1. PHÂN TÍCH VÀ GIẢI QUYẾT BÀI TOÁN NÂNG CẤP CÔNG CỤ SINH UNIT TEST TỰ ĐỘNG [(30% )](#__RefHeading__2810_1527130935)

Dựa vào kết quả tìm hiểu, sử dụng và phân tích các tính năng cũng như ưu nhược điểm của Randoop, bài toán đặt ra là làm sao phát huy được công dụng sinh unit test tự động của Randoop, đồng thời giải quyết được những khó khăn khi sử dụng công cụ này. Để có được bài toán đặt ra, cần tiến hành khảo sát và đưa ra các vấn đề còn tồn tại khi sử dụng công cụ này. Một vài khó khăn đã được tìm ra như áp dụng Randoop vào các dự án lớn gồm nhiều class, method hay sử dụng đầu vào đầu ra theo ý muốn,… Từ các khó khăn còn tồn tại, hướng giải quyết được đưa ra là tạo ra một công cụ UET TestGen có khả năng khắc phục các khó khăn đó. Tất cả các vấn đề đó sẽ được giải quyết trong chương 3.

# Bài toán đặt ra:

# Randoop giải quyết được vấn đề gì trong dự án thực tế?

Để thấy rõ được những giá trị mà Randoop đem lại, cần phải áp dụng Randoop vào dự án thực tế. Ở đây, tôi đã tiến hành thực hiện sinh unit test trong một dự án thực tế có tên Vsmart tại Trung tâm Phát triển Phần mềm thuộc Tập đoàn Viễn thông quân đội Viettel. Đây là một dự án lớn được lập trình bằng ngôn ngữ Java với tổng cộng…. file java, …class,… method, …dòng code. Việc sử dụng Randoop nhằm tự động sinh ra các Unit test đã đem lại những hiệu quả, lợi ích rõ rệt không chỉ về chất lượng mà còn về chi phí.

# Về chất lượng

Chất lượng của các unit test được sinh ra là điều cần được quan tâm hàng đầu bởi mục tiêu cuối cùng của chúng ta vẫn là tạo ra được các unit test chất lượng nhất. Dựa vào các tiêu chí đánh giá một unit test đạt tiêu chuẩn (đã nói kỹ trong chương 2), có thể đánh giá được chất lượng các unit test được sinh ra từ Randoop.

(Hình ảnh ví dụ về một unit test + code implement)

# Về chi phí

Bên cạnh việc đáp ứng được những unit test đạt tiêu chuẩn, Randoop còn đem lại nhiều giá trị to lớn. Đó là giảm thiểu các khoản chi phí đầu tư về thời gian, công sức, nhân lực cũng như cơ sở vật chất. Thực tế, đây là mối quan tâm hàng đầu của tất các doanh nghiệp trong linh vực công nghê thông tin.

# Thời gian, công sức

Việc viết các unit test thực tế đòi hỏi người viết unit test phải có kiến thức chuyên môn và nghiệp vụ. Ngoài các kiến thức cơ bản về unit test, người viết unit test phải hiểu được các phương thức cần xây dựng trong dự án bao gồm đầu ra, đầu vào, các đối tượng sử dụng bên trong phương thức và tính logic giữa chúng. Các yêu tố này luôn thay đổi với mỗi phương thức test. Vì vậy, mỗi lần viết unit test cho một phương thức mới, người viết unit test chắc chắn phải bỏ một lượng thời gian, công sức để tìm hiểu các yếu tố này. Bên cạnh đó, các thư viện hỗ trợ trong việc viết unit test cũng cần được tìm hiểu kỹ lưỡng và biết cách sử dụng.

Ngoài thời gian, công sức để tìm hiểu các kiến thức cần có, viết unit test cũng chiếm lượng lớn thời gian và công sức, đòi hỏi người viết unit test phải bỏ một lượng lớn chất xám để suy nghĩ.

Sử dụng công cụ Randoop, thời gian và công sức bỏ ra cho unit test đã giảm thiểu đáng kể đặc biệt là đối với các dự án thực tế có quy mô lớn, có giới hạn thời gian hoàn thành dự án. Bên cạnh đó, người sử dụng Randoop sinh unit test không cần có quá nhiều kiến thức về unit test, cũng không cần tốn thời gian, công sức tìm hiểu thêm về ngôn ngữ lập trình, các thư viện hay các thành phần trong một phương thức cần test. Để sử dụng công cụ này, người dùng chỉ cần biết cách chạy một dòng lệnh java theo mẫu, một kỹ năng cơ bản của các lập trình viên khi lập trình java.

# Nhân lực và cơ sở vật chất

Việc tốn thời gian, công sức viết unit test, đặc biệt là các unit test trong các dự án lớn chứa hàng nghìn phương thức lớn nhỏ, sẽ dễ khiến cho dự án bị chậm tiến độ. Để kịp hoàn thành tiến độ, việc bổ sung thêm nhân lực là hoàn toàn cần thiết. Điều đó tạo thêm nhiều khó khăn cho doanh nghiệp như tuyển dụng nhân sự, lương và chi phí đầu tư thêm cơ sở vật chất cho nhân viên làm việc. Nhìn chung, Randoop giảm thiểu được đáng kể khoản đầu tư không mong muốn này.

(Sơ đồ biểu diễn lượng coverage và phân tích)

# Các vấn đề gặp phải khi sử dụng Randoop trong dự án thực tế

Bên cạnh những giá trị mà Randoop đem lại, bằng việc tìm hiểu và thực nghiệm Randoop trong dự án thực tế, có thể nhận thấy những vấn đề còn tồn tại trong công cụ này.

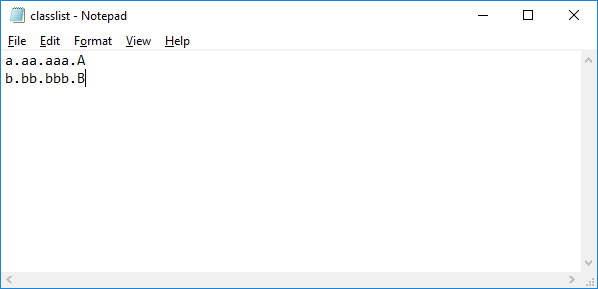
# Tốn thời gian khi sinh unit test cho một lúc nhiều phương thức

Thực tế, Randoop có hỗ trợ tính năng sinh unit test tự động cho nhiều lớp hay phương thức cùng lúc. Để sử dụng tính năng này, người dùng cần sử dụng một file text liệt kê ra các lớp hay phương thức cần test.

Ví dụ: Cho hai lớp sau:

* Class “A” nằm trong package “a.aa.aaa”
* Class “B” nằm trong package “b.bb.bbb”

Để sinh unit test cùng lúc cho hai lớp trên, chúng ta cần tạo một file text liệt kê hai class. File text đó có dạng như sau:

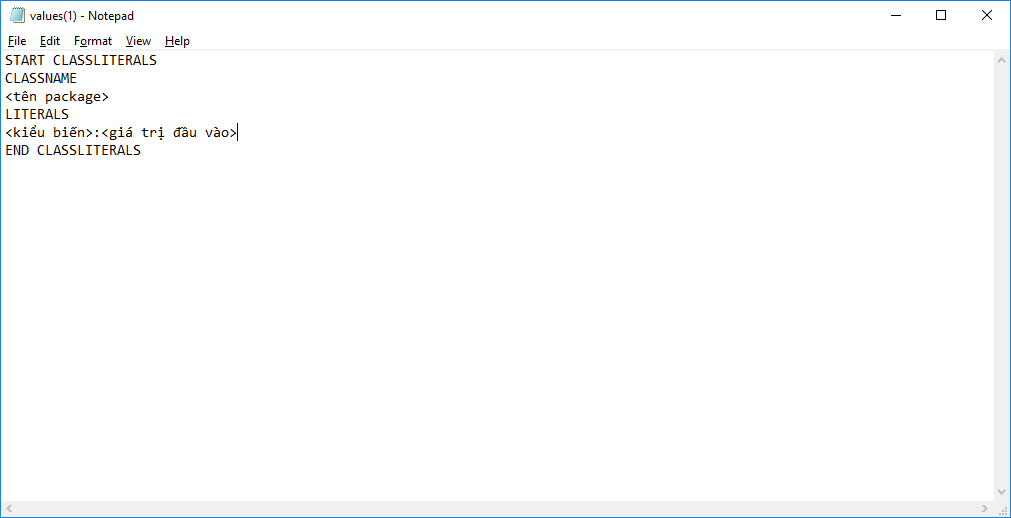


Sau đó, chúng ta cần thêm một argument vào câu lệnh: --classlisst=<tên file>. Chạy câu lệnh sẽ thu được các unit test của cả hai lớp trên.

Có thể nói, tính năng sinh unit test tự động cho cùng một lúc nhiều lớp hay phương thức là một tính năng hữu ích giúp cho người dung tiết kiệm được thời gian chạy nhiều lần các dòng lệnh có định dạng tương tự. Tuy nhiên, có thể thấy rằng, mặc dù tiết kiệm được thời gian thực hiện thao tác này, người dung vẫn cần mất một lượng thời gian để liệt kê ra các lớp và phương thức cần test. Đối với một project đơn giản chỉ gồm 10 hay 20 lớp, việc liệt kê ra các lớp hay phương thức không tốn quá nhiều thời gian. Tuy nhiên, đối với những project lớn chứa hang tram lớp, hang nghìn phương thức, việc liệt kê bằng tay các lớp/phương thức ra file txt sẽ chiếm khá nhiều thời gian, công sức. Bên cạnh đó, đây còn là thao tác đơn giản, công việc lặp đi lặp lại nhàm chán, hoàn toàn không tương xứng với năng lực của một lập trình viên, gây ra việc lãng phí chất xám.

# Thêm đầu vào cho phương thức với dạng file mẫu không thân thiện người dùng

Tương tự tính năng sinh unit test cho nhiều lớp/phương thức cùng lúc, tính năng cho phép nhập đầu vào cho các phương thức theo ý muốn cũng được sử dụng thông qua file text có định dạng như sau:



Vấn đề đặt ra cũng tương tự như với tính năng sinh unit test cho nhiều lớp/phương thức. Với mỗi lần cần tạo đầu vào cho phương thức cần sinh unit test, người dung lại cần tạo một file text như định dạng trên. Một lần nữa, tính năng hữu ích này của Randoop lại vô tình tạo ra một công việc tốn kém thời gian và lãng phí chất xám của người dùng.

# Không hỗ trợ mock/fake các đối tượng.

Trong quá trình viết unit test, một trong những kỹ năng cần thiết đó là mock/fake các đối tượng. Việc mock/fake các đối tượng nhằm tạo ra các đối tượng tương tự các đối tượng được sử dụng trong phương thức. Trong các phương thức có liên kết với nhiều đối tượng khác, việc gọi trực tiếp tới các phương thức đó, đặc biệt là các service hay các phương thức truy vấn tới database, sẽ gây mất thời gian, tốn tài nguyên và không cần thiết. Vì thực tế, mục đích của người test là test phương thức hiện tại. Mock/fake các đối tượng sẽ giúp cô lập các unit test của mỗi phương thức, tiết kiệm tài nguyên mà vẫn đảm bảo được kết quả công việc.

Tuy nhiên, trong các unit test mà Randoop sinh ra, tính năng này vẫn chưa được sử dụng mà vẫn trực tiếp gọi vào các đối tượng đó. Đây là thiếu sót lớn của công cụ này.

# Giải quyết bài toán đặt ra

Từ những nhược điểm còn tồn tại đã nêu, hướng giải quyết của tôi là xây dựng một công cụ nhằm khắc phục những nhược điểm đó, giúp người dung rút ngắn được thời gian, công sức khi tiếp cận với Randoop. Tôi sẽ trình bày kỹ hơn về công cụ hỗ trợ này ngay sau đây.

# Giới thiệu về UET TestGen

# UET TestGen là gì?

Trước những nhược điểm còn tồn tại ở Randoop, một công cụ đã được cho ra đời và phát triển với tên gọi UET TestGen. Đây là công cụ được phát triển nhằm mục đích khắc phục những khuyết điểm của Randoop, giúp người dung có thể tận dụng các tính năng của Randoop một cách dễ dàng hơn. UET TestGen được viết bởi ngôn ngữ Java (version 8), nhằm tương thích tối đã với Randoop, một công cụ cũng được phát triển bằng ngôn ngữ này.

# Các tính năng của UET TestGen

# Sinh classlist

Như đã trình bày ở phần 1, để sử dụng tính năng sinh unit test cho nhiều lớp cùng lúc, người dùng phải tạo một file txt chưa tên của các lớp đó. UET TestGen đã khắc phục nhược điểm này bằng cách giúp người dùng tự động sinh ra file classlist đó. Người dùng sẽ không còn phải kiểm tra bằng mắt, liệt kê bằng tay các lớp vào file classlist.txt mà công cụ sẽ giúp người dùng tự động sinh ra file classlist.txt chứa toàn bộ các class theo đúng định dạng yêu cầu của Randoop.

# Sinh methodlist

Tương tự như file classlist, UET TestGen cũng giúp người dùng tự động sinh ra file methodlist.txt chứa toàn các phương thức cần sinh unit test theo đúng định dạng yêu cầu.

# Hỗ trợ giao diện nhập giá trị đầu vào và sinh file value

Khi người dùng muốn tạo những giá trị đầu vào cho phương thức cần test, UET TestGen sẽ hỗ trợ giao diện để người dùng nhập các giá trị. Giao diện này được xây dựng dựa trên một bảng tính Excel, một giao diện quen thuộc của các tester khi làm công việc kiểm thử của mình. Tại đây, các phương thức đã được liệt kê sẵn. Người dùng chỉ cần tìm kiếm tên phương thức, nhập các giá trị đầu vào cho phương thức cần test. Sau đó UET TestGen sẽ tự động sinh ra các file value tương ứng với mỗi phương thức. File value này có định dạng như Randoop yêu cầu.

# Phân tích bài toán: đầu vào, đầu ra của mỗi tính năng

Mỗi tính năng của UET TestGen đều được phân tích không chỉ dựa trên những ưu nhược điểm của Randoop, mà còn dựa trên tính khả thi. Đã có rất nhiều các phương hướng giải quyết được đặt ra. Tuy nhiên, để lựa chọn phương hướng tối ưu nhất là một khó khăn. Phần tiếp đây sẽ mô tả chi tiết hơn về cách thức xây dựng chương trình UET TestGen cũng như lý giải nguyên nhân các phương hướng giải quyết đó được lựa chọn.

# Cấu trúc chương trình và luồng hoạt động

Luồng hoạt động của UET TestGen có thể được chia ra làm 3 giai đoạn chính:

* Giai đoạn 1: Chương trình nhận đầu vào lần thứ nhất là các file java chứa các class cần test. Những file java này được đọc và bóc tách các ký tự bên trong. Trong quá trình này, chương trình sẽ xác định được tên package, tên class, tên method và các kiểu giá trị đầu vào của method đó và lưu lại vào một list các đối tượng có định dạng Package.
* Giai đoạn 2: Các đối tượng Package đã đọc được ở giai đoạn 1 sẽ được ghi vào một file Excel có định dạng đã định. Chương trình tạo ra file Excel này và nhận đầu vào lần thứ hai bằng cách cho phép người dùng nhập giá trị đầu vào cho các method theo mong muốn.
* Giai đoạn 3: Từ file Excel, chương trình hoàn thiện các package bằng cách bổ sung các giá trị đầu vào mà người dùng đã nhập. Package hiện tại bao gồm: tên package, tên class, tên method, tên kiểu giá trị đầu vào và các giá trị đầu vào mong muốn. Cuối cùng, chwowng trình tạo ra các file classlist, methodlist, value theo mẫu chuẩn Randoop.

File classlist

File methodlist

File value

File Excel

Giá trị đầu vào

File Java cần test

Package không chứa value

Package có chứa value

CHÚ THÍCH

|  |  |
| --- | --- |
|  | Luồng hoạt động |
|  | Package |
|  | Đầu vào |
|  | Đầu ra |
|  | File Excel |

*Sơ đồ cấu trúc hoạt động chương trình UET TestGen*

# Cấu trúc form Package

Để chương trình được thiết kế và lập trình một cách dễ dàng, đồng thời thân thiện với người dùng nhất có thể, điều quan trọng là phải lựa chọn một định dạng phù hợp cho file Excel. Sau nhiều lần thử nghiệm với nhiều định dạng khác nhau, một chuẩn định dạng đã được đưa ra (hình). Định dạng này được chia làm 2 phần chính

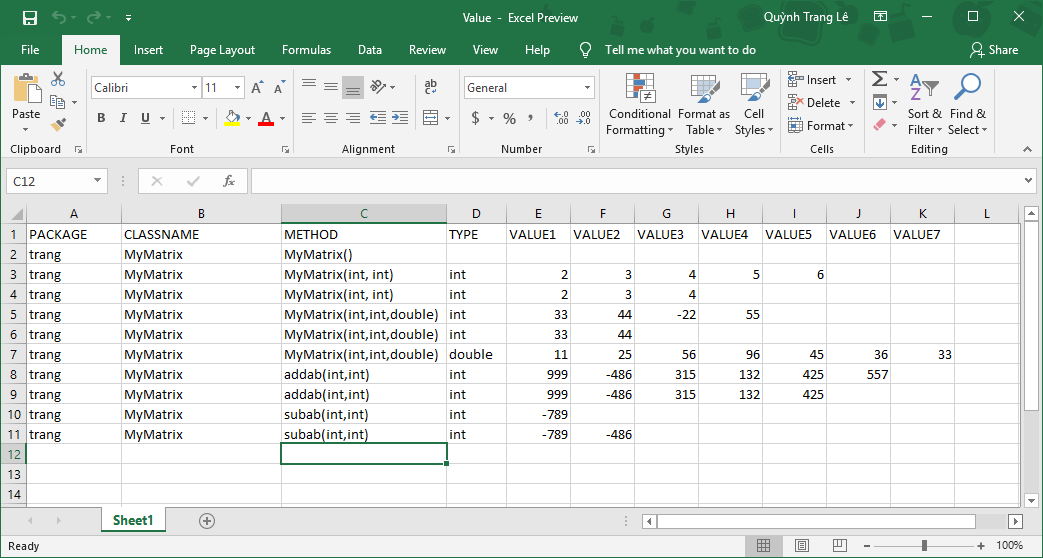
* Phần thứ nhất gồm 4 cột. Đây là phần chứa các thông tin chương trình đọc được từ file java và ghi lại, bao gồm các cột: tên package, tên class, tên medthod và tên kiểu giá trị đầu vào của method tương ứng.

Quan sát trên định dạng có thể thấy, đối với các trường package, trường class, thông tin được ghi lại chính là tên của package, class đó. Tuy nhiên, đối với trường method, thông tin lưu lại có dạng:

*<tên\_method>(<kiểu\_đầu\_vào\_1>, <kiểu\_đầu\_vào\_2>,…<kiểu\_đầu\_vào\_n>)*

Quan sát lại định dạng của file methodlist theo chuẩn của Randoop *(hình)*, các phương thức cũng được liệt kê theo định dạng tương tự như trên. Như vậy, việc lưu lại định dạng như trên sẽ giúp chương trình dễ dàng ghi file methodlist mà không cần xử lý ký tự nhiều lần.

* Phần thứ 2 bao gồm 7 cột. Các cột này đều còn trống và cần người dùng nhập vào khi muốn sinh file value. Số lượng đầu vào được giới hạn bởi trong thực tế, các giá trị đầu vào cần test thường là các giá trị đặc biệt. Số lượng các trường hợp đặc biệt không quá nhiều. Vì vậy, giới hạn số lượng các giá trị đầu vào giúp chương trình giảm thiểu được thời gian hoạt động, tiết kiệm tài nguyên lưu trữ mà vẫn đảm được nhu cầu công việc.



*Hình. Định dạng mẫu file Excel*

# Logic hoạt động

# Đọc file java

Các file java được đọc theo luồng ký tự. Dựa vào cấu trúc của một file java thông thường, có thể xác định được thứ tự các thông tin thu được lần lượt sẽ là: tên package, tên class, các method và các kiểu giá trị đầu vào của method. Bên cạnh đó, các thông tin này đều được khai báo thông qua các từ khóa, ký tự đặc biệt như sau:

* **package** <tên package>
* **class** <tên class>
* **public** <các từ khóa khác> <tên method>**(**<kiểu biến đâu vào> <tên biến đầu vào**,**…**){…}**

Dựa vào các đặc trưng này, các thông tin lần lượt được xác định và bóc tách khỏi các ký tự không cần thiết để tạo ra các đối tượng Package ghi vào file Excel.

# Đọc file Excel

Đối với định dạng file Excel đã thiết kế, việc sinh file classlist hay methodlist trở nên dễ dàng hơn, đặc biệt là file methodlist. Định dạng file classlist đơn giản như sau:

<*Tên\_package\_1>.<Tên\_class\_1>*

*<Tên\_package\_2>.<Tên\_class\_2>*

*……*

*<Tên\_package\_n>.<Tên\_class\_n>*

File methodlist cũng tương tự. Có thể thấy, để đọc thông tin từ file Excel nhằm sinh file classlist và methodlist, chương trình chỉ cần duyệt các trường package, class, method trong file Excel từ trái qua phải, từ trên xuống dưới.

Đối với file value có phần phức tạp hơn. Mỗi đối tượng Package trong chương trình lưu một biến đầu vào của phương thức. Như vậy, số lượng biến đầu vào của phương thức bằng số lượng đối tượng Package mà phương thức tạo ra.

*Ví dụ: phương thức add(int a, int b){…} có hai biến đầu vào kiểu int là a và b. Như vậy phương thức add cần hai đối tượng Package để lưu giá trị của mỗi biến đầu vào.*

Như vậy, để sinh ra một file value nhập giá trị đầu vào cho một phương thức, cần xét tất cả các đối tượng Package có tên class, tên package, và tên method giống nhau. Khi xét tới một class, một package hay một method mới, chương trình sẽ tạo ra file value tiếp theo để lưu giá trị đầu vào cho method mới này.

# Các ứng dụng, công cụ, thư viện hỗ trợ giải quyết vấn đề

# Maven

# Apache Maven là gì?

Apache Maven là một chương trình hỗ trợ quản lý project bao gồm các thao tác như khởi tạo, biên dịch, đóng gói, chạy test,… cho project, đồng thời hỗ trợ quản lý, thêm, thay đổi version hay xóa các thư viện sử dụng trong project đó.

Những nhà phát triển Apache Maven có đưa ra định nghĩa về chương trình này trong phần giới thiệu trên trang chủ như sau: *“Apache Maven is a software project management and comprehension tool. Based on the concept of a project object model (POM), Maven can manage a project's build, reporting and documentation from a central piece of information.”*

# Tại sao nên sử dụng Maven?

# Trong lập trình nói chung

Maven tự động hóa toàn bộ quá trình release project như khởi tạo, [cập nhật](http://congdongthongtin.com/5658/javara-phan-mem-cap-nhat-va-go-bo-cac-phien-ban-java-cu-du-thua) thư viện, build và test, release và tạo [tài liệu](http://congdongthongtin.com/4947/top-5-website-java-ve-cac-vi-du-ung-dung-ma-nguon-tai-lieu-hoc-java) mô tả. Giá trị của các tính năng Maven đem lại càng được khẳng định rõ khi chúng được ứng dụng trong các dự án có quy mô lớn với hàng chục các module làm các công việc khác nhau. Mỗi module được phân công tới các nhóm phát triển khác nhau: các quá trình phân tích yêu cầu, đặc tả, phát triển, kiểm thử và release được diễn ra đồng thời nhưng hoàn toàn độc lập. Trong khi đó,các module này lại phụ thuộc vào nhau, việc phát triển module này lại dựa trên module khác.

* + Tự động hóa toàn bộ qui trình release là yêu cầu hàng đầu: Chỉ cần 1 sự chậm trễ, sai lầm ở một công đoạn có thể làm đổ vỡ cả quá trình release của project. Cho nên việc tự động hóa toàn bộ là ưu tiên hàng đầu, nhờ đó là việc lên kế hoạch hiệu quả và chính xác hơn, việc tích hợp và triển khai cũng đơn giản hơn rất nhiều.
  + Tự động hóa quá trình test với các plugin: Toàn bộ quá trình test được tự động, từ đó các lỗi được phát hiện sớm hơn. Không những thế, lỗi có thể được phát hiện ở giai đoạn phát triển, bởi các lập trình viên đều có thể chạy các test case một cách dễ dàng bằng Maven.

# Trong đồ án này

# JUnit

# Định nghĩa JUnit

*Junit là một framework unit testing của lập trình hướng đối tượng Java. JUnit rất quan trọng trong sự phát triển của test - driven của lập trình viên , và nó là 1 trong những thành phần của unit testing frameworks được gọi chung là xUnit có nguồn gốc với JUnit.*

# Vai trò của JUnit đối với Randoop

Đối với Randoop, JUnit đóng một vai trò quan trọng, là cơ sở để tạo ra các unit test. Đối với các project sử dụng cách viết unit test thông thường, các project được xây dựng bằng ngôn ngữ Java hầu hết đều sử dụng JUnit. Randoop là công cụ sinh unit test tự động trong ngôn ngữ Java. Vì vậy, để có thể tự động sinh ra các unit test cho ngôn ngữ Java, Randoop sử dụng JUnit là điều tất yếu.

# Apache POI

# Apache POI là gì?

Apache POI là một công cụ mã nguồn mở được phát triển bởi Apache cho phép Java có thể đọc ghi các file Excel trên nhiều nền tảng khác nhau. Apache POI hỗ trợ đọc ghi cho cả hai định dạng xls và xlsx.

# Tại sao sử dụng POI?

# Trong lập trình nói chung

Trong mỗi môi trường phát triển phần mềm, dữ liệu là một phần không thể thiếu. Khi phát triển, các dữ liệu thường được lưu trữ trực tiếp trong các file chương trình, mỗi khi có sự thay đổi nào đó liên quan trực tiếp đến dữ liệu trong các file đó thì bắt buộc phải thực hiện biên dịch code và kiểm tra lại các kết quả biên dịch đó.

Vấn đề đặt ra ở đây là, nếu như dữ liệu cần phải thay đổi nhiều lần, liên tục và hàng ngày thì việc thực hiện thay đổi chương trình theo mỗi lần thay đổi đó và biên dịch lại là không thể tránh khỏi. Quá trình biên dịch và sửa lỗi tiêu tốn nhiều thời gian và công sức. Vì vậy, để phát triển phần mềm mà không gặp những trở ngại trên, cách tốt nhất là nên tách rời với phần dữ liệu để khi hoạt động chúng không ảnh hưởng đến nhau và đến các file chương trình. Cô lập dữ liệu với chương trình là cách duy nhất để thực hiện. Dữ liệu có thể lưu trong các file excel, property file, config file, Json file, hay các XML file…

Nếu sử dụng file Excel để lưu trữ dữ liệu, những nhà phát triển chắc chắn phải thao tác với các dữ liệu để sử dụng trong chương trình của mình. Apache POI đóng vai trò làm công cụ hỗ trợ trung gian đó

# Vai trò của Apache POI đối với UET TestGen

Để có thể đọc ghi các package, các giá trị đầu vào từ file Excel, UET TestGen cần có một công cụ hỗ trợ trao đổi dữ liệu giữa project với chúng. Apache POI là một công cụ hoàn toàn thích hợp, hỗ trợ toàn bộ các phương thức tương tác với bảng tính Excel từ các trang tính tới hang, cột, ô hay các hàm tính toán, các kiểu dữ liệu được lưu trữ trong mỗi ô. POI thực sự đã đóng góp một phần quan trọng trong việc phát triển thành công công cụ UET TestGen.

# Tổng kết chương

Như vậy, chương 3 đã chỉ rõ ra được bài toán đồng thời giải thích hướng giải quyết bài toán. Đó làm sao để tận dụng được những ưu điểm của Randoop, đồng thời khắc phục được những nhược điểm còn tồn tại ở công cụ này. Việc phát triển công cụ UET TestGen thực sự đã giải quyết được những nhược điểm đó. Quá trình thao tác với Randoop trở nên dễ dàng hơn, tiết kiệm rất nhiều thời gian, công sức cho những người làm kiểm thử phần mềm. Việc lựa chọn hướng giải quyết, các công cụ hỗ trợ phù hợp đã giúp cho việc phát triển UET TestGen một cách thuận tiện và nhanh chóng.