

BỘ CÔNG THƯƠNG
TRƯỜNG ĐẠI HỌC SAO ĐỎ



ĐỒ ÁN TỐT NGHIỆP

Ngành: Công nghệ thông tin

**TÊN ĐỀ TÀI: NGHIÊN CỨU KỸ THUẬT KIỂM THỦ PHẦN MỀM
VÀ XÂY DỰNG MỘT ỨNG DỤNG KIỂM THỦ TỰ ĐỘNG**

Sinh viên thực hiện : Nguyễn Thị Thanh Hậu

Lớp, khóa : DK10-CNTT, khóa 10

Giảng viên hướng dẫn : ThS Vũ Bảo Tạo

HẢI DƯƠNG – NĂM 2023

LỜI CAM ĐOAN

Tôi xin cam đoan các kết quả đưa ra trong Đồ án tốt nghiệp này là các kết quả thu được trong quá trình nghiên cứu, thực nghiệm của tôi dưới sự hướng dẫn của thầy Vũ Bảo Tạo, không sao chép bất kỳ kết quả nghiên cứu nào của các tác giả khác.

Nội dung nghiên cứu có tham khảo và sử dụng một số thông tin, tài liệu từ các nguồn tài liệu đã được liệt kê trong danh mục các tài liệu tham khảo.

Nếu sai tôi xin chịu mọi hình thức kỷ luật theo quy định.

Hải Dương, ngày 15 tháng 12 năm 2023

Sinh viên thực hiện

Nguyễn Thị Thanh Hậu

Đại học Sao Đỏ

LỜI CẢM ƠN

Để hoàn thành đồ án, trước hết cho em xin gửi lời cảm ơn chân thành và sâu sắc đến thầy giáo Vũ Bảo Tạo, người đã tận tình hướng dẫn em trong suốt quá trình nghiên cứu vừa qua. Trong thời gian được thầy hướng dẫn, em không những tiếp thu thêm nhiều kiến thức bổ ích mà còn học tập được tinh thần, thái độ làm việc nghiêm túc, hiệu quả. Đây là điều rất cần thiết cho em trong quá trình học tập và làm việc sau này.

Em chân thành cảm ơn các thầy cô trong khoa Công nghệ Thông tin đã tâm huyết dạy dỗ, truyền đạt những kiến thức quý báu cho chúng em trong suốt 4 năm học đại học. Những kiến thức đó không chỉ giúp em hoàn thành đồ án tốt nghiệp này mà còn là thứ hành trang quý báu để em có thể tự tin trên quá trình theo đuổi hành công của mình.

Em cũng xin cảm ơn gia đình, bạn bè, người thân và đặc biệt là tập thể lớp DK10 - CNTT đã hỗ trợ em hết mình trong những năm tháng sinh viên.

Xin trân trọng cảm ơn!

Hải Dương, ngày 15 tháng 12 năm 2023

Sinh viên thực hiện

Nguyễn Thị Thanh Hậu

DANH MỤC HÌNH ẢNH

Hình 1.1. Các giai đoạn kiểm thử phần mềm	3
Hình 1.2. Quy trình của kiểm thử tự động	6
Hình 1.3. Công cụ Selenium.....	11
Hình 1.4. Công cụ Appium.....	12
Hình 1.5. Công cụ Cypress.....	12
Hình 1.6. Công cụ Katalon Studio.....	13
Hình 1.7. Công cụ Testing Whiz	14
Hình 1.8. Công cụ TestComplete	14
Hình 1.9. Công cụ RedLine13	15
Hình 1.10. Công cụ TestCraft.....	15
Hình 2.5. Mô hình các khái niệm trong CodeDom	22
Hình 3.1. Kết quả chạy test script lưu trên file MS excel	30
Hình 3.2. Use case tổng quát.	31
Hình 3.3. Use case Tạo Test case.	32
Hình 3.4. Use case Tạo Test script.	32
Hình 3.5. Biểu đồ lớp	35
Hình 3.6. Giao diện chính chương trình.	43
Hình 3.7. Giao diện các kiểu kiểm thử....	43

DANH MỤC BẢNG

Bảng 1.1. Mô tả các bước kiểm thử tự động	6
Bảng 1.2. Bảng so sánh giữa kiểm thử thủ công và kiểm thử tự động.....	7
Bảng 2.1. Bảng liệt kê một số item quan trọng của Namespace System.Reflection....	20
Bảng 2.2. Thành phần chính trong Namespace System.CodeDom.	23
Bảng 2.3. Các kiểu của CodeDom để khai báo kiểu cấu trúc	23
Bảng 2.4. Các kiểu của CodeDom để xây dựng các thành phần.....	23
Bảng 3.1. Các không gian tên được sử dụng phát triển ứng dụng.	28
Bảng 3.2. Dòng sự kiện Use case Tạo Test case.	32
Bảng 3.3. Dòng sự kiện chính Use case Tạo Test script.	33
Bảng 3.4. Danh sách các lớp đối tượng.....	33
Bảng 3.5. Danh sách các thuộc tính của lớp test case	33
Bảng 3.6. Danh sách các phương thức của lớp test case	34
Bảng 3.7. Danh sách các thuộc tính của lớp test script	34
Bảng 3.8. Danh sách các phương thức của lớp test script	34
Bảng 3.9. Danh sách chức năng của module.....	42
Bảng 3.10. Bảng đặc tả giao diện ứng dụng.....	44

MỤC LỤC

MỞ ĐẦU.....	1
1. Lý do chọn đề tài	1
2. Mục đích nghiên cứu	1
3. Khách thể và đối tượng nghiên cứu	1
4. Nhiệm vụ nghiên cứu	1
5. Phạm vi nghiên cứu	1
6. Phương pháp nghiên cứu.....	2
7. Ý nghĩa lý luận và thực tiễn của đề tài	2
CHƯƠNG 1. TỔNG QUAN VỀ KIỂM THỬ PHẦN MỀM.....	3
1.1. Định nghĩa và phân loại kiểm thử phần mềm	3
1.2. Mục tiêu của kiểm thử tự động	5
1.3. Quy trình thực hiện kiểm thử tự động	5
1.4. Các kế hoạch chi tiết kiểm tra	7
1.5. Các công cụ kiểm thử tự động phổ biến.....	11
1.5.1. Công cụ Selenium.....	11
1.5.2. Công cụ Appium.....	11
1.5.3. Công cụ Cypress	12
1.5.4. Công cụ Katalon Studio.....	12
1.5.5. Công cụ Testing Whiz	13
1.5.6. Công cụ TestComplete	14
1.5.7. Công cụ Realine13.....	15
1.5.8. Công cụ Test Craft.....	15
1.6. Kết luận chương 1	16
CHƯƠNG 2. KỸ THUẬT NAMESPACE SYSTEM.REFLECTION VÀ NAMESPACE SYSTEM.CODEDOM	17
2.1. Tìm hiểu về kỹ thuật Reflection trong .Net.....	17
2.1.1. Định nghĩa về Reflection.....	17
2.1.2. Công dụng của Reflection	17
2.1.3. Lớp System.Type	17
2.1.4. Lớp System.Activator	18
2.1.5. Lớp System.Reflection.Assembly	19
2.1.6. Không gian tên System.Reflection	20
2.2.1. Định nghĩa về CodeDom.....	21

2.2.2. Mô hình sử dụng.....	22
2.2.3. Các lớp trong namespace System.CodeDom	22
2.3. Kết luận chương 2	26
CHƯƠNG 3. XÂY DỰNG ỦNG DỤNG KIỂM THỬ TỰ ĐỘNG	27
3.1. Lựa chọn công cụ	27
3.2. Không gian tên của .Net và các lớp phục vụ cho phát triển công cụ	28
3.3. Lưu trữ dữ liệu trên bảng tính worksheet.....	29
3.4. Phân tích thiết kế hệ thống	31
3.4.1. Yêu cầu về chức năng	31
3.4.2. Yêu cầu phi chức năng	33
3.4.3. Phân tích các đối tượng	33
3.5. Thiết kế logic.....	35
3.6. Thiết kế giao diện	42
3.7. Kết luận chương 3	44
KẾT LUẬN VÀ ĐÁNH GIÁ	45
1. Kết quả đạt được	45
2. Hạn chế.....	45
3. Hướng phát triển.....	45
TÀI LIỆU THAM KHẢO.....	46

MỞ ĐẦU

1. Lý do chọn đề tài

Ngành công nghiệp phần mềm Việt Nam đang dần phát triển với nhiều hứa hẹn trong tương lai. Kỹ thuật kiểm thử phần mềm là một phần quan trọng trong quá trình phát triển phần mềm, giúp đảm bảo tính ổn định và chất lượng của ứng dụng. Với những đòi hỏi ngày càng cao về chất lượng và việc rút ngắn tối đa thời gian phát triển, kiểm thử ngày càng trở nên quan trọng và thậm chí nhiều khi là vấn đề sống còn để một dự án phát triển phần mềm.

Các công cụ hỗ trợ kiểm thử tự động trên thị trường cũng có rất nhiều nhưng cũng có những hạn chế nhất định nào đó chưa thực sự tự động hoàn toàn trong quá trình kiểm thử mà vẫn cần rất nhiều công sức của kiểm thử viên. Vì vậy, em đã chọn đề tài “Tìm hiểu và xây dựng một ứng dụng kiểm thử phần mềm tự động” với mong muốn xây dựng một ứng dụng có thể đáp ứng và hỗ trợ cho kiểm thử viên thực hiện công việc một cách dễ dàng, chính xác và nhanh chóng hơn.

2. Mục đích nghiên cứu

- Dưa ra các khái niệm cơ bản của kiểm thử tự động.
- Dưa ra lợi ích của việc xây dựng một ứng dụng kiểm thử tự động.
- Nghiên cứu các kỹ thuật của .NET hỗ trợ xây dựng ứng dụng kiểm thử tự động.
- Thiết kế ứng dụng kiểm thử tự động phần mềm dựa vào kiến thức đã nghiên cứu.

3. Khách thể và đối tượng nghiên cứu

- Khách thể nghiên cứu: Kiến thức về kiểm thử phần mềm tự động, quá trình xây dựng ứng dụng kiểm thử tự động.
- Đối tượng nghiên cứu: Tìm hiểu và xây dựng ứng dụng kiểm thử tự động phần mềm.

4. Nhiệm vụ nghiên cứu

- Nghiên cứu về kiểm thử phần mềm, một số công cụ kiểm thử tự động.
- Nghiên cứu về các kỹ thuật Reflection và CodeDom của .NET.
- Nghiên cứu bài toán kiểm thử tự động phần mềm.
- Áp dụng lý thuyết xây dựng một ứng dụng kiểm thử tự động phần mềm về mặt chức năng của phần mềm đó.

5. Phạm vi nghiên cứu

Nghiên cứu các nội dung bao gồm:

- Khái niệm kiểm thử phần mềm.
- Khái niệm kiểm thử tự động.
- Lợi ích của kiểm thử tự động và xây dựng ứng dụng kiểm thử tự động phần mềm.
- Kỹ thuật Reflection trong .NET.
- Kỹ thuật CodeDom trong .NET.
- Lưu trữ dữ liệu với XML và MS Excel.

6. Phương pháp nghiên cứu

- Tìm kiếm, tham khảo các tài liệu liên quan đến kiểm thử phần mềm, kiểm thử tự động và các kỹ thuật xây dựng ứng dụng tự động, các trang forum, blog về kiểm thử phần mềm như testingvn.com, yinyangit.wordpress.com, testervn.com,...
- Tìm hiểu cách lấy về thông tin của một assembly một cách tự động.
- Tìm hiểu cách xây dựng và thực thi tự động chương trình C#.

7. Ý nghĩa lý luận và thực tiễn của đề tài

- Ý nghĩa lý luận:

Việc nghiên cứu các kỹ thuật kiểm thử phần mềm sẽ cung cấp hiểu biết sâu rộng về lý thuyết và phương pháp kiểm thử, đóng góp vào nâng cao hiệu quả của quá trình phát triển phần mềm.

Đề tài cung cấp cơ hội học hỏi và áp dụng những kiến thức lý luận cơ bản và tiên tiến nhất trong lĩnh vực kiểm thử phần mềm.

- Ý nghĩa thực tiễn:

Áp dụng vào quá trình phát triển phần mềm thực tế: Kiến thức và kỹ năng thu được từ đề tài có thể được áp dụng trực tiếp vào quá trình phát triển phần mềm, giúp cải thiện chất lượng và tính ổn định của ứng dụng.

Phát triển cơ hội nghề nghiệp: Nâng vững kiến thức và kỹ năng trong lĩnh vực kiểm thử phần mềm có thể mở ra cơ hội nghề nghiệp tốt trong ngành Công nghệ Thông tin.

CHƯƠNG 1. TỔNG QUAN VỀ KIỂM THỬ PHẦN MỀM

1.1. Định nghĩa và phân loại kiểm thử phần mềm

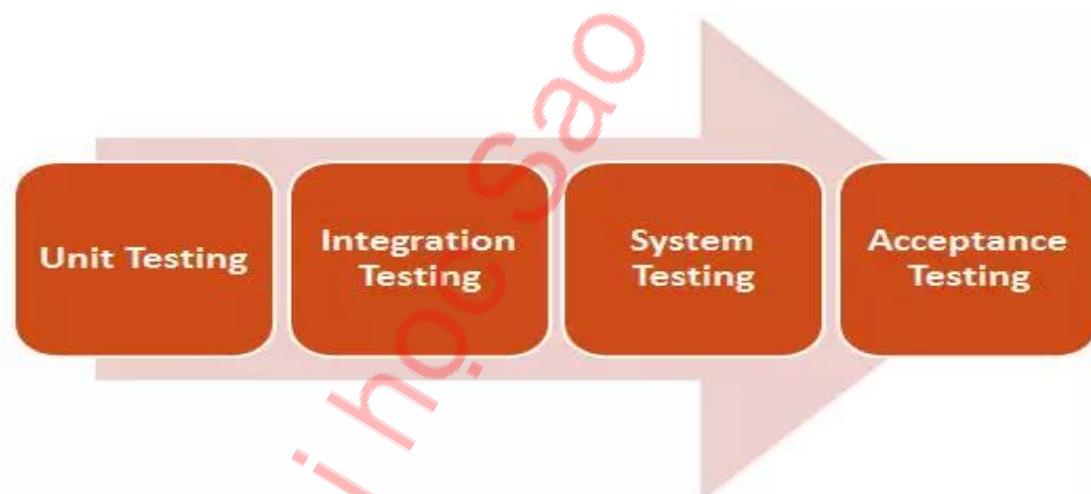
Khái niệm về kiểm thử phần mềm

Kiểm thử phần mềm là bước quan trọng để đảm bảo chất lượng phần mềm, là đánh giá cuối cùng về các đặc tả, thiết kế và mã hóa.

Kiểm thử phần mềm là quá trình chạy một ứng dụng để phát hiện lỗi và xem nó có thỏa mãn các yêu cầu đặt ra không. Trong quá trình phát triển phần mềm, những người phát triển phần mềm và các kỹ sư kiểm thử cùng làm việc để phát hiện lỗi và đảm bảo chất lượng sản phẩm. Một sản phẩm phần mềm được phân phối phải có đầy đủ các chức năng yêu cầu và tương thích với phần cứng của khách hàng.

Mục tiêu đầu tiên của kiểm thử là ngăn ngừa lỗi, ngăn ngừa lỗi còn tốt hơn là sửa lỗi vì ngăn ngừa được lỗi thì sẽ là tốt hơn và không phải sửa mã, giải quyết được vấn đề ngay từ đầu sẽ làm giảm bớt chi phí về thời gian và công sức sửa chữa hơn.

Dưới đây là bốn giai đoạn cơ bản của kiểm thử phần mềm:



Hình 1.1. Các giai đoạn kiểm thử phần mềm

Khái niệm về kiểm thử tự động

Kiểm thử tự động là quá trình thực hiện một cách tự động các bước trong một testcase (trường hợp kiểm thử). Kiểm thử tự động là quá trình sử dụng các công cụ, kịch bản và phần mềm để thực hiện những trường hợp kiểm thử, bằng cách lặp lại những hành động được xác định trước.

Kiểm thử tự động tập trung vào việc thay thế hoạt động thủ công của con người bằng các hệ thống hoặc thiết bị.

Bởi vì kiểm thử tự động được thực hiện thông qua một công cụ tự động hóa, nên nó tiêu tốn ít thời gian hơn trong các thử nghiệm khám phá và hiệu quả hơn trong việc duy trì các kịch bản kiểm tra, đồng thời nâng cao phạm vi kiểm tra tổng thể.

Kiểm thử tự động thích hợp nhất cho các dự án lớn yêu cầu kiểm tra lặp lại các khu vực giống nhau và những dự án đã trải qua quá trình thử nghiệm thủ công ban đầu.

- Một số thuật ngữ được sử dụng trong kiểm thử phần mềm

Test case (trường hợp kiểm thử): Mục đích của Testing là kiểm tra một ứng dụng, phần mềm xem có lỗi phát sinh hay không. Test case là tập hợp các trường hợp có thể xảy ra khi Tester kiểm tra phần mềm.

Việc ứng dụng Test case giúp việc kiểm thử được diễn ra một cách hệ thống và giúp Tester lường trước được mọi khả năng có thể xảy ra. Do đó, việc hiểu được cách viết Test case chuẩn là rất quan trọng.

Test script (kịch bản kiểm thử): Là một công cụ vô cùng hữu ích trong việc thực hiện automation testing. Với các hướng dẫn chi tiết được viết bằng mã code, test script giúp tăng tốc độ kiểm thử tự động và giảm thiểu các lỗi trường hợp thủ công.

Việc sử dụng test script cũng đảm bảo tính hiệu quả và chính xác trong việc thực hiện các ca kiểm thử, từ đó đảm bảo chất lượng sản phẩm được đưa ra thị trường.

Test plan (kế hoạch kiểm thử): Là tài liệu cần phải có đối với bất kỳ quá trình kiểm thử nào. Có thể hình dung Test plan như là chiếc lá bàn mà tester dùng để định hướng đường đi của mình trong suốt quá trình kiểm thử.

Thông qua test plan, tester có thể biết được tất cả những thông tin cần thiết liên quan đến quá trình kiểm tra sản phẩm. Đó có thể là những thông tin về nhân lực, mục tiêu, tài nguyên. Đó cũng có thể là lịch trình cho việc test một phần mềm hoặc phần cứng nào đó.

- Ưu điểm của kiểm thử tự động

Tính tự động hóa: Ưu điểm lớn nhất của kiểm thử tự động là giảm thiểu sự phụ thuộc vào con người lặp đi lặp lại đúng quy tắc các bước kiểm thử nhằm chán, tránh được hao phí về mặt thời gian.

Độ tin cậy: Dù lặp đi lặp lại nhiều lần vẫn cho ra kết quả giống nhau do vậy độ ổn định cao, tránh được rủi ro có thể phát sinh.

Cải thiện chất lượng: Kiểm thử tự động làm giảm rủi ro về mặt chất lượng sản phẩm, việc kiểm thử được thực hiện một cách nhanh chóng. Có thể tái sử dụng các trường hợp kiểm thử.

Tốc độ và hiệu quả: Quá trình thực hiện diễn ra nhanh chóng. Nếu mất 5 phút để kiểm thử thủ công thì chỉ cần mất 30s nếu sử dụng kiểm thử tự động.

Giảm chi phí: Thực hiện nhanh chóng giảm ngắn thời gian và tiết kiệm được lao động giúp cho việc kiểm thử tự động trở nên hiệu quả.

- Nhược điểm của kiểm thử tự động

Chi phí đầu tư ban đầu: Ban đầu thì chi phí cho kiểm thử tự động sẽ cao hơn kiểm thử thủ công vì đòi hỏi sự đầu tư về thời gian và nguồn lực.

Mất chi phí cho các công cụ tự động hóa như bản quyền, bảo trì, tìm hiểu, training.

Khó mở rộng hơn nhiều so với kiểm thử thủ công. Kiểm thử tự động không thể thay thế hoàn toàn kiểm thử thủ công. Một số trường hợp xảy ra vẫn cần sự can thiệp của những người có trình độ chuyên môn cao mới thực hiện được.

Số lượng công việc phải làm để mở rộng cho kiểm thử tự động sẽ nhiều và khó hơn so với kiểm thử thủ công.

1.2. Mục tiêu của kiểm thử tự động

Phần mềm có khiếm khuyết là thông thường và gây ra thiệt hại về kinh tế theo thời gian. Chính vì vậy, các tổ chức về phần mềm dành nhiều thời gian và nguồn lực để phân tích và kiểm thử phần mềm.

Dưới đây là một số mục tiêu cụ thể của kiểm thử tự động

Tăng hiệu quả: Kiểm thử tự động giúp tăng tốc độ kiểm thử, giảm thời gian và chi phí so với kiểm thử thủ công. Các kịch bản kiểm thử có thể được thực hiện nhiều lần một cách nhanh chóng và nhất quán hơn.

Giảm chi phí kiểm thử: Bằng việc tự động hóa quy trình kiểm thử, chi phí và tài nguyên cần thiết cho việc kiểm thử giám sát, từ đó tạo ra hiệu quả kinh tế trong quá trình phát triển phần mềm.

Giảm thiểu sai sót con người: Bằng việc tự động hóa quy trình kiểm thử, nguy cơ phạm sai sót từ con người giảm đi, đảm bảo tính chính xác và nhất quán trong quy trình kiểm thử.

Các mục đích của kiểm thử tự động

- Giảm bớt nhân lực và thời gian thực hiện quá trình kiểm thử.
- Tăng độ tin cậy: Dù có lặp lại nhiều lần nhưng vẫn cho ra kết quả chính xác.
- Giảm chi phí cho tổng quá trình kiểm thử cũng như rèn luyện kỹ năng lập trình cho kiểm thử viên.

Kiểm thử tự động được dùng khi nào

- Khi không đủ tài nguyên: Khi số lượng kịch bản kiểm thử quá nhiều mà kiểm thử viên không thể hoàn thành trong khoảng thời gian yêu cầu nhất định.
- Ứng dụng di động và web: Trong các ứng dụng di động và web, việc kiểm thử tự động cho phép kiểm tra nhanh chóng trên nhiều loại thiết bị và trình duyệt, giúp đảm bảo tính nhất quán trên nhiều nền tảng.
- Những dự án có tính ổn định cao, đặc điểm kỹ thuật được xác định trước, test màn hình - chức năng không thay đổi trong tương lai.

Đảm bảo chất lượng: Bằng việc thực hiện kiểm thử tự động, đảm bảo rằng phần mềm đáp ứng các tiêu chuẩn chất lượng và yêu cầu kỹ thuật được đề ra từ giai đoạn thiết kế đến triển khai.

1.3. Quy trình thực hiện kiểm thử tự động

- Quá trình thực hiện kiểm thử thủ công bằng tay

Lập kế hoạch kiểm thử: Đội ngũ kiểm thử lập kế hoạch chi tiết về phạm vi, các kịch bản kiểm thử và tài nguyên cần thiết cho việc kiểm thử.

Thiết kế các trường hợp kiểm thử: Các ca kiểm thử được thiết kế dựa trên yêu cầu, tình huống sử dụng và các kịch bản kiểm thử. Các trường hợp kiểm thử có thể bao gồm các bước cụ thể để kiểm tra tính năng, hiệu suất và tích hợp của phần mềm.

Chạy ứng dụng với dữ liệu test: Nhân viên kiểm thử thực hiện các trường hợp kiểm thử theo kịch bản đã thiết kế, tương tác với giao diện người dùng và hệ thống để kiểm tra tính đúng đắn và hiệu suất của phần mềm.

Ghi nhận kết quả và so sánh: Kết quả của từng trường hợp kiểm thử được ghi nhận, bao gồm việc xác định lỗi, vấn đề và đề xuất cải thiện.

Phân tích kết quả: Kết quả kiểm thử được phân tích để xác định tính đúng đắn, hiệu suất và tích hợp của phần mềm. Các lỗi và vấn đề được xác định và báo cáo.

Quá trình kiểm thử thủ công đòi hỏi sự can thiệp và xác nhận của con người trong từng bước, từ việc thiết kế trường hợp kiểm thử đến thực hiện và phân tích kết quả. Điều này đôi khi tốn kém thời gian và công sức, đặc biệt khi thực hiện lặp lại kiểm thử cho các phiên bản phần mềm hay tính năng.

- Quy trình của kiểm thử tự động:



Hình 1.2. Quy trình của kiểm thử tự động

Bảng dưới đây mô tả các bước thực hiện kiểm thử tự động:

Bảng 1.1. Mô tả các bước kiểm thử tự động

STT	Các bước thực hiện	Nội dung thực hiện
1	Lập kế hoạch và kiểm soát kiểm thử	Xác định các yêu cầu cần đạt được của kiểm thử.
2	Phân tích và thiết kế kiểm thử	Xác định các điều kiện kiểm thử, thiết kế test, chuẩn bị môi trường test.
3	Thực thi và chạy kiểm thử	Sử dụng các kỹ thuật kiểm thử và tạo các dữ liệu kiểm thử để phát triển và đưa ra độ ưu tiên các trường hợp kiểm thử.

STT	Các bước thực hiện	Nội dung thực hiện
4	Đánh giá và báo cáo	Đánh giá các tiêu chí kết thúc kiểm thử để xác định liệu công việc kiểm thử đã thực hiện đầy đủ hay chưa để so sánh với các tiêu chí ban đầu.
5	Đóng kiểm thử	Đã hoàn thành quá trình thực hiện kiểm thử.

Quy trình kiểm thử tự động phần mềm cũng giống như quy trình thực hiện bằng tay chỉ khác ở chỗ kiểm thử tự động có hỗ trợ của công cụ ít hoặc nhiều như tạo kịch bản (có thể bằng tay hoặc công cụ), công cụ hỗ trợ về ghi lại kết quả và lưu trữ kết quả trong máy tính.

Quy trình này cũng gần tương tự với quy trình phát triển phần mềm, được thực hiện qua nhiều bước, được tiến hành rất sớm trong quy trình phát triển phần mềm và đội kiểm thử tiến hành thư song song cùng đội phát triển phần mềm.

Bảng 1.2. Bảng so sánh giữa kiểm thử công bằng tay và kiểm thử tự động

Kiểm thử thủ công	Kiểm thử tự động
Được con người thực hiện, không chính xác tại mọi lúc, vậy nên nó ít sự tin cậy.	Được thực hiện bởi các công cụ hoặc kịch bản. Đáng tin cậy hơn.
Tốn nhiều thời gian khi kiểm tra.	Được thực hiện bằng các công cụ, vì vậy nó diễn ra nhanh chóng hơn.
Cần nhiều nhân lực để kiểm tra.	Đầu tư cho công cụ kiểm thử.
Phù hợp cho trường hợp thử nghiệm ít lặp đi lặp lại trong khoảng thời gian dài.	Phù hợp cho trường hợp thử nghiệm chạy liên tục trong khoảng thời gian dài.
Phù hợp khi cần có sự giám sát của con người hoặc khách hàng.	Không cần sự giám sát của con người. Nhưng không thể đảm bảo yêu cầu trải nghiệm thực tế của khách hàng.

1.4. Các kế hoạch chi tiết kiểm tra

- Lập kế hoạch

Xác định các ca kiểm thử cần thiết để đảm bảo rằng tất cả các khía cạnh quan trọng của hệ thống được kiểm tra. Lập lịch kiểm thử, xác định nguồn lực và thời gian cần thiết.

- Chuẩn bị môi trường kiểm thử

Thiết lập môi trường kiểm thử, bao gồm cài đặt phần mềm kiểm thử, cấu hình hệ thống, và tạo dữ liệu kiểm thử.

- Xây dựng test script

Tạo các kịch bản kiểm thử tự động bằng cách sử dụng các công cụ và ngôn ngữ lập trình hợp lý. Kịch bản này sẽ giúp tự động hóa quá trình kiểm thử phần mềm.

- Thực hiện kiểm thử

Chạy kịch bản kiểm thử tự động để thực hiện các bước kiểm thử trên hệ thống. Các công cụ kiểm thử tự động này sẽ chạy các kịch bản và trả ra kết quả đạt được.

Ghi lại kết quả: Kiểm tra và ghi lại kết quả của các lần kiểm thử. Đánh giá xem các kịch bản đã hoạt động đúng hay sai, và sau đó ghi lại những lỗi hoặc vấn đề đã xảy ra trong quá trình kiểm thử.

Phân tích kết quả: Xem xét kết quả kiểm thử để xác định được hiệu quả của hệ thống, tìm ra và phân loại các lỗi xảy ra, và đưa ra phương pháp khắc phục.

Báo cáo: Tạo báo cáo trong quá trình kiểm tra và thu được kết quả. Báo cáo này sẽ giúp người quản lý và nhà phát triển nhóm có cái nhìn tổng quan về công việc kiểm tra và hiệu suất của ứng dụng.

Sửa lỗi và kiểm tra lại: Nếu phát hiện ra lỗi trong quá trình kiểm tra, người quản lý sẽ sửa lỗi và kiểm tra lại để đảm bảo rằng lỗi đã được giải quyết bằng một kết quả hiệu quả.

Lặp lại quá trình: Quá trình kiểm tra tự động có thể được lặp lại nhiều lần trong quá trình phát triển và sửa chữa để đảm bảo rằng hệ thống hoạt động đúng và ổn định.

Kiểm tra thử nghiệm tự động giúp tiết kiệm thời gian và tài nguyên, tăng khả năng tái sử dụng và khả năng mở rộng kiểm tra, đồng thời giúp giảm thiểu sai sót nhân công và tăng cường độ tin cậy của ứng dụng.

- Thiết kế Test

Mục tiêu: Chỉ định các thử nghiệm hợp lệ các trường và các bước thử nghiệm chi tiết cho từng phiên bản phần mềm. Giai đoạn thiết kế kiểm tra cực kỳ quan trọng để đảm bảo rằng mọi vấn đề kiểm tra đều quét được tất cả các yêu cầu cần kiểm tra.

- Các bước thiết kế test bao gồm

Xác định và mô tả các trường hợp thử nghiệm: Xác định các điều kiện cần thiết lập trước và trong quá trình kiểm thử. Mô tả đối tượng hoặc dữ liệu đầu vào và mô tả kết quả đạt được sau khi thử nghiệm.

Mô tả các bước kiểm thử chi tiết: Các bước này mô tả chi tiết việc hoàn thành trường hợp kiểm thử trong khi tiến hành kiểm thử. Các trường hợp thử nghiệm nêu trên thường chỉ mô tả đầu vào và đầu ra mà không xác định phương thức thực hiện. Hành động này nhằm trình bày chi tiết các bước của một trường hợp kiểm thử và chỉ định các loại dữ liệu cần thiết để thực hiện trường hợp kiểm thử đó, bao gồm các kiểu dữ liệu trực tiếp, gián tiếp, trung gian và hệ thống,...

Xem xét và khảo sát độ bao phủ của việc kiểm tra: Mô tả các chỉ số và cách thức xác định việc kiểm tra đã hoàn thành hay chưa? Bao nhiêu phần trăm phần mềm đã được

kiểm tra? Để xác định điều này có hai phương pháp: Căn cứ trên yêu cầu của phần mềm hoặc căn cứ trên số lượng code đã viết.

Xem xét Test case và các bước kiểm tra: Việc xem xét cần có sự tham gia của tất cả những người có liên quan, kể cả trưởng dự án nhằm bảo đảm các trường hợp kiểm thử và dữ liệu yêu cầu là đủ và phản ánh đúng các yêu cầu cần kiểm tra, độ bao phủ đạt yêu cầu, cũng như để phát hiện và sửa chữa các sai sót.

- Phát triển Test Script

Mục đích: Bước này thường không bắt buộc trong các loại và mức kiểm tra, chỉ yêu cầu trong những trường hợp đặc thù cần thiết kế, tạo ra các Test Script có khả năng chạy trên máy tính giúp tự động hóa việc thực thi các bước kiểm tra đã định nghĩa ở bước thiết kế test.

- Các bước phát triển Test Script bao gồm

Tạo script kiểm thử: Chạy thử công hoặc sử dụng các công cụ hỗ trợ để tạo script tự động (trong một số trường hợp chúng ta vẫn cần chỉnh sửa ít nhiều script tự động). Mỗi giai đoạn kiểm thử được lên kế hoạch trong phần lập kế hoạch kiểm thử và yêu cầu ít nhất một kịch bản kiểm thử. Kịch bản thử nghiệm có thể được sử dụng lại nhiều nhất có thể để tối ưu hóa hiệu suất.

Kiểm tra tập lệnh kiểm thử: Kiểm tra xem nó có hoạt động tốt hay không để đảm bảo tập lệnh kiểm thử hoạt động theo yêu cầu và thể hiện đúng mục đích của các bước kiểm thử.

Tạo tập dữ liệu bên ngoài cho tập lệnh kiểm tra: Tập lệnh kiểm tra sử dụng tập dữ liệu này để thực hiện kiểm tra tự động. Chúng được gọi là bên ngoài vì chúng được lưu trữ độc lập với tập lệnh kiểm thử, do đó tránh được tình trạng một số người kiểm thử tích hợp dữ liệu vào mã của tập lệnh (thuật ngữ kỹ thuật được gọi là mã cứng). Cách ly dữ liệu cho phép dữ liệu dễ dàng thay đổi trong quá trình thử nghiệm và các tập lệnh được sửa đổi hoặc sử dụng lại sau này.

Xem xét và kiểm tra phạm vi kiểm thử: Đảm bảo tập lệnh kiểm thử bạn tạo bao gồm tất cả các bước kiểm thử cần thiết.

- Thực hiện kiểm tra

Mục tiêu: Thực hiện các bước kiểm tra đã thiết kế và ghi lại các kết quả đạt được.

Việc thực hiện kiểm tra cũng được làm rất nhiều lần trong suốt quá trình kiểm tra, cho đến khi kết quả kiểm tra cho thấy có đủ điều kiện để dừng lại hoặc tạm dừng việc thực hiện kiểm tra.

- Quá trình thực hiện kiểm tra có các bước như sau đây

Thực hiện lần lượt các bước kiểm tra: Làm thủ công hoặc tiến hành các test script nếu là quy trình kiểm tra tự động. Để có thể kiểm tra được, điều đầu tiên ta cần phải làm là xác lập và khởi động môi trường và điều kiện kiểm tra. Việc này để đảm bảo rằng tất cả các bộ phận có liên quan đến (như phần mềm, phần cứng, máy chủ, mạng, dữ liệu,

máy chủ,...) đã được cài đặt và sẵn sàng, trước khi chính thức bắt đầu quá trình thực hiện kiểm tra.

Đánh giá quá trình kiểm tra: Theo dõi chi tiết quá trình kiểm tra thuận lợi cho đến khi hoàn thành hoặc bị treo và dừng lại giữa chừng, có cần sửa chữa hay bổ sung gì không để quá trình kiểm tra được hoàn thiện tốt hơn.

Nếu quá trình bị treo hoặc dừng giữa chừng, kiểm tra viên cần phân tích để xác định nguyên nhân lỗi, khắc phục lỗi và lập lại quá trình kiểm tra.

Nếu quá trình kiểm tra diễn ra suôn sẻ, nhân viên kiểm tra hoàn thành chu kỳ kiểm tra và chuyển tiếp qua bước Thẩm định kết quả kiểm tra.

Thẩm định kết quả kiểm tra: Sau khi kết thúc quá trình, kết quả kiểm tra đạt được cần được xem xét để bảo đảm kết quả nhận được là chuẩn xác, cũng như biết được những lỗi xảy ra không phải do phần mềm mà do dữ liệu dùng để kiểm tra, môi trường kiểm tra hoặc các bước kiểm tra (hoặc Test Script) gây ra. Nếu thực sự lỗi xảy ra do quá trình kiểm tra, cần phải sửa chữa và kiểm tra lại từ đầu.

- Đánh giá quá trình kiểm tra

Mục đích: Đánh giá toàn bộ quá trình kiểm tra, bao gồm xem xét và đánh giá kết quả kiểm tra, liệt kê các lỗi, chỉ ra các yêu cầu thay đổi, và tính toán các số liệu liên quan đến quá trình kiểm tra (như thời gian kiểm tra, số lượng lỗi, số giờ, phân loại lỗi...).

Mục đích của việc đánh giá kết quả là kiểm tra ở bước này hoàn toàn khác với bước thẩm định kết quả kiểm tra sau khi hoàn tất một vòng kiểm tra. Đánh giá kết quả kiểm tra ở giai đoạn này mang tính toàn cục và nhắm vào bản thân giá trị của các kết quả kiểm tra.

Vì việc đánh giá quá trình và kết quả kiểm tra được thực hiện song song với bất kỳ lần kiểm tra nào và chỉ chấm dứt khi quá trình kiểm tra đã hoàn tất.

- Đánh giá quá trình kiểm tra thường thông qua các bước sau

Phân tích kết quả kiểm tra và đề xuất yêu cầu sửa chữa: Chỉ định và đánh giá sự khác biệt giữa kết quả mong chờ và kết quả kiểm tra thực tế, tổng hợp và gửi thông tin yêu cầu sửa chữa đến những người có trách nhiệm trong dự án, lưu trữ để kiểm tra sau đó.

Đánh giá độ bao phủ: Xác định quá trình kiểm tra có đạt được độ bao phủ yêu cầu hay không, tỷ lệ yêu cầu đã được kiểm tra (tính trên các yêu cầu của phần mềm và số lượng code đã viết).

Phân tích lỗi: Dưa ra số liệu phục vụ cho việc cải tiến các quy trình phát triển, giảm sai sót cho các chu kỳ phát triển và kiểm tra sau đó. Ví dụ, tính toán tỷ lệ phát sinh lỗi, xu hướng gây ra lỗi, những lỗi ngoan cố hoặc thường xuyên tái xuất hiện.

Xác định quá trình kiểm tra có đạt yêu cầu đề ra hay không: Phân tích và đánh giá để xem các Test case và chiến lược kiểm tra đã thiết kế có bao gồm hết những thứ cần

kiểm tra hay không? Kiểm tra có đạt yêu cầu dự án đưa ra hay không? Từ những kết quả này kiểm tra viên có thể sẽ phải thay đổi cách thức kiểm tra hoặc chiến lược.

Báo cáo tổng hợp: Tổng hợp lại tất cả kết quả các bước ở trên và phải được gửi cho tất cả những người có liên quan để kiểm tra.

1.5. Các công cụ kiểm thử tự động phổ biến

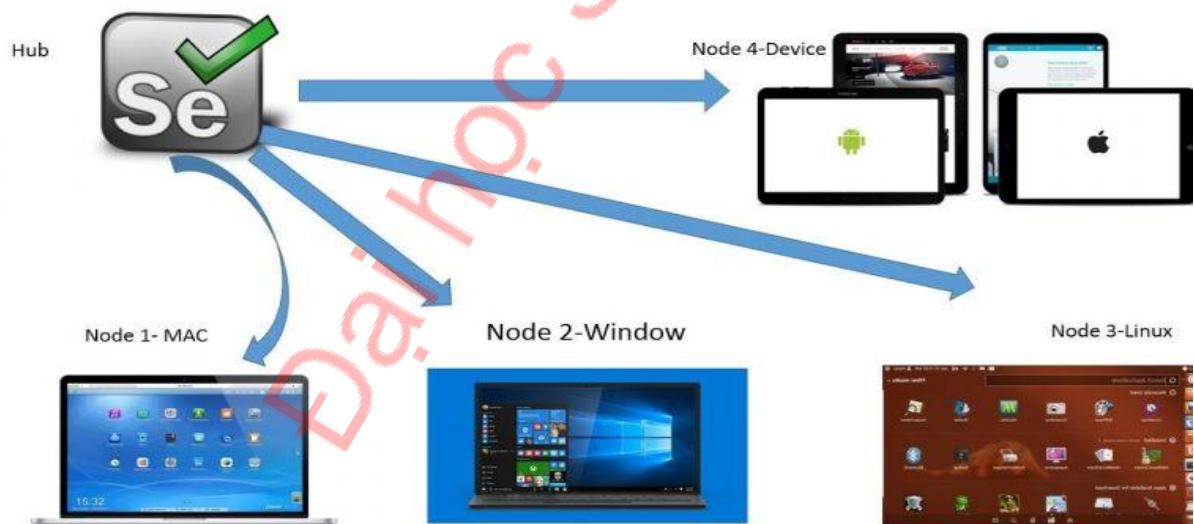
1.5.1. Công cụ Selenium

Được Shinya Kasatani phát triển vào năm 2006, Selenium IDE là một tiện ích mở rộng dành cho Firefox, Chrome giúp tự động hóa chức năng. Thông thường, IDE ghi lại tái sử dụng.

Selenium là một công cụ kiểm thử phần mềm tự động mã nguồn mở miễn phí cho các ứng dụng web trên nhiều trình duyệt và nền tảng khác nhau như Windows, Mac và Linux. Selenium giúp Tester thực hiện kiểm thử bằng nhiều ngôn ngữ lập trình khác nhau như Java, PHP, C#, Python, Groovy, Ruby và Perl.

Selenium hiện có 3 loại: Selenium Webdriver, Selenium IDE, Selenium Grid. Tùy vào kỹ năng, nền tảng và yêu cầu mà bạn có thể lựa chọn sử dụng loại Selenium phù hợp.

Công cụ này phổ biến với tất cả các trình duyệt nổi tiếng hiện tại như Chrome, Mozilla Firefox, Microsoft Edge, Apple Safari, Opera. Vì vậy, Selenium chắc chắn là nền tảng cho hầu hết các công cụ kiểm thử phần mềm khác.



Hình 1.3. Công cụ Selenium

1.5.2. Công cụ Appium

Appium là một công cụ kiểm tra tự động mã nguồn mở, nhưng dành cho các ứng dụng di động. Sử dụng giao thức dây JSON di động, Appium cho phép người dùng viết các bài kiểm tra giao diện người dùng tự động cho các ứng dụng di động gốc, dựa trên web và kết hợp trên cả Android và iOS. Thực thi trên các thiết bị thực, trình mô phỏng và trình giả lập.

Trước đây, các Developer phải kiểm tra tính tương thích của sản phẩm cuối cùng với ba nền tảng riêng biệt (Android, iOS và Windows), đây là một công việc tốn nhiều thời gian và công sức. Với sự ra đời của Appium, các Developer có thể thực hiện kiểm thử tự động toàn bộ sản phẩm cuối cùng bằng cách sử dụng một nền tảng thống nhất.

Bên cạnh đó, Appium có thể khắc phục những nhược điểm của các công cụ tự động hóa khác trên thị trường. Appium có thể chạy liên tục trên nhiều trình giả lập khác nhau, giúp quá trình kiểm thử trở nên tiện lợi hơn bao giờ hết.



Hình 1.4. Công cụ Appium

1.5.3. Công cụ Cypress

Cypress là tool phục vụ cho UI Automation test và chính xác là dành cho web. Nó có chức năng tương đương với Selenium, nhưng được viết bằng Javascript và có cách hoạt động hoàn toàn khác biệt.

Nó được giới thiệu là nhanh, đáng tin cậy hơn selenium và có thể dùng cho nhiều level test, từ Unit, Integration đến End-to-end Test. Vì nó được định hình là framework nên chắc chắn nó sẽ có nhiều tính năng tốt hơn selenium, cái đơn thuần là library.



Hình 1.5. Công cụ Cypress

1.5.4. Công cụ Katalon Studio

Katalon Studio là một công cụ kiểm tra tự động hóa mã code ít và có thể mở rộng cho web, API, máy tính để bàn (Windows) và các ứng dụng di động.

Sau khi loại bỏ các yêu cầu về mã hóa và xây dựng khung tự động thử nghiệm khỏi khung hình vuông, người dùng chỉ có thể tải xuống công cụ và chỉ tập trung vào thử nghiệm. Ngoài ra, Studio cung cấp các bản phát hành thường xuyên để luôn tương thích với các nền tảng, trình duyệt, hệ điều hành mới nhất.

TestComplete có thể tự động hóa kiểm tra chức năng UI cho các ứng dụng máy tính để bàn, thiết bị di động và web.

Với hỗ trợ tích hợp cho hơn 500 điều khiển và khuôn khổ của bên thứ ba, TestComplete có thể xử lý và xác định các phần tử giao diện người dùng động trong hầu hết các công nghệ hiện có.



Hình 1.6. Công cụ Katalon Studio

1.5.5. Công cụ Testing Whiz

TestingWhiz là công cụ kiểm thử phần mềm tự động với phiên bản Enterprise cung cấp một gói hoàn chỉnh gồm nhiều giải pháp test tự động khác nhau.

Trong đó bao gồm có test web, test phần mềm, test database (cơ sở dữ liệu), test API, test ứng dụng di động, bảo trì bộ kiểm tra hồi quy, tối ưu hóa và tự động hóa cũng như kiểm thử trên nhiều trình duyệt.

Ngoài ra, TestingWhiz cung cấp nhiều tính năng quan trọng khác nhau như:

Kiểm thử theo hướng từ khóa (key-word driven), theo hướng dữ liệu (data driven) và kiểm thử phân tán (distributed).

Kiểm thử tiện ích mở rộng trong trình duyệt.

Object Eye Internal Recorder.

Tích hợp với các công cụ theo dõi lỗi như Jira, Mantis, TFS và FogBugz. Tích hợp với các công cụ quản lý kiểm thử như HP Quality Center, Zephyr, TestRail và Microsoft VSTS.



Hình 1.7. Công cụ Testing Whiz

1.5.6. Công cụ TestComplete

Đây là một công cụ kiểm thử tự động rất mạnh và dễ sử dụng từ SmartBear. TestComplete có thể tự động hóa các bài test trên các ứng dụng di động, máy tính để bàn và web, bao gồm các ứng dụng có trang động và điều khiển tùy chỉnh.

Nó hỗ trợ một số ngôn ngữ như JavaScript, Python và VBScript, cũng như một số phương pháp kiểm thử như kiểm thử hồi quy, kiểm thử theo hướng dữ liệu, kiểm thử theo hướng từ khóa và kiểm thử phân tán. Với đặc điểm record-and-replay và công cụ nhận dạng đối tượng mạnh, bạn cũng có thể tạo các tập lệnh kiểm thử tự động ứng dụng web phức tạp với zero code.

Công cụ nhận dạng đối tượng mạnh mẽ của TestComplete đảm bảo rằng UI Testing cho web của bạn không bị gián đoạn hay bị phá vỡ. Người ta có thể chọn từ hơn 50 nghìn thuộc tính đối tượng trên 500 loại điều khiển trở lên.



Hình 1.8. Công cụ TestComplete

1.5.7. Công cụ Realine13

Đây là một công cụ hiện đại đã được khởi chạy cho JMeter Cloud Load Testing. Công cụ này mang lại một số lợi ích và cho phép mở rộng các bài test JMeter của mình thành các kích thước khác nhau. Nó hoạt động với JMeter và nhiều nền tảng và plugin có mã nguồn mở khác.

Ngoài ra, ta có thể dễ dàng thực hiện phần kiểm thử môi trường CI/CD với Jenkins cũng như các nền tảng riêng biệt. RedLine13 là nền tảng kiểm thử rất phổ biến lên tới 50.000 người dùng và 5.000.000 tương tác trở lên có khả năng mở rộng vô thời hạn trong nền tảng đám mây. Nó giúp user kiểm thử mọi ứng dụng đơn lẻ, từ ứng dụng web và thiết bị di động đến các dịch vụ nhỏ và API một cách dễ dàng.



Hình 1.9. Công cụ RedLine13

1.5.8. Công cụ Test Craft

Đây là một nền tảng kiểm tra tự động mạnh mẽ để kiểm tra liên tục và hồi quy, cũng như giám sát các ứng dụng web. Công nghệ Trí tuệ nhân tạo mang tính cách mạng của TestCraft và mô hình trực quan cho phép tạo và triển khai kiểm thử nhanh hơn, đồng thời cho phép chi phí bảo trì kiểm thử triệt để. Các kỹ sư tạo ra các testscript hoàn toàn tự động mà không cần code.

Khách hàng nhanh chóng nhận thấy trực trặc và lỗi, tích hợp với phương pháp CI/CD, phát hành thường xuyên hơn và nâng cao chất lượng hoàn chỉnh cho các sản phẩm kỹ thuật số của họ.



Hình 1.10. Công cụ TestCraft

1.6. Kết luận chương 1

Trong chương này, ta tập trung tìm hiểu khái quát về kiểm thử phần mềm là gì? Kiểm thử tự động là gì? Các lợi ích của kiểm thử tự động. Quy trình kiểm thử tự động? Các kỹ thuật và phương pháp kiểm thử tự động được trình bày một cách cơ bản nhất.

Về các công cụ kiểm thử, trong chương này cũng đã nêu ra một số công cụ kiểm thử phổ biến cũng như chức năng cơ bản của phần mềm.

Đại học Sao Đỏ

CHƯƠNG 2. KỸ THUẬT NAMESPACE SYSTEM.REFLECTION VÀ NAMESPACE SYSTEM.CODEDOM

2.1. Tìm hiểu về kỹ thuật Reflection trong .Net

2.1.1. Định nghĩa về Reflection

Reflection được hiểu là một hàm trong .Net cho phép đọc thông tin từ các siêu dữ liệu (metadata) của assembly để tạo ra một đối tượng (có kiểu là Type) gói gọn thông tin đó. Reflection cho phép trích xuất cuộc gọi và tạo các phương thức, truy cập và thay đổi các thuộc tính của đối tượng khi chạy.

Không gian tên .NET Reflection được dùng để khám phá ra kiểu triển khai hoạt động giống như một lăng kính phản chiếu ánh sáng mặt trời trong một quang phổ ánh sáng. Nó cung cấp các lớp và phương thức để phân biệt một tập hợp thành các loại và thành viên. Do đó, nó được sử dụng như một công cụ kiểm tra tự động để xây dựng dự án.

Không gian tên này trợ giúp để tải một assembly trong thời gian chạy và khám phá các thông tin trong các tệp .exe. Tiếp đó, nó có được danh sách tất cả các loại có trong module nhất định, bao gồm các phương thức, trường, thuộc tính và sự kiện được xác định. Nó cũng có thể lập trình khám phá ra một tập các giao diện được hỗ trợ bởi lớp (hoặc cấu trúc), các tham số của phương thức và các chi tiết liên quan khác (chẳng hạn như các lớp cơ sở, thông tin tên miền,...).

Để lấy thông tin từ một assembly, không gian tên System.Reflection thường sử dụng cùng với các lớp của System.Type. Lớp này có chứa một số phương thức có khả năng truy xuất thông tin về loại đang kiểm tra và nó cũng chứa nhiều loại liên quan để tạo điều kiện thuận lợi cho việc liên kết động cuối cùng của mã assembly.

2.1.2. Công dụng của Reflection

Lấy thông tin về một lớp hoặc đối tượng: Sử dụng reflection để lấy thông tin như tên, thuộc tính, phương thức của một lớp hoặc đối tượng trong thời gian chạy.

Tạo đối tượng: Sử dụng reflection để tạo đối tượng từ một lớp mà không cần biết tên của lớp khi viết mã.

Gọi phương thức: Sử dụng reflection để gọi các phương thức của một đối tượng dựa trên tên phương thức và tham số.

Thực hiện validation: Kiểm tra và xác thực các thuộc tính và phương thức của một đối tượng sử dụng reflection.

2.1.3. Lớp System.Type

Lớp System.Type là một abstract class (lớp trừu tượng) đại diện cho các kiểu dữ liệu. Lớp System.Type còn cung cấp các phương thức và thuộc tính để truy cập thông tin chi tiết về Type, như kiểu dữ liệu, tên, namespace, interfaces triển khai, thuộc tính, phương thức, và nhiều thông tin khác. Đây là lớp chính để thực hiện các cơ chế Reflection đại diện cho các kiểu dữ liệu trong .Net.

Có rất nhiều mục được định nghĩa trong System.Reflection sử dụng lớp trừu tượng System.Type gồm một số phương thức được dùng để lấy về kiểu, thông tin của một đối tượng. Mục đích của việc lấy về này là để gom thông tin cần thiết cho việc kiểm thử. Trong System.Type một tập rất nhiều các thành phần, dưới đây là một số thuộc tính và phương thức sử dụng thường xuyên nhất:

- Phương thức GetType() và toán tử typeof

Dùng phương thức GetType() của lớp Object để trả về đối tượng kiểu Type để diễn tả kiểu dữ liệu của đối tượng. Có được đối tượng Type này rồi, ta sẽ lấy thông tin của kiểu dữ liệu qua các thuộc tính của lớp Type.

Phương thức GetType() chỉ có thể lấy được thông tin từ các biến đối tượng. Trong trường hợp muốn lấy thông tin của lớp thông qua tên lớp, ta phải sử dụng phương thức tĩnh Type.GetType(), tuy nhiên tham số truyền vào cần phải ghi đầy đủ cả namespace.

Sử dụng toán tử typeof, có thể lấy về đối tượng kiểu System.Type của bất kì kiểu nào với cú pháp typeof(type).

- *Lấy các thông tin từ Type*

Gần như các phương thức của System.Type đều được sử dụng để chứa chi tiết các thành viên của kiểu dữ liệu tương ứng – hàm, thuộc tính, phương thức, sự kiện,... có nhiều phương thức nhưng tất cả chúng đều theo nền chung. Ví dụ, có hai phương thức mà nhận chi tiết phương thức của kiểu dữ liệu: getmethod() và getmethods() trả về một tham chiếu đến đối tượng System.Reflection.MethodInfo mà chứa chi tiết của một phương thức.

Getmethods() trả về một mảng tham chiếu trong khi getmethod() trả về chi tiết của một phương thức được chỉ định trong danh sách thông số.

Mỗi phương thức này sẽ trả về một mảng liên quan (ví dụ: GetFields() trả về một mảng FieldInfo, GetMethods() trả về một mảng MethodInfo,...). Các phương thức này có hình thức số ít như GetField(), GetMethod(),GetProperty() để lấy về một mục cụ thể bằng tên hơn là một mảng của tất cả các mục liên quan.

FindMembers(): Trả ra một mảng các kiểu MemberInfo đã được lọc dựa trên cơ sở tìm kiếm nào đó.

GetType(): Trả ra một đối tượng kiểu cho một tên chuỗi.

InvokeMember(): Cho phép ràng buộc cho một mục được đưa ra.

2.1.4. Lớp System.Activator

Lớp System.Activator là một phần của namespace System và cung cấp phương thức tạo các thể hiện của lớp, giao diện hoặc cấu trúc trong thời gian chạy của ứng dụng. Bằng cách sử dụng lớp System.Activator, ta có thể tạo đối tượng một cách động mà không cần phải biết trước tên của lớp.

Một số phương thức chính của lớp System.Activator bao gồm

CreateInstance(string typeName): Tạo một thể hiện của lớp được xác định bởi tên đầy đủ.

CreateInstance(Type type): Tạo một thể hiện của một đối tượng từ đối tượng System.Type đã được xác định trước.

Lớp System.Activator cung cấp một cách linh hoạt để tạo đối tượng trong thời gian chạy của ứng dụng và thường được sử dụng trong kỹ thuật reflection và các kịch bản yêu cầu tạo các đối tượng động.

Ví dụ minh họa dùng hai lớp cho việc tạo đối tượng với constructor không và có tham số.



```

1  namespace ConsoleApplication1
2  {
3
4      class Program
5      {
6          class MyClass1
7          {
8              public void SayHello()
9              {
10                  Console.WriteLine("Hello");
11              }
12          }
13          class MyClass2
14          {
15              string _name;
16
17              public MyClass2(string name)
18              {
19                  _name = name;
20              }
21              public void SayHello()
22              {
23                  Console.WriteLine("Hello, "+_name);
24              }
25          }
26          static void Main(string[] args)
27          {
28              Type mType1 = Type.GetType("ConsoleApplication1.Program+MyClass1");
29              Type mType2 = Type.GetType("ConsoleApplication1.Program+MyClass2");
30
31              object obj1 = Activator.CreateInstance(mType1);
32              object obj2 = Activator.CreateInstance(mType2, "Yin Yang");
33
34              mType1.InvokeMember("SayHello", BindingFlags.InvokeMethod);
35              mType2.InvokeMember("SayHello", BindingFlags.InvokeMethod);
36
37              Console.Read();
38          }
39      }
40  }

```

2.1.5. Lớp System.Reflection.Assembly

Khái niệm: Lớp System.Reflection.Assembly trong .NET Framework đại diện cho một assembly, là một đơn vị mã thực thi (executable code) và tất cả các tệp tài nguyên (resource files) có thể được tải và triển khai trong môi trường thực thi .NET.

Một assembly có thể là một tập hợp các file tài nguyên, bao gồm các tập tin như mã thực thi (executable code), tệp tài nguyên (resource files), metadata, và nhiều nội dung khác liên quan đến ứng dụng.

Lớp System.Reflection.Assembly cung cấp khả năng tương tác với thông tin về một assembly trong quá trình thực thi của ứng dụng. Một số phương thức và thuộc tính phổ biến của lớp System.Reflection.Assembly bao gồm:

GetTypes(): Trả về tất cả các loại (types) đã được định nghĩa trong assembly.

GetExportedTypes(): Trả về danh sách các loại (types) có thể được truy cập từ bên ngoài assembly.

GetName(): Trả về một đối tượng AssemblyName đại diện cho tên của assembly.

GetManifestResourceNames(): Trả về danh sách các tài nguyên có trong assembly.

Lớp System.Reflection.Assembly cho phép truy cập đến thông tin và tài nguyên trong một assembly trong quá trình thực thi của ứng dụng, và thường được sử dụng trong các kịch bản yêu cầu tương tác với các assembly trong quá trình chạy của ứng dụng.

2.1.6. Không gian tên System.Reflection

Như đã nói không gian tên System.Reflection sử dụng để thu thập thông tin trong một assembly, nó định nghĩa một số các lớp kiểu liên quan.

System.Reflection là một namespace trong .NET Framework và .NET Core, chứa các lớp và phương thức cho việc tìm kiếm, xác định và tương tác với thông tin về các loại, thành phần, thuộc tính và phương thức trong khi chương trình đang chạy.

Các thành phần trong namespace System.Reflection cung cấp các cách tiếp cận linh hoạt đến thông tin về metadata của các loại, thuộc tính, phương thức và nhiều nội dung khác của các thành phần của ứng dụng.

Các thành phần trong namespace System.Reflection hỗ trợ các khía cạnh quan trọng của kỹ thuật reflection và là một phần quan trọng của việc phát triển ứng dụng .NET mạnh mẽ và linh hoạt.

Các API giúp thực hiện cơ chế khảo sát kiểu đều nằm trong không gian kiểu System.Reflection.

Tương tự như một số không gian tên khác, nó gồm nhiều item. Bảng sau liệt kê một số item quan trọng:

Bảng 2.1. Bảng liệt kê một số item quan trọng của Namespace System.Reflection

System.Reflection	Công dụng
Assembly	Lớp này (ngoài ra còn nhiều kiểu liên quan) có chứa một số phương thức cho phép bạn tải, tìm hiểu và thao tác với một assembly.
AssemblyName	Lớp này cho phép bạn tìm ra nhiều chi tiết đằng sau một đặc tính của assembly (thông tin phiên bản, thông tin mở rộng).
EventInfo	Lớp này cung cấp thông tin về một sự kiện được trong một lớp.

System.Reflection	Công dụng
FieldInfo	Lớp này cung cấp thông tin về một sự trường dữ liệu (field) trong một lớp.
MethodInfo	Đây là lớp cung cấp thông tin chung về các thành viên (members) của một lớp gồm EventInfo, FieldInfo, MethodInfo và PropertyInfo types.
MethodInfo	Lớp này cung cấp thông tin cho về một phương thức trong một lớp.
Module	Lớp này cho phép bạn truy cập một module trong một assembly đa tệp.
ParameterInfo	Lớp này cung cấp thông tin về một tham số của một phương thức hoặc constructor trong một lớp.
PropertyInfo	Lớp này cung cấp thông tin về một thuộc tính (property) trong một lớp.
ConstructorInfo	Nắm bắt thông tin về thuộc tính của hàm tạo cũng như thông tin mô tả hàm tạo.
ReflectionTypeLoadException	Hàm chức năng Module.GetTypes() sẽ xuất hiện lỗi này khi không tải được lớp nào đó trong mô-đun chỉ định. Không thể thừa kế lớp này.
TargetInvocationException	Lỗi này do hàm chức năng bị triệu gọi thông qua cơ chế Reflection phát ra. Không thể thừa kế lớp này.
TargetParameterCountException	Lỗi này được phát ra khi số tham số truyền cho một hàm bị gọi động không đúng với số tham biến đã khai báo của nó. Không thể thừa kế lớp này.

2.2. Tìm hiểu về kỹ thuật Namespace System.Codedom

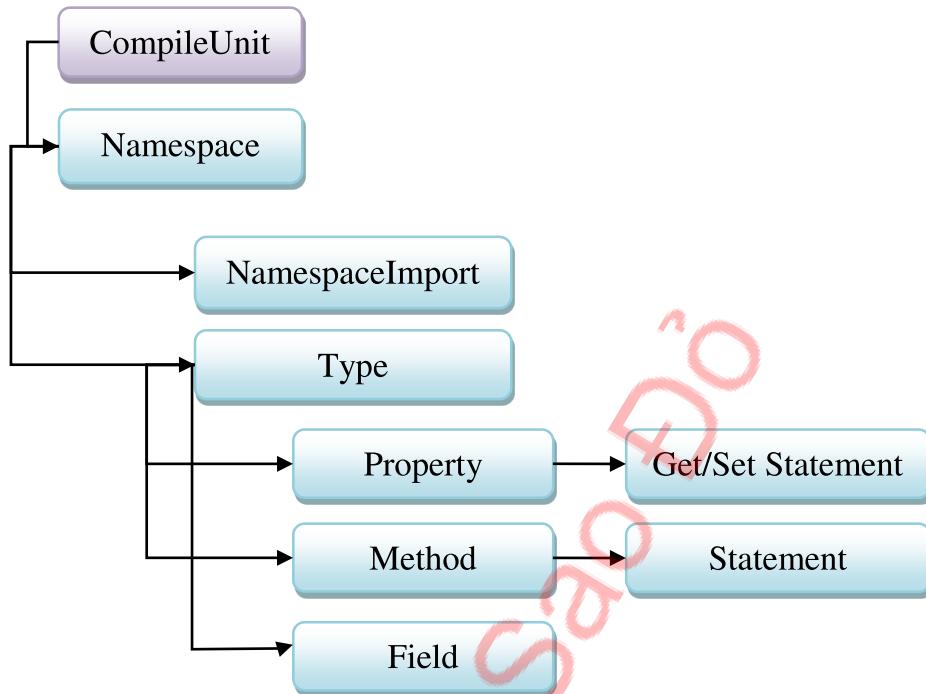
2.2.1. Định nghĩa về CodeDom

CodeDom (Code Document Object Model), nó có thể tự động sinh ra code một cách tự động cho người sử dụng công cụ. Đây là một công nghệ trong .NET Framework cung cấp một cách tiếp cận trừu tượng và độc lập với ngôn ngữ để tạo, biểu diễn và biên dịch mã nguồn.

Với CodeDom, mã nguồn có thể được biểu diễn dưới dạng các đối tượng trong ứng dụng, có thể được duyệt qua và thay đổi trước khi biên dịch. CodeDom cũng cho phép thể hiện mã nguồn không phụ thuộc vào ngôn ngữ lập trình cụ thể, mà thay vào đó phụ thuộc vào cái gọi là provider để tạo ra mã nguồn dựa trên ngôn ngữ lập trình đã chọn.

CodeDom thường được sử dụng trong các trường hợp khi cần tạo mã nguồn một cách động, hoặc khi cần tương tác với ngôn ngữ lập trình từ xa trong quá trình chạy của ứng dụng.

2.2.2. Mô hình sử dụng



Hình 2.1. Mô hình các khái niệm trong CodeDom

Trong đó:

Compile Unit: Đơn vị chứa toàn bộ CodeDom với một collection các Namespace.

Namespace: Tên của namespace.

Namespace Import: Các namespace được dùng(bằng chỉ thị using trong C#).

Type: Các kiểu được dùng như class, structure, interface, enumeration.

Property, Method, Field: Các thành viên của lớp.

Statement/ Expression: Câu lệnh/ biểu thức trong method hoặc property.

2.2.3. Các lớp trong namespace System.CodeDom

Tên miền này cung cấp đầy đủ các kiểu để lập trình viên có thể xây dựng một chương trình có thể sinh ra một chương trình khác hoàn hảo với đầy đủ tên miền, các phương thức, điểm vào để chạy chương trình, các thuộc tính,...

Các kiểu được cung cấp bởi CodeDom để xây dựng namespace cho một chương trình C#.

Không gian tên CodeDom cũng cung cấp một số kiểu để giúp ta xây dựng các thành phần cơ bản của một chương trình. Trong CodeDom, có một số lớp quan trọng được sử dụng để biểu diễn và tương tác với mã nguồn. Một số lớp chủ yếu bao gồm:

Bảng 2.2. Thành phần chính trong Namespace System.CodeDom.

Tên kiểu	Mô tả
CodeNamespace	Xây dựng một khai báo cho một namespace.
CodeNamespaceCollection	Xây dựng một tập các namespace.
CodeNamespaceImport	Xây dựng một trạng thái vào của một namespace.
CodeNamespaceImportCollection	Xây dựng một trạng thái vào của một tập các namespace.

- Các kiểu của CodeDom để khai báo kiểu cấu trúc:

Bảng 2.3. Các kiểu của CodeDom để khai báo kiểu cấu trúc

CodeDom Type để xây dựng các kiểu	Mô tả
CodeTypeDeclaration	Khai báo một lớp, cấu trúc, giao diện. Các định nghĩa kiểu cơ bản tương ứng với một trong các thuộc tính như là: IsClass, IsInterface, IsStruct, and IsEnum.
CodeTypeDeclarationCollection	Khai báo một tập các kiểu.
CodeTypeDelegate	Khai báo một delegate.

- Các kiểu của CodeDom để xây dựng các thành phần:

Bảng 2.4. Các kiểu của CodeDom để xây dựng các thành phần.

CodeDom Type để xây dựng thành phần	Mô tả
CodeConstructor	Tạo một hàm tạo dựng cho một kiểu.
CodeEntryPoint	Tạo một phương thức đi vào của một chương trình với các thuộc tính mặc định (ví dụ, Main()).
CodeMemberEvent	Tạo ra một khai báo sự kiện cho một lớp.
CodeMemberField	Khai báo code của một trường trong lớp.
CodeMemberMethod	Khai báo code của một phương thức trong lớp.

CodeDom Type để xây dựng thành phần	Mô tả
CodeMemberProperty	Khai báo code của một thuộc tính trong lớp.
CodeParameterDeclarationExpression	Khai báo code của một tham số của một thành viên.
CodeTypeConstructor	Đại diện cho hàm khởi tạo tĩnh cho một lớp. Hàm khởi tạo này được gọi khi kiểu của nó được chạy.
CodeTypeMember	Sử dụng một lớp cơ sở trừu tượng để khởi động một thành viên trong một kiểu.
CodeTypeMemberCollection	Khai báo một tập thành viên trong một kiểu được định nghĩa.
MemberAttributes	Khai báo các thuộc tính với các định dạng được sử dụng bởi CodeTypeMember.

- *Sử dụng CodeDom*

CodeDom cung cấp các loại đại diện cho nhiều loại phần tử mã nguồn phổ biến. Ta có thể thiết kế chương trình xây dựng mô hình mã nguồn bằng cách sử dụng các phần tử CodeDom để tập hợp biểu đồ đối tượng.

Biểu đồ đối tượng này có thể được hiển thị dưới dạng mã nguồn bằng cách sử dụng trình tạo mã CodeDom cho ngôn ngữ lập trình được hỗ trợ. CodeDom cũng có thể được sử dụng để biên dịch mã nguồn thành một tập hợp nhị phân.

Một số cách sử dụng phổ biến của CodeDom bao gồm

Tạo mã theo khuôn mẫu: Tạo mã cho ASP.NET, proxy máy khách dịch vụ Web XML, trình hướng dẫn mã, nhà thiết kế hoặc các cơ chế phát mã khác.

Biên dịch động: Hỗ trợ biên dịch mã bằng một hoặc nhiều ngôn ngữ.

Xây dựng đồ thị CodeDom

Không gian tên System.CodeDom cung cấp các lớp để biểu diễn cấu trúc logic của mã nguồn, độc lập với cú pháp ngôn ngữ.

Cấu trúc của đồ thị CodeDom

Cấu trúc của biểu đồ CodeDom giống như một cây chứa các thùng chứa. Vùng chứa trên cùng hoặc gốc của mỗi biểu đồ CodeDom có thể biên dịch được là CodeCompileUnit.

Mọi phần tử của mô hình mã nguồn của bạn phải được liên kết vào biểu đồ thông qua thuộc tính của CodeObject trong biểu đồ.

Tạo một đơn vị biên dịch

CodeDom định nghĩa một đối tượng được gọi là CodeCompileUnit, đối tượng này có thể tham chiếu biểu đồ đối tượng CodeDom mô hình hóa mã nguồn để biên dịch.

CodeCompileUnit có các thuộc tính để lưu trữ các tham chiếu đến các thuộc tính, không gian tên và tập hợp.

Các nhà cung cấp CodeDom xuất phát từ lớp CodeDomProvider chứa các phương thức xử lý biểu đồ đối tượng được tham chiếu bởi CodeCompileUnit.

Để tạo biểu đồ đối tượng cho một ứng dụng đơn giản, ta phải tập hợp mô hình mã nguồn và tham chiếu nó từ CodeCompileUnit.

Xác định một không gian tên

Để xác định một không gian tên, hãy tạo một CodeNamespace và gán tên cho nó bằng cách sử dụng hàm tạo thích hợp hoặc bằng cách đặt thuộc tính tên của nó.

Nhập không gian tên

Để thêm chỉ thị nhập vùng tên vào vùng tên, hãy thêm CodeNamespaceImport cho biết vùng tên cần nhập vào bộ sưu tập CodeNamespace.Imports.

Liên kết các phần tử mã vào biểu đồ đối tượng

Tất cả các phần tử mã tạo thành biểu đồ CodeDom phải được liên kết với CodeCompileUnit, phần tử gốc của cây bằng một loạt tham chiếu giữa các phần tử được tham chiếu trực tiếp từ thuộc tính của đối tượng gốc của biểu đồ.

Đặt đối tượng thành thuộc tính của đối tượng vùng chứa để thiết lập tham chiếu từ đối tượng vùng chứa.

Xác định một loại

Để khai báo một lớp, cấu trúc, giao diện hoặc bảng liệt kê bằng CodeDom, hãy tạo CodeTypeDeclaration mới và gán tên cho nó.

Để thêm một loại vào không gian tên, hãy thêm CodeTypeDeclaration đại diện cho loại cần thêm vào không gian tên vào bộ sưu tập loại của CodeNamespace.

Thêm thành viên lớp vào một lớp

Không gian tên System.CodeDom cung cấp nhiều phần tử khác nhau có thể được sử dụng để đại diện cho các thành viên của lớp. Mỗi thành viên của lớp có thể được thêm vào bộ sưu tập thành viên của CodeTypeDeclaration.

Xác định phương thức điểm nhập mã cho tệp thực thi

Nếu đang xây dựng mã cho một chương trình thực thi, cần phải chỉ ra điểm vào của chương trình bằng cách tạo CodeEntryPointMethod để thể hiện phương thức mà tại đó việc thực thi chương trình sẽ bắt đầu.

Thông tin thêm về cách xây dựng biểu đồ CodeDom

CodeDom hỗ trợ nhiều loại thành phần mã phổ biến được tìm thấy trong các ngôn ngữ lập trình hỗ trợ thời gian chạy ngôn ngữ chung.

CodeDom không được thiết kế để cung cấp các phần tử thể hiện tất cả các tính năng có thể có của ngôn ngữ lập trình. Tuy nhiên, CodeDom không thể dịch các đoạn mã sang các ngôn ngữ khác một cách tự động.

2.3. Kết luận chương 2

Trong chương này, ta đi vào tìm hiểu hai nội dung chính. Đầu tiên là kỹ thuật Namespace System.Reflection trong .Net. Ở phần này ta biết về khái niệm, các lớp cũng như mô hình sử dụng chính của nó. Thứ hai là kỹ thuật Namespace System.CodeDom, phần này ta cũng tìm hiểu khái niệm, một số loại và phương thức thường hay được sử dụng trong kỹ thuật.

Đại học Sao Đỏ

CHƯƠNG 3. XÂY DỰNG ỦNG DỤNG KIỂM THỬ TỰ ĐỘNG

3.1. Lựa chọn công cụ

Xác định được mục tiêu kiểm thử, loại kiểm thử để lựa chọn công cụ kiểm thử phù hợp. Khi lựa chọn công cụ để test, bạn cần chú ý sau:

- Nó có dễ dàng để phát triển và duy trì các script cho công cụ hay không?
- Nó có hoạt động trên các nền tảng như web, điện thoại di động, máy tính để bàn, v.v... không?
- Công cụ có chức năng báo cáo kiểm thử không?
- Công cụ này có thể hỗ trợ bao nhiêu loại kiểm thử?
- Công cụ hỗ trợ bao nhiêu ngôn ngữ?

Một công cụ kiểm thử hoàn thiện có thể giảm chi phí cho việc phát triển và làm tăng chất lượng phần mềm. Một công cụ kiểm thử tự động hiệu quả cần đáp ứng những yêu cầu sau:

- Hỗ trợ nhiều nền tảng và ngôn ngữ: Có khả năng kiểm thử trên nhiều nền tảng và hỗ trợ nhiều ngôn ngữ lập trình khác nhau.
- Tự động hóa: Có khả năng thực hiện kiểm thử tự động trên nhiều môi trường khác nhau, giảm đi sự phụ thuộc vào thủ công và tiết kiệm được nhiều thời gian.
- Báo cáo và theo dõi kết quả: Có khả năng tạo báo cáo chi tiết về quá trình kiểm thử và theo dõi kết quả đạt được để phát hiện lỗi dễ dàng.
- Tích hợp với công cụ quản lý lỗi và yêu cầu: Có khả năng tích hợp với các công cụ quản lý lỗi xảy ra và yêu cầu để theo dõi và quản lý các lỗi được phát hiện trong quá trình kiểm tra.
- Hỗ trợ kiểm thử hiệu suất và tải: Có khả năng thực hiện kiểm thử hiệu suất và tải để đảm bảo ứng dụng có thể hoạt động hiệu quả trong điều kiện tải cao.
- Dễ dàng sử dụng và bảo trì: Có giao diện và cấu trúc dễ sử dụng, giúp người dùng tạo và bảo trì các kịch bản kiểm thử một cách dễ dàng.

Hiện nay, các công cụ ngày càng hoàn thiện đa dạng và nhiều chức năng hơn, cách dùng cũng dễ dàng và tiện lợi hơn rất nhiều so với trước kia.

Đối với đề tài này, để thực hiện đề xuất ra phương pháp xây dựng một ứng dụng kiểm thử tự động phần mềm viết trên ngôn ngữ C#, thực hiện được các chức năng đơn giản và cơ bản như test về mặt chức năng, test đơn vị cho chương trình, thực hiện ghi nhận kết quả test xuất ra file Excel.

Mục đích xây dựng công cụ hỗ trợ cho kiểm thử, phát triển công cụ với mục tiêu đề ra là thực hiện hỗ trợ kiểm thử viên viết test script, nhập dữ liệu các test case từ các bản excel và thực thi tự động test script sau đó xuất kết quả trả về cho Excel, so sánh đánh giá kết quả.

Để thực hiện được việc đó, bằng ngôn ngữ lập trình C#, chúng ta quan tâm đến các không gian tên .NET Reflection, .NET Type class, MS Excel API programming và XML để hoàn thành.

3.2. Không gian tên của .Net và các lớp phục vụ cho phát triển công cụ

Bảng 3.1. Các không gian tên được sử dụng phát triển ứng dụng.

Không gian tên (Namespace)	Công dụng
System	Chứa các lớp có các chức năng cơ bản như chuyển đổi dữ liệu, các toán tử toán học, chương trình gọi, thu gom rác và quá trình quản lý môi trường. Đây là không gian tên lớn nhất được cung cấp bởi .NET.
System.CodeDom	Có các lớp bao gồm các phần tử của dữ liệu mã nguồn. Nó được dùng để sinh ra các test script một cách tự động.
System.Collections	Chứa các lớp xác định các tập của đối tượng giống như các list, queue, sorted list, mảng (array), bảng băm và từ điển. Công cụ kiểm thử sử dụng không gian tên này để lưu trữ dữ liệu tạm thời.
System.ComponentModel	Chứa các lớp tạo ra các thành phần và điều khiển thời gian chạy và thời gian thiết kế.
System.Data	Chứa các lớp định nghĩa kiến trúc truy cập dữ liệu ADO.NET. Các kiến trúc ADO.NET cho phép xây dựng các thành phần có thể quản lý dữ liệu từ nhiều nguồn dữ liệu khác nhau trong cơ chế kết nối hoặc phi kết nối.
System.Diagnostics	Có các lớp có thể được dùng để gỡ lỗi các ứng dụng .NET, để theo dõi việc thực thi các mã code của bạn và để theo dõi hiệu suất bộ đếm và đọc, viết các bản ghi sự kiện, và các quá trình bắt đầu và dừng lại.
System.Drawing	Chứa các lớp để thêm vào giao diện thiết bị đồ họa phục vụ vẽ chức năng.
System.IO	Chứa các lớp có thể đọc và viết các luồng dữ liệu và các tệp trên đĩa với các thông số kỹ thuật của luồng vào/ ra đồng bộ và không đồng bộ.
System.Net	Có các lớp cung cấp một lớp bao xung quanh nhiều giao thức mạng được dùng hiện nay và xử lý các yêu cầu DNS, HTTP và FTP. Nếu bạn quan tâm đến kiểm thử web thì không gian tên này rất hữu ích.

Không gian tên (Namespace)	Công dụng
System.Reflection	Chứa các lớp cung cấp một cái nhìn về kiểu, phương thức với các tham số, thuộc tính và trường có sẵn trong ứng dụng .NET. Thậm chí có thể lập trình tạo và gọi các kiểu (types) ngay tại thời gian chạy.
System.Resources	Cung cấp các lớp giúp người phát triển dự án tạo ra, lưu trữ và quản lý tài nguyên trong ứng dụng nội tại.
System.Runtime.InteropServices	Cung cấp chức năng cho mã nguồn không được quản lý.
System.Text	Chứa các lớp có thể sử dụng để làm việc với các bảng mã ký tự: ASCII, Unicode, UTF-7 và UT.
System.Threading	Chứa các lớp cho phép đa luồng hệ thống vận hành cùng lúc trong ứng dụng .NET, do đó dùng để tạo ra một ứng dụng đa luồng thực sự.
System.Timers	Có các lớp để bắn ra một sự kiện vào một khoảng thời gian đã được định sẵn hoặc theo một lịch trình thời gian phức tạp hơn.
System.Web	Có các lớp để triển khai các giao thức chuyển đổi siêu văn bản (HTTP).
System.Windows.Forms	Chứa các lớp để xây dựng một ứng dụng Windows đầy đủ tính năng với các tính năng như hộp thoại, menu và các nút nhấn.
System.XML	Có các lớp để xử lý dữ liệu XML. Cung cấp hỗ trợ cho XML 1.0, không gian tên XML, lược đồ XML, XPath, XSL và XSLT, DOM Level 2, và SOAP 1.1.

3.3. Lưu trữ dữ liệu trên bảng tính worksheet

Quá trình thực thi cần không gian để lưu trữ dữ liệu cho việc kiểm tra thành phần. Dữ liệu bao gồm tên lớp trong mã hợp ngữ, tên của cấu trúc, lớp hoặc các thành viên khác của cấu trúc và các giá trị đối số của các phương thức hoặc cấu trúc được lấy từ các lớp Reflection ở trên. Người kiểm tra thường phải thu thập dữ liệu được ghi lại này bằng các công cụ hoặc mã có sẵn trên thị trường mã nguồn mở.

Khi người kiểm thử thực hiện kiểm thử thủ công, kỹ sư kiểm thử thường lưu trữ dữ liệu ở nhiều định dạng dữ liệu khác nhau. Một công cụ kiểm tra thương mại cung cấp một phương pháp lưu trữ dữ liệu và yêu cầu tuân thủ phương pháp đó. Thông tin từ cả

công cụ kiểm tra thương mại và công cụ nguồn mở thường được mã hóa cứng vào tập lệnh kiểm tra.

Có nhiều cách để lưu dữ liệu kiểm thử cho công cụ kiểm thử phần mềm này. Bởi vì tài liệu XML đã trở thành tiêu chuẩn để lưu trữ và trao đổi dữ liệu cho nhiều tổ chức phần mềm. Trong Microsoft Visual Studio .NET IDE, C# cho phép các nhà phát triển thêm tài liệu XML và khi làm như vậy, nó sẽ tự động tạo ra một tài liệu trợ giúp XML.

Các nhà phát triển nhận thấy việc thực hiện một trường hợp thử nghiệm trong khi mã hóa một phương thức bằng các dòng lệnh XML rất dễ dàng và hiệu quả. Công cụ kiểm tra tự động đọc dữ liệu được lưu trữ trong tài liệu XML. Sử dụng phương pháp này, người thử nghiệm không bao giờ phải lo lắng về việc đoán mã khi thu thập dữ liệu thử nghiệm. Ứng dụng do nhóm làm việc xây dựng sử dụng bảng tính MS Excel và tài liệu XML để phát triển công cụ.

Một đối tượng MS Excel được tổ chức theo mô hình phân cấp đối tượng của nó. Các ứng dụng nằm ở đầu hệ thống phân cấp. Nó chứa một tập hợp các bảng tính, biểu đồ và các cửa sổ khác.

Bảng tính chứa một số bảng nhỏ. Một bảng nhỏ bao gồm một số phạm vi và ô được sắp xếp theo cột và hàng.

Dưới đây là kết quả khi chương trình lấy tên lớp, các phương thức của lớp đó và các đối số của phương thức, tất cả đều được trình bày trong MS Excel:

TÊN LỚP	TÊN PHƯƠNG THỨC	CÁC THAM SỐ ĐẦU VÀO
AdvancedCalc	AdvancedCalc	1880.444886
AdvancedCalc	AdvancedCalc	1693.173132
AdvancedCalc	Triangle	1386.41377
AdvancedCalc	Triangle	1035.042
AdvancedCalc	Triangle	1266.926162
AdvancedCalc	GetTriangleHeight	2037.454
AdvancedCalc	isRightTriangle	1686.082
AdvancedCalc	Square	1917.966482
AdvancedCalc	Circle	541.0108
AdvancedCalc	SimpleCalc	189.6389
AdvancedCalc	Sum	1192.051083
AdvancedCalc	ToString	840.6792
AdvancedCalc	Equals	1611.207
AdvancedCalc	GetHashCode	FALSE
AdvancedCalc	GetType	

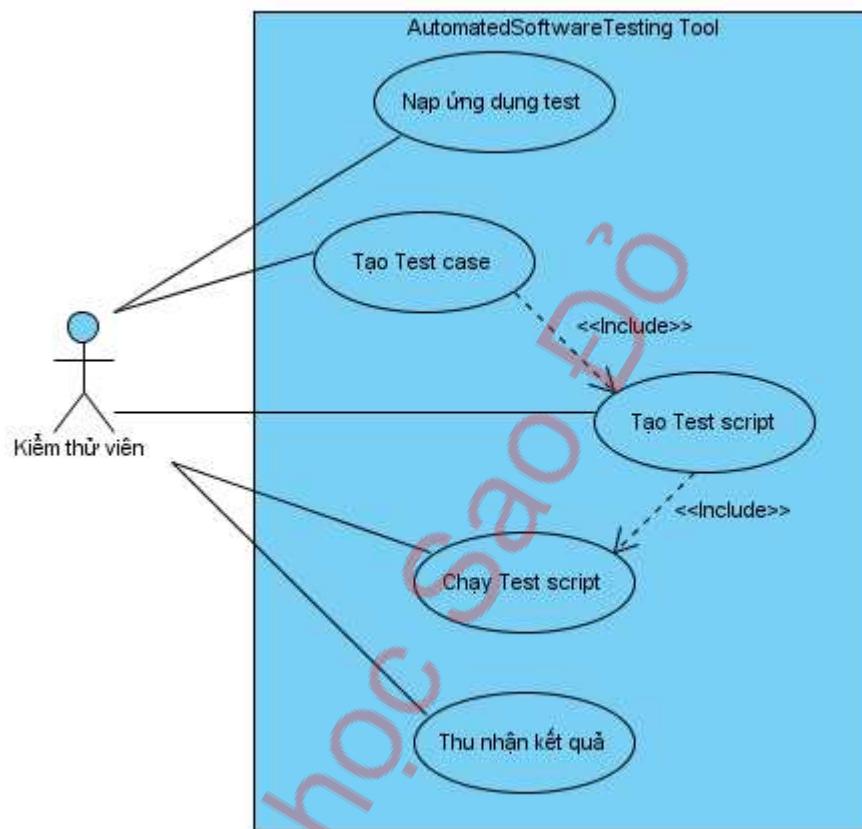
Hình 3.1. Kết quả chạy test script lưu trên file MS excel.

Các bảng tính này còn được sử dụng để lưu các test case phục vụ cho quá trình thực thi test script và sau đó lưu trữ kết quả test sau khi chạy test script.

3.4. Phân tích thiết kế hệ thống

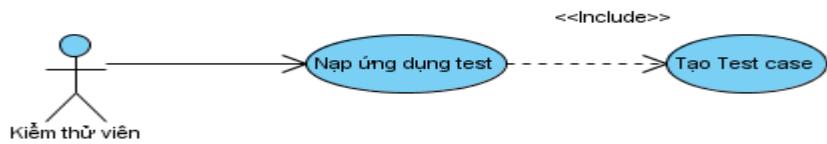
3.4.1. Yêu cầu về chức năng

- Biểu đồ Use case tổng quát của hệ thống:



Hình 3.2. Use case tổng quát.

- Chức năng 1 (Use case Tạo Test case):
 - Mục đích: Sinh ra Test case tự động dựa vào các yếu tố đầu vào là ứng dụng test.
 - Tác nhân liên quan: Kiểm thử viên.
 - Điều kiện trước: Sau khi công cụ test đã được bật, kiểm thử viên đã có chương trình chạy để kiểm thử.
 - Điều kiện sau: Sau khi test case được xuất ra, đối với các tham số của phương thức, hàm thì Kiểm thử viên có thể chỉnh sửa các tham số đầu vào này để các trường hợp kiểm thử phù hợp và có lợi nhất.
 - Biểu đồ Use case



Hình 3.3. Use case Tạo Test case.

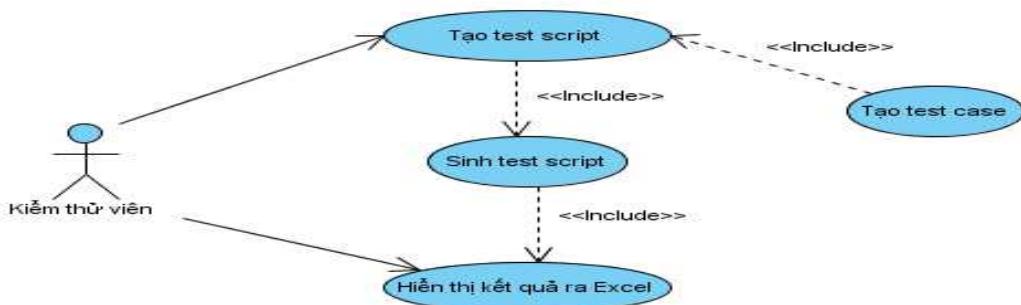
- Dòng sự kiện chính

Bảng 3.2. Dòng sự kiện Use case Tạo Test case.

Hành động của tác nhân	Phản ứng của hệ thống
1. Kiểm thử viên nạp chương trình (file assembly) vào công cụ.	2. Hệ thống hiển thị thư mục đường dẫn thư mục cho phép nạp chương trình test vào công cụ.
3. Kiểm thử viên chọn chương trình cần test.	4. Hệ thống xuất các dữ liệu của chương trình test như: lớp, thuộc tính, phương thức, cấu trúc ra file Excel và mặc định tham số đầu vào đối với các phương thức có tham số.

- Chức năng 2 (Use case Tạo Test script):

- Mục đích: Sinh ra Test script tự động dựa vào các yếu tố đầu vào là ứng dụng test và test case đã tạo ra được trước đó.
- Tác nhân liên quan: Kiểm thử viên.
- Điều kiện trước: Sau khi công cụ test sinh ra dc test case, kiểm thử viên đã chỉnh sửa test case và chấp nhận tạo test script.
- Điều kiện sau: Sau khi test script được tạo thì công cụ tự động chạy test script và cho ra kết quả.
- Biểu đồ Use case



Hình 3.4. Use case Tạo Test script.

- Dòng sự kiện chính

Bảng 3.3. Dòng sự kiện chính Use case Tạo Test script.

Hành động của tác nhân	Phản ứng của hệ thống
1. Nhấn vào nút tạo test script.	2. Hệ thống tự động sinh test script và mở test script lên màn hình.
3. Kiểm thử viên chạy (debug) test script.	4. Hệ thống thực thi test script và xuất kết quả thực hiện ra file Excel.

3.4.2. Yêu cầu phi chức năng

- Yêu cầu về hiệu năng

Ứng dụng thực hiện tạo test case và test script nhanh, thời gian phản hồi kết quả không quá 3s.

- Các ràng buộc thiết kế

Ứng dụng phát triển trên nền tảng .NET framework với công cụ Visual Studio với ngôn ngữ lập trình là C# và sử dụng ứng dụng MS Excel để lưu trữ dữ liệu test.

Ứng dụng được thiết kế có giao diện Window Form sáng sủa, khoa học, dễ thao tác và làm việc.

3.4.3. Phân tích các đối tượng

- Danh sách các lớp đối tượng

Bảng 3.4. Danh sách các lớp đối tượng.

STT	Tên lớp	Mô tả
1	Test case	Trường hợp kiểm thử chức năng cho ứng dụng.
2	Test script	Testscript được tạo ra để test ứng dụng.

- Chi tiết hóa các lớp đối tượng

Lớp Test case:

Danh sách các thuộc tính:

Bảng 3.5. Danh sách các thuộc tính của lớp test case .

STT	Tên thuộc tính	Mô tả
1	MaTC	Mã Test case
2	TenTC	Tên Test case
3	Class	Tên lớp cần kiểm tra

STT	Tên thuộc tính	Mô tả
4	Method	Phương thức thuộc lớp
5	Constructor	Cấu trúc thuộc lớp
6	Property	Thuộc tính của lớp
7	GTVao	Giá trị đầu vào
8	GTRa	Giá trị đầu ra
9	GTMongcho	Giá trị mong chờ của test case

- Danh sách phương thức:

Bảng 3.6. Danh sách các phương thức của lớp test case.

STT	Tên phương thức	Mô tả
1	Sinh test case	Sinh tự động test script

- Danh sách các thuộc tính:

Bảng 3.7. Danh sách các thuộc tính của lớp test script .

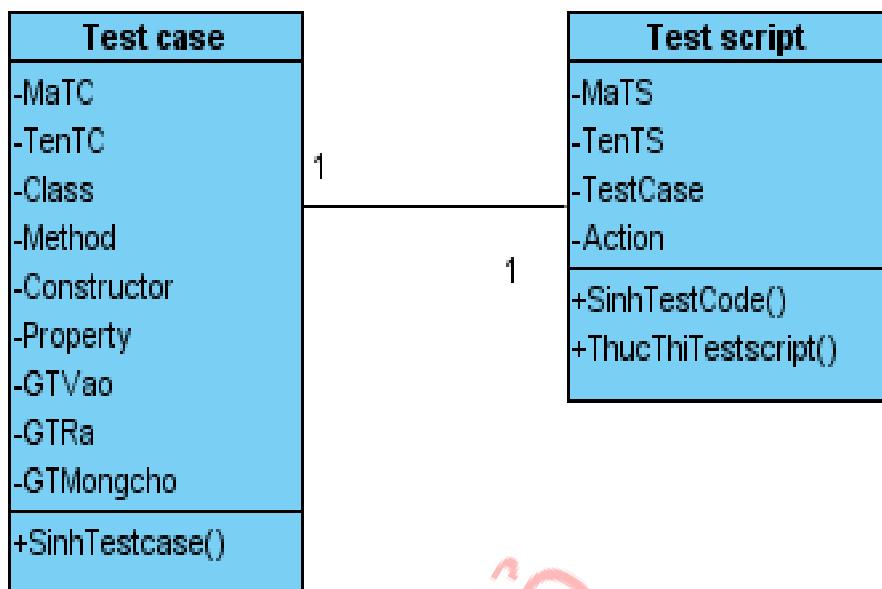
STT	Tên thuộc tính	Mô tả
1	MaTS	Mã Test script
2	TenTS	Tên Test script
3	TestCase	Test case tương ứng
4	Action	Hành động thực hiện

- Danh sách phương thức:

Bảng 3.8. Danh sách các phương thức của lớp test script .

STT	Tên phương thức	Mô tả
1	Sinh Test Script	Sinh tự động test script
2	Thực thi Test script	Thực thi test script để tạo ra kết quả test

- Mô hình hóa các lớp đối tượng



Hình 3.5. Biểu đồ lớp

3.5. Thiết kế logic

- Code phương thức tạo thư mục kiểm thử

```

private void TaoThuMucKiemThu()
{
    DirectoryInfo dir = new DirectoryInfo(tempTestProjDir);
    try
    {
        if (dirName != null)
            dir.CreateSubdirectory(dirName);
    }
    catch (IOException err) { MessageBox.Show(err.Message); }
    dir = new DirectoryInfo(tempTestProjDir + "/" + dirName);
    try
    {
        dir.CreateSubdirectory("Bin");
    }
    catch (IOException err) { MessageBox.Show(err.Message); }
    dir = new DirectoryInfo(tempTestProjDir + "/" + dirName + "/Bin");
    try
    {
        dir.CreateSubdirectory("Debug");
    }
    catch (IOException err) { MessageBox.Show(err.Message); }
}

```

- Phương thức “TaoNguonThuThapDuLieu” và “ThaoTacCopyFile”

```

private void TaoNguonThuThapDuLieu(string Content)
{
    FileInfo f = new FileInfo(System.IO.Path.Combine(tempTestProjDir + "/" + dirName + "/Bin/Debug", "fileArray.txt"));
    StreamWriter sw = f.CreateText();
    sw.Write(Content);
    sw.Close();
}

string rootDir = Environment.CurrentDirectory;
2 references
private void ThaoTacCopyFile(string LoaiFile)
{
    FileInfo f = null;
    if (LoaiFile == "ico")
    {
        f = new FileInfo(rootDir + "/testApp.ico");
        try
        {
            FileInfo AppF = f.CopyTo(AppIconFilename, true);
        }
        catch (Exception err) { MessageBox.Show(err.Message); }
    }
    else
    {
        f = new FileInfo(rootDir + "/TestAssemblyInfo.cs");
        try
        {
            FileInfo AppF = f.CopyTo(AsmInfoFilename, true);
        }
        catch (Exception err) { MessageBox.Show(err.Message); }
    }
}

```

- Phương thức “BatDauDOTNETIDE” được sử dụng để khởi động một ứng dụng có tên là fileName với các tham số args. Trong phương thức này, một đối tượng của lớp Process được tạo ra để khởi tạo một tiến trình mới. Cuối cùng, phương thức Start() của đối tượng Process được gọi để khởi động tiến trình mới với các thông số đã được cấu hình trước đó.

```

private void BatDauDOTNETIDE(string fileName, string args)
{
    try
    {
        Process shell = new Process();
        shell.StartInfo.FileName = fileName;
        shell.StartInfo.Arguments = args;
        shell.Start();
    }
    catch (Exception er)
    { MessageBox.Show(er.Message); }
}

```

- Phương thức TestForm_Load(object sender, EventArgs e):

Đây là một sự kiện xảy ra khi Form được tải lên và sẵn sàng để hiển thị. Trong phương thức này, đoạn mã kiểm tra và lấy đường dẫn của IDE (môi trường phát triển tích hợp, ví dụ như Visual Studio) từ các biến môi trường của hệ thống.

Nếu tìm thấy đường dẫn, nó sẽ hiển thị trong một ô văn bản (textbox) có tên là txtDotNETLocation. Sau đó, phương thức gán trọng tâm vào nút (button) có tên là “btnStart”.

```
private void TestForm_Load(object sender, EventArgs e)
{
    try
    {
        string DevenvPath = Environment.ExpandEnvironmentVariables(
            @"\%VSComnTools%devenv.exe").Replace(@"Tools", "IDE").Replace("\\", "");

        if (!File.Exists(DevenvPath)) //dùng cho visual studio 2008
        {
            DevenvPath = Environment.ExpandEnvironmentVariables(
                @"\%VS90ComnTools%devenv.exe").Replace(@"Tools", "IDE");
        }

        txtDotNETLocation.Text = DevenvPath;
    }
    catch
    {
    }

    btnStart.Focus();
}
```

- Phương thức “btnStart_Click” . Đoạn mã dưới là một sự kiện Click cho button có tên là “btnStart” (Kiểm Thủ) trong ứng dụng Windows Form. Khi button này được click, một chuỗi các hành động được thực hiện theo thứ tự như sau:

```
private void btnStart_Click(object sender, EventArgs e)
{
    //mở file dll hoặc file exe.
    GetAssemblyName();
    //tạo các thư mục để chứa các file .cs, .csproj..trong ổ C
    InitConstStrings();
    //Tạo thư mục kiểm thử
    TaoThuMucKiemThu();
    //

    GetTypesOfAssemblyUnderTest();
    //khởi tạo phương thức kiểm thử
    Khoitaophuongthuckiemthu();

    if (chckXMLDataDoc.Checked)
    {
        ThaoTacDulieuXML();
        btnCreateScript_Click(sender, e);
    }

    //btnCreateScript.Focus();
}
```

- Đoạn mã dưới là một sự kiện Click cho button có tên là “btnCreateScript” (Tạo Kịch Bản) trong ứng dụng Windows Form. Khi button này được click, các hành động sau sẽ được thực hiện:

```

private void btnCreateScript_Click(object sender, EventArgs e)
{
    btnStart.Focus();
    if (dirName == null || xApp == null)
    {
        MessageBox.Show("Click the Start button to collect test information first.");
        return;
    }
    //Create a TextWriter object to save the script
    TextWriter t = null;

    t = new StreamWriter(new FileStream(TestScriptCSFilename, FileMode.Create));
    //start generating script
    try
    {
        TaoMaKiemThu(DUTAsm, t);
    }
    catch (Exception err)
    {
        MessageBox.Show(err.Message);
    }
    finally
    {
        t.Close();
    }

    DongExcelSheet();
    ThaoTacCopyFile("ico");
    ThaoTacCopyFile("cs");
    TaoNguonThuThapDuLieu(xlsDataStoreFilename);
    txtDataStore.Text = xlsDataStoreFilename;
    BatDauDOTNETIDE(txtDotNETLocation.Text, CSProjFilename);
    copyfile();
    //copydll();
    string nguon = @"D:/thao" + "/" + dirName + "/" + dirName + "/Bin/Debug";
    string targetPath = tempTestProjDir + "/" + dirName + "/Bin/Debug";
    //tatcafecopy(nguon, targetPath);
    Copytatca(nguon, targetPath);
}

```

- Phương thức btnExit_Click_1. Khi button "Exit" (Thoát) được click trong ứng dụng Windows Form, phương thức này sẽ gọi phương thức Exit() của lớp Application để đóng ứng dụng.

```

private void btnExit_Click_1(object sender, EventArgs e)
{
    Application.Exit();
}

```

- Phương thức “btnAddDataStore”: Khi button "Add Data Store" (Thêm dữ liệu) trong ứng dụng Windows Form được click, phương thức này kiểm tra xem biến “dirName” có giá trị null không. Nếu có, nó hiển thị một hộp thoại cảnh báo yêu cầu người dùng click vào nút "Start", sau đó nút "Create Script" để tạo một kịch bản kiểm thử trước.
- Phương thức “btnSaveDataStore”: Khi button "Save Data Store" (Ghi dữ liệu) trong ứng dụng Windows Form được click, phương thức này kiểm tra xem textbox “txtDataStore” có dữ liệu không. Nếu không có dữ liệu, phương thức sẽ kết thúc. Nếu có dữ liệu, nó sẽ gọi phương thức “TaoNguonThuThapDuLieu” và truyền nội dung của textbox “txtDataStore” vào đó.

```

private void btnAddDataStore_Click(object sender, EventArgs e)
{
    if (dirName == null)
    {
        MessageBox.Show("Please click Start button, then Create Script button to generate a test script first.");
        return;
    }
    openFileDialog1.Title = "Add more data store";
    openFileDialog1.Filter = "Excel Files (*.xls)|*.xls";
    openFileDialog1.Multiselect = true;
    if (openFileDialog1.ShowDialog() == DialogResult.OK)
    {
        foreach (string fn in openFileDialog1.FileNames)
        {
            txtDataStore.Text += "\n" + fn;
        }
    }
}

1 reference
private void btnSaveDataStore_Click(object sender, EventArgs e)
{
    if (txtDataStore.Text.Trim() == "")
        return;
    TaoNguonThuThapDuLieu(txtDataStore.Text);
}

```

Các kiểu kiểm thử

- Phương thức “btnOK_Click”: Được gọi khi nút "OK" được nhấn. Nó thiết lập giá trị của biến “m_typeState” thành “DialogResult.OK” và ẩn form hiện tại.
- Phương thức “btnCancel_Click”: Được gọi khi nút "Cancel" được nhấn. Nó thiết lập giá trị của biến “m_typeState” thành “DialogResult.Cancel” và ẩn form hiện tại.
- Khi sự kiện "CheckedChanged" của checkbox xảy ra, phương thức này sẽ được gọi. Nó kiểm tra xem checkbox có được chọn hay không. Nếu checkbox được chọn, vòng lặp sẽ duyệt qua tất cả các mục trong “chckListType” và đặt chúng thành đã được chọn. Ngược lại, nếu checkbox không được chọn, vòng lặp sẽ duyệt qua tất cả các mục trong “chckListType” và đặt chúng thành không được chọn.

```

private void btnOK_Click(object sender, EventArgs e)
{
    m_typeState = DialogResult.OK;
    this.Hide();
}

1 reference
private void btnCancel_Click(object sender, EventArgs e)
{
    m_typeState = DialogResult.Cancel;
    this.Hide();
}

1 reference
private void checkBox1_CheckedChanged(object sender, EventArgs e)
{
    if (checkBox1.Checked)
    {
        for (int i = 0; i < chkListType.Items.Count; i++)
            chkListType.SetItemChecked(i, true);
    }
    else
    {
        for (int i = 0; i < chkListType.Items.Count; i++)
            chkListType.SetItemChecked(i, false);
    }
}

```

- Form AutoManualStubbing

- Phương thức “btnAddConstructor_Click” là một sự kiện được gọi khi nút “btnAddConstructor” được nhấn trong một ứng dụng Windows Form.

```

private void btnAddConstructor_Click(object sender, EventArgs e)
{
    try
    {
        if (txtConstructor.Text != "")
        {
            string reasignCstor = lstConstructors.SelectedItem.ToString();
            reasignCstor = reasignCstor.Substring(reasignCstor.IndexOf(" "));
            txtConstructor.Text += "\n\n" + reasignCstor;
        }
        else
            txtConstructor.Text = lstConstructors.SelectedItem.ToString();

        lstConstructors.Items.RemoveAt(lstConstructors.SelectedIndex);
    }
    catch (Exception err)
    {
        MessageBox.Show(err.Message + "\nSelecte a constructor to add.", "Error Reminder");
    }
}

```

- Phương thức btnRemoveConstructor_Click: Khi nút "btnRemoveConstructor" được nhấn, phương thức này sẽ thực hiện các bước sau:

Thêm nội dung đã chọn từ textbox "txtConstructor" vào danh sách "lstConstructors" sau khi loại bỏ dấu xuống dòng. Xóa nội dung đã chọn từ textbox "txtConstructor".

- Phương thức "btnAddMethod_Click": Khi nút "btnAddMethod" được nhấn, phương thức này sẽ thực hiện các bước sau:

Nếu nội dung của textbox txtMethod không rỗng, thêm nội dung đã chọn từ danh sách lstMethods vào textbox txtMethod, cách nhau bởi hai dấu xuống dòng.

Nếu nội dung của textbox txtMethod rỗng, thiết lập nội dung của textbox txtMethod bằng nội dung đã chọn từ danh sách lstMethods. Xóa phần tử đã chọn từ danh sách lstMethods.

```
private void btnRemoveConstructor_Click(object sender, EventArgs e)
{
    try
    {
        lstConstructors.Items.Add(txtConstructor.SelectedText.Replace("\n", ""));
        txtConstructor.Text = txtConstructor.Text.Replace(txtConstructor.SelectedText, "").Trim();
    }
    catch (Exception err)
    {
        MessageBox.Show(err.Message + "\nSelect a constructor to Remove.", "Error Reminder");
    }
}

1 reference
private void btnAddMethod_Click(object sender, EventArgs e)
{
    try
    {
        if (txtMethod.Text != "")
            txtMethod.Text += "\n\n" + lstMethods.SelectedItem.ToString();
        else
            txtMethod.Text = lstMethods.SelectedItem.ToString();
        lstMethods.Items.RemoveAt(lstMethods.SelectedIndex);
    }
    catch (Exception err)
    {
        MessageBox.Show(err.Message + "\nSelect a method to add.", "Error Reminder");
    }
}
```

- Phương thức "btnRemovemethod_Click": Khi nút "btnRemovemethod" được nhấn, phương thức này được kích hoạt. Nó cố gắng thêm văn bản từ textbox "txtMethod" vào danh sách "lstMethods" sau khi loại bỏ bất kỳ ký tự dòng mới nào.
- Phương thức "btnBrowser_Click": Phương thức này liên kết với nút "btnBrowser". Khi nút "btnBrowser" được nhấn, nó thiết lập một hộp thoại để cho phép người dùng chọn một tập tin DLL của kịch bản kiểm thử. Nếu người dùng chọn một tập tin và xác nhận việc chọn, đường dẫn của tệp đã chọn sẽ được đặt làm văn bản của textbox "txtDllToStub".

- Phương thức “btnOK_Click”: Phương thức này được kích hoạt khi nút "btnOK" được nhấn. Nó ẩn form hiện tại.

```

private void btnRemovemethod_Click(object sender, EventArgs e)
{
    try
    {
        lstMethods.Items.Add(txtMethod.SelectedText.Replace("\n", ""));
        txtMethod.Text = txtMethod.Text.Replace(txtMethod.SelectedText, "").Trim();
    }
    catch (Exception err)
    {
        MessageBox.Show(err.Message + "\nSelect a method to Remove.", "Error Reminder");
    }
}

1 reference
private void btnBrowser_Click(object sender, EventArgs e)
{
    openFileDialog1.Title = "Select a test script dll";
    openFileDialog1.Filter = "Script Dlls (*.dll)|*.dll|All Files(*.*)|*.*";
    openFileDialog1.Multiselect = false;

    if (openFileDialog1.ShowDialog() == DialogResult.OK)
    {
        txtDllToStub.Text = openFileDialog1.FileName;
    }
}

1 reference
private void btnOK_Click(object sender, EventArgs e)
{
    this.Hide();
}

```

ĐOẠN SAO

3.6. Thiết kế giao diện

- Thiết kế giao diện cho module kiểm thử chức năng cho một lớp.

Bảng 3.9. Danh sách chức năng của module.

STT	Tên chức năng	Tên form	Cách chọn từ chương trình
1	Kiểm thử chức năng	AutomatedSoftwareTest	Giao diện chính

- Chi tiết hóa các giao diện của module kiểm thử chức năng một lớp.
- Chức năng kiểm thử chức năng.
- Mục đích

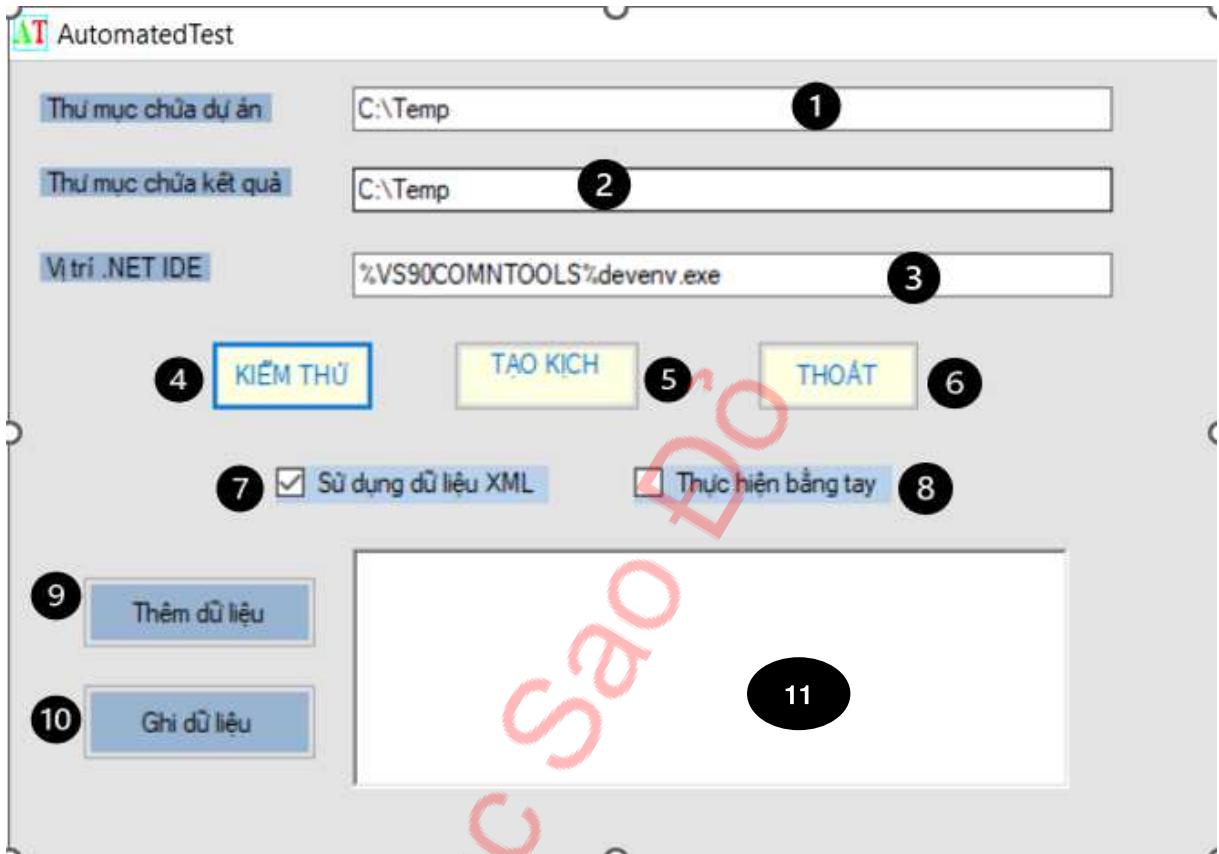
Kiểm thử về mặt chức năng của một lớp trong ứng dụng thực hiện chức năng đúng hay sai và ghi lại kết quả.

- Phạm vi: Kiểm thử về mặt chức năng của ứng dụng cần test.
- Ràng buộc

Dữ liệu đầu vào: Ứng dụng cần test có khả năng chạy mà không bị lỗi.

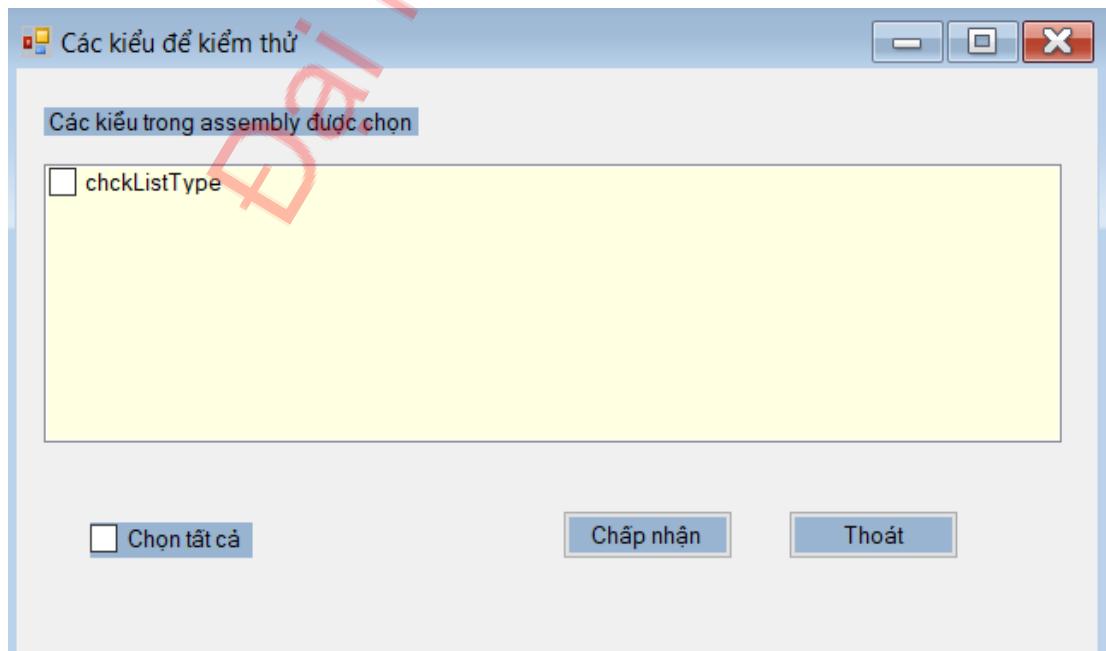
Dữ liệu đầu ra: Kết quả kiểm tra sau khi chạy chương trình được ghi ra file Excel gồm đầy đủ các thông tin về lớp (Phương thức, thuộc tính, đối số,...) và kết quả True/False tương ứng với các giá trị yêu cầu của test case.

- Giao diện chính chương trình



Hình 3.6. Giao diện chính chương trình.

- Giao diện các kiểu kiểm thử



Hình 3.7. Giao diện các kiểu kiểm thử

Bảng 3.10. Bảng đặc tả giao diện ứng dụng.

STT	Tên thành phần	Kiểu	Mô tả
1	txtTargetProj	Textbox	Ghi địa chỉ thư mục chứa dự án.
2	txtCurrDir	Textbox	Ghi địa chỉ thư mục chứa kết quả của dự án.
3	txtDotNETLocation	Textbox	Ghi địa chỉ file assembly của ứng dụng test.
4	btnStart	Button	Kích vào nút này để tạo tự động test case.
5	btnCreateScript	Button	Kích vào để tạo test script tự động.
6	btnExit	Button	Thoát chương trình.
7	chckXMLDataDoc	Checkbox	Sử dụng kiểm thử với chương trình là file XML.
8	chckManualStub	Checkbox	Cho phép kiểm thử bằng tay.
9	btnAddDataStore	Button	Cho phép kiểm thử nhiều chương trình.
10	btnSaveDataStore	Button	Ghi đường dẫn của từng chương trình test vào file text để phục vụ test nhiều chương trình.
11	txtDataStore	Listbox	Thêm file Excel để kiểm thử.

3.7. Kết luận chương 3

Trong chương này, ta đã lựa chọn công cụ để kiểm thử. Tìm hiểu về cách lưu trữ dữ liệu trên bảng tính Worksheet. Phân tích hệ thống yêu cầu chức năng và phi chức năng. Chương 3 đã trình bày ra thiết kế logic cung như thiết kế giao diện cho chương trình kiểm thử tự động.

KẾT LUẬN VÀ ĐÁNH GIÁ

1. Kết quả đạt được

Sau thời gian nghiên cứu và thực hiện đồ án dưới sự hướng dẫn trực tiếp của thầy Vũ Bảo Tạo, em đã đạt được những kết quả sau:

- Trình bày đầy đủ, chính xác các định nghĩa về kiểm thử phần mềm và kiểm thử tự động, trình bày chi tiết về các kỹ thuật Reflection và CodeDom của .NET.

- Từ kỹ thuật đã nghiên cứu và trình bày, đưa ra cách xây dựng một ứng dụng kiểm thử tự động dựa vào các kỹ thuật đã trình bày.

- Áp dụng lý thuyết xây dựng một ứng dụng kiểm thử tự động phần mềm về mặt chức năng của phần mềm đó.

- Kết quả nghiên cứu là tài liệu mang tính khoa học, là nguồn tham khảo cho sinh viên khi nghiên cứu về kiểm thử tự động và ứng dụng nó vào xây dựng công cụ kiểm thử.

2. Hạn chế

Tuy nhiên trong khi thực hiện, em đã hết sức cố gắng để hoàn thiện đồ án nhưng do thời gian nghiên cứu và thực hiện đồ án có hạn cũng như kiến thức chuyên môn chưa cao nên mới xây dựng ứng dụng nhỏ với chức năng còn hạn chế là kiểm thử về mặt chức năng của chương trình, chưa mở rộng kiểm thử được nhiều nội dung khác.

3. Hướng phát triển

Trong thời gian tới, em sẽ tiếp tục nghiên cứu sâu hơn về kiểm thử phần mềm, kiểm thử tự động và các kỹ thuật của .NET hỗ trợ tự động để xây dựng và hoàn thiện ứng dụng của mình.

TÀI LIỆU THAM KHẢO

- [1] Phạm Ngọc Hùng, Trương Anh Hoàng và Đặng Văn Hưng (năm 2014), *Giáo trình kiểm thử phần mềm*.
- [2] Đinh Thao (2023), *Các công cụ kiểm thử phần mềm phổ biến năm 2023.*
<http://diendan.congdongcviet.com/>.
- [3] Phương Mai (19/07/2014), *Xây dựng ứng dụng kiểm thử tự động*
<https://sharecode.vn/>.
- [4] Kanglin Li, Mengqi Wu published by John Wiley & Sons (2004), *Effective Software*.
- [5] Test Automation: Developing an Automated Software Testing Tool.
- [6] Glenford J. Myers Wu published by John Wiley & Sons (2004), *The Art of Software Testing*.

Đại học Sao Đỏ