1. **Unit Testing (Hoàng Hải Tú)**

Unit testing là phương pháp kiểm thử phần mềm mà mỗi đơn vị mã nguồn, tập hợp 1 hoặc nhiều các module chương trình cùng với dữ liệu kiểm soát liên quan, quy trình vận hành, thủ tục lưu trữ được kiểm thử để quyết định xem chúng đã đạt mục đích sử dụng hay chưa.

Một đơn vị có thể được coi như là một phần tử nhỏ nhất mà có thể thực hiện test của chương trình. Trong lập trình thủ tục, một đơn vị có thể là một module, nhưng thường là một hàm (function) hay một quy trình (procedure). Trong OOP, một đơn vị thường là cả một interface, ví dụ một class, nhưng cũng có thể chỉ là một phương thức. Vậy nên có thể coi Unit Test như hoạt động kiểm thử **nhỏ nhất**, chi tiết và xuyên suốt nhất với mục đích đảm bảo chất lượng từng phần riêng biệt của chương trình.

Những Unit Test là các đoạn code nhỏ, được viết **bởi lập trình viên** (not Testers!) hoặc cũng có thể bởi White – box – tester trong quá trình phát triển phần mềm.

Thực hiện kiểm tra đơn vị được thiết kế đơn giản, thường sẽ xác định xem hàm có trả về giá trị như mong muốn hay không. Các Unit Test thường được kiểm tra một cách độc lập, chúng cô lập từng đơn vị để kiểm tra trên chính đơn vị đó mà không dính dáng đến những đơn vị khác. Điều này khác biệt với các quá trình Kiểm thử cài đặt (Integration Tests). Rất có thể trong quá trình kiểm thử, một class sẽ có tham chiếu với những class khác, dẫn đến việc test class này thì cũng test cả class mà nó có tham chiếu đến. Điều này là hoàn toàn **SAI LẦM**, vì nếu xảy ra lỗi thì sẽ không thể biết được lỗi nằm ở unit nào. Nếu vượt qua giới hạn của đơn vị như vậy thì Unit Test sẽ trở thành Integration Test. Để khắc phục thì dev tạo một abstract interface để implement Interface đó với object mà họ muốn kiểm thử. Điều này đảm bảo tính cô lập của quá trình test.

Unit Test thường được thực hiện một cách tự động (automated), nhưng cũng có thể được làm thủ công (manually). Mục tiêu của Unit Test là cô lập một unit và kiểm thử nó. Cách làm test thủ công có thể dùng các tài liệu hướng dẫn theo từng bước, và test tự động cũng có thể làm điều này. But, nếu không cẩn thận thì các hướng dẫn Unit Test Case có thể thực hiện như Integration Test Case, từ đó phá hỏng mục đích của Unit Test.

Để đạt sự cô lập hiệu quả khi thực hiện test theo hướng tự động, các đoạn code của unit được thực hiện trong 1 framework hoạt động ngoài môi trường chương trình chứa nó. Test như vậy có thể vạch ra một số phụ thuộc không cần thiết giữa unit được test và phần còn lại của chương trình, cho phép ta loại bỏ đống rác rưởi này.

Một vài ngôn ngữ hỗ trợ built-in unit testing như: MATLAB, Python, Ruby, Rust …

Những ngôn ngữ khác không hỗ trợ built-in nhưng lại có các thư viện hay framework rất tốt cho unit testing như: C# (Nunit), Java (Junit), JS, PHP (PHPUnit), VB .NET, …

1. **Stability Testing (Ng Hữu Linh)**

Stability testing là gì?

Stability testing là khả năng duy trì hoạt động của sản phẩm xuyên suốt và vượt quá thời hạn sử dụng của nó, mà không hỏng hoặc xảy ra lỗi. Đây là 1 kỹ thuật non-functional, với mục đích đòi hỏi khả năng chịu tải của phần mềm tới mức tối đa. Trong quá trình xác định nó hoạt động tốt thế nào dưới tải ở mức chấp nhận được, mức đỉnh, các tải được tạo ra đột ngột, với số lượng dữ liệu lớn được xử lý… Stability testing được thực hiện để kiểm tra hiệu quả của 1 sản phẩm được phát triển vượt qua mức hoạt động bình thường, hay tới 1 điểm dừng. Có ý nghĩa quan trọng hơn là trong việc xử lý lỗi, độ tin cậy của phần mềm, khả năng chịu tải và khả năng mở rộng của 1 sản phẩm dưới tải lớn chứ không phải là kiểm tra cách hoạt động của hệ thống trong các hoàn cảnh bình thường.

Stability testing còn được gọi là Load hoặc endurance testing.

Chuyện gì sẽ xảy ra nếu 1 hệ thống thử nghiệm chưa trải qua Stability testing?

Đối với 1 ứng dụng đang được thử nghiệm nơi mà 1 số lượng lớn user được giới thiệu và các ứng dụng phải chạy hàng tháng mà không cần khởi động lại, 1 số vấn đề có thể xảy ra:

Các lỗi có thể phải đối mặt như: • Hệ thống chạy chậm lại • Hệ thống gặp các vấn đề về function • Hệ thống hiển thị wired behavior • Hệ thống bị treo

Do đó, Stability testing thường liên quan đến việc thực thi của hệ thống với các heavy user (ảo) và đo các tham số hiệu năng dể xác minh liệu hệ thống có thể hỗ trợ tải dự tính hay không.

Tại sao phải thực hiện Stability testing?

Loại testing này giúp user hiểu được cách thức hệ thống sẽ làm việc trong các tình huống thực tế.

Do đó, Stability testing cho phép bạn kiểm tra:

• Ước lượng độ tin cậy trong sự ổn định của hệ thống của bạn đang thử nghiệm

• Đảm bảo rằng hệ thống của bạn có thể xử lý các chương trình lớn

• Theo dõi hiệu quả của hệ thống của bạn

• Kiểm tra hệ thống ổn định theo yêu cầu

Nó đóng 1 vai trò quan trọng trong phát triển sản phẩm vì nó được sử dụng để xác định những hạn chế của 1 sản phẩm phần mềm đang thử nghiệm trước khi nó được released hoặc những phạm vi cải tiến hơn trước khi sản phẩm goes live hay ở Production. 1 ví dụ rất phổ biến của kỹ thuật Stability testing là

Các cổng mua sắm trực tuyến: Stability testing sẽ kiểm tra cách các website sẽ hoạt động khi:

• Lượng dữ liệu nhập vào lớn ở giờ cao điểm

• Số lượt truy cập ở 1 thời điểm nhất định

• Vấn đề page load ở cùng 1 thời điểm

• Cách vận hành của hệ thống

• Phản ứng của hệ thống và nhiều hơn nữa buộc phải trải nghiệm dưới Stability testing

Các test case cho Stability testing đối với CPU performance

• Để xác minh giới hạn trên của hệ thống

• Cách hệ thống crash hay recover

• Tổng số transaction hoàn thành mỗi yêu cầu

• Các giao dịch phản hồi có ổn định hay không hay tăng theo thời gian

• Làm thế nào hệ thống hoạt động dưới tải nặng

• Phản ứng và cách vận hành của nó dưới sức tải nặng

Các test report cho Stability testing

1 số số liệu thống kê được thu thập và đánh giá trong quá trình thực hiện test; những con số này được phân tích để tạo ra 1 report và để xác định các vấn đề về hiệu năng.

Các ví dụ về số liệu thống kê được thu thập dưới việc test là:

• Transaction response time: Thời gian trung bình thực hiện các giao dịch trong quá trình test. Thống kê này sẽ đánh giá liệu hiệu năng của server nằm trong khoảng thời gian thực hiện các giao dịch tối thiểu và tối đa có thể chấp nhận được định nghĩa cho hệ thống hay không. Thông tin này sẽ đánh giá thời gian xử lý yêu cầu của web server và gửi tới application server, cái mà trong hầu hết các trường hợp sẽ tạo 1 yêu cầu tới database server.

• Hits per second: Số lượt truy cập thực hiện trên server bởi các user. Thống kê này có lợi cho việc xác định lượng tải các user tạo ra, liên quan đến 1 số lần truy cập.

• Throughput: Thông lượng trên web server trong quá trình test được đo bằng byte. Thông lượng là lượng dữ liệu mà các user nhận được từ server tại bất kỳ thời điểm nào. Thống kê này giúp đánh giá lượng tải mà các user tạo ra.

• Transaction per second: Đây là tổng số các giao dịch hoàn thành (cả thành công và thất bại) thực hiện trong quá trình test. Thống kê này giúp kiểm tra tải thực sự của giao dịch trên hệ thống.

• CPU: % CPU sử dụng dành ra trong quá trình test.

• Memory: Bộ nhớ sử dụng trong quá trình test

• Disk: Không gian sử dụng của ổ đĩa dành ra trong quá trình test

Lúc này, bạn có thể dễ dàng nhận ra rằng:

Stability testing đi kèm theo performance testing – một kỹ thuật mà được thực hiện để kiểm tra 1 số thuộc tính chất lượng của phần mềm như tính ổn định, độ tin cậy và tính khả dụng.

Việc test này được sử dụng để xác định mức độ nhanh chóng của 1 hệ thống hoặc 1 hệ thống phụ thực thi dưới 1 khối lượng công việc cụ thể.

Performance testing có nhiều loại và Stability testing là 1 trong số chúng.

• Stress testing: nó là 1 loại test mà kiểm tra sức chịu tải của hệ thống vượt qua khả năng của hệ thống.

• Spike testing: nó được sử dụng để kiểm tra hành vi của 1 hệ thống bằng cách tăng tải của 1 hệ thống ngay lập tức. Mục đích là để kiểm tra tại thời điểm nào hệ thống sẽ có vấn đề về hiệu năng, hay nó sẽ pass.

• Scalability testing: Nó được sử dụng để kiểm tra khả năng của 1 hệ thống. Hệ thống sẽ có hiệu lực hành xử thế nào trong như cầu ngày càng tăng, thay đổi về quy mô và khối lượng.

• Volume testing: Nó là 1 kỹ thuật test non-functional nơi mà phần mềm đang thử nghiệm phải chịu khối lượng công việc khổng lồ, và hành vi của 1 hệ thống được kiểm tra và xác minh phù hợp

• Load or stability testing: đã được mô tả bên trên.

1 vài công cụ cho performance testing như:

• WebLOAD

• LoadRunner

• Apache JMeter

• NeoLoad

• CloudTest

• Loadstorm

• LoadUI

• WAPT

• LoadImpact

• Loadster

• Httperf

• OpenSTA

• Performance Tester

• Testing Anywhere

Kết luận:

Stability testing là 1 quá trình non-functional của việc test hành vi của các ứng dụng bằng cách áp dụng tải tối đa. Nó được thực hiện để tìm ra khả năng mở rộng của hệ thống trong 1 môi trường nhất định.

1. **Acceptance testing - Kiểm thử chấp nhận (Ng. Anh Tuấn Tùng)**

Trong kiểm thử phần mềm, ISTQB (the International Software Testing Qualifications Board) định nghĩa chấp nhận là: kiểm tra quy chuẩn đối với nhu cầu cần thiết của người sử dụng, và các quy trình kinh doanh đã được tiến hành để xác định một hệ thống thỏa mãn các tiêu chí chấp nhận và cho phép người sử dụng, khách hàng và các thực thể được ủy quyền khác xác định có chấp nhận hệ thống hay không. Kiểm thử chấp nhận còn được gọi là kiểm thử chấp nhận phía người dùng (User Acceptance Testing - UAT), kiểm thử người dùng cuối (end-user testing), kiểm thử chấp nhận hoạt động (Operational Acceptance Testing - OAT),...

I: Tổng quan

Kiểm thử là một tập hợp các hoạt động được tiến hành để tạo thuận lợi cho việc khám phá và / hoặc đánh giá các đặc tính của một hoặc nhiều mục được thử nghiệm. Mỗi bài kiểm thử riêng biệt, được gọi là một test case, thực hiện một tập hợp các hoạt động kiểm tra được xác định trước, được phát triển để thúc đẩy việc thực hiện mục kiểm tra để đáp ứng các mục tiêu thử nghiệm; bao gồm việc cài đặt chính xác, xác định lỗi, xác minh chất lượng và các chi tiết có giá trị khác. Môi trường kiểm thử thường được thiết kế giống, hoặc gần nhất có thể, với môi trường dự kiến của sản phẩm. Nó bao gồm tất cả các cơ sở, phần cứng, phần mềm, chương trình cơ sở, quy trình và / hoặc tài liệu dành cho hoặc được sử dụng để thực hiện kiểm thử phần mềm.

Các test case của UAT và OAT được tạo từ sự hợp tác với các khách hàng doanh nghiệp, các nhà phân tích kinh doanh, người thử nghiệm và các nhà phát triển. Điều quan trọng là các bài kiểm thử này bao gồm cả các bài kiểm thử logic nghiệp vụ cũng như các điều kiện môi trường hoạt động. Khách hàng doanh nghiệp (chủ sở hữu sản phẩm) là những bên liên quan chính của các thử nghiệm này. Khi các điều kiện thử nghiệm được chấp nhận, các bên liên quan được đảm bảo rằng sự phát triển đang đi đúng hướng.

II: Quá trình

Bộ kiểm thử chấp nhận có thể cần phải được thực hiện nhiều lần vì tất cả các trường hợp thử nghiệm có thể không được thực hiện trong một lần thử nghiệm đơn lẻ.

Bộ thử nghiệm chấp nhận được chạy bằng cách sử dụng các thủ tục kiểm tra chấp nhận được xác định trước để hướng người thử nghiệm sử dụng dữ liệu nào, các quy trình từng bước theo và kết quả mong đợi sau khi thực hiện. Kết quả thực tế được giữ lại để so sánh với kết quả mong đợi. Nếu kết quả thực tế phù hợp với kết quả mong đợi cho từng trường hợp thử nghiệm, trường hợp kiểm thử được cho là đạt. Nếu số lượng các trường hợp kiểm tra không vượt qua không vi phạm ngưỡng được xác định trước của dự án, thì bộ thử nghiệm được cho là đã vượt qua. Nếu có, hệ thống có thể bị từ chối hoặc chấp nhận theo các điều kiện đã được thỏa thuận giữa nhà tài trợ và nhà sản xuất.

Kết quả dự đoán của việc thực hiện thử nghiệm thành công:

1. Các test case được thực thi, sử dụng dữ liệu được xác định trước.

2. Kết quả thực tế được ghi lại.

3. Kết quả thực tế và mong đợi được so sánh.

4. Kết quả kiểm tra được xác định.

Mục tiêu của việc kiểm thử là để cung cấp sự tự tin rằng sản phẩm phát triển đáp ứng cả các yêu cầu chức năng và phi chức năng. Mục đích của việc tiến hành kiểm thử chấp nhận là khi kiểm thử hoàn thành và cung cấp các tiêu chí chấp nhận được đáp ứng, các nhà tài trợ sẽ ký vào việc phát triển / nâng cao sản phẩm như đáp ứng các yêu cầu đã được xác định (trước đây đã thỏa thuận giữa doanh nghiệp và nhà cung cấp sản phẩm / nhà phát triển).

1. **Automation Test: Kiểm thử tự động (Lương Trọng Hải)**

\* Khái niệm:

Là việc sử dụng phần mềm đặc biệt (tách biệt với phần mềm đang được kiểm thử) để kiểm soát việc thực hiện các bài kiểm tra và so kết quả thực tế với kết quả dự đoán. Tự động kiểm thử có thể tự động hóa một số nhiệm vụ lặp đi lặp lại nhưng cần thiết trong một quá trình thử nghiệm đã được chính thức hóa, hay là các kiểm thử bổ sung nhưng sẽ khó thực hiện thủ công. Kiểm thử tự động là rất quan trọng cho phân phối liên tục và kiểm thử liên tục.

- Một số phần mềm kiểm thử tự động phổ biến hiện nay:

+ Quick Test Profressional - (HP)

+ Selenium

+ Test Architect - (LogiGear)

+ Ranorex

+ Visual Studio CodedUI Testing

+ TestComplete (SmartBear)

+ SOAPUI - Web Services Testing (SmartBear)

\* Tầm quan trọng của kiểm thử tự động (automation test) trong thời kỳ công nghiệp 4.0:

+ Cách mạng công nghiệp lần thứ tư đánh dấu kỷ nguyên vạn vật kết nối Internet. Nó xảy ra dựa trên sự hội tụ của nhiều công nghệ trong đó có công nghệ cốt lõi có công nghệ thông tin với sự phát triển không ngừng của công nghệ Internet từ thời kỳ kết nối nội dung như email đến mạng xã hội, Internet vạn vật, Internet kết nối thiết bị máy móc kết nối quá trình vận hành của các nhà máy. Ngoài công nghệ cốt lõi còn có sự hội tụ của công nghệ in 3D, công nghệ vật liệu tiên tiến, công nghệ lưu trữ…

+ Hiện nay, nền công nghiệp 4.0 đang phát triển mạnh mẽ, kéo theo sự phát triển của rất nhiều ngành nghề, đặc biệt là ngành công nghệ thông tin. Do đó, yêu cầu về nhân lực trong mảng này cũng đòi hỏi cac ứng viên cần phải có kỹ năng tốt hơn nữa về lập trình, technical, các kỹ năng về automation....

+ Nếu search trên các trang mạng tuyển dụng lớn như ITviec, Vietnamworks, LinkIn, Indeed, Dice, Monster, CareerBuilder … thì bạn sẽ thấy có vô vàn kết quả với các từ khóa tìm kiếm như:

Test Automation Engineer

Automation Developer

Automation Testing/Automation Tester

QA Automation Engineer

Software Development Engineer in Test (SDET)

+ Từ đây, bạn có thể thấy được nhu cầu tuyển một QA có kiến thức và kinh nghiệm về automation test là rất lớn. Do đó, đủ để hiểu automation test sẽ là tiềm năng lớn cho tương lai của mảng Test nói riêng và công nghệ thông tin nói chung.

\* Dự án nào thì nên áp dụng automation test?

Chúng ta đều biết rằng các dự án Automation khác với các dự án test Manual. Mặc dù, các dự án tự động hóa không thực sự tồn tại (hoặc không tồn tại như lý tưởng), cả hai dự án automation và manual được xử lý khác nhau ngay trong bước lên plan. Một dự án mà plan không rõ ràng thì không thể thực hiện đúng; điều này không chỉ ảnh hưởng đến các dự án hiện tại và ảnh hưởng không tốt đến việc đánh giá khả năng cá nhân, mà nó còn có thể dẫn đến việc mất niềm tin của khách hàng, quản lý, làm ảnh hưởng đến kinh doanh.

- Đối với một dự án manual testing, các yếu tố phát sinh chi phí là:

• Con người

• Tools – Test/quản lý defect

• Infrastructure – môi trường, cơ sở hạ tầng

• Thời gian

• Đào tạo

- Đối với một dự án automation, ngoài các mục nêu trên, nó cần chi phí cho:

• Automation tools

• Add–in để quản lý test tool

• Add-in để hỗ trợ ứng dụng đang được test (Application Under Test- AUT: ví dụ như SAP, oracle etc)

• Thiết lập Framework

- Trong hoàn cảnh đó, thành công của một dự án automation phụ thuộc vào cách viết code, bao nhiêu phần code được viết có thể tái sử dụng hoặc với bao nhiêu dòng code thì có thể đạt được kết quả mong muốn? Nếu câu trả lời cho câu hỏi này là “KHÔNG” thì bạn đã lên kế hoạch cho các dự án automation không chính xác.

- Thông thường, một test plan có các phần sau:

- Các phần trong Test plan của Automation test:

# 1: Phạm vi (Scope):

• Chọn những Test case / kịch bản có thể sử dụng lại nhiều lần khi test quay vòng

• Đôi khi test case đơn giản nhất cũng cần rất nhiều giải pháp phức tạp để test tự động. Nếu test case này chỉ dùng 1 lần duy nhất thì rõ ràng là không có ý nghĩa. Cũng vì nó có thể dùng lại nhiều lần nên cần tập trung và bỏ công sức vào đó.

• Automation test không thể thực hiện exploratory testing

# 2: Chiến lược kiểm thử (Test strategy):

• Phần này được coi như là framework trong automation test. Một số framework là rất khó để tạo ra và cũng có hiệu quả nhưng đòi hỏi thời gian, nỗ lực và năng lực. Cố gắng tìm một giải pháp tốt nhất để có thể thực hiện mà không gây nguy hại qua việc sử dụng các nguồn tài nguyên.

• Quyết định cách viết code tốt nhất cho automation test, quy ước đặt tên, địa điểm lưu trữ thiết bị test, format của test result,… để duy trì tính đồng nhất và tăng năng suất.

# 3: Tài nguyên / vai trò và trách nhiệm (Resources/roles and responsibilities):

• Bước đầu tiên là cần hiểu khả năng của nhóm và dự đoán trước phạm vi của automation trong tổng thể. Điều này sẽ giúp chọn một team phù hợp với nhu cầu của cả automation và manual testing. Ngoài ra, chọn những người có thái độ đúng – những người không nghĩ rằng manual testing là dưới tầm của họ.

• Chọn một đội thành thạo với AUT, quản lý test, quản lý defect và và các hoạt động SDLC khác

#4: Công cụ (Tools):

Chọn automation tool dựa trên các quy tắc sau đây:

• Nếu tool test không phải miễn phí thì công ty đã có giấy phép cho tool đó chưa

• Có thể tìm các tool open source nhưng cần lựa chọn

• Các thành viên trong nhóm biết về tool này hoặc đã dùng chưa, nếu không thì có cần đưa thêm người mới vào team không hay cần training cho người cũ

# 5: Lịch trình (Schedules):

• Bao gồm thời gian cho code và kiểm tra các kịch bản tự động hóa (automation scripts)

• Duy trì các kịch bản một cách thường xuyên, kịp thời. Nếu bạn tạo một đoạn code mà bạn sẽ không sử dụng trong 6 tháng tới hoặc lâu hơn, thì cần bảo trì để giảm khả năng bị failure

# 6: Môi trường (Environment):

• Các môi trường mà AUT sẽ chạy và automation tool mà bạn muốn sử dụng phải tương thích. Đây là một trong những yếu tố được cân nhắc khi chọn tool.

# 7: Phân phối (Deliverables):

• Bạn tạo test scripts, tuy nhiên, không phải ai cũng hiểu về tự động hóa / ngôn ngữ lập trình. Vì vậy cần có kế hoạch tạo ra một tài liệu “Hướng dẫn” để giúp người sử dụng hiện tại và tương lai để có thể hiểu được kịch bản này ngay cả khi không có bạn.

• Bao gồm các comment trong kịch bản.

# 8: Các rủi ro (Risks):

• Nếu bạn sẽ đề xuất một giải pháp tự động hóa, hãy chắc chắn để chọn được tool hiệu quả và chi phí phù hợp để đảm bảo rằng việc thực hiện automation không trở thành gánh nặng của dự án. Điều quan trọng cần phải xác định rằng lợi ích thu được khi đầu tư automation không nhìn thấy ngay lập tức nhưng có thể được nhìn thấy trong thời gian dài. Vì vậy, nếu bạn đề xuất việc tự động hóa hệ thống, chọn tool có các tính chất như

o Duy trì ổn định và không cần bảo trì quá nhiều

o Có phạm vi cho dãy hồi quy lớn

o Không cần phải can thiệp quá nhiều bằng tay, không phụ thuộc vào trực giác của con người

# 9: Dữ liệu test (Test data):

• Đi vào xem xét các khía cạnh bảo mật của dữ liệu

• Không đưa trực tiếp data test vào script. Điều này có thể gây ra lỗi trong quá trình sửa đổi, bảo trì.

• Dữ liệu test cần rất cụ thể. Đối với bước test thủ công- ‘nhập tên’, theo yêu cầu có thể nhập tên bất kỳ gồm 5 ký tự. Trong khi test, có thể gõ “Swati” hoặc “Seela” hay bất cứ tên nào khác. Nhưng đối với automation test không thể làm giả định như vậy, cần phải nhập vào giá trị chính xác

# 10: Báo cáo / kết quả (Reports/results):

• Kết quả thực hiện Script cũng có tính kỹ thuật và có thể không phải mọi người trong team đều hiểu được. Plan nên viết chi tiết kết quả thực hiện trong file document hoặc file excel như một biện pháp bổ sung

• Framework documents, review results, defect reports, status.. cần có trong report

Đôi khi không dễ dàng để đề xuất khách hàng / quản lý mua các automation tool. Tuy nhiên, khi mục tiêu cuối cùng của automation test là để tối đa hóa lợi nhuận thì hoàn toàn phù hợp với mục tiêu của quản lý / khách hàng. Điều này sẽ đảm bảo rằng không chỉ chúng ta có thể thực hiện tự động hóa trong dự án mà nó còn nhận được sự đồng thuận, hợp tác và hứng thú của nhiều bên liên quan.

Lên kế hoạch và phân tích kỹ lưỡng về tất cả các yếu tố được liệt kê ở trên có thể giúp chúng ta thấy được nhiều lợi ích của automation test và các yếu tố cần thiết khi tìm kiếm một tool test.