu tham khảo, hướng dẫn về kiểm thử tự động.
- Đề xuất mới: Các công cụ kiểm thử tự động thường rất phức tạp, hiện giờ ta có thể đơn giản hóa việc kiểm thử. Nếu ta đã setup được một workflow trên GitHub Action thì ta chỉ cần commit để chạy lên thì ngay lập tức biết được ứng dụng, phần mềm là đúng hay sai. thiện và nghiên cứu thêm về kiểm thử để nâng cao quá trình kiểm thử tự động.
- Kết luận: Việc nghiên cứu GitHub Action kiểm thử tự động giúp tiết kiệm thời gian cho việc kiểm thử, cải thiện tính ổn định và đáng tin cậy, tăng cường chất lượng và hiệu suất phát triển.

Hướng phát triển:

Hiện tại chỉ mới kiểm thử bằng Java Maven với test case sử dụng Unit test nhưng trong tương lai có thể sử dụng Integration test, System test, Acceptance test, End-to-end test,... và kiểm thử tự động cho mã nguồn Python, C, C#, HTML,... Tích hợp với các công cụ kiểm thử khác như Selenium, Cypress,... Tăng cường khả năng kiểm thử tự động, cải thiện chất lượng ứng dụng và phần mềm.

DANH MỤC TÀI LIỆU THAM KHẢO

<https://docs.github.com/en/actions/automating-builds-and-tests/building-and-testing-java-with-maven>
<https://viblo.asia/p/cicd-pipeline-4P856OXOKY3>
<https://blog.japan-itworks.vn/vi/github-actions-la-gi-1404>
<https://viettuts.vn/junit>
<https://github.com/marketplace/actions/test-reporter#supported-formats>

PHỤ LỤC