الجمهورية العربية السورية
جامعة البعث
كلية الهندسة المعلوماتية
قسم هندسة البرمجيات ونظم المعلومات

توليد حالات اختبار باستخدام تقنيات الذكاء الصنعي

دراسة أعدت لنيل درجة الدكتوراه في كلية الهندسة المعلوماتية باختصاص

هندسة البرمجيات ونظم المعلومات

إعداد
م. بشرى طلال غزاله

إشراف
د. يسر السيد سليمان الأتاسي

أستاذة في قسم هندسة البرمجيات ونظم المعلومات

كلية الهندسة المعلوماتية

جامعة البعث

حمص 2021 م-1442 هـ

الملخص

تطبيقات Android هي برمجيات تعمل على الأجهزة الخاصة بأنظمة التشغيل Android، تتطور وتنمو هذه التطبيقات من ناحية عددها وعدد المستخدمين وعدد المطورين لهذه التطبيقات، عام 2016 وصل عدد تطبيقات Android التي تم تطويرها لـ 2 مليون، وتجاوز عددها خلال شهر آذار عام 2017 الـ 2.8 مليون تطبيق وحالياً عدد التطبيقات تجاوز الـ 3.04 مليون تطبيقات حسب إحصائيات Google Play الذي يعتبر متجر التطبيقات الرسمي للأجهزة المعتمدة على نظام تشغيل Android، الانتشار الواسع لتطبيقات Android فتح العديد من التحديات أمام الباحثين في مجال هندسة البرمجيات لإنشاء برمجيات متينة، من التحديات الأساسية إنشاء تطبيقات ذات جودة وخالية من الأخطاء وتوافق متطلبات المستخدمين، لذلك يجب اختبار هذه التطبيقات والتأكد من صحة تنفيذها وخلوها من الأخطاء قبل نشرها للمستخدمين، تطبيقات Android لها خصائص وميزات خاصة تجعل من عملية اختبارها مختلفة عن التطبيقات التقليدية وبالتالي تحتاج لآليات وأدوات وطرائق جديدة للاختبار.

عملية اختبار البرمجيات تستهلك نحو 40% إلى 70% من كلفة وجهد ووقت تطوير البرمجية، المرحلة الأهم في عملية الاختبار هي توليد حالات اختبار تعكس حالة النظام البرمجي وتبين كل حالات تنفيذه وتساعد على اكتشاف الأخطاء والمشاكل في البرمجية، فعالية عملية الاختبار تعتمد على عدد الأخطاء التي وُجِدَت وتم تصحيحها قبل نشر النظام، وهذا بدوره يعتمد على نوعية وفعالية حالات الاختبار التي وُلدت، حالات الاختبار يجب أن تكون فعالة بحيث تغطي كل حالات تنفيذ البرمجية، وبحيث نمنع التكرار في حالات الاختبار.

عدة آليات لتوليد حالات اختبار لتطبيقات Android حسب المنهجية المستخدمة:

- توليد حالات الاختبار بشكل عشوائي
- توليد حالات الاختبار اعتماداً على النموذج
- توليد حالات الاختبار اعتماداً على البحث

تم اقتراح الأداة BeeDroid التي دمجت أسلوبي الاختبار المعتمد على البحث والمعتمد على النموذج، من خلال تطوير الأداة EGator لتحليل تطبيقات Android لاستخراج نموذج يمثل التطبيق ومكوناته كبيان، ثم تمرير هذا البيان لخوارزمية النحل الصنعية لاشتقاق حالات الاختبار، ثم قمنا بتقييم الأداة حسب نسبة تغطية شيفرة التطبيق ووجدنا أن الخوارزمية المقترحة قدمت تغطية أعلى من الأدوات السابقة وكذلك بيّنا قدرة الأداة على اكتشاف الأخطاء في التطبيقات باستخدام اختبار الطفرات.

الكلمات المفتاحية: اختبار تطبيقات Android – توليد حالات اختبار – الاختبار المعتمد على البحث.

ABSTRACT

Android applications are software that works on devices of the Android operating systems. These applications develop and grow in terms of their number, the number of users, and the number of developers for these applications. In 2016, the number of Android applications that were developed reached 2 million, and during March 2017, the number exceeded 2.8 million applications. Currently, the number of applications has exceeded 3.04 million applications, according to the statistics of Google Play, which is the official application store for devices based on the Android operating system. The wide spread of Android applications has opened many challenges for researchers in the field of software engineering to create solid software, one of the main challenges is to create quality and free applications These applications must be informed and ensure that they are implemented correctly and are free of errors before publishing them to users.

Android applications have special characteristics and features that make their testing process different from traditional applications and therefore need new mechanisms, tools and methods for testing.

The software testing process consumes about 40% to 70% of the cost, effort and time of software development. The most important stage in the testing process is to generate test cases that reflect the state of the software system, show all cases of its implementation, and help discover errors and problems in the software. The effectiveness of the testing process depends on the number of errors that are found and debugged before the system is deployed, and this in turn depends on the quality and effectiveness of the test cases that are generated. Test cases must be effective so that they cover all cases of software execution, and so that we prevent redundancy in test cases.

Several mechanisms for generating test cases for Android apps depending on the methodology used:

- Random-Based Testing
- Model-Based Testing
- Search-Based Testing

BeeDroid tool was proposed, which combined the two search-based and model-based testing methods, by developing the EGator tool to analyze Android applications to extract a model representing the application and its components as a graph, then pass this graph to the artificial bee algorithm to derive the test cases, we evaluated the tool according to the percentage of coverage of the application code and found that the proposed algorithm provided higher coverage than previous tools and also demonstrated the tool's ability to detect errors in applications using mutation testing.

Keywords: Android application testing - Generating Test Cases- Search Based Testing.

الإنتاج العلمي

تم خلال هذا البحث نشر ورقتين بحثيتين في مجلة جامعة البعث:

1- توليد حالات اختبار لتطبيقات واجهات المستخدم الرسومية باستخدام خوارزمية النحل الصنعية، في المجلد 43 لعام 2021

2- تحليل تطبيقات Android لبناء نموذج يمثل مكونات التطبيق وآلية تفاعلها فيما بينها، في المجلد 43 لعام 2021

فهرس المحتويات

الملخص ..... **I**

ABSTRACT ..... **III**

الإنتاج العلمي ..... **V**

الفهرس ..... **VII**

قائمة الأشكال ..... **XIII**

قائمة الجداول ..... **XV**

قائمة المصطلحات ..... **XVII**

الفصل الأول: المقدمة **1**

1-1 مقدمة 1

1-2 مشكلة البحث 3

1-3 الهدف من البحث 3

1-4 مبررات البحث 4

1-5 خطة البحث 5

1-6 مخطط الأطروحة 6

الفصل الثاني: اختبار البرمجيات **7**

2-1 مقدمة 7

2-2 مراحل عملية اختبار البرمجيات TESTING PROCESS 7

2-3 حالة الاختبار TEST CASE 7

2-4 تغطية الاختبار TEST COVERAGE 8

2-5 معايير الاختبار 8

2-6 ORACLE VERIFIER 8
2-7 مجموعة الاختبار TEST SUITE 9
2-8 اختبار الطفرة MUTATION TESTING 9
2-9 مستويات الاختبار TESTING LEVELS 10
2-9-1 اختبار الوحدة Unit Testing 10
2-9-2 اختبار التكامل Integration Testing 10
2-9-3 اختبار النظام System Testing 11
2-9-4 اختبار القبول Acceptance Testing 11
2-10 طرائق الاختبار TESTING METHODS 12
2-10-1 اختبار الصندوق الأسود Black Box Testing 12
2-10-2 اختبار الصندوق الأبيض White Box Testing 13
2-10-3 اختبار الصندوق الرمادي Gray Box Testing 13
2-11 أنواع الاختبار TESTING TYPES 13
2-11-1 الاختبار اليدوي Manual Testing 14
2-11-2 الاختبار الآلي Automated Testing 14
2-12 آليات توليد حالات الاختبار بشكل أوتوماتيكي 14
2-12-1 توليد حالات الاختبار بشكل عشوائي 14
2-12-2 توليد حالات الاختبار اعتماداً على النموذج 15
2-12-3 توليد حالات الاختبار اعتماداً على البحث 15
2-13 خاتمة 16
الفصل الثالث: خوارزميات البحث الأمثل 17
3-1 مقدمة 17
3-2 الخوارزميات االتطورية EVOLUTIONARY ALGORITHMS 18

3-3 مراحل الخوارزميات التطورية..... 19

3-4 استخدام الخوارزميات التطورية في مجال اختبار البرمجيات..... 19

3-5 الخوارزمية الجينية..... 21

3-6 خوارزميات السرب أو ذكاء السرب..... 24

3-6-1 خوارزمية النحل الصنعية 24

3-7 خاتمة 28

الفصل الرابع: توليد حالات اختبار لتطبيقات واجهات المستخدم الرسومية GUI 29

4-1 مقدمة 29

4-2 اختبار تطبيقات GUI..... 30

4-3 الدراسة المرجعية..... 31

4-4 خطوات الخوارزمية المقترحة توليد حالات اختبار تطبيقات GUI 33

4-4-1 مخطط تدفق الأحداث EVENTS FLOW GRAPH 34

4-4-2 استخدام خوارزمية النحل الصنعية..... 36

4-4-2-1 إنتاج مواقع مصادر الغذاء الأولية..... 36

4-4-2-2 إرسال النحل العامل إلى مصادر الغذاء 37

4-4-2-3 تقييم الحل من قبل النحل المشاهد على أساس المعلومات المقدمة من النحل العامل 38

4-4-2-2 استخدام النحل الكشاف..... 38

4-5 القسم العملي..... 38

4-6 خاتمة 42

الفصل الخامس: توليد حالات اختبار لتطبيقات ANDROID 43

5-1 مقدمة 43

5-2 خصائص تطبيقات ANDROID 45

5-2-1 مكونات تطبيقات Android 47

5-2-1-1 النشاط Activity 47

5-2-1-2 الخدمات Services 47

5-2-1-3 مستقبل البث Broadcast Receiver 47

5-2-1-4 مزوّد المحتوى Content Provider 47

5-2-2 تحليل واجهات المستخدم 49

5-2-3 سماحيات تطبيقات Android 49

5-2-4 التفاعل بين مكونات التطبيق وأزرار الجهاز 50

5-3 اختبار تطبيقات ANDROID 50

5-4 الدراسة المرجعية 52

5-4-1 الدراسات المتعلقة بعملية تحليل تطبيقات Android 53

5-4-2 الدراسات المتعلقة بعملية توليد حالات اختبار لتطبيقات Android 55

5-5 الآلية المقترحة لبناء الأداة BEEDROID لتوليد حالات اختبار لتطبيقات ANDROID 56

5-5-1 الآلية المتبعة لبناء نموذج يمثل تطبيقات Android 57

5-5-1-1 الخدمات Services 57

5-5-1-2 مستقبل البث Broadcast Receiver 59

5-5-1-3 مزوّد المحتوى Content Provider 60

5-5-1-4 تحليل ملف AndroidManifest.xml لاستخراج السماحيات المطلوبة 61

5-5-1-5 التطبيق العملي 62

5-5-2 استخدام خوارزمية النحل الصنعية لتوليد حالات اختبار لتطبيقات Android 65

5-6 خاتمة 67

الفصل السادس: تقييم الأداة BEEDROID 69

6-1 مقدمة 69

6-2 التطبيقات المستخدمة للتقييم 69
6-3 أدوات اختبار تطبيقات ANDROID 72
6-3-1 الأداة Emma 72
6-3-2 الأداة Robotium 72
6-3-3 الأداة MuDroid 72
6-4 قياس تغطية الشيفرة البرمجية 73
6-4-1 مقارنة الأداة BeeDroid مع الأداة Monkey 73
6-4-2 مقارنة الأداة BeeDroid مع الأداة MobiGUIATR 75
6-4-3 مقارنة الأداة BeeDroid مع الأداة EvoDroid 77
6-5 اختبار الطفرات MUTATION TESTING 78
6-5-1 تعريف عام باختبار الطفرات 78
6-5-2 استخدام اختبار الطفرات مع تطبيقات Android 80
6-6 خاتمة 84
الفصل السابع: النتائج والتوصيات 85
7-1 مقدمة 85
7-2 النتائج 85
7-3 التوصيات المقترحة 87
المراجع العلمية 89

قائمة الأشكال

الشكل 1-1 عدد تطبيقات Android خلال عامي 2019 و2020 5

الشكل 1-2 مستويات الاختبار 12

الشكل 2-2 اختبار الصندوق الأسود 12

الشكل 3–1 المخطط العام لعمل الخوارزميات التطورية 19

الشكل 3–2 تصنيف خوارزميات البحث 20

الشكل 3–3 يمثل مخطط تدفق الخوارزمية الجينية 22

الشكل 3–4 عبور عند نقطة واحدة في الخوارزمية الجينية 23

الشكل 3–5 عبور عند نقطتين في الخوارزمية الجينية 23

الشكل 3–6 عبور القطع والوصل في الخوارزمية الجينية 23

الشكل 3–7 خطوات خوارزمية النحل الصنعية 26

الشكل 4–1 واجهة لإدخال بيانات مستخدم 31

الشكل 4–2 تسلسل تنفيذ خوارزمية اختبار تطبيقات GUI 33

الشكل 4–3 واجهة تطبيق لتحرير النصوص مع مخطط الأحداث لها 34

الشكل 4–4 كيفية حساب قيمة CC اعتماداً على مصفوفة تمثل تدفق الأحداث لتطبيق 36

الشكل 4–5 تطبيقات GUI المستخدمة لمقارنة تنفيذ خوارزميتي النحل والجينية 40

الشكل 5–1 عدد مستخدمي أجهزة الهواتف الذكية بين عامي 2012 و2020 43

الشكل 5–2 أنظمة تشغيل أجهزة الموبايل المستخدمة بين عامي 2012 و2020 44

الشكل 5–3 عدد تطبيقات Android خلال عامي 2019 و2020 45

الشكل 5–4 تحويل Java Bytecode لتنسيق dex 46

الشكل 5–5 الفرق بين ملفات class. وملفات java. 46

الشكل 5–6 دورة حياة النشاط 48

الشكل 5–7 مثال عن Broadcast 48

الشكل 5-8 السماحيات المطلوبة لتطبيق Facebook 49
الشكل 5-9 أزرار أجهزة Android 50
الشكل 5-10 عملية اختبار تطبيقات Android 51
الشكل 5-11 المخطط العام لتوليد حالات اختبار تطبيقات Android باستخدام خوارزميات البحث الأمثلية 52
الشكل 5-12 دورة حياة الخدمة 58
الشكل 5-13 إدارة الوصول للبيانات عن طريق مزود المحتوى 60
الشكل 5-14 آلية عمل استخدام السماحيات 61
الشكل 5-15 تطبيق example 63
الشكل 5-16 عقد التطبيق example 63
الشكل 5-17 بعض حواف البيان الممثل للتطبيق example 64
الشكل 5-18 السماحيات ومستقبلات البث الغير محلية المطلوبة للتطبيق example 65
الشكل 5-19 تسلسل تنفيذ الأداة BeeDroid المقترحة 66
الشكل 6-1 مراحل اختبار تطبيق Android باستخدام الأداة Robotium 72
الشكل 6-2 مقارنة تغطية الأداتين BeeDroid وMonkey 75
الشكل 6-3 مقارنة تغطية الأداتين BeeDroid و وMobiGUITAR 76
الشكل 6-4 مقارنة تغطية الأداتين BeeDroid وEvoDroid 78
الشكل 6-5 خطوات اختبار الطفرات 79
الشكل 6-6 مثالاً يبين اختبار الطفرات 79
الشكل 6-7 مثالاً يبين حالات اختبار الطفرات مصممة للتطبيق في الشكل السابق 80
الشكل 6-8 خطوات عمل الأداة MuDroid 81
الشكل 6-9 مثالاً يبين اختبار الطفرات لتطبيقات Android 82

قائمة الجداول

الجدول 4-1 مقارنة تنفيذ خوارزميتي ABC وGA 41

الجدول 6-1 التطبيقات المستخدمة لتقييم الأداة BeeDroid 71

الجدول 6-2 مقارنة تغطية الشيفرة البرمجية للأداتين BeeDroid وMonkey 74

الجدول 6-3 مقارنة تغطية الشيفرة البرمجية للأداتين BeeDroid وMobiGUITAR 76

الجدول 6-4 مقارنة تغطية الشيفرة البرمجية للأداتين BeeDroid وEvoDroid 77

الجدول 6-5 قواعد الطفرات المستخدمة 82

الجدول 6-6 قواعد الطفرات المطبقة على التطبيقات 83

الجدول 6-7 نتيجة تطبيق قواعد الطفرات على التطبيقات 84

المصطلحات المستخدمة

Requirement	المتطلبات
Design	التصميم
Development	التطوير
Testing	الاختبار
Maintenance	الصيانة
Testing Process	عملية الاختبار
Test Case	حالة الاختبار
Test Coverage	تغطية الاختبار
Test Suite	مجموعة الاختبار
Mutation Testing	اختبار الطفرة
Testing Levels	مستويات الاختبار
Unit Testing	اختبار الوحدة
Integration Testing	اختبار التكامل
System Testing	اختبار النظام
Acceptance Testing	اختبار القبول
Testing Methods	طرائق الاختبار
Black Box Testing	اختبار الصندوق الأسود
White Box Testing	اختبار الصندوق الأبيض
Gray Box Testing	اختبار الصندوق الرمادي
Testing Types	أنواع الاختبار
Manual Testing	الاختبار اليدوي

Automated Testing	الاختبار الآلي
Random-Based Methods	توليد حالات الاختبار بشكل عشوائي
Model-Based Methods	توليد حالات الاختبار اعتماداً على النموذج
Search-Based Methods	توليد حالات الاختبار اعتماداً على البحث
Optimization	الأمثلية
Robustness	المتانة
Efficiency	الكفاءة
Accuracy	الدقة
Evolutionary Algorithms	الخوارزميات التطورية
Fitness Function	تابع الملاءمة
Genetic Algorithm	الخوارزمية الجينية
Selection	الاختيار
Crossover	العبور
Mutation	الطفرة
Swarm Intelligence	ذكاء السرب
Artificial Bee Colony	خوارزمية النحل الصنعية
Worker Bees	النحل العامل
Scout Bees	النحل الكشاف
Activity	النشاط
Services	الخدمات
Broadcast Receiver	مستقبل البث
Content Provider	مزوّد المحتوى
Test Objectives	الغاية من الاختبار
Test Targets	أهداف الاختبار

Test Methodologies	منهجيات الاختبار
Independent Path	المسار المستقل
Cyclomatic Complexity	التعقيد الدائري
Basis Paths Testing	اختبار المسارات الأساسية
Graphical User Interface	واجهة المستخدم الرسومية
Events Flow Graph	مخطط تدفق الأحداث

الفصل الأول: المقدمة

1-1 مقدمة

دورة حياة البرمجية هي مجموعة من الخطوات والأساليب والاستراتيجيات المتبعة لبناء برامج ذات كفاءة عالية وفق الوقت المحدد والجودة المطلوبة والميزانية المتفق عليها، وتوجد عدة أنواع شائعة الاستخدام تمثل دورة حياة أي مشروع برمجي منها نموذج الشلال أو النموذج الحلزوني وغيرها من النماذج، باستخدام أي نموذج فإنّ دورة حياة تطوير البرمجيات تتكون من خمس مراحل أساسية موضحة فيما يلي[1]:

1. جمع وتحديد المتطلبات (Requirements): المرحلة الأولى من مراحل دورة حياة تطوير البرمجية، تُعرِّف المتطلبات الخدمات المتوقع من النظام تقديمها والوظائف المطلوبة منه، حيث يتم استخلاص هذه المتطلبات من خلال التواصل مع المستخدمين الفعليين للنظام ومالكي النظام، خرج هذه المرحلة بناء وثيقة المتطلبات.
2. التصميم (Design): يحدد التصميم هيكلية النظام وبنيته كما يحدد الواجهات ونوافذ المستخدم User Interfaces، والمكونات Components، والوحدات Modules والبيانات الخاصة بالنظام كي يحقق متطلبات الزبون، تتم مرحلة التصميم اعتماداً على المتطلبات التي تم تحديدها في وثيقة المتطلبات من المرحلة الأولى.
3. التطوير (Development): تُحوَّل الخوارزميات والمخططات Diagrams التي تم إنتاجها في مرحلة التصميم إلى برنامج قابل للاستخدام من قبل الزبون، بحيث يلبي احتياجاته الموضحة في وثيقة المتطلبات، تتم خلال هذه المرحلة بعض الاختبارات على بعض أجزاء النظام للتأكد من عمله بطريقة صحيحة.
4. الاختبار (Testing): هدف هذه المرحلة التأكد من موافقة النظام للشروط والمتطلبات وتنفيذها بشكل صحيح ويلبي رغبة المستخدمين، تتم عملية الاختبار للبرامج على أكثر من مرحلة وعلى مستويات مختلفة (سنقوم بتفصيل مرحلة الاختبار في الفصل الثاني).
5. الصيانة (Maintenance): تتم خلال هذه المرحلة عملية تصحيح الأخطاء، وتشمل تحسينات أو تعديلات أو إضافات محتملة من المستخدم النهائي.

سنتناول في دراستنا مرحلة الاختبار لأهمية هذه المرحلة حيث تساهم عملية اختبار البرمجيات بشكل كبير في اكتشاف الأخطاء ومعالجتها قبل نشرها للمستخدمين، كما يجب اختبار البرمجية ضمن بيئة مشابهة للبيئة

التي سيتم نشرها ضمنها للتأكد من ملاءمتها لاحتياجات المستخدمين ومن تكامل تنفيذ وظائف النظام مع بعضها بشكل صحيح، تتطلب مرحلة الاختبار وقتاً وكلفةً تقدر بحوالي 50% من إجمالي كلفة تطوير البرمجية، لذلك من المهم تقليل هذه الكلفة وتحسين كفاءة عملية الاختبار عن طريق أتمتة هذه المرحلة [2].

يعد توليد حالات الاختبار المرحلة الأهم من بين أنشطة الاختبار المتعددة لتأثيرها على كفاءة وفعالية عملية الاختبار بشكل عام، إنّ مجموعة الاختبار الجيدة هي القادرة على اكتشاف الأخطاء بالتالي الاختبار الناجح هو الذي اكتشف هذه الأخطاء في التطبيق قبل نشره للمستخدمين، لاكتشاف كل ما هو ممكن من أخطاء في التطبيق يجب إجراء اختبار شامل للتحقق من جميع المدخلات والمسارات الممكنة التنفيذ في التطبيق، لكن هذا الأمر مكلف ويزداد صعوبة كلما ازداد تعقيد التطبيق، لذلك من الضروري في مرحلة الاختبار اكتشاف الأخطاء والحالات غير المرغوب بها في التطبيق باستخدام عدد محدود من حالات الاختبار مما يوفر الوقت والجهد اللازمين لهذه المرحلة.

تعتمد مرحلة توليد حالات الاختبار وهيكلية الحالات المولّدة بشكل أساسي على نوع التطبيقات المراد اختبارها، مثلاً لاختبار تطبيقات تعالج قيماً رقمية سيتم توليد قيماً لبيانات الدخل اعتماداً على استراتيجيات التجزئة والتقسيم للقيم الرقمية واختيار بيانات حدية، ولاختبار تطبيقات تعالج صوراً يجب توليد حالات اختبار تمثل صوراً لتجريب أنواع مختلفة من الصور حسب طريقة معالجتها، ولتوليد حالات اختبار للتطبيقات المقادة بالأحداث مثل تطبيقات واجهات المستخدم الرسومية GUI أو تطبيقات Android فإنّ هذه الحالات يجب أن تعكس تسلسل أحداث تفاعل المستخدم مع الواجهة والتطبيق، إذاً توليد حالات الاختبار يعتمد على فهم التطبيق وبنيته وكيفية تنفيذ الوظائف.

انتشرت الهواتف النقالة بشكل واسع وتنوعت أنظمة التشغيل التي تدعم هذه الأجهزة، أكثرها استخداماً نظام Android، انتشرت تطبيقات Android وتنوعت الوظائف التي تقدمها وتلبي حاجات المستخدمين، تنوع التطبيقات وانتشارها ساعد المستخدمين وبسّط لهم العديد من المهام الحياتية واليومية والوظيفية، لكن بالمقابل لا تخلو هذه التطبيقات من الأخطاء، وبسبب العرض الكبير لهذه التطبيقات أصبحت جودة التطبيق وصحة تنفيذه وخلوه من الأخطاء أمراً هاماً لضمان وجود التطبيق ومنافسته في السوق، لذا من الضروري إعطاء أهمية كبيرة لتطبيقات Android خلال مرحلة عملية التطوير واستخدام تقنيات هندسة البرمجيات بما فيها عملية اختبار التطبيقات التي لا غنى عنها لضمان نجاح وصحة التطبيق.

1-2 مشكلة البحث

الانتشار الواسع لتطبيقات Android فتح العديد من التحديات أمام الباحثين في مجال هندسة البرمجيات لإنشاء برمجيات متينة، من التحديات الأساسية إنشاء تطبيقات ذات جودة وخالية من الأخطاء وتوافق متطلبات المستخدمين حسب الدراسة [3] تم نشر عدة تطبيقات على متجر Google Play فيها أخطاء وفشل أثناء الاستخدام أو سلوك غير صحيح من حيث تنفيذ الوظائف ففي عام 2014 حوالي 61% من التطبيقات تم استخدامها أقل من عشر مرات قبل حذفها، عدد التطبيقات الهائل جعل الخيارات أكبر للمستخدمين للتحول وحذف التطبيق الذي يفشل.

حسب الدراسة [4] واستناداً إلى مشاريع Android المستضافة على Github تبيّن أنّ أتمتة مرحلة الاختبار نادراً ما تستخدم، ووجدوا أن أربعة تطبيقات فقط من أصل 627 تطبيق لديها حالات اختبار قادرة على تغطية 40% فقط من كود التطبيق، ويعود ذلك إلى أن أتمتة الاختبار لا تزال عملية صعبة بالنسبة لتطبيقات Android لذلك يتم الاختبار يدوياً غالباً بين مطوري هذه التطبيقات.

لا يمكن تطبيق الأدوات والدراسات المتوفرة لاختبار التطبيقات التقليدية واستخدامها في مجال تطبيقات Android بشكل مباشر بسبب طبيعة وبنية تطبيقات Android والتي تعتمد وترتبط بمنصة نظام التشغيل من خلال الاستدعاءات الموجهة من النظام للتطبيق بالإضافة لأحداث المستخدمين من خلال تفاعلهم مع التطبيق حيث تعتبر تطبيقات Android من التطبيقات المقادة بالأحداث، لذلك لاختبار هذه التطبيقات يجب أن تكون حالات الاختبار شاملة للاستدعاءات التي يتلقاها التطبيق من نظام تشغيل Android وهي خاصة بطرائق دورة حياة كل مكون من مكونات التطبيقات بالإضافة للاستدعاءات لطرائق معالجة الأحداث التي يقوم بها المستخدم من خلال التفاعل مع واجهات التطبيق.

1-3 الهدف من البحث

نهدف من خلال هذا الدراسة إلى تحسين عملية اختبار تطبيقات Android من خلال التركيز على مرحلة توليد حالات الاختبار، حيث نهدف إلى تطوير أداة لتوليد واشتقاق حالات اختبار لتطبيقات Android اعتماداً على بنية التطبيق من خلال استخدام تقنيات الذكاء الصنعي وبشكل أخص خوارزمية النحل الصنعية لتحسين عملية توليد حالات الاختبار، سنقوم خلال دراستنا باستعراض آليات واستراتيجيات توليد حالات الاختبار للتطبيقات المقادة بالأحداث ، وجدت عدة معايير لتقييم جودة حالات الاختبار التي تم تنفيذها مثل مقدار التغطية للشيفرة البرمجية وعدد الحالات، لذلك من خلال هذه الدراسة نهدف للوصول للإجابة على الأسئلة الآتية والتي تعتبر جوهرية في أي عملية اختبار:

- كم حالة اختبار سيتم تنفيذها
- كم ستغطي من التطبيق الذي سيتم اختباره
- مدى فعالية هذه الحالات وقدرتها على عكس حالة التطبيق واكتشاف الأخطاء

سنقوم خلال دراستنا بتطوير خوارزمية لاشتقاق حالات اختبار لتطبيقات واجهات المستخدم الرسومية اعتماداً على خوارزمية النحل الصنعية والبيان الذي يمثل التطبيق، ثم تعميم هذه الدراسة واستخدامها لتوليد حالات اختبار لتطبيقات Android اعتماداً على الأداة التي تم تطويرها EGator المسؤولة عن تحليل التطبيق والوصول لبيان يمثل مكوناته المختلفة ثم تطبيق خوارزمية النحل لاشتقاق حالات اختبار من البيان المولد ومقارنة دراستنا مع دراسات سابقة في نفس المجال لتوضيح الميزات التي تمت إضافتها من خلال دراستنا.

سنقدم في نهاية الدراسة الأداة BeeDroid بهدف تسهيل عمل مختبري تطبيقات Android حيث الدخل سيكون ملف apk يمثل التطبيق المراد اختباره والخرج هو وثيقة تمثل مجموعة الاختبار التي سيتم تنفيذها دون حاجة المختبر إلى معرفة أية تفاصيل عن كيفية تطوير التطبيق أو معرفة بنيته ومكوناته.

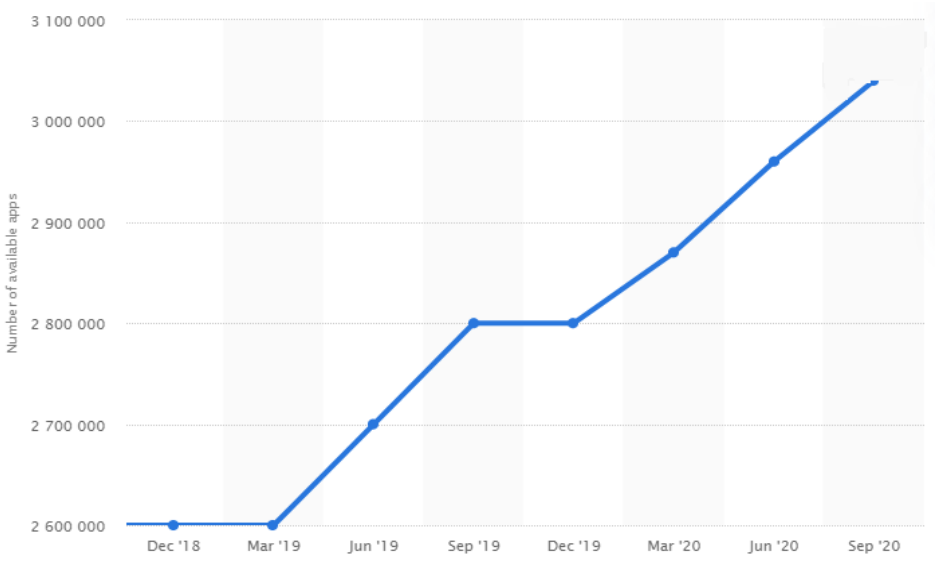
1-4 مبررات البحث

يمكن تلخيص مبررات العمل ضمن هذا البحث للأسباب والتحديات الآتية التي تواجه مختبري تطبيقات Android:

1- انتشار تطبيقات Android بشكل واسع وازدياد عدد هذه التطبيقات بشكل كبير، يوضح الشكل (1-1) عدد هذه التطبيقات بين عامي 2019 و2020 مما جعل الحاجة ملحّة لتطوير هذه التطبيقات بشكل منهجي لضمان صحة التطبيق وموافقته لرغبات المستخدمين ومعايير الجودة، بالتالي لا بد من اختبار هذه التطبيقات قبل نشرها للمستخدمين.

2- بالرغم من أنّ معظم تطبيقات Android مطورة باستخدام لغة Java إلا أنه لا يمكن استخدام أدوات اختبار تطبيقات Java مباشرة مع تطبيقات Android، بسبب طبيعة تطبيقات Android وبنيتها المختلفة والأحداث الناتجة عن تفاعل المكونات مع بعضها.

3- يعتمد معظم مطوري هذه التطبيقات على توليد حالات اختبار بشكل يدوي اعتماداً على فهم التطبيق لتجريب كل الحالات الممكنة وهذا يتطلب تنسيقاً كبيراً بين فريقي التطوير والاختبار خلال عملية الاختبار.



الشكل (1-1) عدد تطبيقات Android خلال عامي 2019 و2020

4- يجب إنشاء حالات اختبار تؤمن تغطية أعظمية لشيفرة التطبيق من أجل إيجاد العيوب في التطبيق، تطبيقات Android مقادة بالأحداث بالإضافة للاستدعاءات من نظام التشغيل للتطبيق وهذا يشكّل المزيد من العقبات لتوليد حالات اختبار أمثليه.

5- عدم توفر الكثير من الأدوات لتوليد حالات الاختبار بشكل آلي، يوجد دراسات في هذا المجال لكل منها ميزات وسلبيات حسب الآلية المتبعة لتوليد حالات الاختبار.

1-5 خطة البحث

1. جمع المعطيات المتعلقة بالبحث من مراجع والاطلاع على الدراسات السابقة.
2. دراسة مرحلة اختبار البرمجيات.
3. دراسة الآليات والاستراتيجيات المتبعة لتوليد حالات الاختبار.
4. تحديد نوع التطبيقات المستخدمة لتوليد حالات اختبار لها، وهنا اعتمدنا تطبيقات Android.
5. دراسة الأدوات والآليات المتوفرة لاختبار تطبيقات Android.
6. تطوير الأداة BeeDroid والتي تساعد على توليد حالات اختبار لتطبيقات Android بشكل آلي مما يسهّل ويقلل من الكلفة والزمن اللازمين لمرحلة الاختبار.
7. مقارنة الأداة المقترحة مع دراسات سابقة ضمن نفس المجال وتقييم النتائج.

1-6 مخطط الأطروحة

يتضمن البحث سبعة فصول كالتالي:

الفصل الأول: مقدمة، وهدف البحث، ثم توضيح التحديات والمشاكل التي تواجه اختبار تطبيقات Android التي كانت المبرر لبحثنا هذا.

الفصل الثاني: يقدم هذا الفصل دراسة لمرحلة اختبار البرمجيات، وكل ما يتعلق بهذه المرحلة من مفاهيم، تم التركيز على مرحلة توليد حالات الاختبار والآليات المستخدمة للتوليد.

الفصل الثالث: يقدم هذا الفصل دراسة عن الخوارزميات التطورية، مكوناتها وآلية عملها بشكل عام، ثم قدمنا في نهاية هذا الفصل دراسة مفصلة للخوارزمية الجينية وخوارزمية النحل الصنعية.

الفصل الرابع: يقدم هذا الفصل دراسة لتطبيقات واجهات المستخدم الرسومية وكيفية اختبارها، ثم نقدم الخوارزمية المقترحة لاشتقاق حالات الاختبار اعتماداً على نموذج يمثل مخطط تدفق الأحداث وباستخدام خوارزمية النحل الصنعية.

الفصل الخامس: يقدم هذا الفصل دراسة مفصلة لتطبيقات Android وخصائصها ومكونات هذه التطبيقات، ثم دراسة الأدوات والآليات المتبعة لاختبار هذه التطبيقات والتحديات الموجودة، بعد ذلك قمنا ببناء الأداة BeeDroid التي تبدأ بتحليل هذه التطبيقات للوصول لنموذج بيان يوصف التطبيق ومكوناته والانتقال بينها، ليكون هذا البيان دخل لخوارزمية النحل لاشتقاق حالات اختبار أمثلية.

الفصل السادس: قمنا في هذا الفصل بمقارنة الأداة BeeDroid وتقييم النتائج التي توصلنا لها من خلال معيارين للتقييم هما تغطية الشيفرة البرمجية واختبار الطفرات، وبيّنا أن الأداة BeeDroid قدمت نتائج أفضل من الدراسات السابقة بالإضافة لفاعلية هذه الأداة وقدرتها على اكتشاف الطفرات.

الفصل السابع: يقدم هذا الفصل النتائج التي تم التوصل إليها بعد الانتهاء من البحث، بالإضافة لمجموعة من التوصيات، والأعمال المستقبلية المقترحة للمتابعة فيها.

الفصل الثاني: اختبار البرمجيات

2-1 مقدمة

تهدف عملية اختبار البرمجيات إلى كشف الأخطاء والتحقق من أنّ البرمجية تلائم متطلبات المستخدمين، تحتاج عملية الاختبار إلى فريق مختص في مجال الاختبار وفي مجال الأخطاء التي قد يقع بها النظام، على المختبر أن يفكر كيف سيستخدم الزبون كل خاصية وكل وظيفة في التطبيق، هذا يعني تجريب كل الحالات المحتملة للتنفيذ وبالتالي محاولة اكتشاف الأخطاء وتصحيحها قبل نشر التطبيق بشكل نهائي للمستخدمين.

تتم عملية الاختبار من خلال تنفيذ التطبيق عبر اختيار مجموعة من الحالات تعتبر كدخل ثم تنفيذ التطبيق ومراقبة النتائج هل توافق الخرج المتوقع أم لا، يجب اختيار حالات من الدخل للبيانات بحيث تعكس كل حالات البرنامج وتظهر الخطأ والمشاكل في البرمجيات لتصحيحها قبل طرحها بشكل نهائي للمستخدمين.
فيما يلي سنوضح بعض المفاهيم والمصطلحات المتعلقة بمرحلة اختبار البرمجيات:

2-2 مراحل عملية اختبار البرمجيات Testing Process

تمر عملية اختبار البرمجية بمراحل أساسية موضحة كما يلي [1]:

- توليد حالات الاختبار
- بناء مجموعة الاختبار
- تنفيذ الاختبار
- التأكد من صحة تنفيذ التطبيق لحالة الاختبار وهل حصل فشل في التطبيق أم لا
- إعطاء تقرير بعملية الاختبار

2-3 حالة الاختبار Test Case

هي وثيقة يتم من خلالها تحديد بيانات الدخل للاختبار، الخرج المتوقع من البرنامج أو هل حصل فشل أم لا خلال التنفيذ، أي الشروط المسبقة للبرنامج قبل تنفيذ الاختبار والشروط اللاحقة بعد التنفيذ، فهي تعتبر نقطة البداية لأي عملية اختبار، كل حالة اختبار تختبر النظام هل يوافق متطلب معيّن أم لا [5]، وفي حالة تطبيقات GUI والتطبيقات المقادة بالأحداث فإنّ حالة الاختبار تمثل تسلسلاً من الأحداث يتم إنشاؤها

وتنفيذها على واجهة التطبيق [6]، يمكن تمثيل تدفق الأحداث في التطبيقات المقادة بالأحداث من خلال رسم بياني موجه، تم استخدام هذا البيان في عملية اختبار التطبيق، الخطوة الأهم في عملية الاختبار هي توليد حالات أمثلية تعكس كل احتمالات استخدام وتفاعل المستخدم مع التطبيق.

2-4 تغطية الاختبار Test Coverage:

مقياس يُستخدم لوصف وتحديد مقدار الشيفرة البرمجية التي تم تنفيذها عند تشغيل مجموعة الاختبار (يتم قياسه كنسبة مئوية)، البرنامج الذي تغطية الاختبار له أعلى يشير إلى فرصة أقل لوجود أخطاء برمجية لم يتم اكتشافها خلال الاختبار، تساعد المختبر على إعطاء قياس لاكتمال الاختبار، يمكن استخدام العديد من المقاييس المختلفة لحساب تغطية الاختبار مثل النسبة المئوية للتعليمات أو المسارات التي تمت تغطيتها وتنفيذها[7].

التغطية الكاملة لكامل مسارات البرنامج مسألة معقدة وأحياناً مستحيلة عندما يكون حلقات في شيفرة التطبيق، لذلك تتجه الأبحاث دائماً لإيجاد طريقة لاختيار المسارات الأمثل التي سيتم تنفيذها من خلال حالات الاختبار والتي تغطيتها للكود أعظمية[8].

2-5 معايير الاختبار:

تستخدم لتحديد مدى جودة حالات الاختبار ولمقارنة أية حالات تم توليدها هي الأفضل، هناك معايير اختبار عامة، نذكر منها[5] :

- عدد الأخطاء التي تم اكتشافها
- عدد حالات الاختبار التي تم تنفيذها
- مقدار تغطية حالات الاختبار لشيفرة التطبيق الذي يتم اختباره
- مدى قدرة حالات الاختبار على اكتشاف الأخطاء في التطبيق إنْ وجدت

2-6 Oracle Verifier:

التنبؤ بصحة تنفيذ حالة الاختبار أم لا يدعى Oracle، حيث يوفر آلية لتحديد ما إذا كان الاختبار الذي تم تنفيذه قد تحقق أم لا (هل التطبيق نفذ كما هو متوقع منه)، وفي حالة التطبيقات المقادة بالأحداث تعني التنبؤ هل تم تنفيذ التطبيق بشكل صحيح أو حصل فشل في التطبيق أو استثناءات، بالإضافة إلى تسلسل الأحداث التي سيتم تنفيذها يجب على مصمم حالة الاختبار أن يحدد ما الذي سيتم التحقق منه في نهاية

تنفيذ حالة الاختبار، عند اختبار تطبيقات GUI يوجد آليتين للتحقق من صحة الاختبار، إما State Verifier وفيها نتأكد من حالة الواجهة بعد التنفيذ وحالة العناصر وقيم خواصها أو Crash Verifier يتم من خلالها التأكد هل فشل التطبيق وإعطاء تقارير بالأخطاء التي تسبب فشل النظام وتوقفه عن العمل أو أية استثناءات يمكن أن تحصل خلال التنفيذ ولم يتم معالجتها[9] .

2-7 مجموعة الاختبار Test Suite:

يتضمن إنشاء مجموعة الاختبار اختيار مجموعة من حالات الاختبار المحتملة للتنفيذ، بناء مجموعة الاختبار يعتمد على مقدار الاختبار الكافي، عادةً ما يقوم المختبرون ببناء المجموعة على أساس مفهوم التغطية المطلوبة لشيفرة التطبيق كي تغطي حالات الاختبار أعلى نسبة من الشيفرة لضمان تنفيذ كل التعليمات، بالنسبة لاختبار تطبيقات GUI يتم اعتماد فكرة تغطية الأحداث أو تركيب الأحداث في سياق معين، عند بناء مجموعة الاختبار لـ GUI غالباً ما يتم اعتماد مقاييس الاختبار الممثلة التغطية وسرعة الأداء والفعالية لحالات الاختبار وقدرتها على كشف الفشل في التطبيق[9] .

2-8 اختبار الطفرة Mutation Testing

هو نوع من الاختبارات يتم خلاله تغيير جزء من شيفرة التطبيق للتحقق من قدرة حالات الاختبار على العثور على الأخطاء في التعليمات البرمجية، الهدف من اختبار الطفرات هو ضمان جودة حالات الاختبار، التغييرات في الشيفرة يجب ألّا تؤثر على الهدف العام من التطبيق، وهو نوع من اختبار الصندوق الأبيض، فيما يلي سنعرض خطوات اختبار الطفرة [10]:

1. يتم إدخال أخطاء في شيفرة المصدر عن طريق إنشاء إصدارات تسمى mutants، كل mutant يحتوي على خطأ تم حقنه في الشيفرة والهدف هو فشل نسخة التطبيق الـ mutant عند الاختبار مما يثبت فعالية حالات الاختبار.
2. يتم تطبيق حالات الاختبار على التطبيق الأصلي والتطبيق المعدل، يجب أن تكون حالة الاختبار مناسبة لاكتشاف الخطأ.
3. مقارنة نتائج الاختبار بين التطبيق الأصلي والمعدل.

4. إذا كانت النتائج من خرجي الاختبار غير متطابقة عندها فإنّ حالة الاختبار جيدة بما يكفي لاكتشاف التغيير بين التطبيق الأصل والمعدل.
5. إذا كانت النتائج من خرجي الاختبار متطابقة هذا يعني أن حالة الاختبار غير فعالة ويجب التعديل عليها لاكتشاف كل الـ mutants.

يتم إجراء التغيير في الشيفرة المصدرية حسب التطبيق، مثلاً يمكن التعديل في شروط أو عداد الحلقات.

يمكن أن يتم اختبار الطفرات بشكل يدوي لكن هذا سيسبب استهلاك المزيد من الوقت والجهد خلال مرحلة الاختبار، يوجد بعض الأدوات تساعد على أتمتة عملية اختبار الطفرات، منها الأداة MuDroid تستخدم لإجراء تغييرات (طفرات) في تطبيقات Android سنذكرها في الفصل السادس، تم استخدامها للتأكد من فعالية حالات الاختبار المولدة لتطبيقات Android.

2-9 مستويات الاختبار Testing Levels

تتم عملية الاختبار على عدة مستويات، حيث تختلف باختلاف الجزء المختار من التطبيق (النص البرمجي) لاختباره، سنوضح ذلك فيما يأتي[1] :

2-9-1 اختبار الوحدة Unit Testing

طريقة لاختبار كل جزء unit من أجزاء النص البرمجي للتطبيق المراد اختباره، unit المقصود بها أصغر جزء من النص البرمجي قابل للاختبار، يكون غالباً عبارة عن تابع أو إجرائية ضمن صف أو واجهة أو يمكن أن يكون صفاً أو عدة صفوف تقدم خدمة واحدة، يُكتب اختبار الوحدة من قبل المطور للتأكد من أنّ هذا الجزء ينفذ كما هو مطلوب ووفقاً للتصميم، يتم اختبار حالة تنفيذ كل وظيفة بشكل مستقل عن الباقي، مما يساعد على اكتشاف الأخطاء أو أي نقص في الوظائف بشكل مبكر من دورة حياة البرمجية.

2-9-2 اختبار التكامل Integration Testing

اختبار التكامل هو اختبار أجزاء مجتمعة من تطبيق ما لتحديد إذا كانت تعمل بشكل صحيح أم لا، اختبار التكامل يمكن ان يتم بطريقتين [2]:

- التكامل من أسفل إلى أعلى

يبدأ هذا الاختبار مع اختبار كل وحدة بشكل مستقل، تليها اختبارات مجموعات على مستوى أعلى مثل اختبار تكامل وظائف الصفوف مع بعضها.

- التكامل من أعلى إلى أسفل

في هذا الاختبار، يتم اختبار وحدات على مستوى أعلى أولاً وتدريجياً يتم اختبار وحدات على مستوى أقل بعد ذلك.

2-9-3 اختبار النظام System Testing

اختبار النظام أو End-To-End Testing هو اختبار للتحقق من سلوك التطبيق بشكل كامل ومن تكامل تنفيذ الوظائف مع بعضها، حيث يتم دمج جميع المكونات ثم اختبار التطبيق ككل ضمن البيئة التي سيعمل بها فيما بعد، أي يتحقق من حالة تنفيذ التطبيق كحزمة متكاملة.

يتم تنفيذ هذا النوع من الاختبارات من قبل فريق متخصص بالاختبار.

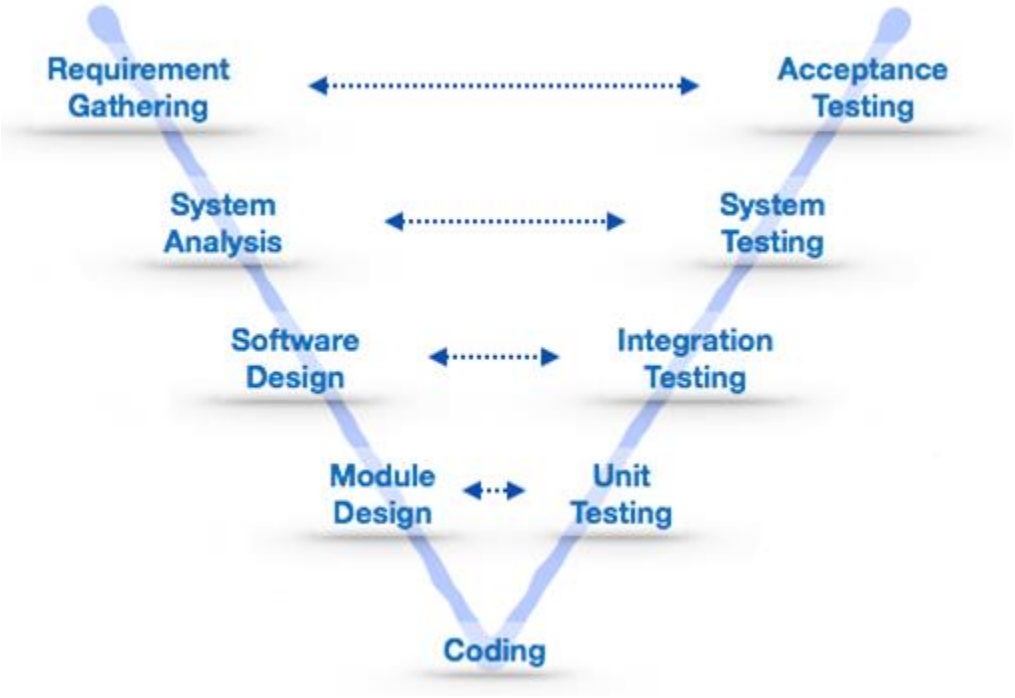
اختبار النظام مهم للأسباب الآتية:

- اختبار النظام هو الخطوة الأولى في تطوير البرمجية، حيث يتم اختبار التطبيق ككل ثم تعديله حسب نتائج الاختبار.
- يتم اختبار التطبيق في بيئة قريبة جداً أو مماثلة للبيئة التي سيتم نشر التطبيق ضمنها.
- اختبار النظام يمكننا من التحقق من صحة كل متطلبات العمل.

2-9-4 اختبار القبول Acceptance Testing

يعتبر الزبون في هذه المرحلة صاحب القرار في قبول التطبيق أو لا، حيث يختبر وظائف التطبيق للتأكد من أنها توافق المطلوب وتعمل بشكل جيد، ويقرر إذا كان التطبيق بحاجة لأيّة عملية تعديل أو تطوير ليوافق المطلوب.

الشكل (2-1) يبين مستويات الاختبار والمرحلة التي يتم التحقق من صحتها خلال مراحل تطوير البرمجية.



الشكل (2-1) مستويات الاختبار

2-10 طرائق الاختبار Testing Methods

2-10-1 اختبار الصندوق الأسود Black Box Testing

هو عبارة عن تقنية لاختبار وظائف التطبيق قيد الاختبار دون معرفة النص البرمجي الداخلي، أو بنية التطبيق أو أية تفاصيل لها علاقة بالتحقيق، ويعتمد في عملية الاختبار على المتطلبات والتوصيف المقدّم لهذا التطبيق، لا يملك اختبار الصندوق الأسود الوصول إلى النص البرمجي الذي تم استخدامه في بناء التطبيق، يهم المختبر في هذه الحالة فقط معرفة عمل التطبيق كما هو متوقع وليس من المهم معرفة كيفية تنفيذ التطبيق لهذه الوظائف، يتم اشتقاق حالات الاختبار في هذه الحالة من وثيقة المتطلبات أو من خلال تصميم البرمجية.

الشكل(2-2) اختبار الصندوق الأسود

يمكن تلخيص خطوات تنفيذ هذا الاختبار بالشكل الآتي[1] :

- يتم فحص المتطلبات وتوصيف التطبيق.

- يختار المختبر دخلاً لاختبار تنفيذ التطبيق للوظائف.
- بعد تحديد الدخل، يقوم المختبر بتحديد الخرج المتوقع لهذا الدخل الذي قام بإدخاله.
- يتم بناء حالات الاختبار، ومن ثم تنفيذها.
- بعد انتهاء مرحلة التنفيذ، يعود المختبر ليقارن النتائج التي حصل عليها مع الخرج المتوقع الذي قام بتحديده من قبل، ليتأكد من أن الخرج صحيح وبالتالي فإنّ الاختبار صحيح، وإلا يتم إصلاح الخطأ وإعادة الاختبار للتأكد من صحة النتائج.

2-10-2 اختبار الصندوق الأبيض White Box Testing

هو اختبار قائم على المعرفة بالنص البرمجي للتطبيق، مثل الاختبارات التي تستهدف بُنى معينة موجودة في النص البرمجي أو محاولة تحقيق مستوى معين من تغطية النص البرمجي، حالات الاختبار في هذه الحالة يتم اشتقاقها من النص البرمجي.

يستخدم اختبار الصندوق الأبيض في عدة مهام أهمها:

- اختبار المسارات ضمن العمليات.
- تتبع مجرى الدخل عبر الكود.
- التأكد من الخرج المتوقع.
- اختبار حلقات التفريع والشروط وصحتها.
- اختبار كل تعليمة، وكل غرض، وكل وظيفة على حِدا.

2-10-3 اختبار الصندوق الرمادي Gray Box Testing

هو عبارة عن دمج بين نوعي الاختبار السابقين (الأبيض والأسود)، ويعتبر اختباراً نصف شفاف.

أي أن هناك معرفة جزئية بالبنية الداخلية وخوارزميات التطبيق كي يتمكن المختبر من تصميم حالات الاختبار، بينما يكون تنفيذ هذه الاختبارات كحالة اختبار الصندوق الأسود.

في النهاية لا يمكن القول إن أحد الأنواع أفضل من الآخر، أو يمكن استخدام هذا النوع ولا يمكن استخدام النوع الآخر، لأن ذلك يتبع لطبيعة التطبيق المدروس، وما هو المطلوب اختباره أيضاً.

2-11 أنواع الاختبار Testing Types

توجد آليتان لتنفيذ عملية الاختبار:

2-11-1 الاختبار اليدوي Manual Testing

يتم في هذ النوع تشغيل الاختبار يدوياً من قبل المطورين/المختبرين، ويلعب المختبر دوراً مهماً كمستخدم نهائي ويتحقق من كافة خصائص التطبيق للتأكد من سلوكه من كافة النواحي، وهنا لا حاجة لامتلاك معرفة عن أيّة أداة، هنا توليد حالات الاختبار، تنفيذها ثم مراقبة النتائج والتأكد من صحة التطبيق كلها تتم بشكل يدوي، يتم اتباع هذا الأسلوب عندما لا تكون الأدوات للاختبار كافية أو استخدامها صعب.

2-11-2 الاختبار الآلي Automated Testing

يتم في هذا النوع استخدام أدوات مؤتمتة لتشغيل الاختبارات وإعادة/تكرار الأفعال المعرّفة مسبقاً، والتأكد من مطابقة النتائج لما يتوقعه المطور، تساعد هذه الأدوات فريق الاختبار للقيام باختبارات أكثر فعالية كون الجهد المفروض على المختبِر كعنصر بشري أقل بكثير من الحالة الأولى، كما يمكن أن تكون مرحلة توليد حالات الاختبار أيضاً مؤتمتة حسب الأداة المستخدمة.

يوجد العديد من الأدوات المتوفرة للاختبار حسب طبيعة التطبيق الذي سنختبره، مثلاً لتطبيقات الويب يوجد أدوات خاصة كذلك أدوات لاختبار التطبيقات المكتبية وأدوات لاختبار تطبيقات Android.

2-12 آليات توليد حالات الاختبار بشكل أوتوماتيكي

تتطلب عملية الاختبار كلفة ووقتاً لذا لتنفيذ اختبارات فعالة تعكس حالات النظام والأخطاء والمشاكل التي فيه يجب اختيار حالات اختبار فعالة بحيث تغطي كل حالات التطبيق، تعد مرحلة توليد حالات الاختبار مهمة شاقة وهامة في مرحلة الاختبار وتعتمد كفاءة وفعالية مرحلة الاختبار ككل على فعالية وكفاءة الحالات المولدة، لذا اتجهت عدة دراسات وأبحاث في مجال اختبار البرمجيات للتركيز على هذه المرحلة مما أدى إلى تطوير العديد من الأدوات والمنهجيات لتوليد حالات الاختبار تلقائياً، من أهم المنهجيات المتبعة لتوليد حالات الاختبار سنذكر ما يلي[11] :

2-12-1 توليد حالات الاختبار بشكل عشوائي Random-Based Methods

تعتمد هذه الطريقة على توليد حالات الاختبار بشكل عشوائي اعتماداً على طبيعة التطبيق، مثلاً في التطبيقات المقادة بالأحداث يتم استخدام منهجية الصندوق الأسود لتوليد تسلسل أحداث بشكل عشوائي مثلاً في حالة تطبيقات Android قدمت Google الأداة Monkey تعمل على توليد عشوائي انطلاقاً من أحداث النظام وأحداث المستخدم وإرسالها للتطبيق ليتم تنفيذها، في هذه الطريقة يتم توليد الأحداث بشكل مستقل عن آلية عمل ووظائف التطبيق، ما يميز هذا النوع أنه سهل التحقيق والنتائج جيدة من أجل التطبيقات البسيطة حيث يتم توليد عدد كبير من الحالات بشكل سهل بالنسبة للمختبر، من سلبياتها أن عملية توليد الحالات يتم

دون أي معرفة أو معلومات عن التطبيق ومن الممكن أن يؤدي إلى توليد حالات اختبار متكررة أو توليد سلسلة من الأحداث الغير قابلة للتنفيذ مما يسبب إضاعة الوقت والجهد دون الوصول لتغطية مناسبة لشيفرة التطبيق.

2-12-2 توليد حالات الاختبار اعتماداً على النموذج Model-Based Methods

الفكرة الأساسية من هذا النوع من الاختبار هو توليد مجموعة من المدخلات اعتماداً على نموذج يوصِّف التطبيق، يقلل التكرار ويحسن كفاءة حالات الاختبار المولدة مقارنة بالاختبار العشوائي، فعالية حالات الاختبار تعتمد على النموذج المصمم والذي يوصِّف سلوك التطبيق بدقة، حسب نوع التطبيقات يتم توليد نموذج يوصفها، عدة دراسات قُدِمت منها استخدم نماذج البيان والـ Finite State Machine أو نموذج ماركوف لتوصيف سلوك التطبيق سنذكر بعض هذه الدراسات في فصول لاحقة لأنّ بناء النموذج يعتمد على نوع وطبيعة التطبيق المراد اختباره، مهما تعددت الطرق لبناء النموذج تبقى المهمة الأساسية هي كيفية توليد حالات الاختبار اعتماداً على هذا النموذج، استخدمت طرقاً تقليدية للمرور على عقد البيان مثل البحث بالعمق أولاً أو بالعرض أولاً، أو توليد حالات اختبار عشوائية اعتماداً على النماذج المولدة.

3-12-2 توليد حالات الاختبار اعتماداً على البحث Search-Based Methods

في هذه الطريقة يتم النظر على مسألة البحث عن حالات الاختبار كمسألة تحسين لإيجاد حلول بأعلى جودة وفعالية، فضاء الدخل كبير وله بنى معقدة تمثل التطبيق وحالاته لذا يمكن استخدام تقنيات الأمثلية للوصول للحل الأمثل، بدأ استخدام هذا الاتجاه في مجال هندسة البرمجيات، يتم تطبيق تقنيات وخوارزميات البحث الأمثل (مثل الخوارزمية الجينية وخوارزميات السرب) لحل مشكلات اختبار التطبيقات حيث تتمثل الفكرة الأساسية بتقليل مساحة البحث من عدد لا حصر له من حالات الاختبار إلى عدد ممكن من الحالات، يستخدم هذا النوع في مرحلة الاختبار لتوليد حالات الاختبار، تحديد أولويات حالات الاختبار، تقليل مجموعات الاختبار، حيث يعتمد على استخدام خوارزميات البحث الأمثل لتوجيه البحث من خلال استكشاف مسارات التطبيق لتوليد حالات الاختبار، استخدمت الخوارزمية الجينية وخوارزميات السرب في هذا المجال [11].

2-13 خاتمة

قدمنا في هذا الفصل دراسة موجزة عن عملية اختبار البرمجيات والمفاهيم التي تتضمنها هذه المرحلة من مراحل عملية التطوير، قمنا بذكر مراحل عملية الاختبار والآليات المتبعة لاختبار التطبيقات والطرائق المتوفرة.

سنركز خلال دراستنا على مرحلة توليد حالات الاختبار لأهمية هذا المرحلة، الآلية التي سنتبعها لتوليد حالات الاختبار هي دمج منهجيتي الاختبار المعتمد على البحث والاختبار المعتمد على النموذج، سندرس تطبيقات Android بشكل مفصل ثم آلية توليد حالات الاختبار لهذه التطبيقات.

سنقدم في الفصل الثالث شرحاً عن خوارزميات البحث الأمثل وأنواعها والخوارزميات المستخدمة في مجال توليد حالات الاختبار.

الفصل الثالث: خوارزميات البحث الأمثل

3-1 مقدمة

أصبحت خوارزميات البحث الأمثل أكثر استخداماً في مجالات عدة ومنها مجال هندسة البرمجيات، استخدمت هذه الخوارزميات وخاصة في مرحلة الاختبار بهدف الوصول لحلول أمثلية لحالات الاختبار، خوارزميات عدة قدمت لكل منها إيجابيات وسلبيات حسب نوع التطبيق المراد اختباره وحسب الهدف من مرحلة الاختبار إذا كانت السرعة أو الفعالية العامل الأهم.

يشير مصطلح الأمثلية (optimization) إلى أفضل حل يمكن الحصول عليه عند التعامل مع مسألة ما، مثلاً يمكن أن نقلل مقدار الوقت اللازم لتنفيذ مهمة ما أو يمكن اختيار خوارزمية أبطأ لكن استهلاكها للذاكرة أقل، إذاً تعتمد الأمثلية على الحل الذي نسعى للوصول له، الأمثلية من الناحية الرياضية هي تقليل أو تعظيم لقيمة تابع ما خاضع لمجموعة قيود على متغيراته [12].

خوارزميات الأمثلة هي خوارزميات تكرارية تبدأ مع تقدير أولي للمتغير x (شعاع يمثل متغيرات الدخل أو فضاء البحث الأولي للخوارزمية) وتوليد سلسلة من التقديرات المحسنة حتى الانتهاء من العمل للوصول إلى حل أمثل، تعتبر الاستراتيجية المستخدمة للانتقال من تكرار لآخر من أهم الصفات التي تميز خوارزمية عن أخرى، بعض الخوارزميات تراكم المعلومات التي تم جمعها في التكرارات السابقة، بينما تستخدم خوارزميات أخرى المعلومات المحلية الوحيدة التي تم الحصول عليها في النقطة الحالية، يجب على خوارزميات الأمثلة أن تملك الخصائص الآتية [12] :

1- المتانة (Robustness): يجب على خوارزمية الأمثلة أن تكون قادرة على حل مجموعة واسعة من المسائل من نفس الفئة، ذلك من أجل جميع القيم المحددة من نقطة البداية.
2- الكفاءة (Efficiency): يجب ألا تتطلب الخوارزمية وقتاً طويلاً للتشغيل أو مساحة كبيرة للتخزين.
3- الدقة (Accuracy): يجب أن تكون الخوارزمية قادرة على إيجاد الحل بدقة.

في هذا الفصل سنقدم أهم الدراسات التي استخدمت خوارزميات البحث الأمثل في مجال الاختبار، ثم تعريف بهذه الخوارزميات وكيفية عملها.

2-3 الخوارزميات التطورية Evolutionary Algorithms

تصنف على أنها أحد مجالات الذكاء الصنعي، استُخدم الذكاء الصنعي بنجاح في مجموعة واسعة من المجالات من بينها النظم الخبيرة ومعالجة اللغات الطبيعية وتمييز الأصوات وتحليل الصور وكذلك التشخيص الطبي، وتداول الأسهم، والتحكم الآلي، والقانون، والاكتشافات العلمية، وألعاب الفيديو وألعاب الأطفال، ومحركات البحث على الإنترنت، وفي مجال هندسة البرمجيات بمراحلها ومنها مرحلة الاختبار بهدف تحسين هذه المرحلة.

خلال سنوات من البحوث، صمم الذكاء الصنعي عدداً كبيراً من الوسائل لحل أصعب المشاكل في علوم الكمبيوتر، منها مشاكل البحث والتحسين، تبدأ عملية البحث بشكل ما من التكهن والتخمين، ثم يعدل التخمين تدريجياً حتى الوصول إلى الدرجة الأمثل التي لا يمكن إجراء أي تحسينات بعدها.

تستخدم الخوارزميات التطورية آليات مستمدة من التطور البيولوجي مثل الطفرات والتزاوج والاختيار وإعادة التركيب، تلعب الحلول الأمثلية المرشحة للخوارزمية دور الأفراد في تجمع سكاني وتابع الملاءمة (Fitness Function) يحدد جودة الحلول، عملية تطور تعداد السكان تتم بعد تطبيق الآليات السابقة لعدة مرات متتالية وفق تسلسل معين[13] .

اكتسبت الخوارزميات التطورية أهمية متزايدة في السنوات الأخيرة، وتم اقتراح العديد من الخوارزميات منها ما يحاكي عملية التطور البيولوجية ومنها ما يحاكي ظاهرة ما في الطبيعة ومنها ما يحاكي سلوك كائن حي ما.

أثبتت هذه الخوارزميات قدرتها كآلية بحث عامة وقوية في كثير من المسائل المعقدة، أهم ما يميزها المرونة والقدرة على التكيف مع شتى المسائل، بالإضافة إلى الأداء القوي.

تصنف الخوارزميات التطورية من حيث تفاصيل تطبيقها وطبيعة المسألة إلى عدة أصناف، الأكثر استخداماً في مجال عملية اختبار البرمجيات هي الخوارزمية الجينية (GA)، وخوارزميات السرب.

تختلف الخوارزميات التطورية فيما بينها بشكل أساسي في العرض التطوري وفي تابع الملاءمة (fitness function) ، وفي المكونات التطورية: الطفرة (التغيير) وإعادة التركيب (Recombination) والاختيار (Selection) والتي تعتبر مكونات البحث في هذه الخوارزميات وعبرها يتم تحديد نطاق البحث، تطبيقها على عناصر الحل يمكن أن يضمن تحسيناً لنتيجة الخوارزمية.

3-3 مراحل الخوارزميات التطورية

يمكن تحديد الخطوات العامة بشكل عام لأغلب للخوارزميات التطورية كما هو مبين في الشكل (3-1) كما يلي[13]:

1- توليد جيل أولي من الأفراد وبشكل عشوائي ويعتبر فضاء البحث الابتدائي للخوارزمية.

2- تقييم تابع الملاءمة لكل فرد من الجيل الأول، يحدد تابع الملاءمة الأساس الذي تبنى عليه عملية الاختيار بين الأفراد، وبالتالي تسهيل عملية تحسين الحلول عبر تحديد التحسينات التي يجب القيام بها.

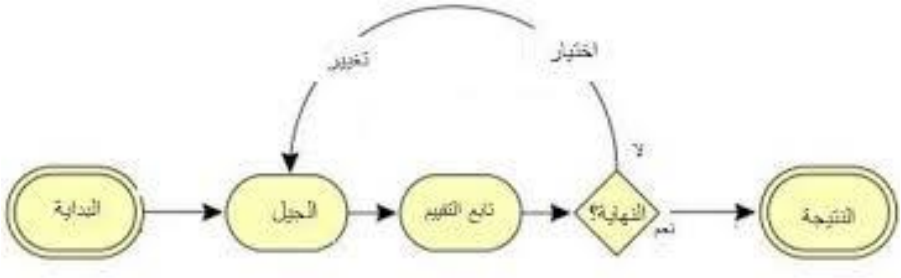
3- التكرار على هذا الجيل حتى الوصول لشرط التوقف (حد زمني، قيمة معينة للتابع الملاءمة، عدد من المرات للتكرارات ...)، وتتضمن هذه المرحلة الخطوات الآتية:

أ- اختبار أفضل الأفراد من الجيل السابق من أجل إنتاج جيل جديد.

ب-توليد أفراد جدد عن طريق عمليات الاختيار والتغيير (crossover) والطفرات (حسب الخوارزمية).

ت-تقييم جودة الأفراد في الجيل الجديد.

ث-استبدال الأفراد الأسوأ بالأفراد الجدد.

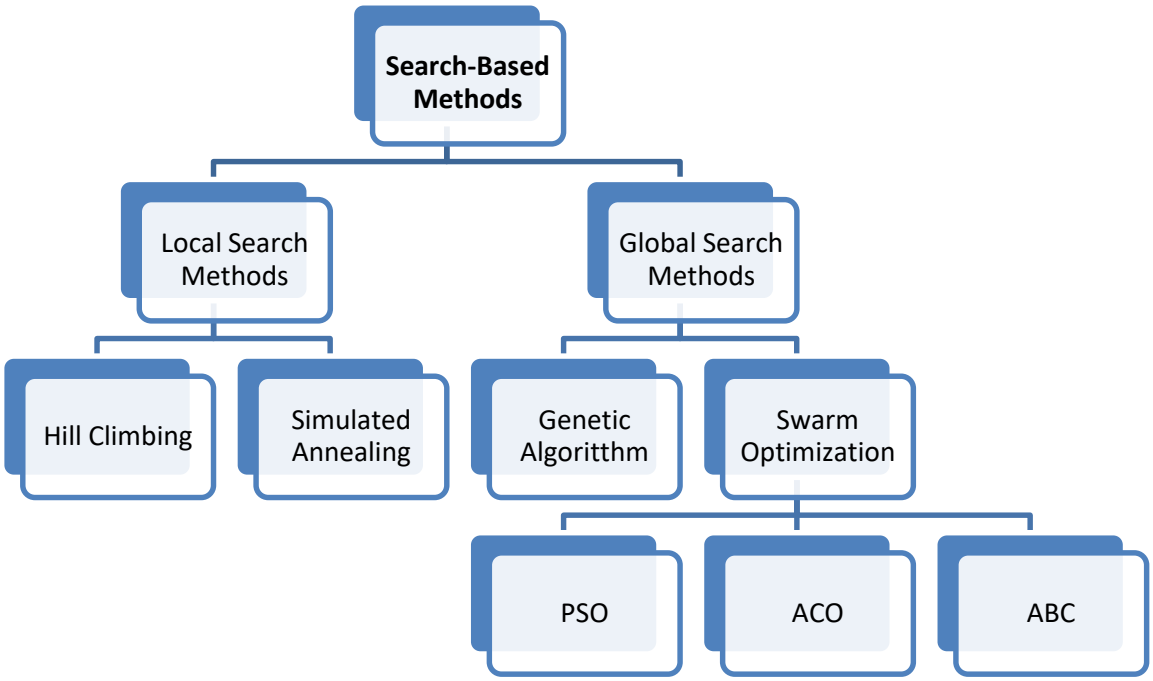


الشكل (3-1) المخطط العام لعمل الخوارزميات التطورية

4-3 استخدام الخوارزميات التطورية في مجال اختبار البرمجيات

سنقدم فيما يلي بعض الدراسات التي استخدمت خوارزميات البحث الأمثل في مجال توليد حالات الاختبار، مقاييس عدة طورت لقياس جودة حالة الاختبار ذكرناها في الفصل الثاني منها الزمن، تغطية شيفرة التطبيق، تعقيد الخوارزمية المستخدمة للتوليد، بعض الأبحاث ركزت على تحسين تغطية شيفرة التطبيق أو تقليل الزمن اللازم للتوليد حسب الأمثلية التي يراد الوصول لها، عدة خوارزميات استخدمت لتوليد حالات الاختبار بشكل أوتوماتيكي الهدف هو الوصول لحالات اختبار ذات تغطية أعظمية للشيفرة وعدد حالات أصغري خلال أقل

زمن ممكن، خوارزميات البحث التي استخدمت صنفت إلى خوارزميات بحث محلي أو بحث عام حسب فضاء البحث الذي يتم تغطيته، يبدأ البحث المحلي من عقدة وينتقل فقط لجيران هذه العقدة ميزة هذه الخوارزميات هي استهلاك قليل للذاكرة مثل خوارزمية Hill Climbing HC وخوارزمية Simulated Annealing SA، خوارزميات البحث العام تحل مشكلة البحث المحلي من خلال محاولة إيجاد حلول أمثلية عامة مما يعطي فعالية أكبر في البحث مثل الخوارزمية الجينية وخوارزميات السرب الشكل (3-2) يبين تصنيف هذه الخوارزميات[11] .



الشكل (3-2) تصنيف خوارزميات البحث

استخدمت الخوارزمية الجينية بشكل شائع لتوليد حالات الاختبار وأثبتت فعاليتها من حيث زمن التنفيذ ونسبة التغطية في عدة أنواع للتطبيقات إلا أنها من الممكن أن تنتقل لمشكلة حل أمثلي محلي بدلاً من حل عام من حيث التغطية مما يجعلها أبطأ بالإضافة لتابع الملاءمة الذي يعتبر عامل أساسي في كيفية أداء الخوارزميات [14] [15] ، بعض خوارزميات السرب قدمت وبينت أداء وفعالية أكبر وأبسط لكن يجب ضبط معاملاتها بشكل صحيح، استخدمت خوارزمية النمل الصنعية ACO وأثبتت قدرتها على تقليل حجم مجموعة الاختبار لكنها تعاني من مشكلة تكرار العقد ضمن نفس التسلسل بدون فائدة على معايير الاختبار [16]، كما استخدمت خوارزمية سرب الطيور PSO وأثبتت سرعتها لكن في [17] تمت مقارنتها مع خوارزمية النحل الصنعية وخوارزمية Harmony Search وأثبتت خوارزمية النحل الصنعية ABC أنها أفضل من

الطرق الأخرى في التحسين من حيث تقليل عدد الأجيال (التكرارات) والزمن اللازم للتوليد، حجم المستعمرة (عدد النحلات) له تأثير على الأداء يجب ضبطه بشكل صحيح.

في [18] تم تطبيق ستة من خوارزميات السرب (النحل، الخفاش، سرب الطيور، اليرقات، خوارزمية cuckoo، خوارزمية hill–climbing) في توليد حالات اختبار للتطبيقات التقليدية ومقارنة سرعتها وفعالية حالات الاختبار المولدة وبيّنت أن الأداء الأفضل لخوارزمية النحل حيث تمت المقارنة من ناحية أفضل زمن لازم وتغطية المسارات للحالات المولدة، تم مقارنة الخوارزميات بعد بناء مخطط تدفق التحكم للتطبيق بشكل يدوي، بينت الدراسة أفضل أداء لخوارزمية النحل الصنعية في مجال التطبيقات التقليدية، لكن لم يتم تقييمها من أجل أنواع أخرى من التطبيقات مثل التطبيقات المقادة بالأحداث.

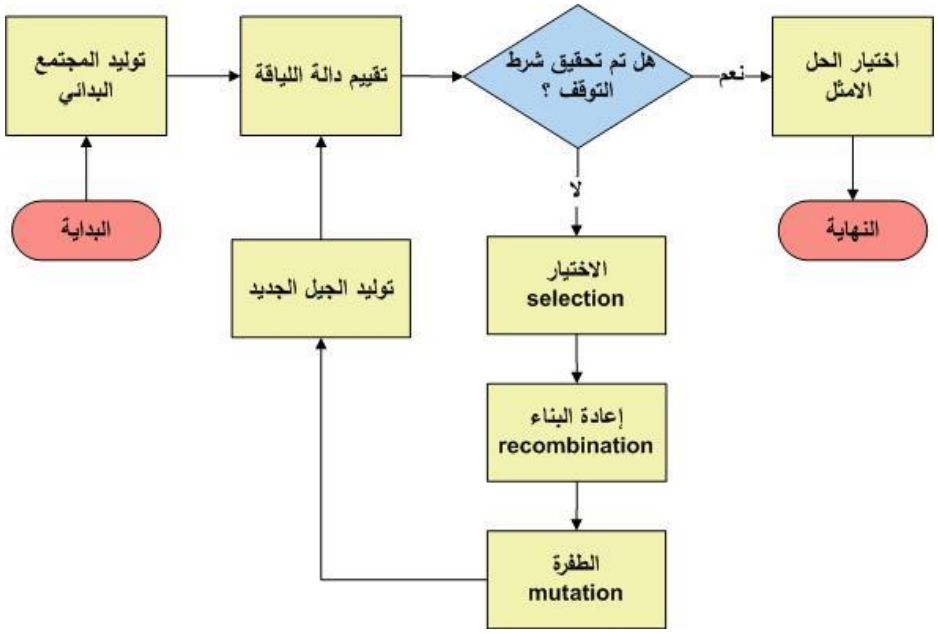
تبين من الدراسات أنّ أكثر الخوارزميات شيوعاً للاستخدام هي الخوارزمية الجينية بينما حسب الدراسة [18] تبين أداء أفضل لخوارزمية النحل من حيث الزمن والتغطية، لذلك لاعتماد إحدى هاتين الخوارزميتين لتحقيق أداة لتوليد حالات اختبار لتطبيقات Android قدمنا في الفصل الرابع حالة دراسية قارنا فيها بين هاتين الخوارزميتين لتوليد حالات اختبار لتطبيقات واجهات المستخدم الرسومية GUI كونها تعتبر من أنواع التطبيقات المقادة بالأحداث لاعتماد الأفضل من ناحية الزمن والتغطية لتحقيق الأداة BeeDroid لتوليد حالات اختبار لتطبيقات Android، فيما يلي سنقدم تعريفاً بالخوارزميتين وآلية عمل كل منهما.

3–5 الخوارزمية الجينيةGenetic Algorithm

ظهرت الخوارزمية الجينية عام 1975 من قبل John Holland، تحاول الخوارزمية الوصول إلى الحل الأنسب للمسألة المطروحة بالاعتماد على مبدأ العالم داروين في الاصطفاء الطبيعي القائم على الاحتفاظ بالميزات والصفات الجيدة الموجودة في جيل الآباء لنقلها لجيل الأبناء بهدف الحصول جيل يتمتع بأفضل الصفات المأخوذة من الجيل السلف، الحلول تمثل كصبغيات Chromosomes، تبدأ الخوارزمية من تعداد سكاني (صبغيات) عشوائي وهذا يحدث في كل الأجيال، في كل جيل يتم حساب تابع الملاءمة لكل الحلول ويتم تقييم أفضل الحلول اعتماداً على قيمة تابع الملاءمة ومن ثم عمل تهجين وأيضاً طفرة.

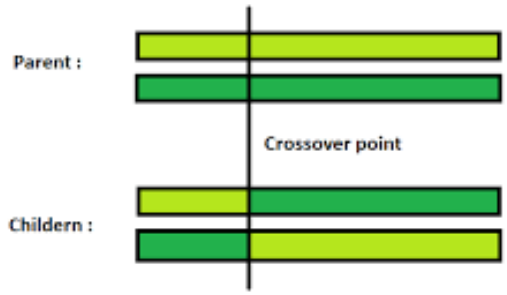
خطوات الخوارزمية [19]:

يمثل الشكل(3–3) مخطط التدفق لخطوات الخوارزمية الجينية:



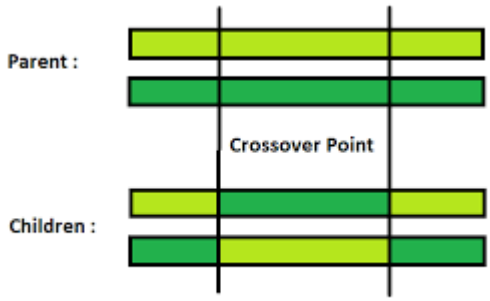
الشكل(3-3) يمثل مخطط تدفق الخوارزمية الجينية

- التهيئة (initialization): في البداية يتم توليد العديد من الحلول الفردية بشكل عشوائي تمثل الصبغيات البدائية، يعتمد حجم الصبغيات على طبيعة المشكلة، بشكل تقليدي يتم توليد الصبغيات بشكل عشوائي بحيث تغطي مجموعة كاملة من الحلول الممكنة فضاء البحث(search spaces) .
- الاختيار (Selection): خلال كل الأجيال المتعاقبة، هنالك نسبة من الصبغيات الحالية هي المختارة لإنتاج جيل جديد. ويتم اختيار هذه الصبغيات الاعتماد على تابع الملاءمة، حيث تكون نسبة الاختيار على أفضلية تابع الملاءمة.
- الاستنساخ: هي عملية لتوليد جيل ثان من الحلول التي تم انتقاؤها من خلال عملية الاختيار ومن ثم عمل عميلة العبور (crossover) والطفرة (mutation) لإنتاج الأبناء.
- عملية العبور: تعد عملية العبور من العمليات المهمة في الخوارزميات الجينية والتي تحاكي عملية التزاوج البيولوجي بين الأحياء، توجد العديد من التقنيات التي تَستعمل في عملية العبور:
  - عبور عبر نقطة واحدة: يتم تحديد نقطة واحدة على سلاسل الآباء، ويتم تبادل جميع البيانات خارج تلك النقطة والسلاسل الناتجة تمثل الأبناء، كما هو موضح في الشكل (3-4).



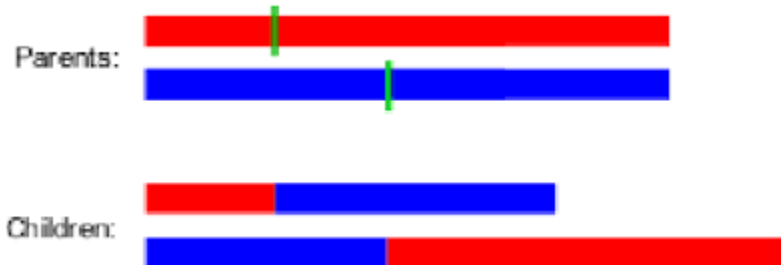
الشكل (3-4) عبور عند نقطة واحدة في الخوارزمية الجينية

- عبور عبر نقطتين: يتم تحديد نقطتين على سلاسل الآباء، ويتم تبادل جميع البيانات الموجودة بين تلك النقطتين والسلاسل الناتجة تمثل الأبناء، كما هو موضح في الشكل (3-5).



الشكل (3-5) عبور عند نقطتين في الخوارزمية الجينية

- القطع والوصل (القص واللصق): حيث تعمل هذه العملية على قطع البيانات من منطقة من سلسلة الأب الأول تختلف عن منطقة الأب الثاني مما يودي إلى اختلاف في طول السلاسل الأبناء كما يبين الشكل (3-6).



الشكل (3-6) عبور القطع والوصل في الخوارزمية الجينية

- الطفرة: هي عملية تغير مفاجئة في الأبناء الناتجة من عملية العبور بحيث يحدث تغير في شكل سلسلة الأبناء عن طريق تغير أحد مكونات الصبغي ليكون لها أثر إيجابي في الاقتراب من الحل الأمثل.

عملية الاستنساخ (العبور والطفرة) في النهاية تؤدي إلى إنتاج حلول (جيل) جديد لم يتم تطبيق عليها الدالة الأمثلية لتقييمك الأبناء الجدد.

- الإنهاء: عملية إيجاد جيل جديد تستمر حتى تتحقق إحدى شروط توقف الخوارزمية وهي:

أ- الوصول إلى الحل الأفضل.

ب-الوصول إلى العدد من الأجيال المطلوب.

ت-الوصول إلى قيمة معينة (budget) مثل حساب (الزمن).

ث-باستخدام مجموعة من الأسباب السابقة.

3-6 خوارزميات السرب أو ذكاء السرب Swarm Intelligence

تعتبر إحدى أنواع الخوارزميات التطورية، نموذج من الذكاء الجماعي يحاكي مجتمع الحشرات والأسماك وأسراب الطيور والثدييات من خلال تفاعلها واتصالها فيما بينها للوصول للهدف [20].

3-6-1 خوارزمية النحل الصنعية Artificial Bee Colony ABC

استخدمت خوارزمية النحل الصنعية في مجال اختبار البرمجيات وأثبتت تفوقها من ناحية تقليل الوقت اللازم وعدد التكرارات للوصول للحالات الأمثلية في مجال البرمجيات التقليدية.

اقترحت الخوارزمية من قبل Karaboga عام 2005، تعتمد خوارزمية النحل على ذكاء سلوك سرب النحل في البحث عن الطعام، فمن المعلوم أن النحل عندما يجد الطعام خلال رحلة البحث فإنه يعود للخلية بعيّنة منه ليخبر بقية النحل العاملة عن مكان الطعام واتجاهه من خلال قيام النحلة برقصة اهتزازية في اتجاه معين وبعدد معين من المرات للإشارة إلى مكان وجود الطعام، وتخبر عن موقع الغذاء واتجاهه وكميته وجودته عن طريق تلك الرقصات [21].

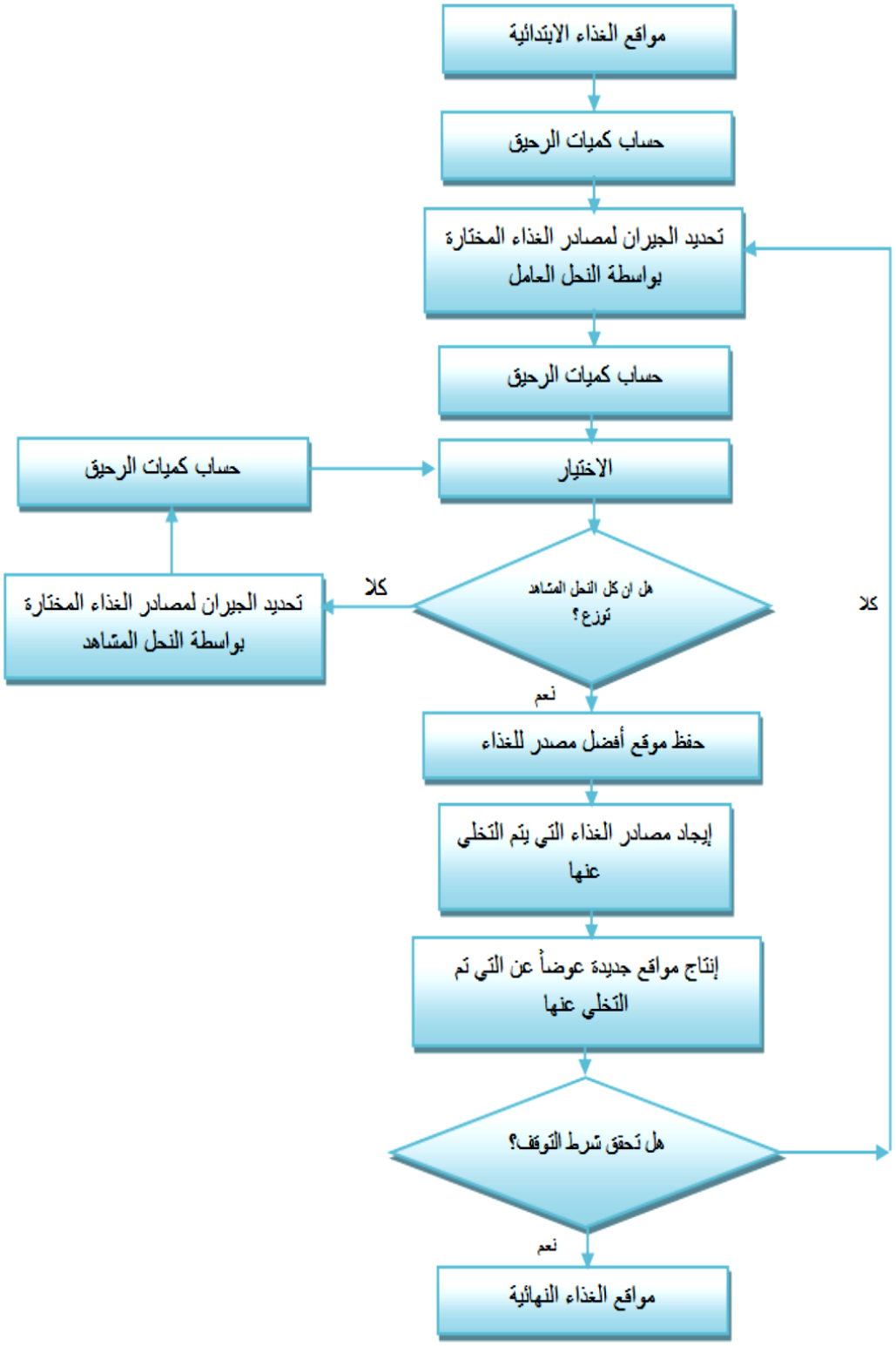
يتكون مجتمع النحل من ثلاث مجموعات: النحل العامل Worker bees ، النحل المشاهد والنحل الكشاف Scout bee ، النحل العامل هو المسؤول عن استغلال مصادر الرحيق وإعطاء معلومات للنحل المشاهد الذي ينتظر في الخلية عن جودة ومصدر الغذاء ومكانه عن طريق رقصات يقوم بها النحل العامل حيث كل حركة لها معنى يعبر عن المكان والمسافة للوصول للرحيق، وهنا يأتي دور المشاهدات، حيث تشاهد رقصات العاملات وتختار مصدر الطعام المناسب حسب الرقصات، أما العاملات اللاتي هُجرت (استُثنِيت) أماكن طعامهن فتصبح كشافة وتبدأ البحث عن مصادر جديدة للطعام.

اقترح Karaboga أن موقع مصدر الغذاء يمثل أحد الحلول الممكنة لمسألة الأفضلية، وكمية الرحيق من الغذاء يتوافق مع ربحية الحل المرتبطة بها (تابع الملاءمة أو جودة الحل)، عدد النحل العامل مساوٍ لعدد مصادر الغذاء حول الخلية [22].

خطوات الخوارزمية [21]:

الشكل(3-7) يمثل مخطط التدفق لخطوات خوارزمية النحل:

- إنتاج مصادر الغذاء الأولية لجميع النحلات العاملات وحساب كمية الرحيق لهذه المصادر.
- إرسال النحل العامل إلى مواقع مصدر الغذاء الموجود في ذاكرتها وتحدد مصدر مجاور له، وتقيّم مقدار الرحيق الموجود في المصدر المجاور.
- بعد أن يكمل النحل العامل عمليات البحث الخاصة بهم، يقوم بمشاركة المعلومات المتعلقة بكمية الرحيق ومواقع مصدر الغذاء مع النحل المشاهد من خلال الرقصات التي تؤديها، يقيّم النحل المشاهد معلومات الرحيق المأخوذة من كل النحل العامل.
- اختيار موقع مصدر الغذاء من قبل النحل المشاهد على أساس المعلومات المقدمة من النحل العاملات.
- يتم تحديد مصدر الغذاء المهجور واستبداله بمصدر جديد يكتشف من خلال الكشافة.
- يتم الاحتفاظ بأفضل مصدر غذاء تم العثور عليه حتى الآن في الذاكرة.



الشكل (3-7) خطوات خوارزمية النحل الصنعية

القاعدة العامة وخطوات عمل الخوارزمية:

تعتمد هذه الخوارزمية على حجم الخلية (عدد النحلات)، حيث يختلف حجم الخلية باختلاف المسألة المطروحة لتمثل فضاء البحث الأولي، حيث يقسم سرب النحل إلى 50% نحلات عاملات و50% نحلات مشاهدة، كما أن عدد النحلات العاملات يساوي عدد الحلول، النحل الكشاف هو عبارة عن عدد من النحلات العاملات التي تخلت عن المصدر الغذائي للبحث عن مصدر جديد، حيث:

$$Xi=\{x_{i,1}\ ,\ x_{i,2,}\ .....\ ,\ x_{i,n}\ \}$$

متجه يمثل الحل رقم i في الخلية، N يمثل مصادر الطعام (عدد النحلات العاملات).

كل نحلة عاملة Xi تولد حل محتمل Vi مجاور للحل الذي تملكه Xi، تعطى Vi بالمعادلة التالية:

$$vij = xij + \vartheta ij\ (xij - xkj) \qquad (1)$$

حيث X_k حل يتم اختياره عشوائياً بحيث K هو رقم عشوائي يتم اختياره ضمن المجال [1 .... N] ويختلف عن i، ϑij رقم عشوائي ضمن المجال [−1, 1].

بعد تحديد أماكن الغذاء Vi يتم مقارنتها حسب تابع الملاءمة، فإذا كان مصدر الغذاء يساوي أو أفضل من القديم يتم استبداله، وإلا يبقى المكان القديم للغذاء ويتم استبعاد المكان الجديد.

بعد أن يكمل كل النحل العامل عملية البحث يشاركون معلومات مصادر طعامهم مع النحل المشاهد من خلال رقصات الاهتزاز، يقوم النحل المشاهد بتقييم معلومات الرحيق المأخوذة من جميع النحل العامل ويختار مصدرًا للغذاء مع احتمال يتعلق بكمية الرحيق. هذا الاختيار الاحتمالي يوصف بالمعادلة أدناه:

$$pi = a * \frac{fitness(i)}{\max(fitnessi)} + b \qquad (2)$$

حيث a,b تقع قيمتما في المجال [1 − 0]،

الحل الأمثل وهو الحل الحاصل على أعلى احتمال Pi،

إن لم تتحسن القيمة الاحتمالية لمصدر غذاء X_k بعد عدد معين من الدورات، فسوف يتم استثناء هذا المصدر (الحل) واستبعاده، لتبدأ النحل الكشاف بالبحث عن مصدر جديد للغذاء يمثل حلاً جديداً وفق المعادلة (1) .

3-7 خاتمة

قدمنا في هذا الفصل دراسة موجزة عن الخوارزميات التطورية ومكوناتها وخطوات عملها، والدراسات التي استخدمت هذه الخوارزميات في مجال اختبار البرمجيات وخاصة مرحلة توليد حالات الاختبار، ثم قدمنا تفصيل لآلية عمل الخوارزميتين الجينية وخوارزمية النحل الصنعية.

سنقوم في الفصل الرابع بدراسة اختبار تطبيقات واجهات المستخدم الرسومية، ثم اقتراح خوارزمية لتوليد حالات اختبار اعتماداً على خوارزمية النحل الصنعية وباستخدام مخطط تدفق الأحداث، ثم مقارنة أداء خوارزمية النحل مع الخوارزمية الجينية لاعتماد الأفضل لاستخدامها في الأداة المطورة لتوليد حالات اختبار لتطبيقات Android.

الفصل الرابع: توليد حالات اختبار لتطبيقات واجهات المستخدم الرسومية GUI

4-1 مقدمة

قدمنا في الفصل السابق دراسة عن الخوارزميات التطورية وأنواعها واستخدامها في مجال اختبار البرمجيات، وبينا أن الخوارزمية الجينية هي الأكثر شيوعاً وخوارزمية النحل الصنعية بيّنت أداء أفضل وأسرع من عدة خوارزميات تستخدم في مجال الاختبار للتطبيقات العادية، سنقدم في هذا الفصل دراسة لتوليد حالات اختبار لتطبيقات واجهات المستخدم الرسومية باستخدام خوارزمية النحل ثم مقارنة النتائج مع دراسة ولدت حالات الاختبار باستخدام الخوارزمية الجينية لنبين أنّ الأفضل والأسرع هي خوارزمية النحل تأكيداً لاستخدامها في مجال اختبار تطبيقات Android.

تتميز معظم التطبيقات الحالية بواجهة رسومية يتفاعل من خلاها المستخدم مع التطبيق، واجهة المستخدم الرسومية Graphical User Interface GUI هي واجهة أمامية رسومية هرمية البنية لنظام برمجي يقبل الأحداث التي ينشئها المستخدم أو النظام كمدخلات من مجموعة ثابتة من الأحداث وتنتج مخرجات رسومية. تحتوي واجهة المستخدم الرسومية على كائنات رسومية (مثل الأزرار والقوائم)، كل كائن لديه مجموعة ثابتة من الخصائص، خلال تنفيذ واجهة المستخدم الرسومية هذه الخصائص لها قيم منفصلة، تشكل هذه القيم حالة واجهة المستخدم الرسومية.
يمكن للمستخدم الوصول وتنفيذ وظائف التطبيق من خلال تسلسل محدد للأحداث تغير من حالة واجهة المستخدم الرسومية.

يمكن اعتبار GUI على أنها مجموعة عناصر مرتبطة بمعالجات للأحداث، يتم تعيين معالجة الأحداث للاستجابة لأحداث المستخدم، تختلف استجابة النظام للأحداث اعتماداً على الحالة الحالية للواجهة، والتي تم الوصول إليها من خلال الأحداث السابقة وترتيب التنفيذ.

تعتبر تطبيقات GUI من التطبيقات المقادة بالأحداث، لها بنية مختلفة عن التطبيقات التقليدية بالتالي لا يمكن تطبيق نفس الآليات أو استخدام نفس أدوات الاختبار المطبقة في التطبيقات العادية بشكل مباشر لاختبار تطبيقات Android، بيّنا في الفصل الثالث من خلال الدراسة [18] التي تم تطبيقها لتوليد حالات

اختبار للتطبيقات التقليدية أن خوارزمية النحل هي الأفضل من حيث السرعة والأداء سنقارن هذه الخوارزمية مع الخوارزمية الجينية لنبين الأسرع والأفضل من أجل التطبيقات المقادة بالأحداث ومنها تطبيقات GUI وتطبيقات Android.

2-4 اختبار تطبيقات GUI

انتشار تطبيقات GUI التي سهلت تفاعل المستخدم مع النظام البرمجي وضرورة التأكد من صحة تنفيذها وموافقتها لمتطلبات المستخدمين وعدم وجود أية أخطاء أو مشاكل في التطبيق جعل الحاجة ملحة لتقديم طرائق وآليات جديدة لدعم عملية اختبارها حيث أنّ الآليات المستخدمة لاختبار التطبيقات التقليدية لا تكفي ولا تناسب بشكل كبير هذه التطبيقات بسبب اختلاف بنية وطبيعة تطبيقات GUI.

تساعد مرحلة الاختبار على اكتشاف العيوب والأخطاء في البرنامج، خلال مرحلة الاختبار يتم توليد حالات اختبار وتنفيذها على البرمجية، توليد هذه الحالات يتم بشكل يدوي أو أوتوماتيكي باستخدام أداة، في تطبيقات GUI حالة الاختبار تمثل تسلسل من الأحداث ليتم تنفيذها على الواجهة[23].

اختبار تطبيقات GUI يتحقق من واجهة المستخدم الرسومية للتطبيق قيد الاختبار، يتضمن فحص الواجهات والنوافذ باستخدام عناصر التحكم مثل الأزرار والقوائم من خلال تجريب الأحداث المرتبطة بهذه العناصر، لا يكفي تنفيذ كل حدث بشكل مستقل وإنما يجب تنفيذه مع تسلسل أحداث أخرى، لذا فإنّ مصمم الاختبار بحاجة لتطوير حالات اختبار (تسلسل أحداث) تختبر كل الإدخالات الممكنة من المستخدم [8] .

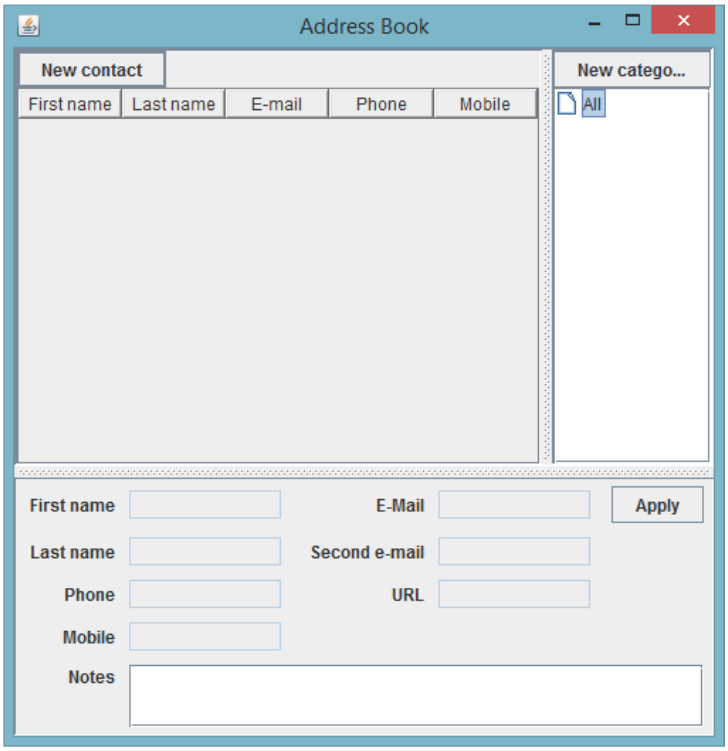
تتمثل المهمة الأساسية لاختبار واجهة المستخدم الرسومية في إنشاء حالات اختبار وتنفيذها للمرور واختبار جزء من الواجهة والأحداث التي عليها لاكتشاف الأخطاء بطريقة فعالة.

عملية اختبار هذه التطبيقات تبدأ من خلال توليد حالات الاختبار ممثلة كتسلسل من الأحداث يشابه تسلسل أحداث المستخدم مع التطبيق يتم إنشاؤها وتنفيذها على البرنامج، حيث أحداث المستخدم كبيرة جداً في معظم التطبيقات غير البسيطة، كما أنّ بعض الأحداث تكون قابلة للتنفيذ في سياق وتسلسل معين، ويمكن أن تسبب فشل في التطبيق إذا نفذت في سياق آخر.

تتطلب عملية اختبار هذه التطبيقات جهداً كبيراً بسبب طبيعة وهيكلية هذه البرمجيات، تم اقتراح تقنيات مختلفة لأتمتة اختبارها.

يمثل الشكل(4-1) تطبيقاً لإضافة بيانات مستخدم كواجهة، لاختبار هذا التطبيق سيتم تنفيذ سلسلة من الأحداث، مثلاً حالة الاختبار ستكون عبارة عن سلسة من الأحداث التالية

{New contact, First name, Last name} والتي تعني أن المستخدم سيضغط الزر New contact ثم يدخل بيانات في مربعات النص First name ثم Last name.



الشكل (4-1) واجهة لإدخال بيانات مستخدم

اختبار تطبيقات GUI يعد مكلفاً بسبب العدد الهائل الممكن لحالات الاختبار وهذا سيزيد من كلفة وزمن مرحلة الاختبار والذي سينعكس على عملية تطوير البرمجية ككل، تتجه الدراسة في هذا الفصل لتقليل هذه الكلفة من خلال التركيز على مرحلة توليد حالات الاختبار التي سيتم تنفيذها لتطبيقات واجهة المستخدم من خلال استخدام تقنيات الذكاء الصنعي وبشكل أخص خوارزميات البحث لتحسين عملية توليد حالات الاختبار.

4-3 الدراسة المرجعية:

عملية اختبار وظائف واجهات المستخدم مرحلة هامة وحرجة للتأكد من صحة تنفيذها وضمان جودتها، تستهلك تقنيات اختبار واجهات المستخدم الكثير من الوقت والجهد، تم توسيع معظم التقنيات المستخدمة في

اختبار التطبيقات البرمجية التقليدية لتلائم اختبار تطبيقات GUI، ومع ذلك هذه التقنيات لا تعمل إذا كان التطبيقات معقدة وفيها حالات كبيرة.

يوجد تقنيات عدة مستخدمة لاختبار تطبيقات GUI وهي:

الاختبار اليدوي: في هذه الحالة يتم اختبار الواجهات بواسطة المختبر وفقاً للمتطلبات المنصوص عليها في مستند المتطلبات.

الاختبار العشوائي: من خلال توليد حالات اختبار وأحداث عشوائية وتنفيذها، الاختبار العشوائي سهل التحقيق لكن غير فعال من ناحية عدد حالات الاختبار المولدة بسبب التكرار الممكن وفعالية هذه الحالات [24].

تقنية capture/reply : الاختبار هنا لا يحتاج لمعرفة بالكود البرمجي، المختبر يتفاعل مع واجهات التطبيق، تخزن المكونة capture تفاعلات المختبر في ملف ليتم إعادة استخدامها وتنفيذها من خلال المكونة reply، ومن مساوئ هذا النوع من الاختبار صعوبة إيجاد تسلسلات من الأحداث التي تكتشف الأخطاء، ويستهلك الكثير من الوقت حيث تبين بالتجربة [25] أن توليد حالة اختبار من 50 حدث لعناصر مختلفة على الواجهة تستغرق بين 20–30 دقيقة، وغالباً ما يحتاج إلى عدد من المختبرين لتوليد مجموعة اختبار كاملة، يتم استخدامها غالباً في الاختبار التكراري بعد تصحيح البرمجية لتنفيذ نفس الاختبارات السابقة [26].

الاختبار المعتمد على التصميم: على الرغم من انتشار استخدام هذه النوع لاختبار تطبيقات GUI إلا أنّ إنشاء نموذج يمثل النظام يعتبر مكلفاً، بذلت جهوداً لتطوير نماذج مختلفة لاختبار GUI، في [27] تم اعتماد مخطط الحالة لتمثيل سلوك التطبيق واشتقاق حالات اختبار منه، يؤدي أي حدث إدخال إلى الانتقال من حالة لحالة أخرى، المسار يمثل تسلسل الحواف أثناء الانتقالات بين الحالات والذي يمثل بدوره حالة اختبار، [28] تم تطوير أداة GUITAR لاشتقاق نموذج للواجهات اعتماداً على التحليل الستاتيكي من خلال الهندسة العكسية للتطبيق ثم اشتقاق حالات اختبار.

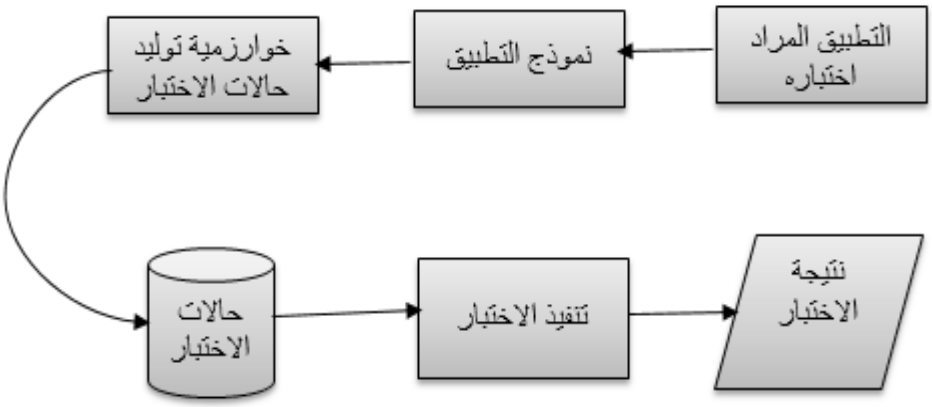
الاختبار المعتد على البحث: العديد من الدراسات اتجهت لاستخدام هذا النوع كما ذكرنا في الفصل الثاني حيث يتم النظر على مسألة البحث عن حالات الاختبار كمسألة تحسين لإيجاد حلول بأعلى جودة وفعالية، العديد من الدراسات استخدمت الخوارزمية الجينية لاختبار البرمجيات لتوليد حالات اختبار لتغطية الفروع،

في [29] استخدمت الخوارزمية الجينية لتوليد حالات اختبار لتطبيقات GUI حيث تم اقتراح نموذج يدوي لبناء مخطط التدفق للواجهات ثم توليد فضاء بحث ابتدائي لتوليد حالات الاختبار منه لكن لم يتم إنشاء أداة تمثل هذا النموذج المقترح وأثبتت الخوارزمية الجينية فعاليتها لاختبار التطبيقات المحدودة لكن مع زيادة حجم التطبيقات ستصبح غير ملائمة، [30] تم اقتراح استخدام خوارزمية النمل لتوليد حالات اختبار من نموذج للتطبيق مثّل كشجرة، [31] تم مقارنة الخوارزمية الجينية و Hill Climbing و Simulated Annealing لاشتقاق حالات الاختبار من مخطط تدفق الأحداث وبيّنت أداء أفضل للخوارزمية الجينية.

بناءً على ما سبق تم اعتماد نوع الاختبار المعتمد على البحث لتوليد حالات اختبار لتطبيقات GUI، وحسب ما بيّنا في الدراسات المرجعية لم تتم مقارنة الخوارزمية الجينية وخوارزمية النحل في مجال اختبار تطبيقات GUI، لذلك سنبين في هذا الفصل مقارنة الأداء والسرعة للخوارزميتين.

4-4 خطوات الخوارزمية المقترحة توليد حالات اختبار تطبيقات GUI:

يبيّن الشكل (4-2) المكونات الأساسية لعملية اختبار تطبيقات GUI، بداية يتم اشتقاق نموذج يمثل التطبيق وتخزينه ضمن بنية معينة (مصفوفة، ملف xml)، بعدها يتم قراءة هذا النموذج لاشتقاق حالات اختبار باستخدام خوارزمية النحل، الجزء المهم هنا آلية عمل الخوارزمية، بعد هذه المرحلة يتم تمرير حالات الاختبار للتطبيق ليتم تنفيذها ومراقبة حالة النظام، لتمثيل التطبيق استخدمنا مخطط تدفق الأحداث.



الشكل (4-2) تسلسل تنفيذ خوارزمية اختبار تطبيقات GUI

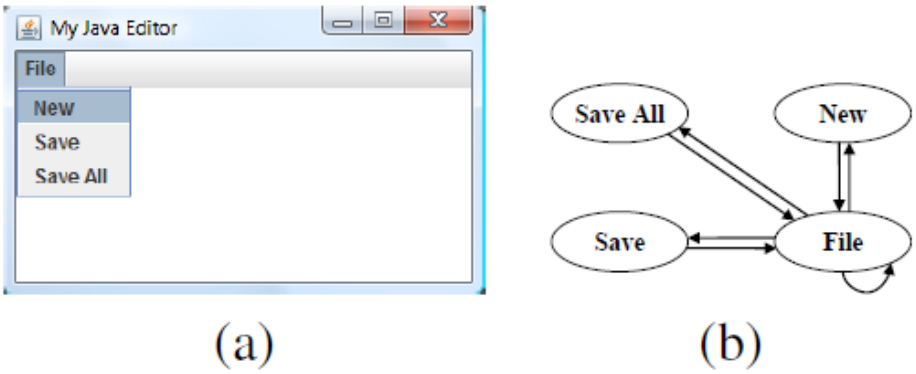
4-4-1 مخطط تدفق الأحداث Events Flow Graph

لتمثيل تطبيق GUI تم استخدام نموذج مخطط تدفق الأحداث، تدفق أحداث واجهة المستخدم الرسومية خلال التنفيذ يمكن تمثيلها كرسم بياني، يمثل مخطط تدفق الأحداث جميع التفاعلات الممكنة للمستخدم، يتكون المخطط من عقد تمثل كلاً منها حدث في الواجهة والحواف تمثل الانتقال من حدث لحدث آخر استجابة لتفاعل المستخدم، المعيار لاختبار الواجهة هو تغطية عالية للتطبيق لتوليد حالات اختبار، هناك عدد كبير من المسارات في مخطط التدفق.

يمثل مخطط تدفق الأحداث لواجهة <V, E, B> حيث :

- تمثل V العقد والتي بدورها تمثل الحدث المرتبط بعنصر ما على الواجهة
- تمثل E الحواف الموجهة بين العقد، $E \subseteq V \times V$، ei (Vx , Vy) يمثل حدث Vy يتبع الحدث Vx
- تمثل B الأحداث البدائية الممكنة للمستخدم عندما يتم استدعاء الواجهة $B \subseteq V$

الشكل (4-3) يمثل واجهة لتطبيق يقدم ميزات بسيطة لتحرير النصوص ومخطط تدفق الأحداث المقابل لهذه الواجهة.



الشكل (4-3) واجهة تطبيق لتحرير النصوص مع مخطط الأحداث لها

المرور بكل المسارات لتوليد كل حالات الاختبار مسألة مكلفة، اختيار الحد الأدنى من هذه الحالات هو التحدي في عملية الاختبار من خلال استخدام وتوجيه عملية البحث ضمن المخطط باستخدام خوارزمية النحل الصنعية.

عدة معايير تم تقديمها لاختبار واجهة المستخدم الرسومية مثل تغطية العقد والحواف لمخطط تدفق الأحداث، اختيار المعيار الصحيح لتابع الملاءمة أمر مهم لإيجاد التسلسل الأمثل للأحداث.

سنذكر فيما يلي بعض التعريفات المتعلقة بعملية اختيار المسارات الأساسية للاختبار [32].

المسار المستقل Independent Path: أيّ مسار خلال البرنامج يقدم على الأقل مسار مجموعة جديدة من التعليمات، عند رسم مخطط تدفق الأحداث فإنّ المسار المستقل يجب أن يتضمن على الأقل حدثاً جديداً لم يتم تغطيته في المسارات المولدة مسبقاً.

التعقيد الدائري Cyclomatic Complexity CC: مقياس برمجي يشير إلى مدى تعقيد البرنامج، مقياس كمي لعدد المسارات المستقلة خلال الكود، يتم حساب قيمة هذا التعقيد باستخدام مخطط التدفق للبرنامج حيث عقد المخطط تمثل تعليمات أو أجزاء مستقلة من الكود والحواف تمثل تعليمة أو حدث (أمر) سيتم تنفيذها مباشرة بعد التعليمة الأولى للانتقال للتعليمة التالية.

اختبار المسارات الأساسية Basis Paths Testing: إحدى استراتيجيات الاختبار يتم فيها اختبار كل مسار مستقل خطياً من خلال البرنامج، في هذه الحالة عدد حالات الاختبار سيساوي التعقيد CC للبرنامج، إذا تم تصميم حالات اختبار بحيث تغطي المسارات الأساسية هذا يعني أنه تم تغطية كل الكود الذي يتم اختباره، كل مسار من هذه المسارات الأساسية سيغطي على الأقل عقدة أو جزء في المخطط لم يغطى من قبل.

لحساب قيمة CC لبيان باستخدام العلاقة التالية[32] :

$$CC = \sum (V(g) - 1) + 1$$

حيث V(g) تمثل عدد الحافات التي تنبثق من العقدة.

وبإسقاط المعادلة على البيان وحسب [32] فإنّ البيان يتم تمثيله كمصفوفة تمثل العلاقات بين العقد، قيمة كل عنصر من المصفوفة 1 أو 0 يمثل إن كان يوجد انتقال بين الحدثين أم لا، إذا طبقنا المعادلة على المصفوفة المبينة في الشكل (4-4) على فرض لدينا في واجهة ثمانية أحداث والانتقالات بينها ممثلة من

خلال قيم عناصر المصفوفة، فإنّ قيمة CC=4 يتم حسابها كما هو موضح في الشكل (4-4)، أعمدة المصفوفة تمثل عقد البيان، والأسطر تمثل العلاقة بين هذه العناصر.

	1	2	3	4	5	6	7	8	
1	0	1	0	0	0	0	0	0	1 – 1 = 0
2	0	0	1	0	0	0	0	0	1 – 1 = 0
3	0	0	0	1	0	1	0	0	2 - 1 = 1
4	0	1	0	0	1	0	0	0	2 – 1 = 1
5	0	0	0	0	0	0	0	0	---
6	0	0	0	0	0	0	1	1	2 – 1 = 1
7	0	0	0	1	0	0	0	0	1 – 1 = 0
8	0	0	0	0	0	0	1	0	1 – 1 = 0

3+1=4

الشكل (4-4) كيفية حساب قيمة CC اعتماداً على مصفوفة تمثل تدفق الأحداث لواجهة

اعتماداً على هذه المفاهيم وانطلاقاً من البيان الذي يمثل التطبيق الذي يعتبر فضاء البحث البدائي لخوارزمية النحل الصنعية، سنوضح فيما يلي خطوات الخوارزمية لتوليد حالات الاختبار، مع العلم أنه تم بناء مخطط تدفق الأحداث للواجهات بشكل يدوي باعتبار أن تركيزنا في هذا الفصل على مقارنة أداء خوارزميات البحث.

4-4-2 استخدام خوارزمية النحل الصنعية

4-4-2-1 إنتاج مواقع مصادر الغذاء الأولية

تم توليد حالات اختبار (تمثل حالات الاختبار بتسلسل من الأحداث) بشكل عشوائي اعتماداً على المصفوفة التي تمثل بيان تدفق الأحداث ومعلومات عن الأحداث الأولية للتطبيق والأحداث اللاحقة للحدث الحالي، تعتبر هذه الحالات المولدة هي فضاء البحث الابتدائي للخوارزمية، تم أخذ مسارات عددها يساوي قيمة CC وهذا بدوره يمثل عدد مصادر الطعام وبالتالي عدد كلاً من النحل العامل والنحل المشاهد، N حجم الخلية يساوي عدد النحل العامل والمشاهد.

4-4-2-2 إرسال النحل العامل إلى مصادر الغذاء

كل نحلة عاملة ترتبط مع حالة اختبار، تحدث النحلة العاملة التعديل على حالة الاختبار اعتماداً على العلاقة التالية لإيجاد حل جديد، التعديل هنا يمَّل في تغيير أحد الأحداث لحالة الاختبار، وهذه العلاقة هي الشكل الأساسي للبحث عن حلول جديدة في خوارزمية النحل

$$vij = xij + \vartheta ij\,(xij - xkj) \qquad (1)$$

xij الحل الحالي، K هو رقم عشوائي يتم اختياره ضمن المجال [N .... 1] ويختلف عن i، ϑij رقم عشوائي ضمن المجال [1 ,1−]، بعد إنتاج الحل الجديد vij يتم تقييمه اعتماداً على قيمة تابع الملاءمة، تابع الملاءمة تم حسابه اعتماداً على الأحداث والاستدعاءات المتتالية التي تمت تغطيتها في حالة الاختبار بالنسبة للطول الكلي لحالة الاختبار.

يتم اختيار الأفضل بين xi وvi اعتماداً على قيمة تابع الملاءمة، إذا كانت قيمة تابع الملاءمة للحل vi أعلى من xi عندها يتم استبدال الحل القديم بالحل الحالي، وإلا يتم الاحتفاظ بالحل السابق ويتم زيادة عداد التجارب بـ 1.

بعد أن يتم كل النحل العامل عمليات البحث ومحاولة التحسين للحل، يعطي هذه المعلومات للنحل المشاهد لتقييم هذه المعلومات، النحلة المشاهدة تقوم بتقييم المعلومات المأخوذة من كل النحل العامل وتختار موقع الحل بالاعتماد على الاحتمالية.

هذا الاختيار الاحتمالي يعتمد على قيمة تابع الملاءمة للحلول في المجتمع كما موضح في المعادلة الآتية:

$$pi = a * \frac{fitness(i)}{\max(fitnessi)} + b \qquad (2)$$

حيث a,b تقع قيمتما في المجال [0 − 1]

4-4-2-3 تقييم الحل من قبل النحل المشاهد على أساس المعلومات المقدمة من النحل العامل

يتم توليد عدد حقيقي عشوائي ضمن نطاق[1 – 0] لكل مسار. إذا كانت القيمة الاحتمالية Pi في المعادلة 2 المتعلقة بذلك المسار هي أكبر من هذا الرقم العشوائي فالنحل المشاهد تقوم بالتعديل على هذا المسار عن طريق استخدام المعادلة 1 كما هو الحال بالنسبة للنحلة العاملة.

بعدها يتم تقييم الحل الجديد بتطبيق الاختيار الأفضل لقيمة تابع الملاءمة، والنحل المشاهد إما يخزن الحل الجديد مع نسيان القديم أو يبقي الحل القديم، إذا لم يتم تحسين الحل يتم زيادة عداد التجارب بـ 1 وإلا يتم تعيينه إلى 0، وتتكرر هذه العملية حتى يتم توزيع جميع النحل المشاهد على المسارات.

4-4-2-4 استخدام النحل الكشاف

في الدورة الواحدة بعد أن تكمل كل من النحل العامل والمشاهد عمليات البحث الخاصة بهم، الخوارزمية تدقق لمعرفة إذا كان هناك أي مصدر يمكن التخلي عنه، يتم التخلي عن المصدر الذي تجاوز عدد التجارب له قيمة محددة للخوارزمية، يصبح هذا المصدر مستنفذاً ويتم التخلي عنه ليبدأ النحل الكشاف البحث عن حل أفضل (حالة اختبار أفضل) من خلال وتوليد حل جديد عشوائي كما في بداية الخوارزمية.

4-5 القسم العملي

تم تحقيق الخوارزمية المقترحة باستخدام لغة البرمجة Java وبيئة التحقيق Eclipse، بناء مخطط تدفق الأحداث للواجهة تم بشكل يدوي باعتبار أن تركيزنا في هذا الفصل على مقارنة خوارزميات البحث، ثم تمرير هذا المخطط لخوارزمية النحل التي تمت ملاءمتها للبحث ضمن الفضاء البدائي وتوليد حالات الاختبار.

تم اختيار التطبيقات الموضحة في الشكل (4-5) لتدريب الخوارزمية، حيث تم اشتقاق مخطط تدفق الأحداث للواجهة، ثم بناء مصفوفة التفاعل فيما بين الأحداث وحساب قيمة CC لكل تطبيق،

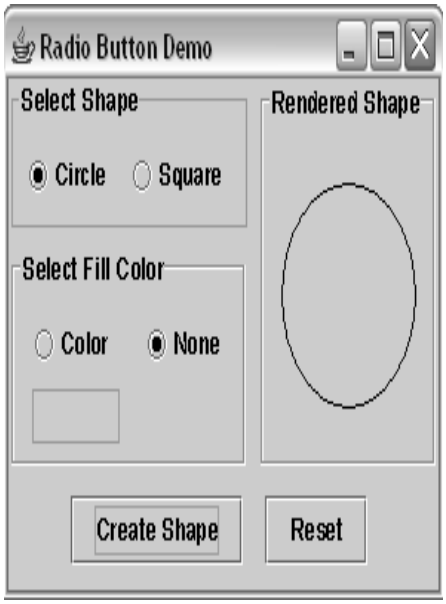
لتقييم الخوارزمية المقترحة ومقارنتها مع أعمال سابقة تم اختيار الخوارزمية الجينية المقدّمة في [33]، أعدنا تنفيذ الخوارزمية الجينية بنفس قيم البارامترات

Crossover Rate=0.8 ،Mutation Rate=0.2

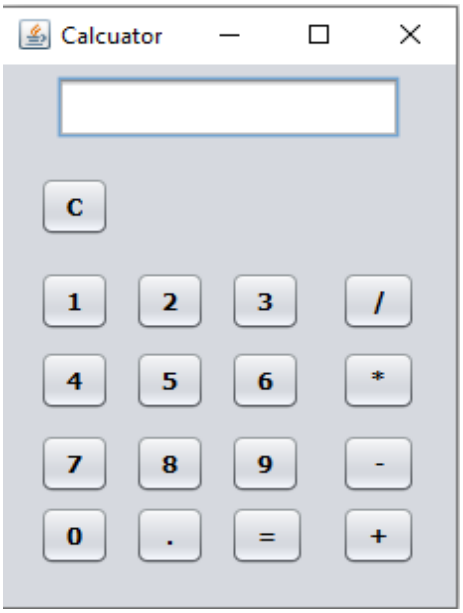
خطوات تسلسل الخوارزمية الجينية موضحة فيما يلي:

START
Generate the initial population
Compute fitness
REPEAT
Selection
Crossover
Mutation
Compute fitness
UNTIL population has converged
STOP

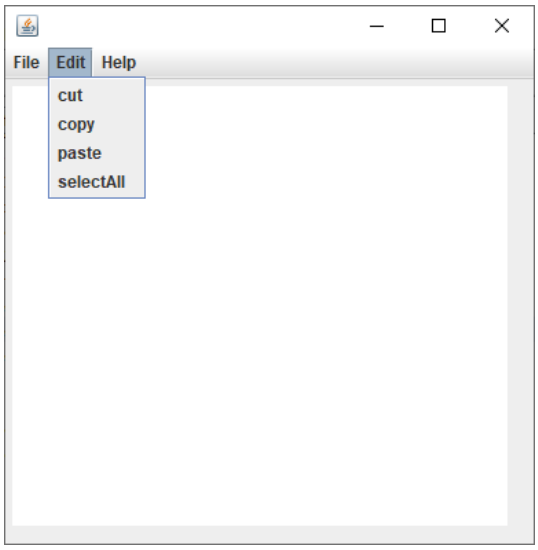
بعد اختيار التطبيقات وتوليد مخطط الأحداث لها تم حساب قيمة التعقيد الدائري CC، ثم تنفيذ الخوارزمية الجينية GA وخوارزمية النحل المقترحة ABC في عملنا هذا ومراقبة النتائج، من حيث زمن التنفيذ اللازم لتدريب الخوارزمية للوصول للنتيجة المطلوبة ومتوسط تابع الملاءمة خلال دورات التدريب، وتبيّن كما هو موضح في الجدول (4-1) أنّ خوارزمية ABC تعطي أداء أفضل من حيث قيمة تابع الملاءمة وأسرع من الخوارزمية الجينية.



(a) P1



(b) P3



(c) P2

الشكل (4-5) تطبيقات GUI المستخدمة لمقارنة تنفيذ خوارزميتي النحل والجينية

الجدول (4-1) مقارنة تنفيذ خوارزميتي ABC وGA

البرنامج	التعقيد الدائري CC	عدد تكرارات التدريب	متوسط زمن التنفيذ (ميللي ثانية)		متوسط تابع الملاءمة	
			ABC	GA	ABC	GA
P1	30	10-1	57.5	63.01	92.93	87.5
		20-11	213.03	237.04	93.5	88.5
		50-21	293.75	303.8	94.27	93.2
		70-51	360	395.1	97	95.4
P2	52	10-1	127.2	156.3	85.67	75
		20-11	209.18	225.64	88.71	82.1
		50-21	359.58	498.01	92.5	87.5
		70-51	450.6	615.5	95	90.5
P3	214	10-1	562.7	627.8	92	85.42
		20-11	1195.7	1349.4	95.6	87.55
		50-21	2792.4	3030.5	97.2	92.73
		70-51	3550.7	3860	99	96

6-4 خاتمة

قدمنا في هذا الفصل دراسة لكيفية اختبار تطبيقات GUI، بينا الآليات المتوفرة لتوليد حالات الاختبار، ثم تم تقديم آلية لتوليد حالات الاختبار باستخدام منهجية الاختبار المعتمد على البحث اعتماداً على نموذج مخطط تدفق الأحداث لعناصر الواجهة ثم تمرير هذا النموذج لخوارزمية النحل للبحث عن حالات اختبار أمثلية، بعدها تمت مقارنة خوارزمية النحل مع الخوارزمية الجينية وتبيّن أن خوارزمية النحل أسرع وتعطي قيمة أفضل لتابع الملاءمة.

تم اعتماد خوارزمية النحل الصنعية لتحقيق الأداة BeeDroid المستخدمة لتوليد حالات اختبار لتطبيقات Android كما سنبيّن في الفصل الخامس، حيث سنقدم بدايةً دراسة لتطبيقات Android وخصائص هذه التطبيقات ومكوناتها، ثم آلية تمثيلها من خلال بيان يمثل الأحداث واستدعاءات نظام Android للتطبيق، ثم اقتراح الأداة BeeDroid التي سيكون دخلها ملف apk يمثل التطبيق مراد اختباره والخرج هو ملف يثمل حالات الاختبار المولّدة للتطبيق.

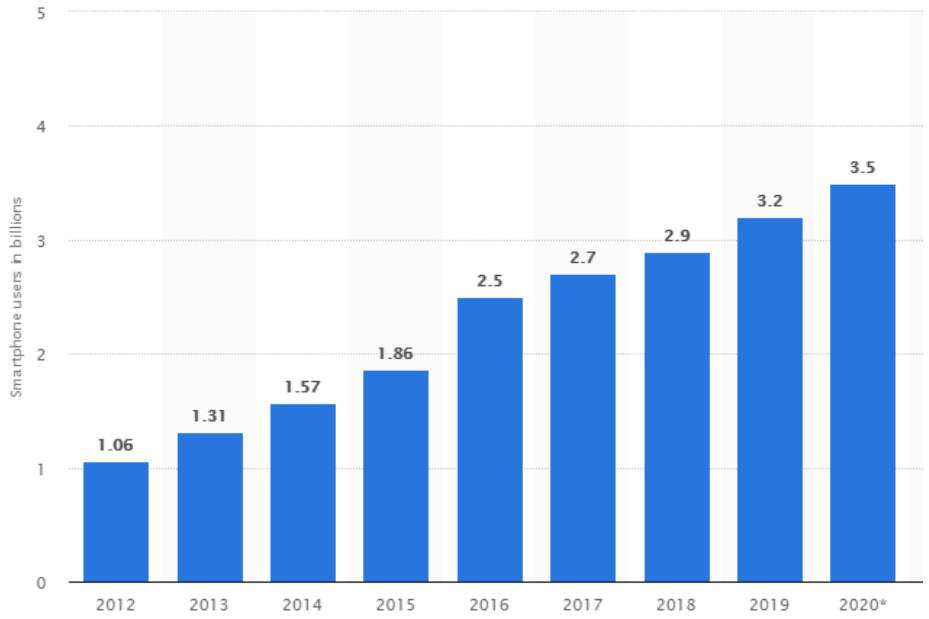
الفصل الخامس: توليد حالات اختبار لتطبيقات Android

5-1 مقدمة

قدمنا في الفصل السابق دراسة بيّنت أنّ خوارزمية النحل أفضل وأسرع من الخوارزمية الجينية في مجال توليد حالات اختبار لتطبيقات GUI، سنستخدم النتائج التي توصلنا لها لتطوير أداة BeeDroid تستخدم خوارزمية النحل لتوليد حالات اختبار لتطبيقات Android.

انتشرت الهواتف النقالة بشكل واسع وتنوعت أنظمة التشغيل التي تدعم هذه الأجهزة، أكثرها استخداماً نظام Android، انتشرت تطبيقات Android بشكل واسع وتنوعت الوظائف التي تقدمها وتلبي حاجات المستخدمين.

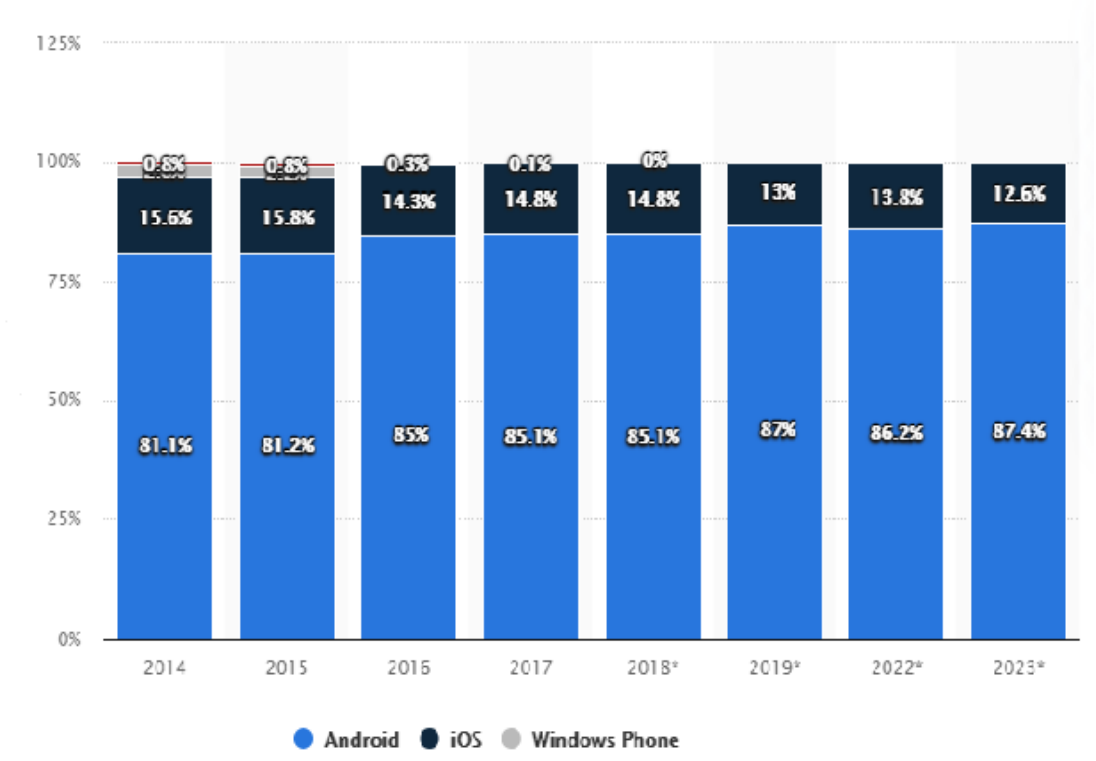
يستخدم الناس أجهزة الهاتف المحمول حالياً أكثر من أجهزة الكمبيوتر للوصول للأنترنت أو لمتابعة أعمالهم اليومية أو للتواصل مع بعضهم، حسب [34] عام 2016 وصل عدد مستخدمي الهواتف الذكية لـ 2.5 بليون بينما حسب إحصائيات حديثة وصل العدد عام 2020 لـ 3.5 بليون مستخدم، يبيّن الشكل (5-1) تزايد عدد المستخدمين بين عامي 2012 و2020، شهدت أجهزة الموبايل تطوراً كبيراً بدأت كأجهزة بطيئة محدودة الاستخدام للاتصال بدون إمكانيات للتعامل مع الصور والألوان وتطورت خلال العقد الماضي لتصبح أجهزة متفوقة من ناحية المعالجة والتخزين، غيّرت هذه الأجهزة وأثرت على محاور عدة في حياة الناس سواء في مجال التسوق أو التعليم أو التواصل مع الآخرين.



الشكل (5-1) عدد مستخدمي أجهزة الهواتف الذكية بين عامي 2012 و2020

تطبيقات الموبايل هي برمجيات تحمّل وتنفذ على أجهزة الموبايل، عوامل عديدة ساهمت في انتشار أجهزة الموبايل بشكل كبير منها رخص هذه الأجهزة وانتشار شبكة الأنترنت والميزات التي تقدمها تطبيقات هذه الأجهزة، أنظمة التشغيل Android, IOS, Windows صممت لأجهزة الموبايل، الأكثر شيوعاً هو نظام التشغيل Android نظراً لتنوع الأجهزة التي تدعمه وتنوع أسعارها ما ساهم في انتشار هذه الأجهزة أكثر، الشكل (5-2) يبيّن انتشار استخدام أنظمة تشغيل أجهزة الموبايل بين عامي وتوقع 2012 انتشار استخدامها لعام 2022، هيمن نظام Android وكان الأوسع انتشاراً واستخداماً، خلال الأعوام الماضية تقريباً حوالي 85% من الأجهزة استخدمت نظام أندرويد بينما 14% من الأجهزة استخدمت نظام IOS وأجهزة Windows كانت فقط 1%[35] .

تطبيقات Android هي برمجيات تعمل على الأجهزة الخاصة بأنظمة التشغيل Android، تتطور وتنمو هذه التطبيقات من ناحية عددها وعدد المستخدمين وعدد المطورين لهذه التطبيقات، عام 2016 وصل عدد تطبيقات Android التي تم تطويرها لـ 2 مليون، وتجاوز عددها خلال شهر آذار عام 2017 الـ 2.3 مليون تطبيق وحالياً عدد التطبيقات تجاوز الـ 3.04 مليون تطبيق حسب إحصائيات Google Play لذي يعتبر متجر التطبيقات الرسمي للأجهزة المعتمدة على نظام تشغيل Android، الشكل (5-3) يبين تزايد عدد التطبيقات خلال عامي 2019 و2020 [36].



الشكل (5-2) أنظمة تشغيل أجهزة الموبايل المستخدمة بين عامي 2012 و2020

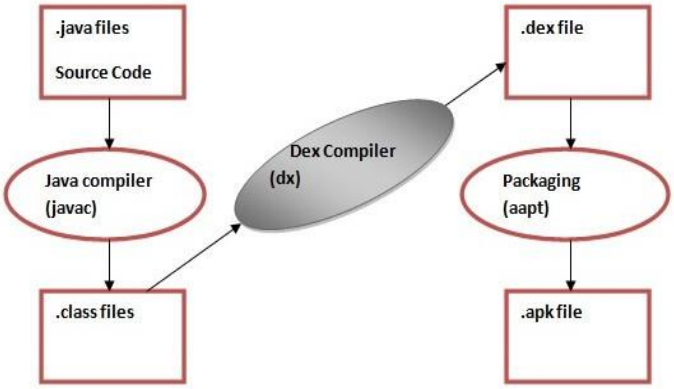
الشكل(5-3) عدد تطبيقات Android خلال عامي 2019 و2020

5-2 خصائص تطبيقات Android

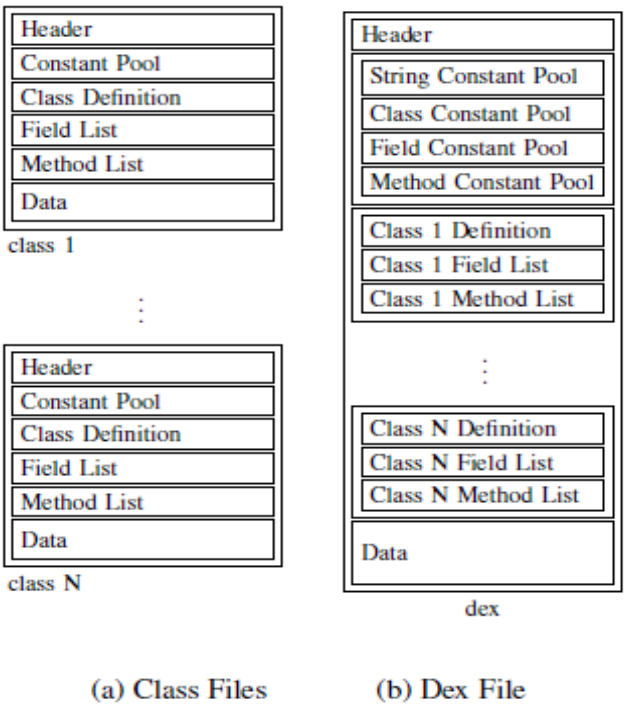
نظام Android هو نظام تشغيل مجاني ومفتوح المصدر مبني على أساس نواة Linux صمم أساساً للأجهزة ذات شاشات اللمس كالهواتف الذكية والحواسيب اللوحية، يمكن للمطورين إنشاء وتطوير تطبيقات Android باستخدام مجموعة أدوات تطوير برامج Android.

تُكتب تطبيقات Android [37] غالباً بلغة البرمجة Java لكن لا يتم توزيعها على أنها Java Bytecode لكن Dalvik Bytecode، يتم بداية تجميع كود التطبيق كـ Java Bytecode ثم بعد ذلك يتم تحويلها إلى Dalvik Bytecode وتخزن بلاحقة dex كما هو موضح في الشكل (5-4).

تم استخدام تنظيم Dalvik المضغوط للأنظمة المقيّدة من حيث الذاكرة وسرعة المعالج لتحسين الأداء، بهدف تحسين أداء الذاكرة فإنّ Dalvik لها ميزات أساسية عن بيئة Java الافتراضية حيث ملف dex التنفيذي أصغر حجماً وكل الصفوف تتشارك نفس constant pool، كما أنّ آلة جافا الافتراضية تعتمد على بنية المكدسات وهذا يتطلب تعليمات أكثر ووقت أكبر للوصول للذاكرة بينما آلة Dalvik الافتراضية تعتمد على بنية السجلات بالتالي يتم الوصول للبيانات مباشرة بين السجلات وأسرع من التعامل مع المكدسات، يبين الشكل (5-5) الفرق بين الملفات البايتية لجافا class. وملفات dex. ، في بيئة Java الافتراضية تتم ترجمة كل صف لملف class. منفصل، بينما يتم تجميع كل الصفوف في ملف dex. واحد



الشكل (5-4) تحويل Java Bytecode لتنسيق dex



الشكل (5-5) الفرق بين ملفات class. وملفات dex.

تتكون تطبيقات Android من عدة مكونات (سنذكرها فيما بعد) وتستخدم أغراض Intent للتفاعل والاتصال بينها، عادةً أغراض الـ Intent تحمل البيانات المطلوبة للمكونات التي طلبتها، أغراض الـ Intent لها دور أساسي في تطبيقات Android، تطبيقات Android لها خصائص وميزات تختلف عن التطبيقات الأخرى، فيما يلي سنذكر أهم هذه الخصائص [37] :

5-2-1 مكونات تطبيقات Android

5-2-1-1 النشاط Activity

يمثل النشاط شاشة (واجهة المستخدم الرسومية)، يتضمن النشاط عناصر مثل الأزرار وعناصر إدخال النصوص، من خلال إعادة تحقيق طرائق النشاط يمكن إضافة تحقيق واستجابة لعناصر الواجهة، يتم فصل المنطق البرمجي للواجهة عن بنية التصميم من خلال وضع التصميم وتوضّع العناصر في ملف xml.

يمثل الشكل (5-6) دورة حياة النشاط، للنشاط ثلاث حالات أساسية Running، Paused، Stopped ، بعد طلب النشاط ليتم عرضه يتم استدعاء الطرائق onCreate(), onStart(), onResume() على التتالي قبل وصول النشاط للحالة Running عندها سيكون النشاط مرئي على الشاشة للمستخدم، ينتقل النشاط للحالة Paused إذا أطلق المستخدم نشاط آخر لكن هذا النشاط لا يزال يغطي جزء من الشاشة فقط عندها النشاط الحالي سيستدعي الطريقة onPause()، إذا أصبح النشاط الجديد يغطي الشاشة والنشاط السابق لم يعد يظهر عندها ينتقل النشاط للحالة Stopped.

5-2-1-2 الخدمات Services

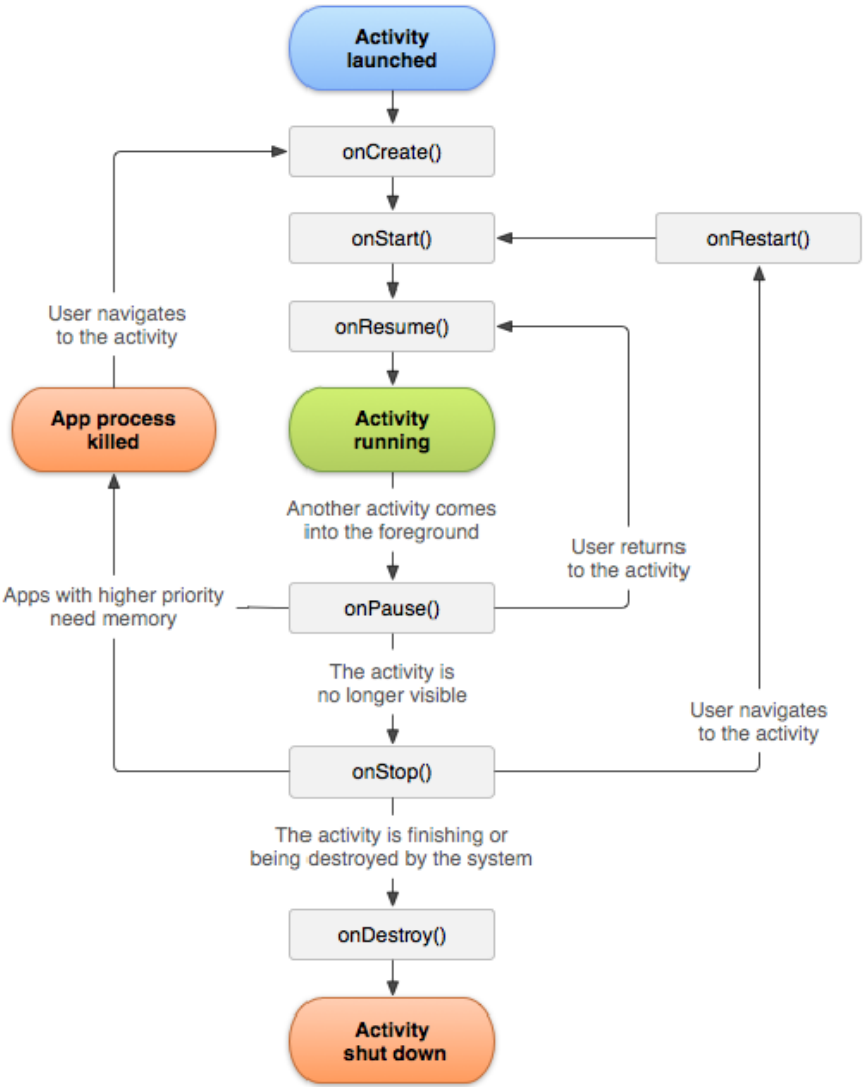
الخدمات لا تملك واجهة رسومية، تعمل في الخلفية، تستخدم عادة لأداء مهام طويلة مثل تشغيل المنبه أو الموسيقى، يمكن بدء الخدمة باستخدام مكونات أخرى مثل الأنشطة.

5-2-1-3 مستقبل البث Broadcast Receiver

يمكن لتطبيقات Android إرسال أو استقبال رسائل من النظام أو تطبيقات أخرى، هذه الرسائل يتم إرسالها عند حدث معين، نظام Android يرسل Broadcast عند أحداث النظام المختلفة مثلاً عند بدء شحن الجهاز أو انخفاض شحن البطارية، يمكن للتطبيقات أيضاً أن ترسل Broadcast مخصصة لإعلام التطبيقات الأخرى بحدث ما مثلاً عند انتهاء تحميل بيانات، الشكل (5-7) يوضح مثالاً.

5-2-1-4 مزوّد المحتوى Content Provider

يسمح بتوفير وإدارة الوصول للبيانات بشكل آمن من تطبيق لتطبيق آخر، تلك البيانات التي يتم تخزينها في قواعد البيانات أو نظام ملفات أندرويد مثل الصور وجهات الاتصال.



الشكل (5-6) دورة حياة النشاط

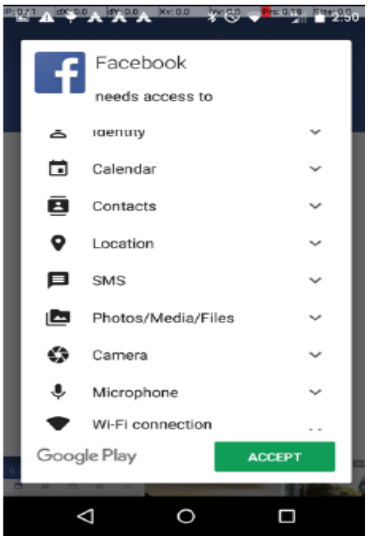
الشكل (5-7) مثال عن Broadcast

5-2-2 تحليل واجهات المستخدم

تعتبر تطبيقات Android تفاعلية بشكل كبير من خلال واجهات المستخدم والعناصر التي تظهر عليها، تستخدم تطبيقات أندرويد ملفات xml بشكل كبير سواءً لتعريف الإعدادات أو تعريف ملفات الـ Layout التي تحدد العناصر التي ستظهر على الواجهة وترتيبها وتوضعها فهي تمثل الجزء المرئي من الواجهة، برمجيات واجهات المستخدم التقليدية قليلاً أو ربما نادراً أن تحتوي على ملفات xml لذلك تقنيات تحليلها كانت تعتمد على شيفرة التطبيق الأساسية فقط ولا تأخذ بعين الاعتبار ملفات xml، في تطبيقات Android يجب أن يتم تضمين هذه الملفات في عملية التحليل لأهمية المعلومات التي تحويها، وقد يتم ربط عناصر التحكم على الواجهة بمعالجة الأحداث يجب أيضاً معالجتها عند التحليل لإنشاء نموذج كامل للتطبيق، لذلك أيّ أداة تحليل يجب أن تحلل وتعالج ملفات xml لكل الأنشطة لاستخراج عناصر الواجهات وخصائصها.

5-2-3 سماحيات تطبيقات Android

لحماية أمن نظام تشغيل Android وخصوصية المستخدمين يتم تنفيذ كل تطبيق ضمن sandbox منفصل بسماحيات محدودة، إذا احتاج التطبيق الوصول لموارد النظام أو بيانات المستخدم يجب التصريح عن هذه السماحيات في ملف الـ AndroidManifest.xml، وعند تثبيت التطبيق سيطلب من المستخدم السماح بالوصول للموارد، مثلاً الشكل (5-8) يمثل السماحيات المطلوبة لتثبيت تطبيق Facebook.



الشكل(5-8) السماحيات المطلوبة لتطبيق Facebook

5-2-4 التفاعل بين مكونات التطبيق وأزرار الجهاز

كل جهاز Android له الأزرار الموضحة في الشكل (5-9)، تؤثر هذه الأزرار على آلية تنفيذ التطبيق، مثلاً الزر الذي على اليسار Back يجعل المستخدم ينتقل للشاشة السابقة، الزر Back يقطع تسلسل التنفيذ الحالي ويجب أخذ هذه الحالات بعين الاعتبار عند تحليل أو اختبار هذه التطبيقات كونها ستؤثر على آلية وتنفيذ التطبيق.

الشكل (5-9) أزرار أجهزة Android

عند اختبار تطبيقات Android يجب تحليل كل مكونات التطبيق وأخذ حالات كل مكون ودورة حياته بعين الاعتبار وكيفية استجابته لأحداث النظام، بالإضافة لتحليل ملف AndroidManifest.xml الموجود في كل تطبيقات Android، يمثل الملف تعريفاً لمكونات التطبيق حيث يسجل المكونات الموجودة في التطبيق بما فيها الأنشطة الموجودة وأنشطة البداية للتطبيق وباقي المكونات والسماحيات المطلوبة، لذلك يجب عند تحليل التطبيقات المرور على الأنشطة ودورة حياتها من خلال تحليل الطرائق التي تم إعادة كتابتها، وكذلك باقي المكونات ليتم تمثيلها [38] .

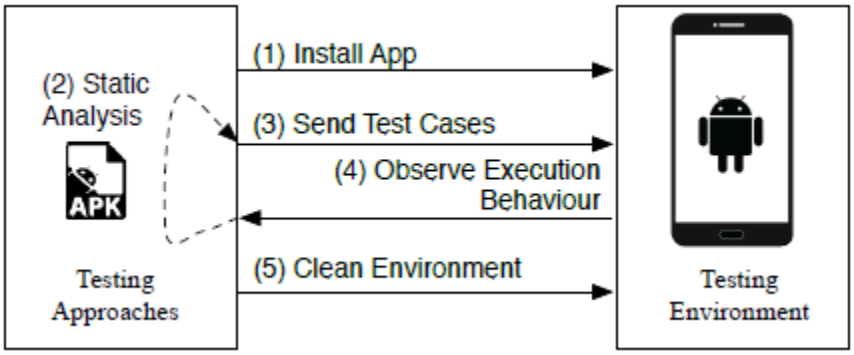
5-3 اختبار تطبيقات Android

تهدف عملية اختبار تطبيقات Android [39] إلى اختبار وظائف التطبيقات التي تعمل، يمكن أن نصنف عملية اختبار هذه التطبيقات إلى:

- Local Unit Tests: يتم الاختبار ضمن بيئة Java الافتراضية دون محاكاة استدعاءات نظام Android.
- Instrumentation Tests: يتم اختبار التطبيق ضمن بيئة Android ومحاكاة استدعاءات وحالة النظام، ركزنا في عملنا على هذا النوع من الاختبار.

يوضح الشكل (5-10) مراحل عملية اختبار تطبيقات Android والتي تبدأ بتثبيت التطبيق على الجهاز المراد تنفيذ الاختبار عليه، ثم تحليل التطبيق لاستخراج حالات اختبار منه وتعتبر هذه المرحلة جوهرية وأساسية وتختلف آلية تنفيذها باختلاف الطريقة المتبعة لتوليد حالات الاختبار بعضها يعتمد على توليد عشوائي للحالات وبعضها يدوي، وهذا ما سنتكلم عنه فيما بعد، بعد توليد حالات الاختبار يتم إرسالها لجهاز

Android لتنفيذ التطبيق حسب الحالات، يتم مراقبة سلوك التطبيق وتنفيذه وآلية استجابته، في الخطوة الأخيرة يتم إزالة تثبيت التطبيق تمهيداً لعملية اختبار لاحقة.



الشكل (5-10) عملية اختبار تطبيقات Android

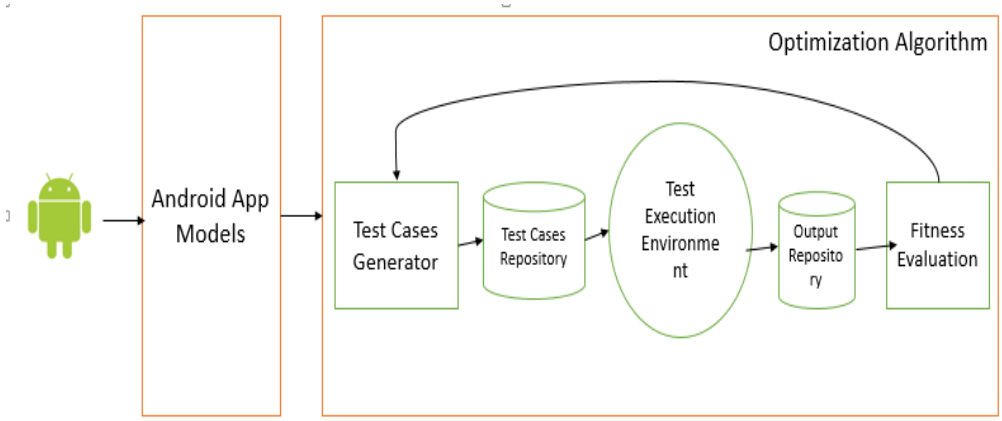
تحديات عدة تواجه عملية اختبار تطبيقات Android ذكرناها في الفصل الأول أهمها يكمن هي كيفية إنشاء تغطية أعظمية لشيفرة التطبيق من خلال حالات الاختبار المولدة من أجل إيجاد العيوب في التطبيق، تطبيقات Android مقادة بالأحداث بالإضافة للاستدعاءات من نظام التشغيل للتطبيق وهذا يشكل المزيد من التحديات والصعوبات لتوليد حالات اختبار أمثليه.

يمكن تصنيف أبحاث اختبار تطبيقات Android حسب عدة أبعاد[39] :

1. الغاية من الاختبار Test Objectives: حسب الغاية من الاختبار إذا كان لاختبار الوظائف واكتشاف الأخطاء أو اختبار الأداء للتطبيق أو اختبار الأمن، ركزنا في بحثنا على اختبار وظائف التطبيق ومحاولة اكتشاف أي أخطاء أو استثناءات عند التنفيذ.
2. أهداف الاختبار Test Targets: هل التركيز سيكون فقط على أحداث المستخدم وتفاعله مع عناصر واجهات التطبيق، أو اختبار التفاعل بين مكونات التطبيق فيما بينها، أو بين التطبيقات، تم في بحثنا اختبار كامل مكونات التطبيق مع الأخذ بعين الاعتبار مكونات الواجهات وأحداث المستخدم واستدعاءات النظام من خلال تحليل التطبيق كما سيتم ذكره لاحقاً.
3. مستويات الاختبار Test Levels: حسب الجزء المختار من التطبيق لاختباره، مثلاً اختبار الوحدة Unit Testing يهدف لاختبار كل جزء أو وظيفة من التطبيق، اختبار التكامل Integration Testing لاختبار تكامل تنفيذ عدة أجزاء من التطبيق مع بعضها، اختبار النظام System Testing لاختبار التطبيق بشكل كامل ضمن البيئة التي سيعمل ضمنها وهو المستوى الذي تم اعتماده في بحثنا.

4. منهجيات الاختبار Test Methodologies: تحدد المنهجية الأساسية المتبعة لتوليد بيانات وحالات الاختبار، سيتم ذكرها بالتفصيل فيما بعد.

عملية اختبار تطبيقات Android ستمر بعدة مراحل موضحة في الشكل (5-11) حيث يوضح الشكل المخطط العام المتبع لتوليد حالات اختبار باستخدام خوارزميات البحث الأمثلية، بدايةً سنقوم ببناء نموذج يمثل التطبيق، يعتبر هذا النموذج فضاء البحث لخوارزمية البحث المقترحة، ليتم البحث ضمنه عن حالات اختبار أمثلية، قمنا باقتراح وتحقيق الأداة BeeDroid لتوليد حالات الاختبار لتطبيقات Android اعتماداً على الخطوات الموضحة في الشكل (5-11).



الشكل (5-11) المخطط العام لتوليد حالات اختبار تطبيقات Android باستخدام خوارزميات البحث الأمثلية

5-4 الدراسة المرجعية

كما وضحنا في الشكل (5-11) لتوليد حالات الاختبار يوجد مكونان أساسيان للأداة:

1. بداية يجب تحليل تطبيق Android لاشتقاق نموذج شامل يوصف التطبيق ومكوناته والانتقالات فيما بينها.
2. استخدام خوارزمية بحث أمثلي (استخدمنا خوارزمية النحل الصنعية) لتوليد حالات الاختبار انطلاقاً من النموذج الذي تم اشتقاقه للتطبيق.

لذلك الدراسة المرجعية تمت على مرحلتين كما سنوضح فيما يلي:

5-4-1 الدراسات المتعلقة بعملية تحليل تطبيقات Android

قُدِمّت دراسات عدة لتحليل تطبيقات Android منها بهدف اختبارها كاختبار الصندوق الأبيض ومنها بهدف تحليلها لمراقبة السلوك الخبيث واكتشاف محاولة الحصول على بيانات هامة وحساسة من المستخدم ومنها دراسات قُدِمّت بهدف تحليل هذه التطبيقات لاستخراج تمثيل ونموذج يوصّفها، سنستعرض فيما يلي أهم الدراسات التي اعتمدنا عليها في بحثنا وكانت المنطلق للوصول لآلية تمكننا من بناء بيان يمثل تدفق الأحداث، قدمت الدراسة [40] حزمة برمجية Dexpler كان الهدف منها تحويل ترميز Dalvik لـ Jimple وبنيت هذه الحزمة فوق Soot، سنوضح هذه المفاهيم كونه تم الاعتماد عليها في أغلب الدراسات، تحليل الكود المصدري المكتوب بلغة Java أو Dalvik مباشرة صعب ومعقد كون هذه اللغات صممت بهدف تحسين الأداء أثناء التنفيذ وليس للتحليل الستاتيكي، تحوي برامج Java على أكثر من 200 رمز تشغيلي إذا تم أخذ كل هذه المعاملات بعين الاعتبار عند التحليل ورسم مخططات التدفق سيزيد من صعوبة وتعقيد العملية، تعتبر Jimple مستوى أعلى من التجريد من الـ Java Bytecode التي تدعم تركيبات معقدة وتحولها Jimple لتسلسل تعليمات أبسط، بالتالي يمكن أن نعرف Jimple أنه تمثيل وسيط لبرامج Java مصمم ليكون أسهل في التحسين من خلال اعتماده على الترميز ثلاثي العناوين three-address المستخدم كـ لغة وسيطة داخل المترجمات المحسنة بهدف تحسين التعليمات البرمجية، تدعم Jimple الترميز ثلاثي العناوين وعمليات تبسط تحليل التدفق للتطبيقات، في تحليل البرامج بشكل ستاتيكي تعتبر Soot إطار عمل لمعالجة وتحسين الترميز لبرامج Java، [41,42] تم إنشاء Soot كأداة لتحليل وتحويل برامج Java يمكن استخدامها لتحليل الكود للتأكد من بعض الخصائص أو لضمان صحة البرامج، دخل الأداة Soot برامج Java أو Java Bytecode، مهما كان الدخل يتم تحويله لترميز Jimple، بالتالي يمكن عرض أي إجرائية كرسم بياني لتعليمات Jimple مرتبطة بقائمة متغيرات محلية، تحوي إصدارات Soot الحالية على تحليلات تفصيلية للبرامج مثل تحليل بيان الاستدعاءات وتحليل التدفق الحساس للسياق، لكن Soot لا تدعم تحويل ترميز Dalvik (المستخدم في تطبيقات أندرويد) لـ Jimple تم تطوير الحزمة Dexpler ودُمِجت مع Soot بنفس الآلية والبنية المتبعة لتحليل برامج Java بالتالي فتحت الطريق لتحليل تطبيقات Android لكن يوجد بعض الحالات والمشاكل لم تتم معالجتها، في الدراسة [43] تم تقديم AspectDroid لتحليل تطبيقات Android لاكتشاف الأنشطة غير المشروعة من خلال الدراسة تم تحليل الكود والبحث عن استدعاءات

API معينة وحساسة تعتبر مصدراً لنشاط غير مشروع مثل محاولة الوصول لجهات اتصال المستخدم وإرسال SMS وكتحسين لهذه الحزمة تم اقتراح توسيعها لاحتواء واكتشاف سياسات التلاعب بشكل أكبر، في [44] كان هدف الدراسة تحليل تطبيقات Android بشكل ستاتيكي لاكتشاف تسريب البيانات من خلال تقديم الأداة FlowDroid التي تمكِّن المستخدم من تقييم التطبيق قبل استخدامه وبالتالي حماية بياناته الشخصية، تكتشف FlowDroid السلوك غير السليم للتطبيقات من خلال تحديد الاستدعاءات التي تمثل مصدراً يمكن أن يسرِّب البيانات من جهاز المستخدم Source وتوصيف الاستدعاءات التي تعتبر الوسيلة لنقل هذه البيانات وتسريبها Sink، لتحليل تطبيقات Android اعتمدت FlowDroid بشكل أساسي على Soot وتمثيلها للكود باستخدام ترميزات Jimple كما استخدمت الأداة Dexpler المسؤولة عن تحويل ترميزات Dalvik لـ Jimple كما سبق ذكره، لكن تم التركيز فقط على عناصر أساسية تعتبر مصدر لتسريب البيانات مثل مربعات النص ومربعات كلمات المرور على واجهة المستخدم لكن لا تزال تستبعد بعض العناصر غير القابلة للتغيير مثل الأزرار حيث لا تملك أية معلومات من المستخدم، كما أنّ FlowDroid لا تركز على التمييز بين الإعدادات مثلاً حجم أو طريقة دوران الشاشة أو اللغة المستخدمة والتي تأخذ قيمتها أثناء مرحلة التنفيذ فهي تعتمد على الافتراضي كون الإعدادات لا تلعب دوراً في عملية تسريب البيانات أيضاً FlowDroid تعمل على التحليل الستاتيكي لمكونة واحدة فقط intra-component فهي لا تمثل الاتصال بين المكونات والانتقال من خلال الـ intent، في الدراسة [45] تم تحليل تطبيقات أندرويد لاستخراج مخطط التدفق للطرائق وتسلسل استدعائها من خلال بيان يمثل هذه الاستدعاءات callback control-flow graph (CCFG)، بُنيَت على أساس Soot، تم التركيز على تحليل طرائق بدء وانتهاء تنفيذ الأنشطة فقط كتحسين للتحليل تم اقتراح تضمين كل طرائق دورة حياة باقي مكونات التطبيق، كما لم يتم تمثيل الأحداث الخارجية في مخطط التدفق مثل دوران الشاشة أو قفل الشاشة أو زر الرجوع خلال عمل التطبيق حيث تمثيل هذه الأحداث سيضيف حواف جديدة للبيان، كما تم اقتراح استخدام نتائج التحليل هذه لبناء بيان يبين تفاعل مكونات التطبيق مع بعضها مما يساعد على التحليل الستاتيكي لعملها والذي يمكن استخدامه في مجال توليد حالات اختبار لهذه التطبيقات.

5-4-2 الدراسات المتعلقة بعملية توليد حالات اختبار لتطبيقات Android

الهدف الأساسي لكل اختبار هو التحقق من أن التطبيق يعمل بشكل صحيح ولا يظهر سلوكيات غير متوقعة أثناء التنفيذ، تقنيات الاختبار تنفّذ التطبيق أكبر قدر ممكن من خلال توليد المدخلات ومراقبة سلوك التطبيق، لاختبار تطبيقات Android يجب استخراج خصائص التطبيق مثل الأنشطة والمكونات الأخرى، بالإضافة لتحليل المكونين الأساسيين من كل تطبيق Android لاختبار التطبيق بمنهجية اختبار الصندوق الرمادي وهما ملف AndroidManifest.xml الذي يحوي تعريف السماحيات المطلوبة ومكونات التطبيق، بالإضافة لملف dex. وفيه صفوف التطبيق مترجمة لتنسيق Dalvik، قُدِمّت عدة دراسات لتوليد حالات اختبار لتطبيقات Android فيما يلي سنذكر أهم هذه الدراسات والطريقة التي اعتمدت عليها كل دراسة لتوليد حالات الاختبار.

الاختبار العشوائي: يتم استخدام منهجية الصندوق الأسود لتوليد تسلسل أحداث بشكل عشوائي من أحداث النظام وأحداث المستخدم وإرسالها للتطبيق ليتم تنفيذها، في هذه الطريقة يتم توليد الأحداث بشكل مستقل عن آلية عمل ووظائف التطبيق لكن تحقيقها سهل، قدمت منصة Android الأداة Monkey [46] للاختبار العشوائي قادرة على توليد أحداث المستخدم مثل النقرات أو أحداث النظام مثل لقطة الشاشة بشكل عشوائي وقابلة للتكرار بسبب اعتماد هذه الأداة من Google كجزء من أدوات تطوير Android يتم استخدامها بشكل واسع لاختبارات الصندوق الأسود من قبل مختبري ومطوري تطبيقات Android، قدمت الراسة [47] الأداة Dynodroid للاختبار العشوائي لكن بشكل فعال أكثر من Monkey لاعتماد هذه الأداة على استكشاف واجهات المستخدم وتوليد أحداث عشوائية أكثر كفاءة عن طريق استخدام الأداة Hierarchy Viewer وهي أداة تم إضافتها لنظام Android لتحديد التسلسل الهرمي لشاشة التطبيق وواجهات المستخدم.

الاختبار المعتمد على النموذج: الفكرة الأساسية من هذا النوع من الاختبار هو توليد مجموعة من المدخلات اعتماداً على نموذج يوصِّف التطبيق، يقلل التكرار ويحسن كفاءة حالات الاختبار المولدة مقارنة بالاختبار العشوائي، تعتمد فعالية حالات الاختبار على النموذج المصمم والذي يصور سلوك التطبيق، مجموعة من الدراسات قدمَّت اعتماداً على النموذج، قدمت الدراسة [48] الأداة MobiGUITAR التي حللت واجهات المستخدم الرسومية لاستخراج مخطط تدفق الأحداث لتوصيف التطبيق هذا الرسم البياني عادة ما يولد تسلسل من الأحداث غير المنطقية مما يقلل فعالية حالات الاختبار المولدة، قدمت [49] الأداة ORBIT لتوليد حالات الاختبار باستخدام منهجية اختبار الصندوق الرمادي عن طريق توليد نموذج يوصف سلوك واجهة المستخدم الرسومية للتطبيق عن طريق تحليل شيفرة Java لتطبيق Android لاستخراج الأحداث المرتبطة بمكونات الواجهة، ثم تطبيق خوارزمية البحث بالعمق مع مراعاة أخذ الحدث Back لتوليد حالات

الاختبار، تم تمثيل التطبيق باستخدام نموذج Finite State Machine لكن الاستكشاف فقط لواجهات المستخدم بالتالي الأداء محدود، قدمت الدراسة [50] الأداة DroidBot لتوليد الدخل للاختبار من خلال التفاعل مع تطبيق Android لتوليد نموذج مخطط الحالات من خلال حفظ الحالة الحالية ومراقبة تغيرات الحالة بعد إرسال دخل، إذا تغيرت حالة التطبيق يتم إضافة الحالة للمخطط كحافة وعقدة جديدة، تستخدم SwiftHand [51] تقنيات التعلم الآلي من أجل تعلم نماذج التطبيقات أوتوماتيكياً، الهدف منها هو تحسين استراتيجية الاستكشاف للواجهات، في [52] تم اقتراح الأداة AppTag اعتمدت نموذجاً لتوصيف الأنشطة للتطبيق والانتقال بينهم ديناميكياً أثناء التنفيذ، تم اعتماد بعض عناصر الواجهات والأحداث المرتبطة بهم لتمثيل التطبيق بالإضافة لزر الخلف وهنا تم استخدام المكدسات لتخزين حالة النشاط السابقة، مهما تعددت الطرق لبناء النموذج تبقى المهمة الأساسية هي توليد حالات الاختبار اعتماداً على النموذج، استخدمت طرقاً تقليدية للمرور على عقد البيان مثل البحث بالعمق أولاً أو بالعرض أولاً، أو توليد حالات اختبار عشوائية اعتماداً على النماذج المولدة.

الاختبار المعتمد على البحث: العديد من الدراسات اتجهت لاستخدام الاختبار المعتمد على البحث، يعتمد على استكشاف مسارات التطبيق لاستخراج حالات الاختبار باستخدام الخوارزميات التطويرية لتوجيه البحث لاستكشاف الحالات التي يصعب الوصول إليها باستخدام الأسلوب العشوائي، أول أداة استخدمت هذا النهج لاختبار تطبيقات Android هي EvoDroid [53] حيث استخدمت الخوارزمية الجينية لتوليد تغطية عالية لاختبار واجهات المستخدم الرسومية عن طريق استخراج نموذجين من كود التطبيق، نموذج يعتمد على تحليل ملف AndroidManifest.xml والآخر اعتماداً على تحليل الكود باستخدام الأداة MoDisco لبناء بيان للاستدعاءات لواجهات المستخدم الرسومية لكن في هذه الأداة تم تثميل فقط واجهات التطبيق كتطوير لها تم اقتراح بناء نموذج يوصف التطبيق بشكل أفضل، قدمت الدراسة [54] الأداة AGRippen والتي استخدمت الخوارزمية الجينية للبحث عن حالات الاختبار بعد استخراج نموذج يوصف عناصر واجهات المستخدم والأحداث المرتبطة معها من خلال تحليل كود التطبيق، تم اختبار هذه الأداة على خمسة تطبيقات Android مفتوحة المصدر ولم تتم مقارنتها مع أدوات أخرى.

5-5 الآلية المقترحة لبناء الأداة BeeDroid لتوليد حالات اختبار لتطبيقات Android

لبناء الأداة تم العمل على مرحلتين كما ذكرنا سابقاً، بداية تم تطوير الأداة EGator لتوليد نموذج يمثل التطبيق ومكوناته الأربعة التي سبق ذكرها بالإضافة لاستخراج السماحيات المطلوبة للتطبيق لنصل لبيان يوصّف ويمثل التطبيق، ثم تمرير هذا البيان ليكون دخلاً لخوارزمية النحل الصنعية التي تم ملاءمتها لتوليد حالات اختبار أمثلية لتطبيقات Android، سنبين فيما يلي تفاصيل كل مرحلة:

5-5-1 الآلية المتبعة لبناء نموذج يمثل تطبيقات Android

تم اقتراح الأداة EGator بالاعتماد على الأداة Gator التي قدمتها [45] كمصدر مفتوح، الأداة Gator فقط تمثل الأنشطة ودورة حياتها، حيث تم تمثيل التطبيق من خلال بيان عقده تمثل النشاطات في التطبيق، كل عقدة لها معلومات تمثل الـ window وعدد المسارات الداخلة أو الخارجة منها، كل مسار (حافة Edge) يمثل الانتقال من عقدة لعقدة أخرى أو لنفس العقدة عند حدوث حدث معين، كل مسار يمثل معلومات عن العقدة المصدر له والهدف، ومعلومات عن الحدث الذي تسبب بالانتقال من حالة لأخرى والتي تمثل اسم الحدث والإجرائية المعالِجة له، بالإضافة لمعلومات عن استدعاءات النظام (مثل إنشاء نشاط)، يمكن الانتقال بين النشاطات من خلال أغراض Intent تم تمثيل الانتقالات ضمن البيان.

ولتمثيل تطبيقات Android بشكل كامل يجب تمثيل وتحليل التطبيق لاستخراج مكوناته الأخرى وآلية الانتقال بين هذه المكونات، بالإضافة للسماحيات التي يطلبها التطبيق، تم إضافة وظائف للأداة Gator لتقوم بتحليل أفضل وأكمل للتطبيق، سنستعرض فيما يلي أهم خصائص واستدعاءات كل مكون والتي يجب تضمينها عند عملية بناء النموذج.

كما تم تمثيل الأحداث الخارجية من النظام واكتفينا بحالة الضغط على زر back كونه سيمثل انتقال من مكون لآخر بالتالي سيسبب تغيّر في حالة التطبيق.

5-5-1-1 الخدمات Services

لتمثيل الخدمة في بيان التدفق لتطبيقات Android لابد بداية من فهم آلية عملها ودورة حياتها وأهم الاستدعاءات التي تشغِّل الخدمة أو توقفها.

للخدمات ثلاثة أنواع[37] :

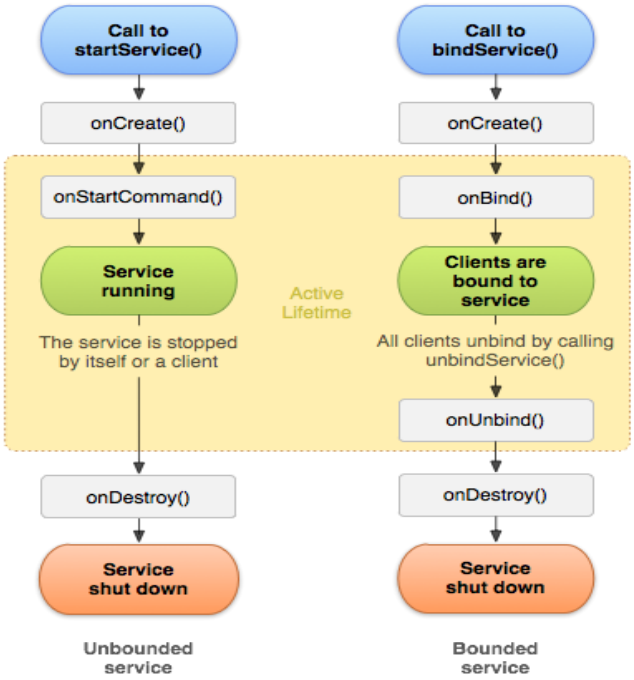
1. Foreground: تنجز عمليات تكون مرئية ويلاحظها المستخدم مثل تطبيق لتشغيل صوت سيستخدم خدمة من هذا النوع لتشغيل الصوت، ستبقى الخدمة تعمل حتى عندما المستخدم لا يتواصل مع التطبيق.
2. Background: تنجز عمليات لا يلاحظها المستخدم مباشرة، مثل تطبيقات المنبه.
3. Bound: تستخدم عندما مكونات التطبيق بحاجة للتواصل مع الخدمة لإرسال طلبات أو تلقي النتائج، مثل خدمة للاتصال بقواعد البيانات.

أهم الاستدعاءات Callbacks التي يستدعيها نظام التشغيل لتشغيل الخدمة والاتصال معها:

1. ()onStartCommand: من خلال إعادة تحقيق هذه الطريقة ضمنها يُكتب الكود الذي سينفذ في الخلفية عند تشغيل الخدمة، سيستدعي نظام أندرويد هذه الطريقة عندما مكون آخر يريد بدء الخدمة من خلال التعليمة ()startService، عندها الخدمة ستعمل وتنفذ الأمر المطلوب.
2. ()onBind: تستخدم مع النوع الثالث من الخدمات، عندما التطبيق يريد الاتصال مع الخدمة، نظام أندرويد سيستدعي هذه الطريقة عندما مكون آخر يستدعي ()bindService.
3. ()onCreate: سوف يستدعيها النظام لتهيئة الخدمة قبل استدعاء ()onStartCommand أو ()onBind.
4. ()onDestroy: يستدعي نظام أندرويد هذه الطريقة عندما يتم الانتهاء من استخدام الخدمة لتحرير الموارد التي تم حجزها.

يمثل الشكل (5-12) دورة حياة الخدمة في تطبيقات Android.

تعتبر الخدمات جزءاً هاماً من التطبيق، يجب تمثيل الخدمات ضمن البيان من خلال عقد تمثل الخدمة وحالاتها والأحداث التي تسببت بالانتقال لهذه الخدمة، لذلك تم تعديل الأداة Gator وإضافة وظائف لها لتمثيل كل خدمة من خلال عقدة ضمن البيان يتم الانتقال إليها عند أحد الاستدعاءات التي تم ذكرها، العقدة التي تمثل الخدمة يمكن أن تنتقل لنفسها عند الانتقال من حالة create إلى حالة start، الانتقال من نشاط أو مكون آخر للتطبيق لعقدة خدمة يكون عند ()onStartCommand أو ()onBind،



الشكل (5-12) دورة حياة الخدمة

5-5-1-2 مستقبل البث Broadcast Receiver

Broadcasts هي رسائل يرسلها نظام Android وتطبيقات Android عند وقوع أحداث قد تؤثر على وظائف تطبيقات أخرى، مثلاً نظام Android يرسل حدثاً عند توصيل سماعات الرأس أو فصلها، كما يمكن لتطبيق Android بث أحداث لتطبيقات أخرى أيضاً، مثلاً عند اكتمال تحميل بيانات جديدة، تمثل عمليات البث مكونات مراسلة للتواصل عبر التطبيقات عند حدوث أحداث مهمة، يمكن للتطبيق التسجيل لتلقي بث معين، عند إرسال بث يقوم النظام تلقائياً بتوجيه عمليات البث إلى التطبيقات التي سجلت (تسمى مستقبلات البث) لتلقي هذا النوع من البث [37].

رسالة البث يتم تغليفها ضمن غرض Intent، من خلال خاصية action للغرض يتم تحديد الحدث الذي حصل مثلاً android.intent.action.AIRPLANE_MODE، التطبيق حتى يرسل بث يمرر الغرض Intent المغلف له بإحدى الطرق:

1. في الحالة العادية يمرر الغرض للإجرائية sendBroadcst() حيث يتم إرسال البث لكل التطبيقات المسجلة لاستقباله بأي ترتيب.
2. تمرير الغرض للإجرائية sendOrderedBroadcast() لإرسال البث لمستقبل واحد في الوقت، يستخدم هذا النوع عندما يكون الترتيب ضرورياً بين المستقبلات وبيانات سيتم تبادلها بين المستقبلات أو الإرسال سيتم حسب الأولويات.
3. لإرسال البث محلياً ضمن التطبيق نفسه نستخدم الإجرائية LocalBroadcastManager.sendBroadcast()

الصف المستقبل للبث سيحقق الطريقة onReceive() التي سيتم استدعاؤها عندما المستقبل يستقبل بث معين،

حتى يستقبل تطبيق ما بث يمكن أن يسجل بإحدى طريقتين:

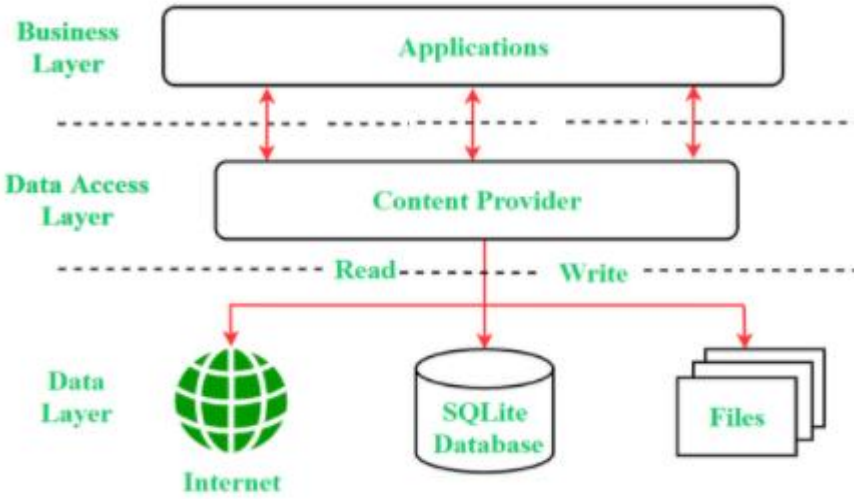
- بشكل ستاتيكي ضمن ملف AndroidManifest.xml مع تحديد البث الذي سيستلمه.
- أو ضمن الكود من خلال الإجرائية registerReceiver() التي نحدد لها الغرض التي سيستلم البث ونوع البث الذي سيستلمه.

يجب تمثيل مستقبلات البث لذلك قمنا بتعديل الأداة Gator وإضافة وظائف لها لتمثيل مستقبلات البث ضمن البيان من خلال استكشاف مستقبلات البث وإنشاء عقد في البيان تمثل هذه المستقبلات، تمت معالجة حالة إرسال البث محلياً، لأن البيان الناتج يمثل الانتقالات بين مكونات التطبيق الحالي فقط، إرسال البث محلياً سيسبب انتقال من مكون لتطبيق Android إلى عقدة أخرى تمثل مستقبل البث ليتم تنفيذ

()onReceive، كما تم أخذ باقي المستقبلات بعين الاعتبار وإعطاء تقرير عن المستقبلات المسجلة لبث ونوع البث لأنّ هذه المعلومات تفيد في حالة تحليل التطبيق.

5-5-1-3 مزوّد المحتوى Content Provider

مكون هام في تطبيقات Android، حيث يساعد التطبيق على الوصول لبياناته المخزنة أو بيانات تطبيقات أخرى، فهو يوفر وسيلة لمشاركة البيانات بين التطبيقات بطريقة آمنة، يسمح نظام Android لمزود المحتوى بتخزين البيانات بعدة طرق، يمكن للمستخدمين إدارة تخزين بيانات التطبيق مثل الصور أو مقاطع الصوت أو جهات الاتصال عن طريق تخزينها في ملفات أو قواعد بيانات أو حتى على شبكة، يبين الشكل (5-13) دور مزود المحتوى في إدارة التحكم بالوصول للبيانات،



الشكل (5-13) إدارة الوصول للبيانات عن طريق مزود المحتوى

يعتبر Content URI معرّفاً للبيانات، يتضمن اسم المزود واسم يشير إلى جدول أو ملف معين، كل طريقة للوصول للبيانات في مزود المحتوى يجب أن نمرر لها URI لتحديد الجدول أو الملف المراد الوصول له، تستخدم مكونات تطبيق Android مثل الأنشطة غرض من نوع CursorLoader لإرسال طلبيات استعلام لغرض من نوع ContentResolver ليرسل هذه الطلبات (استعلام، حشر، حذف أو تعديل) بدوره إلى غرض ContentProvider، ليقوم بمعالجة الطلب وإعادة النتيجة، الصف ContentProvider فيه ست إجرائيات تسمح بإنشاء مزود محتوى والوصول للبيانات، يجب عند تحقيق هذا الصف إعادة كتابة هذه الإجرائيات والتي هي كما يلي:

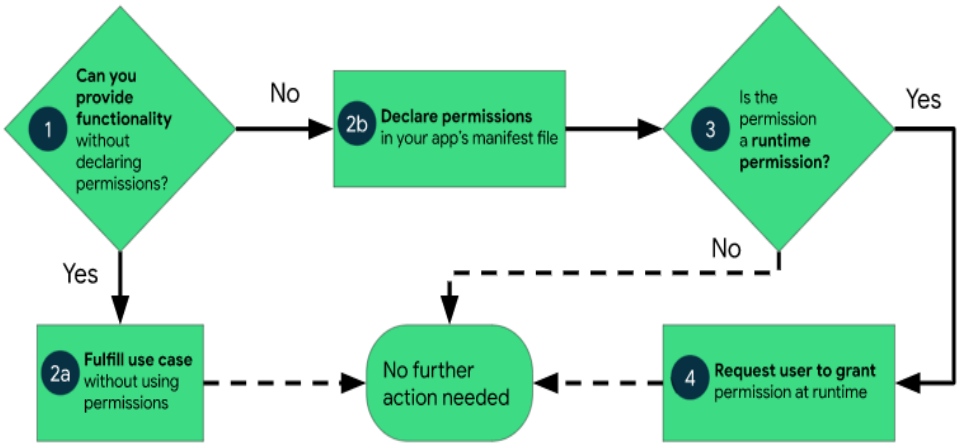
1. ()onCreate: عند إنشاء مزود المحتوى فإنّ نظام Android سيستدعي هذه الإجرائية مباشرة لتهيئة المزود.

2. ()query: إجرائية لاسترجاع البيانات من الهدف المطلوب حيث يعيد مؤشراً للبيانات التي تم جلبها.
3. ()insert: لإضافة بيانات جديدة.
4. ()update: لتعديل بيانات موجودة.
5. ()delete: لحذف بيانات.
6. ()getType: تعيد نوع MIMI للـ URI المعطى، MIMI type هو معيار يشير لطبيعة وتنسيق المستند أو مجموعة من البايتات.

مزود المحتوى جزء هام للتطبيقات لا بد من تمثيله وتمثيل الاستدعاءات الخاصة به عند تمثيل أي تطبيق Android، قمنا بإضافة وظائف للأداة Gator لمعالجة التطبيق واستخراج الصفوف التي تمثل مزودات للمحتوى وتمثيلها ضمن البيان الناتج للتطبيق كعقد يتم الانتقال إليها عند إنشاؤها أو طلب خدمة منها من خلال أغراض الـ Inent، حيث تم استكشاف الصفوف من نوع CursorLoader التي تتعامل مع أغراض من نوع ContentResolver التي بدورها توجه الطلب لمزود لمحتوى ليتم معالجته.

5-5-1-4 تحليل ملف AndroidManifest.xml لاستخراج السماحيات المطلوبة

كما ذكرنا مسبقاً تساعد سماحيات التطبيق في دعم خصوصية المستخدم من خلال حماية الوصول إلى بيانات المستخدم المقيدة مثل جهات الاتصال أو الوصول إلى إجرائيات معينة مقيدة مثل تسجيل الصوت، إذا كان التطبيق يريد الوصول لبيانات أو إجراءات مقيدة يجب التصريح عن السماحيات المطلوبة، تُمنح السماحيات أثناء تثبيت التطبيق وبعضها أثناء عمل التطبيق، يبيّن الشكل (5-14) سير العمل لاستخدام السماحيات،



الشكل (5-14) آلية عمل استخدام السماحيات

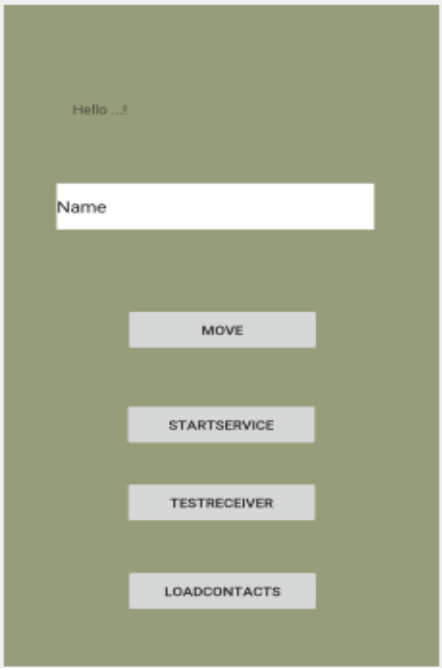
يجب التصريح عن أيّ سماحية ضمن ملف AndroidManifest.xml كما يلي:

```
<manifest ...>
    <uses-permission android:name="android.permission.CAMERA"/>
    <application ...>
        ...
    </application>
</manifest>
```

يجب معرفة السماحيات المطلوبة لأي تطبيق عند تحليله سواء كان الهدف من تحليل التطبيق اختباره كون هذه السماحيات ستؤثر على طريقة التنفيذ أو إذا كان الهدف من تحليل التطبيق اكتشافه هل هو برمجية خبيثة أم لا حيث يتم من خلال السماحيات المطلوبة والبيانات التي يحاول الوصول لها التطبيق معرفة ماهيّة التطبيق، قمنا بمعالجة ملف AndroidManifest.xml لاستخراج السماحيات المصرح عنها.

5-5-1-5 التطبيق العملي

بعد الإضافات والتعديلات التي قمنا بها لتحليل تطبيقات Android وأخذ المكونات الأساسية بعين الاعتبار عند بناء بيان الأحداث الخاص بالتطبيق قمنا بتجريب كيفية التحليل والنتائج المعادة على التطبيق example.apk الممثل في الشكل (5-15) باستخدام الأداة EGator، التطبيق مكون من نشاطين الأول MainActivity هو الظاهر في الشكل بعد إدخال اسم المستخدم والنقر على زر MOVE سينتقل للنشاط الثاني SecondActivity الذي مهمته فقط عرض رسالة ترحيبية حسب الاسم المدخل، الزر STARTSERVICE مهمته البدء بخدمة عند النقر عليه، والزر TESTRECEIVER عند النقر عليه سيتم إرسال بث لمستقبل بث محلي ليبدأ بتنفيذ بعض التعليمات مع العلم أنه ضمن ملف AndroidManifest.xml تم تسجيل مستقبل بث ثاني يستقبل البث من نظام Android عند تفعيل وضع الطيران، الزر LOADCONTACTS مهمته الوصول لجهات الاتصال وقراءتها وطباعة عددها (هنا اعتمدنا على مكون مزود المحتوى للوصول لهذه البيانات) بالإضافة لطلب سماحية الوصول لجهات اتصال المستخدم لتنفيذ هذا الإجراء، التطبيق example.apk هو الدخل للأداة EGator ليتم تحليله واستخراج مخطط يمثل بيان الأحداث والانتقال بين المكونات استجابة للأحداث أو لاستدعاءات نظام Android الخاصة بكل مكون كما تم ذكره فيما سبق.



الشكل (5-15) تطبيق example

تبين نتيجة التحليل أن البيان مكون من 6 عقد (النشاطان والخدمة ومستقبل البث المحلي ومزود المحتوى بالإضافة لعقدة تمثل بداية البيان)، يمثل الشكل (5-16) عقد التطبيق التي تم استخراجها، حواف البيان وهي تمثل الأحداث المسببة للانتقالات بين هذه المكونات، يعرض الشكل (5-17) جزءاً من هذه الانتقالات، الشكل (a) يمثل الانتقال عند الضغط على الزر STARTSERVICE، العقدة المصدر هي النشاط MainActivity والهدف هو العقدة التي تمثل الخدمة TestService عند الحدث الممثل بالإجرائية TestStartService حيث تبدأ الإجرائية onCreate الخاصة ببدء الخدمة، الشكل (b) يمثل الانتقال من النشاط MainActivity لبدء مستقبل البث المحلي LocalReciever، الشكل (c) يمثل الانتقال بين النشاطين MainActivity و SecondActivity عند الضغط على الزر MOVE.

```
[Node]: ContentProvider[com.example.exampleapp.MyContentProvider]
[Node]: Activity[com.example.exampleapp.MainActivity]
[Node]: Service[com.example.exampleapp.TestService]
[Node]: Activity[com.example.exampleapp.SecondActivity]
[Node]: BroadcastReceiver[com.example.exampleapp.LocalReciever]
[Node]: LAUNCHER_NODE[]3008
```

الشكل (5-16) عقد التطبيق example

```
Edge Current Edge ID: 840794863
  Source Window: Activity[com.example.exampleapp.MainActivity]
  Target Window: Service[com.example.exampleapp.TestService]
  EventType: implicit_service_lifecycle_event
  Event Callbacks:
      <com.example.exampleapp.MainActivity: void TestStartService(android.view.View)>
  Lifecycle Callbacks:
      <android.app.Service: void onCreate()>
```

(a) الانتقال عند الضغط على الزر STARTSERVICE

```
Edge Current Edge ID: 824715394
  Source Window: Activity[com.example.exampleapp.MainActivity]
  Target Window: BroadcastReceiver[com.example.exampleapp.LocalReciever]
  EventType: send_broadcast
  Event Callbacks:
      <com.example.exampleapp.MainActivity: void TestReciver(android.view.View)>
  Lifecycle Callbacks:
      <android.content.BroadcastReceiver: void onReceive(android.content.Context,android.content.Intent)>
```

(b) يمثل الانتقال من النشاط MainActivity لبدء مستقبل البث المحلي LocalReciever

```
Edge Current Edge ID: 1108192494
  Source Window: Activity[com.example.exampleapp.MainActivity]
  Target Window: Activity[com.example.exampleapp.SecondActivity]
  EventType: click
  Event Callbacks:
      <com.example.exampleapp.MainActivity: void move(android.view.View)>
  Lifecycle Callbacks:
      <com.example.exampleapp.SecondActivity: void onCreate(android.os.Bundle)>
```

(c) الانتقال بين النشاطين MainActivity و SecondActivity عند الضغط على الزر MOVE

الشكل (5-17) بعض حواف البيان الممثل للتطبيق example

كما ذكرنا مسبقاً قمنا أيضاً بتحليل ملف الـ AndroidManifest.xml لاستخراج السماحيات المطلوبة للنظام وإعطاء تقرير بها، سنعتمد على هذه السماحيات في اختبار الطفرات للتأكد من فعالية حالات الاختبار المولدة، تم تمثيل مستقبلات البث المحلية فقط في البيان الناتج، أما مستقبلات البث للنظام أو لتطبيقات أخرى فقد تم إعطاء تقرير بها أيضاً سنستخدمها في اختبار الطفرات في الفصل السادس، يبين الشكل (5-18) السماحيات ومستقبلات البث الغير المحلية.

```
[BroadcastReceiver]className com.example.exampleapp.MyReceiver action: android.intent.action.AIRPLANE_MODE
```

```
[permission]uses-permission android.permission.READ_CONTACTS
```

الشكل (5-18) السماحيات ومستقبلات البث الغير محلية المطلوبة للتطبيق example

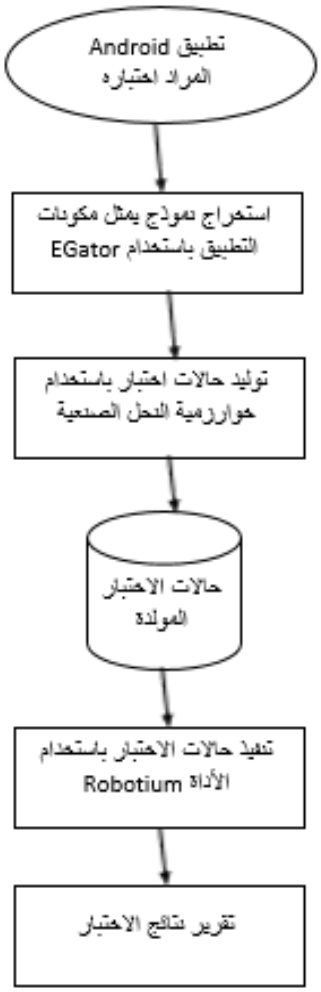
قدمنا من خلال الأداة EGator طريقة لبناء بيان يمثل مكونات تطبيق Android والانتقالات بين هذه المكونات من خلال الأحداث التي يطلقها المستخدم بتفاعله مع عناصر الواجهة أو من خلال استدعاءات النظام التي تسبب تغير في حالة المكون، يعتبر هذا البيان دخلاً لخوارزمية البحث لتوليد حالات الاختبار.

قدمت الدراسات السابقة تحليلاً محدوداً لتطبيقات Android من خلال التركيز فقط على واجهات المستخدم الرسومية الممثلة بالأنشطة [45] أو التركيز فقط على بعض الاستدعاءات التي تسبب تسريب في البيانات [44] من خلال تحليل كل مكونة من مكونات التطبيق بشكل مستقل دون إظهار التفاعل بين المكونات أو آلية الانتقال بينها، قدمنا من خلال الأداة EGator تحليلاً شاملاً للمكونات الأربعة الأساسية لتطبيقات Android من خلال تحليل الاستدعاءات المتعلقة بكل مكون ثم إظهار التفاعل بين هذه المكونات وآلية الانتقال بينها عند استدعاءات معينة للوصول لبيان يمثل التطبيق ويوضح المكونات وآلية الانتقال بينها كما وضحنا في التطبيق العملي البيان الذي تم بناؤه والانتقالات التي وضحناها بين عقد البيان والتي أعطت تمثيلاً أدق للتطبيق وأشمل من الدراسات السابقة التي لم تأخذ بعين الاعتبار كل حالات مكونات التطبيق ولم تقدم تحليلاً وتمثيلاً كافياً لتطبيقات Android، وهذا بدوره سينعكس إيجابياً عند استخدام هذا البيان وتوظيفه فيما بعد في عملية توليد حالات الاختبار.

5-5-2 استخدام خوارزمية النحل الصنعية لتوليد حالات اختبار لتطبيقات Android

اعتماداً على هذا المخطط في الشكل (5-11) تم اقتراح الأداة BeeDroid، العمل في هذه الأداة مكون من جزأين أساسيين كما بيّنا مسبقاً، الأول يتمثل في اشتقاق نموذج (بيان) يمثل تطبيق Android ومكوناته وآلية الانتقال بين هذه المكونات، تم الاعتماد على الأداة EGator لبناء بيان يمثل مكونات تطبيق Android والانتقالات بين هذه المكونات كما ذكرنا مسبقاً، أما بالنسبة للجزء الثاني من الأداة يمثل خوارزمية البحث المراد تطبيقها حيث تم اعتماد خوارزمية النحل الصنعية.

اعتماداً على ما تم دراسته فإنّ مخطط عمل الأداة BeeDroid المقترحة ممثل في الشكل (5-19)، الهدف من الخوارزمية المقترحة توليد حالات اختبار تشمل تغطية أكبر للكود لاكتشاف الحالات التي تسبب فشل التطبيق أو حدوث أي استثناء، وتجريب فعالية الأداة وقدرتها على اكتشاف الأخطاء في التطبيق إن وجدت.



الشكل (5-19) تسلسل تنفيذ الأداة BeeDroid المقترحة

دخل الأداة BeeDroid هو ملف apk. يمثل التطبيق المراد اختباره، يمرر الدخل للأداة EGator يتم اشتقاق بيان يمثل التطبيق ومكوناته، ثم يتم تمرير هذا البيان لخوارزمية النحل الصنعية لتبدأ خطوات الخوارزمية.

ذكرنا في الفصل الرابع المفاهيم المتعلقة بعملية اختيار المسارات الأساسية للاختبار، اعتماداً على هذه المفاهيم وانطلاقاً من البيان الذي يمثل التطبيق الذي يعتبر فضاء البحث البدائي لخوارزمية النحل الصنعية بدأت خطوات خوارزمية النحل بالعمل كما وضحنا الخطوات في الفصل الرابع.

بدايةً من خلال توليد فضاء البحث البدائي العشوائي الذي يمثل حالات الاختبار البدائية والتي تمثّل كتسلسل من أحداث المستخدم والنظام (تم اشتقاق تسلسل الأحداث من حواف البيان الذي يمثل الانتقالات بين المكونات عند حدث ما) وعددها يساوي قيمة CC يُحسب اعتماداً على البيان المولد وطول هذه الحالات عشوائي، لتبدأ بعدها النحلات العاملات التعديل على حالات الاختبار لمحاولة إيجاد حلول أفضل وتحسين الحلول الحالية وتقدم هذه الحلول (حالات الاختبار) للنحل المشاهد ليقيمها، التعديل يتم من خلال استبدال حدث بآخر في تسلسل حالات الاختبار للوصول لتغطية أعظمية للكود، إذا لم يتم تحسين أحد الحلول يتم استبعاده والتخلي عنه ليبدأ النحل الكشاف البحث عن حل أفضل.

6-5 خاتمة

قدمنا في هذا الفصل دراسة شاملة عن تطبيقات Android وخصائصها ومكوناتها، تم اقتراح الأداة BeeDroid لتوليد حالات الاختبار، تبدأ من خلال تحليل التطبيق باستخدام الأداة EGator لاشتقاق نموذج بيان يمثل التطبيق ثم تطبيق خوارزمية النحل الصنعية للوصول لحالات اختبار أمثلية.

سنقوم في الفصل السادس بتقييم الأداة BeeDroid من خلال استخدام معيارين هما مقدار تغطية الكود ثم مقارنة التغطية التي تغطيها الأداة مع أدوات مشابهة، المعيار الثاني هو مدى كفاءة حالات الاختبار المولدة من خلال اختبار الطفرات وباستخدام الأداة MuDroid.

الفصل السادس: تقييم الأداة BeeDroid

6-1 مقدمة

قدمنا في الفصل الرابع آلية بناء الأداة BeeDroid وخطوات التنفيذ لتوليد حالات اختبار لتطبيقات Android انطلاقاً من ملف الـ apk مروراً بمرحلة تحليل التطبيق لاستخراج بيان يمثل التطبيق ومكوناته باستخدام الأداة EGator ثم تمرير البيان الناتج عن التحليل لخوارزمية النحل الصنعية لتبدأ البحث ضمن البيان والوصول لحالات اختبار أمثليه.

في هذا الفصل سنقوم بتقييم الأداة وفعاليتها كما يلي:

1- استخدام معيار تغطية الشيفرة البرمجية ومقارنة التغطية التي تقدمها الأداة BeeDroid مع الأدوات Monkey، EvoDroid، وMobiGUITAR، الأداة ذات التغطية الأعلى تعتبر أفضل لأن احتمال اكتشاف الأخطاء أكبر، تم اعتماد هذا المعيار في معظم دراسات توليد حالات الاختبار [5] [55] [48] [49] [50] [53] .

2- استخدام اختبار الطفرات للتأكد من فاعلية حالات الاختبار المولدة وقدرتها على اكتشاف الأخطاء في الكود إن وجدت، من خلال حقن أخطاء في التطبيق ثم التأكد من قدرة حالات الاختبار المولدة على اكتشافها، تم استخدام هذا المعيار في اختبار التطبيقات التقليدية للتأكد من فاعلية حالات الاختبار [5] [55] [56] [57]، ثم في مجال تطبيقات Android تم تطوير الأداة MuDroid [58] لاختبار الطفرات كما سنذكر فيما بعد.

6-2 التطبيقات المستخدمة للتقييم

تم اختيار مجموعة من التطبيقات موضحة في الجدول (6-1)، هذه التطبيقات الأشيع والمستخدمة غالباً في الأبحاث لاختبار أو تحليل تطبيقات Android، تم اختيار هذه التطبيقات من [59] F-Droid والذي يعتبر مستودعاً مفتوح المصدر لتطبيقات Android وتم اعتماده في دراسات عدة لاستخدام التطبيقات منه للاختبار [50] [51] [53]، يبين الجدول (6-1) توصيف بسيط للتطبيقات التي تم اختيارها للاختبار.

بداية تم تحليل هذه التطبيقات باستخدام الأداة EGator لاستخراج مكونات التطبيق الأربعة الأساسية لبناء بيان يمثل التطبيق للبحث ضمنه وتوليد حالات الاختبار، يبيّن الجدول (6-1) أيضاً المكونات الأساسية لكل تطبيق، حيث يمثل التطبيق example التطبيق الذي تم استخدامه للتحليل في الفصل الرابع.

الجدول (6-1) التطبيقات المستخدمة لتقييم الأداة BeeDroid

	التطبيق	توصيف التطبيق	عدد الأنشطة	عدد الخدمات	عدد مستقبلات البث	عدد مزود المحتوى
1	Example	مثال الفصل الرابع	2	1	2	1
2	Calculator	آلة حاسبة	1	0	0	0
3	ToDoManager	إدارة المهام اليومية	12	8	7	1
4	Tic-Tac-Toe	لعبة	8	7	8	2
5	Compass	بوصلة	5	1	1	0
6	PDFViewer	مستعرض pdf	2	0	0	2
7	PasswordMaker	إنشاء كلمة مرور	3	0	0	0
8	NotePad	تسجيل ملاحظات	17	5	6	3
9	TomDroid	إدارة ملاحظات	11	0	0	2
10	AlarmClock	منبه	26	13	8	4
11	AardDict	قاموس	7	1	2	0
12	BookWorm	إدارة كتب الكترونية	3	4	10	3
13	OpenManager	مستعرض ملفات	6	8	5	3
14	ContactManager	إدارة جهات الاتصال	6	8	5	2
15	TippyTipper	حساب فواتير (المال)	10	0	2	0
16	MultiSMS	إرسال SMS لعدة مستخدمين	10	12	15	5
17	MunchLife	تتبع لعبة ورق (ترفيه)	2	0	1	1
18	Weight	مراقبة الوزن (صحة)	5	3	7	3
19	Ringdroid	اختيار نغمات	3	1	0	2

بعد تحليل التطبيقات وتمريرها لخوارزمية النحل من خلال الأداة BeeDroid، النتيجة هي عبارة عن ملف بلاحقة java يحوي حالات الاختبار المولّدة، وتم اختيار تنسيق صف java الناتج بشكل يتلاءم مع صفوف الأداة Robotium التي استخدمناها لتنفيذ الاختبارات.

6-3 أدوات اختبار تطبيقات Android

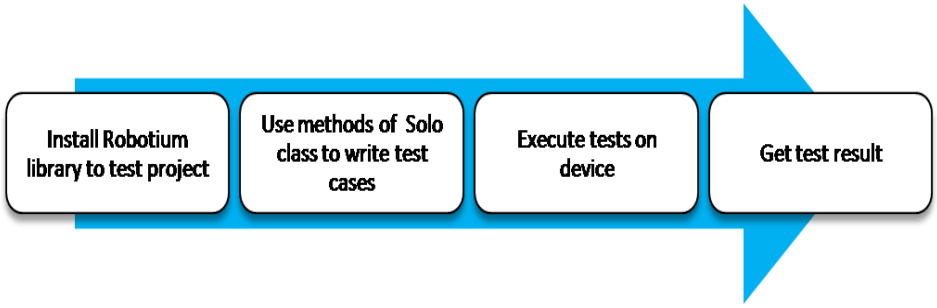
سنذكر فيما يلي أهم الأدوات التي استخدمناها لإنجاز عملية الاختبار بعد توليد الحالات والأدوات المستخدمة للتقييم من خلال حساب تغطية الشيفرة واختبار الطفرات للتحقق من فاعلية الحالات.

6-3-1 الأداة Emma [60]:

أداة مفتوحة المصدر لقياس وإعطاء تقرير لمقدار تغطية كود Java، بما أنّ تطبيقات Android غالباً تكتب بلغة Java يستخدم الباحثون هذه الأداة لتقييم فعالية عملية الاختبار، تم تضمين هذه الأداة مع Android SDK توفر تقارير (كنسبة مئوية) لتغطية الصفوف/كتل التعليمات/ الأسطر البرمجية/ التوابع. (استخدمنا تغطية statement لتقييم تغطية كامل تعليمات التطبيق وقياس كمية التعليمات المنفذة في التطبيق).

6-3-2 الأداة Robotium [61] :

إطار عمل لأتمتة اختبار Android، يسهل عملية كتابة وتنفيذ اختبارات الصندوق الأسود لتطبيقات Android، يمكن لمسؤولي عملية الاختبار كتابة حالات وسيناريوهات اختبار والتي تغطي الأنشطة والمكونات المتعددة لتطبيقات Android، اعتمدنا على هذه الأداة لتنفيذ حالات الاختبار التي تم توليدها، يوضح الشكل (6-1) المراحل لتنفيذ الاختبار باستخدام هذه الأداة.



الشكل (6-1) مراحل اختبار تطبيق Android باستخدام الأداة Robotium

6-3-3 الأداة MuDroid [58] :

أداة لاختبار الطفرات Mutation Testing في تطبيقات Android، اختبار الطفرات يعتبر مكملاً لاختبار التكامل للتأكد من أن التطبيقات تعمل كما هو متوقع دون عيوب، الهدف الأساسي لاختبار الطفرات هو

تحديد قدرة أداة الاختبار على اكتشاف الأخطاء إن وجدت في التطبيق من خلال حقن التطبيق بمجموعة أخطاء ثم استخدام أداة الاختبار للتأكد من قدرتها على كشف الخطأ، يستخدم اختبار الطفرات في الدراسات لتقييم أداة الاختبار، سنستخدم الأداة MuDroid للتحقق من جودة حالات الاختبار التي تم توليدها ولتقييم جودة الخوارزمية المطورة سنذكر تفاصيل عملها لاحقاً.

6-4 قياس تغطية الشيفرة البرمجية

ذكرنا في الفصل الثاني أن من معايير تقييم حالات الاختبار قياس مدى تغطية هذه الحالات للشيفرة البرمجية المراد اختبارها، باستخدام الأداة Emma تم قياس مدى تغطية حالات الاختبار للشيفرة لتطبيقات Android المختارة ثم مقارنة النتائج مع تغطية الأدوات الأخرى للشيفرة البرمجية.

6-4-1 مقارنة الأداة BeeDroid مع الأداة Monkey

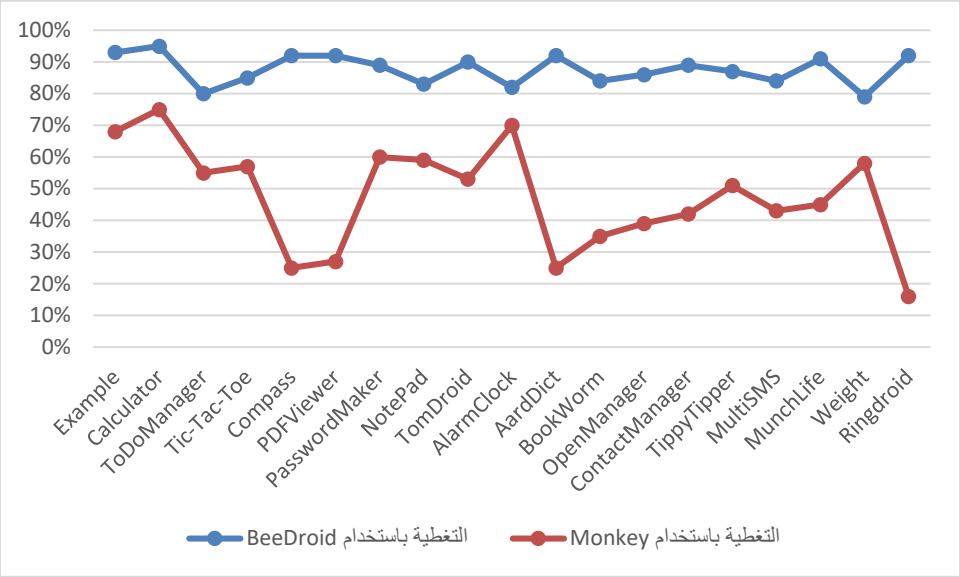
تعتبر الأداة Monkey [46] من أدوات الاختبار العشوائي، مدعومة من Google ومضمنة مع الـ SDK، كل دراسات الاختبار لتطبيقات Android تعتمد المقارنة مع هذه الأداة [47] [48] [49] [52] [53]، تقوم Monkey بإرسال تدفق من أحداث المستخدم العشوائية للتطبيق، تمت المقارنة من خلال أحداث نفس عدد حالات الاختبار المولدة من الأداة BeeDroid ثم قياس تغطية الشيفرة للتطبيق بعد استخدام الأداتين للاختبار.

يبين الجدول (6-2) مقارنة تغطية الشيفرة البرمجية للأداة BeeDroid مع Monkey.

من الجدول (6-2) نلاحظ أنّ التغطية باستخدام Monkey منخفضة وهذا يتوافق مع نتائج الدراسات السابقة [47] [48] [49] [52] [53]، Monkey ترسل أحداث عشوائية بغض النظر عن طبيعة أو بنية التطبيق، لا توجد قواعد تتبعها للأحداث المولدة، بينما BeeDroid تنطلق من فضاء بحث هو البيان الذي يمثل التطبيق ومكوناته بالتالي الأحداث المولدة تعتمد على أحداث التطبيق وأحداث النظام واستدعاءاته للمكونات، البيان الناتج يمثل مستقبلات البث المحلية فقط كونها تعبّر عن التفاعلات بين مكونات التطبيق وهذا ما يبرر انخفاض نسبة التغطية لبعض التطبيقات التي فيها مستقبلات بث أكثر، مع العلم أنّ العدد الممثل في الجدول (6-2) يمثل عدد مستقبلات البث الكلي للتطبيق.

الجدول (6-2) مقارنة تغطية الشيفرة البرمجية للأداتين BeeDroid وMonkey

التطبيق	التغطية باستخدام BeeDroid	التغطية باستخدام Monkey
Example	93%	68%
Calculator	95%	75%
ToDoManager	80%	55%
Tic-Tac-Toe	85%	57%
Compass	92%	25%
PDFViewer	92%	27%
PasswordMaker	89%	60%
NotePad	83%	59%
TomDroid	90%	53%
AlarmClock	82%	70%
AardDict	92%	25%
BookWorm	84%	35%
OpenManager	86%	39%
ContactManager	89%	42%
TippyTipper	87%	51%
MultiSMS	84%	43%
MunchLife	91%	45%
Weight	79%	58%
Ringdroid	92%	16%



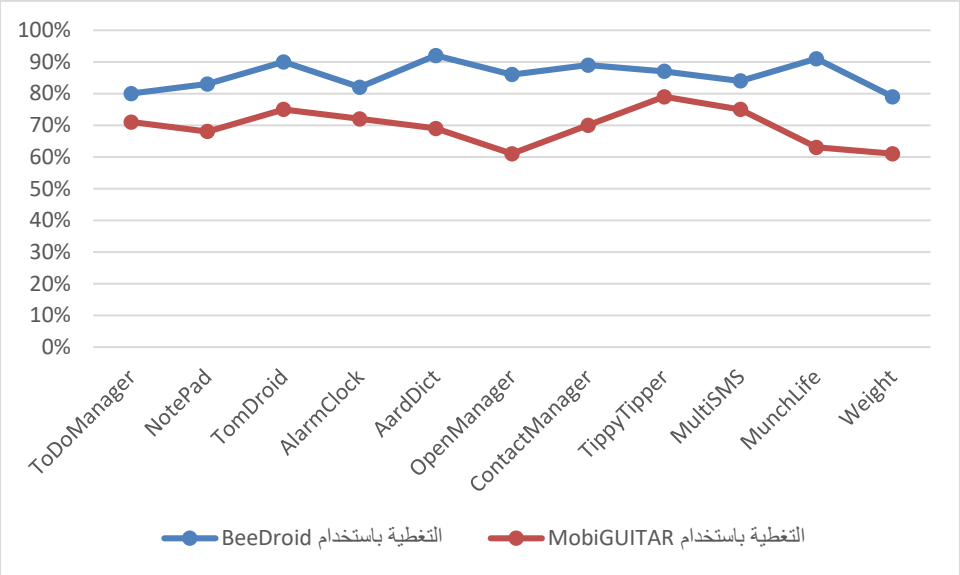
الشكل (6-2) مقارنة تغطية الأداتين BeeDroid وMonkey

6-4-1 مقارنة الأداة BeeDroid مع الأداة وMobiGUITAR

تعتبر الأداة MobiGUITAR [48] من أدوات الاختبار المعتمد على النموذج التي اعتمدت على تحليل واجهات المستخدم لبناء بيان يمثل التطبيق، تم الاعتماد على النتائج المنشورة والمقدمة من الأبحاث للمقارنة مع هذه الأداة، في الجدول (6-3) تم مقارنة تغطية الشيفرة البرمجية للأداة BeeDroid مع الأداة وMobiGUITAR ، الأداة BeeDroid تفوقت في التغطية على الأداة وMobiGUITAR بسبب التمثيل المحدود للتطبيق من قبل وMobiGUITAR والذي اقتصر على واجهات المستخدم فقط بالإضافة لآلية توليد حالات الاختبار التي يمكن أن تولد تسلسل أحداث غير منطقي، بينما عند بناء فضاء البحث الابتدائي باستخدام BeeDroid تم الأخذ بعين الاعتبار خصائص العقد السابقة واللاحقة لكل حدث واستخدام مفهوم المكدسات لتوليد تسلسل منطقي للأحداث.

الجدول (6-3) مقارنة تغطية الشيفرة البرمجية للأداتين BeeDroid وMobiGUITAR

التطبيق	التغطية باستخدام BeeDroid	التغطية باستخدام MobiGUITAR
ToDoManager	80%	71%
NotePad	83%	68%
TomDroid	90%	75%
AlarmClock	82%	72%
AardDict	92%	69%
OpenManager	86%	61%
ContactManager	89%	70%
TippyTipper	87%	79%
MultiSMS	84%	75%
MunchLife	91%	63%
Weight	79%	61%



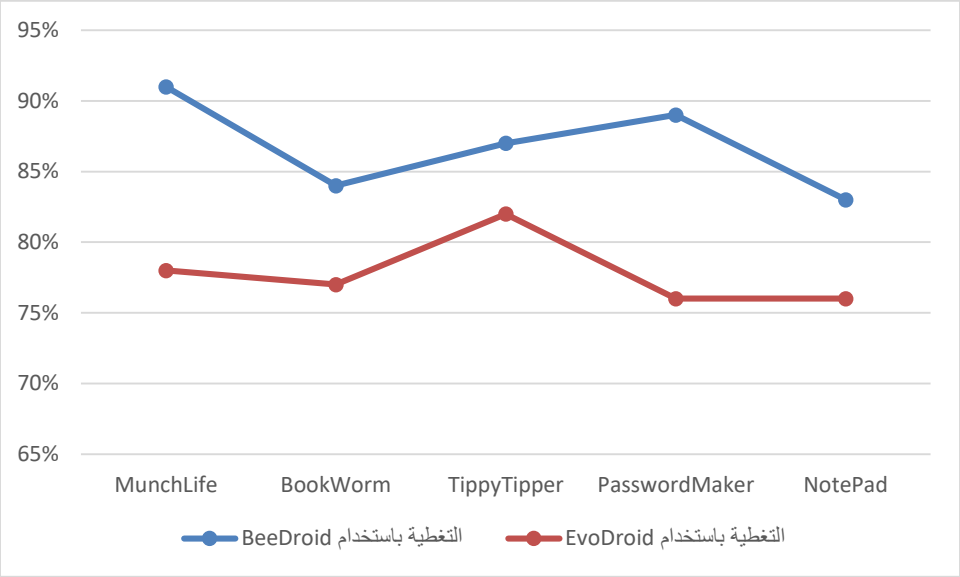
الشكل (6-3) مقارنة تغطية الأداتين BeeDroid وMobiGUITAR

6-4-1 مقارنة الأداة BeeDroid مع الأداة EvoDroid

تعتبر الأداة EvoDroid [53] من أدوات الاختبار المعتمد على البحث والتي استخدمت الخوارزمية الجينية للبحث بعد توليد نموذجين لتوصيف التطبيق، كما ذكرنا في الفصل الرابع تم بناء بيان يمثل الاستدعاءات لواجهات المستخدم الرسومية حيث تم تثميل فقط واجهات التطبيق، الأداة غير متاحة لذلك تم الاعتماد على النتائج المقدمة من الدراسة والمقارنة مع التطبيقات المتوافقة مع التي اخترناها للاختبار، يبين الجدول (6-4) نتائج مقارنة تغطية الشيفرة البرمجية للأداة BeeDroid مع الأداة EvoDroid، قدمت الأداة BeeDroid أداء أفضل من ناحية التغطية بسبب طبيعة البيان الممثل للتطبيق والذي شمل مكونات التطبيق الأربعة والتي لم يتم تمثيلها في الأداة EvoDroid.

الجدول (6-4) مقارنة تغطية الشيفرة البرمجية للأداتين BeeDroid و EvoDroid

التطبيق	التغطية باستخدام BeeDroid	التغطية باستخدام EvoDroid
MunchLife	91%	78%
BookWorm	84%	77%
TippyTipper	87%	82%
PasswordMaker	89%	76%
NotePad	83%	76%

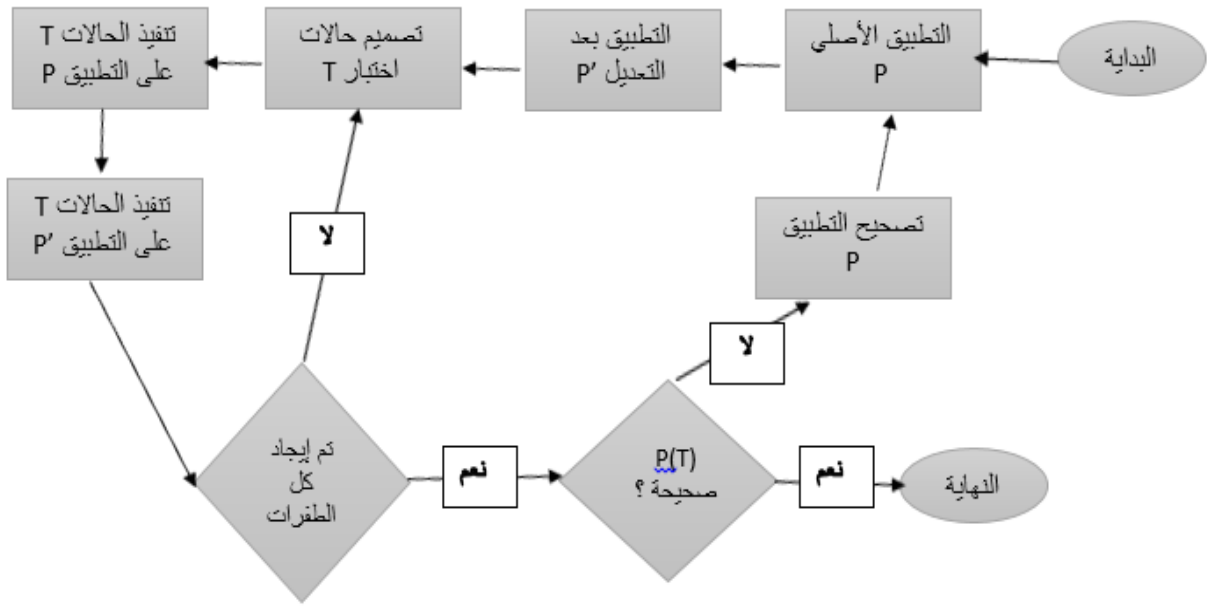


الشكل (6-4) مقارنة تغطية الأداتين BeeDroid و EvoDroid

6-5 اختبار الطفرات Mutation Testing

6-5-1 تعريف عام باختبار الطفرات

قدمنا في الفصل الثاني تعريفاً لاختبار الطفرات، يستخدم اختبار الطفرات (أو تحليل الطفرة) لتصميم الاختبارات للبرامج الجديدة وتقييم جودة حالات اختبار البرامج الموجودة، ظهر اختبار الطفرات عام 1978 لتحديد وكشف نقاط الضعف في أدوات الاختبار، الشكل (6-5) يبيّن خطوات عملية اختبار الطفرات، بداية يتم تعديل في التطبيق P المراد اختباره لينتج التطبيق P' والذي يكون عادةً إصدار فيه خطأ يتم إنشاء هذه الأخطاء باستخدام قواعد تسمى قواعد الطفرات Mutation Operators، مثلاً في لغة Java توجد قواعد لطفرات Relational Operator Replacement (ROR) تستبدل معاملات المقارنة مثل إشارة < بمعاملات أخرى منها (<=, >=, ==, !=)، بعد تعديل التطبيق يقوم المختبر بتنفيذ حالات الاختبار على التطبيق الأصلي P والمعدل P'، إذا كانت النتائج من الاختبارين غير متطابقة عندها نقول أنّ الاختبار T وجد (اصطاد) الخطأ p.



الشكل (6–5) خطوات اختبار الطفرات

يبيّن الشكل (6–6) مثالاً لتطبيق ثم تعديل هذه التطبيق من خلال تعديل باستخدام قواعد الطفرات ROR، بعد تصميم حالات الاختبار وتنفيذها حتى تكون الحالات فعالة يجب أن تعطي نتائج مختلفة عند تنفيذها على التطبيقين الأصلي والمعدل، يبيّن الشكل (6–7) مثالاً لحالات اختبار، في الحالة الأولى غير فعالة لأن نتائج الاختبارين متشابهة ولم يتم تمييز التغيير في التطبيق، بينما الحالة الثانية فعالة من أجل هذا التطبيق المعدل ونقول إنها اكتشفت التعديل.

```
int lastZero (int [ ] x)
{
  for (int i = x.length-1; i >= 0; i–)
  {
    if (x [i] == 0)
      return i;
  }
  return -1;
}
```

```
int lastZero (int [ ] x)
{
  for (int i = x.length-1; i > 0; i–) // ROR mutant
  {
    if (x [i] == 0)
      return i;
  }
  return -1;
}
```

الشكل (6–6) مثالاً يبين اختبار الطفرات

```
Input: x = [1, 1, 2]
Original output: -1
Mutant output: -1
```

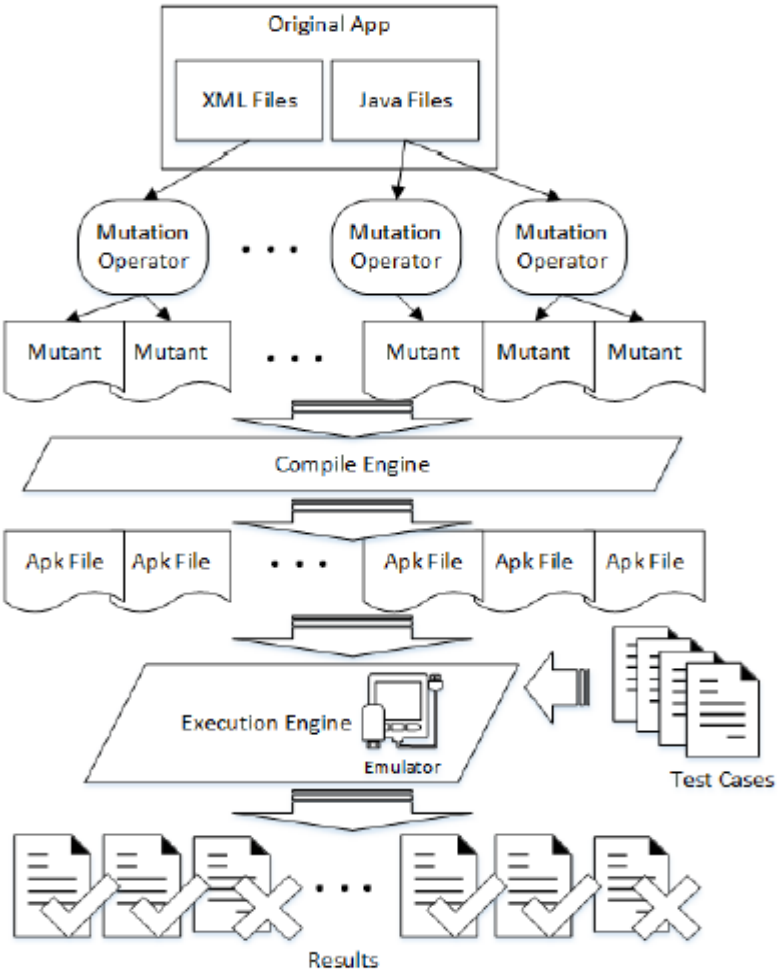
```
Input: x = [0, 1, 2]
Original output: 0
Mutant output: -1
```

الشكل (6–7) مثالاً يبين حالات اختبار الطفرات مصممة للتطبيق في الشكل السابق

6–5–2 استخدام اختبار الطفرات مع تطبيقات Android

لا يمكن تطبيق اختبار الطفرات مباشرة على تطبيقات Android بنفس الآلية المستخدمة لاختبار الطفرات لتطبيقات Java التقليدية لاختلاف البنية والخصائص لتطبيقات Android كما ذكرنا مسبقاً حيث بعد كل تعديل يجب ترجمة التطبيق كملف بلاحقة apk ثم تنزيل التطبيق على جهاز موبايل أو محاكي لتنفيذ الاختبارات، والطفرات يجب أن تعدل في ملفات xml التابعة للتطبيق بالإضافة للاستدعاءات الخاصة بدورة حياة كل مكون.

قدمت الدراسة [59] الأداة MuDroid لتعديل تطبيقات Android وإنشاء طفرات في ملفات xml وملفات الشيفرة Java وآلية عمل هذه الأداة موضحة في الشكل (6–8)، اعتمدت الأداة على إضافة سبعة عشر طفرة أو تعديل لتطبيقات Android،



الشكل (6-8) خطوات عمل الأداة MuDroid

من المعاملات التي تم تغييرها عند استخدام اختبار الطفرات باستخدام الأداة MuDroid معاملات تغيير ملفات الـ xml المرتبطة بكل واجهة من خلال حذف زر أو مربع نص أو سماحية، أو معاملات حذف استدعاءات لها علاقة بدورة حياة الأنشطة أو الخدمات، ومعاملات تغير الـ intent الهدف في استدعاءات معالجة الأحداث، يبيّن الشكل (6-9) مثالاً لتعديل الـ intent الهدف لإحدى الاستدعاءات.

```
public void startActivityB (View v)
{
  Intent intent = new Intent (ActivityA.this, ActivityB.class);
  startActivity (intent);
}
```

A. Original

```
public void startActivityB (View v)
{
  Intent intent = new Intent (ActivityA.this, ActivityC.class);
  startActivity (intent);
}
```

B. Mutant

الشكل (6-9) مثالاً يبين اختبار الطفرات لتطبيقات Android

لتقييم الأداة BeeDroid استخدمنا نفس المعاملات التي قدمتها الأداة MuDroid لتعديل التطبيق بالإضافة لمعاملين لتعديل إجرائيات دورة حياة المكونين Broadcast Receiver وContent Provider بنفس الآلية المتبعة في الأداة MuDroid للتعديل.

بعد إجراء التعديلات على التطبيق حسب نوع المعامل الذي سيطبق، يتم توليد حالات اختبار لتنفيذها وهنا استخدمنا أيضاً الأداة Robotium لتنفيذ حالات الاختبار بعد تعديل التطبيق ومراقبة النتائج قبل وبعد تطبيق الطفرات، فعالية اختبار الطفرات (Score) يقيم من خلال حساب عدد الطفرات التي تم اكتشافها، إذا كان Score =1 هذا يشير إلى أن حالات الاختبار اكتشفت كل الطفرات.

$$\text{Score} = (\Sigma \text{ faults found} / \Sigma \text{ faults injected})$$

يبيّن الجدول (6-5) قواعد الطفرات التي تم استخدامها لتعديل الشيفرة لتطبيقات Android.

الجدول (6-5) قواعد الطفرات المستخدمة

التصنيف	قاعدة طفرة Android
طفرات متعلقة بالأحداث	Intent Target Replacement (ITR)
	OnClick Event Replacement (ECR)
طفرات متعلقة بدورة حياة المكونات	Activity Lifecycle Method Deletion (MDL)
	Service Lifecycle Method Deletion (SMDL)
	Broadcast Receiver Lifecycle Method Deletion (BMDL)
	Content Provider Lifecycle Method Deletion (PMDL)
طفرات متعلقة بتعديل ملفات XML	Activity Permission Deletion (APD)

تم تطبيق القواعد السابقة على التطبيقات المحددة في الجدول (6-6)، بعض المعاملات تم تطبيقها عدد من المرات حسب مكونات التطبيق بالإضافة للاستفادة من السماحيات المطلوبة لكل تطبيق والتي تم اشتقاقها خلال عملية تحليل التطبيق باستخدام EGator، ثم حساب قيمة الـ Score:

الجدول (6-6) قواعد الطفرات المطبقة على التطبيقات

التطبيق	عدد مرات القواعد المطبقة لكل تطبيق						
	ITR	ECR	MDL	SMDL	BMDL	PMDL	APD
Example	4	3	3	1	4	2	1
Calculator	1	5	1	0	0	0	0
ToDoManager	15	3	7	5	7	1	1
Compass	5	3	2	1	1	0	1
TomDroid	10	7	5	0	0	1	0
BookWorm	5	5	3	3	2	2	0
OpenManager	5	4	3	10	7	3	2
TippyTipper	7	3	5	0	1	0	0
MultiSMS	10	5	6	5	5	3	3
MunchLife	2	3	1	0	0	1	0

وضحنا في الجدول (6-7) نتيجة تطبيق القواعد على التطبيقات ونتيجة الـ Score لكل تطبيق، كما نلاحظ من الجدول قدرة حالات الاختبار المولدة باستخدام الأداة BeeDroid على اكتشاف معظم الطفرات، تراوحت نسبة الـ Score بين 0.91 و1 وهذا يؤكد فاعلية الحالات المولدة وقدرتها على اكتشاف الأخطاء والتغييرات في التطبيق، بعض الحالات لم يتم اكتشافها التي لها علاقة بمستقبلات البث الخارجية أو بعض السماحيات المطلوبة ضمن هذه المستقبلات، لكن الطفرات المتعلقة بالأنشطة والخدمات وباقي المكونات تم اكتشافها بسبب تمثيل هذه المكونات ضمن البيان ودورة حياة كل مكون مما يولد حالات تجرب كل حالات المكون وأي تغيير سوف يتم اكتشافه وهذا ما أكدته التجربة.

الجدول (6-7) نتيجة تطبيق قواعد الطفرات على التطبيقات

التطبيق	قيمة MS الخاصة بكل قاعدة							قيمة Score النهائية
	ITR	ECR	MDL	SMDL	BMDL	PMDL	APD	
Example	1	1	1	1	0.75	1	1	0.95
Calculator	1	1	1	0	0	0	0	1
ToDoManager	1	1	1	1	0.7	1	1	0.94
Compass	1	1	1	1	1	0	1	1
TomDroid	1	1	1	0	0	1	0	1
BookWorm	1	1	1	1	0.7	1	0	0.95
OpenManager	1	1	1	1	0.7	1	1	0.91
TippyTipper	1	1	1	0	1	0	0	1
MultiSMS	1	1	1	1	0.8	1	0.7	0.95
MunchLife	1	1	1	0	0	1	0	1

6-6 خاتمة

استخدام الأداة BeeDroid سوف يسهل عمل المختبرين في هذا المجال، المختبر ليس بحاجة لمعرفة آلية تنفيذ لتطبيق ومساراته وحالاته ومكوناته المختلفة، دخل الأداة ملف apk. للتطبيق المراد اختباره والخرج هو وثيقة حالات اختبار متوافقة مع حالات الأداة Robotium ليتم تنفيذها بسهولة لاختبار تطبيقات Android، مما سيوفر الوقت والجهد اللازمين لمرحلة عملية الاختبار.

قدمنا في هذا الفصل تقييم للأداة BeeDroid من ناحية تغطية الشيفرة البرمجية للتطبيق وتبين أن BeeDroid حققت أعلى نسبة في التغطية مقارنة مع أدوات سابقة، بالإضافة لاستخدام اختبار الطفرات للتأكد من فاعلية الحالات المولدة وقدرتها على اكتشاف الخطأ في التطبيق، التطبيقات التي اعتمدنا عليها للتقييم تم استخدامها في أغلب الدراسات البحثية في مجال الاختبار لذلك استخدمناها.

في الفصل السادس سنقدم نتائج الدراسة بالإضافة لمقترحات وتوصيات لتكون أعمال مستقبلية للباحثين في هذا المجال.

الفصل السابع: النتائج والتوصيات

7-1 مقدمة

سنعرض في هذا الفصل النتائج التي توصلنا إليها من خلال بحثنا، بالإضافة إلى عرض بعض التوصيات المقترحة للتحسين لزيادة فاعلية الأداة BeeDroid.

7-2 النتائج

ظهور تطبيقات الموبايل وانتشارها بشكل واسع جعل الحاجة ملحة لاختبارها والتأكد من صحة الخدمات والوظائف المقدمة للمستخدمين، الأشيع والأكثر استخداماً هي تطبيقات Android، بنية هذه التطبيقات تختلف عن بنية التطبيقات التقليدية مما استدعى وجود آليات ودراسات جديدة لاشتقاق أدوات لاختبار تطبيقات Android بشكل فعال وسهل بالنسبة للمختبرين، المرحلة الأهم في عملية الاختبار هي توليد حالات الاختبار، قدمنا من خلال الدراسة الأداة BeeDroid التي أتمتت عملية توليد حالات الاختبار لتطبيقات Android والتي تساعد المختبرين بشكل كبير، حيث دخل الأداة ملف apk. يمثل التطبيق المراد اختباره والخرج هو وثيقة بلاحقة ملف java تحتوي على حالات الاختبار ليتم تنفيذها من خلال أدوات اختبار تطبيقات Android، المختبر ليس بحاجة لأية معرفة عن بنية التطبيق ومساراته وآلية التنفيذ، لبناء الأداة BeeDroid اعتمدنا الدمج بين منهجيتي توليد حالات الاختبار المعتمد على البحث والمعتمد على النموذج، حيث يتم استخدام خوارزميات البحث الأمثل لتوجيه البحث ضمن فضاء بحث بدائي، في حالتنا فضاء البحث البدائي ليس عشوائياً وإنما بيان يمثل مكونات التطبيق وآلية الانتقال فيما بينها، تم بناء هذا البيان باستخدام الأداة التي طورناها EGator التي تشتق نموذج هو بيان يمثل التطبيق.

كانت مراحل دراستنا مكونة من عدة مراحل كما يلي:

- في البداية قدمنا دراسة لمفهوم اختبار البرمجيات وأهم النقاط المتعلقة بهذه المرحلة من مراحل تطوير البرمجية وبالأخص مرحلة توليد حالات الاختبار، وهذا ما تم في الفصل الأول.
- ثم في الفصل الثالث قمنا بدراسة المفاهيم المتعلقة بالخوارزميات التطورية، ثم كيفية استخدامها في مجال اختبار البرمجيات، وقدمنا تفصيلاً لخوارزميتي النحل الصنعية والخوارزمية الجينية حيث تبين من خلال الدراسة أنهما الأشيع استخداماً في مجال توليد حالات الاختبار للبرمجيات.

- في الفصل الرابع قدمنا مقارنة بين خوارزميتي النحل الصنعية والحينية في مجال توليد حالات اختبار لتطبيقات واجهات المستخدم الرسومية GUI، كون هذه التطبيقات تعتبر من التطبيقات المقادة بالأحداث والتي تتشابه مع تطبيقات Android من حيث مكون الواجهة والاستجابة لأحداث المستخدم، وتبين أن خوارزمية النحل أفضل وأسرع من الخوارزمية الجينية.
- في الفصل الخامس قدمنا دراسة مفصلة لتطبيقات Android، بنية ومكونات هذه التطبيقات، الآليات والدراسات المتعلقة في مجال توليد حالات اختبار لتطبيقات Android، ثم قدمنا الأداة BeeDroid المكونة من مرحلتين، بداية تحليل ملف apk للتطبيق المراد اختباره لاشتقاق بيان يمثل التطبيق بشكل واسع باستخدام الأداة EGator التي تم تطويرها لتناسب وتحلل مكونات التطبيق بشكل كامل، ثم تمرير هذا البيان لخوارزمية النحل الصنعية لاشتقاق حالات اختبار أمثليه منه.
- في الفصل السادس قدمنا تقييماً للأداة BeeDroid، المعايير المعتمدة للتقييم هي تغطية الشيفرة البرمجية للحالات المولدة واختبار الطفرات للتأكد من فاعلية الحالات المولدة، وتبين أن الأداة BeeDroid قدمت تحسيناً في مجال تغطية الشيفرة كونها تعتمد على البيان الذي يمثل التطبيق، بالتالي توليد الحالات ينطلق من بنية التطبيق، بالإضافة لقدرتها على اكتشاف الطفرات التي تم حقنها من خلال اختبار الطفرات.

3-7 التوصيات المقترحة

يعتبر مجال اختبار تطبيقات Android من المجالات الحديثة والتي استدعت انتباه الباحثين في مجال هندسة البرمجيات، بدأت الدراسات في هذا المجال لأتمتة مرحلة الاختبار لتوفير الوقت والجهد اللازمين لهذه المرحلة، الأداة BeeDroid تساعد المختبرين بشكل كبير، من التحسينات التي يمكن أن نوصي بها كأعمال مستقبلية لتطوير هذه الأداة:

- ✓ أخذ آلية الاتصال والتفاعل بين تطبيقات Android بعين الاعتبار وخاصة استقبال البث من تطبيقات خارجية وتضمين هذا التفاعل في مرحلة توليد البيان الذي يمثل، مما يحسن من البيان الذي تم بناؤه، بالتالي تحسين تغطية الشيفرة في حالة التطبيقات التي تحوي عدداً من مستقبلات البث الخارجي.
- ✓ ذكرنا في الفصل الخامس تنوع أنظمة التشغيل التي تدعم تطبيقات الموبايل، من التحسينات الممكنة تطوير الأداة BeeDroid واعتماد المنهجية المتبعة بها لتطوير أداة لتوليد حالات اختبار لتطبيقات IOS أو تطبيقات Windows.

المراجع العلمية

[1] Sommervilla.L, 2011- Software Engineering, Ninth Edition, Library of Congress, Control Number 20090538.

[2] Badgett.T, Sandler.C, 2012– The Art of Software Testing, Third Edition Published in Canada.

[3] Web Page, https://info.localytics.com/blog/app-retention-improves, last access September 2018.

[4] Kochhar.P.S, Thung.F, Nagappan.N, Zimmermann.T, and DAVID.L, 2015- Understanding the Test Automation Culture of App Developers. In Software Testing, Verification and Validation (ICST), IEEE 8th International Conference on, pages 1-10, April 2015.

[5] Kaner.C, 2003- What Is a Good Test Case? ,Florida Institute of Technology Department of Computer Sciences

[6] Bauersfeld.S, Wappler.S, 2011- A Metaheuristic Approach to Test Sequence Generation for Applications with a GUI, Berner und Mattner Systemtechnik GmbH, Gutenbergstr. Berlin, Germany.

[7] Horgan.J, London.S, BELLCORE.M, 1994- Achieving Software Quality with Testing Coverage Measures, IEEE ISSN 0018-9162.

[8] Arlt.S, Podelski.A, 2012- Black-Box Verification for GUI Applications, Albert-Ludwigs-Universit¨at Freiburg.

[9] Carino.S, 2016- Dynamically Testing Graphical User Interfaces, The University of Western Ontario

[10] Delamaro.M.E, Vincenzi.A, 2006- Muta-Pro: Towards The Definition Of A Mutation Testing Process, Instituto de Informática Universidade Federal de Goiás Goiânia, GO, Brazi.

[11] Mohi-Aldeen.M, Deris.S, Mohamad.R, 2014- Systematic Mapping Study in Automatic Test Case Generation, University of Mosul, Mosul, Iraq.

[12] Jeorge.N, Stephen.W, 2006- Numerical Optimization, Second Edition, Library of Congress, Control Number 2006923897.

[13] Felix.S, 2001- Introduction to Evolutionary Algorithms, University of Tuebingen.

[14] Bhateja.N, 2016- Various Artificial Intelligence Approaches in Field of Software Testing, Faculty, Department of Computer Science & Engineering, Amity University Gurgaon, Haryana, India.

[15] Bao.X, Qian.J, 2017-Path-Oriented Test Cases Generation Based Adaptive Genetic Algorithm, Department of Information Technology, Zhejiang Sci-Tech University.

[16] Sayyari.F, Emadi.S, 2015- Automated Generation Of Software Testing Path Based On Ant Colony, Islamic Azad University, Mashhad, Iran.

[17] Ojha.D, Sahoo.R, 2016-Automated Test Case Generation And Optimization: A Comparative Review, International Journal of Computer Science & Information Technology.

[18] Sinah.A, Khari.M, 2019- Performance analysis of six meta-heuristic algorithms over automated test suite generation for path coverage-based optimization, Germany, part of Springer Nature 2019

[19] David.B, Ralph.M, 1993- An Overview of Genetic Algorithms Part 1 Fundamentals, Inter University Committee on Computing.

[20] Cui.X, Potok.T, 2012- Swarm Intelligence: Concepts, Models and Applications, School of Computing, Kingston, Ontario, Canada.

[21] Karaboga.D, 2005- An Idea Based on Honey Bee Swarm For Numerical Optimization, Technical Report-TR06, Department of Computer Engineering, Engineering Faculty, Erciyes University.

[22] Akay.B, Karaboga.D, 2010- A modified Artificial Bee Colony algorithm for real-parameter optimization, Turkey, Elsevier, journal.

[23] Bauersfeld.S, Wappler.S, 2011- A Metaheuristic Approach to Test Sequence Generation for Applications with a GUI, Berner und Mattner Systemtechnik GmbH, Gutenbergstr. Berlin, Germany

[24] Pacheco.C, Lahiri.K, ERNST.M, 2007 -Feedback-Directed Random Test Generation. In Proceedings of the 29th International Conference on Software Engineering.

[25] Hicinbothom.J, Zachary.W, 1993 -A Tool For Automatically Generating Transcripts Of Human-Computer Interaction. In Proceedings of the Human Factors and Ergonomics Society 37th Annual Meeting, volume 2 of SPECIAL SESSIONS: Demonstrations, page 1042.

[26] Banerjee,A. Memon, 2003- GUI Ripping: Reverse Engineering of Graphical User Interfaces for Testing, Department of Computer Science University of Maryland

[27] Carino.S, 2016- Dynamically Testing Graphical User Interfaces, The University of Western Ontario

[28] Nguyen.N, Robbins.B, 2013- GUITAR: An Innovative Tool For Automated Testing Of GUI-Driven Software, Springer Science+Business Media New York

[29] Jones.B, Sthamer.H, 1996- Automatic Structural Testing Using Genetic Algorithms, The Software Engineering Journal 11(5) p 299-306.

[30] Yongzhong.L, Danping.Y, 2008 - Development of an Improved Gui Automation Test System Based on Event-Flow Graph. International Conference on Computer Science and Software Engineering.

[31] Wilke.F, Pulvermuller.E, 2017 - Search based GUI Test Generation in Java Comparing Code-based and EFG-based Optimization Goals, Institute of Computer Science, University of Osnabr¨uck, Wachsbleiche 27, 49090 Osnabr¨uck, Germany

[32] Mccabe.T, 1982- Structured Testing: A Software Testing Methodology Using the Cyclomatic Complexity Metric, NBS Special Publication, National Bureau of Standards.

[33] Jaffar.A, Rauf.A, 2018– Fully Automated Gui Testing And Coverage Analysis Using Genetic Algorithms, International journal of innovative computing, information & control.

[34] Web Page, URL: https://www.statista.com/statistics/271520/number-of-smartphone-users, Last access Feb-2021

[35] Web Page, URL: https://www.statista.com/statistics/272307/market-share-forecast-for-smartphone-operating-systems/, Last access Feb-2021

[36] Web Page, URL: https://www.statista.com/statistics/276623/number-of-apps-available-in-leading-app-stores, Last access Feb-2021

[37] Wiely.J, 2013- Android Application Development Cookbook, Indianapolis, Indiana

[38] Yang.S, Yan.D, 2015- Static Control-Flow Analysis Of User-Driven Callbacks In Android Applications. In International Conference on Software Engineering, pages 89–99

[39] Kong.P, Klien.J, 2019- Automated Testing of Android Apps: A Systematic Literature Review, Interdisciplinary Centre for Security, Reliability and Trust, University of Luxembourg, Luxembourg.

[40] Bartel.A, Klein.J, 2013 -Dexpler: Converting Android Dalvik Bytecode to Jimple for Static Analysis with Soot, National Research Fund, Luxembourg.

[41] Hendren.L, Sundaresan.V, PATRICK.E, 1999 -Soot - A Java Optimization Framework. In Proceedings of CASCON 1999, pages 125–135.

[42] Lam.P, Bodden.E, 2011. The Soot Framework For Java Program Analysis: A Retrospective. In Cetus Users and Compiler Infastructure Workshop (CETUS 2011), Oct. 2011.

[43] Ahmed.I, Ali-Gombe.A, 2014 -AspectDroid: Android App Analysis System, Dept. of Computer Sciencek University of New Orleans.

[44] Arzt.S, 2017 -Static Data Flow Analysis for Android Applications, Heidelberg, Germany, PHD thesis.

[45] Yang.S, Wang.Y, Rountev.A, 2015 - Static Control-Flow Analysis of User-Driven Callbacks in Android Applications, Chio State University.

[46] Monkey, URL: https://developer.android.com/studio/test/monkey, Last access May-2021.

[47] Machiry.A, Tahiliani.R, Naik.M, 2013- Dynodroid: An Input Generation System For Android Apps. In Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering.

[48] Amalfitano.D, Fasolino.A.R, Tramontana.P, Memon.A.M, 2015- Mobiguitar: Automated model-based testing of mobile pps, IEEE Software, vol. 32, no. 5, pp. 53–59, Sept 2015.

[49] Wei.Y, Mukul.P.R, Tao. X, 2013- A Grey-Box Approach for Automated GUIModel Generation of Mobile Applications, In: FASE. Vol. 13. Springer, pp. 250–265.

[50] Yuanchun.L, 2017- DroidBot: a Lightweight UI-Guided Test Input Generator For Android, In: Proceedings of the 39th International Conference on Software Engineering Companion. IEEE Press, pp. 23–26. ISBN: 1538615894.

[51] Wontae.C, George.N, Koushik.Sen, 2013- Guided GUI Testing of Android apps with minimal restart and approximate learning, In Proceedings of the 2013 ACM SIGPLAN International Conference on Object Oriented Programming Systems Languages & Applications.

[52] Yan.J, Zhang.J, Wu.T, 2017- Target Directed Event Sequence Generation for Android Applications, Technology Center of Software Engineering, Institute of Software, Chinese Academy of Sciences.

[53] Riyadh.M, Nariman.M, Sam.Malek, 2014- Evodroid: Segmented Evolutionary Testing Of Android Apps, In: Proceedings of the 22nd ACM SIGSOFT Internationa Symposium on Foundations of Software Engineering.

[54] Amatucci.N, Fasolino.A, 2015- AGRippin: A Novel Search Based Testing Technique for Android Applications, Department of Electrical Engineering and Information Technologies University Federico II of Naples.

[55] Kosindrdecha.N, Daengdej.J, 2009- A Test Case Generation Technique And Process, Autonomous System Research Laboratory Faculty of Science and Technology Assumption University, Thailand

[56] Prasanna.M, Chandram.K.R, 2009- Automatic Test Case Generation for UML Object diagrams using Genetic Algorithm, Int. J. Advance. Soft Comput. Appl., Vol. 1, No. 1, July 2009, ISSN 2074-8523

[57] Shanthi.A.V, 2011- Automated Test Cases Generation For Object Oriented Software, A.V.K.Shanthi et al./ Indian Journal of Computer Science and Engineering (IJCSE)

[58] Deng.L, 2017- Mudroid: Mutation Testing For Android Apps, George Mason University, PHD thesis.

[59] F-droid: https://f-droid.org/, Last access May-2021.

[60] Emma: https://www.emma-toolkit.org/toolkit, Last access May-2021.

[61] Robotium :https://github.com/robotiumtech/robotium, Last access May-2021.

Syrian Arab Republic
Al-Baath University
Faculty of Informatics Engineering
Department of Software Engineering
and Information Systems

Test Cases Generation using Artificial Intelligence Techniques

FA Thesis Submitted in Fulfillment of The Requirements for PHD Degree in Software Engineering and Information System

Prepared by

Eng. Bushra Talal Ghazala

Supervised by

Dr. Yosser AlSayed Souliman Atassi

A Professor at the Department of Software Engineering and Information Systems, Faculty of Informatics Engineering
Al Baath University

Homs 2021 A.D - 1442 A.H