ĐẠI HỌC QUỐC GIA HÀ NỘI
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ

NGUYỄN THỊ HỒNG NHUNG

NGHIÊN CỨU SINH MÃ KIỂM THỬ TỰ ĐỘNG DỰA TRÊN KỊCH BẢN KIỂM THỬ HƯỚNG HÀNH VI

LUẬN VĂN THẠC SĨ

Ngành: Kỹ thuật phần mềm

HÀ NỘI- 2018

ĐẠI HỌC QUỐC GIA HÀ NỘI
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ

NGUYỄN THỊ HỒNG NHUNG

NGHIÊN CỨU SINH MÃ KIỂM THỬ TỰ ĐỘNG DỰA TRÊN KỊCH BẢN KIỂM THỬ HƯỚNG HÀNH VI

Ngành: Công nghệ thông tin
Chuyên ngành: Kỹ thuật phần mềm
Mã số: 8480103.01

LUẬN VĂN THẠC SỸ

Ngành: Kỹ thuật phần mềm

NGƯỜI HƯỚNG DẪN KHOA HỌC:
PGS.TS TRƯƠNG ANH HOÀNG

HÀ NỘI - 2018

TÓM TẮT

Tóm tắt: Trong lĩnh vực làm phần mềm ngày càng có nhiều công việc được tự động hóa. Kiểm thử là một trong những giai đoạn làm phần mềm tốn nhiều chi phí cũng như nguồn nhân lực của các chuyên gia, kỹ sư đảm bảo chất lượng phần mềm, làm sao để tự động và đơn giản hóa quá trình kiểm thử đang là một vấn đề được quan tâm. Luận văn tập trung nghiên cứu về kiểm thử tự động hướng hành vi, nghĩa là kiểm thử chấp nhận trên hành vi của người dùng. Từ việc nghiên cứu sinh mã kiểm thử động của các công cụ kiểm thử tự động luận văn tìm hiểu về việc ứng dụng các kĩ thuật kiểm thử tự động trong kiểm thử dựa trên kịch bản kiểm thử hướng hành vi. Bằng việc thực nghiệm công cụ kiểm thử với các kịch bản dưới dạng ngôn ngữ tự nhiên có cấu trúc, từ kết quả kiểm thử tự động, luận văn đưa ra đánh giá, nhận xét và đề xuất phương pháp cải tiến công cụ tự động trong kiểm thử phần mềm hướng hành vi.

Từ khóa*: kiểm thử, kiểm thử tự động, kiểm thử hướng thành phần, kiểm thử hướng hành vi.*

LỜI CẢM ƠN

Trước tiên tôi xin dành lời cảm ơn chân thành nhất đến thầy giáo, PGS. TS. Trương Anh Hoàng – Thầy đã giúp tôi định hướng trong quá trình nghiên cứu cũng như học tập tại khoa CNTT, đồng thời thầy cũng là người hướng dẫn, khích lệ tôi trong quá trình học tập và hoàn thành luận văn của mình.

Tôi xin gửi lời cảm ơn tới các thầy cô giáo khoa Công nghệ thông tin, trường Đại học Công nghệ, ĐHQGHN đã đào tạo, cung cấp cho tôi những kiến thức tôi trong suốt quá trình học tập, nghiên cứu tại trường.

Tôi cũng xin cảm ơn tất cả những người thân yêu trong gia đình tôi cùng toàn thể bạn bè, đồng nghiệp tại khoa CNTT, trường ĐH CNVT những người đã giúp đỡ, động viên, tạo điều kiện thuận lợi cho tôi học tập và nghiên cứu chương trình thạc sĩ tại Đại học Công nghệ, ĐH QGHN.

Tôi xin chân thành cảm ơn!

Học viên thực hiện

Nguyễn Thị Hồng Nhung

LỜI CAM ĐOAN

Tôi xin cam đoan đề tài nghiên cứu sinh mã kiểm thử tự động dựa trên kiểm thử hướng hành vi được trình bày trong luận văn là do tôi thực hiện dưới sự hướng dẫn của PGS.TS Trương Anh Hoàng, không sao chép bất kì kết quả nghiên cứu của các tác giả khác. Nội dung trong luận văn có tham khảo một số tài liệu và sử dụng nguồn từ các bài viết, tạp chí đều đã được nêu đầy đủ trong mục tài liệu tham khảo.

Hà Nội, ngày tháng năm 2018

Học viên thực hiện

Nguyễn Thị Hồng Nhung

MỤC LỤC

LỜI CẢM ƠN ii
LỜI CAM ĐOAN iii
MỤC LỤC iv
DANH MỤC TỪ VIẾT TẮT vi
DANH MỤC BẢNG BIỂU vii
DANH MỤC HÌNH VẼ viii
CHƯƠNG 1: MỞ ĐẦU 1
1.1 Khái quát vấn đề 1
1.2 Giải pháp 2
1.3 Bố cục luận văn 3
CHƯƠNG 2: MỘT SỐ KIẾN THỨC NỀN TẢNG 5
2.1 Phát triển phần mềm dựa trên phương pháp Agile 5
2.2 Phát triển phần mềm hướng kiểm thử (TDD) 6
2.3 Phát triển hướng BDD 8
2.4 Xử lý ngôn ngữ tự nhiên 11
2.5 Khái quát về tự động kiểm thử trong BDD 12
CHƯƠNG 3: MỘT SỐ CÔNG CỤ KIỂM THỬ TỰ ĐỘNG HƯỚNG HÀNH VI 13
3.1 Công cụ kiểm thử Cucumber 13
3.2 Công cụ kiểm thử Jasmine 14
3.3 Công cụ kiểm thử Rspec 19
CHƯƠNG 4: THỰC NGHIỆM FRAMEWORK KIỂM THỬ TỰ ĐỘNG VÀ ĐÁNH GIÁ 21
4.1 Các thành phần của Framework kiểm thử sử dụng Cucumber 21
4.1.1 Công nghệ Java 21
4.1.2 Selenium Webdriver 28
4.1.3 Cucumber 31
4.2. Báo cáo kết quả kiểm thử 38
4.3 Đánh giá Framework kiểm thử 41

4.4 Phương pháp sinh mã kiểm thử tự động 42
KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN 52
TÀI LIỆU THAM KHẢO 54

DANH MỤC TỪ VIẾT TẮT

Ký hiệu	Dạng đầy đủ
BDD	Behavior Driven development – Phát triển phần mềm hướng hành vi
NLP	Natural Language Processing – Xử lý ngôn ngữ tự nhiên
TDD	Test Driven Development Phát triển phần mềm hướng kiểm thử

DANH MỤC BẢNG BIỂU

Bảng 3-1 Ngôn ngữ Gherkin..... 14
Bảng 4-1 Các câu lệnh thường dùng trong Selenium Webdriver..... 30
Bảng 4-3 Thời gian chạy 1 kịch bản kiểm thử tự động..... 41
Bảng 4-4 Mô tả phương pháp sinh thân hàm cho kịch bản đăng nhập trên trang stackoverflow.com..... 49
Bảng 4-5 Mô tả phương pháp sinh thân hàm cho kịch bản đăng nhập vào trang web demo..... 50

DANH MỤC HÌNH VẼ

Hình 1-1 Tiến trình sinh mã kiểm thử tự động dựa trên kịch bản BDD 4
Hình 2-1 Quy trình TDD 6
Hình 2-2 Mô hình BDD – TDD được mô tả trong Agile bởi Paul LittleBury 9
Hình 3-1 Kiểm thử với Jasmine 16
Hình 3-2 Chạy Kịch bản kiểm thử trên Jasmine..... 18
Hình 3-3 Chạy ca kiểm thử khi hàm có lỗi 19
Hình 3-4 Cấu trúc các bước kiểm thử của Rspec 20
Hình 4-1 Các thành phần Framework kiểm thử 21
Hình 4-2 Tìm kiếm thư viện trong Eclipse..... 22
Hình 4-3 Cài đặt thư viện Maven trong Eclipse..... 23
Hình 4-5 Cấu trúc 1 dự án Maven..... 24
Hình 4-6 Kiểm tra cài đặt các thư viện trong file pom.xml..... 28
Hình 4-7 Các Hooks trong Cucumber..... 29
Hình 4-8 Quy trình kiểm thử với Framework Cucumber 31
Hình 4-9 Các steps được sinh ra từ Feature file..... 33
Hình 4-10 Chạy kịch bản kiểm thử..... 37
Hình 4-11 Thực thi ca kịch bản kiểm thử trên nền web 37
Hình 4-12 Cấu trúc thư mục sinh báo cáo trong kiểm thử..... 39
Hình 4-13 Cấu trúc sinh bảng báo cáo từ các feature file..... 39
Hình 4-14 Cấu hình để sinh báo cáo kiểm thử 40
Hình 4-15 Báo cáo kiểm thử dưới dạng report.html..... 41
Hình 4-16 Báo cáo kiểm thử dưới dạng html 41

CHƯƠNG 1: MỞ ĐẦU

Ứng dụng tự động hoá trong nhiều lĩnh vực ngày càng phát triển. Trong lĩnh vực làm phần mềm càng lúc lại càng được tự động hóa nhiều hơn. Kiểm thử là một trong những giai đoạn làm phần mềm, làm sao để tự động và đơn giản hóa quá trình kiểm thử đang là một vấn đề được quan tâm. Luận văn tập trung nghiên cứu về kiểm thử động hướng hành vi. Từ việc nghiên cứu sinh mã kiểm thử tự động của các công cụ kiểm thử tự động, luận văn áp dụng ứng dụng công cụ vào kiểm thử và tìm hiểu phương pháp xây dựng công cụ kiểm thử tự động trong kiểm thử hướng hành vi.

1.1 Khái quát vấn đề

Kiểm thử phần mềm theo hướng kiểm thử hướng hành vi (BDD – Behavior Driven Testing) [2] là một khái niệm mở rộng của TDD (Test Driven Development). TDD là khái niệm kiểm thử theo hướng kiểm thử từng phần, viết ca kiểm thử [12] trước rồi lập trình sau. Trong kiểm thử hướng hành vi dựa trên yêu cầu của người sử dụng chúng ta xây dựng các ca kiểm thử.

Một trong những điểm yếu khi lập trình phần mềm là rất khó để đáp ứng được đúng yêu cầu của người sử dụng, có nhiều nguyên nhân, trong đó có nguyên nhân là người lập trình không hiểu được yêu cầu người dùng. Vì vậy nếu sử dụng được trực tiếp ngôn ngữ của người sử dụng vào các ca kiểm thử thì sẽ có được một phần mềm đúng với yêu cầu người sử dụng mà không lãng phí nhiều tài nguyên. Đồng thời kiểm thử tự động với mã kiểm thử hướng hành vi sẽ giúp tiết kiệm nguồn nhân lực kiểm thử, giảm thiểu tối đa chi phí làm phần mềm, sản phẩm. Qua tìm hiểu một số công cụ kiểm thử tự động trong kiểm thử hướng hành vi hiện nay sinh phương thức kiểm thử nhưng không sinh thân hàm kiểm thử. Ví dụ với công cụ kiểm thử tự động Cucumber, Từ kịch bản kiểm thử ngôn ngữ tự nhiên:

```
Feature: search Wikipedia
  Scenario: direct search article
    Given Enter search term 'Cucumber'
    When Do search
    Then Single result is shown for 'Cucumber'
```

Thông qua công cụ kiểm thử tự động, các phương thức được sinh ra như sau:

```
@Given ("^Enter search term 'Cucumber'$")
    public void enter_search_term_Cucumber() throws Throwable {
        // Write code here that turns the phrase above into concrete actions

    }
    @When("^Do search$")
    public void do_search() throws Throwable {
        // Write code here that turns the phrase above into concrete actions
    }
    @Then("^Single result is shown for 'Cucumber'$")
    public void single_result_is_shown_for_Cucumber() throws Throwable {
        // Write code here that turns the phrase above into concrete actions
    }
```

Tuy nhiên, công cụ tự động hiện tại không sinh được thân hàm do vậy cần đưa phương án để sinh thân các phương thức để tự động kiểm thử với kịch bản kiểm thử BDD đã có.

1.2 Giải pháp

Giải pháp luận văn đưa ra là nghiên cứu sinh các mã kiểm thử tự động từ kịch bản kiểm thử hướng hành vi. Đồng thời sử dụng framework Cucumber kết hợp tích hợp các công cụ để kiểm thử tự động trong kiểm thử hướng hành vi dựa trên viết mã kịch bản kiểm thử bằng ngôn ngữ tự nhiên có cấu trúc. Từ việc nghiên cứu quy trình xử lý ngôn ngữ tự nhiên dựa trên NLP (Natural Language

Processing) luận văn cũng đưa ra và phân tích tự động hóa trong kiểm thử hướng hành vi. Với một từ điển các ca kiểm thử trên ngôn ngữ tự nhiên, từ dữ liệu đó có một hành vi tương ứng khi đọc đến ca kiểm thử hành vi đối với phần mềm đang xây dựng. Khi đó, mỗi lúc gọi đến kịch bản kiểm thử sẽ dẫn đến hành vi xử lý trong phần mềm tương ứng khác nhau.

Luận văn đề xuất phương án xây dựng một bộ dữ liệu bao gồm các ca kiểm thử dưới dạng ngôn ngữ tự nhiên và các ca kiểm thử tự động sinh tương ứng. Từ dữ liệu đó, áp dụng ứng dụng NLP và công cụ kiểm thử tự động hướng hành vi để khi gọi đến ca kiểm thử, các kịch bản tự động sinh ra các bước kiểm thử hướng hành vi tương ứng. Tuy nhiên khối lượng dữ liệu là lớn, do vậy trong luận văn chỉ đưa ra phương pháp và sử dụng công cụ tự động trong kiểm thử hướng hành vi để sinh phương thức là các hàm mà chưa tự động hoá được hoàn toàn sinh các thân hàm kiểm thử. Từ việc chạy công cụ kiểm thử tự động trong kiểm thử dựa trên kịch bản BDD ta có thể thấy được quy trình tự động, cũng như phương pháp phát triển, cải tiến công cụ kiểm thử tự động hiện có.

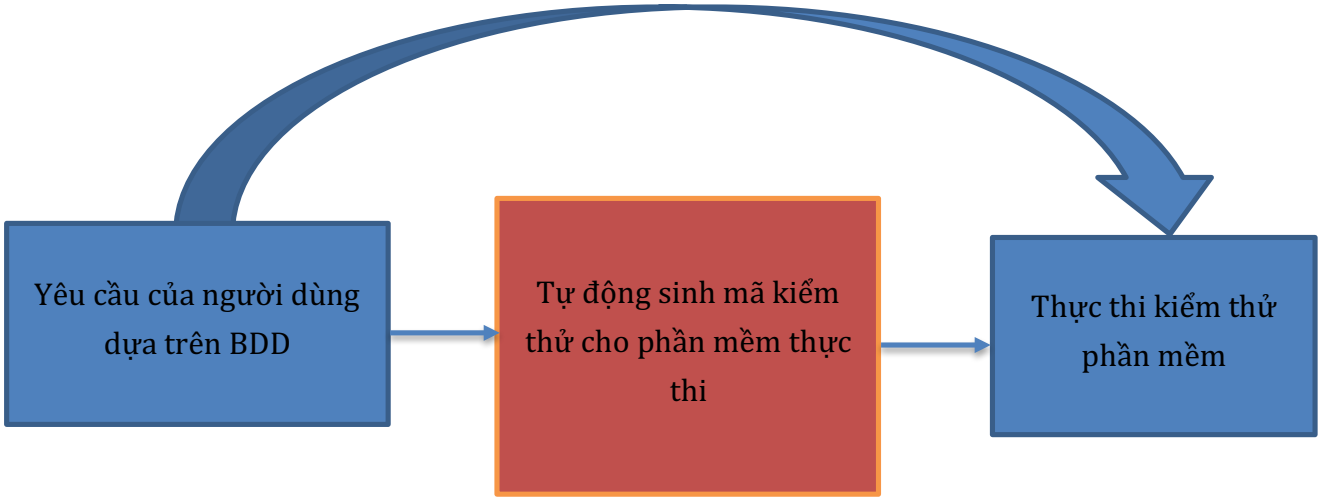
1.3 Bố cục luận văn

Phần còn lại của luận văn gồm các nội dung chính như sau:

Chương 2 Đưa ra các kiến thức nền tảng về quy trình phần mềm, một số khái niệm về tự động hóa trong kiểm thử đồng thời cũng giới thiệu một số công cụ tự động trong kiểm thử tự động BDD, các ứng dụng có thể của xử lý ngôn ngữ tự nhiên trong kiểm thử phát triển phần mềm.

Chương 3 Giới thiệu khái quát, ưu, nhược điểm, cách hoạt động của một số công cụ kiểm thử tự động thường dùng trong kiểm thử hướng hành vi. Đồng thời đưa ra một số ví dụ kiểm thử với các công cụ đưa ra để rút ra kết luận và so sánh giữa các công cụ kiểm thử tự động trong kiểm thử hướng hành vi.

Chương 4 Cài đặt và chạy thực nghiệm Framework kiểm thử tự động với BDD và đưa ra quy trình, viết cơ sở dữ liệu chạy cho công cụ. Dựa trên tìm hiểu các kiến thức và công cụ tự động trong kiểm thử, luận văn phân tích và đưa ra phương pháp sinh mã kiểm thử tự động dựa trên quy trình xử lý ngôn ngữ tự nhiên.



Hình 1-1 Tiến trình sinh mã kiểm thử tự động dựa trên kịch bản BDD

Chương cuối cùng sẽ là các kết luận và hướng phát triển.

CHƯƠNG 2: MỘT SỐ KIẾN THỨC NỀN TẢNG

2.1. Phát triển phần mềm dựa trên phương pháp Agile

Trong nhiều thế kỉ qua, có nhiều quy trình phần mềm được ứng dụng trong các doanh nghiệp, tổ chức làm phần mềm như quy trình thác nước, quy trình xoắn ốc một cách thành công. Tuy nhiên với nhiều yêu cầu phức tạp trong nhu cầu làm phần mềm ngày nay quy trình luôn được đổi mới và lặp đi lặp lại. Agile [11] là phương pháp phát triển phần mềm theo hướng linh hoạt, chia quy trình là các pha nhỏ để phát triển phần mềm. Quy trình Agile có các đặc điểm sau:

- Các công việc trong tiến trình làm phần mềm được chia và kết hợp lại từ các giai đoạn làm phần mềm khác nhau.
- Phần mềm được triển khai dựa trên tài liệu hướng dẫn.
- Khách hàng được đàm phán thông qua hợp đồng.
- Phản hồi thay đổi theo kế hoạch một cách nhanh chóng, linh hoạt.

Agile là sự kết hợp của sự đa dạng các quy trình làm phần mềm truyền thống khác nhau như: Scrum, RUP…Với tiến trình Agile phần mềm được phát triển một cách linh hoạt, tạo ra một sự kết hợp giữa các phương pháp phát triển một cách chặt chẽ. Quy trình Agile đã kết hợp nhiều phương thức phát triển tiên tiến như: Test Driven Development (TDD- phát triển phần mềm kiểm thử), Behavior Driven Development (BDD- Phát triển theo hướng hành vi),…

❑ Kiểm thử phần mềm trong quy trình Agile

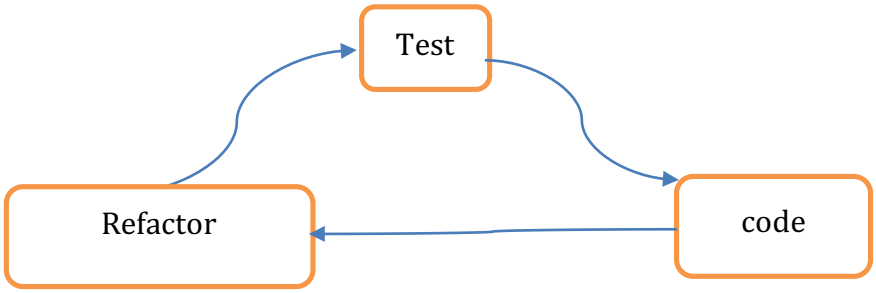
Quy trình Agile là mô hình phát triển linh hoạt, phát triển dựa trên quy trình lặp. Đặc điểm của quy trình này là: dự án được chia thành các mảng, module nhỏ để dễ sử dụng và nhanh chóng thay đổi khi yêu cầu khách hàng thay đổi, các phần nhỏ sẽ được kiểm thử ngay trong khi làm dự án mà không cần đợi đến khi kết thúc dự án. Trong quy trình phát triển phần mềm theo tiến trình Agile sản phẩm được xây dựng tốt ngay từ đầu, sau đó nhận các phản hồi của khách hàng và cải tiến lại sao cho đúng yêu cầu.

Kiểm thử phần mềm trong quy trình phát triển phần mềm Agile không phải là một giai đoạn của dự án nữa mà là một phần của dự án ngay từ đầu. Để đạt được chất lượng phần mềm tốt với bất kì quy trình phát triển phần mềm nào, kiểm

thử luôn là một giai đoạn tốn kém và mất nhiều chi phí, nhân công. Đặc biệt là trong quy trình Agile với sự linh hoạt và chia nhỏ các module phát triển, nhân công cho kiểm thử càng gia tăng, chất lượng phần mềm càng khó kiểm soát và tốn nhiều kinh phí. Chính vì vậy, kiểm thử được đánh giá là khá mất thời gian và công sức của cả nhóm phát triển phần mềm Tự động hóa kiểm thử là một việc làm giúp cải thiện chất lượng phần mềm mà không mất nhiều thời gian, qua mỗi lần thay đổi yêu cầu của người dùng dự án lại được xây dựng lại do đó kiểm thử cần chính xác và đảm bảo nhanh chóng kịp. Trong quy trình Agile, chu trình thực hiện các ca kiểm thử bị dồn lại khá nhiều thời gian dành cho kiểm thử hồi quy là khá ít, ứng dụng kiểm thử tự động trong quy trình Agile là một việc làm thiết yếu.

2.2. Phát triển phần mềm hướng kiểm thử (TDD)

Test - Driven development - TDD là phát triển phần mềm theo hướng kiểm thử. Người lập trình sẽ viết một đoạn một đoạn mã để kiểm thử mã thực thi của chương trình, sau đó viết mã chương trình. Từ kết quả của mã kiểm thử để cải thiện, thay đổi lại code của chương trình. Trong quy trình phát triển phần mềm theo hướng kiểm thử từng phần, có nhiều quy trình sẽ được lặp đi lặp lại, các yêu cầu được xác định rõ hơn làm cho phần mềm đáp ứng được yêu cầu khách hàng một cách đầy đủ, rõ ràng hơn.



Hình 2-1. Quy trình TDD

Trong quy trình TDD, phần mềm được phát triển theo hướng kiểm phát triển kiểm thử trước, sau đó từ kết quả kiểm thử người phát triển điều chỉnh lại mã nguồn sao cho đúng với yêu cầu nghiệp vụ của dự án.

Các ưu điểm của phát triển phần mềm hướng kiểm thử [12] là:

- Khả năng kiểm thử: khả năng kiểm thử của hệ thống được cải thiện thông qua hoạt động viết ca kiểm thử trước, khiến cho các lập trình viên khi lập trình sẽ phải nghĩ về việc làm sao để chương trình đáp ứng được đúng ca kiểm thử và yêu cầu đã thiết kế. Do vậy làm giảm tối thiểu việc kiểm thử cho hệ thống.
- Tính đơn giản: Người lập trình viên phát triển phần mềm theo hướng TDD trong quá trình triển khai các chức năng của hệ thống sẽ tập trung để làm sao cải tiến hoặc chỉnh sửa mã nguồn của mình đã chạy qua được ca kiểm thử, do đó mã nguồn sẽ dễ hiểu hơn và thống nhất hơn.
- Chất lượng: Kiểm thử hồi quy được xây dựng ngay từ đầu khiến cho những thay đổi được xác minh liên tục. Việc này khiến cho những thay đổi càng về sau của hệ thống không gây nên những tác động tiêu cực dẫn đến chất lượng phần mềm được đảm bảo hơn, đưa đến cho nhà phát triển và khách hàng có niềm tin hơn rằng những thay đổi của họ đã được đáp ứng.

Việc chấp nhận phát triển phần mềm theo hướng phát triển hướng TDD đòi hỏi một số kỹ năng từ người phát triển, viết tốt mã kiểm thử là một kỹ năng cần thiết đối với người lập trình theo hướng này. Để thực sự hữu ích trong dự án, ca kiểm thử cần: Nhanh, độc lập, lặp lại và kịp thời.

Các cấp độ TDD:

- Mức lập trình (Developer TDD) là mức tương đương với unit test thường ở mức xử lý chi tiết và trong thiết kế chương trình.
- Mức chấp nhận (Acceptance TDD – ATDD) hay còn gọi là kiểm thử hướng hành vi (BDD). Ở mức ATDD ta viết ca kiểm thử chấp nhận và viết các xử lý để đáp ứng yêu cầu của khách hàng.

Chính vì vậy có thể nói BDD chính là sự mở rộng của TDD (Test – Driven Development). Tuy nhiên, BDD tập trung vào yêu cầu của khách hàng, trong khi ATDD nghiêng về phía nhà phát triển hơn, BDD tập trung vào khía cạnh hành vi của hệ thống, TDD tập trung vào khía cạnh thực thi của hệ thống.

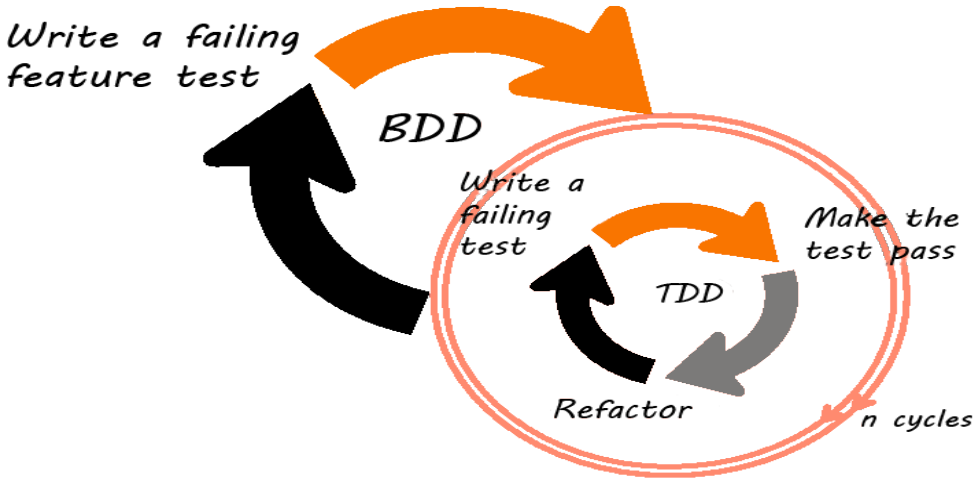
2.3 Phát triển hướng BDD (Behavior-Driven Development)

Trong quy trình phát triển phần mềm theo hướng TDD, người phát triển phần mềm cũng làm luôn chức năng của người kiểm thử phần mềm, do vậy vai trò của kiểm thử viên dường như không cần thiết nữa và người phát triển đồng thời cũng là người lập trình, tuy nhiên việc cộng gộp các vai trò dẫn đến các phát sinh cho người phát triển phần mềm. Các câu hỏi khó xử lý thường thấy của các kiểm thử viên như: Bắt đầu kiểm thử từ đâu? Cái gì cần phải kiểm thử và cái gì không? Thời gian dành cho việc kiểm thử là bao lâu? Cần viết bao nhiêu ca kiểm thử? Tại sao [8] ca kiểm thử lại thất bại?

Do vậy để tận dụng những lợi ích mà quy trình phát triển Test Driven Development mang lại, Dan North đã phát triển mô hình quy trình mới tên là Behavior Driven Development dựa trên nguyên tắc TDD. Khi phát triển phần mềm, không chỉ người lập trình, người kiểm thử mà các bên liên quan khác như: khách hàng, nhà đầu tư...muốn tìm hiểu các vấn đề và thảo luận để đưa ra các ví dụ chức năng hoặc thuộc tính họ mong muốn. Trong phát triển phần mềm, đôi khi là rất khó để tất cả các bên liên quan đều hiểu phần mềm như người lập trình hoặc người thiết kế, kiểm thử viên phần mềm. Chức năng chính của BDD là thể hiện được hành vi của phần mềm bằng ngôn ngữ tự nhiên. BDD sử dụng ngôn ngữ tự nhiên để làm phương tiện, cải thiện khả năng giao tiếp giữa các bên liên quan trong dự án.

BDD là phương pháp nhanh, đa phương thức, đa nền tảng, đa chiều, dựa trên nhiều bên liên quan, tính tự động hoá cao. Nó mô tả một chu trình tương tác với các đầu ra và đầu vào rõ ràng, [13] phân phối công việc, kiểm thử được mạch lạc.

Dưới đây là mô hình BDD – TDD trong quy trình Agile được mô tả bởi Paul Littlebury: [15]



Hình 2-2. Mô hình BDD – TDD được mô tả trong Agile bởi Paul LittleBury

Kiểm thử BDD có vòng đời như sau: [4]

- Định nghĩa các đặc trưng (Features) của nghiệp vụ phần mềm.
- Định nghĩa kịch bản dựa trên các đặc trưng đã lựa chọn.
- Định nghĩa các bước cho mỗi kịch bản.
- Chạy các chức năng và các lỗi.
- Viết mã nguồn để cho các ca kiểm thử là đúng.
- Chỉnh sửa lại mã, tạo thư viện mã để có thể dùng lại.
- Kiểm thử các chức năng cho đúng.
- Sinh báo cáo kết quả kiểm thử.

User stories:

BDD mô tả bằng ngôn ngữ tự nhiên các yêu cầu mức cao của hệ thống là các features. Các features có thể quá lớn để hoàn thành trong một lần lặp. Do vậy, các features được chia nhỏ hơn được gọi là user stories (hay còn gọi là các yêu cầu của người dùng).

Cấu trúc của 1 User story như sau:

As a – [user role] (Vai trò của người dùng)

I want – [functionality] (Chức năng thực hiện)

In order to – [provide business value] (Giá trị mong muốn)

User story là sản phẩm của cuộc đối thoại liên quan đến nhiều user. Người phân tích nghiệp vụ sẽ thảo luận với các bên liên quan về tính năng và yêu cầu của hệ

thống giúp cho có một khung hợp lí của story. Sau đó người kiểm thử giúp xác định phạm vi của story theo dạng tiêu chí của kiểm thử chấp nhận bằng cách xác định kịch bản nào quan trọng và kịch bản nào ít hữu ích hơn. Các user story có thể lặp đi lặp lại, các bên liên quan sẽ có khái niệm về những điều họ mong muốn nhưng thường không biết bao nhiêu thời gian họ sẽ tham gia hoặc làm sao để công việc đó được phân bổ. Với sự giúp đỡ của các nhà phát triển và kiểm thử, tất cả các bên liên quan trong dự án sẽ hiểu được chi phí, lợi ích của từng kịch bản và đưa ra quyết định họ có mong muốn hay không [13].

Một số đặc điểm của một user story tốt:

- Tiêu đề phải mô tả được hoạt động.
- User Story phải bao gồm vai trò, tính năng, lợi ích.
- Tiêu đề kịch bản phải nêu lên sự khác biệt.
- Một user story phải vừa nhỏ để đủ với một lần lặp.

Các features, scenarios, steps tạo nên các ca kiểm thử hành vi. Ngôn ngữ được biết đến phổ biến nhất để viết kịch bản kiểm thử hướng hành vi là ngôn ngữ Gherkin. Gherkin là ngôn ngữ mô tả các nguyên tắc hoạt động của phần mềm dựa trên kịch bản hành vi của người dùng bằng kí pháp ngôn ngữ tự nhiên có cấu trúc của nó.

Với nguyên tắc phát triển phần mềm theo hướng TDD, người phát triển phần mềm viết các mã test trước khi triển khai viết mã ứng dụng. Các mã kiểm thử có thể viết bằng java, C#, C++, python…

Mở rộng hơn ở khái niệm BDD, các ca kiểm thử cũng được viết trước nhưng nó ở dạng dễ hiểu, rõ ràng và minh bạch hơn với người đọc. Ngôn ngữ Gherkin có cú pháp đơn giản, rõ ràng, trực quan gần với ngôn ngữ tự nhiên, chính vì vậy những ca kiểm thử BDD chủ yếu được viết bằng ngôn ngữ Gherkin. Qua khảo sát một số công cụ tự động theo hướng kiểm thử BDD, Cucumber được xem là công cụ phổ biến và khá hiệu quả, giúp cải tiến quy trình phát triển phần mềm. Trong cucumber, kịch bản được viết dưới dạng ngôn ngữ Gherkin, đặc tả ngôn ngữ tự nhiên của ca kiểm thử được mô tả một cách chặt chẽ cho phép bên phát triển, khách hàng và các bên liên quan khác đều hiểu được. Các ca kiểm thử chính là kiểm thử chấp nhận và cấu cấu trúc của nó là tập hợp các features. Mỗi kịch

bản cấu thành 1 ca kiểm thử là dựa trên cấu trúc Give - When - Then, trong mỗi mệnh đề là các bước.

Như đã biết, kịch bản kiểm thử hướng hành vi được viết dưới dạng plain text language bằng ngôn ngữ Gherkin. Từ kịch bản kiểm thử mã kiểm thử được tự động sinh ra. Có khá nhiều công cụ kiểm thử tự động theo hướng kiểm thử hướng hành vi. Trong khuôn khổ của luận văn sẽ cài đặt và nghiên cứu phương pháp sinh mã tự động thông qua một số công cụ kiểm thử tự động theo hướng kiểm thử hướng hành vi sẽ được trình bày ở chương 3.

2.4 Xử lý ngôn ngữ tự nhiên

NLP [1] là một lĩnh vực nghiên cứu trong khoa học máy tính, tập trung vào phát triển hệ thống có cho phép máy tính tương tác với con người dựa trên ngôn ngữ tự nhiên. NLP là một cách để các máy tính phân tích, hiểu ý nghĩa của ngôn ngữ tự nhiên của con người một cách thông minh và hữu ích. Bằng cách sử dụng NLP các nhà phát triển phần mềm có thể tổ chức và cấu trúc dữ liệu kiến thức để thực hiện các công việc, tác vụ như: tổng hợp tự động, dịch thuật, nhận dạng thực thể, phân tích trạng thái, nhận dạng giọng nói, phân loại chủ đề, … NLP thuộc một nhánh nghiên cứu của trí tuệ nhân tạo. Trong các lĩnh vực nghiên cứu của trí tuệ nhân tạo xử lý ngôn ngữ tự nhiên là khá phức tạp vì nó tập trung vào ngôn ngữ của con người (Ngôn ngữ giao tiếp giữa tư duy và giao tiếp). Kỹ thuật sinh mã kiểm thử BDD từ kịch bản kiểm thử viết bằng ngôn ngữ tự nhiên đưa ra các ca kiểm thử tương ứng hoặc gần nghĩa vì vậy cần sử dụng các kĩ thuật xử lý ngôn ngữ tự nhiên.

Hiện nay có các kĩ thuật xử lý ngôn ngữ tự nhiên được sử dụng như:

- POS tagging.
- Phân tích phụ thuộc.

WordNet [2] là một cơ sở dữ liệu lớn từ vựng tiếng anh, được phát triển bởi đại học PrinceTon được sử dụng để thiết kế dưới sự kiểm soát. WordNet nhóm các danh từ, động từ, tính từ thành các bộ đồng nghĩa. Mỗi từ trong cơ sở dữ liệu có thể có nhiều ý nghĩa khác nhau của từ đó. WordNet bao gồm tổng cộng 90000 từ khác nhau và nhiều hơn 166000 cặp kết nối với những ý nghĩa tương đương. WordNet định nghĩa các mối quan hệ khác nhau giữa các từ ngữ nghĩa và mô tả phân cấp, nó cũng cung cấp một biến đếm, giúp xác định từ trong việc dùng chung các từ trong ngữ nghĩa nhất định.

Đầu vào của việc sinh mã kiểm thử cho các kịch bản kiểm thử hướng hành vi là ngôn ngữ tự nhiên. Do vậy việc xử lý ngôn ngữ tự nhiên đóng với trò quan trọng trong việc sinh mã kiểm thử tự động trong kịch bản BDD.

2.5 Khái quát về tự động kiểm thử trong BDD

Tự động sinh mã chạy chương trình được cho là một biện pháp có tác động sâu sắc tới dự án, làm giảm chi phí phát triển phần mềm, chu trình phát triển được rút ngắn, chất lượng phần mềm cũng được rút ngắn.

Theo định nghĩa công cụ tự động hỗ trợ BDD là một framework hỗ trợ tự động giống như trong TDD (Test Driven Development). Tuy nhiên ngôn ngữ kịch bản trong BDD sẵn sàng để các bên liên quan phần mềm đều hiểu được chứ không phải chỉ các nhà phát triển. Do vậy, công cụ tự động hỗ trợ kiểm thử trong [9] BDD là:

- Công cụ đọc được tài liệu đặc tả.
- Công cụ trực tiếp hiểu các phần chính thức của ngôn ngữ kịch bản kiểm thử BDD (chẳng hạn như từ khoá When được mô tả trong ngôn ngữ Gherkin). Dựa vào đó công cụ sẽ phá vỡ từng kịch bản kiểm thử thành các mệnh đề có ý nghĩa.
- Mỗi mệnh đề riêng lẻ trong một kịch bản được chuyển thành một số tham số cho một ca kiểm thử cho yêu cầu của người dùng, tùy thuộc vào từng dự án phần mềm.
- Sau đó thực hiện phép thử cho từng kịch bản, với các tham số từ kịch bản trên.

Một số IDE ngày nay như Eclipse, Netbean, Microsoft Visual Studio... cho phép tạo mã nguồn tự động dựa trên sự tương tác của lập trình viên. Trong các IDE này, một số phần mềm hay ứng dụng có thể được tạo ra bằng cách kéo thả hoặc sinh tự động. Bước đầu làm đơn giản cho lập trình viên hơn trong quá trình lập trình, làm phần mềm.

CHƯƠNG 3: MỘT SỐ CÔNG CỤ KIỂM THỬ TỰ ĐỘNG HƯỚNG HÀNH VI

Ở chương trên là nội dung giới thiệu nền tảng và các định nghĩa, phương pháp của một số phần mềm ứng dụng cho kiểm thử theo hướng BDD. Trong chương này, luận văn giới thiệu một số công cụ trong kiểm thử tự động trong kiểm thử hướng hành vi như [14]: Cucumber, Jasmine, Rspec. Các công cụ này đã được ứng dụng trong nhiều dự án phần mềm, dễ dùng và mang lại hiệu quả cao trong kiểm thử hướng hành vi.

3.1 Công cụ kiểm thử Cucumber

3.1.1 Một số ưu điểm khi sử dụng Cucumber

- Các tài liệu đặc tả, thông số kĩ thuật và tài liệu kiểm thử thành một sự gắn kết hoàn chỉnh.
- Các ca kiểm thử được tự động bởi Cucumber, do đó thông số đặc tả luôn được cập nhật.
- Giúp các bên liên quan có thể theo dõi việc kiểm thử mà không cần có chuyên môn về công nghệ thông tin.

3.1.2 Các thành phần của Cucumber

❖ Feature

Feature được hiểu như một chức năng, hay một đơn vị ứng dụng của dự án. Chẳng hạn khi sử dụng một trang mạng xã hội, người dùng sẽ có thể thực hiện các chức năng của phần mềm như:

- Đăng kí tài khoản
- Đăng nhập tài khoản
- Kết nối bạn bè
- Thay đổi thông tin…

Trong công cụ Cucumber, mỗi feature là một chức năng độc lập của sản phẩm.

Các feature file viết bằng ngôn ngữ gherkin.

Cấu trúc của ngôn ngữ Gherkin như sau:

Bảng 3-1. Ngôn ngữ Gherkin

Feature: [title]	Mô tả feature
As a [role]	Vai trò của đối tượng
I want [action]	Hành động đối tượng muốn thực hiện
So that [business]	Nghiệp vụ mong muốn
Scenario: [title]	Mô tả mục đích của kịch bản kiểm thử
Given [context] And [more context]	Các điều kiện để thực hiện kịch bản
When [action] And [other action]	Các hành động đầu vào
Then [outcome] And [more outcome]	Các kết quả mong muốn

❖ Scenario: Là kịch bản kiểm thử được viết dưới dạng ngôn ngữ Gherkin.

❖ StepDefination: Là mô tả các bước trong kịch bản kiểm thử. StepDefination được sinh tự động một phần qua kịch bản viết bởi ngôn ngữ Gherkin.

3.2 Công cụ kiểm thử Jasmine

3.2.1 Giới thiệu về Jasmine

Thời gian gần đây ngôn ngữ JavaScript khá phát triển và được sử dụng phổ biến. Jasmine là công cụ kiểm thử hướng hành vi cho mã nguồn viết cho ngôn ngữ JavaScripts. Jasmine không phụ thuộc và bất kì nền tảng Javascript nào, không yêu cầu cấu trúc DOM, có cú pháp rõ ràng, dễ hiểu để có thể dễ dàng viết

các ca kiểm thử. Vì vậy, Jasmine phù hợp cho kiểm thử web, các dự án Node.js hoặc bất của cứ nơi đâu mà mã JavaScript có thể chạy.

Đặc điểm của Jasmine:

- Chi phí thấp, không phụ thuộc vào các tác nhân bên ngoài.
- Bắt đầu một quá trình kiểm thử với mọi thứ cần thiết để kiểm tra.
- Chạy trình duyệt và các ca kiểm thử Node.js sử dụng cùng một hệ thống.

Dưới đây là một ca kiểm thử pass của Jasmine:

Hình 3-1. Kiểm thử với Jasmine

3.2.2. Kiến trúc của Jasmine

Kiến trúc của Jasmine được trình bày trên trang chủ: https://jasmine.github.io/ và bao gồm một phiên bản Jasmine kiểm thử độc lập cho mã nguồn Javascript. Trong một phiên bản của Jasmine bao gồm:

- Một ứng dụng ví dụ
- Các ca kiểm thử ví dụ.

File SpecRunner.html chứa tệp nguồn và các thông số kỹ thuật đi kèm trong thẻ <head>

```
<meta charset="utf-8">
<title>Jasmine Spec Runner v2.6.2</title>
<link rel="shortcut icon" type="image/png" href="lib/jasmine-2.6.2/jasmine_favicon.png">
<link rel="stylesheet" href="lib/jasmine-2.6.2/jasmine.css">
<script src="lib/jasmine-2.6.2/jasmine.js"></script>
<script src="lib/jasmine-2.6.2/jasmine-html.js"></script>
<script src="lib/jasmine-2.6.2/boot.js"></script>
<!-- include source files here... -->
```

```
  <script src="src/Player.js"></script>
  <script src="src/Song.js"></script>

  <!-- include spec files here... -->
  <script src="spec/SpecHelper.js"></script>
  <script src="spec/PlayerSpec.js"></script> -->
</head>
```

Cấu trúc của công cụ kiểm thử Jasmine bao gồm 3 thành phần chính:

- Lib: Bao gồm các thư viện.
- Spec: Mô tả kịch bản test.
- Src: Các file mã nguồn bằng javascript.

Trong bản mẫu của Jasmine cũng có kiểm thử 2 chương trình javascript là player.js và song.js.

Để ví dụ cho kịch bản kiểm thử với jasmine, luận văn kiểm thử bài toán kiểm tra mã nguồn chương trình JavaScript như sau:

- Bài toán đặt ra: kiểm thử hàm calculator có các chức năng tính toán cộng, trừ, nhân, chia.
- Mã nguồn:

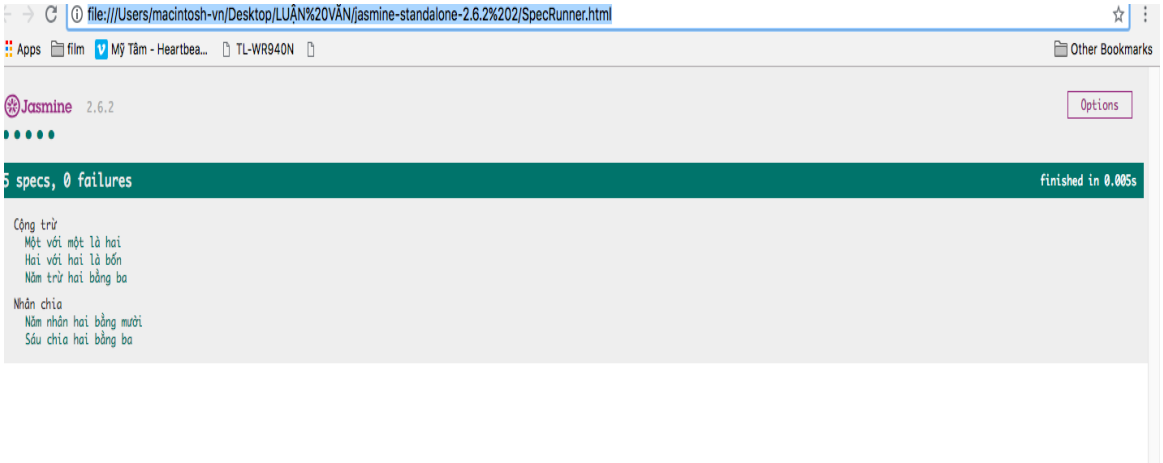
```
function Calculator()
{
  this.add = function(a, b) { return a+b;};
  this.minus = function(a, b) { return a-b;};
  this.multiply = function(a, b) { return a*b;};
  this.divide = function(a,b) {return a/b;} ;
}
```

- Kịch bản test:

```
describe("Cộng trừ", function() {
 var cal = new Calculator();
 it("Một với một là hai", function() {
```

```
    expect(2).toBe(cal.add(1,1));});
  it("Hai với hai là bốn", function() {
    expect(4).toBe(cal.add(2,2));});
  it("Năm trừ hai bằng ba", function() {
    expect(3).toBe(cal.minus(5,2)); });});
  describe("Nhân chia", function() {
  var cal = new Calculator();
  it("Năm nhân hai bằng mười", function() {
    expect(10).toBe(cal.multiply(5,2));});
  it("Sáu chia hai bằng ba", function() {
    expect(3).toBe(cal.divide(6,2)); });});
describe("cong tru nhan chia ", function(){
   var cal2 = new Calculator();
   expect(20).tobe(cal2.a_plus_b(10,10));});
```

- Kết quả khi chạy file specRunner.html ta được kết quả như sau:



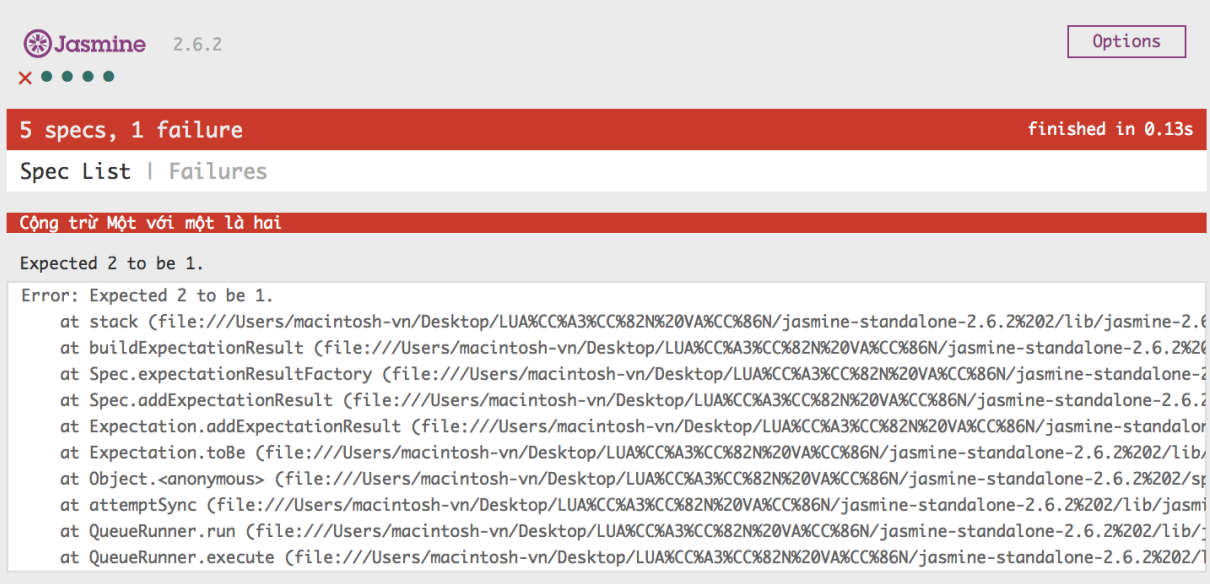
Hình 3-2. Chạy kịch bản kiểm thử trên Jasmine

- Khi chỉnh sửa hàm add() trong chương trình thành mã có lỗi:

```
function Calculator()
{
  this.add = function(a, b) { return a*b;};
  this.minus = function(a, b) { return a-b;};
```

```
  this.multiply = function(a, b) { return a*b;};
  this.divide = function(a,b) {return a/b;} ;
}
```

- Kết quả thì chạy kịch bản kiểm thử sẽ cho kết quả báo lỗi:



Hình 3-3. Chạy ca kiểm thử khi hàm có lỗi

Như vậy ta biết được ca kiểm thử bị sai ở hàm add() để thực thi hàm cộng trong tính toán. Từ đó có thể kiểm tra được mã nguồn Javascript để sửa lại cho đúng.

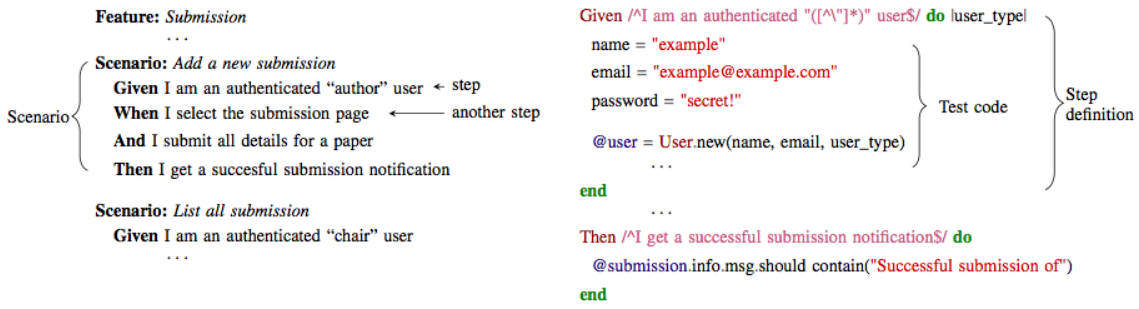
3.3 Công cụ kiểm thử Rspec

Rspec là một Framework phát triển phần mềm hướng hành vi cho lập trình viên Ruby. Lập trình viên viết các ví dụ thực thi hành vi một phần nhỏ điều khiển nội dung. Trái ngược với Cucumber nó không tuyệt đối chia các thành phần ngôn ngữ tự nhiên.

Rspec được phát hành năm 2005 giống như một thử nghiệm của Stenven Baker. Rspec ra đời năm 2007 và giữ nguyên nhiều tính năng cho đến nay. Hiện nay, Rspec được phát triển và cải tiến bởi đông đảo cộng đồng hàng trăm cộng tác viên.

Rspec bao gồm nhiều thư viện, được thiết kế để có thể hoạt động độc lập hoặc sử dụng cùng với các công cụ kiểm thử tự động khác.

Cấu trúc của Rspec step definition như sau: [14]



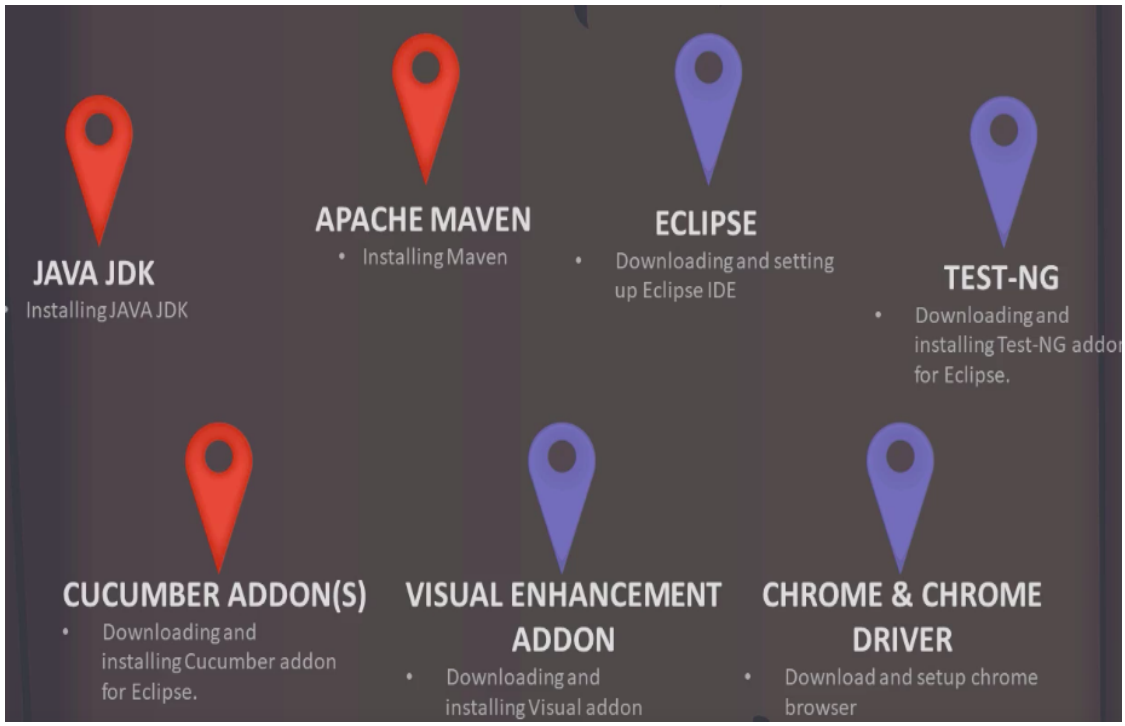
Hình 3-4. Cấu trúc các bước kiểm thử của Rspec

Trong chương này, qua khảo sát các công cụ kiểm thử phần mềm hướng hành vi nhận thấy công cụ kiểm thử Cucumber là công cụ dễ dùng và có quy trình kiểm thử rõ ràng khi kết hợp với các công cụ kiểm thử tự động khác. Với mục đích ứng dụng công cụ vào trong dự án thực tế, luận văn đi sâu tìm hiểu công cụ Cucumber và xây dựng Framework kiểm thử tự động hướng hành vi sử dụng Cucumber bằng cách kết hợp công cụ Selenium tích hợp trên nền tảng Java với Eclipse để kiểm thử ứng dụng web. Nội dung thực nghiệm Framework kiểm thử tự động dựa trên BDD sẽ được trình bày trong chương 4 sau đây.

CHƯƠNG 4: THỰC NGHIỆM FRAMEWORK KIỂM THỬ TỰ ĐỘNG VÀ ĐÁNH GIÁ

4.1 Các thành phần của Framework kiểm thử sử dụng Cucumber

Trong khuôn khổ áp dụng của luận văn framework Cucumber được xây dựng dựa trên phối hợp của 3 nền tảng sau: Selenium Webdriver, công nghệ Java và công cụ kiểm thử BDD là Cucumber.



Hình 4-1. Các thành phần Framework kiểm thử

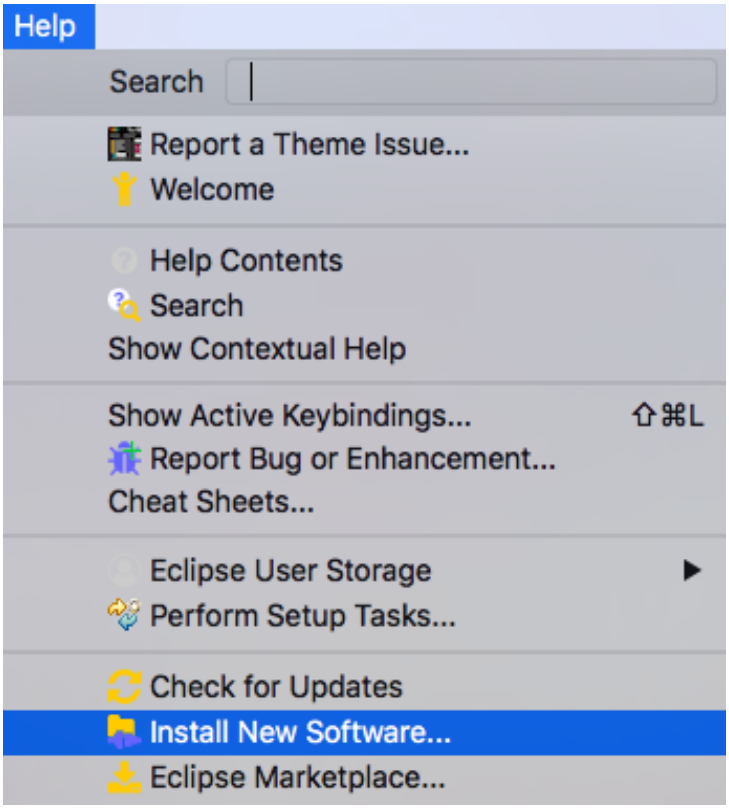
4.1.1 Công nghệ Java

Eclipse là một môi trường tích hợp phát triển cho các ngôn ngữ lập trình. Nền tảng Eclipse là công cụ mã nguồn mở được phát hành theo giấy phép EPL (Eclipse Public License) để có thể sử dụng miễn phí đồng thời cho phép sửa đổi và phân phối bởi cộng đồng.

Sau khi cài đặt nền tảng Java và Eclipse, đồng thời tích hợp Maven vào Eclipse. Maven là công cụ quản lý dự án phần mềm, cung cấp một cái nhìn mới về mô hình đối tượng dự án (POM). Maven cho phép tự động hoá cấu trúc thư mục ban đầu, thực hiện việc biên dịch và kiểm thử cũng như đóng gói sản phẩm cuối cùng. Nó rất tốt trong việc giảm số lượng các bước trong quá trình xây dựng phần mềm. Maven giúp đơn giản và chuẩn hoá tiến trình xây dựng dự án, xử lý biên soạn, phân phối tài liệu, kết hợp các nhiệm vụ một cách liền mạch. Thư viện Maven được lưu trữ mặc định trên: http://mvnrepository.com/, ta có thể lên địa chỉ trên để tải tất cả các thư viện liên quan.

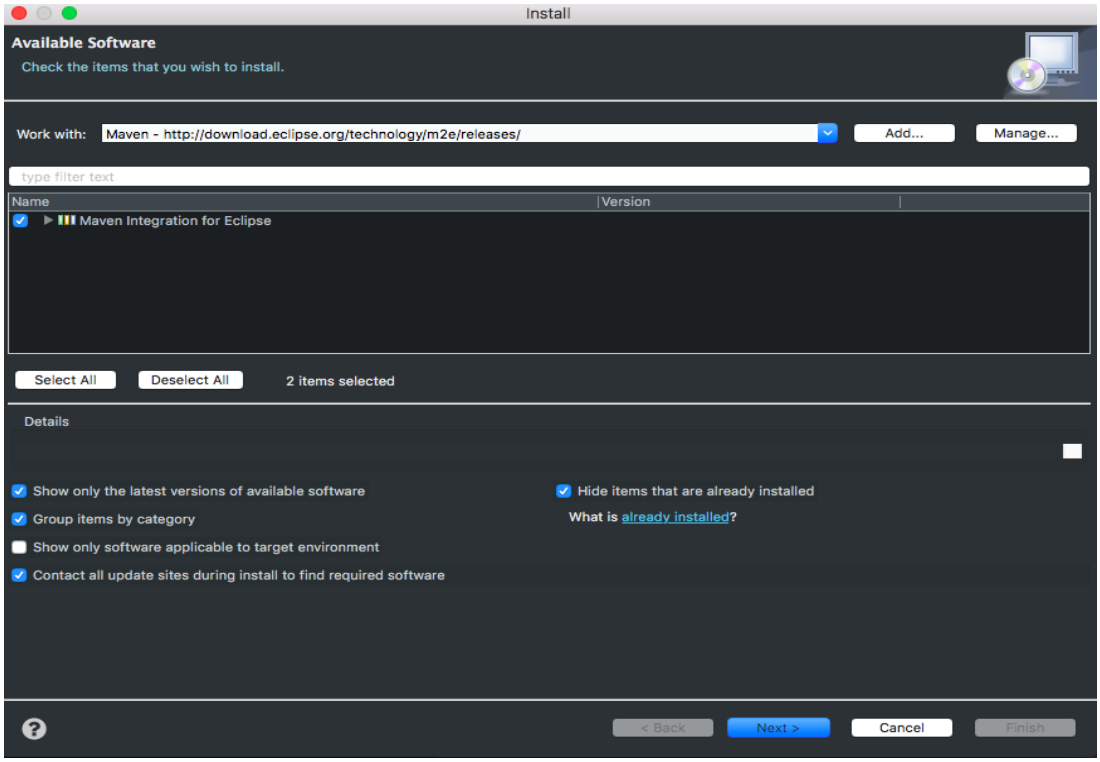
Các bước Cài đặt Maven trong Eclipse IDE:

- Từ menu công cụ Eclipse, chọn help -> Install new software:



Hình 4-2. Tìm kiếm thư viện trong Eclipse

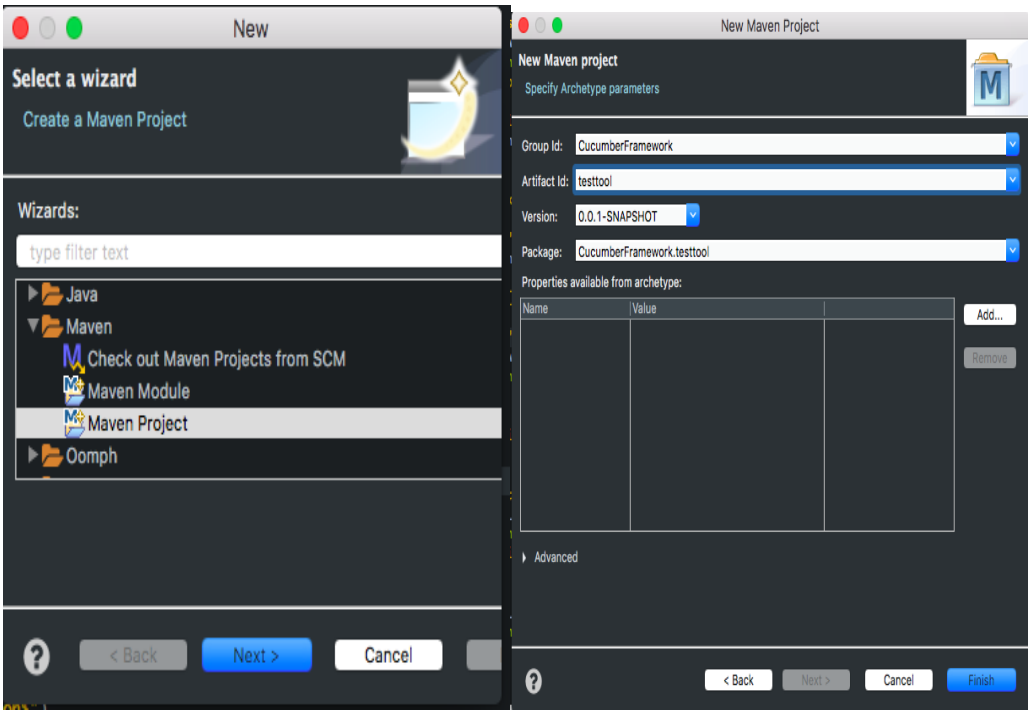
- Gõ tìm kiếm Maven trong box work with rồi chọn button add, eclipse sẽ chỉ đến đường dẫn của maven, chọn và nhấn button next.



Hình 4-3. Cài đặt thư viện Maven trong Eclipse

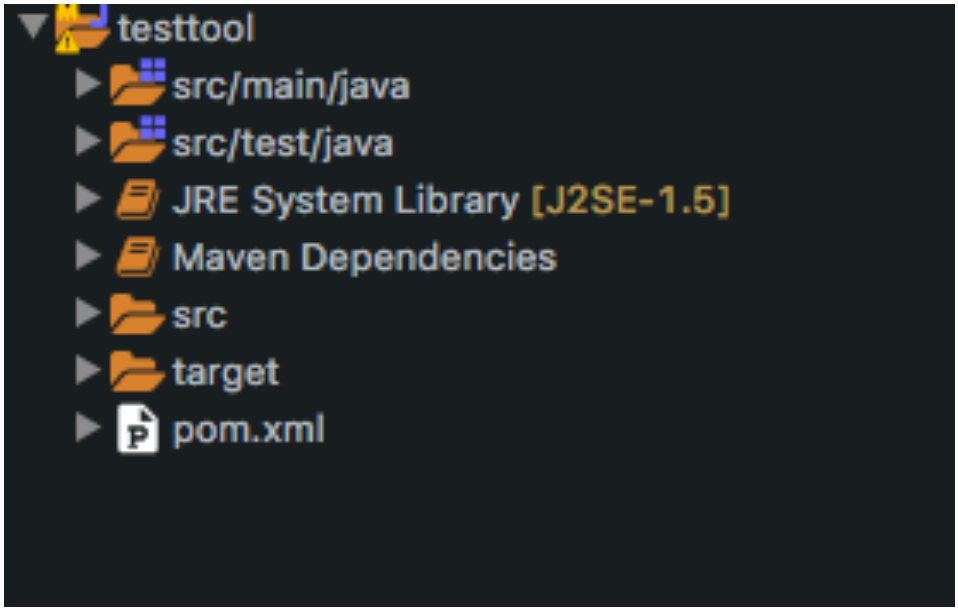
Sau khi đã chạy được maven trên Eclipse IDE.

Để tạo Framework kiểm thử tự động với Cucumber ta tạo một dự án Maven :



Hình 4-4. Tạo dự án maven

Ta có cấu trúc của dự án sẽ như sau:



Hình 4-5. Cấu trúc 1 dự án Maven

Các kịch bản kiểm thử BDD sẽ nằm trong thư mục /src/test/java. Trong cấu trúc của dự án Maven ta có tệp pom.xml, Pom(Project Object Model) là tệp Xml bao gồm thông tin về dự án và cấu hình maven được sử dụng để xây dựng dự án, nó chứa các giá trị mặc định cho hầu hết các dự án.

Các "dependence" là các thư viện, cái mà yêu cầu bởi dự án. Ví dụ như selenium, apache, junit...Các dependence được gọi tới trong tệp pom.xml của Maven để xây dựng Framework Cucumber ta cần các dependence để gọi các thư viện của Selenium, Junit, Cucumber

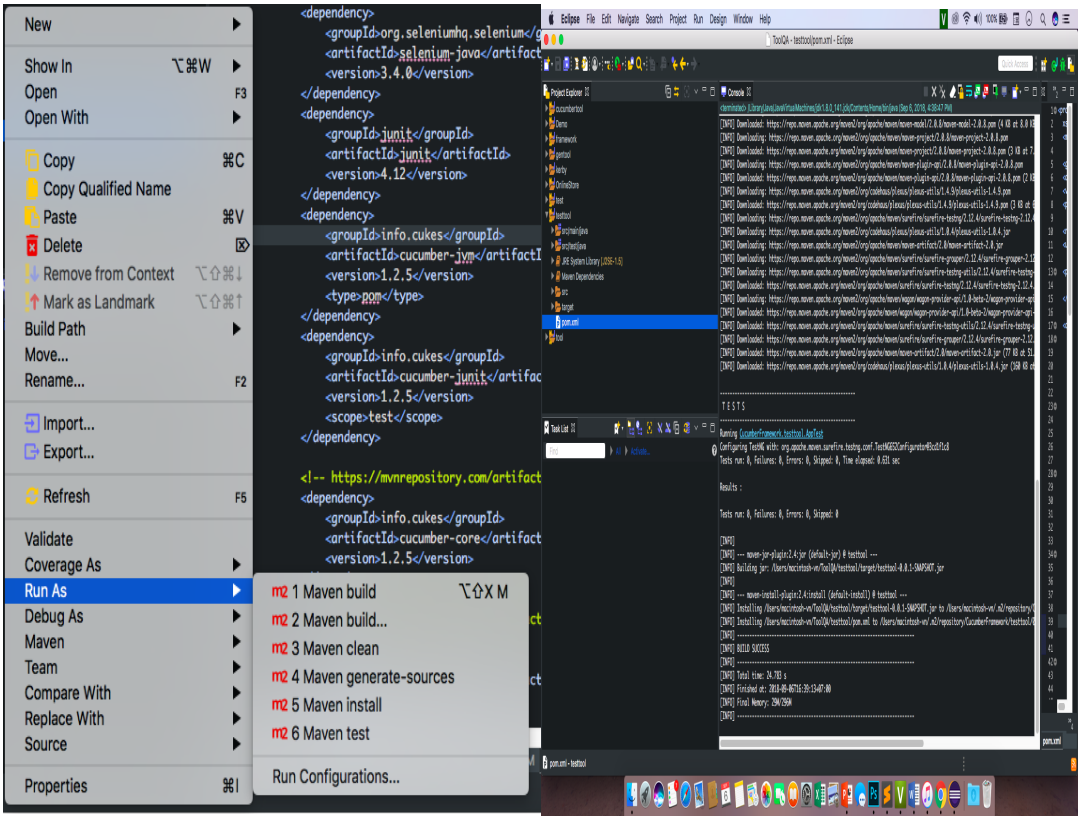
```
<dependencies>
        <dependency>
            <groupId>org.seleniumhq.selenium</groupId>
            <artifactId>selenium-java</artifactId>
            <version>3.4.0</version>
        </dependency>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.12</version>
```

```
</dependency>
<dependency>
     <groupId>info.cukes</groupId>
     <artifactId>cucumber-jvm</artifactId>
     <version>1.2.5</version>
     <type>pom</type>
</dependency>
<dependency>
     <groupId>info.cukes</groupId>
     <artifactId>cucumber-junit</artifactId>
     <version>1.2.5</version>
     <scope>test</scope>
</dependency>
<dependency>
     <groupId>info.cukes</groupId>
     <artifactId>cucumber-core</artifactId>
     <version>1.2.5</version>
</dependency>
<dependency>
     <groupId>info.cukes</groupId>
     <artifactId>cucumber-html</artifactId>
     <version>0.2.3</version>
</dependency>
<dependency>
     <groupId>info.cukes</groupId>
     <artifactId>cucumber-java</artifactId>
     <version>1.2.5</version>
</dependency>
<dependency>
```

```
            <groupId>info.cukes</groupId>
            <artifactId>cucumber-jvm-deps</artifactId>
            <version>1.0.5</version>
        </dependency>
        <dependency>
            <groupId>info.cukes</groupId>
            <artifactId>gherkin</artifactId>
            <version>2.12.2</version>
        </dependency>
        <dependency>
            <groupId>org.hamcrest</groupId>
            <artifactId>hamcrest-all</artifactId>
            <version>1.3</version>
        </dependency>
        <dependency>
            <groupId>info.cukes</groupId>
            <artifactId>cucumber-
picocontainer</artifactId>
            <version>1.2.5</version>
        </dependency>
        <dependency>
            <groupId>info.cukes</groupId>
            <artifactId>cucumber-testng</artifactId>
            <version>1.2.5</version>
        </dependency>
        <!-- Extent Reports -->
        <dependency>
            <groupId>com.aventstack</groupId>
            <artifactId>extentreports</artifactId>
```

```
            <version>3.0.5</version>
        </dependency>
        <dependency>
            <groupId>org.freemarker</groupId>
            <artifactId>freemarker</artifactId>
            <version>2.3.26-incubating</version>
        </dependency>
        <dependency>
            <groupId>net.masterthought</groupId>
            <artifactId>cucumber-reporting</artifactId>
            <version>3.6.0</version>
        </dependency>
        <dependency>
            <groupId>com.vimalselvam</groupId>
            <artifactId>cucumber-
extentsreport</artifactId>
            <version>2.0.4</version>
</dependency>
    </dependencies>
```

Sau khi thêm các dependency vào file pom.xml, chọn maven install để cài đặt và kiểm tra các cài đặt có thành công không:



Hình 4-6. Kiểm tra cài đặt các thư viện trong file pom.xml

4.1.2 Selenium Webdriver

Công cụ kiểm thử Cucumber là công cụ kiểm thử BDD phổ biến trên trình duyệt Web nhưng nó không bao gồm các thư viện để tương tác với trình duyệt Web. Vì vậy cần sử dụng thư viện trong framework Selenium để tương tác hiệu quả với các phần tử của website trên trình duyệt.

Selenim Webdriver là công cụ mã nguồn mở và hoàn toàn miễn phí.

Trước khi chạy các steps trong bước trên, chúng ta cần viết thêm các tiền điều kiện và hậu điều kiện. Giả sử khi cần chạy một web trên Chrome thì chúng ta cần sử dụng công cụ google chrome và kết thúc cũng cần đóng lại trình duyệt, trong Framework Cucumber gọi là các Hooks, Cucumber hỗ trợ 2 hooks là @Before và @After. Với @Before, là các thực thi cần sẵn sàng trước khi chạy

các steps và @After là các thực thi sau khi đã hoàn thành chạy các steps trong kịch bản test.

```
public class login {
    WebDriver driver;
    @Before

        public void setup() {
            System.setProperty("webdriver.chrome.driver", "/Applications/chromedriver");
            this.driver = new ChromeDriver();
            this.driver.manage().window().maximize();
            this.driver.manage().timeouts().pageLoadTimeout(100, TimeUnit.SECONDS);
            this.driver.manage().timeouts().setScriptTimeout(100, TimeUnit.SECONDS);
        }

    @After
        public void teardown() throws InterruptedException  {
        Thread.sleep(3000);
        this.driver.manage().deleteAllCookies();
        this.driver.quit();
        this.driver = null;
    }
```

Hình 4-7. Các Hooks trong Cucumber

Khi gọi các hooks điều cần lưu ý là gọi đến các gói chứa các hooks, đó là: cucumber.api.java.After; cucumber.api.java.Before.

Trong nội dung này, cần chắc chắn các phương thức được viết dưới các hooks @After và @Before đều thực thi được, để trước khi chạy ca kiểm thử đầu tiên kiểm thử web, trình duyệt đã mở và kết thúc ca kiểm thử trình duyệt cũng tự động đóng.

Trong ca kiểm thử tự động phía trên sử dụng selenium webdriver, và cài đặt property cho webdriver là google chrome. Chrome driver là một thành phần riêng biệt được cung cấp để chạy trình duyệt google chrome. Với nền tảng kiểm thử xây dựng có thể chạy kiểm thử web trên nền firefox, tuy nhiên trong luận văn với framework này được thử nghiệm trên google chrome.

Các câu lệnh thường được sử dụng trong selenium webdriver như: Các câu lệnh trình duyệt, câu lệnh tìm phần tử…

Bảng 4-1 Các câu lệnh thường dùng trong Selenium Webdriver

Câu lệnh	Mục đích
Driver.get(URL)	Mở trình duyệt
Driver.quit(), Driver.close()	Đóng trình duyệt
Driver.navigate().refresh()	Làm mới trình duyệt hiện tại
Driver.navigate().to(URL)	Chuyển hướng trình duyệt tới một trang khác
Driver.navigate().forward()	Di chuyển đến trang tiếp theo
Driver.navigate().back()	Quay về trang trước
Driver.manage().timeout()	Thiết lập thời gian chờ đợi kịch bản
Driver.manage().deleteAllCookies()	Xoá tất cả cookie cho tên miền hiện tại
Driver.fineElement()	Tìm kiếm phần tử

Để tìm chính xác phần tử trên web [10] ta có những cách sau:

Bảng 4-2 Các cách định vị phần tử trên web

Locators	Tìm kiếm phần tử trên trang web
ID	Tìm kiếm với ID của phần tử
Classname	Tìm kiếm với class name của phần tử
Name	Tìm kiếm với tên của phần tử
Link text	Tìm kiếm với đề mục của link
Xpath	Tìm kiếm phần tử động và chuyển đổi giữa các phần tử khác nhau của trang web
CSS path	Định vị phần tử không có name, class hoặc ID

Trong luận văn có sử dụng phương thức tìm kiếm phần tử bởi Xpath. Xpath là phương thức để tìm kiếm các thành phần trên trang web. Xpath được định nghĩa là một XML path. Cú pháp thông thường của Xpath như sau:

Xpath=//tagname[@Atribute = ‘value’]

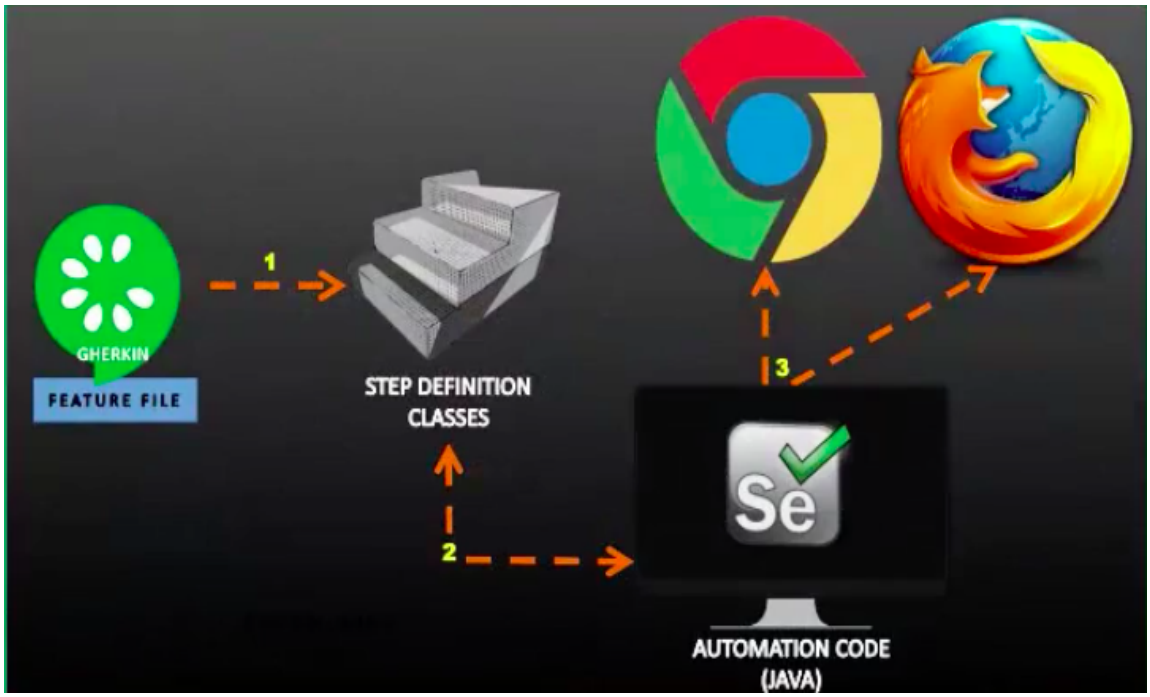
Ví dụ: Xpath =.//*[@name='uid']

+, //: chọn vùng gần node.

+, Attribute: Tên thuộc tính của node.

+, Value: Giá trị của thuộc tính.

4.1.3 Cucumber



Hình 4-8. Quy trình kiểm thử với Framework Cucumber

Quy trình kiểm thử của Framework kiểm thử:

Đầu tiên viết kịch bản kiểm thử dưới dạng ngôn ngữ Gherkin (ngôn ngữ viết kịch bản bằng ngôn ngữ tự nhiên có cấu trúc). Sau đó từ kịch bản đã viết, tự động sinh ra mã nguồn là các phương thức mô tả các bước kiểm thử tương ứng. Cuối cùng sử dụng thư viện của selenium webdrive để chạy các ca kiểm thử đã viết.

Như đã đề cập ở chương 3, Cucumber là công cụ tự động sử dụng cho kiểm thử, chạy các ca kiểm thử chấp nhận dưới dạng BDD. Trong framework kiểm thử này, sử dụng hầu hết các thư viện đã xây dựng của Cucumber. Để mô phỏng cho quy trình này, dưới đây là ca kiểm thử tự động với hành vi đăng nhập của người dùng vào hệ thống.

Bài toán đặt ra là kiểm thử tự động kịch bản kiểm thử đăng nhập thành công vào trang web demo với dữ liệu kiểm thử đúng.

Bước 1: Viết các ca kiểm thử dưới dạng ngôn ngữ Gherkin. Ta có kịch bản nghiệp vụ kiểm như sau:

Người dùng vào trang chủ của trang web demo:

http://www.demo.guru99.com/V4/ sau đó nhập user và password, người dùng đăng nhập thành công và vào trang của user.

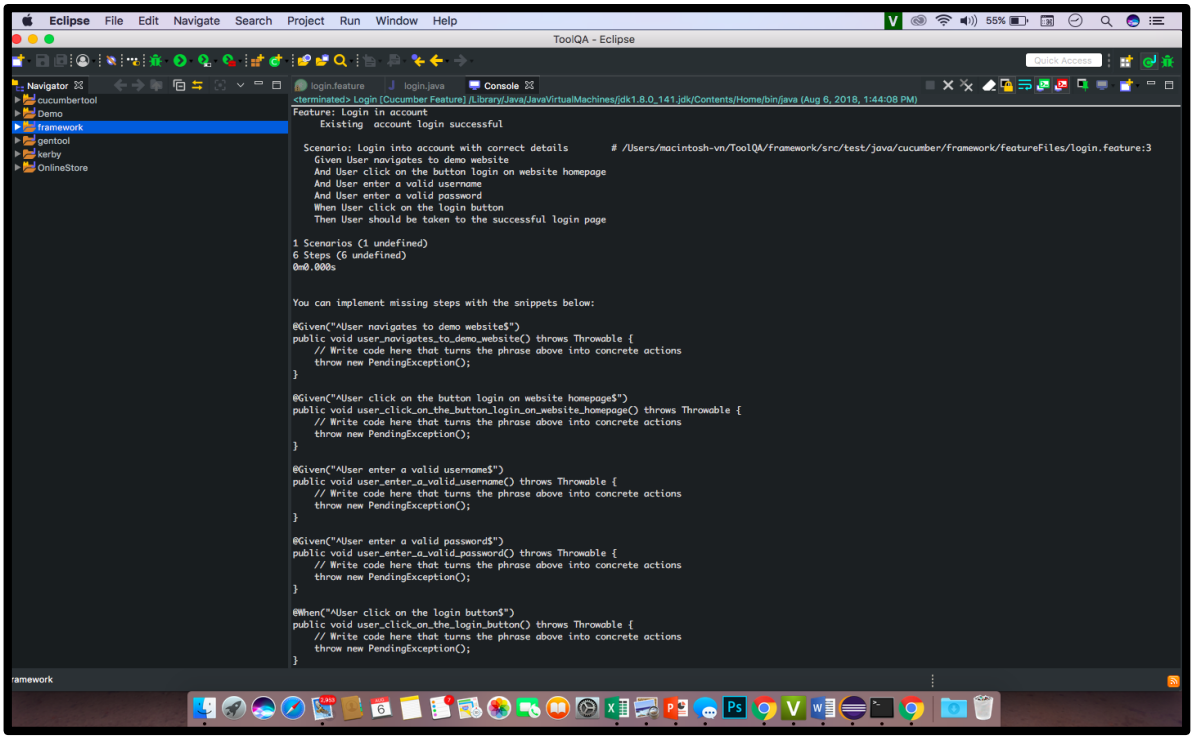
Kịch bản đăng nhập dưới dạng ngôn ngữ Gherkin sẽ như sau:

```
Feature: Login in account
    Existing account login successful
Scenario: Login into account with correct details
    Given User navigates to demo website
    And User click on the button login on website homepage
    And User enter a valid username
    And User enter a valid password
    When User click on the login button
    Then User should be taken to the successful login page
```

Để thực hiện kiểm thử, tạo một package feature trong project, rồi thêm tệp feature có nội dung như trên. Trong file feature trên mô tả 1 ca kiểm thử login vào hệ thống. Khi hệ thống login vào website demo, người dùng nhập một username và password đúng thì người dùng sẽ login thành công vào website.

Bước 2: Từ các Features đã viết ta sinh ra các phương thức trong ngôn ngữ java mô tả các bước (ca) chạy kiểm thử.

Từ kịch bản trên khi chạy ta sinh được các phương thức như sau:



Hình 4-9. Các steps được sinh ra từ Feature file

Từ các phương thức được sinh ra ta có các mô tả phương thức bao gồm các bước của test case. Sau đó bổ sung các hàm test theo trình tự kiểm thử như trong mô tả của tệp feature.

Sau khi hoàn thiện lại các phương thức đã sinh ra ta được các step trong ca kiểm thử login vào hệ thống website, các mô tả step definition được viết bổ sung vào các phương thức là các hàm để kịch bản cucumber có thể thực thi kiểm thử.

```
package testframework.test.StepFiles;
import java.sql.Driver;
import java.util.concurrent.TimeUnit;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.chrome.ChromeDriver;
import cucumber.api.PendingException;
import cucumber.api.java.After;
```

```
import cucumber.api.java.Before;
import cucumber.api.java.en.And;
import cucumber.api.java.en.Given;
import cucumber.api.java.en.Then;
import cucumber.api.java.en.When;
public class login {
     @Given("^User navigates to demo website$")
     public void user_navigates_to_demo_website() throws
Throwable {
    // Write code here that turns the phrase above into
concrete actions
    driver.get("http://www.demo.guru99.com/V4/");
     }
     @And("^User click on the button login on website
homepage$")
     public void
user_click_on_the_button_login_on_website_homepage()
throws Throwable {
     Thread.sleep(200);
driver.findElement(By.xpath(".//*[@name='uid']")).sendKeys
("mngr150187");
     }
     @And("^User enter a valid username$")
     public void user_enter_a_valid_username() throws
Throwable {
     Thread.sleep(200);
     driver.findElement(By.xpath((".//*[@name='password']")
)).sendKeys("agAtymE");
     }
```

```
    @And("^User enter a valid password$")
    public void user_enter_a_valid_password() throws
Throwable {
   // Write code here that turns the phrase above into
concrete actions
    Thread.sleep(200);
    driver.findElement(By.xpath("html/body/form/table/tbod
y/tr[3]/td[2]/input[1]")).click();
    }
    @Then("^User should be taken to the successful login
page$")
    public void
user_should_be_taken_to_the_successful_login_page() throws
Throwable {
        // Write code here that turns the phrase above
into concrete actions
    System.out.println("method 6");
    }}
```

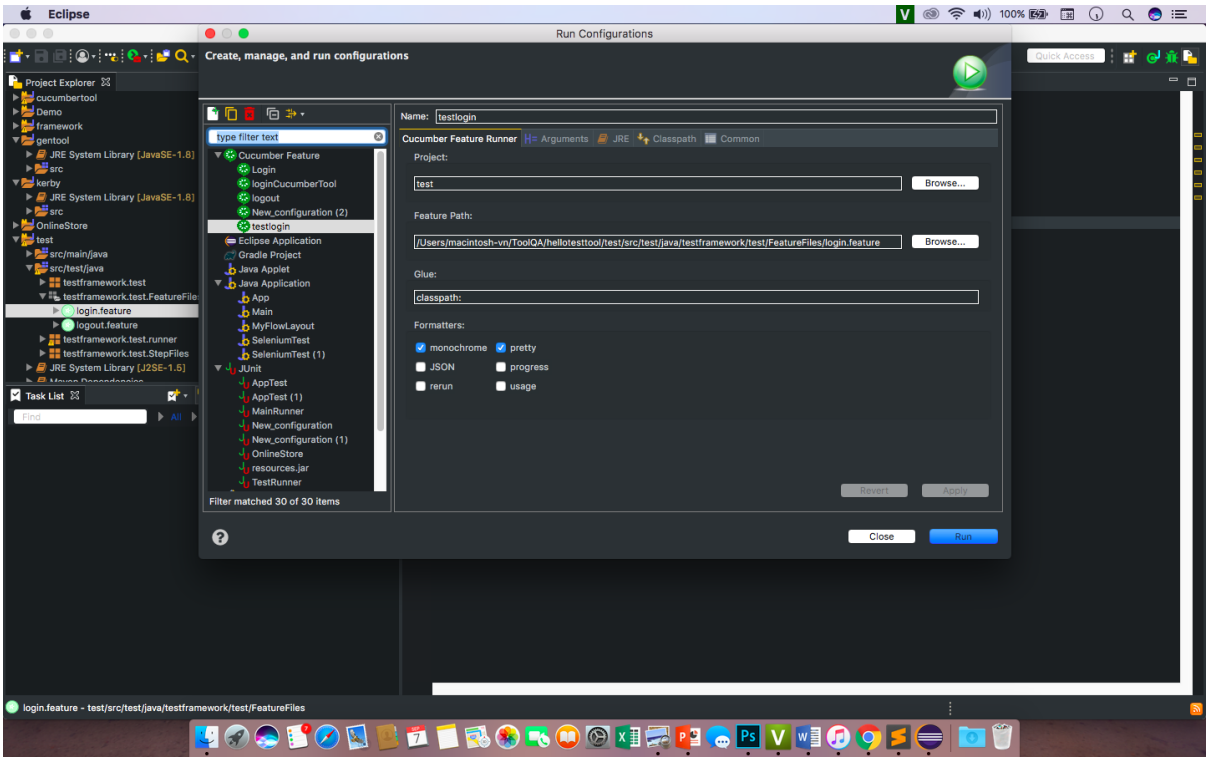
Bước 3: Cuối cùng sử dụng Selenium Webdriver để chạy ca kiểm thử một cách tự động trên web. Webdriver để chạy tự động chạy các ca kiểm thử có thể là chrome hoặc fireFox. Trong kiểm thử tự động với BDD, Cucumber là công cụ để mô tả các các bước kiểm thử với ngôn ngữ tự nhiên tuy nhiên không cung cấp thư viện để tương tác với các trình duyệt web. Chính vì vậy cần kết hợp sử dụng Selenium - công cụ cung cấp các thư viện để tương tác, hoạt động với trình duyệt web.

Để gọi thư viện trong Selenium ta dùng 2 Hook @Before và @After như sau. Với hook @Before định nghĩa các phương thức trước khi chạy ca kiểm thử và @After là sau khi kết thúc ca kiểm thử đã mô tả trong kịch bản ở file Feature.

```
WebDriver driver;
     @Before
          public void setup() {
         System.setProperty("webdriver.chrome.driver",
"/Applications/chromedriver");
               this.driver = new ChromeDriver();
     this.driver.manage().window().maximize();
     this.driver.manage().timeouts().pageLoadTimeout(100,
TimeUnit.SECONDS);
     this.driver.manage().timeouts().setScriptTimeout(100,
TimeUnit.SECONDS);}
     @After
          public void teardown() throws
     InterruptedException  {
          Thread.sleep(3000);
          this.driver.manage().deleteAllCookies();
          this.driver.quit();
          this.driver = null;
     }
```

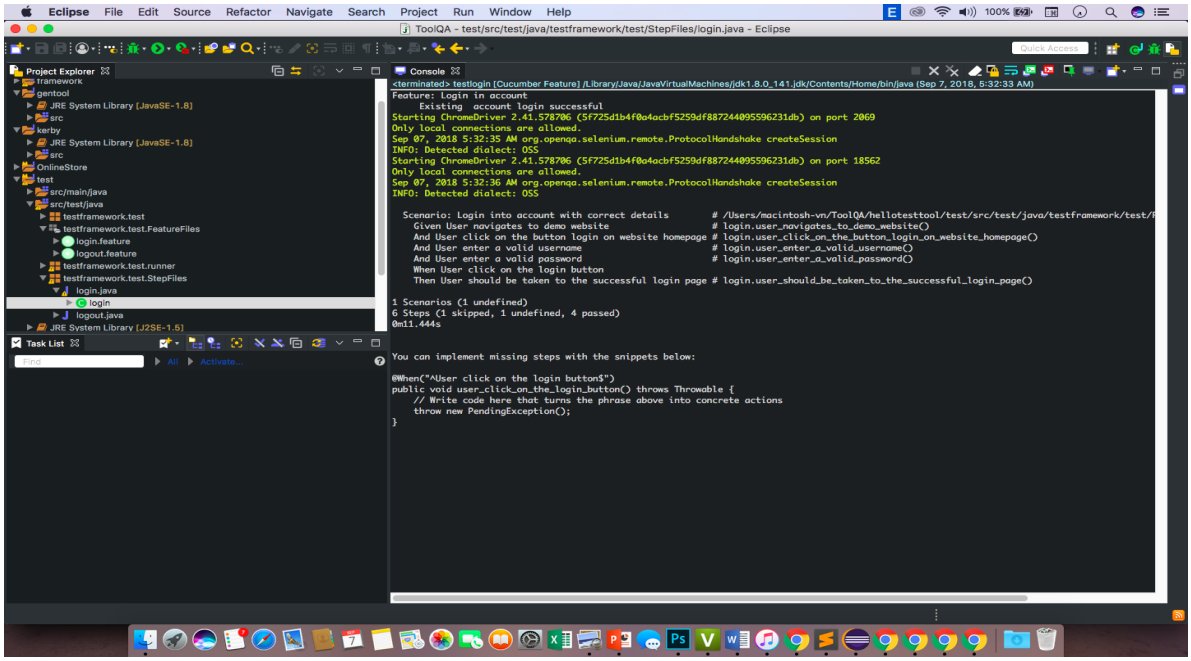
Chạy lại file Feature ta sẽ được kết quả kiểm thử của các bước đã xây dựng trong kịch bản test.

Trong kịch bản kiểm thử đăng nhập vào hệ thống, sau khi hoàn chỉnh các step defination, chạy file login feature như sau:



Hình 4-10. Chạy kịch bản kiểm thử

Framework sẽ tự động chạy các step trong kịch bản đã mô tả:



Hình 4-11. Thực thi ca kịch bản kiểm thử trên nền web

Sau khi chạy tự động ca kiểm thử trên, trình duyệt dừng khi ca kiểm thử kết thúc và thông qua giao diện kết quả ta có thể thấy kết quả kiểm thử, các bước kiểm thử thành công và bước kiểm thử chưa thành công. Cuối cùng, ta cấu hình lại framework để sinh được báo cáo kiểm thử một cách chi tiết hơn.

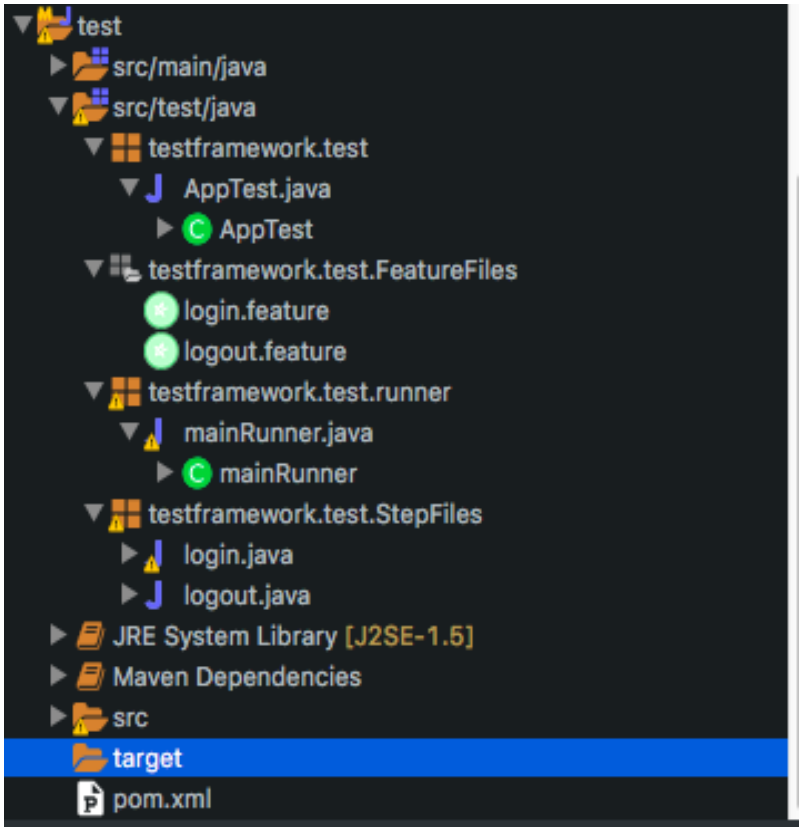
4.2. Báo cáo kết quả kiểm thử

Khi chạy một kịch bản kiểm thử, cho dù là thủ công hay tự động, đầu ra cần có định dạng rõ ràng và cụ thể mô tả tổng thể của việc kiểm thử.

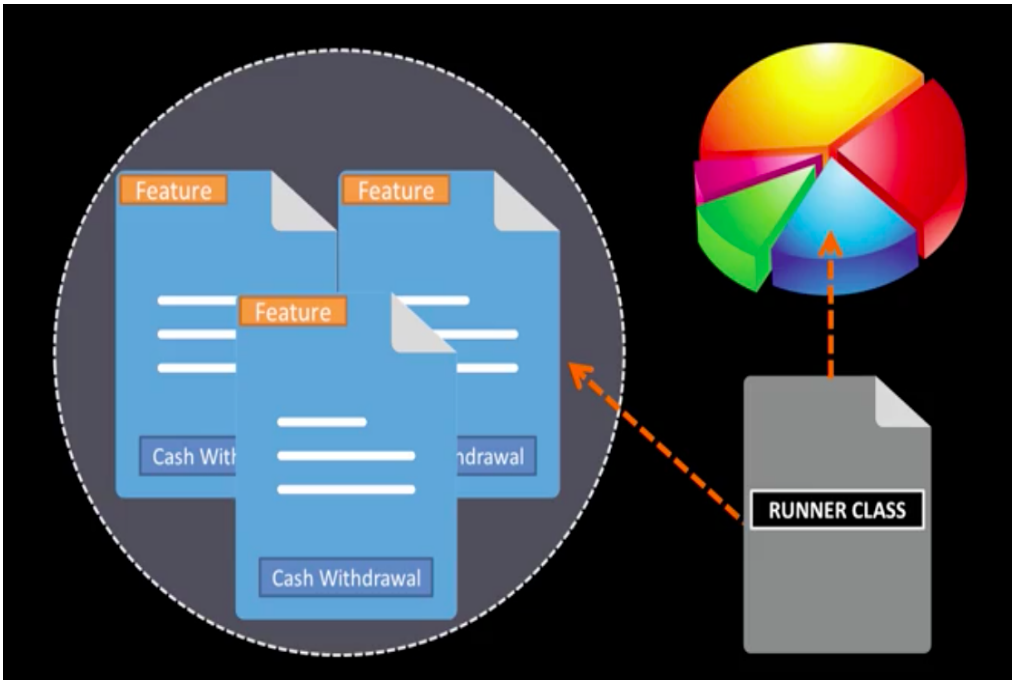
Để framework tự động sinh báo cáo kết quả kiểm thử, ta sử dụng các thư viện có sẵn của cucumber bằng các tạo class runner trong tệp runner như sau:

```
@RunWith(Cucumber.class)
@CucumberOptions (
 features=
{"src/test/java/CucumberFramework/featureFiles/"},
glue = {"CucumberFramework.stepFiles"},
monochrome = true,
plugin = {"pretty", "html:target/cucumber",
"json:target/cucumber.json",
"com.cucumber.listener.ExtentCucumberFormatter:target/
report.html"})
public class MainRunner {}
```

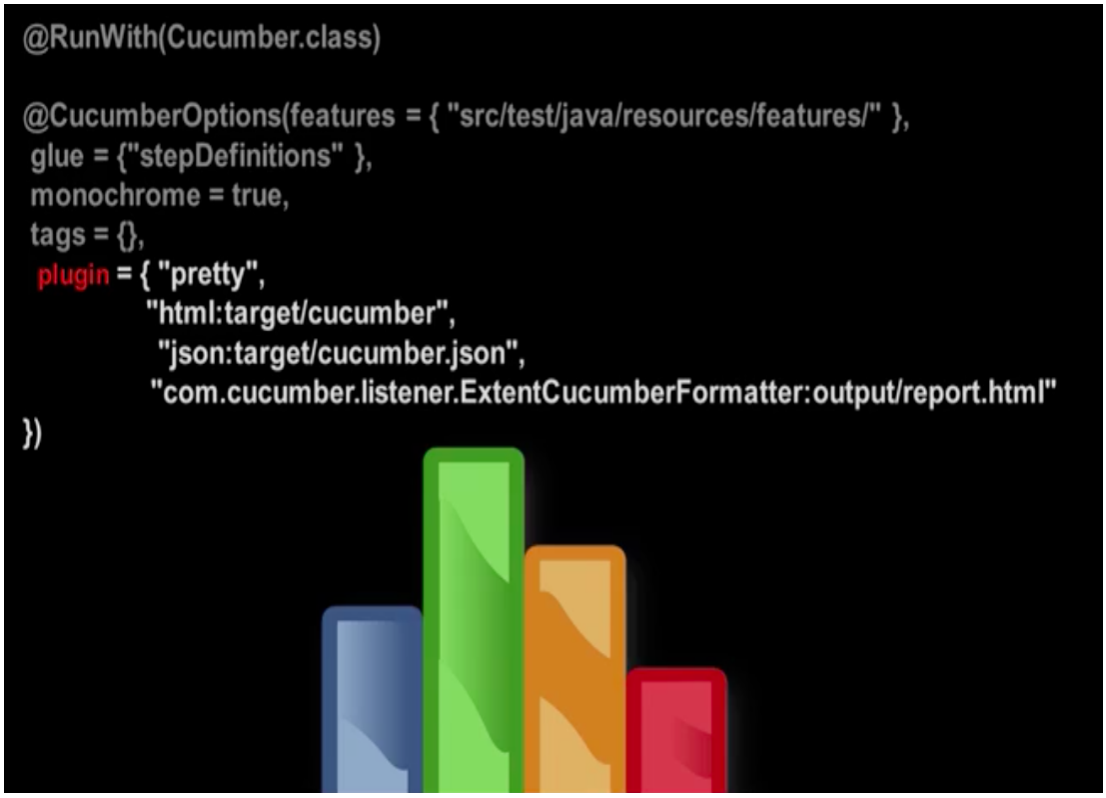
Các file report sẽ được tổng hợp trong thư mục target của cấu trúc mã nguồn Framework kiểm thử.



Hình 4-12. Cấu trúc thư mục sinh báo cáo trong kiểm thử

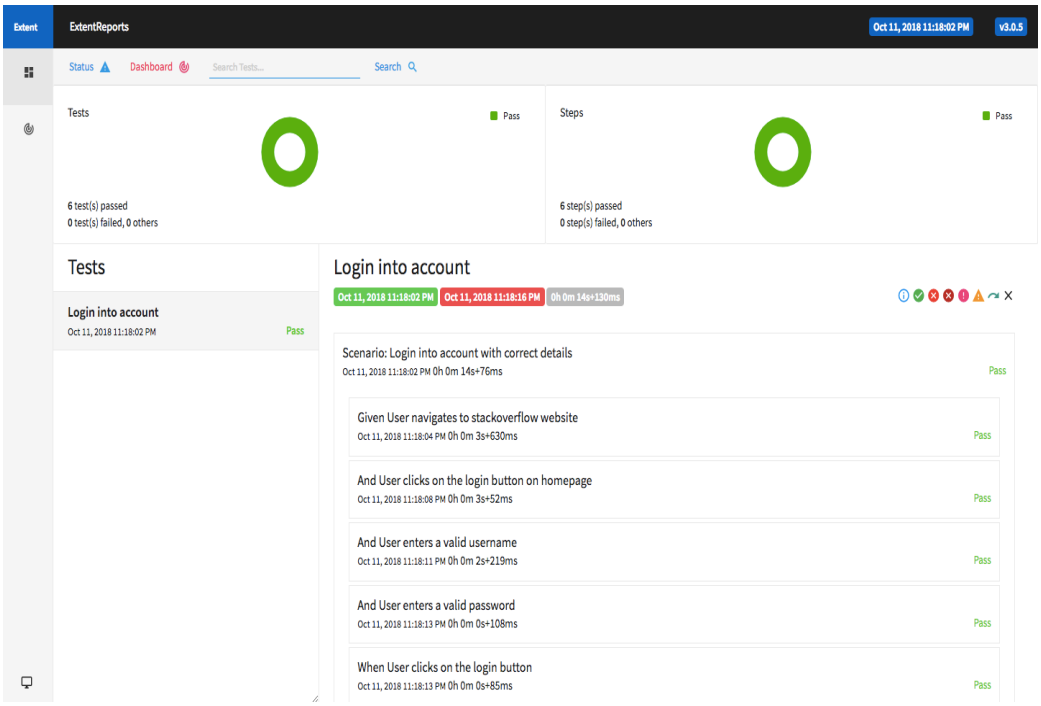


Hình 4-13. Cấu trúc sinh bảng báo cáo từ các feature file

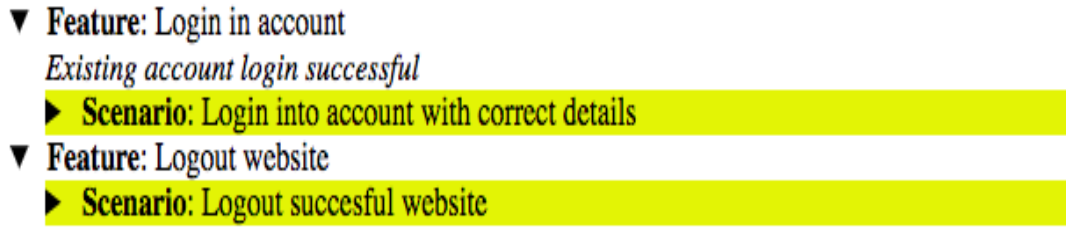


Hình 4-14. Cấu hình để sinh báo cáo kiểm thử

Kết quả dưới dạng file report.html cho một kịch bản kiểm thử:



Hình 4-15. Báo cáo kiểm thử dưới dạng report.html



Hình 4-16. Báo cáo kiểm thử dưới dạng html

4.3 Đánh giá Framework kiểm thử

Framework kiểm thử tự động trên với sự tích hợp của nhiều công cụ kiểm thử tự động và cách sử dụng phù hợp với kiểm thử ứng dụng web hoàn toàn có thể kiểm thử tự động theo hướng kiểm thử hướng hành vi, tạo thuận lợi cho nhiều bên liên quan và mang lại hiệu quả cao, tiết kiệm thời gian và nhân lực.

Với kịch bản kiểm thử được viết bằng ngôn ngữ tự nhiên, các bên liên quan trong dự án phát triển phần mềm dễ dàng có thể hiểu được dự án và theo dõi sự hoàn thiện của phần mềm mà cụ thể ở đây là kiểm thử ứng dụng web. Từ kết quả kiểm thử của ca kiểm thử đăng nhập vào 1 trang web ở hình [4-15] ta có thể thấy được thời gian kiểm thử của cả kịch bản kiểm thử đã viết trong feature là: 14s + 76 ms , và đồng thời của các bước trong kịch bản kiểm thử như sau:

Bảng 4-3 Thời gian chạy 1 kịch bản kiểm thử tự động

Bước kiểm thử	Thời gian
Given User navigates to stackoverflow website	3s + 630ms
User click on the login buton on homepage	3s + 52 ms
User enter valid username	2s + 219 ms
User enter valid password	108 ms
User click on the login button	85 ms
User should be taken to the successful login page	3s + 54ms

Từ kết quả kiểm thử, với thông số chi tiết về thời gian tự động chạy kịch bản kiểm thử khá nhanh và giảm thiểu được tối đa về thời gian kiểm thử. Với đặc trưng của kiểm thử chấp nhận là tốn khá nhiều chi phí về nguồn nhân lực và thời gian kiểm thử thì sử dụng công cụ kiểm thử tự động này làm giảm thiểu tối đa những chi phí không cần thiết trong kiểm thử.

Tuy nhiên, ở phần thân của các ca kiểm thử tự động sinh ra chỉ có các phương thực và tên hàm ánh xạ từ kịch bản kiểm thử hướng hành vi. Do vậy, cần sinh ra nhiều hơn các mã thực thi kiểm thử thì framework sẽ trở nên hữu ích hơn và đạt được mục đích hoàn toàn tự động trong kiểm thử hướng hành vi.

4.4 Phương pháp sinh mã kiểm thử tự động

Việc tạo mã tự động sẽ có tác động sâu sắc đến dự án phần mềm, chi phí phần mềm giảm, chu trình phát triển được rút ngắn, chất lượng phần mềm đạt được tối ưu. Trong phát triển phần mềm hướng hành vi, các kịch bản kiểm thử cung cấp thông tin ban đầu làm cơ sở cho thiết kế phần mềm và các bên liên quan. Tuy nhiên kịch bản các ca kiểm thử mô tả lại theo một cấu trúc nhất định để có thể tự động hoá sinh các ca kiểm thử tự động.

Vì vậy, trong nội dung này, luận văn đề xuất một phương pháp để sinh thân hàm các ca kiểm thử dựa trên quy trình xử lý ngôn ngữ tự nhiên, bước đầu sẽ tự động hoá chi tiết hơn quy trình kiểm thử.

Có thể thấy trong 1 ca kiểm thử dựa trên kịch bản kiểm thử hướng hành vi theo quy trình như sau:

Kịch bản -> Các bước kiểm thử -> Khung ca kiểm thử -> Thực thi

Ví dụ:

Scenario (Kịch bản):

Feature: Login in account

Existing account login successful

Scenario: Login into account with correct details

Step definition (Mô tả các bước)

Given User navigates to demo website

And User click on the button login on website homepage

And User enter a valid username

And User enter a valid password

When User click on the login button

Then User should be taken to the successful login page

Code skeleton (Khung các phương thức)

```
@Given("^User navigates to demo website$")
public void user_navigates_to_demo_website() throws
Throwable {
    // Write code here that turns the phrase above into
concrete actions
    throw new PendingException();
}

@Given("^User click on the button login on website
homepage$")
public void
user_click_on_the_button_login_on_website_homepage()
throws Throwable {
    // Write code here that turns the phrase above into
concrete actions
    throw new PendingException();
}

@Given("^User enter a valid username$")
public void user_enter_a_valid_username() throws Throwable
{
    // Write code here that turns the phrase above into
concrete actions
```

```
    throw new PendingException();
}
@Given("^User enter a valid password$")
public void user_enter_a_valid_password() throws Throwable
{
    // Write code here that turns the phrase above into
concrete actions
    throw new PendingException();
}
@When("^User click on the login button$")
public void user_click_on_the_login_button() throws
Throwable {
    // Write code here that turns the phrase above into
concrete actions
    throw new PendingException();
}
@Then("^User should be taken to the successful login
page$")
public void
user_should_be_taken_to_the_successful_login_page() throws
Throwable {
    // Write code here that turns the phrase above into
concrete actions
    throw new PendingException();
}
```

Implementation(Mã thực thi)

```
@When("^User navigates to demo website$")
     public void user_navigates_to_demo_website() throws
Throwable {
```

```
    // Write code here that turns the phrase above into
concrete actions
    driver.get("https://stackoverflow.com");
     }
     @And("^User click on the button login on website
homepage$")
     public void
user_click_on_the_button_login_on_website_homepage()
throws Throwable {
          //a[text()='Log In']
     Thread.sleep(200);
    driver.findElement(By.xpath("//a[text()='Log
In']")).click();
     }
     @And("^User enter a valid username$")
     public void user_enter_a_valid_username() throws
Throwable {
     Thread.sleep(200);
     driver.findElement(
By.xpath((".//*[@id='email']"))).sendKeys("autotestudemy@g
mail.com");
     }
     @And("^User enter a valid password$")
     public void user_enter_a_valid_password() throws
Throwable {
    // Write code here that turns the phrase above into
concrete actions
     Thread.sleep(200);
```

```
    driver.findElement(By.xpath(".//*[@id='password']")).s
endKeys("Password321!");
    }
    @When("^User click on the login button$")
    public void user_click_on_the_login_button() throws
Throwable {
    // Write code here that turns the phrase above into
concrete actions
    driver.findElement(By.xpath(".//*[@id='submit-
button']")).click();
    }
    @Then("^User should be taken to the successful login
page$")
    public void
user_should_be_taken_to_the_successful_login_page() throws
Throwable {
        // Write code here that turns the phrase above
into concrete actions

    }
```

Với Framework kiểm thử tự động ở mục trên, từ kịch bản kiểm thử viết bằng ngôn ngữ tự nhiên - > tự động sinh ra các phương thức kiểm thử (khung kiểm thử). Tuy nhiên với mã thực thi kiểm thử để tự động chạy được các ca kiểm thử, người phát triển phải tự viết các bước kiểm thử dưới dạng ngôn ngữ máy để chạy kịch bản tự động.

Vì vậy, luận văn đề xuất một phương án để sinh các mã kiểm thử tự động từ kịch bản dạng ngôn ngữ Gherkin, để sinh các mã kiểm thử tự động để có thể thực thi chương trình. Với ứng dụng của xử lý ngôn ngữ tự nhiên và áp dụng học máy, cần có một dữ liệu đủ lớn để từ kịch bản ngôn ngữ tự nhiên có thể tự động

sinh ra các mã kiểm thử theo dạng ngôn ngữ để framework tự động thực thi được các lệnh kiểm thử dựa trên kịch bản kiểm thử hướng hành vi bằng ngôn ngữ tự nhiên.

4.4.1 Phương pháp xử lý ngôn ngữ tự nhiên

Stanford parser (Phân tích cú pháp)

Stanford parser là một mã nguồn mở được công bố bởi nhóm Stanford NLP, nó phần tích cấu trúc câu trong những ngôn ngữ khác nhau và trả về một cấu trúc cây (PST) biểu diễn ngữ nghĩa của câu. PST là một cây không tuần hoàn với đỉnh gốc biểu diễn câu, chóp và lá biểu diễn cho cấu trúc ngữ pháp.….

Trong bài toán sinh mã kiểm thử tự động, Stanford parser đóng vai trò để xử lý cấu trúc của các câu mô tả kịch bản, nhóm các từ cùng nhau, phân tích ra cấu trúc ngữ pháp của câu. Ví dụ với các từ đi cùng nhau thì xác định từ nào là chủ thể hoặc là đối tượng của động từ. Xác suất phân tích đúng nhất dựa trên những kiến thức về ngôn ngữ để tạo ra khả năng cao nhất của câu mới, các phân tích của stanford parser còn một số lỗi, tuy nhiên độ chính xác khá cao. Trình phân tích này được xem như là một bước đột phá lớn nhất của xử lý ngôn ngữ tự nhiên trong năm 1990.

Cấu trúc của câu khi phân tích cú pháp sẽ có định dạng như sau:

Dưới đây là một áp dụng phân tích cho câu đơn giản: “When Customer click on the button login”.

Sử dụng StanFord Parser phân tích câu ta được:

```
(ROOT
  (SBAR
    (WHADVP (WRB When))
    (S
      (NP (NN customer) (NN click on))
      (VP (VBZ the)
        (NP (NN button) (NN login))))))
```

[nsubj(sentence-4, This-1), cop(sentence-4, is-2), det(sentence-4, another-3), root(ROOT-0, sentence-4)]

Có thể thấy, mỗi từ của câu là lá và được gắn một phần với ngôn ngữ tự nhiên. Có một nhóm các từ để tạo thành cụm thẻ từ.

4.4.2 Tập dữ liệu

Với mục đích xây dựng một tệp dữ liệu để khi chạy ca kiểm thử, chương trình tự động sinh ra các thân hàm trong hàm kiểm thử. Nội dung các bước trong mã kiểm thử được dùng để tự động chạy kiểm thử gồm các hàm và dữ liệu kiểm thử. Do vậy từ các gợi ý có trong kịch bản viết bằng ngôn ngữ tự nhiên cần ánh xạ được các thân hàm, các bước kiểm thử này khi kiểm thử trên web có nội dung khá trùng lặp trong các trường hợp giống nhau và hoàn toàn có thể xử lý với quy trình xử lý ngôn ngữ tự nhiên. Các dữ liệu dùng cho kiểm thử có khối lượng khá lớn và tuỳ thuộc vào từng dự án kiểm thử BDD.

Luận văn đưa ra một phương án đề xuất khi xây dựng tập dữ liệu kiểm thử như sau (với kịch bản kiểm thử ca kiểm thử đăng nhập vào trang web stackoverflow.com).

Bảng 4-4 Mô tả phương pháp sinh thân hàm cho kịch bản đăng nhập trên trang stackoverflow.com

Kịch bản kiểm thử	Thân hàm cần sinh	Suy ra từ NLP	Từ điển cho phần còn thiếu
User navigates to stackoverflow website	driver.get("https://stackoverflow.com/");	navigate -> driver.get($1);	stackoverflow website -> "https://stackoverflow.com/"
User clicks on the login button on homepage	driver.findElement(By.xpath("//a[text()='Log In']")).click();	click -> driver.findElement($1)).click();	login button -> By.xpath("//a[text()='Log In']"
User enters a valid username	driver.findElement(By.xpath(".//*[@id='email']")).sendKeys("nhungnguyen.uet@gmail.com");	enters -> driver.findElement($1).sendKeys($2);	valid username -> "nhungnguyen.uet@gmail.com" username-> By.xpath(".//*[@id='email']")
User enters a valid password	driver.findElement(By.xpath(".//*[@id='password']")).sendKeys("nhung@93");	enters -> driver.findElement("$1").sendKeys("$2");	valid password -> "nhung@93" password -> By.xpath(".//*[@id='password']"
User clicks on the login button	driver.findElement(By.xpath(".//*[@id='submit-button']")).click();	click -> driver.findElement($1)) .click()	login button -> By.xpath(".//*[@id='submit-button']")

Với kịch bản kiểm thử cho ca kịch bản đăng nhập trên trang web demo ta có mô tả như sau:

Bảng 4-5 Mô tả phương pháp sinh thân hàm cho kịch bản đăng nhập vào trang web demo

Kịch bản kiểm thử	Thân hàm cần sinh	Suy ra từ NLP	Từ điển cho phần còn thiếu
User navigates to demo website	driver.navigate().to(“ http://www.demo.guru99.com/V4/ ");	navigate -> driver.get($1);	wikipedia website -> " http://en.wikipedia.org "
User enter a valid username	driver.findElement(By.xpath(".//*[@name='uid']")).sendKeys("testuser_1");	Enter-> driver.findElemet($1).senKey($2)	valid username -> “testuser_1”
User enter a valid password	driver.findElement(By.xpath((".//*[@name='password']"))).sendKeys("Test@153 ");	Enter -> driver.findElement($1).senkey($2)	Valid password -> “Test@153”
User click on the login button	driver.findElement(By.xpath("html/body/form/table/tbody/tr[3]/td[2]/input[1]")).click();	Click -> driver.findElement(By.xpath("$1")).click();	Login button -> "html/body/form/table/tbody/tr[3]/td[2]/input[1]"

Từ các bảng [4-4] [4-5] có thể thấy, ta có thể áp dụng xây dựng từ điển dữ liệu cho các ca kiểm thử ở các nội dung kiểm thử khác. Như vậy sẽ giúp giảm thiểu đáng kể thời gian xây dựng chương trình để tự động sinh ra mã kiểm thử từ kịch bản kiểm thử viết bằng ngôn ngữ Gherkin.

Tuy nhiên thân hàm sinh ra có thể còn nhiều lỗi hoặc chỉ là các gợi ý trong thân hàm kiểm thử, ở đây người lập trình cần hoàn thiện lại mã thực thi để chương trình kiểm thử tự động có thể chạy được.

KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

1. Kết luận

1.1 Lý thuyết

Luận văn nghiên cứu về kiểm thử tự động, quy trình phát triển phần mềm hướng thành phần, cụ thể đi sâu vào kiểm thử hướng hành vi trong quy trình phát triển phần mềm theo hướng Agile. Những lợi ích, thuận lợi khi sử dụng kiểm thử hướng hành vi trong phát triển phần mềm, quy trình xây dựng phần mềm theo hướng kiểm thử hướng hành vi. Tìm hiểu về đặc trưng trong kiểm thử tự động hướng hành vi và các ưu điểm, khuyết điểm cũng như đánh giá khả năng áp dụng kiểm thử tự động hướng hành vi. Các phần lý thuyết liên quan về xử lý ngôn ngữ tự nhiên khi sinh mã kiểm thử từ kịch bản kiểm thử bằng ngôn ngữ tự nhiên, đưa ra phương pháp sinh hàm trong mã kiểm thử tự động.

1.2 Thực nghiệm

Dựa trên kết hợp công cụ kiểm thử hướng hành vi là Cucumber, kết hợp công nghệ Java và nền tảng kiểm thử Web với Selenium, luận văn đã thực thi kiểm thử hướng hành vi một cách tự động các ca kiểm thử tự động dựa trên kịch bản được viết dưới dạng ngôn ngữ tự nhiên là ngôn ngữ Gherkin, xây dựng, thu thập, thực nghiệm sinh mã kiểm thử tự động. Luận văn cũng thực thi tự động kiểm thử các ca kiểm thử trên nền web, sinh báo cáo kiểm thử và đưa ra kết quả, đánh về tính hiệu quả của công cụ kiểm thử hướng hành vi.

Kết quả chạy các ca kiểm thử đã chứng minh tính đúng đắn, khả năng ứng dụng phát triển tự động kiểm thử theo hướng hành vi tự động cho phần mềm. Giảm thiểu thời gian cũng như chi phí phát sinh trong quy trình phát triển phần mềm, tăng tính hiệu quả quả kiểm thử tự động trong phát triển phần mềm hướng hành vi.

2. Hướng phát triển

Trong luận văn tìm hiểu về các công cụ kiểm thử tự động trong kiểm thử hướng hành vi, đồng thời kết hợp các công cụ kiểm thử tự động để xây dựng, triển khai kiểm thử tự động hướng hành vi. Dựa trên tìm hiểu các nguyên tắc xây dựng công cụ, vận hành công cụ, phần nào hiểu được cách phát triển và cơ sở để làm một công cụ kiểm thử tự động trong kiểm thử tự động hướng hành vi. Vì vậy, trong thời gian tới tôi sẽ tiếp tục nghiên cứu để phát triển công cụ kiểm thử tự động với mã kiểm thử tự động kiểm thử hoàn thiện, đầy đủ hơn trong quy trình phát triển phần mềm hướng hành vi hoặc tích hợp các công cụ kiểm thử một cách hiệu quả và chính xác nhất.

TÀI LIỆU THAM KHẢO

[1] Melanie Diepenbeck, Mathias Soeken, Daniel Große, Rolf Drechsler - "Towards Automatic Scenario Generation from Coverage Information", 2013

[2] Mathias Soeken, Robert Wille, Rolf Drechsler - "Assisted Behavior Driven Development Using NLP", 2012

[3] Prerana Pradeepkumar Rane - "Automatic generation of Test Cases for Agile using NLP", 2017

[4] Https://medium.com/agile-vision/behavior-driven-development-bdd-software- testing-in-agile-environments-d5327c0f9e2d

[5] Sunil Kamalakar - "Automatically Generating Tests from Natural Language Descriptions of software Behavior", 2013

[6] http://toolsqa.com/cucumber/

[7] https://jasmine.github.io

[8] https://en.wikipedia.org/wiki/Behavior-driven_development

[9] https://www.udemy.com/

[10] https://www.guru99.com/xpath-selenium.html

[11] Victor Szalvay, Danube Technologies - "An Introduction to Agile Software Development" ,2004

[12] Lasse Koskela - "Test First model-driven development", 2008

[13] Dan North - "Introducing BDD". March 2006.

[14] David Chelimsky, Dave Astels, Zach Dennis, Aslak Hellesøy, Bryan Helmkamp, Dan North - "The Rspec Book Behavior Driven Development with Rspec, Cucumber and Friends", 2010

[15] https://techblog.vn/index.php/tong-quan-ve-kiem-thu-tu-dong-tdd-va-bdd-trong-mo-hinh-phat-trien-phan-mem-agile