ĐẠI HỌC QUỐC GIA HÀ NỘI

TRƯỜNG ĐẠI HỌC CÔNG NGHỆ

TRẦN THỊ HỒNG SIM

NGHIÊN CỨU MỘT SỐ PHƯƠNG PHÁP SINH ĐẦU VÀO KIỂM THỬ TỰ ĐỘNG CHO ANDROID

LUẬN VĂN THẠC SĨ CÔNG NGHỆ THÔNG TIN

Hà Nội – 2017

ĐẠI HỌC QUỐC GIA HÀ NỘI

TRƯỜNG ĐẠI HỌC CÔNG NGHỆ

TRẦN THỊ HỒNG SIM

NGHIÊN CỨU MỘT SỐ PHƯƠNG PHÁP SINH ĐẦU VÀO KIỂM THỬ TỰ ĐỘNG CHO ANDROID

Ngành: Công Nghệ Thông Tin

Chuyên ngành: Kỹ thuật Phần mềm

Mã số: 60480103

LUẬN VĂN THẠC SĨ CÔNG NGHỆ THÔNG TIN

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

Hà Nội – 2017

Lời cam đoan

Tôi xin cam đoan các nội dung trong luận văn với đề tài "Nghiên cứu một số phương pháp sinh đầu vào kiểm thử tự động cho Android" là công trình nghiên cứu của bản thân, xuất phát từ những yêu cầu phát sinh trong công việc để hình thành ra hướng nghiên cứu. Các số liệu có nguồn gốc rõ ràng và tuân thủ đúng nguyên tắc, kết quả thực nghiệm trình bày trong luận văn được thu thập được trong quá trình nghiên cứu là trung thực, chưa từng được công bố trước đây.

Hà Nội, Ngày 12 tháng 12 năm 2017

Tác giả luận văn

Trần Thị Hồng Sim

Lời cảm ơn

Đầu tiên, em xin gửi lời cảm ơn chân thành và biết ơn sâu sắc tới PGS.TS Trương Anh Hoàng, giảng viên bộ môn Kỹ thuật Phần mềm, khoa Công Nghệ Thông Tin, trường Đại học Công Nghệ, Đại học Quốc Gia Hà Nội. Trong suốt quá trình học tập và thực hiện luận văn này, thầy đã là người trực tiếp hướng dẫn và đưa ra những định hướng quý báu cho quá trình nghiên cứu. Chính nhờ sự nhiệt tình chỉ bảo, dành thời gian quý báu của thầy trong suốt quá trình hướng dẫn mà em đã hoàn thành việc nghiên cứu.

Em cũng xin gửi lời cảm ơn chân thành đến các thầy giáo, cô giáo là giảng viên trường Đại học Công Nghệ đã giảng dạy, truyền đạt kiến thức cho em trong hơn hai năm học tại trường. Những kiến thức mà các thầy cô đã truyền thụ là nền tảng cho em trong công việc sau này và là những kiến thức tiên quyết trong việc nghiên cứu và tìm hiểu đề tài trong luận văn.

Và cuối cùng, tôi xin gửi lời cảm ơn đến bạn bè, đồng nghiệp và đặc biệt là gia đình, những người đã luôn ở bên động viên, giúp đỡ, tạo điều kiện tốt nhất cho tôi trong suốt quá trình học tập và thực hiện luận văn.

Hà Nội, tháng 12/2017

Trần Thị Hồng Sim

Mục lục

Đặt vấn đề

Như chúng ta đã biết nhu cầu sử dụng các thiết bị di động thông minh của con người ngày càng cao, số lượng các nhà sản xuất thiết bị cũng ngày càng nhiều và đa dạng hơn về chủng loại, mẫu mã. Mỗi chiếc điện thoại thông minh ngày nay không đơn thuần chỉ để nghe, gọi và nhắn tin như trước, mà nó giống như một máy tính để bàn thu nhỏ, chúng ta có thể lướt web, chat wifi, mua hàng trực tuyến, tìm kiếm thông tin, xử lý thông tin mạng, kết nối thiết bị ngoại vi, điều khiển ô tô, điều khiển robot, giải quyết công việc với đối tác ở bất kì nơi đâu và vô vàn những lợi ích lớn lao khác.

Để các thiết bị di động có được sức mạnh như vậy trước hết là nhờ các công ty phát triển phần mềm mà cụ thể ở đây là Google với Android, Apple với iOS và Microsoft với Windows Phone. Các công ty này đã tập trung lớn nguồn lực của họ vào việc phát triển các nền tảng di động kể trên để đưa chúng lên một tầm cao hơn trước đây, mà ngày nay chúng ta thường hay gọi là "HỆ ĐIỀU HÀNH".

Trong số các hệ điều hành cho các thiết bị di động, Android hiện đang là hệ điều hành phổ biến và lớn mạnh nhất. Với tính chất nguồn mở, Android thu hút nhiều nhà sản xuất thiết bị trên thế giới như Sony, Samsung, LG, HTC, v.v… Ngày nay Android không chỉ được sử dụng là hệ điều hành trên điện thoại thông minh và máy tính bảng, mà còn được ứng dụng vào các dự án khác như: đồng hồ thông minh (smartwatch), nhà thông minh (smart home), tivi thông minh (smart tivi), robot, điều khiển ô tô, kính thực thể ảo …

Theo số liệu thống kê, Android hiện đang nắm giữ 86% [1] tổng thị phần cho hệ điều hành di động. Sự phổ biến ngày càng tăng của các thiết bị Android có tác động trực tiếp đến cửa hàng ứng dụng Google Play. Đây là cửa hàng ứng dụng lớn nhất thế giới và có 3.3 triệu ứng dụng (tháng 9/2017) có sẵn để tải xuống. Chỉ riêng trong quý 2 năm 2017, đã có gần 17 tỷ lượt tải xuống các ứng dụng từ Google Play. Với những con số thống kê như trên, có thể thấy việc xây dựng các ứng dụng cho thiết bị Android đã, đang và sẽ vẫn là một xu hướng phát triển mạnh mẽ và bền vững.

Trong vòng đời phát triển phần mềm, kiểm thử là một hoạt động quan trọng và không thể bỏ qua đối với phần mềm nói chung và các ứng dụng Android nói riêng.

Hoạt động kiểm thử có thể tiến hành một cách thủ công tuy nhiên điều này sẽ mất thời gian, tốn chi phí và đôi khi không mang lại hiệu quả cao. Thậm chí trong một vài những phương pháp kiểm thử, hoạt động kiểm thử thủ công là không thể thực hiện được. Do đó đòi hỏi phải có một hình thức kiểm thử tự động hỗ trợ. Tuy nhiên kiểm thử tự động cũng có nhiều kỹ thuật khác nhau với các mức độ tự động khác nhau. Đối với nhiều các công cụ kiểm thử, vẫn cần có sự tham gia của kiểm thử viên vào trong quá trình. Kiểm thử viên sẽ phải xây dựng các kịch bản kiểm thử hoàn toàn thủ công để các công cụ kiểm thử có thể thực thi được từ các kịch bản đó. Đây là một công việc không hề đơn giản, tốn thời gian và nhân lực. Vậy một câu hỏi được đặt ra, làm sao để hoạt động kiểm thử hoàn toàn tự động, từ thao tác sinh ra các kịch bản kiểm thử cho đến việc thực thi những kịch bản kiểm thử đó. Đã có rất nhiều các nghiên cứu về các kỹ thuật sinh dữ liệu kiểm thử tự động. Và trong nội dung luận văn này cũng sẽ tìm hiểu về các kỹ thuật sinh dữ liệu kiểm thử tự động, và cụ thể nó được áp dụng vào quá trình kiểm tra tự động cho các ứng dụng Android ra sao.

Cụ thể luận văn được xây dựng bao gồm 4 chương với chi tiết như sau:

- Chương 1: Trình bày tổng quan về hệ điều hành Android bao gồm các tầng trong Android và cấu trúc tập tin Manifest là tập tin kê khai những thông tin thiết yếu về ứng dụng với hệ thống
- Chương 2: đi sâu vào tìm hiểu hai phương pháp sinh đầu vào kiểm thử tự động là phương pháp kiểm thử Fuzz (Fuzzing) và phương pháp kiểm thử dựa trên mô hình (model-based testing)
- Chương 3: tìm hiểu hai công cụ kiểm thử tự động cho Android đại diện cho hai phương pháp kiểm thử Fuzz và kiểm thử dựa trên mô hình là Monkey và DroidBot.
- Chương 4: tiến hành nghiên cứu thực nghiệm bằng cách sử dụng hai công cụ Monkey và DroidBot để kiểm tra cho một danh sách các ứng dụng Android, đồng thời đo lại các kết quả về số lượng lỗi tìm được, độ bao phủ mã nguồn, từ đó đưa ra những phân tích và đánh giá cho kết quả thực nghiệm đạt được

Cuối cùng là kết luận và tài liệu tham khảo

Chương 1. Nền tảng Android

1.1. Giới thiệu chung về Android

Android là hệ điều hành mã nguồn mở, dựa trên Linux, được tạo ra cho một loạt các thiết bị và yếu tố hình thức. Sơ đồ ở hình 1.1 cho biết các thành phần chính của nền tảng Android [2]:

- *Tầng hạt nhân Linux*: nền tảng của hệ điều hành Android là hạt nhân Linux. Tất cả mọi hoạt động của thiết bị đều phải thực thi ở tầng này. Tầng này bao gồm các tiến trình quản lý bộ nhớ (memory management), giao tiếp với phần cứng (driver model), bảo mật (security), quản lý tiến trình (process). Sử dụng hạt nhân Linux cho phép Android tận dụng các tính năng bảo mật then chốt và cho phép các nhà sản xuất thiết bị phát triển trình điều khiển phần cứng cho hạt nhân nổi tiếng này
- *Lớp trừu tượng phần cứng* (Hardware Abstraction Layer - HAL): cung cấp các giao diện chuẩn để phần cứng thiết bị có thể giao tiếp với các nền tảng Java API ở cấp cao hơn. Lớp trừu tượng phần cứng này bao gồm nhiều mô đun thư viện, mỗi mô đun này lại thực thi một giao diện cho một loại thành phần phần cứng cụ thể, chẳng hạn như là mô đun máy ảnh hoặc mô đun Bluetooth. Khi bộ khung API (API framework) thực hiện cuộc gọi để truy cập phần cứng thiết bị, hệ thống Android sẽ tải mô đun thư viện cho thành phần phần cứng đó.
- *Thời gian chạy Android* (Android Runtime – ART): đối với các thiết bị chạy Android phiên bản 5.0 (API 21) trở lên, mỗi ứng dụng chạy trong tiến trình của chính nó với thể hiện chính nó của thời gian chạy Android. Thời gian chạy Android được viết để chạy nhiều máy ảo trên các thiết bị có bộ nhớ thấp bằng cách thực thi các tập DEX, một định dạng byte-code được thiết kế đặc biệt cho Android, được tối ưu hóa cho bộ nhớ tối thiểu. Xây dựng các công cụ, chẳng hạn như Jack, biên soạn các nguồn Java vào mã DEX byte-code, có thể chạy trên nền tảng Android. Một số tính năng chính của thời gian chạy Android bao gồm:

- Biên dịch trước thời hạn (Ahead-Of-Time - AOT) và trong thời hạn (Just-In-Time - JIT)
- Tối ưu hóa thu gom rác (Garbage Collection - GC)

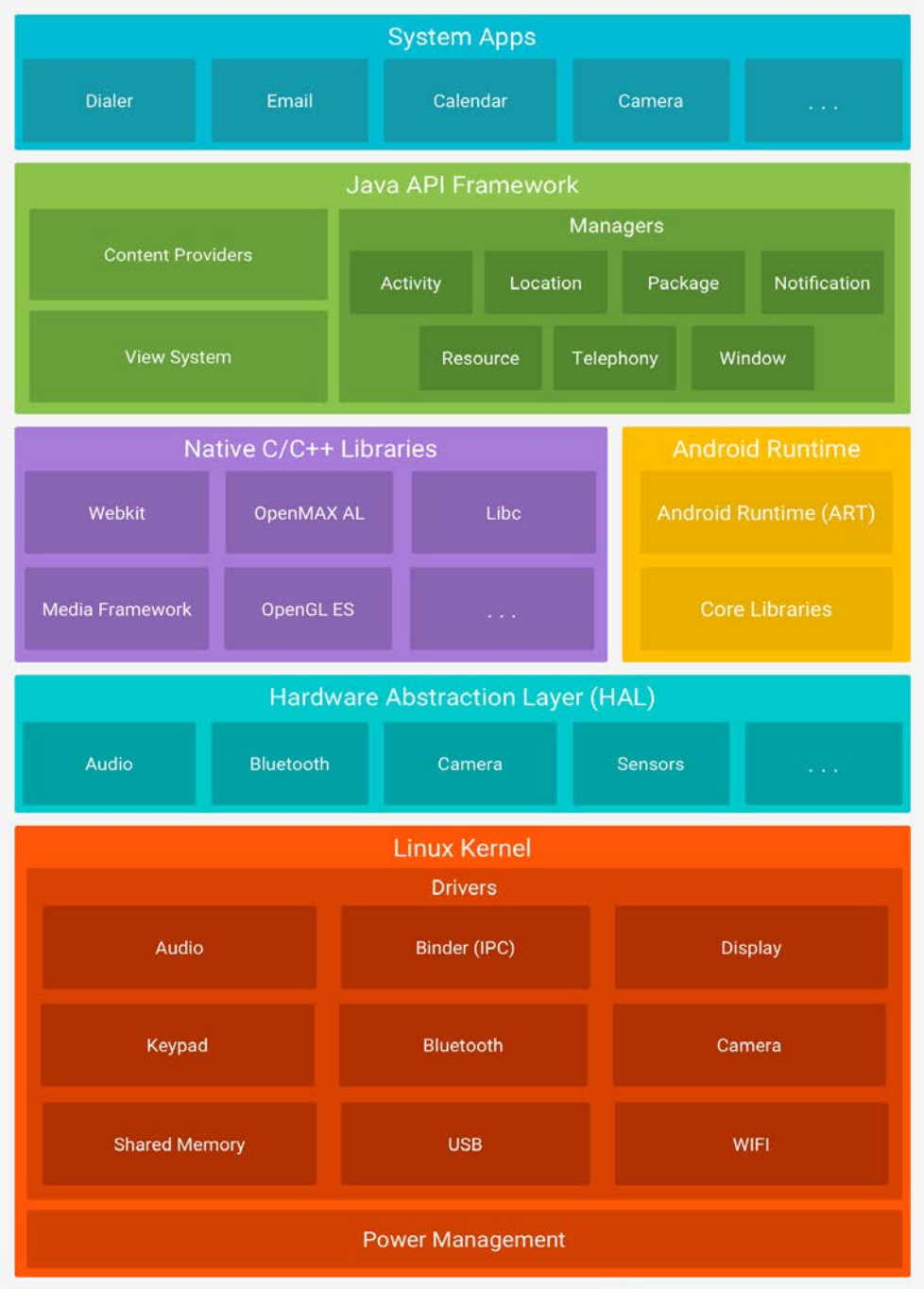
- Hỗ trợ gỡ lỗi tốt hơn, bao gồm một hồ sơ mẫu riêng, các ngoại lệ chuẩn đoán chi tiết, báo cáo sự cố và khả năng thiết lập các điểm quan sát để giám sát các lĩnh vực cụ thể.

Trước phiên bản Android 5.0 (Cấp độ API 21), Dalvik chính là thời gian chạy Android. Nếu ứng dụng người dùng chạy tốt trên ART, thì cũng có thể làm việc trên Dalvik, nhưng ngược lại có thể không đúng.

Android cũng bao gồm một tập hợp các thư viện chạy lõi cung cấp hầu hết các chức năng của ngôn ngữ lập trình Java, bao gồm một số tính năng ngôn ngữ Java 8, mà khuôn khổ Java API sử dụng

- *Tầng thư viện C/C++*: nhiều thành phần và dịch vụ cốt lõi của hệ thống, như là HAL và ART, được xây dựng từ mã nguồn cục bộ yêu cầu các thư viện gốc được viết bằng C và C++. Nền tảng Android cung cấp các API của bộ khung Java để phô bày tính năng của một số thư viện gốc này cho các ứng dụng. Ví dụ, có thể truy cập vào OpenGL ES thông qua các Java OpenGL API của bộ khung Android để thêm hỗ trợ vẽ và thao tác đồ họa 2D và 3D trong ứng dụng. Nếu ứng dụng phát triển yêu cầu mã nguồn C hoặc C ++, ta có thể sử dụng Android NDK để truy cập vào một số thư viện nền tảng gốc trực tiếp từ mã nguồn gốc.
- Tầng khung Java API: toàn bộ tập hợp các tính năng của hệ điều hành Android đã có sẵn thông qua các API được viết bằng ngôn ngữ Java. Các API này tạo thành những khối xây dựng sẵn mà chúng ta hoàn toàn có thể lấy ra sử dụng khi muốn xây dựng các ứng dụng Android. Các API này giúp đơn giản hóa việc tái sử dụng phần lõi Android cũng như là các thành phần và dịch vụ của hệ thống mô đun sau:

- View System phong phú và có khả năng mở rộng, người dùng có thể sử dụng để xây dựng UI của ứng dụng, bao gồm các danh sách, lưới, hộp văn bản, các nút bấm, và thậm chí một trình duyệt web nhúng
- Trình quản lý tài nguyên, cung cấp quyền truy cập vào các tài nguyên không mã nguồn như là chuỗi cục bộ, đồ họa, các tệp bố cục.
- Trình quản lý thông báo: cho phép tất cả các ứng dụng hiển thị cảnh báo tùy chỉnh trong thanh trạng thái
- Trình quản lý hoạt động: quản lý vòng đời của ứng dụng và cung cấp ngăn xếp trở lại chuyển hướng thông dụng.



[1]

Hình 1.1: Cấu trúc ngăn xếp phần mềm Android

- Trình cung cấp nội dung: cho phép ứng dụng truy cập dữ liệu từ các ứng dụng khác, chẳng hạn như ứng dụng danh bạ hoặc để chia sẻ dữ liệu riêng của chúng.

Nhà phát triển có quyền truy cập đầy đủ vào cùng bộ khung API mà các ứng dụng hệ thống Android sử dụng.

- *Tầng ứng dụng hệ thống*: Android đi kèm với một bộ các ứng dụng cốt lõi như email, tin nhắn SMS, lịch, trình duyệt internet, danh bạ và hơn thế nữa. Ứng dụng đi

[1] https://developer.android.com/guide/platform/index.html

kèm với nền tảng này không có trạng thái đặc biệt so với các ứng dụng mà người dùng tự cài đặt. Vì vậy, ứng dụng của bên thứ ba vẫn có thể trở thành trình duyệt web mặc định, tin nhắn SMS mặc định, hoặc thậm chí bàn phím mặc định của người dùng.

Các ứng dụng hệ thống hoạt động như một ứng dụng cho người dùng và đồng thời cung cấp các khả cơ bản mà nhà phát triển có thể truy cập từ ứng dụng của riêng họ. Ví dụ, nếu một ứng dụng của người dùng muốn gửi tin nhắn SMS, người dùng không cần phải tự xây dựng chức năng đó, thay vào đó có thể gọi ứng dụng SMS nào đó đã được cài đặt để gửi tin nhắn tới người nhận chỉ định.

1.2. Bản kê khai ứng dụng AndroidManifest

Mọi ứng dụng Android đều phải có một tệp AndroidManifest.xml trong thư mục gốc của mình. Tệp kê khai này trình bày những thông tin thiết yếu về ứng dụng với hệ thống, thông tin mà hệ thống phải có trước khi có thể chạy bất kỳ mã nào của ứng dụng. Các giá trị trong tập tin Manifest bị ràng buộc vào ứng dụng tại thời điểm biên dịch và không thể thay đổi sau đó, trừ khi thực hiện biên dịch lại ứng dụng. Một trong các loại thông tin chính mà tập tin Manifest này chứa là chúng khai báo các thành phần của ứng dụng. Các thành phần này có thể là một hoặc nhiều trong các loại sau:

- Hoạt động (Activity)
- Dịch vụ (Service)
- Bộ nhận quảng bá (Broadcast Receiver)
- Trình cung cấp nội dung (Content Provider)

1.2.1. Hoạt động (activity) [3]

Khái niệm

Hoạt động là thành phần phụ trách giao diện người dùng của ứng dụng. Một hoạt động là một giao diện trực quan mà người dùng có thể thực hiện trên đó mỗi khi được kích hoạt. Một ứng dụng có thể có nhiều hoạt động và chúng có thể gọi đến nhau hoặc chuyển giữa các hoạt động với nhau. Mỗi hoạt động là một dẫn xuất của lớp android.app.Activity.

Mỗi hoạt động có một cửa sổ để vẽ lên. Thông thường cửa sổ này phủ đầy màn hình, ngoài ra nó cũng có thể có thêm các cửa sổ con khác như là hộp thoại. Nội dung

cửa sổ của hoạt động được cung cấp bởi một hệ thống cấp bậc các Khung nhìn (là đối tượng của lớp Views).

Vòng đời của hoạt động

Các hoạt động trong hệ thống được quản lý bởi một cấu trúc dữ liệu ngăn xếp. Khi có một hoạt động được khởi tạo, nó được đẩy vào trong ngăn xếp, chuyển sang trạng thái thực thi và hoạt trộng trước đó sẽ chuyển sang trạng thái chờ. Hoạt động này chỉ trở lại trạng thái kích hoạt khi mà hoạt động vừa khởi tạo kết thúc việc thực thi.

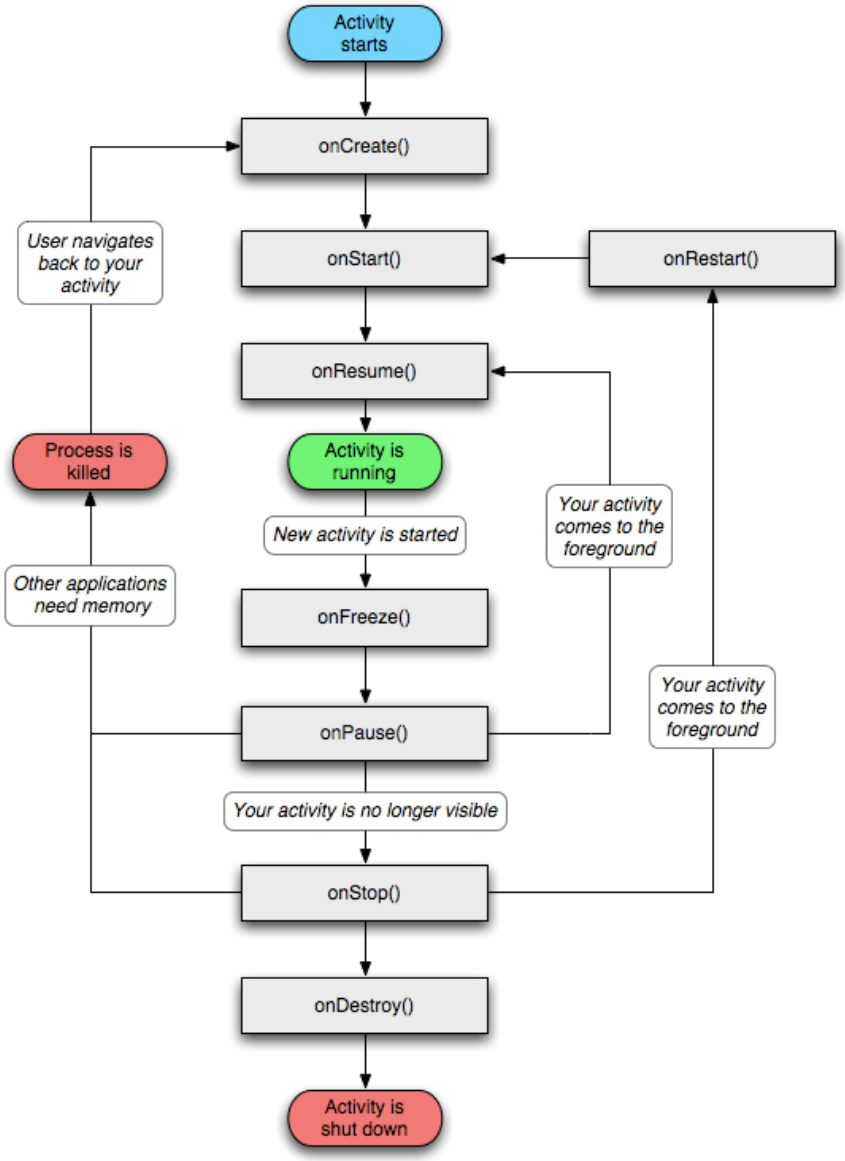
Một hoạt động có bốn trạng thái chính là:

- *Hoạt động* hoặc *đang chạy:* khi nó đang hiển thị ở trên màn hình và nhận tương tác người dùng
- *Tạm dừng:* khi hoạt động không còn là trọng tâm trên màn hình nhưng vẫn hiển thị trước người dùng.
- *Dừng:* Khi một hoạt động hoàn toàn bị che khuất, nó sẽ rơi vào trạng thái *Dừng*. Tuy nhiên, nó vẫn còn lưu trữ toàn bộ thông tin trạng thái. Và nó thường bị hệ thống đóng lại khi có tình trạng thiếu bộ nhớ.

Khi chuyển giữa các trạng thái, ứng dụng sẽ gọi các hàm gọi lại ứng với các bước chuyển:

```
void onCreate(Bundle savedInstanceState)
void onStart()
void onRestart()
void onResume()
void onPause()
void onStop()
void onDestroy()
```

Biểu đồ sau mô tả trạng thái trong vòng đời của một hoạt động. Hình chữ nhật thể hiện các phương thức gọi lại mà chúng ta có thể khai báo để gọi thực thi một số thao tác khi hoạt động chuyển sang trạng thái khác (phương thức gọi lại là phương thức được gọi lại bởi một phương thức khác khi có một sự kiện xảy ra). Các trạng thái chính của một hoạt động được thể hiện bởi các hình viên thuốc:



[2]

Hình 1.2: Vòng đời của một hoạt động (activity) trong ứng dụng Android

Vòng đời của một hoạt động được thể hiện trong những quá trình sau:

Toàn bộ vòng đời của một hoạt động bắt đầu từ lời gọi đầu tiên tới phương thức onCreate (Bundle) cho tới khi có lời gọi phương thức onDestroy(). Trong quá trình này, một hoạt động sẽ khởi tạo lại tất cả các tài nguyên cần sử dụng trong phương thức onCreate() và giải phóng chúng khi phương thức onDestroy() được thực thi.

Vòng đời có thể nhìn thấy của một hoạt động bắt đầu từ lời gọi tới phương thức onStart(), cho tới khi phương thức onStop() của nó được thực thi. Toàn bộ các tài nguyên đang được sử dụng bởi hoạt động vẫn tiếp tục được lưu giữ, người dùng có thể thấy

[2] https://developer.android.com/reference/android/app/Activity.html

giao diện nhưng không tương tác được với hoạt động do trong quá trình này hoạt động không ở trạng thái chạy tiền cảnh.

Thời gian sống tiền cảnh của một hoạt động là quá trình bắt đầu từ khi có lời gọi tới phương thức onResume()và kết thúc bằng lời gọi tới phương thức onPause(). Trong thời gian này, hoạt động chạy ở tiền cảnh và có thể tương tác với người dùng.

1.2.2. Dịch vụ (service) [4]

Khái niệm

Một dịch vụ (service) là các đoạn mã được thực thi ngầm bởi hệ thống mà người sử dụng không thấy được. Mỗi dịch vụ đều được mở rộng từ lớp cơ sở là dịch vụ trong gói android.app, có thể kết nối tới hoặc kích hoạt một dịch vụ thông qua giao diện mà dịch vụ đưa ra. Ví dụ như một chương trình chơi nhạc, sẽ có vài hoạt động cho phép người dùng duyệt danh sách các bài hát và lựa chọn bài nào đó để phát. Tuy nhiên, chức năng chơi nhạc không được thiết kế như một hoạt động bởi chúng ta sẽ muốn chuyển qua cửa sổ khác, như khi soạn tin nhắn thì bài nhạc vẫn tiếp tục được chơi. Trong trường hợp này, ứng dụng chơi nhạc sẽ khởi tạo một dịch vụ bằng cách sử dụng phương thức Context.startService().

Một ứng dụng có thể dễ dàng thực hiện liên kết tới một dịch vụ đang chạy (thậm chí khởi động nếu nó chưa thực thi) bằng phương thức Context.bindService(). Khi đó dịch vụ này sẽ cung cấp cho ứng dụng cơ chế để giao tiếp với chúng thông qua một giao diện được gọi là IBinder (đối với dịch vụ chơi nhạc có thể cho phép dừng hoặc chuyển qua bài nhạc kế tiếp).

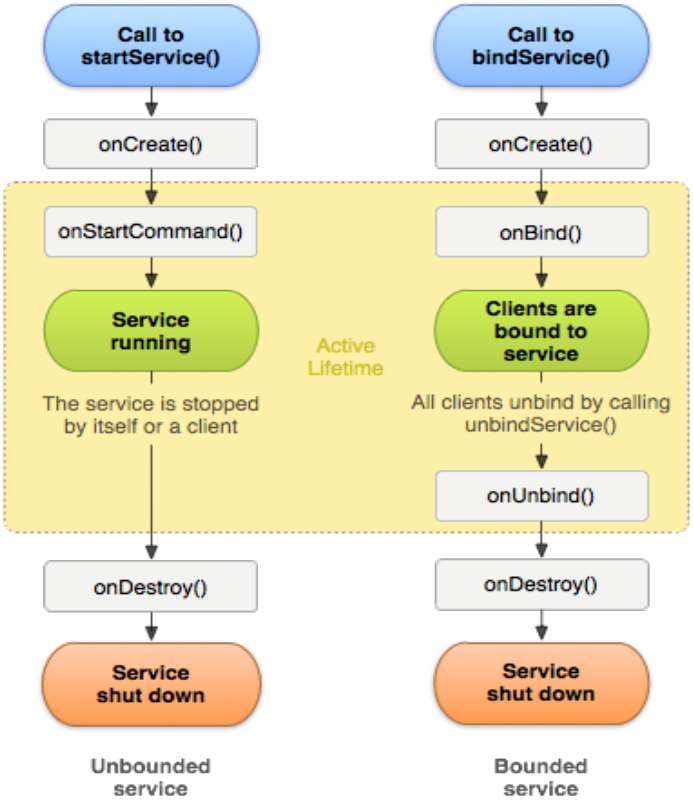
Vòng đời của một dịch vụ

Vòng đời của một dịch vụ được hiểu là quá trình hoạt động từ khi nó được tạo ra cho tới khi bị loại khỏi hệ thống. Có hai cách thức để một dịch vụ có thể được chạy trong hệ thống:

Khi hệ thống có lời gọi tới phương thức Context.startService(). Trong trường hợp này, dịch vụ sẽ được thực hiện liên tục cho tới khi hệ thống gọi phương thức Context.stopService().

Khi các ứng dụng gọi phương thức Context.bindService() để tạo kết nối với dịch vụ (dịch vụ sẽ được khởi tạo nếu tại thời điểm đó nó đang không hoạt động). Ứng dụng sẽ

nhận được một đối tượng IBinder do dịch vụ trả lại để có thể gọi các phương thức Callback phù hợp để truy cập tới các trạng thái của dịch vụ. Nếu do lời gọi Context.bindService() mà dịch vụ được khởi tạo thì nó sẽ được thực thi cho tới khi nào kết nối trên (tức là đối tượng IBinder) vẫn còn tồn tại.



[3]

Hình 1.3: Vòng đời của một dịch vụ trong ứng dụng Android

1.2.3. Bộ nhận quảng bá (Broadcast Receiver) [5]

Khái niệm

Bộ nhận quảng bá là một thành phần không làm gì cả nhưng nó nhận và phản hồi lại các thông báo quảng bá. Nhiều quảng bá có nguồn gốc từ mã hệ thống, ví dụ thông báo thay đổi múi giờ, pin yếu, ảnh đã chụp hay thay đổi ngôn ngữ. Các ứng dụng có thể khởi động quảng bá, ví dụ để các ứng dụng khác biết rằng dữ liệu đã được tải về xong trên thiết bị và sẵn sàng sử dụng.

Một ứng dụng có thể có số lượng bộ nhận quảng bá bất kỳ để nhận những thông báo quan trọng với nó. Tất cả các bộ nhận quảng bá được kế thừa từ lớp BroadcastReceiver.

[3] https://developer.android.com/guide/components/services.html

Bộ nhận quảng bá không có giao diện. Tuy nhiên, chúng có thể khởi động một hoạt động để đáp lại thông tin mà nó nhận được, hay chúng có thể sử dụng bộ quản lý thông báo để thông báo người dùng biết. Các thông báo có thể được sự chú ý của người dùng theo các cách khác nhau như là sáng màn hình, rung thiết bị, bật âm thanh nào đấy, … Thông thường, chúng đặt thông báo trên thanh trạng thái, nơi người dùng có thể nhận được thông báo.

1.2.4. Trình cung cấp nội dung (Content Provider) [6]

Khái niệm

Các ứng dụng có thể lưu trữ dữ liệu của mình trong các tập tin hoặc sử dụng cơ sở dữ liệu SQLite sẵn có v.v… Trình cung cấp nội dung có chức năng cung cấp một tập hợp các phương thức cho phép một ứng dụng có thể lưu trữ và lấy dữ liệu được quản lý bởi trình cung cấp nội dung đó.

Trình cung cấp nội dung là một đặc trưng riêng của Android, nhờ đó mà các ứng dụng có thể chia sẻ dữ liệu với nhau một cách dễ dàng. Giống với tất cả các phần mềm, hành vi của ứng dụng có thể phụ thuộc nhiều vào trạng thái của các trình cung cấp nội dung (ví dụ như danh sách liên lạc có thể trống hoặc chứa các dữ liệu trùng nhau). Do đó, các công cụ kiểm thử cần phải "giả mạo" các trình cung cấp nội dung nhằm mục đích tạo ra các ca kiểm thử mong muốn và đạt được độ bao phủ cao hơn trong hành vi của ứng dụng.

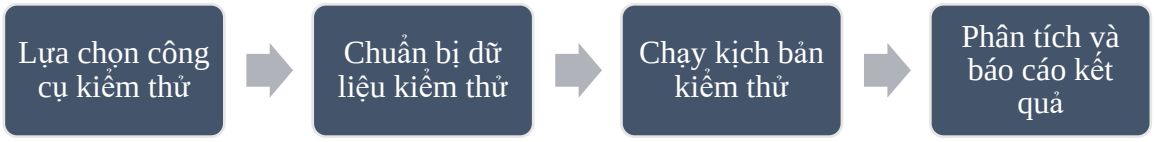
Chương 2. Sinh đầu vào kiểm thử tự động

Kiểm thử tự động từ lâu đã là một phần không thể thiếu trong hoạt động kiểm thử của phần mềm nói chung và các ứng dụng Android nói riêng. Giả sử khi một ứng dụng sau khi được sửa lỗi và phát hành, làm thế nào để đảm bảo rằng bản phát hành mới với các lỗi đã được sửa sẽ không gây ra bất kỳ lỗi mới nào trên các chức năng cũ. Vì vậy, sẽ tốt hơn là chúng ta cũng sẽ kiểm tra lại cả các chức năng cũ trên bản phát hành ứng dụng mới. Tuy nhiên sẽ rất khó để có thể kiểm tra lại bằng tay tất cả các chức năng của ứng dụng sau mỗi lần sửa lỗi hoặc bổ sung tính năng mới. Vì vậy mà kiểm thử tự động sẽ là một phần cần thiết trong hoạt động kiểm thử bởi nó giúp mang lại những hiệu quả về chi phí, về nhân lực và thời gian, v.v…

Một số các phương pháp kiểm thử tự động chính cho ứng dụng Android có thể được kể đến như sau [7]:

- Kiểm thử dựa trên mô hình (Model based)
- Ghi và phát lại (Record and Replay)
- Kiểm thử trên hệ thống (Systematic Testing)
- Kiểm thử mờ (Fuzzing)
- Kiểm thử ngẫu nhiên (Random Testing)
- Kiểm thử theo kịch bản (Scripted based Testing)

Các bước cơ bản trong một hoạt động kiểm thử tự động cho ứng dụng phần mềm có thể kể đến như sau:



Hình 2.1: Các bước cơ bản của kiểm thử tự động

Trong hoạt động kiểm thử tự động, hầu như chắc chắn trong tất cả các phương pháp, việc chạy kịch bản kiểm thử là hoàn toàn tự động. Tuy nhiên việc tạo ra dữ liệu kiểm thử, mỗi phương pháp lại có những cách thức khác nhau. Dữ liệu kiểm thử có thể được tạo ra một cách thủ công bởi kiểm thử viên, hoặc hoàn toàn tự động bởi các công cụ.

Sinh dữ liệu kiểm thử, một phần quan trọng của kiểm thử phần mềm, là quá trình tạo ra một bộ dữ liệu để kiểm tra tính đầy đủ của ứng dụng phần mềm. Đây có thể là dữ liệu thực tế được lấy từ các hoạt động kiểm thử ứng dụng trước đó hoặc là các dữ liệu nhân tạo được tạo ra có mục đích. Việc tạo ra các dữ liệu kiểm thử được xem là một vấn đề phức tạp, tốn nhiều thời gian và chi phí. Chính vì thế, việc tự động hóa cả việc sinh dữ liệu kiểm thử sẽ hoàn toàn tối ưu hoạt động kiểm thử tự động, làm tăng hiệu quả của kiểm thử tự động, giảm tải được thời gian và chi phí cho hoạt động sản xuất ứng dụng phần mềm. Trong nội dụng luận văn này, chúng ta sẽ cùng tập trung vào tìm hiểu về các phương pháp sinh dữ liệu kiểm thử tự động cho các ứng dụng Android.

Trước tiên, cùng làm quen với một số thuật ngữ chính được sử dụng cho nghiên cứu tạo dữ liệu kiểm thử tự động [8]:

Dữ liệu kiểm thử, dữ liệu đầu vào (Test data/ test input): là các dữ liệu đã được xác định cụ thể để sử dụng trong kiểm thử các ứng dụng phần mềm

Ca kiểm thử (Test case): là một tập hợp các điều kiện hoặc các biến mà dựa vào đó kiểm thử viên sẽ xác định ứng dụng hoặc hệ thống phần mềm có hoạt động chính xác hay không.

Bộ kiểm thử (Test suite): một tập hợp các ca kiểm thử được gọi là một bộ kiểm thử

Kế hoạch kiểm thử (Test plan): là một tài liệu chứa toàn bộ các thông tin về việc kiểm thử của tất cả các giai đoạn

Tự động kiểm thử (Test automation): là việc phát triển các phần mềm phục vụ cho hoạt động kiểm thử các ứng dụng phần mềm

Độ bao phủ (Coverage): độ bao phủ của phần mềm hoặc lỗi. Mục đích của phương pháp kiểm thử dựa trên độ bao phủ là để các ca kiểm thử có thể đảm bảo bao phủ phần mềm, đáp ứng một số các tiêu chí bao phủ nhất định. Mục đích của tiêu chí bao phủ lỗi là để bao phủ được tối đa số lỗi trong phần mềm.

Đường dẫn (Path): đường dẫn là một chuỗi các nút và cạnh. Nếu chúng ta bắt đầu từ một nút đầu vào và kết thúc tại một nút đầu ra thì chúng ta gọi đó là một đường dẫn hoàn chỉnh.

Vị ngữ nhánh (Branch predicate): là một điều kiện trong một nút mà có thể dẫn đến một đường dẫn đúng hoặc một đường dẫn sai.

Đường dẫn vị từ (Path predicate): một đường dẫn vị từ được định nghĩa là tập hợp của nhánh vị từ yêu cầu phải đúng để đi qua một đường dẫn

Đường dẫn khả thi (Feasible path): đường dẫn khả thi là một đường dẫn nơi mà có đầu vào hợp lệ thực hiện trong đường dẫn

Đường dẫn không khả thi (Infeasible path): đường dẫn không khả thi là đường dẫn mà không có đầu vào hợp lệ thực hiện trong đó

Ràng buộc (Constraint): một ràng buộc là một biểu thức xác định ngữ nghĩa của một phần tử và nó phải luôn luôn đúng

Bộ sinh ràng buộc (Constraint generator): là một chương trình có thể tự động tạo ra tập hợp các điều kiện hoặc ràng buộc trong một đường dẫn

Bộ giải quyết ràng buộc (Constraint solver): bộ giải quyết ràng buộc là một chương trình cung cấp giá trị cho các biến đầu vào của một vị từ đường dẫn mà nó đáp ứng tất cả các ràng buộc của vị từ đường dẫn tại một thời điểm.

Lập trình ràng buộc (Constraint programming): là một mô hình lập trình trong đó các mối quan hệ giữa các biến được biểu diễn dưới dạng ràng buộc.

Trong khuôn khổ của luận văn này với mục đích nghiên cứu về các phương pháp sinh đầu vào tự động, do đó trong nội dung của phần sau sẽ tập trung làm rõ hai phương pháp sinh đầu vào kiểm thử tự động nổi bật là phương pháp kiểm thử Fuzz (fuzzing) và phương pháp kiểm thử dựa trên mô hình (model based testing).

2.1. Phương pháp kiểm thử Fuzz (Fuzzing)

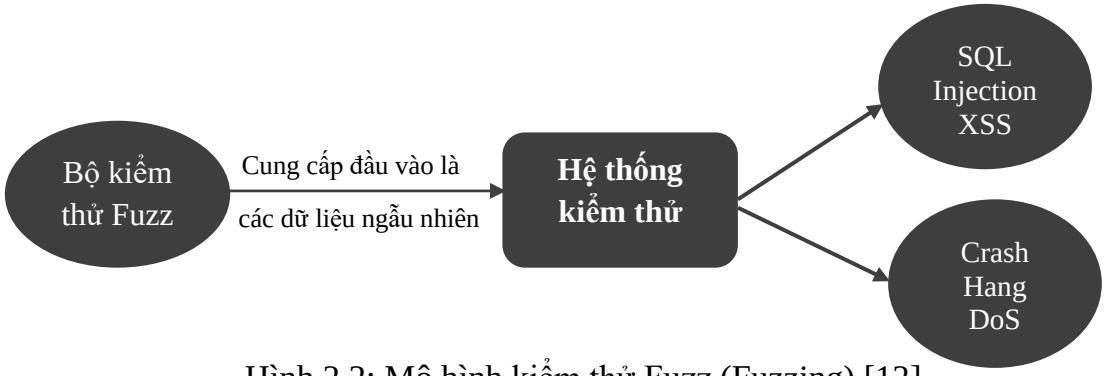
2.1.1. Kiểm thử Fuzz là gì?

Kiểm thử Fuzz [9] là một phương tiện có sẵn của kỹ thuật kiểm thử hộp đen, rất hiệu quả trong việc khám phá những sai lầm bảo mật quan trọng của sản phẩm mà các kỹ thuật kiểm tra thông thường không phát hiện ra được. Phương thức của kiểm thử Fuzz là cố ý nhập vào các dữ liệu ngẫu nhiên không hợp lệ để kích hoạt các điều kiện lỗi hoặc gây ra lỗi cho phần mềm

Do kiểm thử Fuzz là một kỹ thuật của kiểm thử hộp đen nên nó không đòi hỏi quyền truy cập vào mã nguồn, vì vậy nó có khả năng tìm thấy lỗi một cách nhanh chóng và tránh được việc phải xem mã nguồn.

Các chương trình và bộ khung được dùng để tạo ra kỹ thuật kiểm thử Fuzz hoặc thực hiện kiểm thử Fuzz được gọi là bộ kiểm thử Fuzz (Fuzzer) [10]. Tùy theo môi trường và ứng dụng cần kiểm tra mà người ta có các phương án khác nhau để xây dựng bộ kiểm thử Fuzz.

Kiểm thử Fuzz về cơ bản cũng giống như các kỹ thuật kiểm thử phần mềm khác [11], tuy nhiên nó được sử dụng để phát hiện ra một loạt các vấn đề như là lỗi mã hóa, lỗ hổng bảo mật giống như kịch bản hóa trang web chéo (Cross Site Scripting - XSS), tràn bộ đệm (Buffer Overflow), từ chối dịch vụ (DoS), chèn câu truy vấn (SQL Injection) như mô tả ở hình 2.2



Hình 2.2: Mô hình kiểm thử Fuzz (Fuzzing) [12]

2.1.2. Các giai đoạn của kiểm thử Fuzz

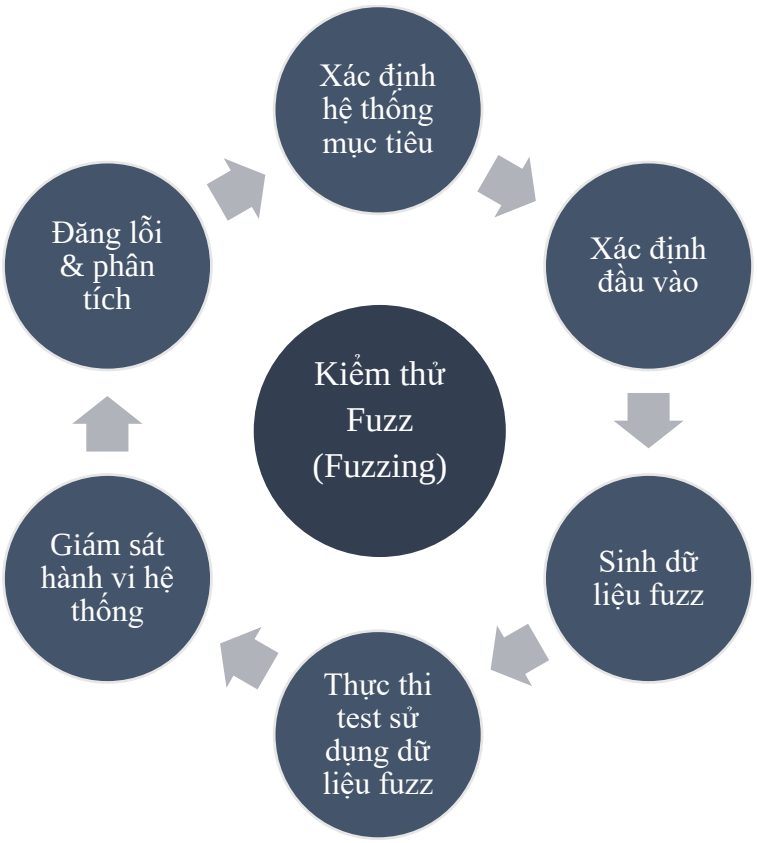
Các giai đoạn trong kiểm thử Fuzz được thể hiện bằng biểu đồ hình 2.3 [13]:

2.1.2.1. Xác định hệ thống mục tiêu (Identify target system)

Các mục tiêu được đánh giá có nguy cơ rủi ro cao gồm:

- Các lỗ hổng do lỗi của người lập trình hệ thống: SQL đơn ánh, mã nguồn đơn ánh, kịch bản hóa trang web chéo, chuyển hướng URL … Hoặc các lỗi do việc cấu hình hệ thống không an toàn như để đường dẫn vào trang quản trị hệ thống là mặc định, tài khoản mặc định…
- Các ứng dụng nhận dữ liệu qua mạng - có khả năng bị tổn hại từ xa vì tạo điều kiện cho việc thực thi mã từ xa để tạo ra các chương trình độc hại (virus, worm …).

- Các ứng dụng chạy ở mức ưu đãi cao hơn so với thông thường - gây chú ý cho kẻ tấn công thực thi mã ở mức độ đặc quyền cao hơn, được gọi là leo thang đặc quyền.
- Các ứng dụng xử lý thông tin có giá trị - loại ứng dụng này thu hút kẻ tấn công phá vỡ các điều khiển, sự toàn vẹn, tin cậy của ứng dựng để lấy dữ liệu có giá trị.
- Các ứng dụng xử lý thông tin cá nhân – kẻ tấn công có thể lấy dữ liệu cá nhân có giá trị để dùng cho mục đích riêng không hợp pháp (ví dụ: Windows Explorer, Window Registry, tập tin đa phương tiện, tài liệu văn phòng, các tập tin cấu hình)



Hình 2.3: Các giai đoạn trong kiểm thử Fuzz

2.1.2.2. Xác định đầu vào (Identify inputs)

Đầu vào ứng dụng có thể có qua nhiều hình thức thức khác nhau, hoặc từ xa (giao thông mạng), hoặc cục bộ (các tệp tin, các khóa đăng ký, các biến môi trường, đối số dòng lệnh, tên đối tượng…).

Một số bộ kiểm thử Fuzz đã được tạo ra để phục vụ cho nhiều loại đầu vào. Các lớp đầu vào ứng với các bộ kiểm thử Fuzz phổ biến như sau:

- Đối số dòng lệnh
- Các biến môi trường (ShareFuzz)

- Các ứng dụng Web (WebFuzz)
- Các định dạng tệp tin (FileFuzz)
- Các giao thức mạng (SPIKE)
- Bộ nhớ
- Các đối tượng COM (COMRaider)

2.1.2.3. Sinh dữ liệu fuzz hay còn gọi là tạo các ca kiểm thử (generate fuzzed data)

Mục đích của một bộ kiểm thử Fuzz là để kiểm tra sự tồn tại của lỗ hổng bảo mật có thể truy cập thông qua đầu vào trong các ứng dụng phần mềm. Do đó dữ liệu sinh ra trong kiểm thử Fuzz phải đạt được những yêu cầu sau:

- Tạo ra dữ liệu thử nghiệm ở các mức độ khác nhau, đảm bảo thỏa mãn điều kiện đầu vào của ứng dụng.
- Dữ liệu đầu vào được tạo ra có thể có dạng tệp tin nhị phân (Binary files), tệp tin văn bản (Text files) được sử dụng lặp đi lặp lại trong quá trình kiểm tra
- Việc tạo ra dữ liệu kiểm thử với nhiều ca kiểm thử lặp đi lặp lại để bắt lỗi khi chạy chương trình.

Bộ kiểm thử Fuzz được phân loại dựa trên hai tiêu chí khác nhau:

- *Vector đơn ánh (Injection vector) hoặc vector tấn công (Attack vector)*

Các bộ kiểm thử Fuzz có thể được chia dựa trên các lĩnh vực ứng dụng mà chúng sử dụng, nhưng về cơ bản theo hướng vector tấn công. Đối với bộ kiểm thử Fuzz theo loại vector đơn ánh nó sẽ thực hiện kiểm thử hộp đen thông qua việc nhập dữ liệu đầu vào. Các bộ kiểm thử Fuzz loại này dùng để kiểm thử phía client và một số khác để kiểm thử phía server. Đối với bộ kiểm thử Fuzz kiểm thử phía client với giao thức HTTP hoặc TLS sẽ nhằm mục tiêu vào các trình duyệt. Đối với các bộ kiểm thử Fuzz kiểm thử phía Server sẽ thực hiện kiểm thử trên máy chủ Web Server. Một số bộ kiểm thử Fuzz khác hỗ trợ kiểm thử trên cả hai Server và Client, hoặc thậm chí cả hai (dùng để phân tích proxy hoặc phân tích lưu lượng).

- *Kỹ thuật ca kiểm thử*

Bộ kiểm thử Fuzz cũng có thể được phân loại dựa trên các ca kiểm thử phức tạp. Các ca kiểm thử được tạo ra trong kiểm thử Fuzz với mục tiêu tạo ra các lớp khác

nhau trong phần mềm, và nhờ đó có thể thâm nhập vào các lớp logic khác nhau trong ứng dụng.

Bộ kiểm thử Fuzz mà thay đổi các giá trị khác nhau trong các giao thức sẽ kiểm tra được các dạng lỗ hổng như là các vấn đề về số nguyên. Khi cấu trúc thông điệp bị biến đổi dị thường, các bộ kiểm thử Fuzz sẽ tìm thấy sai sót trong phân tích cú pháp thông điệp (ví dụ như trong đặc tả XML và ASN.1).

Một số phương pháp phân loại dựa trên sự phức tạp của ca kiểm thử trong một bộ kiểm thử Fuzz:

- Bộ kiểm thử Fuzz dựa trên mẫu tĩnh và ngẫu nhiên (Static and random template-based Fuzzer): thường chỉ kiểm tra các giao thức đáp ứng những yêu cầu đơn giản hoặc các định dạng tập tin.
- Bộ kiểm thử Fuzz dựa trên khối (Block-based Fuzzer): sẽ thực hiện cấu trúc cơ bản cho một giao thức đáp ứng yêu cầu đơn giản và có thể chứa một số chức năng động thô sơ như tính toán về kiểm tra tổng và chiều dài các giá trị (lengthvalues).
- Bộ kiểm thử Fuzz dựa trên tiến hóa hoặc bộ sinh động (Dynamic generation or evolution based Fuzzer): những bộ kiểm thử Fuzz này không nhất thiết phải hiểu được giao thức hoặc định dạng tập tin đang được làm mờ, nhưng có thể tìm hiểu nó dựa trên một vòng phản hồi từ hệ thống mục tiêu.
- Bộ kiểm thử Fuzz dựa trên mô phỏng hoặc dựa trên mô hình (Model-based or simulation-based Fuzzer): những bộ kiểm thử Fuzz này thực hiện kiểm thử giao diện hoặc thông qua một mô hình hay là một mô phỏng, hoặc nó cũng có thể được triển khai đầy đủ theo một giao thức nào đó. Không chỉ có cấu trúc thông điệp được làm mờ, mà những thông điệp bất thường trong chuỗi được tạo ra cũng có thể được làm mờ.

Hiệu quả của kiểm thử Fuzz phụ thuộc vào:

- Độ bao phủ không gian đầu vào: Không gian đầu vào của giao diện kiểm thử càng tốt thì hiệu quả đạt càng cao.
- Chất lượng của dữ liệu kiểm thử: Các đầu vào độc hại tiêu biểu và dị hình sẽ làm tăng khả năng kiểm tra đối với các yếu tố hoặc cấu trúc trong định nghĩa giao diện.

2.1.2.4. Thực thi dữ liệu fuzz (execute fuzzed data)

Trong giai đoạn này, các bộ kiểm thử Fuzz thực hiện phần lớn các chức năng của các cách tiếp cận nêu trên nhưng bằng các giải pháp đặc biệt để tự động hóa quá trình xử lý kiểm thử.

Đối tượng tiếp cận của kiểm thử Fuzz bao gồm:

- Số (số nguyên dương, số âm, số thực...)
- Ký tự (urls, đầu vào dòng lệnh)
- Siêu dữ liệu
- Các chuỗi nhị phân, định dạng tệp tin (.pdf, png, .wav, .mpg…)
- Các giao thức mạng (http, SOAP, SNMP…)
- Các giao diện đầu I/O, các dòng lệnh tùy chọn, nhập/ xuất, các biểu mẫu, nội dung hay yêu cầu do người dùng tạo ra v.v…

Cách tiếp cận chung cho kiểm thử Fuzz là:

- Sinh tập dữ liệu giá trị nguy hiểm (còn được gọi là fuzz vectors) ứng với từng loại đầu vào cụ thể, các lỗ hổng, các định dạng tệp tin, mã nguồn, các giao thức hoặc tổ hợp của các dữ liệu này.
- Chèn thêm mã thực thi vào mã máy của chương trình.
- Phân tích hoạt động của chương trình trong quá trình thực thi.

2.1.2.5. Giám sát dữ liệu fuzz (monitor for execution fuzzed data)

Trong giai đoạn này, các bộ kiểm thử Fuzz không chỉ đơn thuần phát hiện các lỗ hổng qua quá trình kiểm thử mà còn phải định nghĩa các lỗi được phát hiện. Điều này có ý nghĩa hết sức quan trọng trong việc phân tích và báo cáo lỗi. Để có được một báo cáo lỗi đầy đủ và rõ ràng, đòi hỏi sự hiểu biết rõ về hoạt động xử lý. Quá trình này có thể được tích hợp vào trong sự kiện phân loại lỗi tự động.

2.1.2.6. Đăng lỗi và phân tích

Sau khi một hoặc một số lỗi phần mềm đã được xác định, các bộ kiểm thử Fuzz gửi một danh sách các lỗi này tới đội ngũ phát triển để họ có thể sửa chữa chúng.

2.1.3. Phân loại kiểm thử Fuzz

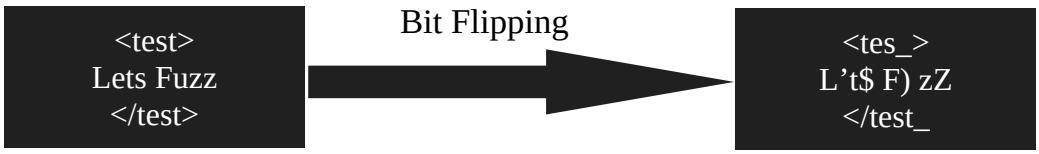
Mục tiêu của kiểm thử Fuzz đối với một ứng dụng bao gồm các tệp tin được định dạng, giao thức mạng, tham số dòng lệnh, biến môi trường, ứng dụng Web và nhiều thứ khác. Tệp tin định dạng và các giao thức mạng là mục tiêu phổ biến nhất của kiểm thử Fuzz, tuy nhiên bất kỳ loại đầu vào nào cũng có thể được làm mờ.

Việc phân loại kiểm thử Fuzz có thể tùy thuộc vào các vector tấn công, các mục tiêu kiểm thử Fuzz khác nhau hay các phương pháp kiểm thử Fuzz khác nhau, v v… Tuy nhiên phổ biến nhất là phân loại thành hai phương pháp như ở dưới đây [14]:

2.1.3.1. Kiểm thử fuzz dựa trên đột biến (Mutation based Fuzzing)

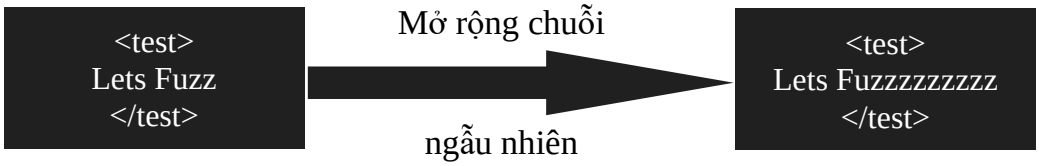
Kiểm thử Fuzz dựa trên đột biến hay còn gọi là kiểm thử fuzz câm (Dumb Fuzzing), là phương pháp biến đổi mẫu dữ liệu hiện có để tạo dữ liệu kiểm thử. Phương pháp tiếp cận này có một số tính chất sau:

- Người thực hiện ít hoặc không có kiến thức về cấu trúc của các đầu vào giả định.
- Tính dị thường được thêm vào các đầu vào hợp lệ hiện có một cách ngẫu nhiên hoặc dựa trên kinh nghiệm.
- Phụ thuộc vào các yếu tố đầu vào được sửa đổi.
- Yêu cầu ít hoặc không cần thiết trong việc thiết lập thời gian.
-



Hình 2.4: Kỹ thuật Bit Flipping

Tương tự, chúng ta cũng có thể nối thêm chuỗi vào cuối đầu vào hiện có



Hình 2.5: Kỹ thuật mở rộng chuỗi ngẫu nhiên

Một số công cụ để thực hiện cho phương pháp này: Taof, GPF, ProxyFuzz, Peach Fuzzer v.v…Ví dụ Bit Flipping [12] là một trong những kỹ thuật được sử dụng trong kiểm thử đột biến, trong đó các bit trong chuỗi được xáo trộn một cách ngẫu nhiên.

2.1.3.2. Kiểm thử fuzz dựa trên thế hệ (Generation based Fuzzing)

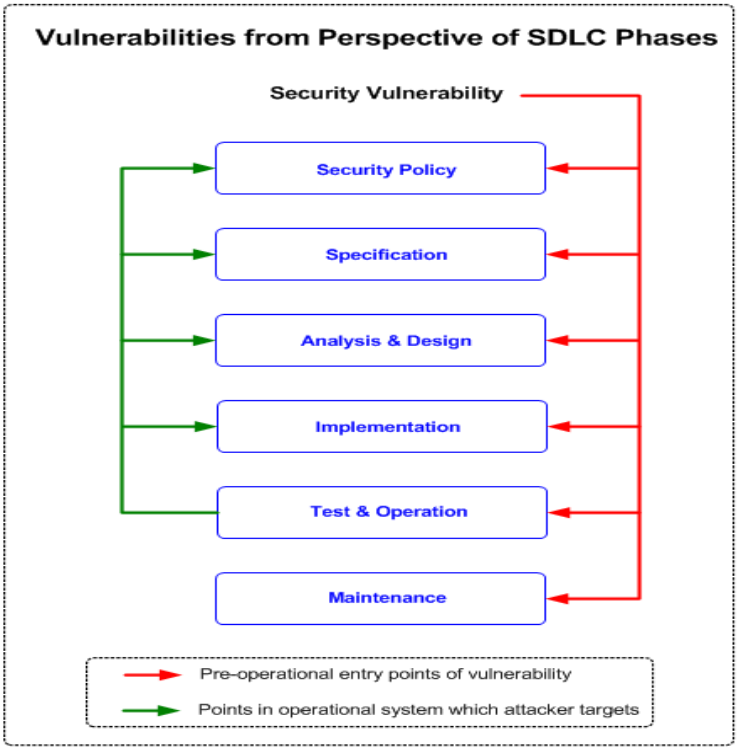
Kiểm thử Fuzz dựa trên thế hệ hay còn gọi là kiểm thử fuzz thông minh (Smart Fuzzing) thực hiện xác định dữ liệu kiểm thử mới dựa trên mô hình đầu vào.

Phương pháp tiếp cận này có một số tính chất sau:

- Các ca kiểm thử được tạo ra từ mô tả về các định dạng: RFC, các định dạng tài liệu v.v…
- Tính dị thường được thêm vào mỗi điểm có thể có trong các đầu vào.
- Hỗ trợ kiến thức về giao thức nên cho kết quả tốt hơn so với kiểm thử Fuzz ngẫu nhiên.
- Có thể mất thời gian đáng kể để thiết lập.

Một số công cụ thực hiện kiểm thử Fuzz dựa trên thế hệ: SPIKE, Sulley, Mu-4000 v.v…

2.1.4. Các lỗ hổng được phát hiện bởi kiểm thử Fuzz



Hình 2.6 [15]: Các giai đoạn trong SDLC mà các lỗ hổng được phát hiện

Lỗ hổng của ứng dụng phần mềm được tạo ra trong giai đoạn khác nhau của chu trình vòng đời phát triển phần mềm (SDLC) [14]: đặc tả, sản xuất và phát triển. Chính vì điều đó nên sản phẩm cuối cùng không tránh khỏi các vấn đề về an toàn. Kiểm thử Fuzz thường có thể phát hiện các khuyết tật bị bỏ qua khi phần mềm được viết và sửa lỗi. Thực tế cho thấy hơn 70% số lỗ hổng bảo mật hiện đại do lập trình sai sót, chỉ có ít hơn 10% là vấn đề cấu hình và khoảng 20% là vấn đề thiết kế.

Kiểm thử Fuzz làm việc tốt nhất trong việc phát hiện ra lỗi về tràn bộ nhớ (buffer overflow), kịch bản hóa chéo trang (XSS), từ chối dịch vụ (DoS), lỗi chuỗi định dạng (Format String Errors), chèn câu truy vấn (SQL Injection) v.v... Vì thế với kiểm thử Fuzz người ta có thể kiểm tra sự an toàn của bất kỳ quá trình, dịch vụ, thiết bị, hệ thống, hoặc mạng máy tính v.v…

Đối với việc kiểm tra tính bảo mật, một kỹ thuật phổ biến là kiểm thử dựa trên tập vector fuzz cụ thể nào đó, bao gồm các kịch bản để kích hoạt các loại của lỗ hổng cụ thể:

- Kịch bản hóa chéo trang (XSS)
- Tràn bộ đệm và lỗi chuỗi định dạng
- Tràn số nguyên
- Chèn truy vấn SQL
- Chèn truy vấn SQL chủ động/ bị động
- Chèn LDAP
- Chèn XML/XPATH v.v…

Kiểm thử Fuzz cũng có thể được sử dụng bởi tin tặc trong việc tìm cách có được thông tin về các hệ thống và đáp ứng hệ thống để sử dụng trong việc xây dựng các cuộc tấn công. Vì vậy điều quan trọng là phải xác định, đánh giá được rủi ro và nguy cơ trong việc tấn công các ứng dụng của mình.

Kiểm thử Fuzz một mình không thể cung cấp một bức tranh hoàn chỉnh của bảo mật tổng thể, chất lượng, hiệu quả của một chương trình trong một tình huống hoặc ứng dụng cụ thể. Các bộ kiểm thử fuzz (Fuzzer) có hiệu quả nhất khi được sử dụng kết hợp với mở rộng kiểm thử hộp đen, kiểm thử beta và các phương pháp gỡ lỗi đã được chứng minh khác.

2.1.5. Ưu nhược điểm của kiểm thử fuzz [16]

- Ưu điểm:
    - Kiểm thử Fuzz giúp tìm thấy những lỗi nghiêm trọng nhất về bảo mật hoặc khiếm khuyết.
    - Kết quả sử dụng kiểm thử Fuzz hiệu quả hơn khi sử dụng các phương pháp kiểm thử khác.
    - Kiểm thử Fuzz cải thiện vấn đề về an ninh khi kiểm tra phần mềm.
    - Lỗi được tìm thấy bằng kiểm thử Fuzz đôi khi nghiêm trọng và hầu hết là những lỗi mà tin tặc hay sử dụng. Trong đó có crashes, rò rỉ bộ nhớ, ngoại lệ không xử lý được, v.v…
    - Những lỗi không được tìm thấy khi kiểm thử bị hạn chế về thời gian và nguồn lực thì cũng được kiểm thử Fuzz tìm ra.
- Nhược điểm:
    - Chỉ riêng kiểm thử Fuzz thì không thể xử lý hết được các mối đe dọa an ninh tổng thể hoặc các lỗi.
    - Kiểm thử Fuzz kém hiệu quả với các lỗi mà không gây treo chương trình, chẳng hạn như một số loại virus, computer Worm, Trojan, …, nó chỉ hiệu quả trong các tình huống cụ thể.
    - Kiểm thử Fuzz không cung cấp nhiều kiến thức về hoạt động nội bộ của các phần mềm.
    - Với chương trình có các đầu vào phức tạp đòi hỏi phải tốn thời gian hơn để tạo ra một bộ kiểm thử Fuzz đủ thông minh.

2.1.6. Một số công cụ kiểm thử Fuzz

- Monkey
- Dynodroid [17]
- Intent fuzzer [18]
- DroidFuzzer [19]

2.2. Phương pháp dựa trên mô hình (Model based Testing)

2.2.1. Kiểm thử dựa trên mô hình (Model based Testing – MBT) là gì?

2.2.1.1. Mô hình là gì?

Mô hình [20] là một mô tả hành vi của hệ thống. Hành vi được mô tả ở đây có thể là về trình tự các đầu vào, hành động, điều kiện, đầu ra và luồng dữ liệu từ đầu vào tới đầu ra. Nó trên thực tế phải có thể hiểu được, có thể tái sử dụng được, có thể chia sẻ được và có được mô tả chính xác của hệ thống kiểm thử.

Có rất nhiều mô hình có sẵn và nó mô tả các khía cạnh khác nhau của hành vi hệ thống. Các ví dụ về mô hình như là:

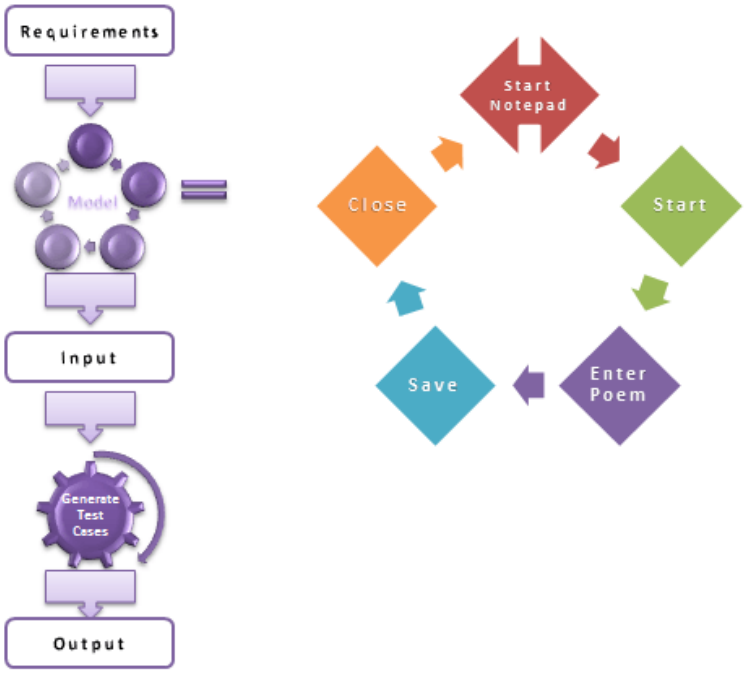
- Luồng dữ liệu
- Luồng điều khiển
- Biểu đồ phụ thuộc
- Bảng quyết định
- Máy chuyển đổi trạng thái

2.2.1.2. Kiểm thử dựa trên mô hình

Kiểm thử dựa trên mô hình [20] là một kỹ thuật kiểm tra, nơi mà hành vi thời gian chạy của một phần mềm kiểm thử được kiểm tra để chống lại những dự đoán được thực hiện bởi một đặc tả hoặc mô hình chính thức. Trong một ý nghĩa khác, nó mô tả cách hệ thống hoạt động như một phản ứng đối với một hành động (xác định bởi một mô hình). Cung cấp hành động, và xem, nếu hệ thống đáp ứng theo mong đợi.

Đây là một phương pháp hình thức nhẹ để xác nhận một hệ thống. Kiểm thử này có thể được áp dụng cho cả với phần cứng và phần mềm.

Mô hình ở hình 2.7 giải thích về cách tiếp cận đơn giản của việc viết một bài thơ trong notepad và các hành động có thể liên quan đến trong từng bước. Đối với mỗi và mọi hành động (như là bắt đầu, nhập nội dung bài thơ, lưu lại), các ca kiểm thử sẽ được sinh ra và đầu ra sẽ được xác minh.



[4]

Hình 2.7: Mô hình sinh ca kiểm thử chương trình nhập một bài thơ

2.2.2. Các loại kiểm thử dựa trên mô hình

Có hai hình thức kiểm thử dựa trên mô hình là [20]:

- Offline/ a priori: Sinh ra các bộ kiểm thử trước khi thực thi chúng. Bộ kiểm thử chính là tập hợp của các ca kiểm thử
- Online/ on-the-fly: Sinh ra các bộ kiểm thử ngay trong khi thực thi kiểm thử.

2.2.3. Các mô hình khác nhau trong kiểm thử

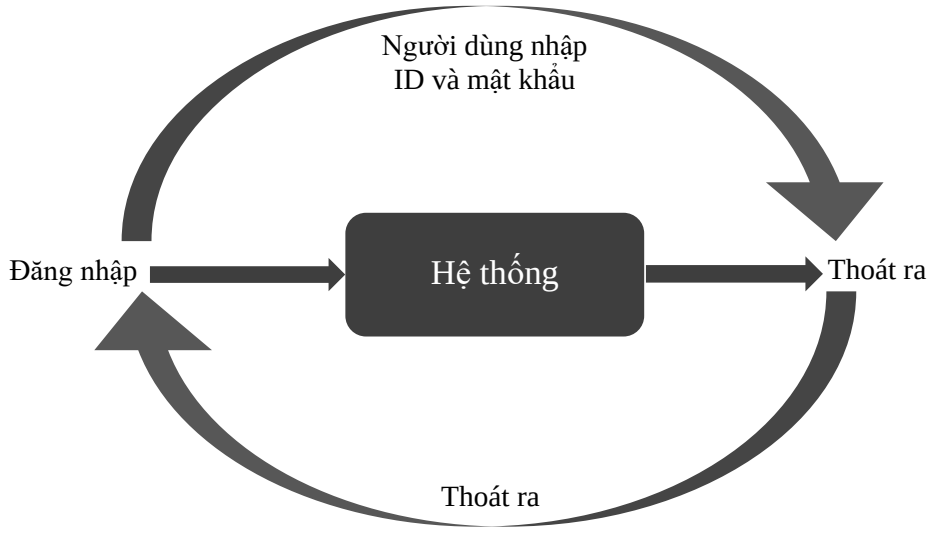
Để có thể hiểu được kiểm thử dựa trên mô hình, chúng ta cần phải hiểu được một số mô hình sẽ được trình bày dưới đây [20].

2.2.3.1. Máy trạng thái hữu hạn

Mô hình này giúp kiểm thử viên đánh giá kết quả phụ thuộc vào dữ liệu đầu vào được lựa chọn. Có thể có sự kết hợp khác nhau của đầu vào dẫn tới các kết quả trong các trạng thái tương ứng của hệ thống.

Hệ thống sẽ có trạng thái cụ thể và trạng thái hiện tại được điều chỉnh bởi bộ dữ liệu vào được đưa ra bởi kiểm thử viên.

[4] https://www.guru99.com/model-based-testing-tutorial.html



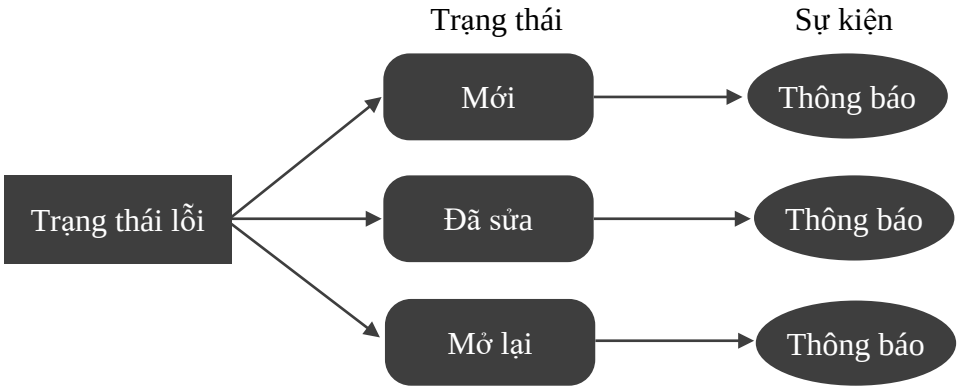
Hình 2.8: Biểu đồ trạng thái đăng nhập hệ thống

Hãy cùng xem xét một ví dụ:

Có một hệ thống cho phép nhân viên đăng nhập vào ứng dụng. Bây giờ trạng thái hiện tại của nhân viên là “Out” và nó sẽ trở thành “In” một khi nhân viên đó đăng nhập vào hệ thống. Khi ở trong trạng thái “In”, nhân viên có thể xem, in, quét tài liệu trong hệ thống.

2.2.3.2. Biểu đồ trạng thái

Nó là một phần mở rộng của máy hữu hạn trạng thái và có thể được sử dụng cho các hệ thống thời gian thực và phức tạp. Biểu đồ trạng thái được sử dụng để mô tả các hành vi khác nhau của hệ thống. Nó xác định một số lượng trạng thái. Các hành vi của hệ thống được phân tích và biểu diễn dưới dạng các sự kiện của mỗi trạng thái.



Hình 2.9: Mô hình biểu đồ trạng thái hệ thống quản lý lỗi

Một ví dụ: Các lỗi được đưa lên một công cụ quản lý lỗi với trạng thái là “Mới”. Một khi lỗi được sửa bởi lập trình viên, nó sẽ được chuyển trạng thái sang “Fixed”. Nếu lỗi vẫn chưa được sửa, trạng thái của nó sẽ chuyển sang “Re-open”. Biểu đồ trạng thái nên được thiết kế theo cách mà mỗi sự kiện được gọi cho mỗi trạng thái.

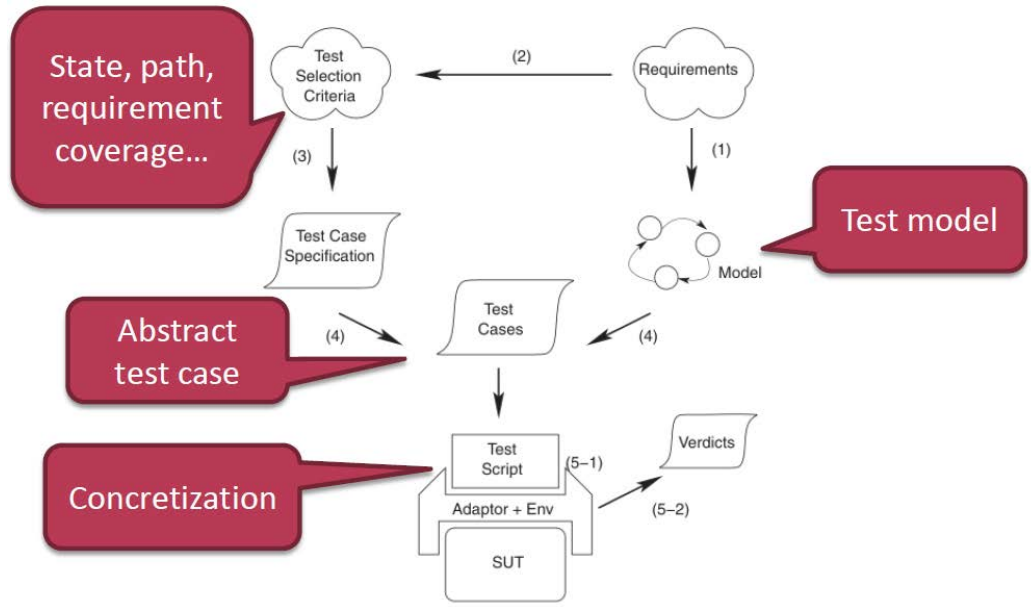
2.2.3.3. Ngôn ngữ mô hình hóa thống nhất (UML)

Ngôn ngữ mô hình thống nhất (UML) là một ngôn ngữ mô hình hóa theo mục đích chuẩn hóa chung. UML bao gồm một tập hợp các kỹ thuật ký hiệu đồ họa để tạo ra các mô hình trực quan có thể mô tả hành vi rất phức tạp của hệ thống.

UML có các ký hiệu như:

- Các hoạt động
- Các nhân tố
- Quy trình nghiệp vụ
- Các thành phần
- Ngôn ngữ lập trình

2.2.4. Tiến trình kiểm thử dựa trên mô hình



Hình 2.10: Các giai đoạn trong quá trình kiểm thử theo mô hình

Hình 2.10 [21] mô tả một quá trình kiểm thử dựa trên mô hình. Từ các yêu cầu ban đầu, thực hiện bước đầu tiên trong chuỗi các hoạt động kiểm thử là mô hình hóa. Việc tạo ra mô hình kiểm thử đòi hỏi phải mô tả những đặc tính muốn kiểm tra đủ chi

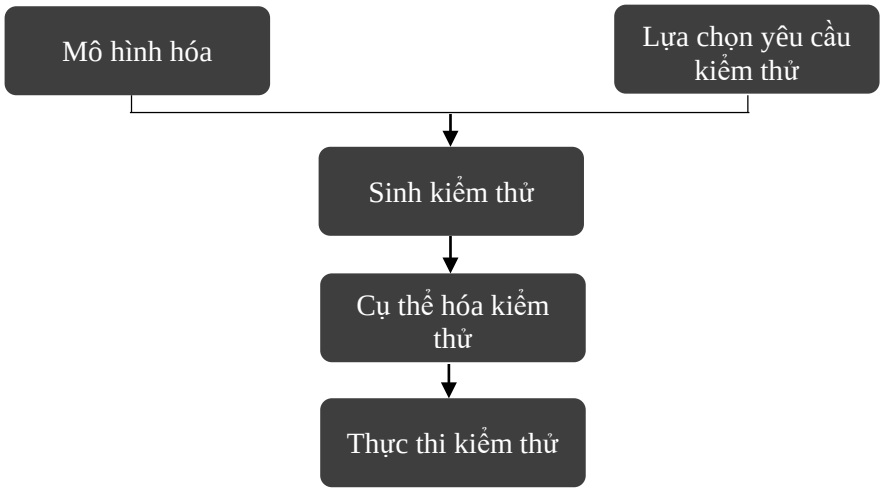
tiết. Đồng thời với hoạt động mô hình hóa là việc xác định các tiêu chí lựa chọn các trường hợp kiểm thử, từ đó sinh ra các tài liệu đặc tả cho các ca kiểm thử.

Từ hoạt động mô hình hóa và các tài liệu kiểm thử sẽ giúp sinh ra các ca kiểm thử. Việc sinh ra các ca kiểm thử trừu tượng sẽ được thực hiện hoàn toàn tự động từ mô hình bằng các sử dụng các công cụ kiểm thử dựa trên mô hình.

Bước tiếp theo trong chuỗi hoạt động là việc chuyển đổi các ca kiểm thử trừu tượng này thành các kịch bản kiểm thử có thể thực thi được bởi các công cụ kiểm thử tự động.

Và cuối cùng, sau khi đã có các kịch bản kiểm thử tự động, các công cụ kiểm thử sẽ thực thi việc kiểm tra ứng dụng kiểm thử theo các kịch bản đã được xây dựng đó.

Để hiểu rõ hơn về quy trình kiểm thử dựa trên mô hình, chúng ta cùng đi vào tìm hiểu chi tiết hơn các bước thức hiện của phương pháp này theo một lược đồ dạng đơn giản hơn như hình 2.11 [22]



Hình 2.11: Các bước trong kiểm thử dựa trên mô hình

2.2.4.1. Mô hình hóa

Trong bước này, chúng ta tiến hành việc xây dựng một mô hình cho hệ thống kiểm thử dựa trên các nền tảng là các yêu cầu. Yêu cầu đối với việc mô hình hóa là mô hình thiết kế phải đạt được các mục đích kiểm thử. Để xây dựng được một mô hình phù hợp, cần phải xem xét đến việc lựa chọn một ký tự phù hợp cho mô hình, lựa chọn đúng mức độ cho việc trừu tượng hóa (nó chính là những mặt của phần mềm kiểm thử

mà chúng ta cần kiểm tra). Việc mô hình hóa có thể là một mối quan hệ nhiều – nhiều giữa các hoạt động của mô hình với hoạt động củ hệ thống kiểm thử. Một khi mô hình được tạo ra, cần phải đảm bảo rằng mô hình đó được tạo một cách ngắn gọn và chính xác nhất.

Một số các ký hiệu mô hình hóa [23]:

Ký hiệu dựa trên trạng thái (Pre/Post): VDM, Z, Spec#

Một ví dụ về VDM++

```
class Stack
instance variables
stack: seq of int := [];
--inv
operations
Stack : () ==> ()
Stack () == stack := []
post stack = [];
Push : int ==> ()
Push (i) == stack := [i] ^ stack
post stack = [i] ^ -stack;
Pop() ==> ()
Pop() == stack := tl stack
pre stack <> []
post stack = tl – stack;
Top : () ==> int
Top() == return (hd stack)
pre stack <> []
post RESULT = hd stack
and stack = - stack;
end Stack
```

Các ký hiệu dựa trên chuyển tiếp: máy hữu hạn trạng thái

Việc lựa chọn một bộ các trạng thái là một bước quan trọng

Biểu đồ trạng thái cũng là một lựa chọn khác của ký hiệu này

- Các ký hiệu chức năng
- Các ký hiệu hoạt động
- Các ký hiệu luồng dữ liệu

Trong việc lựa chọn một bộ ký hiệu, Pre/Post (sử dụng cho mô hình hóa những dữ liệu phức tạp) và dựa trên chuyển đổi trạng thái (cho mô hình hóa điều khiển) là những bộ ký hiệu phổ biến nhất trong quy trình kiểm thử phần mềm dựa trên mô hình. Tuy nhiên với bất cứ bộ ký hiệu nào mà chúng ta chọn, nó phải có ngôn ngữ chính thức với

ngữ nghĩa chính xác để có thể viết được các mô hình chính xác sử dụng trong việc kiểm thử oracles.

2.2.4.2. Lựa chọn yêu cầu kiểm thử:

Đây là bước được sử dụng để điều khiển việc sinh ra các kiểm thử. Các thao tác được thực hiện trong bước này là [22]:

- Các tiêu chí lựa chọn trường hợp kiểm thử được xác định
- Các tiêu chí lựa chọn trường hợp kiểm thử sau đó được chuyển đổi thành các đặc tả ca kiểm thử

Nói một cách cụ thể hơn, trong bước này chúng ta cần xây dựng một tập hợp các bộ kiểm thử bao gồm chuỗi các hành động để thực hiện, dữ liệu đầu vào và kết quả mong đợi. Bộ kiểm thử được coi là tốt nhất khi nó có số lượng nhỏ nhất nhưng lại có thể tìm ra được số lỗi nhiều nhất. Bộ kiểm thử lý tưởng là sự kết hợp giữa việc bao phủ mã nguồn tốt đồng thời bao phủ các yêu cầu (hoặc đặc tả) tối đa. Chính vì thế mà chúng ta có những tiêu chí và phân tích cho việc bao phủ.

Các tiêu chí về bao phủ giúp sinh ra các bộ kiểm thử đảm bảo và giúp xác định được khi nào sẽ dừng kiểm tra, nhưng như đã nói ở trên, sự hiểu biết về ứng dụng kiểm tra của kiểm thử viên sẽ là một nhân tố quyết định cho việc thành công

Việc phân tích độ bao phủ mục đích để đo lường phạn vi mà hoạt động xác minh đã đạt được và nó cũng có thể được sử dụng để đánh giá chất lượng của bộ kiểm thử đồng thời nó cũng giúp xác định khi nào sẽ dừng hoạt động xác minh lại. Nó thường được biểu thị bằng tỉ lệ phần trăm để hoàn thành một phần của một hoạt động.

Một số các tiêu chí bao phủ chính sử dụng để đánh giá cho bộ kiểm thử được sinh ra [23]:

- Tiêu chí bao phủ cấu trúc: nhằm mục đich thực hiện mã nguồn hoặc mô hình liên quan đến một vài mục đích bao phủ
- Tiêu chí bao phủ dữ liệu: mục đích để bao phủ không gian dữ liệu đầu vào của một hoạt động hoặc một chuyển đổi trạng thái
- Tiêu chí dựa trên lỗi: mục đích để sinh ra các bộ kiểm thử phù hợp cho việc phát hiện ra những loại lỗi cụ thể

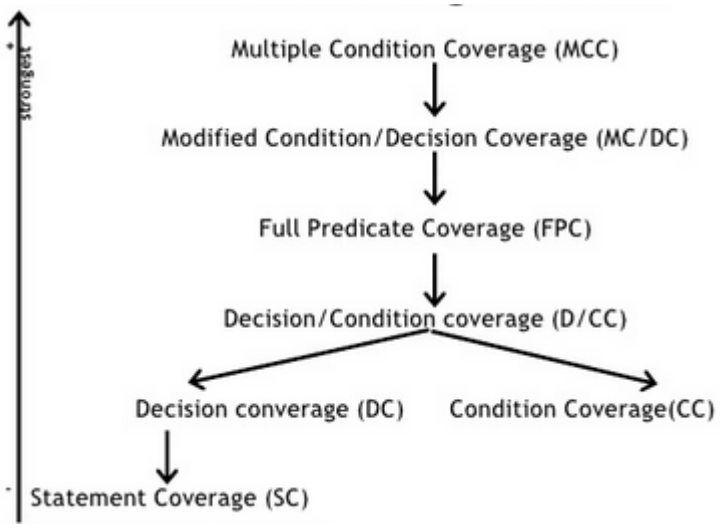
- Tiêu chí bao phủ yêu cầu: mục đích để đảm bảo rằng mỗi một yêu cầu đều được kiểm tra

Chúng ta cùng đi sâu vào tìm hiểu chi tiết từng tiêu chí một:

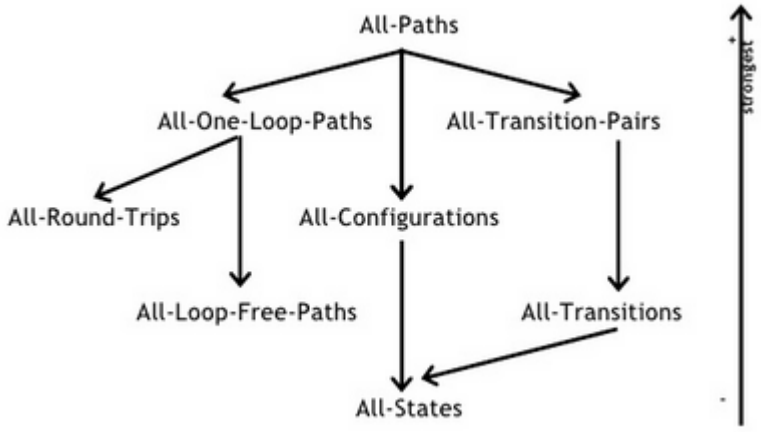
Tiêu chí bao phủ cấu trúc

Bao phủ cấu trúc sẽ bao gồm việc bao phủ ở những thành phần nhỏ hơn trong cấu trúc đó:

- Bao phủ câu lệnh (Statement coverage – SD): mỗi câu lệnh có thể được thực thi sẽ được gọi đến ít nhất một lần
- Bao phủ quyết định (Decision coverage – DC): những kết quả biểu hiện ra phải được kiểm tra đối với cả trường hợp "True" và "False" (ví dụ (A or B) được kiểm tra cho TF và FF)
- Bao phủ điều kiện (Condition coverage – CC): mỗi một điều kiện trong biểu thức đều có tất cả các đầu ra có thể (ví dụ (A or B) được kiểm tra cho TF và FT)
- Bao phủ điều kiện/ rẽ nhánh (Decision/condition coverage – D/CC): là sự kết hợp giữa hai tiêu chí ở trên (ví dụ (A or B) được kiểm tra cho TT và FF)
- Bao phủ điều kiện/rẽ nhánh được sửa đổi (Modified condition/decision coverage – MC/DC): mỗi điều kiện ảnh hưởng độc lập đến kết quả của quyết định (ví dụ (A or B) được kiểm tra cho TF, FT và FF)
- Bao phủ đa điều kiện (Multiple condition coverage – MCC): kiểm tra mỗi sự kết hợp có thể của các dữ liệu đầu vào. Kiểm thử 2^n lần cho một rẽ nhánh với n đầu vào (hầu như là không khả thi)



Hình 2.12 [23]: Luồng điều khiển tiêu chí kiểm thử cấu trúc



Hình 2.13 [23]: Tiêu chí kiểm thử cấu trúc với máy trạng thái hữu hạn

Tiêu chí bao phủ dữ liệu

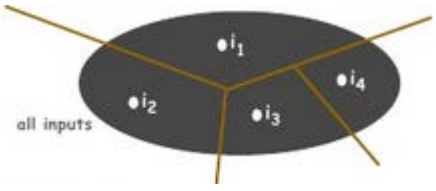
Áp dụng tiêu chí bao phủ cho dữ liệu sẽ có ích cho việc lựa chọn những giá trị dữ liệu tốt để sử dụng cho các đầu vào kiểm thử. Để lựa chọn giá trị của dữ liệu trên một miền D, hai tiêu chí bao phủ dữ liệu cực đoan là:

- Một giá trị: chọn ít nhất một giá trị từ D (kết hợp với các tiêu chí kiểm thử khác có thể hữu ích)
- Toàn bộ các giá trị: toàn bộ các giá trị trong miền D (để thực hiện hết các trường hợp là không khả thi).

Ngoài hai trường hợp lựa chọn giá trị ở trên, chúng ta còn có một số phương pháp lựa chọn khác mang lại hiệu quả cao hơn:

- Phân lớp tương đương:

- Một phân vùng của một vài tập S là một tập hợp của các tập con không rỗng SS_1, …, SS_n, trong đó mỗi SS_i và SS_j được phân chia và việc kết hợp của toàn bộ các tập SS_i sẽ bằng S.
- Nếu một lỗi được phát hiện bởi một thành phần của lớp, nó được kỳ vọng rằng một lỗi tương tự cũng sẽ được phát hiện bởi một thành phần khác trong cùng lớp đó



Hình 2.14: Ví dụ về phân lớp tương đương

- Phân tích giá trị biên:

- Phân tích giá trị biên kiểm tra các điều kiện biên của các lớp tương đương để lựa chọn giá trị biên đầu vào. Kỹ thuật này dựa trên các kiến thức về giá trị đầu vào tại biên hoặc vượt ra ngoài biên của miền giá trị với mong muốn gây ra lỗi trong hệ thống

- Sinh giá trị ngẫu nhiên: việc lựa chọn sinh giá trị ngẫu nhiên việc phát hiện lỗi cũng hiệu quả như với phân lớp tương đương. Nó giúp tiết kiệm chi phí hơn. Giá trị của một biến dữ liệu được đưa ra trong bộ kiểm thử để thực hiện theo những phân phối thống kê trong miền dữ liệu.

- Phương pháp hướng mục tiêu: Phương pháp hướng mục tiêu cố gắng để điều khiển hệ thống vào trong một mục tiêu đưa ra bằng hai phương thức khác nhau: cách tiếp cận chuỗi và tiếp cận hướng khẳng định.

- Phương pháp tiếp cận chuỗi cố gắng tìm một đường dẫn để thực hiện một nút mục tiêu nhất định dựa trên phân tích phụ thuộc dữ liệu.

- Phương pháp tiếp cận hướng khẳng định cố gắng tìm bất cứ đường dẫn nào tới được một khẳng định mà nó không bị giữ lại

- Phương thức hướng đường dẫn: một ví dụ về kiểm thử biểu tượng (symbolic testing). Nó thay thế các biến của chương trình bằng các biểu tượng và tính toán các ràng buộc, cái mà đại diện cho các đường dẫn thực thi biểu tượng có thể. Khi biến của chương trình được thay đổi trong quá trình thực thi, một giá trị mới được thể hiện như là một ràng buộc thông qua các biến biểu tượng. Một hệ thống giải quyết các ràng buộc có thể được sử dụng để tìm kiếm, và khi có thể, các giá trị rời rạc gây ra việc thực thi của đường dẫn được mô tả bằng mỗi ràng buộc.

Tiêu chí dựa trên lỗi

Trong tiêu chí này, chúng ta sử dụng một kỹ thuật kiểm thử phần mềm trong đó các dữ liệu kiểm thử được thiết kế để chứng minh sự vắng mặt của một tập hợp các lỗi đã được xác định trước (những lỗi đã biết hoặc lỗi lặp lại)

Kiểm thử đột biến (mutantion testing) được sử dụng để đạt tiêu chí dựa trên lỗi. Kỹ thuật đột biến giới thiệu những thay đổi nhỏ (các lỗi) bằng cách áp dụng các hoạt động đột biến vào trong đặc tả ban đầu. Các đặc tả thay đổi này được gọi là các đột biến.

Mục đích của phương pháp này là để xây dựng các ca kiểm thử phân biệt được mỗi đột biến so với nguyên bản ban đầu bằng cách sản sinh ra các kết quả khác nhau. Nếu xảy ra, nó có thể nói rằng ca kiểm thử đã giết chết một đột biến.

Một các kiểm thử tốt phải có khả năng giết chết được các đột biến bởi vì nếu nó có thể phát hiện ra những thay đổi nhỏ được sinh ra bởi các hoạt động của đột biến, nó có khả năng sẽ tìm ra được những lỗi thật của hệ thống.

Tỉ lệ của việc giết các đột biến (sau khi đã loại bỏ các đột biến mà tương đương với mã nguồn ban đầu) đưa ra dấu hiệu về tỉ lệ của số lỗi chưa được phát hiện mà có thể tồn tại trong mã nguồn ban đầu.

Một trong những vấn đề của kiểm thử đột biến là nó không đủ các kỹ thuật để tạo ra dữ liệu kiểm thử.

Tiêu chí bao phủ yêu cầu

Bao phủ yêu cầu thường ít có tính hệ thống và thường không bao gồm toàn bộ các đặc tả của hành vi hệ thống. Tuy nhiên, có ít nhất hai hướng tiếp cận để cố gắng hệ thống hóa nó hơn:

- Ghi lại các yêu cầu bên trong mô hình hành vi (như là các ký hiệu trong một vài phần của mô hình) để mà quá trình sinh trường hợp kiểm thử có thể đảm bảo rằng toàn bộ các yêu cầu đã được kiểm tra
- Chính thức hóa mỗi yêu cầu và sau đó sử dụng những biểu hiện chính thức đó như là một tiêu chí lựa chọn kiểm thử để điều khiển việc sinh tự động của một hoặc nhiều kiểm thử từ mô hình hành vi

2.2.4.3. Sinh kiểm thử

Một khi mô hình và đặc tả ca kiểm thử đã được xác định, một bộ kiểm thử trừu tượng sẽ được tạo ra [22].

Kỹ thuật được sử dụng ở đây là kiểm tra mô hình (model checking). Bất cứ khi nào một thuộc tính, được biểu thị trong logic tạm thời, không chứa trong một hệ thống được mô tả như là một máy hữu hạn trạng thái, kiểm tra mô hình cố gắng để sinh một ví dụ truy cập

Khi ví dụ truy cập được sản sinh, nó có thể được sử dụng như một chuỗi ca kiểm thử của việc chuyển đổi trong máy hữu hạn trạng thái với các đầu vào và đầu ra mong đợi

Để có hiệu quả như một kỹ thuật sinh các ca kiểm thử, các thuộc tính về hệ thống nên được mô tả theo cách mà các ví dụ truy cập được sinh ra khi chúng được sử dụng bởi các ca kiểm thử.

Có hai phương pháp sinh ca kiểm thử bởi kiểm tra mô hình là:

- Sinh ca kiểm thử từ mô hình dựa trên thuộc tính

• Các kỹ thuật sử dụng để sinh các ca kiểm thử từ những tài liệu đặc tả được viết lại và xử lý các ràng buộc

• Đưa ra một tập các biểu thức (các khẳng định logic hoặc các mối quan hệ tương đương) và một tập hợp các biến trong những biểu thức đó, các kỹ thuật giải quyết ràng buộc cố gắng để tìm ra giải thích của các biến mà làm giảm sự đúng đắn của biểu thức.

- Sinh ca kiểm thử từ mô hình dựa trên hành vi: Phân tích các dấu vết thực thi để sinh ra các ca kiểm thử

2.2.4.4. Cụ thể hóa kiểm thử (chuyển đổi) [22]

Trong bước này, thực hiện cụ thể hóa những bộ kiểm thử trừu tượng được sinh ở bước trên thành các kịch bản kiểm thử có thể thực thi được bằng công cụ.

Bước này được thực hiện bởi công cụ kiểm thử dựa trên mô hình sử dụng các bảng chuyển đổi cung cấp bởi kỹ sư kiểm thử.

Kết quả thực thi kiểm thử có thể là JUnit ở trong Java hoặc là một ngôn ngữ động như là Tcl hoặc Python, hoặc trong các ngôn ngữ kịch bản thử nghiệm chuyên dụng.

2.2.4.5. Thực thi kiểm thử

Trong giai đoạn này các kịch bản kiểm thử sẽ được thực thi, kết quả thực tế đầu ra sẽ được so sánh với các kết quả mong đợi từ đó mà đưa ra được những kết quả Pass, Fail cho từng kịch bản kiểm thử.

2.2.5. Ưu nhược điểm của kiểm thử dựa trên mô hình

- Ưu điểm:
    - Cho phép kiểm thử toàn diện ứng dụng

- Hoàn toàn phù hợp cho kiểm tra chức năng/ tính chính xác của ứng dụng
- Các mô hình có thể dễ dàng đáp ứng các thay đổi từ ứng dụng

- Nhược điểm:
  - Yêu cầu phải có một mô hình/ đặc tả chính thức
  - Các vấn đề về việc bùng nổ các ca kiểm thử
  - Việc sinh các ca kiểm thử phải được điều khiển một cách thích hợp để các ca kiểm thử được sinh ra có khối lượng có thể quản lý được
  - Một thay đổi nhỏ từ mô hình có thể dẫn đến kết quả là toàn bộ bộ kiểm thử bị thay đổi
  - Mất thời gian trong việc phân tích cho các kiểm thử lỗi (mô hình, phần mềm kiểm thử)

2.2.6. Một số công cụ kiểm thử dựa trên mô hình [24]

- GUIRipper
- ORBIT
- A3E Depth First
- SwiftHand
- PUMA

Chương 3. Một số công cụ sinh đầu vào kiểm thử tự động cho ứng dụng Android

3.1. Công cụ kiểm thử ngẫu nhiên – Monkey tool

3.1.1. Tổng quan chung về Monkey tool

Monkey là một phần của Android SDK được phát triển bởi Google sử dụng cho việc kiểm thử tự động các ứng dụng Android. Với việc tích hợp sẵn trong Android Studio, Monkey là một công cụ hữu ích cho các lập trình viên trong việc kiểm tra ứng dụng ngay trong quá trình phát triển. Nó sử dụng kỹ thuật ngẫu nhiên/ mờ trong việc sinh ra các sự kiện người dùng làm đầu vào cho quá trình kiểm thử.

Monkey [25] chạy bằng dòng lệnh, người dùng có thể chạy trên bất kỳ trình mô phỏng nào hoặc trên thiết bị thật. Nó sẽ gửi một luồng ngẫu nhiên của các sự kiện người dùng vào trong hệ thống, và sẽ thực hiện chạy stress test trên ứng dụng phần mềm.

Monkey bao gồm một số tùy chọn, nhưng chúng được chia thành bốn loại chính:

- Các tùy chọn cấu hình cơ bản, như là cài đặt số lượng các sự kiện mong muốn thực hiện
- Các ràng buộc về hoạt động, như là giới hạn kiểm tra với một gói duy nhất
- Loại sự kiện và tần số
- Tùy chọn gỡ lỗi

Khi Monkey chạy, nó tạo ra các sự kiện và gửi chúng đến hệ thống. Nó cũng theo dõi hệ thống đang được kiểm tra và tìm kiếm ba điều kiện mà nó xử lý đặc biệt:

- Nếu người dùng hạn chế Monkey chỉ cho phép chạy trong một hoặc nhiều gói cụ thể, nó sẽ theo dõi những cố gắng di chuyển tới bất kỳ các gói khác và chặn chúng lại
- Nếu ứng dụng người dùng bị treo hoặc nhận bất kỳ loại ngoại lệ không được xử lý, Monkey sẽ dừng lại và báo cáo lỗi
- Nếu ứng dụng tạo ra một lỗi không phản hồi, Monkey sẽ dừng lại và báo cáo lỗi

Tùy thuộc vào mức độ lệnh mà người dùng chọn mà thấy được các báo cáo về tiến trình của Monkey và các sự kiện đang được tạo ra một cách tương ứng

Sử dụng Monkey cơ bản

Thực hiện khởi chạy Monkey bằng cách sử dụng một dòng lệnh trên máy hoặc từ các kịch bản có sẵn. Vì Monkey chạy trong môi trường thiết bị mô phỏng hoặc thiết bị thật, do đó người dùng phải khởi chạy nó từ một shell trong chính môi trường đó. Có thể thực hiện điều này bằng cách đặt adb shell trước mỗi lệnh, hoặc bằng cách vào shell và nhập lệnh Monkey trực tiếp [25]

Cú pháp cơ bản là:

$ adb shell monkey [options] <event-count>

Không có tùy chọn nào được chỉ định, Monkey sẽ khởi chạy ở chế độ tĩnh (non-verbose), và sẽ gửi sự kiện đến bất kỳ (và tất cả) các gói cài đặt trên thiết bị kiểm tra. Đây là một dòng lệnh điển hình hơn, nó sẽ khởi chạy ứng dụng và gửi đi 500 sự kiện ngẫu nhiên vào nó:

$ adb shell monkey -p your.package.name -v 500

Chi tiết hơn về các lệnh của Monkey có thể tham khảo tại trang Android Developer ở link sau: https://developer.android.com/studio/test/monkey.html

3.1.2. Kiểm thử Fuzz với Monkey

Bước 1: Xác định hệ thống mục tiêu

Xác định hệ thống mục tiêu bằng cách truyền các tham số về gói và danh mục của ứng dụng cần thực thi kiểm thử vào trong lệnh chạy -p <allowed-package-name> -c <main-category>, khi đó Monkey sẽ thực thi trên các thành phần đã được lựa chọn

Bước 2: Xác định đầu vào

Xác định các đầu vào kiểm thử trong Monkey được thực hiện bằng cách lựa chọn và thiết lập tỉ lệ các sự kiện mong muốn sinh ra trong quá trình thực thi kiểm thử thông qua việc truyền vào các lệnh:

-s <seed>	Giá trị hạt nhân cho bộ sinh mã giả ngẫu nhiên. Nếu chạy lại Monkey với cùng một giá trị hạt nhân sẽ cho cùng một chuỗi các sự kiện
--throttle <milliseconds>	Chèn thời gian trì hoãn giữa các sự kiện
--pct-touch <percent>	Điều chỉnh tỉ lệ phần trăm các sự kiện chạm màn hình
--pct-motion <percent>	Điều chỉnh tỉ lệ phần trăm các sự kiện chuyển động
--pct-trackball <percent>	Điều chỉnh tỉ lệ phần trăm các sự kiện trackball

--pct-nav <percent>	Điều chỉnh tỉ lệ phần trăm các sự kiện điều hướng đơn giản
--pct-majornav <percent>	Điều chỉnh tỉ lệ phần trăm các sự kiện điều hướng phức tạp
--pct-syskeys <percent>	Điều chỉnh tỉ lệ phần trăm các sự kiện sử dụng phím hệ thống
--pct-appswitch <percent>	Điều chỉnh tỉ lệ phần trăm việc khởi động một hoạt động ứng dụng
--pct-anyevent <percent>	Điều chỉnh tỉ lệ phần trăm của các loại sự kiện khác

Bảng 3.1: Các sự kiện đầu vào trong Monkey

Bước 3: Sinh dữ liệu kiểm thử

Sau khi đã xác định được hệ thống mục tiêu và thành phần các sự kiện sẽ sinh ra trong quá trình thực thi kiểm thử thông qua lệnh truyền vào cho Monkey, Monkey sẽ tiến hành việc sinh các đầu vào kiểm thử là các sự kiện tương ứng với yêu cầu được đưa ra. Các sự kiện này được sinh ra một cách ngẫu nhiên nhưng vẫn tuân theo một tỉ lệ nhất định mà chúng ta đã đưa vào trong lệnh ban đầu, dưới dạng các lệnh ADB để truyền tới thiết bị kiểm thử. Trong hình 3.1 là màn hình console thể hiện các sự kiện kiểm thử đang được sinh ra bởi Monkey.

```
C:\Windows\System32>adb shell monkey -p com.simplemobiletools.camera --throttle
1000  --ignore-crashes --ignore-timeouts --ignore-security-exceptions -v 100
:Monkey: seed=1512269950778 count=100
:AllowPackage: com.simplemobiletools.camera
:IncludeCategory: android.intent.category.LAUNCHER
:IncludeCategory: android.intent.category.MONKEY
// Event percentages:
//   0: 15.0%
//   1: 10.0%
//   2: 2.0%
//   3: 15.0%
//   4: -0.0%
//   5: -0.0%
//   6: 25.0%
//   7: 15.0%
//   8: 2.0%
//   9: 2.0%
//   10: 1.0%
//   11: 13.0%
:Switch: #Intent;action=android.intent.action.MAIN;category=android.intent.categ
ory.LAUNCHER;launchFlags=0x10200000;component=com.simplemobiletools.camera/.acti
vities.SplashActivity;end
    // Allowing start of Intent { act=android.intent.action.MAIN cat=[android.in
tent.category.LAUNCHER] cmp=com.simplemobiletools.camera/.activities.SplashActiv
ity launchParam=MultiScreenLaunchParams { mDisplayId=0 mFlags=0 } } in package c
om.simplemobiletools.camera
:Sending Touch (ACTION_DOWN): 0:(203.0,1736.0)
:Sending Touch (ACTION_UP): 0:(199.55788,1732.1193)
:Sending Touch (ACTION_DOWN): 0:(170.0,1793.0)
:Sending Touch (ACTION_UP): 0:(174.31602,1805.9003)
:Sending Touch (ACTION_DOWN): 0:(568.0,399.0)
:Sending Touch (ACTION_UP): 0:(563.54297,398.90967)
```

Hình 3.1. Sinh dữ liệu kiểm thử với Monkey

Bước 4: Thực thi kiểm thử

Với dữ liệu kiểm thử là các dòng sự kiện được sinh ra ở bước 3, dưới dạng các lệnh ADB. Các lệnh ADB này được truyền tới thiết bị kiểm thử và ở đó, thiết bị kiểm

thử được thực thi các thao tác sử dụng như một người dùng bình thường một cách hoàn toàn tự động

Bước 5: Giám sát hành vi hệ thống

Trong quá trình Monkey thực thi các thao tác trên thiết bị kiểm thử, các sự kiện sinh ra đều được lưu lại dưới dạng các tệp tin log. Đồng thời hành vi của ứng dụng kiểm thử cũng được giám sát đầy đủ, những bất thường của ứng dụng như crash, lỗi timeout hay lỗi ngoại lệ về bảo mật đều được Monkey bắt lại và thông báo cho chúng ta

Bước 6: Đăng lỗi và phân tích

Như đã nói ở bước 5, những bất thường của hành vi ứng dụng đều được Monkey ghi nhận lại và sinh ra các thông báo lỗi trên màn hình điều khiển. Những thông báo lỗi này được sinh ra dưới dạng các tệp tin log như trong hình 3.2. Những nội dung trong tệp tin log này sẽ là những thông tin giúp lập trình viên phân tích và sửa lỗi

```
C:\Windows\System32>adb shell monkey -p com.simplemobiletools.camera --throttle
100  --ignore-crashes --ignore-timeouts --ignore-security-exceptions -v 50000 >
log.txt
// CRASH: com.simplemobiletools.camera (pid 20949)
// Short Msg: java.lang.IllegalStateException
// Long Msg: java.lang.IllegalStateException: You need to use a Theme.AppCompat
theme (or descendant) with this activity.
// Build Label: samsung/nobleltedv/noblelte:7.0/NRD90M/N920IDVU3CQH2:user/releas
e-keys
// Build Changelist: N920IDVU3CQH2
// Build Time: 1503300945000
// java.lang.RuntimeException: Unable to start activity ComponentInfo{com.simple
mobiletools.camera/com.simplemobiletools.camera.activities.SettingsActivity}: ja
va.lang.IllegalStateException: You need to use a Theme.AppCompat theme (or desce
ndant) with this activity.
//      at android.app.ActivityThread.performLaunchActivity(ActivityThread.java:
2927)
//      at android.app.ActivityThread.handleLaunchActivity(ActivityThread.java:2
988)
//      at android.app.ActivityThread.-wrap14(ActivityThread.java)
//      at android.app.ActivityThread$H.handleMessage(ActivityThread.java:1631)
//      at android.os.Handler.dispatchMessage(Handler.java:102)
//      at android.os.Looper.loop(Looper.java:154)
//      at android.app.ActivityThread.main(ActivityThread.java:6682)
//      at java.lang.reflect.Method.invoke(Native Method)
//      at com.android.internal.os.ZygoteInit$MethodAndArgsCaller.run(ZygoteInit
.java:1520)
//      at com.android.internal.os.ZygoteInit.main(ZygoteInit.java:1410)
// Caused by: java.lang.IllegalStateException: You need to use a Theme.AppCompat
 theme (or descendant) with this activity.
//      at android.support.v7.app.AppCompatDelegateImplV9.createSubDecor(AppComp
```

Hình 3.2: Thông tin log lỗi sinh bởi Monkey

3.2. Công cụ kiểm thử dựa trên mô hình – DroidBot

3.2.1. Tổng quan chung về DroidBot

DroidBot là một công cụ sinh đầu vào kiểm thử mã nguồn mở dạng nhẹ dựa trên UI cho các ứng dụng Android, được phát triển bởi Yuanchun Li, nghiên cứu sinh của Học viện phần mềm, Đại học Bắc Kinh.

Nguyên tắc thiết kế của DroidBot [26] là hỗ trợ việc sinh đầu vào kiểm thử dựa trên mô hình với những yêu cầu tối thiểu. DroidBot cung cấp bộ sinh đầu vào theo hướng dẫn UI dựa trên mô hình chuyển đổi trạng thái được tạo ra khi đang chạy. Sau đó nó sẽ sinh ra đầu vào kiểm thử theo hướng dẫn UI dựa trên mô hình chuyển tiếp. Mặc định đầu vào sẽ được sinh ra với chiến lược tham ăn breadth-first, tuy nhiên người dùng cũng có thể tùy chỉnh chiến lược thăm dò bằng cách tự viết các kịch bản kiểm thử hoặc tích hợp các thuật toán của riêng mình bằng cách mở rộng các mô đun sinh sự kiện.

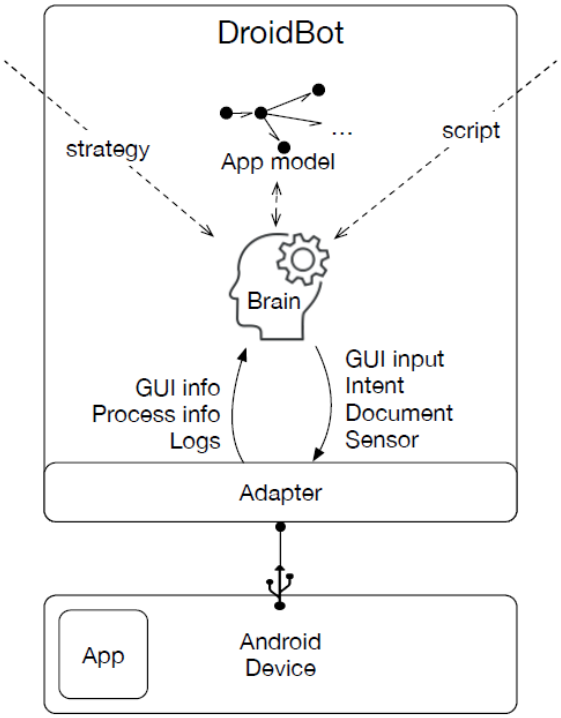
DroidBot là công cụ khá nhẹ vì nó không đòi hỏi những kiến thức trước về những phần mã nguồn chưa được khám phá. DroidBot chỉ thực hiện mô hình hóa những trạng thái đã được khám phá dựa trên một bộ công cụ kiểm tra/ gỡ lỗi được tích hợp sẵn của Android. Mặc dù điều này có thể làm cho DroidBot khó kích hoạt một số trạng thái cụ thể nhưng bù lại, nó cho phép DroidBot hoạt động với bất kỳ ứng dụng nào (bao gồm cả các ứng dụng đã được che giấu/ mã hóa mà không thể đo đạc được) trên hầu hết các thiết bị tùy biến (trừ những thiết bị đã cố tình xóa bỏ những mô đun kiểm tra/ gỡ lỗi tích hợp sẵn trong nền tảng Android ban đầu, mà điều này thì hiếm khi xảy ra)

DroidBot cũng đưa ra một cách mới để đánh giá tính hiệu quả của các đầu vào kiểm thử. Các phương pháp hiện tại chủ yếu sử dụng EMMA cho các ứng dụng mã nguồn mở hoặc ứng dụng có thể đo đạc để tính độ bao phủ của kiểm thử. Tuy nhiên, đối với những ứng dụng chống đo đạc (ví dụ như xác minh chữ ký hoặc mã hóa mã nguồn), sẽ rất là khó khăn hoặc thậm chí là không thể lấy được thông tin độ bao phủ kiểm thử của những ứng dụng này. DroidBot có thể tạo ra dấu vết ngăn xếp cuộc gọi cho mỗi đầu vào kiểm thử, trong đó bao gồm các phương thức của ứng dụng và phương thức của hệ thống được kích hoạt bởi đầu vào kiểm thử. Chúng ta có thể sử dụng ngăn xếp cuộc gọi như một thước đo gần đúng để định lượng tính hiệu quả của các đầu vào kiểm thử.

Mã nguồn của DroidBot được lưu trữ và chia sẻ trên GitHub: https://github.com/honeynet/droidbot

Kiến trúc của DroidBot

Kiến trúc tổng quan của DroidBot được biểu diễn như trong hình 3.3. Để kiểm tra một ứng dụng trên một thiết bị, DroidBot yêu cầu thiết bị phải được kết nối thông qua ADB. Thiết bị có thể là một máy giả lập, một thiết bị thực tế hoặc một sandbox tùy chỉnh như là TaintDroid và DroidBox.



Hình 3.3. Kiến trúc tổng quan của DroidBot

Thành phần đầu tiên của DroidBox ở đây là mô đun Adapter dùng để cung cấp tính trừu tượng của thiết bị và ứng dụng kiểm thử. Nó đối phó với những vấn đề kỹ thuật ở mức độ thấp như là khả năng tương thích với các phiên bản Android khác nhau và các cỡ màn hình khác nhau, duy trì kết nối với thiết bị, gửi lệnh tới thiết bị và xử lý các kết quả lệnh, v.v…

Adapter cũng hoạt động như là một cầu nối giữa môi trường kiểm thử và thuật toán kiểm thử. Một mặt, nó theo dõi trạng thái của thiết bị và ứng dụng kiểm thử và chuyển đổi thông tin trạng thái sang dữ liệu có cấu trúc. Mặt khác, nó nhận được đầu vào kiểm thử được tạo ra bởi thuật toán và dịch chúng thành các lệnh. Với Adapter,

DroidBot có thể cung cấp một bộ các API mức cao dễ sử dụng cho người dùng để viết các thuật toán trong khi đảm bảo rằng các thuật toán này hoạt động trong các môi trường thử nghiệm khác nhau.

Mô đun Brain nhận thông tin của thiết bị và ứng dụng được tạo ra từ Adapter trong thời gian chạy và gửi các đầu vào kiểm thử được sinh ra đến Adapter. Việc sinh đầu vào kiểm thử được dựa trên một đồ thị trạng thái chuyển đổi được xây dựng trong quá trình diễn ra. Mỗi một nốt của đồ thị được đại diện cho một trạng thái thiết bị, trong khi cạnh giữa mỗi cặp nốt đại diện cho đầu vào kiểm thử đã kích hoạt quá trình chuyển đổi trạng thái.

3.2.3. Kiểm thử dựa trên mô hình với DroidBot

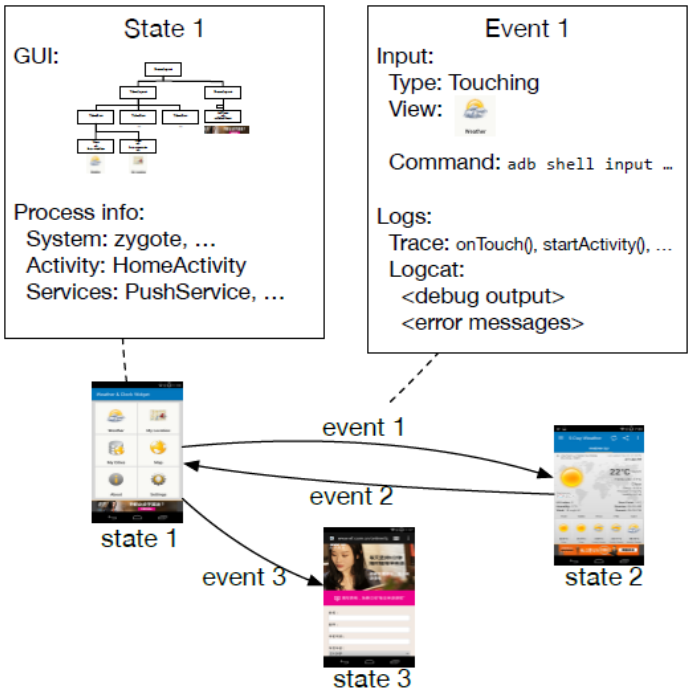
Bước 1: Mô hình hóa

DroidBot tìm nạp các thông tin của thiết bị/ ứng dụng từ thiết bị và gửi đầu vào kiểm thử tới thiết bị thông qua ADB. Để thực hiện được việc mô hình hóa, DroidBot lấy các thông tin GUI từ ứng dụng kiểm thử: đối với mỗi UI, DroidBot ghi lại ảnh chụp màn hình và cây phân cấp UI sử dụng UI Automator (đối với phiên bản SDK 16 trở lên) hoặc Hierarchy Viewer (đối với các phiên bản thấp hơn)

DroidBot sinh một mô hình của ứng dụng kiểm thử dựa trên thông tin được giám sát ngay trong thời gian chạy. Mô hình này nhằm giúp các thuật toán sinh đầu vào để đưa ra các lựa chọn đầu vào kiểm thử tốt hơn.

Hình 3.4 là một ví dụ của mô hình chuyển đổi trạng thái. Về cơ bản, mô hình này là một đồ thị trực tiếp, trong đó mỗi nốt đại diện cho một trạng thái của thiết bị, và mỗi cạnh giữa hai nốt đại diện cho một sự kiện đầu vào kiểm thử đã kích hoạt sự chuyển đổi trạng thái. Một nốt trạng thái thông thường chứa các thông tin về GUI và thông tin tiến trình đang chạy, và một cạnh sự kiện chứa các chi tiết của đầu vào kiểm thử và các phương thức/ log được kích hoạt bởi đầu vào.

Biểu đồ chuyển đổi trạng thái được xây dựng ngay lập tức. DroidBot duy trì thông tin về trạng thái hiện tại và giam sát sự thay đổi trạng thái sau khi gửi đi một đầu vào kiểm thử tới thiết bị. Một khi trạng thái của thiết bị được thay đổi, nó sẽ thêm đầu vào và trạng thái mới vào biểu đồ như là một cạnh mới và nốt mới.



Hình 3.4. Mô hình chuyển đổi trạng thái của DroidBot

Quá trình xây dựng biểu đồ dựa trên thuật toán so sánh trạng thái nền tảng. Hiện tại, DroidBot sử dụng so sánh dựa trên nội dung, trong đó hai trạng thái với các nội dung UI khác nhau được coi như là hai nốt khác nhau.

Bước 2: Lựa chọn yêu cầu kiểm thử

Ở phiên bản hiện tại 1.0.2b3, DroidBot tích hợp bốn thuật toán thăm dò khác nhau là naive depth-first, naive breadth-first, greedy depth-first và greedy breadth-first bên cạnh lựa chọn khám phá bằng Monkey. Nó cũng cho phép người dùng tích hợp các thuật toán riêng của họ hoặc sử dụng tập lệnh dành riêng cho ứng dụng để cải thiện chiến lược kiểm thử. Như vậy người dùng có thể tùy chọn một chiến lược khám phá UI phù hợp với yêu cầu kiểm thử của mình

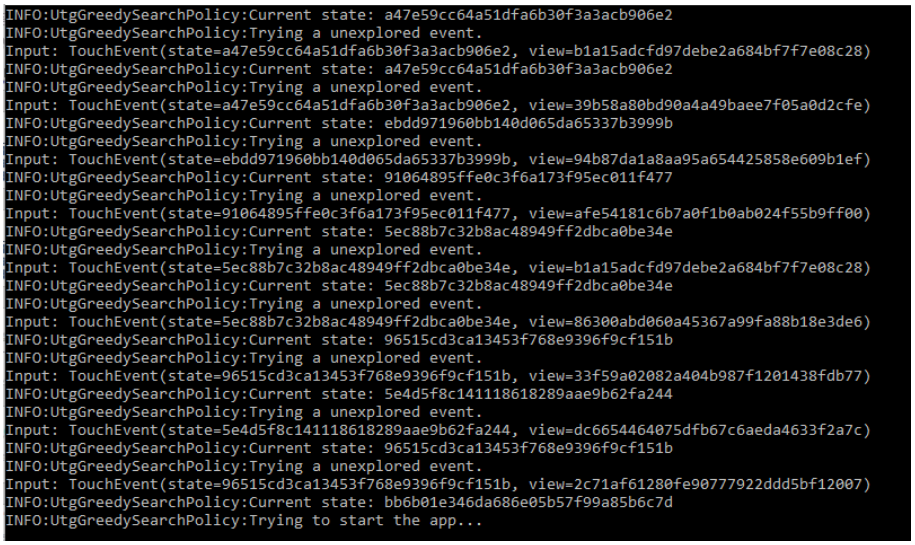
Bước 3: Sinh kiểm thử

Các loại đầu vào kiểm thử được DroidBot hỗ trợ bao gồm đầu vào UI (như là hành động chạm vào màn hình, cuộn một màn hình, v.v…), các intent (BOOT COMPLETED broadcast, v.v…), tài liệu tải lên (ảnh, tập văn bản, v.v…) và các dữ liệu cảm biến (tín hiệu GPS, v.v…). Chú ý rằng mô phỏng cảm biến chỉ được hỗ trợ bởi các máy giả lập. Các đầu vào kiểm thử này được sinh ra dựa trên mô hình UI được sinh ra trong bước trên.

DroidBot cung cấp một danh sách các API dễ sử dụng cho việc tìm nạp thông tin từ thiết bị và gửi đầu vào tới thiết bị. Ví dụ, lập trình viên có thể đơn giản gọi hàm device.dump_views () để lấy danh sách của các UI view và gọi hàm view.touch () để gửi đầu vào tới một view.

Bước 4: Cụ thể hóa kiểm thử

Để các sự kiện đầu vào được tạo ra ở bước trên có thể thực hiện được trên thiết bị kiểm thử, DroidBot tìm nạp các thông tin của thiết bị/ ứng dụng từ thiết bị và gửi đầu vào kiểm thử tới thiết bị thông qua ADB. Các sự kiện đầu vào ở trên sẽ được sinh ra dưới dạng các lệnh ADB để thiết bị có thể thực thi được



Hình 3.5: Các sự kiện sinh trong DroidBot

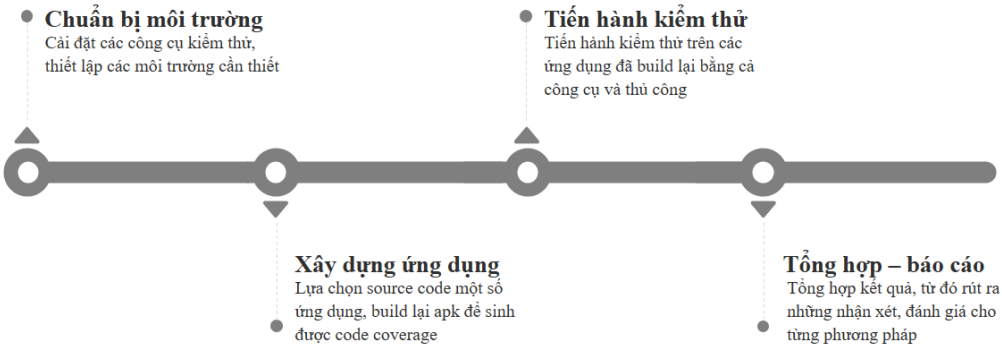
Bước 5: Thực thi kiểm thử

Các sự kiện sinh ra bởi DroidBot thông qua ADB sẽ được thực thi trên thiết bị kiểm thử. Đồng thời trong quá trình thực thi, các thông tin sẽ được giám sát và lưu lại:

- Thông tin tiến trình: DroidBot giám sát trạng thái tiến trình mức hệ thống sử dụng lệnh ps và giám sát trạng thái tiến trình mức ứng dụng sử dụng công cụ dumpsys trên Android
- Logs: logs bao gồm các dấu vết phương thức được kích hoạt bởi mỗi đầu vào kiểm thử và log được sinh ra bởi ứng dụng. Chúng có thể được lấy ra từ công cụ định hình Android và logcat.

Chương 4: Nghiên cứu thực nghiệm

Trong chương này, chúng ta tiến hành thực hiện một thực nghiệm nhỏ: kiểm thử một số ứng dụng Android với 2 công cụ kiểm thử tự động được giới thiệu ở chương 3 là Monkey và DroidBot và kiểm thử thủ công. Trong quá trình thực nghiệm sẽ tiến hành việc đo đạc các số liệu về thời gian thực hiện, số lỗi tìm được, độ bao phủ mã nguồn, từ đó giúp đưa ra được những ưu, nhược điểm của từng phương pháp. Các bước tiến hành thực nghiệm được thể hiện trong hình 4.1



Hình 4.1: Quy trình tiến hành thực nghiệm

4.1. Thiết lập môi trường thực nghiệm

4.1.1. Chuẩn bị công cụ kiểm thử

Về phía kiểm thử tự động, hai công cụ được lựa chọn cho thực nghiệm là Monkey đại diện cho kỹ thuật kiểm thử ngẫu nhiên/ kiểm thử mờ và DroidBot đại diện kỹ thuật kiểm thử dựa trên mô hình.

Kiểm thử thủ công: được thực hiện bởi người dùng.

Cài đặt Monkey:

Do Monkey được tích hợp trong bộ phát triển phần mềm Android (Android SDK) nên để chạy được Monkey trước hết cần tiến hành cài đặt Android SDK. Tiếp theo đó là cài đặt các biến môi trường:

Tạo biến ANDROID_HOME = ~/Android/Sdk

Tạo biến Path = %PATH%; %ANDROID_HOME%\tools; %ANDROID_HOME%\platform-tools

Cài đặt DroidBot

Cài đặt Python 2.7.14

Cài đặt các gói liên quan: - Androguard 3.0.1

- Networkx 2.0

- Pillow 4.3.0

Cài đặt Droidbot: sao chép mã nguồn từ: https://github.com/honeynet/droidbot. Thực hiện cài đặt Droidot bằng lệnh: pip install –e droidbot

Kiểm thử thủ công

Người dùng cài đặt ứng dụng trên thiết bị kiểm thử, tiến hành các thao tác trên ứng dụng một cách ngẫu nhiên theo các kịch bản của người dùng thông thường trong một khoảng thời gian 3 ~ 5 phút cho mỗi ứng dụng.

4.1.2. Chuẩn bị thiết bị kiểm thử

Sử dụng Samsung Galaxy Note 5 (N920I), hệ điều hành Android Nougat 7.0 để cài đặt các ứng dụng và tiến hành kiểm thử

4.2. Xây dựng ứng dụng kiểm thử

Bước 1: Tải mã nguồn ứng dụng:

STT	TÊN ỨNG DỤNG	LINK MÃ NGUỒN
1	AAT	https://f-droid.org/en/packages/ch.bailu.aat/
2	A Photo Manager	https://f-droid.org/en/packages/de.k3b.android.androFotoFinder/
3	AnyMemo	https://f-droid.org/en/packages/org.liberty.android.fantastischmemo/
4	Calculator	https://f-droid.org/en/packages/com.xlythe.calculator.material/
5	Camera	https://f-droid.org/en/packages/com.simplemobiletools.camera/
6	Catan Dice Game	https://f-droid.org/en/packages/com.ridgelineapps.resdicegame/
7	Clear List	https://f-droid.org/en/packages/douzifly.list/
8	FreeShisen	https://github.com/knilch0r/freeshisen
9	Giggity	https://f-droid.org/en/packages/net.gaast.giggity/
10	Glucosio	https://f-droid.org/en/packages/org.glucosio.android/
11	Good Weather	https://f-droid.org/en/packages/org.asdtm.goodweather/
12	Inbox Pager	https://f-droid.org/en/packages/net.inbox.pager/
13	Internet Radio	https://f-droid.org/en/packages/community.peers.internetradio/
14	Clip Stack	https://f-droid.org/packages/com.catchingnow.tinyclipboardmanager/

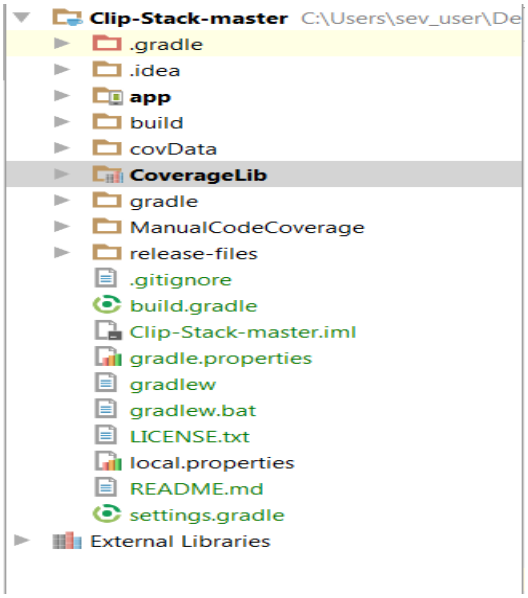
Bảng 4.1: Danh sách ứng dụng thực nghiệm

Danh sách các ứng dụng lựa chọn cho thực nghiệm được thể hiện ở bảng 4.1. Các ứng dụng này được lấy mã nguồn từ https://f-droid.org/ [27], là một cửa hàng ứng dụng Android mã nguồn mở.

Bước 2: Sử dụng Jacoco, build lại apk để đo độ bao phủ mã nguồn:

Sau khi đã có mã nguồn các ứng dụng, sử dụng Jacoco để xây dựng lại apk, phục vụ cho việc đo độ bao phủ mã nguồn. Các bước xây dựng lại apk sử dụng Jacoco và Gradle:

- Xây dựng một thư viện CoverageLib để lấy ra các tập tin .ec chứa thông tin bao phủ mã nguồn.



Hình 4.2: Thư viện CoverageLib

- Cấu hình tập tin build.gradle:

- Áp dụng Jacoco và bật tính năng đo độ bao phủ:

```
apply plugin: 'jacoco'
buildTypes {
   debug{
      testCoverageEnabled true
   }
}
```

- Tạo một task sinh báo cáo bao phủ mã nguồn

```
task jacocoTestReportAndroidTest(type: JacocoReport) {
   def coverageSourceDirs = [
         "${rootDir}/covData/src/main/java"
   ]
   group = "Reporting"
   description = "Generates Jacoco coverage reports"
   reports {
      csv.enabled true
```

```
        xml{
            enabled = true
            destination "${rootDir}/covData/reportcov/jacoco/jacoco.xml"
        }
        html{
            enabled true
            destination "${rootDir}/covData/reportcov/jacocoHtml"
        }
    }
    classDirectories = fileTree(
            dir: "${rootDir}/covData/classfiles",
            excludes: ['**/R.class',
                       '**/R$*.class',
                       '**/BuildConfig.*',
                       '**/Manifest*.*',
            ]
    )

    sourceDirectories = files(coverageSourceDirs)
    additionalSourceDirs = files(coverageSourceDirs)
    //poiter to coverage data
    executionData=fileTree("${rootDir}/covData/covfiles")
}
```

4.3. Tiến hành kiểm thử

Bước 1: Cài đặt các ứng dụng và lần lượt thực hiện kiểm tra tự động với Monkey và Droidbot, kiểm tra thủ công bởi người dùng.

STT	TÊN ỨNG DỤNG	CHẠY DROIDBOT			CHẠY MONKEY			KIỂM THỬ THỦ CÔNG
		1000	5000	10000	1000	5000	10000	3 ~ 5 phút
1	AAT	O			O			O
2	A Photo Manager	O			O			O
3	AnyMemo	O	O	O	O	O	O	O
4	Calculator	O	O	O	O	O	O	O
5	Camera	O			O	O	O	O
6	Catan Dice Game	O			O			O
7	Clear List	O	O	O	O	O	O	O
8	FreeShisen	O			O			O
9	Giggity	O			O			O
10	Glucosio	O	O	O	O	O	O	O
11	Good Weather	O			O	O	O	O
12	Inbox Pager	O			O	O	O	O
13	Internet Radio	O			O	O	O	O
14	Clip Stack	O	O	O	O	O	O	O

Ghi chú	O	Thực thi kiểm thử

Bảng 4.2: Danh sách ứng dụng thực thi kiểm thử

Với việc kiểm tra tự động, mỗi ứng dụng có thể được thực hiện một lần hoặc nhiều lần với số các sự kiện trong mỗi lần kiểm tra là khác nhau (1000 sự kiện, 5000 sự kiện, 10000 sự kiện). Với việc kiểm thử thủ công, mỗi ứng dụng sẽ được kiểm tra trong khoảng thời gian từ 3 ~ 5 phút, chi tiết như bảng 4.2.

Với Monkey, thực hiện chạy lệnh:

adb shell monkey -p <tên package> --throttle <số lượng sự kiện: 1000/5000/10000> --ignore-crashes --ignore-timeouts --ignore-security-exceptions -v <số lượng sự kiện> > log.txt

Với Droidbot, thực hiện chạy lệnh:

droidbot –a <đường dẫn apk> -o <đường dẫn sinh báo cáo> -count <số lượng sự kiện: 1000/5000/10000> -grant_perm

Kiểm thử thủ công: thao tác với ứng dụng kiểm tra bằng tay, thực hiện các kịch bản của người dùng thông thường một cách ngẫu nhiên, không theo các kịch bản có sẵn. Người dùng lần lượt khám phá các chức năng trong ứng dụng nhiều nhất có thể trong khoảng thời gian từ 3 ~ 5 phút.

Đồng thời với quá trình kiểm tra bằng Monkey, Droidbot và kiểm tra thủ công, chạy lệnh để lấy tập tin .ec sau mỗi khoảng thời gian 30 giây:

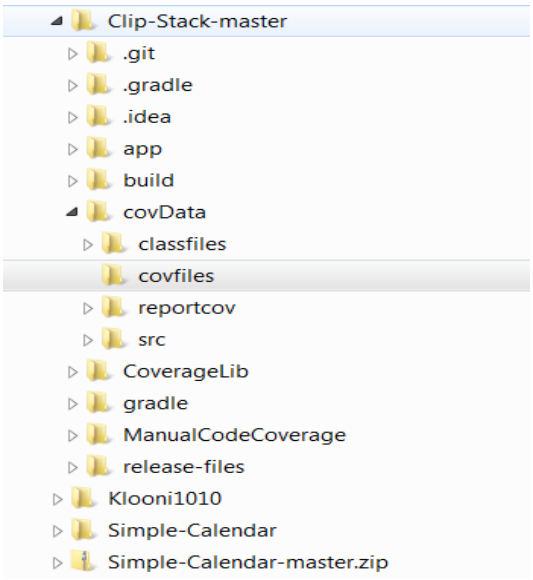
```
:lable
timeout -t 30
adb shell am broadcast -a SQA.COM.ACTION.GETCOVERAGE.DATA
goto lable
```

Bước 2: lấy tập tin .ec và sinh báo cáo cho độ bao phủ mã nguồn

- Các tập tin .ec được sinh ra sau mỗi lần lệnh adb shell am broadcast -a SQA.COM.ACTION.GETCOVERAGE.DATA được thực hiện. Các tập tin này được lưu trong thiết bị, trong một thư mục được có tên chính là tên của gói (package) tương ứng mà tập tin .ec được sinh ra

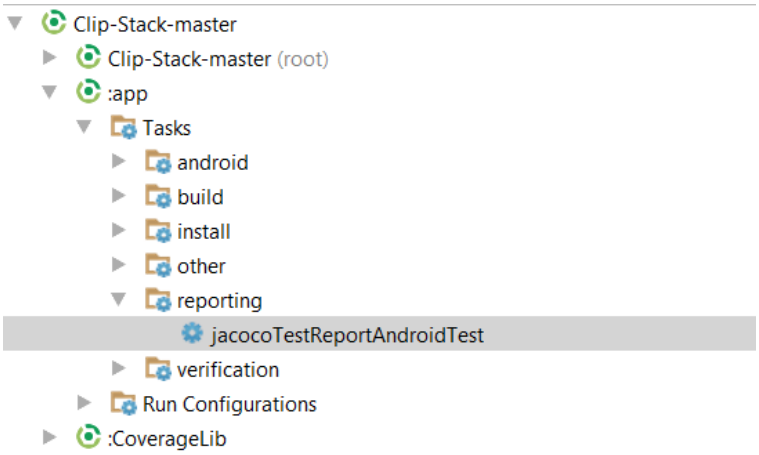
Hình 4.3: Thư mục chứa các tập tin .ec

- Các tập tin .ec được đưa vào thư mục covfiles được tạo ra trước đó



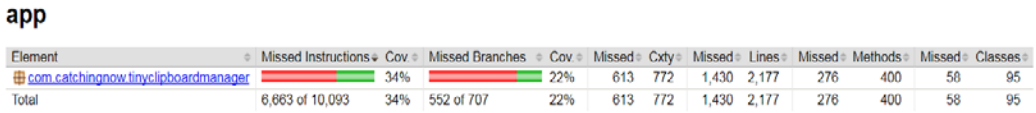
Hình 4.4: Cây thư mục covData

- Chạy lệnh reporting ở Gradle trong Android Studio để sinh ra báo cáo



Hình 4.5: Sinh báo cáo bao phủ mã nguồn trong Gradle

- Sau khi Gradle hoàn thành việc chạy build, báo cáo mức độ bao phủ sẽ được sinh ra trong thư mục reportcov, cho ta các thông tin về kết quả bao phủ của mã nguồn:


app

Element	Missed Instructions	Cov	Missed Branches	Cov	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
com.catchingnow.tinyclipboardmanager		34%		22%	613	772	1,430	2,177	276	400	58	95
Total	6,663 of 10,093	34%	552 of 707	22%	613	772	1,430	2,177	276	400	58	95

Hình 4.6: Báo cáo bao phủ mã nguồn

Bước 3: Tổng hợp số liệu và phân tích kết quả

- Phân tích các các tập tin log, thống kê số lượng lỗi tìm được
- Tổng hợp số liệu kết quả đo độ bao phủ

4.4. Kết quả thực nghiệm

Sau quá trình kiểm tra các ứng dụng bằng hai công cụ DroidBot, Monkey và kiểm tra thủ công, các kết quả thu được từ thực nghiệm này như sau:

Thời gian kiểm tra các ứng dụng trung bình được thể hiện ở bảng 4.3

PHƯƠNG THỨC KIỂM THỬ	DROIDBOT			MONKEY			KIỂM THỬ THỦ CÔNG
	1000	5000	10000	1000	5000	10000	
THỜI GIAN TRUNG BÌNH	117 phút	610 phút	1210 phút	3.55 phút	14.04 phút	28.69 phút	3 ~ 5 phút

Bảng 4.3: Thời gian thực thi kiểm thử

Số lượng lỗi crash phát hiện được trong từng phương thức kiểm thử thể hiện ở bảng 4.4

STT	TÊN ỨNG DỤNG	DROIDBOT			MONKEY			KIỂM THỬ THỦ CÔNG
		1000	5000	10000	1000	5000	10000	3 ~ 5 phút
1	AAT	O			O			O
2	A Photo Manager	O			O			O
3	AnyMemo	O	O	O	O	O	O	O
4	Calculator	O	O	O	O	O	O	O
5	Camera	O			O	O	O	O
6	Catan Dice Game	O			O			O
7	Clear List	O	O	O	O	O	O	O
8	FreeShisen	O			O			O

9	Giggity	O			O			O
10	Glucosio	O	O	O	O	O	O	O
11	Good Weather	O			O	O	O	O
12	Inbox Pager	O			O	O	O	O
13	Internet Radio	O			O	O	O	O
14	Clip Stack	O	O	O	O	O	O	O

Thực thi kiểm thử	O
Có lỗi xảy ra	

Bảng 4.4: Danh sách số lượng lỗi crash

Mức độ bao phủ mã nguồn của từng phương thức kiểm thử:

TÊN ỨNG DỤNG	DROIDBOT					
	1000		5000		10000	
	Instructions	Branches	Instructions	Branches	Instructions	Branches
AAT	18%	10%				
A Photo Manager	10%	6%				
AnyMemo	26%	18%	26%	18%	40%	26%
Calculator	32%	19%	36%	21%	38%	23%
Camera	46%	33%				
Catan Dice Game	77%	25%				
Clear List	47%	23%	47%	23%	47%	22%
FreeShisen	30%	11%				
Giggity	50%	38%				
Glucosio	10%	7%	14%	10%	13%	8%
Good Weather	31%	16%				
Inbox Pager	7%	2%				
Internet Radio	92%	74%				
Clip Stack	61%	44%	62%	45%	67%	51%
Trung bình	**38%**	**23%**	**37%**	**23%**	**41%**	**26%**

Bảng 4.5: Độ bao phủ mã nguồn của DroidBot

TÊN ỨNG DỤNG	MONKEY					
	1000		5000		10000	
	Instructions	Branches	Instructions	Branches	Instructions	Branches
AAT	31%	20%				
A Photo Manager	18%	11%				
AnyMemo	12%	9%	26%	18%	19%	14%
Calculator	39%	27%	51%	38%	57%	45%
Camera	45%	32%	55%	38%	56%	42%
Catan Dice Game	85%	51%				
Clear List	44%	15%	43%	15%	53%	28%
FreeShisen	66%	54%				
Giggity	51%	37%				
Glucosio	9%	6%	9%	7%	19%	12%
Good Weather	20%	8%	63%	35%	69%	41%

Inbox Pager	2%	0%	2%	1%	13%	5%
Internet Radio	37%	13%	92%	73%	94%	78%
Clip Stack	43%	32%	49%	40%	65%	52%
Trung bình	**36%**	**23%**	**43%**	**29%**	**49%**	**35%**

Bảng 4.6: Độ bao phủ mã nguồn của Monkey

TÊN ỨNG DỤNG	KIỂM THỬ THỦ CÔNG	
	3 ~ 5 phút	
	Instructions	Branches
AAT	48%	32%
A Photo Manager	35%	24%
AnyMemo	35%	23%
Calculator	55%	40%
Camera	65%	45%
Catan Dice Game	88%	54%
Clear List	54%	33%
FreeShisen	90%	75%
Giggity	21%	11%
Glucosio	29%	16%
Good Weather	75%	48%
Inbox Pager	12%	5%
Internet Radio	94%	78%
Clip Stack	69%	54%
Trung bình	**55%**	**38%**

Bảng 4.7: Độ bao phủ mã nguồn của kiểm thử thủ công

4.5. Phân tích – đánh giá

4.4.1. Tính hiệu quả trong việc phát hiện lỗi

Dựa trên số liệu lỗi ở bảng 4.4 ta có thể thấy cả hai công cụ Droibot và Monkey đều khá hiệu quả trong việc tìm ra lỗi so với kiểm thử thủ công. Tuy nhiên Monkey khi chạy với số lượng sự kiện lớn hơn thì việc phát hiện ra lỗi cũng cao hơn so với DroidBot.

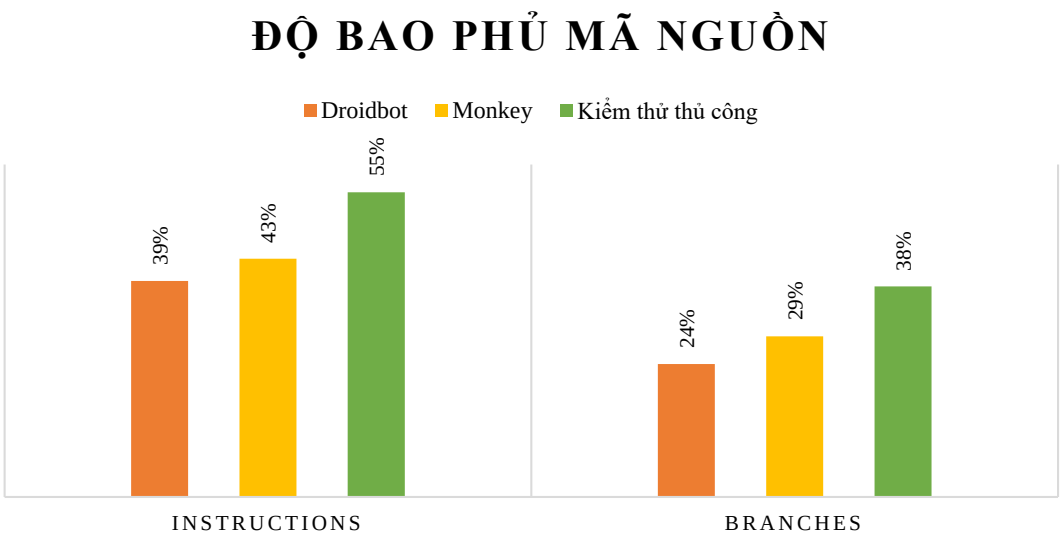
4.4.2. Tính hiệu quả trong chiến lược khám phá

Monkey sinh các sự kiện một cách ngẫu nhiên, không theo một luồng nhất định. Do vậy, tuy thực hiện chạy với số lượng sự kiện nhỏ thì nó vẫn có khả năng khám phá nhiều chức năng khác nhau trong ứng dụng. Trong khi đó DroidBot sinh các sự kiện dựa trên mô hình UI, sử dụng chiến lược tham ăn và thuật toán duyệt theo chiều rộng, do đó các sự kiện được sinh ra lần lượt theo những luồng nhất định. Chính vì vậy khi thực thi với số lượng sự kiện nhỏ thì khả năng bao phủ mã nguồn của Monkey sẽ tốt hơn so với DroidBot.

Mặc dù vậy, các công cụ tự động đều có hạn chế khi gặp phải những giao diện có trường nhập thông tin chứa các yêu cầu đặc biệt, hoặc những giao diện mà các thành phần thông tin không hiển thị sẵn trên màn hình, cần phải qua các thao tác gạt sang phải/ trái hoặc lên/ xuống để hiển thị. Trong các trường hợp này, cả hai công cụ đều gặp khó khăn để vượt qua, thậm chí là mắc kẹt tại đó.

Hình 4.7: Màn hình bắt đầu của ứng dụng quản lý bệnh tiểu đường

Hình 4.7 là giao diện bắt đầu của một ứng dụng quản lý bệnh tiểu đường. Chỉ khi nhập các thông tin đầy đủ và hợp lệ, chạm vào "GET STARTED" mới có thể bắt đầu sử dụng ứng dụng. Tuy nhiên, vấn đề xảy ra ở đây là trường thông tin nhập tuổi chỉ hợp lệ khi số nhập vào nhỏ hơn 100. Điều này hoàn toàn gây khó khăn cho cả hai công cụ tự động, bởi nếu không may mắn để nhập được thông tin về tuổi hợp lệ, thì việc kiểm tra sẽ bị tắc tại đây và không thể khám phá ứng dụng sâu thêm được nữa. Chính vì những hạn chế này mà ta có thể thấy kết quả bao phủ mã nguồn của cả hai công cụ tự động đều thấp hơn so với việc kiểm thử thủ công.



Biểu đồ 4.1: Độ bao phủ mã nguồn

4.4.3. Tính khả dụng

Cả Monkey và DroidBot đều là các công cụ chạy bằng dòng lệnh, việc cài đặt và sử dụng không quá phức tạp. Tuy nhiên, với cùng một số lượng sự kiện, thời gian thực hiện của DroidBot lớn hơn rất nhiều so với Monkey. Có sự chênh lệch quá lớn này một phần vì với mỗi sự kiện, DroidBot sẽ lưu lại kịch bản thực hiện, ảnh chụp màn hình và lưu lại các luồng giao diện đã đi qua. Mặc dù vậy thì với khoảng thời gian phải bỏ ra quá nhiều như hiện tại, hiệu suất của DroidBot sẽ là chưa thực sự tốt so với Monkey

Kết luận

Sau quá trình nghiên cứu và tìm hiểu về đề tài "Nghiên cứu một số phương pháp sinh đầu vào kiểm thử tự động cho Android", các kết quả mà luận văn đã đạt được là:

Đầu tiên, luận văn đã giúp đưa ra một cái nhìn tổng quan về kiểm thử tự động dành cho phần mềm nói chung và kiểm thử tự động cho các ứng dụng Android nói riêng.

Từ cái nhìn tổng quan về kiểm thử tự động, luận văn đã giúp đưa ra khái niệm chi tiết hơn về sinh đầu vào kiểm thử tự động là gì cùng với các kỹ thuật phổ biến đang được sử dụng để sinh đầu vào kiểm thử tự động: phương pháp kiểm thử Fuzz và phương pháp kiểm thử dựa trên mô hình. Đưa ra các ưu, nhược điểm của các phương pháp này để từ đó giúp người đọc có được những đánh giá, so sánh và đưa ra lựa chọn một phương pháp phù hợp cho mục đích sử dụng của mình. Luận văn cũng đã đưa ra những tìm hiểu về một số hướng tiếp cận các phương pháp trên áp dụng cho các ứng dụng Android

Để có cái nhìn cụ thể và chi tiết hơn về hai phương pháp sinh đầu vào kiểm thử tự động được trình bày ở trên, luận văn đã lựa chọn hai công cụ tự động tiêu biểu tương ứng cho hai phương pháp là DroidBot và Monkey để tìm hiểu. Bên cạnh việc tìm hiểu về lý thuyết, đã tiến hành làm thực nghiệm để so sánh với nhau đồng thời cũng so sánh với việc kiểm thử thủ công. Sau thực nghiệm đã thu được kết quả về số lượng lỗi, độ bao phủ mã nguồn, thời gian thực thi của mỗi công cụ. Từ những kết quả thu được đó đã giúp đưa ra được những so sánh, phân tích và đánh giá cho tính hiệu quả của từng phương pháp kiểm thử.

Tuy nhiên luận văn còn có hạn chế trong việc tiến hành thực nghiệm: số lượng các công cụ kiểm thử còn hạn chế, số lượng ứng dụng lựa chọn chưa phong phú

Với những hạn chế nêu trên, một số hướng mở rộng nghiên cứu và tìm hiểu trong tương lai:

- Mở rộng thực nghiệm với số lượng các công cụ lựa chọn lớn hơn, tiến hành kiểm tra với số lượng ứng dụng nhiều hơn và có độ phức tạp cao hơn, đồng thời kiểm tra với số lượng sự kiện lớn hơn nữa.

- Có kế hoạch cho việc lựa chọn các công cụ thích hợp để cải tiến và phát triển, để áp dụng thực tế vào công việc kiểm thử phần mềm cho các ứng dụng Android tại Samsung.

Tài liệu tham khảo

[1] "Statista," [Online]. Available: https://www.statista.com/statistics/266136/global-market-share-held-by-smartphone-operating-systems/.

[2] "Android Developer," [Online]. Available: https://developer.android.com/guide/platform/index.html.

[3] "Android Developer," [Online]. Available: https://developer.android.com/reference/android/app/Activity.html.

[4] "Android Developer," [Online]. Available: https://developer.android.com/guide/components/services.html.

[5] "Android Developer," [Online]. Available: https://developer.android.com/guide/topics/manifest/receiver-element.html.

[6] "Android Developer," [Online]. Available: https://developer.android.com/guide/topics/providers/content-providers.html.

[7] Abel Méndez-Porras, Christian Quesada-López, and Marcelo Jenkins, "Automated Testing of Mobile Applications: A Systematic Map and Review," p. 1.

[8] Hitesh Tahbildar, Bichitra Kalita, "Automated software test data generation: Direction of research," in *International Journal of Computer Science & Engineering Survey (IJCSES) Vol.2*, 2011, pp. 2-3.

[9] WANG Tao, LI Yanling, MA Yingli, GUO Wei, "Research and Application of a New Fuzz-test Framework," p. 2.

[10] "VIBLO," [Trực tuyến]. Available: https://viblo.asia/p/tim-hieu-ve-fuzz-testing-YWOZrDzv5Q0.

[11] "Guru99," [Online]. Available: https://www.guru99.com/fuzz-testing.html.

[12] P. Garg, "Fuzzing - mutation vs generation".*INFOSEC Institute.*

[13] "Tutorials point," [Online]. Available: https://www.tutorialspoint.com/software_testing_dictionary/fuzz_testing.htm.

[14] "Khoa CNTT, Đại học Duy Tân," [Trực tuyến]. Available: http://kcntt.duytan.edu.vn/Home/ArticleDetail/vn/128/2461/bai-01-so-luoc-ve-fuzzing-testing.

[15] "OWASP," [Online]. Available: https://www.owasp.org/index.php?title=File:CLASP_Vulnerabilities_SDLC_Phases.gif&setlang=en.

[16] "Guru99," [Online]. Available: https://www.guru99.com/fuzz-testing.html.

[17] R. T. M. N. Aravind MacHiry, "Dynodroid: An Input Generation System for Android Apps".

[18] J. R. Raimondas Sasnauskas, "Intent Fuzzer: Crafting Intents of Death".

[19] ANTOJOSEPH, "DROID-FF – THE ANDROID FUZZING FRAMEWORK".

[20] "Guru99," [Online]. Available: https://www.guru99.com/model-based-testing-tutorial.html.

[21] Zoltan Micskei, Istvan Majzik, "Model-based test generation," *Software and Systems Verification (VIMIMA01)*, pp. 13-17.

[22] E. Karaman, "Model Based Software Testing," *SWE550 Boğaziçi University,* pp. 7-17.

[23] P. Ana, "Model – based testing," *MDSE – Model Driven Software Engineering*, pp. 3-12.

[24] Shauvik Roy Choudhary, Alessandra Gorla, Alessandro Orso, "Automated Test Input Generation for Android: Are We There Yet?".

[25] "Android Developer," [Online]. Available: https://developer.android.com/studio/test/monkey.html.

[26] Yuanchun Li, Ziyue Yang, Yao Guo, Xiangqun Chen, "DroidBot: A Lightweight UI-Guided Test Input Generator for Android".

[27] [Online]. Available: https://f-droid.org/.