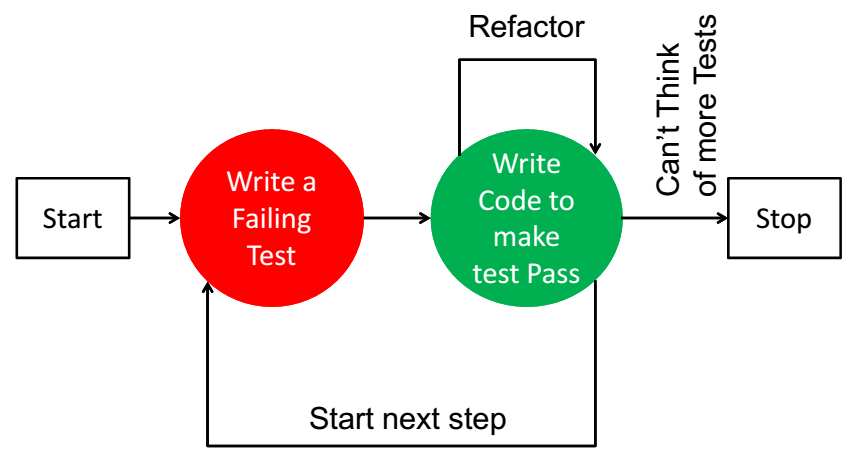
1. **Agile Software Development:**

* Agile là phương pháp phát triển phần mềm linh hoạt. Trong phương pháp này ta sẽ:
* Đề cao sự tương tác giữa các cá nhân với nhau hơn là các quy trình, các công cụ.
* Phần mềm chạy được hơn là chứng minh bằng các tài liệu.
* Tập trung vào hợp tác, thường xuyên giao tiếp với khách hàng hơn là dựa trên những đàm phán được ghi trên hợp đồng.
* Thường xuyên thay đổi trong quá trình phát triển hơn là làm theo một kế hoạch định sẵn.
* Sau đây là một số nguyên tắc trong phương pháp Agile:
* Tương tác với khách hàng thường xuyên là cần thiết để có thể hiểu rõ các yêu cầu của khách hàng vì các yêu cầu này thường khó hiểu.
* Các yêu cầu về dự án phải rõ ràng, chứa đủ thông tin để giúp tất cả các thành viên trong nhóm hiểu rõ những gì mình đang làm.
* Chia dự án thành các phần nhỏ và phát triển từng phần đó vì một dự án thường quá lớn.
* Các chức năng phải được hoàn thành hoàn toàn một chức năng (được kiểm tra, thiết kế và được khách hàng chấp nhận) trước khi chuyển qua một chức năng khác, nghĩa là không được làm 90% và xem nó là đã xong và giao cho khách hàng.
* Thường xuyên giao các phần mềm chạy được cho khách hàng để khách hàng đánh giá.
* Thường xuyên kiểm tra xuyên suốt dự án.
* Sự hợp tác giữa các thành viên trong quá trình phát triển dự án là cần thiết.

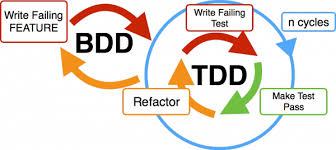
1. **Test Driven Development (TDD):**

* TDD là một phương pháp phát triển phần mềm dựa trên phương pháp agile tập trung vào việc kiểm thử phần mềm vì vậy mà các phần mềm được áp dụng phương pháp này thường ít có lỗi.
* Lập trình viên sử dụng phương pháp này sẽ thực hiện theo từng bước nhỏ:
* Viết test cho 1 hàm nhưng đảm bảo cho nó thất bại.
* Sau đó thì viết code để có thể pass test.
* Sau đó thì tối ưu hóa code.
* Lặp lại bước đầu tiên.
* Mô hình TDD:

****

1. **Behavior driven develop (BDD):**

* BDD là một phương pháp phát triển phần mềm được phát triển từ TDD.
* BDD sẽ dựa vào các yêu cầu của khách hàng để viết ra các kịch bản (Scenarios) để test, sau đó ta sẽ bắt đầu viết code để có thể pass các kịch bản này.
* Các kịch bản sẽ được viết bằng ngôn ngữ tự nhiên, dễ hiểu để có thể tăng tính cộng tác, giao tiếp trong nhóm phát triển phần mềm.
* Mô hình BDD:



1. **Gherkin**
2. **Định nghĩa:**

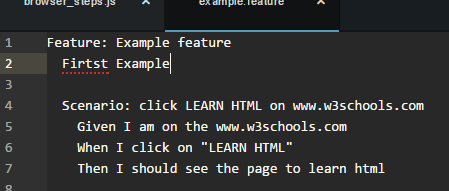
* Gherkin là ngôn ngữ dễ hiểu, rõ ràng đối với cả những người không rành về lập trình, được viết dưới dạng văn bản thuần túy, được thiết kế để phản ánh các bước trong nghiệp vụ.

1. **Các quy tắc cơ bản:**

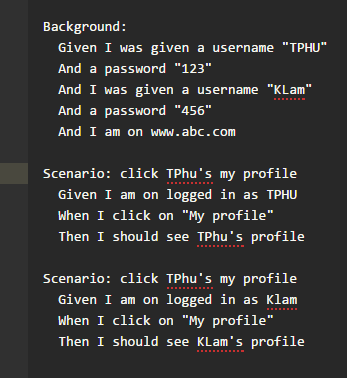
* Mỗi file chỉ mô tả cho 1 tính năng và được lưu với định dạng “.feature”.

1. **Cú pháp:** gồm feature scenario và step.

* Feature thường bao gồm các list **Scenario**.Băt đầu bằng từ khóa “**Feature:** ” Người sử dụng có thể ghi mô tả 1 cách tường minh tính năng bằng ngôn ngữ tự nhiên. Có thể dùng tag để nối các feature lại với nhau.
* **Scenario** bắt đầu với từ khóa “**Scenario:** ” + một tiêu đề tùy ý. Mỗi **Scenario** sẽ bao gồm nhiều **step**.



* **Background** cho phép chúng ta thêm ngữ cảnh cho tất cả các Scenarios trong cùng một file feature.
* **Steps**: Gồm 3 loại chính Given, When, Then
* **Given:** mô tả bối cảnh ban đầu của hệ thống, cấu hình hệ thống ở 1 trạng thái rõ ràng : tạo, cập nhật dữ liệu test.
* **When:** Mô tả hành động của người sử dụng.
* **Then:** Mô tả kết quả test mong muốn.
* Ngoài ra nếu có nhiều **Given, When, Then** thì có thể có thêm **And, But**(các từ khóa này giúp thay thế sự lặp lại when/then và giúp code nhìn mạch lạc hơn).



* Ngoài ra còn có các từ khóa :
* Scenario Outline & Example:  
  Scenario Outline : dùng để gom nhóm các kịch bản có chung steps nhưng nhiều input và output. Các steps trong Scenario Outline cung cấp template mẫu và không bao giờ được chạy trực tiếp mà các Example sẽ nhận các biến input và chạy trên template này. Các biến được khai báo theo cú pháp “<tên biến>”.  
  Example : Dùng để khai báo giá trị các biến trong Scenario Outline.
* Tag: dùng để quản lý, sắp xếp các tính năng (feature) và kịch bản (scenario). Tag có tính kế thừa, bất cứ tag nào ở mức Feature đều được kế thừa xuống Scenario, Scenario Outline, Example.   
  Người dùng có thể triển khai các Scenario có tag @tag bằng câu lệnh: cucumber - -tags @tag hoặc ngược lại triển khai các các Scenario không có tag bằng lệnh : cucumber - - tags ~@tag.  
  Bên cạnh đó người dùng có thể dùng các phép and hoặc or với tag.  
   Đối với phép OR thì cú pháp là :   
  cucumber - - tags @A,@B.   
  Đối với phép And : cucumber - - tags @A - - tags @B

1. **Cucumber:**

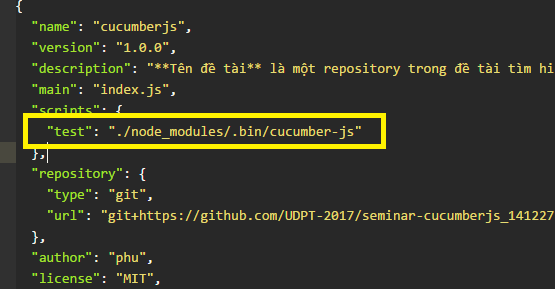
* Cucumber là một công cụ được dùng để kiểm tra tự động dựa trên việc thực hiện các chức năng, kiểm tra xem các chức năng này thành công hay thất bại so với các kịch bản, các bước trong kịch bản mà ta định sẵn và công cụ này hỗ trợ cho BDD.
* Ngôn ngữ được Cucumber sử dụng là ngôn ngữ Gherkin.

1. **CucumberJS:**
2. **Tổng quan:**

* CucumberJS là một phần của Cucumber và nó được dùng để kiểm tra các kịch bản và các bước trong kịch bản được định nghĩa bằng JavaScript.
* CucumberJS sẽ thực thi file loại .feature và trong file này sẽ chứa các kịch bản, các bước được viết bằng Gherkin.
* Trong Gherkin sẽ có các từ khóa feature, scenario, steps: given, when, then, and, but; background, scenario outline, examples.
* Sau khi tạo file .feature ta sẽ đến bước định nghĩa các bước vì CucumberJS sẽ không hiểu cách để thực thi file .feature chứa các scenario, steps (vì chúng là văn bản thuần túy) vì vậy ta phải định nghĩa các bước (Step Definitions) để CucumberJS có thể hiểu và thực thi. Trong CucumberJS thì ta định nghĩa các bước bằng JavaScript.

1. **Cài đặt:**

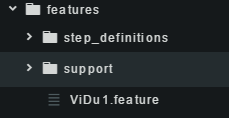
* Để cài đặt ta chỉ cần sử dụng dòng lệnh:
* npm install cucumber
* npm install --save-dev cucumber@latest selenium-webdriver@3.0.1 [chromedriver@2.25.1](mailto:chromedriver@2.25.1) (để sử dụng trong nodejs)
* Sau đó ta vào package.json thêm:

****

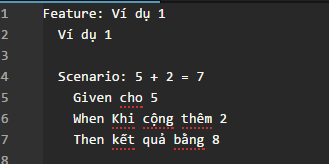
* Để test ta gõ lệnh: npm test

1. **Ví dụ 1:**

* Đầu tiên là ta tạo folder features.

****

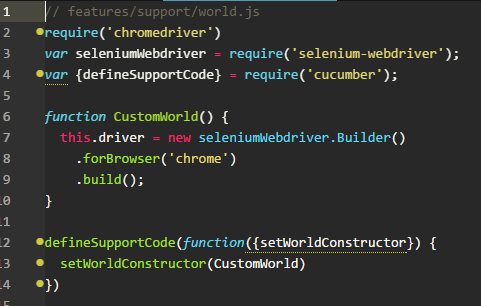
* Sau đó ta thêm file ViDu1.feature

****

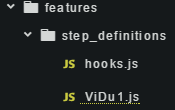
* tiếp theo ta thêm file world.js, trước hết chúng ta phải tạo folder support trong folder features rồi thêm file world.js vào folder support

****

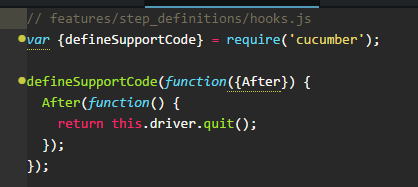
* nội dung của file world.js:

****

* tiếp theo ta sẽ tạo folder step\_definitions trong folder features, rồi thêm file hooks.js vào thư mục step\_definitions

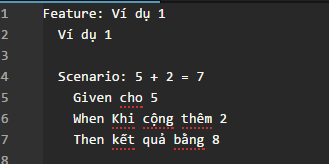
****

* nội dung của file hook.js:

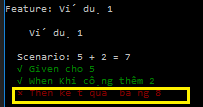
****

* tiếp theo ta sẽ tới bước định nghĩa cho file ViDu1.feature, ta sẽ tạo file ViDu1.js trong thư mục step\_definitions.

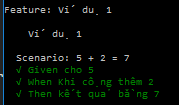
****

****

* setDefaultTimeOut để chỉnh sửa lại thời gian test như vậy sẽ không gây ra lỗi timeout.
* Kết quả:

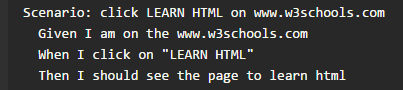
****

* Bước 3 chạy sai là vì ta định nghĩa trong file feature là 5 + 2 = 8 nhưng khi test thì 5 + 2 lại bằng 7 nên mới báo lỗi
* Vì vậy chỉnh sửa lại trong file feature 5 + 2 = 7 và ta sẽ có kết quả đúng

****

1. **Ví dụ 2:**

* Cho scenario:

****

* Step definition:

****

* Kết quả:

**Untitled.png**

1. **Continuous Intergration:**

* Continuous Intergration (CI) là một phương pháp phục vụ cho Align, nơi mà các lập trình viên trong một nhóm gửi code của mình lên và sau đó CI sẽ tự động được build để xác định các lỗi và ngay lập tức gửi lỗi để cho lập trình viên sửa lỗi như vậy sẽ giúp team giảm thiểu tối đa các lỗi khi tích hợp code với nhau và giúp xây dựng phần mềm nhanh hơn nhiều.
* Có thể áp dụng CI thông qua các framework TFS, Codeship, TeamCity, Hudson, Circle, Jenkin, Travis, …

1. **Acceptance Test, Unit Test, Integration Test:**

* Unit test được viết bởi các lập trình viên để kiểm tra xem một đơn vị code hay một thành phần có hoạt động chính xác hay không.
* Integration Test được viết để kiểm tra các unit khi tích hợp lại với nhau thì có hoạt động đúng như mong muốn hay không.
* Acceptance Test được viết bởi bộ phận kinh doanh và QA và bài test này sẽ kiểm tra xem chương trình có thực hiện đúng hay không.

1. **Tài liệu tham khảo:**

* <http://blog.co-mit.com/post/9/T%C3%ACm+hi%E1%BB%83u+m%C3%B4+h%C3%ACnh+TDD+(Test+-+Driven+Development)+v%C3%A0+c%C3%A1ch+%C3%A1p+d%E1%BB%A5ng>
* <https://code4shares.wordpress.com/2016/07/20/ap-dung-behavior-driven-develop-cho-du-an-agile-phan-1/>
* <http://labs.septeni-technology.jp/bdd/kiem-thu-tu-dong-su-dung-bdd-2/>
* <http://www.allaboutagile.com/what-is-agile-10-key-principles/>
* <https://www.custardbelly.com/blog/blog-posts/2014/01/08/bdd-in-js-cucumberjs/>
* <https://cucumber.io/docs/reference>
* <https://toidicodedao.com/2015/08/27/giai-thich-don-gian-ve-ci-continuous-integration-tich-hop-lien-tuc/>
* <https://www.martinfowler.com/articles/continuousIntegration.html>
* <https://viblo.asia/Thanh/posts/jamoG8ndRz8P>
* <http://labs.septeni-technology.jp/bdd/kiem-thu-tu-dong-su-dung-bdd-2/>
* <https://github.com/cucumber/cucumber/wiki/Tags>
* <https://github.com/cucumber/cucumber/wiki/Scenario-Outlines>

https://docs.google.com/forms/d/e/1FAIpQLScmsjm88T8VjBzClPjoC11BJSE58aI0VyQ7YnHvCyQ3MA0pog/viewform?edit2=2\_ABaOnueZsXiEH\_fiDV4ddIAhju-Yogt0MvCQEJQaFDVPX966feCV9aWvPcRvM8E