南京大學

研 究 生 毕 业 论 文

（申请工程硕士学位）

论 文 题 目 安卓自动化测试工具的多维评估系统的设计与实现

作 者 姓 名 程鹏

学科、专业名称 工程硕士（软件工程领域）

研 究 方 向 软件工程

指 导 教 师 陈振宇 教授，房春荣 助理研究员

2021 年 5 月 20 日

学　　号 ：**MF1932027**

论文答辩日期 ：**2021 年 5 月 20 日**

指 导 教 师 ：　　**（签字）**

The Design and Implementation of Multidimensional Evaluation System for Android Automated Testing Tools

By

Peng Cheng

Supervised by

Professor **Zhenyu Chen**

Assistant Research Fellow **Chunrong Fang**

A Thesis
Submitted to the Software Institute
and the Graduate School
of Nanjing University
in Partial Fulfillment of the Requirements
for the Degree of
Master of Engineering

Software Institute

May 2021

南京大学研究生毕业论文中文摘要首页用纸

毕业论文题目：安卓自动化测试工具的多维评估系统的设计与实现

工程硕士（软件工程领域） 专业 2019 级硕士生姓名： 程鹏

指导教师（姓名、职称）： 陈振宇 教授，房春荣 助理研究员

摘 要

安卓设备的广泛应用得益于移动互联网的发展，因其开源特性，设备之间差异较大，碎片化问题日益严峻，测试对安卓应用愈发重要。应用间的激烈竞争导致开发周期缩短，也使用户对应用的要求越来越高。手工测试因时间成本过高无法满足当前开发的需要，自动化测试工具产业得以蓬勃发展。尽管大多数工具自称测试效果较好，但其测试实验对评估指标、应用集及设备等方面考虑不够全面，导致结果不够客观。因此本文提出一个较为全面客观的自动化测试工具衡量系统。该系统有利于工具的横向比较，也为后续工具的开发提供参考。

本文对现有的自动化测试工具进行调研，抽象出了异常、变异、代码覆盖、UI 控件覆盖以及碎片化支持等五个多维评判指标，并针对以上指标设计了相关检测方法，解决了评估指标不全面的问题。系统拥有可扩展的待测应用集，其中包括插桩后的原始应用以及对应的变异应用，用户只需提供应用源码方可对应用集进行扩充。通过实现相应接口，系统即可在真机集群上执行待测工具。系统在测试执行期间对自动化测试工具及设备进行监控，同时会收集应用运行时生成的文件、设备日志和被测工具的日志信息。这些数据会存储为原始数据，作为后续评分模块的输入。根据本文提出的多维评估指标公式，对原始应用和变异应用数据进行分析，可以得到测试工具的具体得分。系统后端使用 Spring Boot 框架完成，通过 ADB 对待测设备进行连接，前端使用 React 实现，通过 Echarts 框架将结果以图形化的方式展示。用户可以从图像中简明直观地了解测试结果。

本文选取了 17 个开源应用进行自动插桩变异操作，除系统环境因素导致 1 个应用编译失败外，其它应用均成功执行该操作。包括变异应用在内，共形成了 1089 个应用的待测应用集合。系统已对 5 个知名自动化测试工具进行适配。本文使用本系统对以上工具进行测试，测试在 6 个设备上共执行了 5040 次自动化测试工具的测试任务。在测试过程中系统可对设备以及测试工具进行有效管理，测试数据收集、日志收集、监控等功能表现正常。测试后使用系统对收集的数据进行分析，成功使用本文提出的多维评估指标对自动化测试工具进行评估，评估结果清晰展示了待测工具在五个维度的测试表现以及加权所得的综合得分。

关键词：安卓测试，自动化测试，变异测试

南京大学研究生毕业论文英文摘要首页用纸

THESIS: The Design and Implementation of Multidimensional Evaluation System for Android Automated Testing Tools

SPECIALIZATION: Software Engineering

POSTGRADUATE: Peng Cheng

MENTOR: Professor **Zhenyu Chen**
Assistant Research Fellow **Chunrong Fang**

Abstract

With the rapid development of mobile Internet, Android devices have been widely used. However, due to the open source nature of Android, there are great differences between Android devices, which makes the fragmentation of the Android ecosystem become a growing problem. Therefore, testing is becoming more and more important for Android applications. The fierce competition between applications has led to ever-shortening development cycles. It also makes users demand more and more of the application. Manual testing cannot meet the current development needs due to the high time cost. Therefore, the automatic test tool industry is thriving. Although most tools claim to be more effective, their testing effects do not consider the evaluation metrics, application sets, and equipment comprehensively enough, causing the results are not objective enough. Therefore, this thesis proposes a more comprehensive and objective measurement system for automatic testing tools. This system is conducive to the horizontal comparison of tools, and provides a reference for the development of subsequent automatic testing tools.

In this thesis, we investigate the existing automatic testing tools, abstracts five multidimensional evaluation metrics, including exceptions, mutation, code coverage, UI component coverage and fragmentation support. The thesis designs detection methods for the above indicators. The problem of incomplete evaluation metrics is solved. The system has a scalable set of applications to be tested, which includes the original application after instrument and the corresponding mutated applications. The user only needs to provide the application source code to expand the application set. By implementing the corresponding interfaces, the system can execute the tools to be tested on

the real cluster. The system monitors the automatic test tools and devices during the test execution, collects the files generated during application runtime, device logs and log information of the tested tool. These data are stored as raw data and as input to the subsequent analysis module. According to the multidimensional evaluation index formula proposed in this thesis, the raw application and mutated application data are analyzed, the specific scores of the tested tools can be obtained. The back-end of the system is completed using the Spring Boot framework. and connect to the device to be tested through ADB. The front-end is implemented using React. The results are displayed graphically through the Echarts framework. Users can understand the test results from the images in a simple and intuitive way.

In this thesis, seventeen open source applications are selected for automatic instrument and mutation operation. Except for one application that failed to compile due to system environment factors, all the other applications are successfully executed the operation. Including the mutated applications a total of 1089 applications sets have been formed to be tested. The system has been adapted to five well-known automatic testing tools. In this thesis, we use the system to test all these tools. A total of 5040 test tasks of the automatic test tools were performed on 6 devices. The system can effectively manage the devices and test tools during the testing process. All features including test data collection, log collection, monitoring performed normally. The system was used to analyze the collected data after the tests. The automatic test tool was successfully evaluated using the multidimensional evaluation indexes proposed in this thesis. The evaluation results clearly show the test performance of the automatic test tool in five dimensions and the weighted composite score.

Keywords: Android Testing, Automated Testing, Mutation Testing

目录

表 目 录 ····· **x**

图 目 录 ····· **xii**

第一章 引言 ····· **1**

1.1 项目背景与意义 ····· 1

1.2 国内外研究现状 ····· 2

1.2.1 自动化测试现状 ····· 2

1.2.2 安卓自动化测试工具现状 ····· 2

1.2.3 自动化测试工具的评估现状 ····· 3

1.3 本文主要工作 ····· 4

1.4 本文组织结构 ····· 5

第二章 相关技术综述 ····· **7**

2.1 自动化软件测试与安卓自动化测试 ····· 7

2.2 测试相关技术 ····· 7

2.2.1 ADB ····· 7

2.2.2 Jacoco ····· 8

2.2.3 MDroid+ ····· 11

2.2.4 Gradle Transform ····· 12

2.2.5 ASM ····· 12

2.3 自动化测试工具 ····· 13

2.3.1 采用随机探索策略的自动化测试工具 ····· 13

2.3.2 采用模型探索策略的自动化测试工具 ····· 14

2.3.3 采用系统探索策略的自动化测试工具 ····· 15

2.4 系统相关技术 ····· 15

2.4.1 Spring Boot ····· 15

2.4.2 Docker ····· 15

2.5 本章小结 ····· 17

第三章 安卓自动化测试工具评估系统的需求分析与设计 ····· 19
3.1 系统概述 ····· 19
3.2 系统需求分析 ····· 20
3.2.1 功能性需求分析 ····· 20
3.2.2 非功能性需求分析 ····· 23
3.2.3 系统用例设计 ····· 23
3.2.4 用例描述 ····· 24
3.3 系统总体设计 ····· 30
3.3.1 系统架构设计 ····· 30
3.3.2 系统 4+1 视图 ····· 31
3.3.3 插桩流程设计 ····· 35
3.3.4 覆盖率检测流程设计 ····· 37
3.3.5 测试流程设计 ····· 38
3.3.6 评估指标选取与设计 ····· 39
3.4 本章小结 ····· 41

第四章 安卓自动化测试工具评估系统的详细设计与实现 ····· 43
4.1 UI 控件覆盖检测插件详细设计与实现 ····· 43
4.1.1 UI 控件覆盖检测插件概述 ····· 43
4.1.2 UI 控件覆盖检测插件设计实现 ····· 43
4.2 源码插桩变异模块 ····· 47
4.2.1 源码插桩变异模块概述 ····· 47
4.2.2 源码插桩变异模块的设计实现 ····· 47
4.3 测试模块 ····· 54
4.3.1 测试模块概述 ····· 54
4.3.2 测试模块的设计实现 ····· 55
4.4 测试结果评估模块 ····· 63
4.4.1 测试结果评估模块概述 ····· 63
4.4.2 测试结果评估模块的设计实现 ····· 63
4.5 本章小结 ····· 72

第五章 安卓自动化测试工具评估系统的测试与案例分析 ····· **73**
5.1 系统测试 ····· 73
5.1.1 测试目标 ····· 73
5.1.2 测试环境 ····· 73
5.1.3 功能测试 ····· 74
5.2 案例分析 ····· 77
5.3 本章小结 ····· 85

第六章 总结与展望 ····· **87**
6.1 总结 ····· 87
6.2 展望 ····· 88

参考文献 ····· **89**

致谢 ····· **95**

表 目 录

3.1 插桩相关的功能性需求列表 ····· 21
3.2 变异相关的功能性需求列表 ····· 21
3.3 测试相关的功能性需求列表 ····· 22
3.4 结果分析相关的功能性需求列表 ····· 22
3.5 评估结果可视化相关的功能性需求列表 ····· 23
3.6 非功能性需求列表 ····· 23
3.7 新建代码插桩应用用例描述 ····· 25
3.8 创建代码插桩任务用例描述 ····· 25
3.9 创建代码变异任务用例描述 ····· 26
3.10 审核处理后应用用例描述 ····· 27
3.11 创建工具测试任务用例描述 ····· 27
3.12 评估测试结果用例描述 ····· 28
3.13 查看测试结果可视化结果用例描述 ····· 29
3.14 查看测试工具的历史测试数据总览用例描述 ····· 29

4.1 UI 控件覆盖检测插件主要类 ····· 45
4.2 插桩变异模块主要类 ····· 50
4.3 AppSourceCode 类详情列表 ····· 51
4.4 AppPackage 类详情列表 ····· 51
4.5 MutationAppPackage 类详情列表 ····· 52
4.6 测试模块主要类 ····· 58
4.7 Device 类详情列表 ····· 59
4.8 TestThread 类详情列表 ····· 59
4.9 测试结果评估模块主要类 ····· 65
4.10 DeviceTestApkResult 类详情列表 ····· 66
4.11 TestApkResult 类详情列表 ····· 67
4.12 TestAppResult 类详情列表 ····· 67

5.1 系统测试环境 ····· 73
5.2 完成应用插桩和变异的测试用例 ····· 74
5.3 审核插桩变异的应用的测试用例 ····· 74
5.4 执行测试任务的测试用例 ····· 75
5.5 执行测试任务结果生成的测试用例 ····· 76
5.6 查看测试报告的测试用例 ····· 76
5.7 应用详情信息 ····· 77
5.8 失败详情信息 ····· 78
5.9 测试选用设备详情 ····· 79
5.10 未接入的自动化测试工具 ····· 80
5.11 系统适配的自动化测试工具 ····· 80
5.12 案例测试结果 ····· 83

插图

2.1 ADB 架构图 ····· 8
2.2 插桩分类 ····· 9
2.3 Jacoco 测试报告 ····· 10
2.4 安卓打包部分流程 ····· 12
2.5 ASM 字节码修改 ····· 13
2.6 Docker ····· 16

3.1 系统模块图 ····· 20
3.2 系统用例图 ····· 24
3.3 系统架构图 ····· 30
3.4 逻辑视图 ····· 32
3.5 开发视图 ····· 33
3.6 进程视图 ····· 34
3.7 部署试图 ····· 35
3.8 插桩流程图 ····· 36
3.9 编译埋点流程图 ····· 37
3.10 测试执行控件收集流程图 ····· 38
3.11 测试流程图 ····· 38

4.1 应用插桩时序图 ····· 44
4.2 UI 控件覆盖检测插件关键类图 ····· 45
4.3 FeatSDKAPI 关键代码 ····· 46
4.4 插件接入时序图 ····· 48
4.5 变异时序图 ····· 49
4.6 源码插桩变异模块关键类图 ····· 52
4.7 instrument 关键代码 ····· 53
4.8 makeMutation 关键代码 ····· 54
4.9 分配测试任务的时序图 ····· 56

4.10 运行测试任务的时序图 ····· 57
4.11 测试模块关键类图 ····· 60
4.12 TestThread 关键代码 ····· 61
4.13 TestToolService 代码 ····· 62
4.14 获取测试结果的时序图 ····· 64
4.15 测试结果评估模块关键类图 ····· 68
4.16 多应用 generatePoint 关键代码 ····· 70
4.17 单应用 generatePoint 关键代码 ····· 71
4.18 TestApkResult 关键代码 ····· 72

5.1 插桩变异应用列表 ····· 78
5.2 OSMBugs 变异应用展示列表 ····· 79
5.3 启动测试任务界面 ····· 81
5.4 任务相关信息 ····· 82
5.5 任务测试结果 ····· 82
5.6 结果箱型图 ····· 84

第一章　引言

1.1　项目背景与意义

近年以来，互联网的普及程度越来越高，当代人对互联网的需求程度也大幅增加，这主要得益于移动设备的广泛普及，以及移动应用对人们生活便利性的帮助。移动设备已经成为大部分人接触网络的最重要的渠道。在现代的生活中，不管是娱乐还是工作甚至疫情防控都离不开移动互联网。截至 2020 年 6 月，我国国民使用手机上网的比例高达 99.2 % [1]，专业数据公司 IDC 的数据表明，2020 年安卓机的市场占有率达到了 84.8 %，预计未来 5 年的出货量将保持 1.2 % 的增长率，在 2024 年出货量将达到 13 亿部。

面对如此巨大的用户群体，2020 年 Google Play 商店的应用数量到达了三百万之多，而低质量的应用占比 14 %，巨量的市场以及安卓系统的开放性也导致了安卓设备的碎片化问题。碎片化为安卓应用开发造成了极大的隐性开销，巨量的安卓市场也使安卓应用的开发竞争愈发严重，而软件兼容程度的好坏直接影响了应用的竞争力。所以测试在安卓应用的开发之中显得尤为重要。但是在较短的开发周期内使移动应用满足质量要求并不是一件容易的事情。开发人员和测试人员需要以多种方式全面的验证，才能够保证程序的最佳特性。移动应用的特性导致需要进行大量的适配测试以及真机测试，而大部分的测试都是重复且耗时的。以上原因导致手工测试成本很高。测试开销过高导致很大一部分应用很少进行测试甚至于放弃测试。为了减少时间成本并且让人们不再执行重复的测试任务，专家们将目光转向自动测试技术。

目前测试技术主要分为三个方面，单元测试、集成测试以及界面测试。受碎片化影响最大的部分就是界面测试。当前自动化测试工具主要目标就是解决界面测试的问题，所以理想的自动化测试工具可以有效的提高测试的质量与效率。通过自动化测试，可以有效降低测试的人力成本，同时可以去除测试人员对测试流程的主观影响。但自动化测试也面临一些问题，一方面是目前提出的自动化测试工具通常是在单设备或者仿真器上进行评估的，并没有尝试在真正的移动应用集群上进行过测试，而且测试的应用数量也不够充分，有些只使用了少量现实环境中的应用进行实测。这导致了评估结果并不够全面。另一个方面是缺乏统一客观的评测环境，大多数新提出的技术都生成其在覆盖率和故障检测方面有所改进，但是 Choudhary [2] 等人发现，根据其提出的评估标准，大部分提出的自动化测试技术的效果可能比采用随机策略的 Monkey 还要差。

为了使自动化测试技术的评估结果更加全面且更加具有可参照性，安卓自动化测试工具多维评估系统应运而生。本文结合当前自动化测试工具面临的问题，在开发者关注的方面综合提炼出多个评估指标，并建立了多维评测体系。可以对自动化测试工具的测试效果进行多维度评估，与当前的评估标准相比更为全面客观。本文中的系统选取了 F-Droid 中不同品类中的用户量较大的开源应用作为待测应用集，同时选取不同品牌不同 SDK 版本的多台机型对其进行测试。从检测变异的能力、UI 控件覆盖率、检测异常能力、代码覆盖率、对碎片化的支持程度等方面获取原始分。通过对多种方面的综合评估，将不同方面的原始得分加权得到最终得分。最后以可视化图表配合数据的方式将测试结果展现出来。

1.2　国内外研究现状

1.2.1　自动化测试现状

自动化测试是当前行业内测试的主流方式，通过自动化测试可以减少大量的重复工作，降低测试的时间开销。自动化测试对当今流行的持续交付与持续集成的测试而言十分关键。而且自动化测试不会被测试人员的主观因素干扰，可以有效避免由于测试人员惯性思维导致的测试错误。此外，由于安卓的开源特性，导致安卓的碎片化问题严重。应用之间竞争激烈，使得应用开发周期不断缩短。海量的安卓应用、激烈的竞争、应用的快速迭代以及严重的碎片化问题导致测试问题越发严峻，自动化测试对安卓开发尤为重要。所以当前国内外的安卓自动化测试相关研究发展迅速，大量安卓自动化测试工具应运而生 [3]。

1.2.2　安卓自动化测试工具现状

Choudhary 等人在 2015 年对安卓自动化测试工具的现状进行总结归纳。将自动化测试工具根据测试使用的策略抽象为三类，分别为：采用随机探索策略、采用模型探索策略以及采用系统探索策略的自动化测试工具。

当前安卓自动化测试工具有了更进一步的发展，主要体现在两个方面。一是 Cloudhary 论文中提到的部分测试效果优异的测试工具在这几年仍在不断迭代，如采用随机策略的 Monkey [4] 等，该工具生成通过随机生成事件来对应用程序进行黑盒测试，常被用于压力测试。二是许多优异的新型自动化测试工具不断涌现，如 TesterHome 的黄延胜提出的 App Crawler [5]，该工具采用模型探索策略，采用 Web 爬虫类似的思想。又如字节跳动提出的利用强化学习对应用进行自动化测试的 Fastbot [6]。Fastbot 是基于模型的测试工具，通过对 GUI 进行建模，同时结合机器学习与强化学习技术用以增强对模型的探索。该工具采

用 n 步资格迹强化学习，首先在 APP 遍历时构建状态转义模型，在遍历的同时更新 n 步价值收益，最后使用 $Epsilon-Greedy$ [7] 算法来进行决策。

1.2.3　自动化测试工具的评估现状

Choudhary 等人对当时的自动化测试工具进行了研究 [2]，他们从易用性、对碎片化的支持程度、代码覆盖率和异常检测四个方面对当时的自动化测试工具进行评估与对比。当前自动化测试工具对其自动测试效果的评估大都使用以上全部或部分评估指标。但是在当前环境下，其研究存在一些可以改进的方面：

论文中只给出了当时部分测试工具的测试效果，随着近几年自动化测试工具产业的蓬勃发展，部分自动化测试工具有了技术更为先进的替代品，与此同时自动化测试工具本身也在不断迭代，当时的测试结果并不能体现当前的效果。所以需要一个可以实时对自动化测试工具测试的系统，以应对快速迭代的安卓环境，本系统可以有效解决该问题。

尽管 Choudhary 等人当时选择了 60 多个应用进行测试，但是由于安卓应用迭代较快，其所选择应用集的测试结果时效性较差，对当前环境参考价值较小。为解决上述问题，本文的系统支持动态扩充待测应用集，用户只需提供源码即可对待测应用集进行扩充，以增强评估结果的实时性。

论文中测试设备采用的是安卓仿真器，当前工具对自身进行评估大都使用少量安卓真机，或直接使用安卓仿真器。测试并没有在一定规模的真机环境下执行，导致测试结果可能与真实环境中的效果不一致。本文的系统运行在真机集群之上，并且可以通过 ADB 与远程真机连接，有效解决了上述问题。

已有的研究对自动化测试工具评估指标的选取不够全面。为了提高对工具检测错误能力以及测试充分性的评估水平，在使用原有的异常得分、代码覆盖率以及对碎片化支持程度的同时，本文引入了变异、UI 控件覆盖率作为扩充的评估指标。自动化测试工具的主要目标即为暴露现有的故障，因此系统采用了原有的异常得分指标。变异测试 [8] 可以通过改变源代码而达到模拟缺陷的目的，系统采用该指标来加强检测测试工具检验缺陷能力。Yue Jia 的研究成果 [9] 表明变异测试可以有效的对测试效果进行衡量，且当前的变异测试发展迅速，变异工具已经十分成熟、已经可以用于工作环境之中。所以本文通过使用变异的方式对测试工具的测试能力进行评估。代码覆盖率是衡量测试充分性的指标。Brader [10] 提出，一个测试覆盖率高的程序，与测试覆盖率低的程序相比，未检测软件 Bug 的机率更低。通过代码覆盖率可以有效反映测试的质量，系统支持对代码覆盖率进行评估。安卓机器的页面由 UI 控件组成，本文受到代码覆盖率思想的启发，开发了 UI 控件覆盖检测的插件。该插件用以记录检测安卓 UI 控件的使用情况，

与代码覆盖率配合，更为全面的对测试工具的充分性进行评估。由于对易用性的衡量较为主观，所以舍弃了该评估指标。

1.3　本文主要工作

本文根据当前安卓自动化测试工具的特征，分析了自动化测试工具目前所面临的瓶颈问题。完成了以下工作：

1. 为自动化测试工具提出了多维评估体系。该评估体系基于自动化测试工具对不同应用在不同设备上的表现，分为以下几个评估指标：代码覆盖率、UI 控件覆盖率、检测异常的能力、杀死变异的能力以及对碎片化设备的支持程度。
2. 设计了可以对应用进行 UI 控件覆盖率测试的安卓插件。通过对字节码修改框架的调研选择，分析了不同字节码框架的可用性和易用性，系统选择 ASM 作为字节码修改框架。通过对字节码进行修改实现对点击事件的插桩，这样在自动化测试工具进行点击操作时可以记录其使用的 UI 控件，之后系统可以根据运行时的日志分析统计其 UI 控件的覆盖状况。
3. 设计并实现了自动化测试工具的测试模块，通过对自动化测试工具接口的适配，该模块可以将不同的自动化测试工具接入到系统之中。自动化测试工具在接入系统之后，系统将提供对自动测试任务监控、收集自动化测试工具所测试的设备日志、收集自动化测试工具执行日志、收集自动化测试工具运行时生成的文件等功能。根据对以上日志以及文件的分析即可获得不同方面的原始分。系统目前适配了五个自动化测试工具，经测试均可正常运行于本系统之中。
4. 系统实现了对自动化测试工具测试效果的评估模块。通过对测试模式接口的适配，可以获得不同侧重的测试结果评估。将测试模块收集的日志以及运行时文件作为该模块的输入，该模块可以获得以上几个评估指标的原始分。通过 Jacoco 的报告，对应用的多项覆盖率进行加权可以得到代码覆盖率的原始分。通过 UI 控件覆盖插件的日志，可以获得 UI 控件的原始分。通过对代码杀死异常的数目进行分析，可以获得变异的原始分。通过对日志的分析，可以得到异常的原始分。最后通过对原始分的加权，可以获得自动化测试工具的最终标准得分，该最终得分经过可以用于不同测试工具的横向比较之中。
5. 系统实现了自动化测试工具的可视化模块，可以通过图表的形式更为直观

的将不同测试工具的具体情况展示出来。不仅可以看到在不同安卓平台的适配程度，以及对不同应用的测试结果也均有体现。

1.4　本文组织结构

本文以如下的结构进行对本系统进行介绍，共分为六章，如下所示：

第一章是本文的引言部分，主要介绍了文章的背景以及国内外研究，同时介绍了安卓自动化测试工具多维评估系统的研究意义和目的。

第二章介绍了系统使用的相关技术，首先介绍了安卓应用及设备插桩相关的技术，如 Jacoco、Gradle Transform API、ASM 等。随后简要介绍了自动化测试工具相关的技术与分类同时介绍了系统开发的相关技术，包括 Spring Boot 以及 Docker 等。

第三章介绍了系统的需求分析与设计，其中包括系统需求分析、系统总体设计以及一些核心模块的流程设计。需求分析从功能性需求、非功能性需求、用例等方面进行介绍。系统总体设计从系统的架构图、4+1 视图对系统的整体架构进行描述。最后对一些核心模块的设计进行解释描述，如插桩模块、覆盖率检测模块和测试模块。

第四章介绍了本系统的详细设计与实现，具体介绍了 UI 控件覆盖检测插件、源码插桩变异模块、测试模块、测试结果评估模块。介绍从对模块的概述开始，随后通过时序图、主要类表、类图、详细代码几个方面对模块进行详细阐述。

第五章是系统测试与案例分析。首先根据第三章的需求进行测试用例的设计，对系统进行功能测试。随后对系统的实际测试案例进行分析。

第六章是总结与展望，首先对本文进行概述总结，随后为本系统进一步的改进作出了规划。

第二章　相关技术综述

2.1　自动化软件测试与安卓自动化测试

在软件测试中，测试自动化 [11] 是指使用与被测软件相分离的软件来控制测试的执行，同时将测试运行的实际结果与预测中的结果相比较，以达到测试的目的。测试自动化可以将一些繁琐但又不得不做的工作自动化，也可以执行一些人工难以完成的额外测试。测试自动化对于持续交付而言至关重要。而软件测试对比人工测试的优点是，人工测试的结果会由主观影响，不能保证多次测试的一致性，自动化测试可以排除在测试过程中人为因素导致的错误和遗漏 [12]。但自动化测试也有可变性较弱等缺点。在当前软件开发竞争激烈、测试人员供不应求的环境下，只采用人工测试无法满足软件开发的需求，所以自动化测试迎来了蓬勃发展。

自动化测试在节约了大量人力成本与时间成本的情况下，依然可以保证测试拥有足够的质量。安卓的自动化测试技术主要分为两个方向，分别是自动化测试框架和自动化测试工具。框架需要拥有相关测试经验的专业人员编写脚本，通过调用框架提供的相关接口来模拟人工测试的相关操作。该类脚本可以有效减少以往需要人工的重复操作，可以对脚本所涉及的测试对象进行测试，但对脚本的质量有一定要求。由于安卓碎片化问题严重，脚本要求拥有较好的鲁棒性，以保证脚本在不同设备不同环境下的可复用性。而安卓应用的快速迭代也给脚本的维护造成了不小的压力。自动化测试工具是面向所有应用的，理论上可以对绝大多数的控件进行检测。其优点在于工具不需要对每个应用进行特殊配置，自动化测试工具即可通过其自身策略对应用进行测试。虽然在定制性方面相较自动测试框架有所不足，但是部分自动化测试工具也可以通过为应用编辑配置文件来提升自动化测试工具的测试效果。目前安卓测试有大量研究都与自动化测试工具相关。

2.2　测试相关技术

2.2.1　ADB

ADB 是一款来自安卓 SDK 平台工具包中的通用命令行工具，它的全称是 Android Debug Bridge [13]，通过使用 ADB 可以帮助开发机和设备进行调试、连接等操作。它还可以让你方便的通过执行命令完成与设备进行文件传输、在设

备上安装调试应用程序等操作。ADB 还提供了对设备上的 Unix shell 的访问功能，通过该功能用户可以在设备上运行 Unix 命令。

ADB 采用主从式架构进行实现，把客户端与服务端分开来，每个服务端可以同时直接向多个客户端进行通信。如图 2.1所示 ADB 具体包括三个组件：

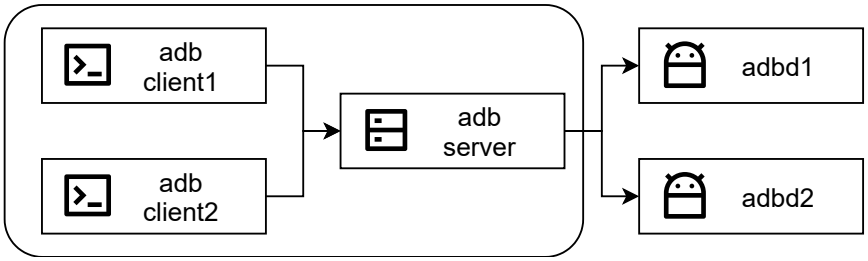

图 2.1: ADB 架构图

守护进程 (adbd)，该进程在每个设备的后台执行，负责与服务器进行连接，同时可以在设备上执行客户端发起的 shell 命令。

客户端，运行在开发机器上，可以通过发出 ADB 命令从命令行来调用客户端。客户端也可以通过服务端进入设备的 Unix shell，直接在设备上通过守护进程执行 shell 命令。

服务端提供客户端和守护进程之间通信的服务，它在开发机器的后台运行。服务端使用建立 TCP 连接的方式与设备相连接。通过 TCP 连接可以有效支持远程调试。本文部分真机使用 TCP 的方式与远端设备建立连接，使用云上真机进行测试。

2.2.2　Jacoco

代码覆盖率是一种度量方式，在本系统中用于测量在自动化测试期间执行了多少行代码。Jacoco [14] 是为 Java 代码测量使用覆盖率且提供相关覆盖率报告的开源工具包。它的全称是 Java Code Coverage Library。Jacoco 作为 Emma 的替代品于 2009 年推出，由 EclEmma 团队根据多年来使用经验并与现有工具对比后创建，该工具采用 Java 语言实现。

Jacoco 与同类字节码插桩框架（如 Emma [15]、Cobertura [16]）相比，在本系统的使用情境下有以下几点优点：

(1) 易用性较好。Gradle 中直接嵌入了对 Jacoco 的支持，对于安卓应用而言十分方便。只需在配置文件中启用插件即可。

(2) 兼容性较好。因为业界很大一部分应用使用的 JDK 版本为 8，而 Emma 并不支持 JDK8，导致可用性不高。

(3) 性能较好。在于 Emma、Cobertura 的对比中，性能最好。安卓编译本身对性能要求较高，因为系统待测应用集的准备过程涉及到大量的编译操作，性能差别对实验准备过程造成较大影响。

(4) 覆盖粒度完整。Jacoco 有着最为完善的覆盖粒度，可以对方法、类、行、分支、指令、圈复杂度进行分别统计。方便测试后的数据统计。

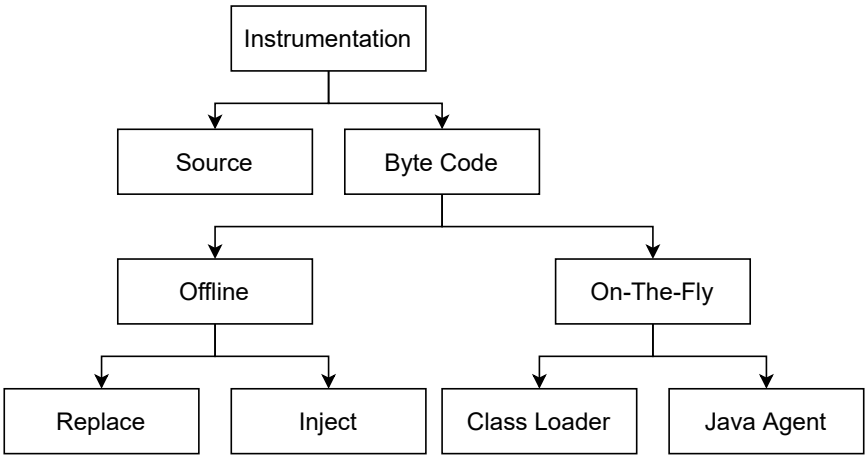

图 2.2: 插桩分类

Java 的插桩方式如图 2.2所示，可以选择对源码进行插桩或者对 Java 字节码进行修改。Jacoco 底层依赖 ASM 框架进行实现，ASM 采用修改字节码的方式对源代码进行插桩，已达到分析覆盖率的目的。后文将对 ASM 框架进行详细介绍。其中字节码插桩的方式也有所不同，Jacoco 支持 On-The-Fly 和 Offline 两种插桩方式。On-The-Fly：通过对 JVM 的参数进行设置以启动 Java 的代理。这种代理可以在类加载期间在内存中对类文件进行预处理，在 Class Loader 装载类之前对类进行动态修改。即使用该方式测试覆盖率可以在程序运行时完成。不需要预装插件，也不需要对源代码进行调整。Offline：在测试之前，即在源码编译之前就对程序进行插桩。随后编译直接生成插桩后的 class 文件和 jar 包。在进行测试之后会根据程序的执行情况生成相应的 coverage.ec 文件，配合源代码编译后的 class 文件即可获得相应的覆盖率数据。虽然 Offline 模式看起来有很多不足，但由于安卓程序运行在与 Java 不同的 Dalvik 虚拟机上，并不支持动态的 Java 代理模式，所以本文采用 Offline 模式对安卓代码进行插桩。

经过测试之后就可得到形如图 2.3的覆盖率报告。测试报告中含有以下几项：

JaCoCo　　Sessions

JaCoCo

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
org.jacoco.examples		58%		64%	24	53	97	193	19	38	6	12
org.jacoco.core		97%		93%	111	1,384	116	3,320	20	710	2	136
org.jacoco.agent.rt		77%		84%	31	121	62	310	21	74	7	20
jacoco-maven-plugin		90%		80%	37	186	46	412	8	111	0	19
org.jacoco.cli		97%		100%	4	109	10	275	4	74	0	20
org.jacoco.report		99%		99%	4	572	2	1,345	1	371	0	64
org.jacoco.ant		98%		99%	4	163	8	429	3	111	0	19
org.jacoco.agent		86%		75%	2	10	3	27	0	6	0	1
Total	1,358 of 27,250	95%	150 of 2,141	92%	217	2,598	344	6,311	76	1,495	15	291

Code Coverage Report for JaCoCo 0.8.7-SNAPSHOT　　Created with JaCoCo 0.8.7.202101150304

图 2.3: Jacoco 测试报告

(1) 指令覆盖率：是 Jacoco 的最小计量单位。该覆盖率基于单一 Java 字节码指令，可以体现字节码指令的执行状况。这个指标完全独立于源码的格式，且在任何情况下均有效。指标不需要提供类文件的调试信息即可获得。

(2) 分支覆盖率：统计所有的 if 和 switch 分支的数量，并同时计算这些分支的使用情况。此指标在任何情况下均有效，不需要提供类文件的调试信息。

(3) 圈复杂度：Jacoco 计算了每个非抽象方法的循环复杂度，同时也总结计算了不同类、包以及组的复杂度。圈复杂度的定义由 McCabe 在 1976 年提出 [17]，指线性无关的总路径数量，即为合理检测错误所需要的路径条数的最小值。圈复杂度越大，测试完全的难度越高。同样在类文件没有调试信息的情况下可以被计算出来。

(4) 行覆盖率：对于已经编译过调试信息的类文件，可以计算出每行代码的覆盖率信息。一行通常可以转换为多条指令，有一条指令执行就视为在源代码中此行已被执行。

(5) 方法覆盖率：每个抽象方法映射到字节码文件之中，至少拥有一条指令。当属于该方法的任一指令被执行时，这个方法就可以认为被测试覆盖。由于 Jacoco 工作在字节码层面，所以构造函数和静态初始函数也被算作方法。所以有一些方法覆盖率，在源码中并没有以一一对应的关系展现出来。

(6) 类覆盖率：在一个类中有任一方法在测试过程中执行，就认为该类执行。

本文通过对 Jacoco 测试报告分析加权得到测试工具的覆盖率的原始分。

2.2.3 MDroid+

这个项目是由威廉玛丽学院的 SEMERU 与安第斯大学、桑尼奥大学以及卢加诺大学于 2018 年共同合作完成的。此项目的目的是为了帮助提高移动应用开发者和测试人员测试的质量。MDroid+ [18] 根据研究过去的测试经验得出了一组安卓特定的变异因子，共 38 个。可以自动生成变异体到目标应用中，并用于分析测试用例。

变异测试，有时也被称为变异分析。变异通过改变源代码的细节，产生一个与先前类似且满足语法要求的语句，以达到模拟缺陷的目的。条件变异 [19] 是常见的变异操作之一，比如将 a && b 转换成为 $a \parallel b$。由于源代码的改变，会造成一些异常的发生。在执行先前预设的测试用例时，变异应用有机率会出现与未变异应用不同的测试结果。如果测试单元执行失败则代表变异被杀死。变异即可以帮助测试者定位测试数据的缺陷，也可以作为评估不同测试套件测试能力的指标之一。因为通过对其杀死变异能力的分析，可以体现出测试套件的健壮性。MDroid+ 在与一些支持 Java 的变异工具（Major [20] 和 PIT [21]）和专用于安卓的变异工具（muDroid [22]）相比有以下优点：

(1) 更好的适用性。与 Java 变异工具相比此工具添加了对安卓的特定支持。比如 MDroid+ 可以对安卓表单文件生成特定变异，以获得更全面的变异体。由 Java 编写，多平台可用。

(2) 更好的可用性。与 muDroid 相比，可编译成功的变异占比更高。且工具支持多线程产生变异，大幅减少了生成变异的时间开销。在服务级的硬件上，MDroid+ 每个变异生成的时间平均只需要 19 秒。

(3) 更好的变异质量。在与三种工具的对比中，MDroid+ 生成的不可执行的变异的占比最低，而且使用的变异因子更为具体且有效。变异因子耦合性较高使生成的变异总量最少但是成功率最高，相较于其他变异工具更适用于自动化测试工具的评估，因为自动化测试工具作为界面测试工具所需要的时间开销较大，即使可以产生较多变异，但可测试的数目依旧有限。在进行有限次数测试的情况下，选用此工具可以获得覆盖率更大的测试数据。且安卓应用的编译开销是很大的，更高的成功率对安卓测试来说十分必要。

本文使用此工具对 F-Droid 上所选开源应用集源码进行变异。通过分析检测杀死变异的比例获得变异的原始分，作为多维测试中较为重要的评估项。

2.2.4 Gradle Transform

Gradle [23] 是一个支持多语言开发的自动化工具，该工具可以完成编译、打包、测试、部署、发布等一系列的开发流程。此框架对 Java（Kotlin、Groovy、Scala），C/C++，JavaScript 语言提供支持。

从 Gradle 的 1.5.0-beta1 版本开始，Gradle 插件中包含了一个 Transform API。在 class 文件转换为 dex 文件之前，第三方插件通过调用此 API 可以对 class 文件进行修改。

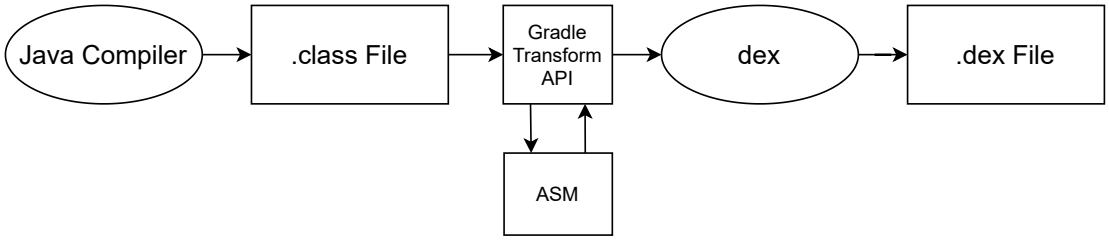

图 2.4: 安卓打包部分流程

如图 2.4是与 Gradle Transform API 相关的安卓打包的部分流程。首先编译器将 Java 或 Kotlin 文件编译为 class 文件，随后编译器将源代码转换为 dex 文件，此文件中包含在安卓设备上可运行的字节码。Transform API 就作用在这个过程。该 API 由安卓官方提供，可在项目构建阶段修改 class 文件。它的主要使用场景是字节码插桩或代码注入等功能的实现。在工业界中，神策 SDK 等数据分析的无痕埋点功能都是借助此功能来进行的。因为使用 Transform API 可以接触到所有的 class 以及 jar 文件，再借助 ASM 此类字节码修改框架就可以对代码进行非侵入式的埋点 [24]，后文将详细介绍其具体实现。

2.2.5 ASM

ASM [25] 是一个具有多种用途的 Java 字节码操作和分析框架。这个项目最初由 Eric Bruneton 构思和开发，目前由 OW2 基金会资助。它可以用来修改现有的类和方法，或者直接以二进制形式动态生成类。

Java 字节码是用于 Java 虚拟机的指令集，由 Java 编译器编译而来。因编译器会对代码进行优化，开发者可以通过字节码了解虚拟机中具体执行的指令。

ASM 提供了一些常见的字节码转换和分析算法，通过调用该框架提供的 API，用户也可以建立自定义的代码转换和代码分析工具。ASM 的功能与其他 Java 字节码框架类似，但 ASM 的性能表现更为优越。因为它的设计和实现理念

是尽可能的小而快，所以很适合在动态系统中使用（当然也可以已静态的方式使用，例如在编译器中使用）。通过 ASM 框架，开发者可以在运行时对 JVM 中的类进行修改并将其重载。可用于项目的热部署，也可用于对项目进行无侵入的监控。

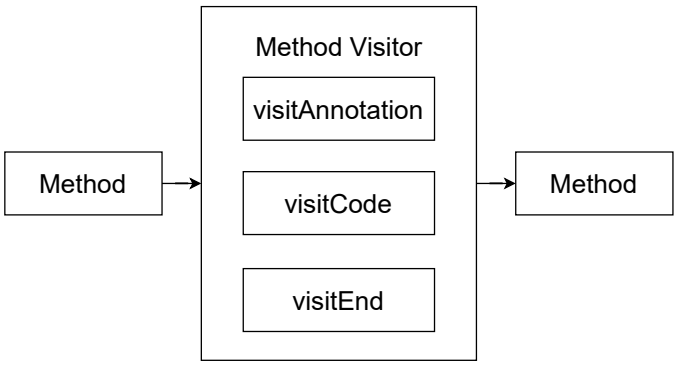

图 2.5: ASM 字节码修改

ASM 框架与 AspectJ [26]、APT（Annotation Processing Tool）等框架相比，它的语法较为简单，且兼容性较强。对第三方的类库以及 Lambda 语法的支持较好，性能较强 [27]。ASM 对方法修改的运行原理如图 2.5所示。通过 Method Visitor 访问方法的字节码，其中封装了对方法不同位置的访问方法。简而言之 ASM 使用访问者模式对字节码进行处理，使处理后的字节码可以获得用户赋予的新功能，但不需要修改源代码。系统在安卓应用执行的过程中，使用 ASM 框架对点击操作进行插桩，以获得足够的元数据信息。通过对信息进行分析，可以确定自动测试所工具点击的 UI 控件，在生成 UI 控件唯一标识后对其进行记录。

2.3　自动化测试工具

根据 Choudhary 的研究结果，可以根据测试策略的不同将常见的自动化测试工具分为三类。分别是采用随机探索策略的自动化测试工具、采用模型探索策略的自动化测试工具和采用系统探索策略的自动化测试工具。下文将详细介绍三类测试工具的优缺点和该类较为典型的测试工具。

2.3.1　采用随机探索策略的自动化测试工具

随机探索策略就是测试工具采用随机的方式为安卓应用程序生成 UI 事件。由于采用随机的方式，系统不需要进行额外的分析计算工作，程序可以高效的产生 UI 事件，也使得此类测试工具很适合于应用程序的压力测试。随机策略通

常可以探索出一些正常测试不易察觉的错误，而且可用性较高，在三种策略中采用此策略的自动化测试工具执行的成功率最高。但是随机生成系统事件的效率很低，应用程序往往对很多事件不能进行反应，即使可以对操作进行反应，也有很多操作是冗余的。不能够根据程序的当前情况进行分析，导致了随机事件很难进入较深的入口。而且也没有探索成功的停止标准，一般只能够通过限制随机事件的生成次数或者超时事件来停止测试。

Monkey 是采用随机探索策略的自动化测试工具的典型代表。它是由 Google 官方开发的运行在设备上的一个命令行工具。它可以模拟生成用户（点击、触摸或手势）的伪随机事件流，同时可以生成一些系统级的事件。Monkey 可以限定在指定的包中运行，并在程序崩溃、产生未处理异常以及程序没有响应的时候报告错误。Monkey 可以配置不同的日志等级，通过日志可以用来实现 Bug 复现等操作。本文对选用 Monkey 作为随机探索策略的代表对其进行测试。

2.3.2　采用模型探索策略的自动化测试工具

模型探索策略的自动化测试工具的思想来源于 Web 爬虫和 GUI 测试工具，这种工具使用待测应用程序的 GUI 模型 [28, 29] 来生成事件，系统探索应用程序的各种行为。这些模型通常是有限状态机，将页面作为状态，将事件作为转移动作。工具动态的为待测应用产生精确的模型，在不能发现新的状态时中止测试。采用此策略的优势是在页面覆盖率的表现上会较为优异。因为可以有效的中止测试加上可以判断当前的 GUI 模型状态，所以冗余测试较少，测试的效率较高。理论上可以进入较深的 UI 入口。但是此策略容易忽略应用程序的内部状态，在 GUI 模型相同的情况下，程序内部的区别可以导致行为的不同。但是此种测试工具一般难以检测到这种场景。

采用此策略的工具主要有黄延胜开发的 App Crawler、以及 Google Jetpack 套件的 App Crawler [30] 等，字节跳动的 Fastbot 以及 Zhao Zhang 提出的 Maxim 也部分使用了此策略。此种测试工具的实现思想大致相同。以思寒的 App Crawler 为例，它是基于 Appium 实现的一个测试框架，默认通过 Activity 或者 Navigatorbar 来定义唯一界面，同时通过 resource-id、content-desc、text 和 index 属性来确定在页面中的唯一控件。AppCrawler 工具在支持自动爬取的同时还支持丰富的自定义配置，包括界面的定义条件和控件的定义条件都可以手动进行配置。

本文选用以上提到的采用模型探索策略的自动化测试工具作为待测工具进行实验，对本文提出的多维评估系统进行可用性验证。

2.3.3 采用系统探索策略的自动化测试工具

有些应用程序拥有特定触发条件，只有在提供特定的输入时才可以显现出来。所以有一部分安卓测试工具使用符号执行以及遗传算法，用以探索之前测试未能覆盖的代码。

测试工具通过使用此种算法可以探索到采用随机策略难以探索的目标，Evo-Droid [31] 的研究中提到了这一点，此工具采用启发式规则进化，以最大化代码的覆盖率。但是由于此类工具的复杂度较高，导致工具的伸缩性较差，目前在工业界中还不能够成熟应用。在 Choudhary 的综述中，他们找到了 4 种测试采用系统探索策略的自动化测试工具，由于采用随机策略对环境要求过高，在日常的测验中均不可用。

2.4 系统相关技术

2.4.1 Spring Boot

Spring Boot 是一个基于 Java 平台的一个开源框架。其最主要特点是通过控制反转对 Java 的对象进行管理，同时可以较为简易的使用面向切面编程等特性。该框架从 2002 年开始立项，经历了十八年的更新，是目前最为稳定、生态环境最为完备的框架之一。Spring Boot 可以较为简单的创建生产级别的应用程序。通过 Maven 或者 Gradle 等包管理工具，可以十分便捷的调用 Spring 官方实现的类库和第三方用户或机构发布的类库。Spring Boot 由 Java 语言实现，在支持 Java 语言的基础上，Spring 5.0 版本引入了对 Kotlin 语言的支持。使用 Kotlin [32] 的语言特性可以有效简化代码的繁琐程度，本系统的部分代码使用了 Kotlin 语言。

系统选用 Spring Boot 作为后端，为前端提供具有较强可用性的 Restful [33] 接口，同时通过使用 Spring Data MongoDB 等库来实现对数据库的数据交互。本系统通过借鉴成熟 Spring Boot 项目的开发经验，对系统进行分层。使系统具有较强的灵活性以及可维护性。同时借助 Spring 接入的 Log4j2 [34] 日志框架来帮助系统进行日志记录以及异常追踪。

2.4.2 Docker

Docker 是一个用于开发、发布以及运行应用程序的开放平台。Docker 在 2011 年开始由 Docker 公司研发，项目由 Go 语言编写。Docker 是支持 Windows、macOS、Linux 操作系统的平台即服务产品。Docker 公司于 2013 年将项目开源。

Docker 使我们可以在虚拟的容器环境里开发以及部署程序，意味着程序可以在不同机器上以同样的形态运行，但不需要额外的配置。Docker 的容器可以部署到几乎任何机器上，它让开发工作只需考虑开发本身，而不需要考虑繁琐的环境以及部署问题。用户可以轻松的管理容器，但不影响主机上操作系统。Docker 十分轻量。虽然来自虚拟化的形式，但是与虚拟机有很大的不同。它底层使用 LXC 实现与主机共享资源 [35]，这是 Linux 一个内核级的功能，可以保证 Linux 程序沙盒化，避免了虚拟机启动和维护的开销，但是又可以保证容器与容器之间以及容器与主机之间的独立，以保证容器的安全 [36]。

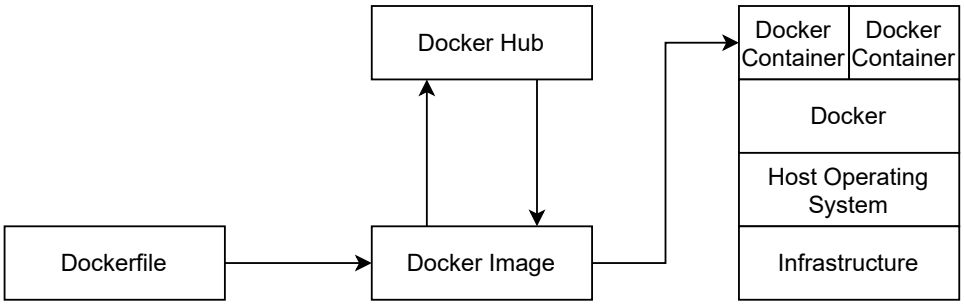

图 2.6: Docker

Docker 开发需要用到图 2.6中的组件，分别是：

(1) Dockerfile 是指导 Docker 构建 Docker 镜像的文本文件，可以通过读取文件中的命令来自动构建镜像，对 Dockerfile 的进行编写就可以达到对镜像进行环境配置的目的。构建过程以层级的方式对操作进行缓存，这样可以保证镜像的快速构建，这也是 Docker 相较虚拟机更为优秀的部分。同时使用 Dockerfile 也可以让 Docker 镜像具有较强的可拓展性，也让模块更新变得更加轻松简便。

(2) Docker Image 即为 Docker 镜像，其中包含了可执行系统、应用、源代码以及相关的依赖。镜像是只读且不可运行的。镜像的创建方法共有两种，一种是使用 Dockerfile 进行创建，另一种是通过对现有的容器执行内部进行操作改动，最后通过提交的方式将改动以类似快照的方式记录下来。

(3) Docker Container 即为 Docker 容器，容器是一个虚拟化的运行时环境。创建容器实际上相当于创建一个 Docker 镜像的读写副本，在 Docker 镜像的基础上添加一个容器层，允许修改且运行整个副本，所以容器不能脱离镜像存在。容器的虚拟化发生在应用程序层，这样使多个容器可以共享同一个内核。Docker 公司称容器拥有业内最为强大的隔离能力。

(4) Docker Hub 是 Docker 公司提供的公共的镜像中心。用户可以通过 docker pull 命令直接从网络上下载镜像文件。通过这种方式，用户可以方便的共享相同的开发环境，也可以保证程序运行在相同的依赖环境之中。现在 Docker Hub 上的镜像已经达到了 520 万之多，通过使用用户共享的镜像可以很大程度上减少开发中的开销。如果出于私密性的考虑，用户也可以自行搭建 Docker Registry。

由于自动化测试工具的环境搭建较为困难，本文选用 Docker 作为服务端的容器，方便部署和后续更新。

2.5　本章小结

本章首先介绍了自动化软件测试与安卓自动化测试；随后主要介绍了系统开发实现过程中使用到的关键技术，从测试相关技术、自动化测试工具以及系统相关技术三个方面对相应技术进行了详细介绍。

第三章　安卓自动化测试工具评估系统的需求分析与设计

3.1　系统概述

随着安卓市场的逐步扩大，软件开发行业竞争越发激烈，对开发的时间要求以及效率要求越来越高，即使是成熟软件也需要通过频繁迭代来优化用户体验。虽然迭代对软件的改动很小，但测试开销与正常改动无异。同时伴随着设备种类的不断增加，碎片化问题越发严重。两方面的问题使手工测试的成本节节攀升，也促进了自动化测试工具的蓬勃发展。

虽然目前有许多类型的测试工具可用，但是缺乏一个客观全面的标准来衡量这些工具，当前的衡量标准在评估指标、评估应用集以及评估设备方面有一定缺陷。本文所阐述的安卓自动化测试工具多维评估系统，通过引入变异、UI控件覆盖率、对碎片化的支持等标准，加之结合代码覆盖率以及异常覆盖率的传统标准，在真机集群上对自动化测试工具进行多维度的评估。所展示的结果直接准确，可以对自动化测试工具进行客观评价。

图 3.1是本系统的模块图，从功能方面分析，本系统的功能模块具体有：源码插桩变异模块、自动化测试模块、测试结果评估以及测试结果可视化模块。以上四个功能分别负责以下工作：使 APP 可以产出可用的测试数据、将测试工具接入到系统中、制定多维评估体系以及根据结果将数据展现为可视化的前端页面等。从具体的执行流程方面分析，此系统的主要执行路径分为三条：

第一条是系统对源代码改动的具体执行路径。系统首先对源代码进行插桩，使得插桩后的应用拥有获取代码覆盖率信息以及 UI 控件覆盖信息的能力。随后对源代码进行变异操作，对每个应用程序生成对应的变异集。最后将以上处理过的源码进行编译，生成可用待测程序集合。

第二条是系统对自动化测试工具进行测试的具体执行路径。系统将事先准备好的应用程序集视为自动化测试工具的待测应用集，随后为工具实现系统的相关接口，即可将工具接入到系统之中。系统可以自动运行测试工具，使其在真机上分别测试变异以及未变异的应用程序。并在测试过程中对真机以及自动化测试工具进行相关的数据收集，同时对自动化测试工具进行监控。

第三条是系统对测试时收集数据进行处理的具体执行路径。系统将测试后待测应用集生成的数据收集处理，根据日志获取异常得分以及 UI 控件覆盖率得分，根据 coverage.ec 以及源码计算生成代码的覆盖率得分，根据变异测试与源

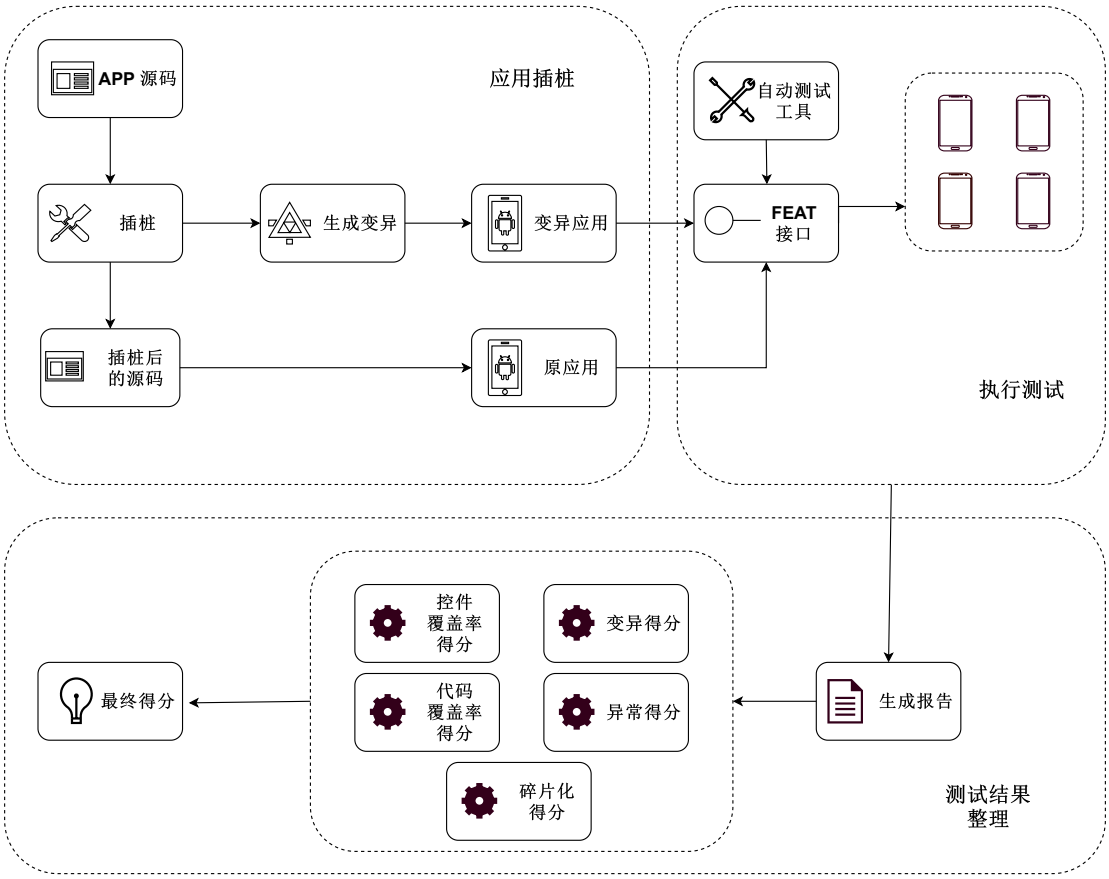

图 3.1: 系统模块图

应用测试数据对比得到变异得分。最后将得分汇总分析之后渲染到前端页面之上，如此可以获得可视化的测试结果。

3.2　系统需求分析

系统需求分析在开发之中十分重要。本节分别从功能性需求分析、非功能性需求分析以及系统用例设计三个方面进行介绍。

3.2.1　功能性需求分析

本小节将对安卓自动化测试工具多维评估系统的功能性需求进行详细描述，并以表格形式进行展现。对于本系统的功能性需求，通过对系统功能进行抽象，可以将需求粗粒度划分为安卓代码插桩、安卓代码变异、自动化测试工具测试、测试效果评估以及评估结果可视化等几个方面，下文中按照以上划分的方式对其进行具体介绍。

安卓代码插桩相关的功能性需求如表3.1所示，由前端提供上传源代码的服务以及插桩后的下载服务，后端负责实现代码的插桩，并在插桩之后进行编译。插桩后的源代码将保存在服务器上，作为后续变异模块的输入数据。

表 3.1: 插桩相关的功能性需求列表

需求编号	需求名称	需求内容	优先级
R1	创建代码插桩任务	用户可以在代码提交之后，选取插桩类型，点击插桩按钮，触发代码的插桩任务	高
R2	下载代码插桩结果	用户可以在插桩之后，点击下载按钮，对插桩后的代码进行下载	低
R3	编译插桩后的代码	用户在插桩结束之后，可以手动开启编译。并在前端页面实时看到应用的编译日志	高

表3.2是安卓代码变异相关的功能性需求。可以在前端开启源代码的变异任务，用户可以选择随机生成变异因子或者手动选择变异因子。后端会将源代码根据所选的变异因子进行变异，随后将变异后的源码进行编译以得到待测变异集合。在管理员人工审核可用性之后，可以将应用添加到待测应用集中，为之后自动化测试工具的测试工作提供应用支持。

表 3.2: 变异相关的功能性需求列表

需求编号	需求名称	需求内容	优先级
R4	选择变异因子	用户可以手工或者随机选择变异因子，用以源码的变异	低
R5	生成代码变异任务	系统内部将插桩后的代码，根据用户提供的变异因子，对源代码进行变异操作	高
R6	编译变异代码任务	系统首先将变异后的源代码拷贝到项目目录，对其进行编译，随后将编译应用的信息存储到数据库中。生成的应用在人工校验之后可以用于后续自动工具的测试	高

自动化测试工具测试相关的需求如表3.3所示，该部分对应系统的测试模块，

测试模块是本工具的核心，其相关的详细设计会在第四章进行讲解。模块的作用在于，使自动化测试工具可以接入系统中，并对所有待测应用进行测试。在测试过程中对设备和测试工具进行监控，使测试可以在多设备上并行执行。该模块负责测试之前的环境配置以及测试之后的环境清理工作，且在测试执行过程中收集应用和工具的日志以及生成的文件。

表 3.3: 测试相关的功能性需求列表

需求编号	需求名称	需求内容	优先级
R7	自动化测试工具测试运行	用户可以手工选择待测应用，触发自动测程，系统会自动对选择的应用集（插桩后的应用以及对应的变异应用）进行自动测试	高
R8	应用数据收集	测试完成后，系统会自动收集系统日志以及结果所需的信息，并将其存储于服务端	高
R9	查看测试任务进度	用户可以通过看到系统的实时日志以及已测应用的占比，用以了解测试的进度	低

结果分析相关的需求如表 3.4所示，该部分对应系统的结果分析模块，该模块也是系统的核心模块之一，模块主要负责将自动化测试工具测试后的数据集解析，随后根据测试应用的不同属性调用相应的评分模块，再对待测集中的应用运行时生成的数据进行整合、处理，根据公式得到原始分数，在得到评分之后将其持久化到系统之中。

表 3.4: 结果分析相关的功能性需求列表

需求编号	需求名称	需求内容	优先级
R10	解析数据集	系统将通过对设备日志以及应用运行时产生的文件进行解析，得到评估结果所需的必要数据	高
R11	评估测试结果	调用系统的评分模块，根据公式得到测试的原始得分	高
R12	保存最终结果	系统将测试结果持久化到数据库中	高

表 3.5: 评估结果可视化相关的功能性需求列表

需求编号	需求名称	需求内容	优先级
R13	查看测试的可视化结果	用户在结果界面中可查看当次测试的可视化结果，包括设备、不同应用等多维度的详细分析信息	高
R14	查看历史评分记录	用户可以查看自动化测试工具的单次历史得分记录	低
R15	查看测试数据总览	用户可以查看不同自动化测试工具评分的总体数据的可视化图表	高

评估结果可视化相关的功能需求如表 3.5所示，主要包括测试结果的可视化，历史评估记录的展现，以及得分的总览。包括不同测试工具在平台层级、应用层级数据汇总以及测试效果的多维分析详情。

3.2.2　非功能性需求分析

本系统的非功能性需求如表 3.6所示，目标是在开发效率以及系统质量方面使系统拥有较好的体验。主要在可扩展性、容错性、可用性三个方面进行描述。

表 3.6: 非功能性需求列表

需求编号	需求名称	需求内容	优先级
R16	可扩展性	系统应在适配新的自动化测试工具的过程中，保证较低的学习成本以及开发成本	高
R17	容错性	应在工具崩溃或者异常输入的情况下，保证系统的正常运行，崩溃需要将测试数据持久化	中
R18	可用性	系统应保证可以较长时间的稳定运行	高

3.2.3　系统用例设计

安卓自动化测试工具多维评估系统的用例图如图 3.2所示，该图展示管理员和用户在系统中的所有用例，管理员拥有用户的所有权限。管理员和用户的相关操作大多与前端交互相关，工具的主要逻辑会在测试工具的后端自动执行。

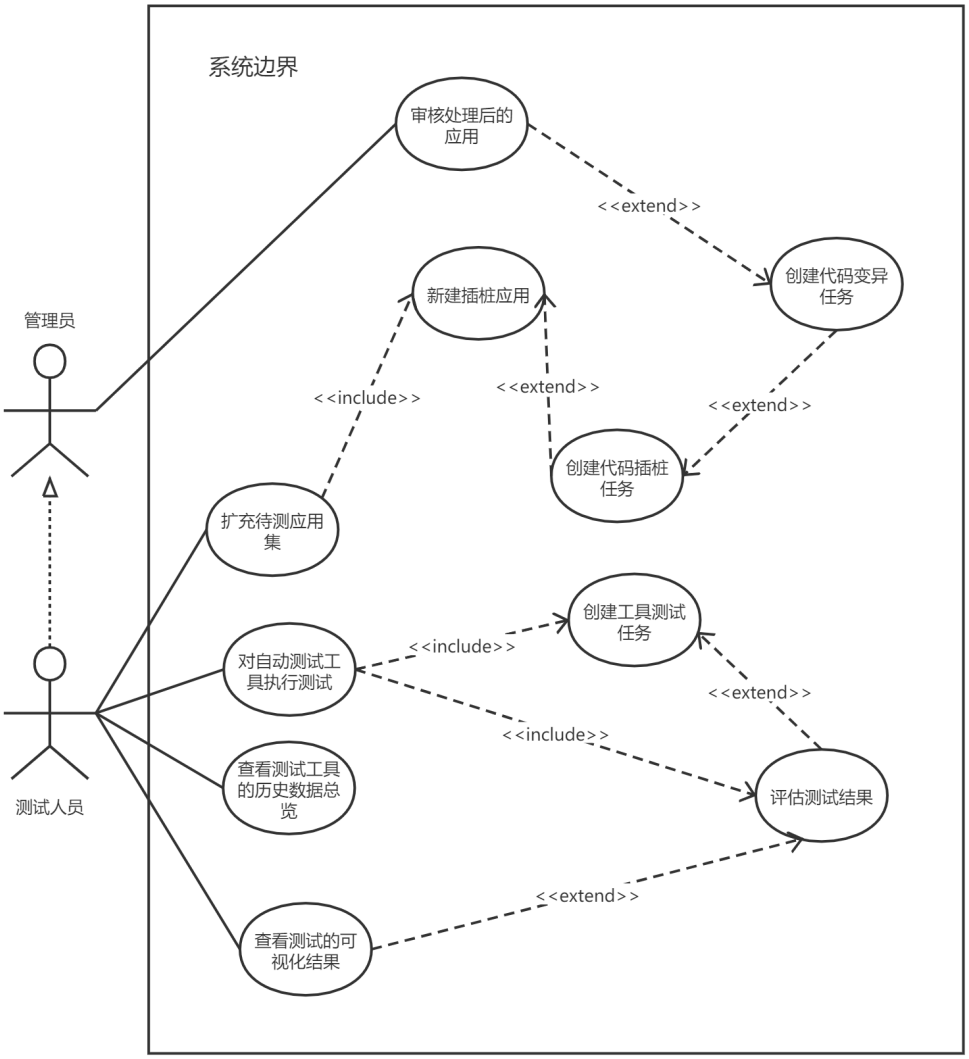

图 3.2: 系统用例图

3.2.4　用例描述

在这一小节将对用例图中提到的系统用例进行详细解释。

系统已经准备了可用的待测应用集，但是随着时间的推移，应用集也有迭代的需求。所以系统需要上传应用以扩充应用集的功能。该用例如表 3.7所示。将源码压缩包上传，且等待系统解压缩之后，即可在系统中对源码进行操作。

在测试人员上传应用的源代码之后，可以选择对应用进行插桩。创建插桩任务的用例描述如表 3.8所示。申请之后系统会对应用进行插桩操作。只需将本系统实现的安卓端插件及 Jacoco 插件通过修改 Gradle 的配置文件接入到安卓应用中，再将与应用 SDK 版本相符合的 SDK 模块拷贝至应用中，即可完成插桩工

作。UI 控件覆盖率插件会在运行时自动对应用的点击操作进行代理，Jacoco 会在编译时完成对代码的插桩。这样可以使移动应用拥有 UI 控件覆盖率检测及代码覆盖率检测的能力。在插桩完成后，应用即可获得进行变异操作的权限。

表 3.7: 新建代码插桩应用用例描述

描述项	**说明**
用例编号	UC1
用例名称	新建待插桩应用
参与者	测试人员
优先级	低
用例描述	测试人员通过系统前端页面新建待插桩应用。
前置条件	测试人员拥有权限，可以对本系统进行相关操作。
主事件流	1. 测试人员点击上传应用按钮。 2. 测试人员选择待插桩的应用源码压缩包，上传应用源码。 3. 系统显示上传成功。 4. 系统自动解压。 5. 系统显示解压成功。
后置条件	前端出现上传成功的提示。用户可以在插桩列表中看到上传成功的应用，在系统中可以读取该应用。
扩展事件流	4a. 系统在后台解压失败，显示解压失败。

表 3.8: 创建代码插桩任务用例描述

描述项	**说明**
用例编号	UC2
用例名称	创建代码插桩任务
参与者	测试人员
优先级	高
用例描述	测试人员通过前端页面向系统提交代码插桩任务。
前置条件	测试人员拥有权限，可以对本系统进行相关操作。
主事件流	1. 用户选择待插桩的应用。 2. 用户向系统提交插桩的申请。 3. 系统显示申请成功，在后端开始插桩操作。 4. 系统显示插桩成功。
后置条件	在插桩应用集中显示插桩后的应用。
扩展事件流	3a. 系统没有检测到待插桩的配置文件，显示插桩失败。

创建变异任务是本工具的亮点，表3.9是对变异任务的用例描述。用户可以手动选择变异因子，随后对插桩后的源代码进行变异。根据变异因子执行变异操作后，系统会自动对应用源码进行编译，并对编译后的应用进行归档。有些变异因子可能会因为代码中不存在可变异的对象导致变异失败。因为不同的应用差别很大，插桩和变异都是对源代码进行改动。源代码改动之后有可能导致应用无法编译，或者编译后的应用无法运行。所以编译后的应用需要管理员的审核才可以进入待测应用集之中。

表 3.9: 创建代码变异任务用例描述

描述项	**说明**
用例编号	UC3
用例名称	创建代码变异任务
参与者	测试人员
优先级	高
用例描述	测试人员选择应用在该统中进行变异。
前置条件	应用应执行过插桩操作且插桩成功，且存储在系统插桩应用集之中。
主事件流	1. 测试人员在插桩应用集中选择待变异的应用。 2. 测试人员选择变异因子。 3. 向系统提交变异任务。 4. 系统显示变异代码生成成功。 5. 系统自动开始变异代码的编译。 6. 系统重复第 5 步直到将所有变异源码编译。 7. 系统返回变异结果。
后置条件	管理员可以在变异生成列表中看到变异后且编译成功的应用，用于后续审核操作。
扩展事件流	4.a 用户选择所有的变异因子均变异失败，显示执行失败。

表3.10是对管理员审核的用例描述。管理员会通过源码阅读及真机测试来确定审核的结果。由管理员认证插桩及变异成功的应用，可以在后续的自动化测试工具的测试过程中选取。

表3.11是创建工具测试任务的用例描述。用户在选择自动化测试工具，待测应用集、以及连接待测安卓设备之后。可以由工具自动执行并监控测试的进行，并保证测试正确完整的执行。在执行的过程中手机插桩后自动生成的数据文件，在执行之后将所有应用测试的数据均持久化到系统之中。这些数据可以用于后

续的数据得分汇总。

表 3.10: 审核处理后应用用例描述

描述项	**说明**
用例编号	UC4
用例名称	审核处理后的应用
参与者	管理员
优先级	中
用例描述	管理员对变异完毕的插桩应用进行审核，审核成功的应用将加入待测应用集中。
前置条件	系统存在正在审核过程中的变异应用。
主事件流	1. 管理员打开待审核的应用列表，选中需要审核的任务。 2. 管理员通过真机测试以及阅读源码对应用进行审核。 3. 系统显示审核完毕。
后置条件	审核成功的应用会作为待测应用加入系统的待测应用集。
扩展事件流	无

表 3.11: 创建工具测试任务用例描述

描述项	**说明**
用例编号	UC5
用例名称	创建工具测试任务
参与者	测试人员
优先级	高
用例描述	测试人员开始对自动化测试工具进行测试。
前置条件	系统存储中含待测应用集，且连接可用安卓设备。系统对待测工具需完成适配。
主事件流	1. 用户选择待测自动化测试工具、待测应用集以及测试机器。 2. 系统准备自动化测试工具的环境。 3. 系统开始在设备上执行对应用的测试。 4. 重复执行 3 直至每个待测应用均在设备上执行过测试。 5. 系统收集日志以及运行时文件，并将收集的数据持久化到系统之中。

后置条件	系统内可以查看此次测试的相关测试结果，用以后续系统评分。
扩展事件流	3.a 应用的自动测试执行失败，记录失败。 3.b 自动化测试工具不能正确测试应用，表现为不能打开应用或不能将测试应用置为前台，系统将自动杀死单次测试，并继续进行后续测试。

而评估模块用以解析软件测试所生成的数据以及对其进行汇总得分，并制定评估相关的标准。系统首先通过分析数据文件得到多个角度的原始分，随后将原始分进行归纳之后汇总得到综合得分，即生成可以客观表示自动化测试工具测试效果的数据。表 3.12是评估测试结果的用例描述。

表 3.12: 评估测试结果用例描述

描述项	**说明**
用例编号	UC6
用例名称	评估测试结果
参与者	测试评估模块
优先级	高
用例描述	测试评估模块通过之前执行测试的数据得出评估结果。
前置条件	自动测试任务执行成功，测试数据及日志持久化成功。
主事件流	1. 获取测试任务的数据集。 2. 解析数据集，将插桩运行时文件解析成为系统可读报告。 3. 调用当前插桩应用集的评估模块，对解析的数据集进行预处理，并获得数据原始分。 4. 将原始分数汇总，并将其持久化到系统之中。 5. 系统显示评估成功。
后置条件	评估得到的结果可以为数据可视化提供基础数据。
扩展事件流	1.a 数据获取失败

表 3.13是展示测试结果的相关用例描述，结果包括异常触发情况、Jacoco 的覆盖率报告、UI 控件覆盖率报告以及异常得分等情况，同时会展现该次测试的结果箱型图以及堆叠条形图，用以图形化的结果展示。

表 3.14是展示相应测试工具历史测试数据总览的用例描述，通过将同一测试工具历史上的历次测试任务的数据汇总，对数据按照用户配置进行加权平均，

用以得到测试工具的综合评分。展示的数据项与单次测试的数据相同，也同样以图形化的方式展现出来。

表 3.13: 查看测试结果可视化结果用例描述

描述项	**说明**
用例编号	UC7
用例名称	查看测试的可视化测试结果
参与者	用户
优先级	中
用例描述	用户可以在测试列表中通过点击测试结果入口查看测试结果可视化的界面。
前置条件	相应测试工具的自动测试执行完毕。
主事件流	1. 用户点击任务列表中的某次分析任务的条目。 2. 系统显示该次分析后的结果。
后置条件	无
扩展事件流	无

表 3.14: 查看测试工具的历史测试数据总览用例描述

描述项	**说明**
用例编号	UC8
用例名称	查看测试工具的历史数据总览
参与者	用户
优先级	中
用例描述	用户可以按测试工具分类，获得所有测试的数据总览，查看整体测试效果的可视化报告。
前置条件	测试工具在框架中执行成功过。
主事件流	1. 用户点击选择某测试工具的条目。 2. 用户选择需要汇总分析的多个任务。 3. 系统显示测试工具的测试结果总览。
后置条件	无
扩展事件流	无

3.3　系统总体设计

3.3.1　系统架构设计

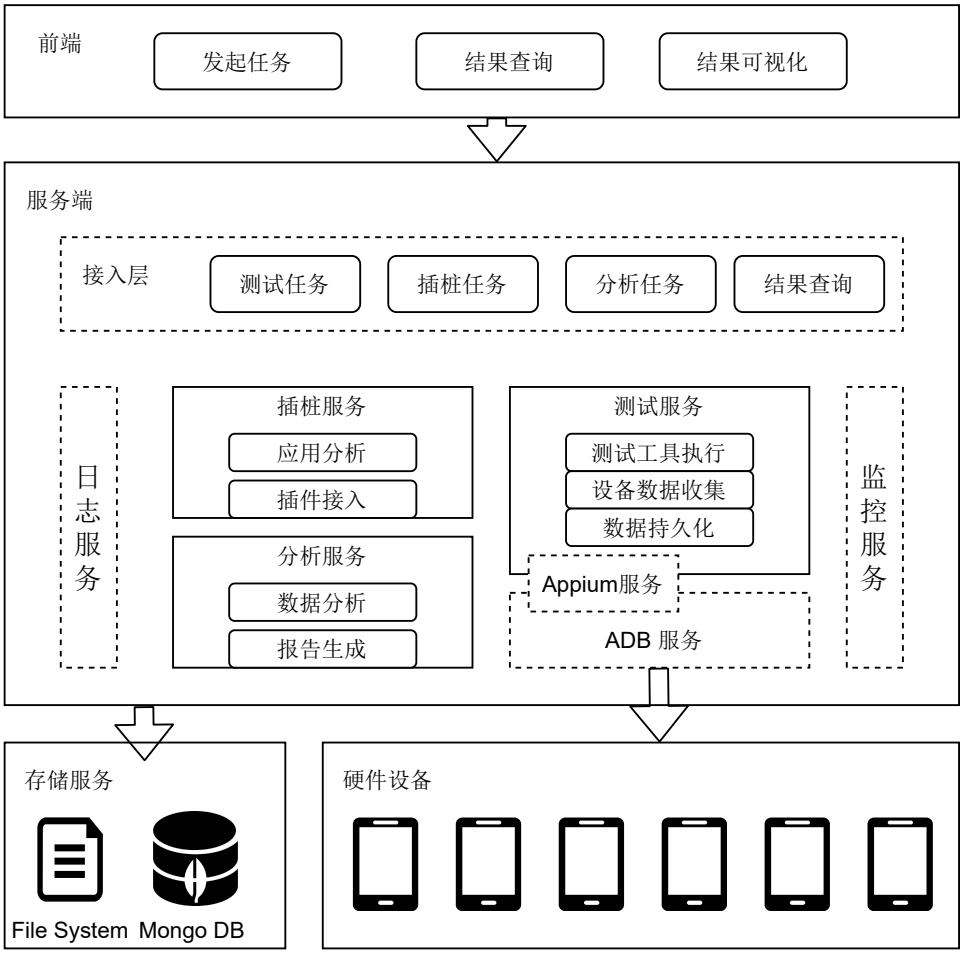

图 3.3: 系统架构图

本系统的架构图如图 3.3所示，分别为前端、服务端、存储服务以及硬件设备，其中服务端又分为接入层以及服务层。本节将会对核心架构进行简要介绍，随后的几个小节将会详细介绍核心模块的设计，并对核心模块的执行流程进行分析和阐明。

系统采用前后端分离的开发形式，前后端解耦与业界主流开发方式相同。前端拥有向后端上传应用、申请对应用源码的插桩以及变异、申请对变异源码进行编译、申请对自动化测试工具进行测试和查看测试结果等功能。框架会在后端接收这些请求，并分别对不同请求采取不同逻辑的操作，调用后端不同模块以完成前端的请求。申请的流程即为普通的 Restful API 调用，是目前开发中最

为常见的实现方式，因此此处不再详细描述。下面对此系统的后端的核心业务模块进行简要介绍。

移动应用管理模块负责源代码的插桩、变异以及编译等操作。自动化测试工具的目标是对应用有更好的测试效果，移动应用管理模块通过对应用的插桩改造，使应用有展现自动化测试工具测试效果的能力。通过 ASM 插桩对应用的所有点击操作进行动态代理，并在运行时日志中打印输出控件的 ID 以及页面信息。工具借此信息可以获得自动化测试工具点击控件以及页面的覆盖率情况；通过使用 Jacoco 对源码进行插桩，对所有的字节码文件进行埋点。在运行时保存相应的运行时数据文件，以此可以获得自动化测试工具测试对源代码覆盖率的情况；使用 MDroid+ 对源码进行变异，并对 MDroid+ 进行改造，在执行至变异代码时打印日志。这样在测试时可以获取自动化测试工具杀死变异的相关结果。

测试模块在用户发起对框架的测试请求时被启动。自动测试模块运行在 Docker 之中，可以有效解决自动化测试工具测试环境配置困难，导致系统可移植性差的问题。模块启动之后，将负责自动化测试工具的准备以及执行工作。在自动化测试之前，框架将设备接入到框架之中。通过 ADB 对设备进行环境的设置，使自动化测试工具可以正确的在设备上执行测试。在测试的过程中，通过 ADB 对设备的执行情况进行实时检查，此模块会自动处理运行时的错误情况，并将自动化测试工具以及设备的运行时日志持久化保存到系统之中。在单次测试结束之后，模块将插桩后应用自动生成的运行时文件同样持久化到系统之中，用以后续的评分，直至所有测试执行完毕。

3.3.2　系统 4+1 视图

系统 4+1 视图 [37] 是采用多个不同的视角的视图，来对软件密集型系统的体系结构进行描述。分别是逻辑视图、开发视图、物理视图和过程视图。此外选取用例图作为加一视图对架构服务进行说明。4+1 视图通用性很强，已经在很多大型项目中使用。视图从用户、开发人员、系统工程师和项目经理等不同利益相关者的视角来对系统进行解释，通过使用 4+1 视图让不同的用户均可得到自己想要的信息。

本节将工具使用 4+1 视图的方式进行详细介绍。其中用例图与前文中提到的用例图相对应，在本节中不再赘述。

图 3.4是本系统的逻辑视图。逻辑视图从用户的视角出发，对系统中提供给用户的功能和服务进行详细解释。系统分为从逻辑角度可以分为四个模块，分别是：Web Module、Test Module、Instrument Module、Analysis Module。Web Module 负责向后端请求任务，如果执行插桩或者变异任务，需要通过 Instrument Service

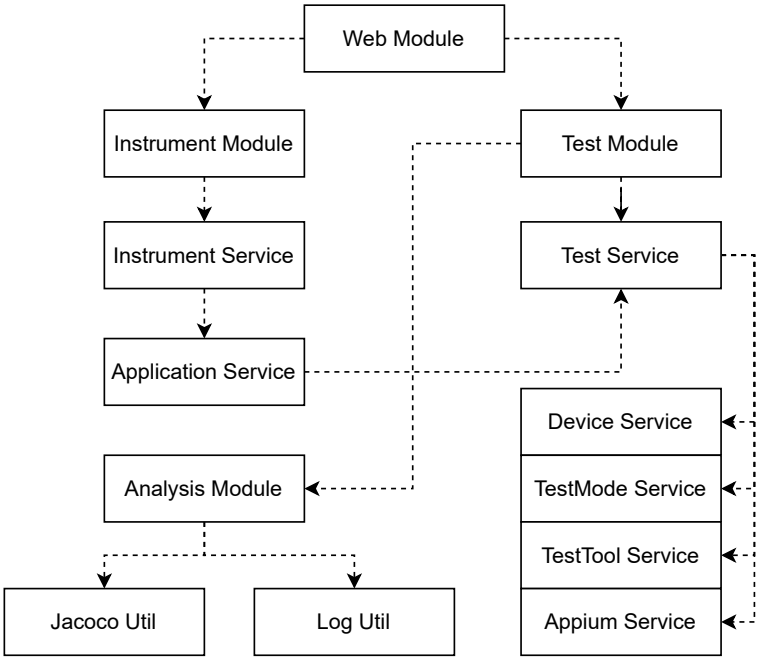

图 3.4: 逻辑视图

来执行插桩或者变异的详细业务逻辑，通过 Application Service 可以对源代码进行修改变异以及编译，在编译之后可以实现对应用的插桩或者变异。插桩和变异后的应用可以提供给 Test Module 作为待测应用集。用户执行对自动化测试工具的测试需要使用此模块，工具通过 Test Service 对自动测试进行统一调配。在执行的过程中需要调用 Device Service 来对设备进行操作核管理。测试不同项目时需要对待测工具以及设备做不同的准备工作，所以可以通过调用 TestMode Service 来管理不同的测试项目。而通过 TestTool Service 可以实现对自动化测试工具的管理、控制。有些工具需要依赖 Appium 工具。系统封装了 Appium，Appium Service 可以启动相应程序，在执行过程中可以将 Appium 的日志保存到相应目录下，并在测试结束后自动杀死 Appium，对占用端口其进行回收。在测试执行完毕之后可以通过 Analysis Module 来对测试工具的表现进行评价。其中用到 Jacoco Util 来解析 Jacoco 插桩后的应用运行时生成的记录文件，分析文件后可以得到自动测试框架的代码覆盖率等信息。然后通过 Log Util 来解析设备运行时的日志文件，以获得异常、UI 控件覆盖率以及变异三个方面的得分。

图 3.5是本工具的开发视图，开发视图是以开发人员的视角来对系统的组织架构进行解析。整个系统可以分为三大模块：前端的 UI 模块、后端的业务模块以及技术的服务模块。由 JavaScript、Echarts 以及 React 来构成与用户交互的前端模块。后端使用 Spring Boot 进行实现，参考成熟项目架构而设计。

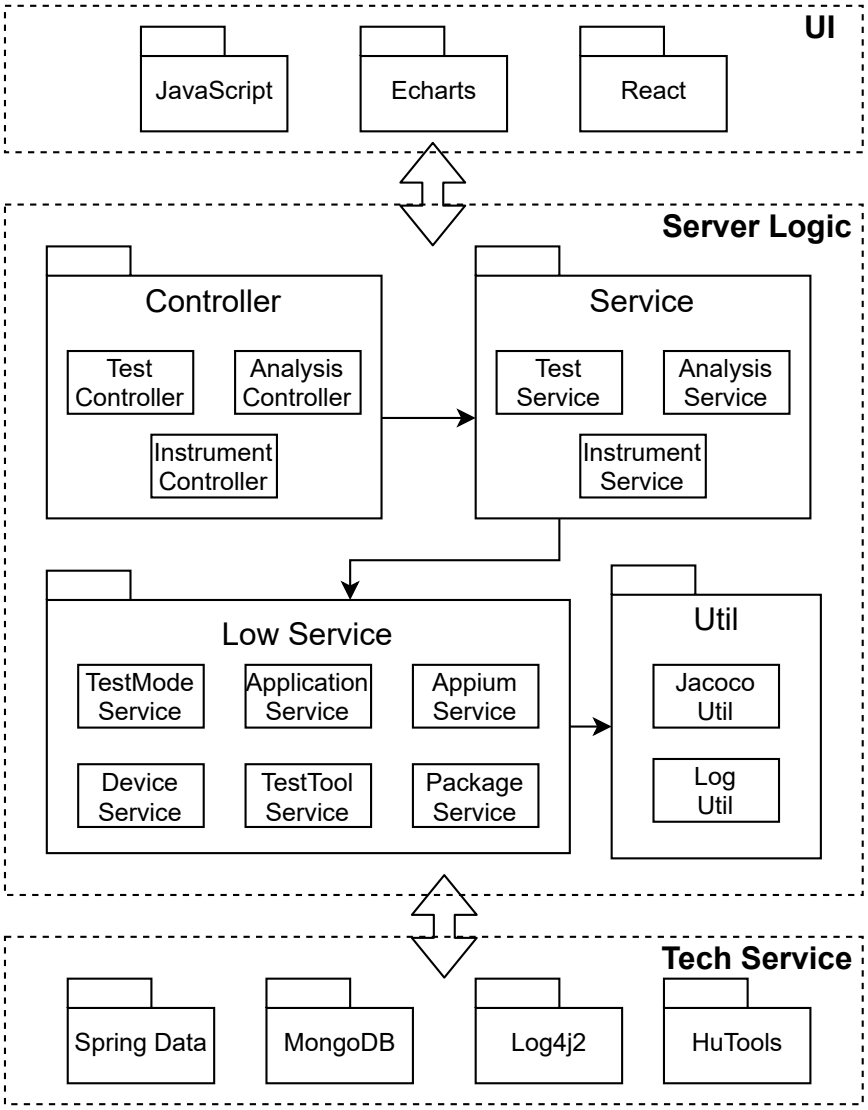

图 3.5: 开发视图

- Controller 层主要负责对前端的申请的处理转发、参数的基本校验等。
- Service 层主要负责为 Controller 层提供服务，负责相对具体的业务。
- Low Service 层是通用业务的处理层，为了避免 Service 层之间互相调用增加系统耦合抽象出此层。是对 Service 层通用能力的具体下沉，也负责对第三方工具进行封装，使其接入工具之中。并与负责曾删改查 Reporsitory 进行直接交互。
- Util 包主要负责对通用操作进行封装抽象，减少代码的重复。与业务逻辑没有直接相关。

技术服务使用了工具包 HuTools、日志框架 Log4j2、数据库 MongoDB 等。

进程试图是以系统工程师的视角对系统进行解释。可以详细解释系统的动态运行方式，着重描述系统的运行时行为。对解决并发、分布式、性能和可伸缩性等问题有非常重要的作用。

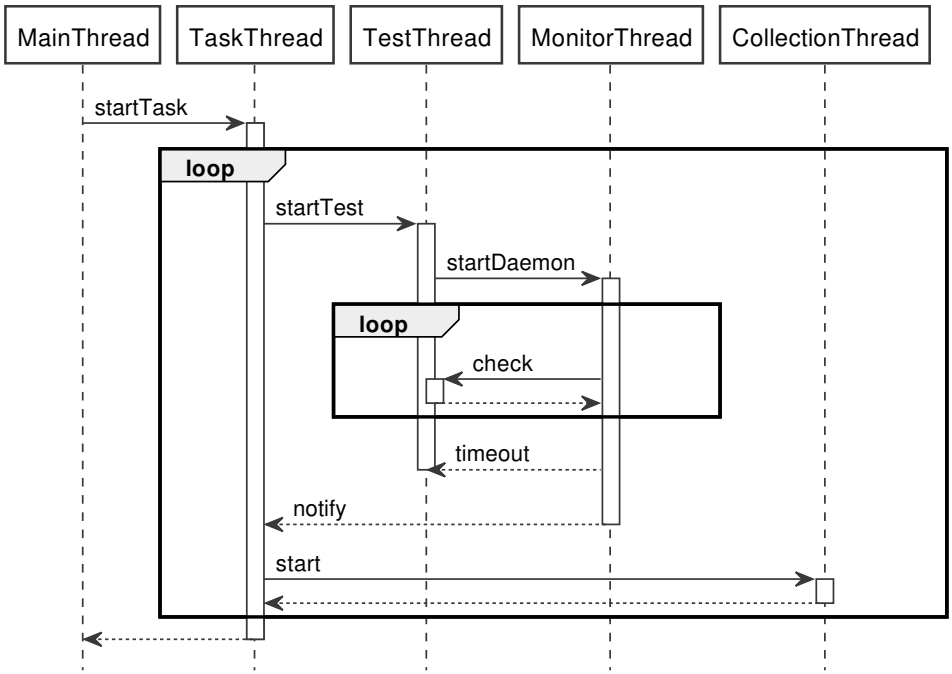

图 3.6: 进程视图

上图 3.6是本系统的进程试图。主线程收到测试请求时会启动一个任务线程。因为每个任务一般都涉及到自动化测试工具的多次测试，所以任务线程需要再启动一个测试线程。测试以单个设备测试单个应用的测试任务为一个测试单位，分别在单例线程池进行串行执行。在测试开始的时候，会启用两个存储日志的线程，线程将会分别自动存储设备的 Logcat 日志以及自动化测试工具的执行日志到系统之中。为保证工具的正确执行，在测试线程执行的时候会启动一个监控线程来时刻监控测试工具的测试表现。如果出现长时间没有启动测试应用或设备长时间没有停留在待测程序的情况，将会主动杀死自动化测试工具，进行下一个测试。若自动化测试工具测试超时，也会主动杀死该工具。如果自动化测试工具测试完成，监控线程将主动结束。在测试完成之后，将会进行测试的信息收集，其中包括 Jacoco 的运行时文件等。这些文件将用以后续的评分模块。

部署视图是以运维人员的视角对系统进行解释。它着重于解释说明整个系统的物理结构，包括组件之间的拓扑连接等等，又称为物理试图。图 3.7是本系统的部署视图。

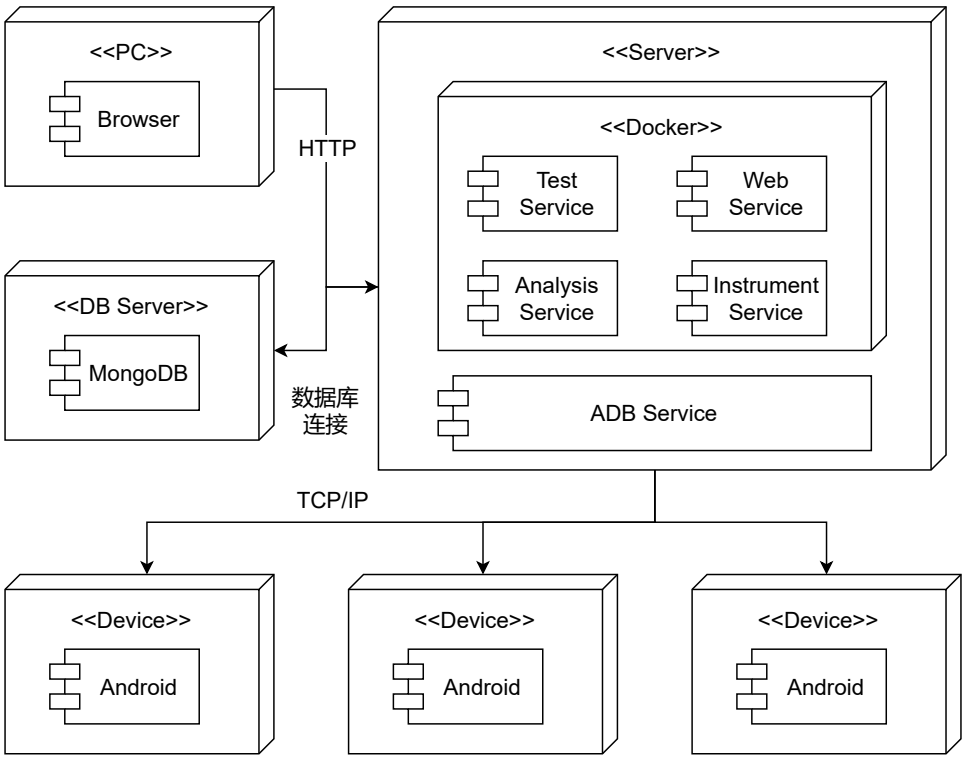

图 3.7: 部署试图

用户可以通过 PC 端的浏览器访问系统的前端页面，并对系统的服务器发起 HTTP 请求。后端接收 HTTP 请求之后，会根据请求启动不同的服务。系统的应用信息都存储在 MongoDB 中。系统的自动工具的测试服务、结果分析服务插桩服务等核心业务均在服务器的 Docker 环境内运行。用以提供给自动化测试工具测试的设备通过 ADB Server 与服务器相连，设备与服务器相连可以采取两种不同的方式，一是可以使用有线连接测试本地的机器，也可以使用 TCP/IP 的方式测试云端的机器。当用户进行插桩服务，系统会在容器中完成对应用的插桩及编译操作，并将插桩后的应用信息存储到数据库中，用以后续测试。

3.3.3　插桩流程设计

图 3.8是系统对应用进行插桩的流程图。此模块主要负责对应用源码进行改造，使应用可以在不改变先前应用行为的情况下满足测试需求，为测试提供需要的数据。同时负责相关变异应用的生成，用以满足后续变异测试的需求。执行插桩操作可以解决以下几个问题:

- Jacoco 插件使用 Offline 方式对代码进行埋点。该方式需要在系统中存储 coverage.ec 文件，通过对该文件进行分析可以得到代码覆盖率报告。由于

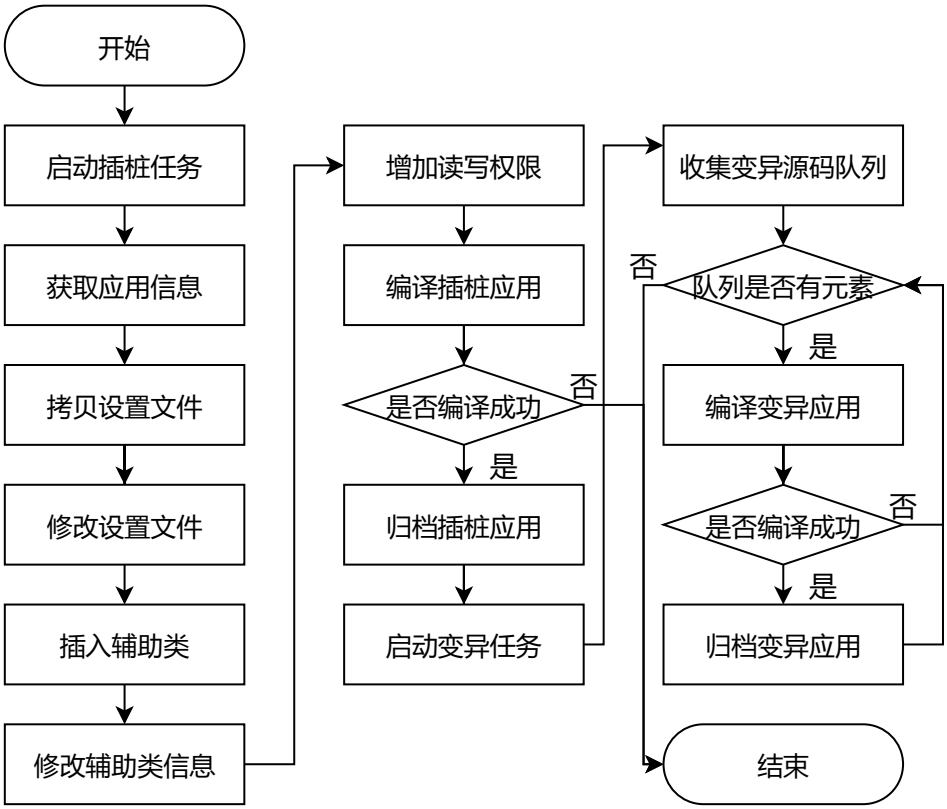

图 3.8: 插桩流程图

有些应用不涉及文件的读写操作，所以需要修改应用的文件读写权限。

- 生成运行时文件以及开启对点击事件的监控都需要初始化，需要修改应用初始化的相应文件。
- 实现覆盖率测试需要插件以及模块的引入，需要对应用源码进行相应的配置。

通过该流程的执行，可以有效解决以上问题。首先应用启动插桩任务，系统会读取相应的应用信息。将插件的配置文件拷贝到待插桩的应用中。修改测试文件使引入的插件配置可以正确被读取以及启用相应插件。随后插入辅助类，保证应用初始化时会初始化插件。然后修改辅助类信息，保证辅助类的包名、引用等配置正确。随后增加读写权限，保证应用可以正确保存运行时文件至 SDCARD 目录中。如果编译成功将会把编译成功的应用归档置系统中，并可以进行后续的变异。启用变异任务后，MDroid+ 将会生成数个变异后的源码。源码需要拷贝到项目之中才可以编译。由于考虑到 CPU 开销以及存储开销，系统采用串行的方式对所有变异代码进行编译。有的变异会导致编译失败，而编译成功的代码

将会作为原应用（插桩成功后编译的应用）的变异应用归档到系统之中，用于后续测试。

3.3.4　覆盖率检测流程设计

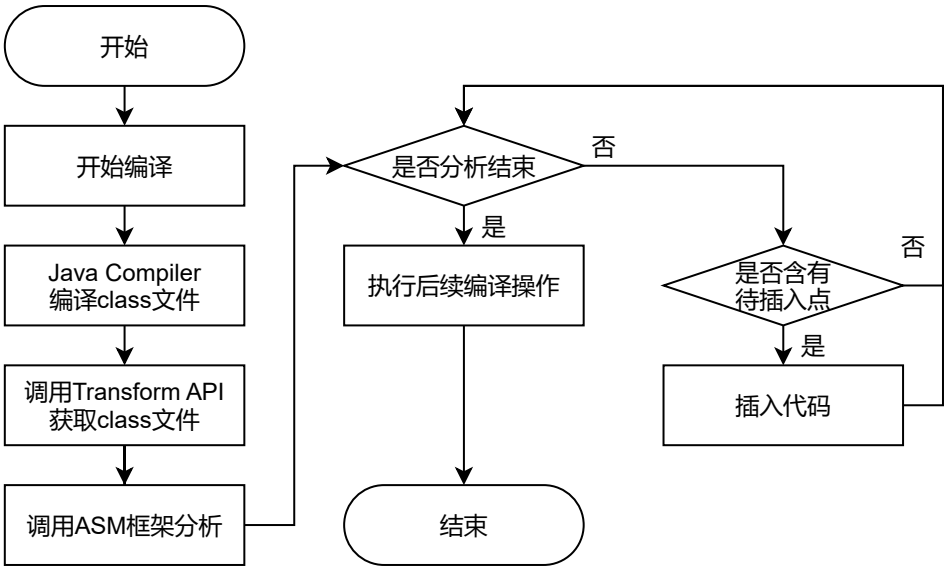

图 3.9: 编译埋点流程图

本系统的关键创新部分是如何实现系统 UI 控件覆盖率的统计。根据分析源码难以获得当前应用所有控件的信息，此功能旨在记录当前测试中共点击了多少个控件，并对其信息进行记录。所以需要对控件的点击方法进行修改。本工具通过在应用编译代码的过程中，使用 ASM 框架对 class 文件进行修改来实现插桩。图 3.9就是编译埋点的流程图。在编译过程中，首先会把 java 或 kt 文件编译成 class 文件。随后通过 Gradle 的 Transform API 可以访问 jar 包以及应用的 class 文件，通过该 API 我们可以使用 ASM 框架对所有的 class 文件进行加载、分析。进以判断 class 文件中是否拥有控件的点击方法，最后修改相应的点击方法并插入修改后字节码，再对原文件进行覆盖，从而达到埋点的效果。后续由于安卓使用 DEX 虚拟机，并不能执行 class 文件，需要将 class 通过 SDK 工具转换为 dex 文件。这时被转换的 class 文件已经是我们修改后的文件，所以点击操作也将拥有我们修改的行为。后续与正常编译的流程相同，并不是本文的关注点，在此不做赘述。

图 3.10是代码在运行时记录 UI 控件信息的具体流程图。因为源代码已经分别为各种控件的点击事件进行了插桩，在代码运行时，其点击方法已经被 ASM 框架在编译过程中修改。在系统执行自动化测试工具之后，工具会对应用进行

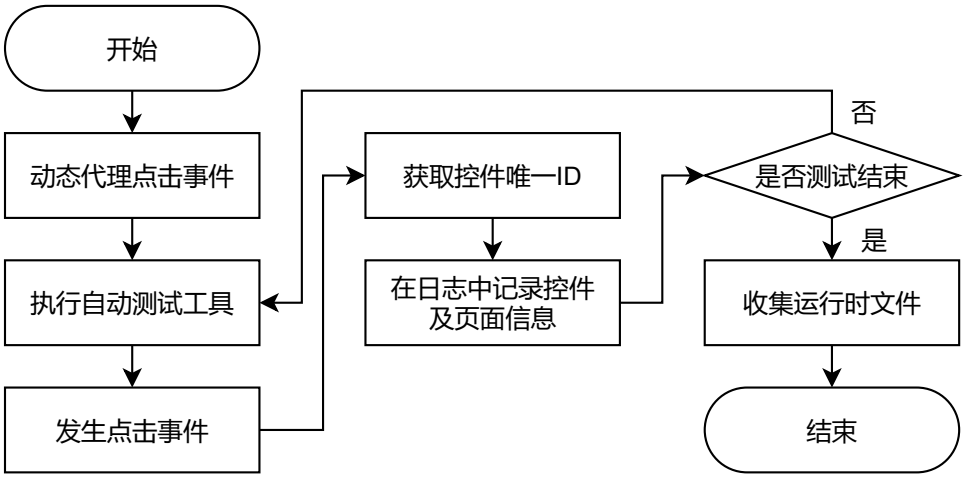

图 3.10: 测试执行控件收集流程图

测试。在控件被点击后就会调用在编译过程中埋入的方法，该方法会将控件的信息进行收集，并为其生成唯一的 ID，将控件信息以及 ID 信息记录在系统日志之中。本工具采用控件在控件树中的全路径来表示唯一的 ID。随后将控件的信息、页面的信息以及控件的 ID 记录到日志当中。在测试结束之后将日志信息进行收集，即可得到当前测试的控件使用情况。随后与历史测试数据进行对比，通过计算可以得到测试的原始分。

3.3.5　测试流程设计

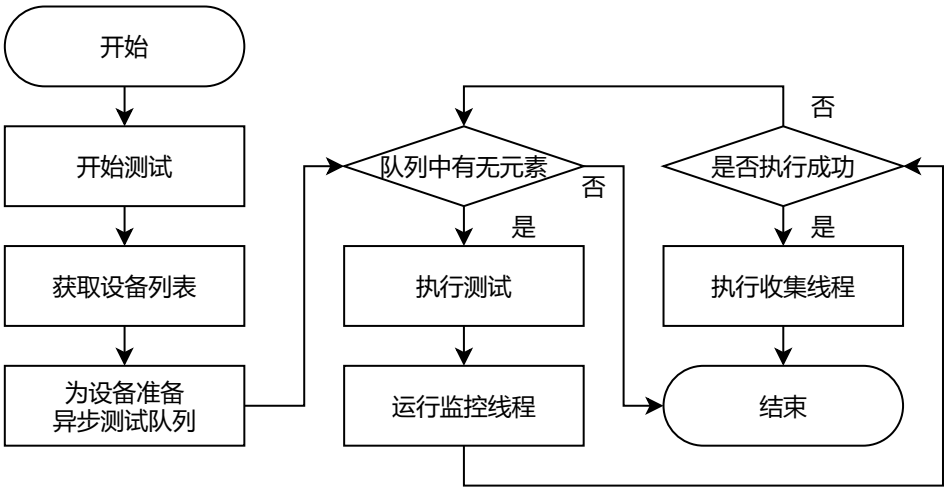

图 3.11: 测试流程图

图 3.11是系统在正常状态下收到测试请求的流程图。此模块的主要功能是

保证测试可以使用异步的形式高效、正确的执行，并且可以得到系统需要的信息。为了保障以上需求的实现，系统采用以上流程进行执行。系统在收到前端通过 URL 发起的请求之后，由测试工具启动测试服务。待测测试工具已经实现了 TestToolService 的接口，这样可以保证系统可以正确执行以及监控测试工具。开始测试之后，首先会获取将要进行测试的设备列表。列表中的安卓设备包括后续测试将要使用的 SDK 版本、序列号等信息。随后以设备为单位，为每个设备准备一个异步测试的队列。队列中的每个元素是一次完整的测试任务，其中包括：准备测试环境、启动监控线程、启动日志记录、启动测试工具以及收集运行时文件。由实现了 TestToolService 接口的相应工具服务类进行设备的测试环境准备，这是由于每个测试工具对设备的环境要求不同，需要对其进行特殊适配。系统实现了抽象的 TestThread 线程，TestThread 线程可以提供日志记录以及工具日志记录以及启用监控线程的服务。最后由实现了 TestModeService 的具体服务类进行运行时文件的收集，该类服务会自动收集此类测试需要的运行时文件。

3.3.6　评估指标选取与设计

该指标分数基于安卓自动化测试工具的运行时数据，主要从代码覆盖率、异常得分、UI 控件覆盖得分、变异得分、碎片化得分五个方面对自动化测试工具的测试效果进行评估。其中代码覆盖率、异常得分以及碎片化得分参考了 Cloudhary 对自动化测试工具评估的指标。针对第一章国内外相关研究中提到 Cloudhary 研究的问题，本文提出了变异得分以及 UI 控件覆盖得分对其进行补充，具体选取的原因在本文 1.2.3节中有详细介绍。本小节将具体介绍以上几个方面的得分公式并对其进行详细解释。

得分的最小单元为单个测试工具在单个设备上测试单个应用，每个最小单元的得分由异常得分、变异得分、代码覆盖率得分、UI 控件得分四个部分组成。异常原始分即为杀死异常的比率，获得的方式如公式3.1所示：

$$MutantRate = \frac{Mutant_{killed}}{Mutant_{total}} \times 100\% \tag{3.1}$$

在该公式中 $Mutant_{killed}$ 是自动化测试工具杀死变异的数目，$Mutant_{total}$ 是测试变异应用的总数，通过该公式可以计算出该应用的变异得分。

异常得分主要考察自动化测试工具检测异常的能力，系统会记录所有测试工具对该应用测试时生成的异常。通过对比测试查找的异常数目与系统记录的异常数目，可以得到其异常得分，如公式3.2所示:

$$ExceptionRate = \frac{Exception_{found}}{Exception_{most_found}} \times 100\% \tag{3.2}$$

公式中 $Exception_{found}$ 为该次测试找到的异常去重后的数目（后文中的异常数目均指去重后的数目），而 $Exception_{most_found}$ 为系统中所有测试工具找到最大的异常数目。通过该公式的计算即可得该应用的异常原始得分。

UI 控件覆盖率可以用来考察自动化测试工具在 UI 测试中对控件覆盖的能力，系统同样会记录该工具对该应用测试时所点击的 UI 控件信息，如公式3.3所示，通过对比测试点击的控件数目与系统记录的控件数目，可以得到控件得分。

$$ComponentRate = \frac{Component_{found}}{Component_{most_found}} \times 100\% \tag{3.3}$$

在本公式中 $Component_{found}$ 为该次测试中自动化测试工具所点击的控件数量，而 $Component_{most_found}$ 与异常得分计算方式相同，为测试中所有测试工具检测控件的最大值，通过该公式的计算可以获得控件覆盖的原始分。

代码覆盖率展示的则是自动化测试工具执行测试覆盖代码的能力。通过 Jacoco 对源代码进行插桩，在应用运行时会生成数据文件。通过数据文件可以生成应用的代码覆盖率报告，计算覆盖率报告中的数据，如公式 3.4所示：

$$CoverageRate = \sum_{i=1}^{n}\left(\beta_i \times \frac{Meta_i - Missed_i}{Meta_i}\right) \times 100\%, \sum_{i=1}^{n}\beta_i = 1 \tag{3.4}$$

覆盖率的最终得分为所有不同覆盖率的加权，即行覆盖率、分支覆盖率、指令覆盖率、方法覆盖率、圈复杂度、类覆盖率的加权。以行覆盖率为例，公式中 $Meta$ 指 Jacoco 检测到的总行数，而 $Missed$ 指在自动化测试工具测试过程中没有命中的行数，$Meta$ 与 $Missed$ 的差比上总行数，即为测试覆盖行数在总行数的占比。而 β_i 则是不同尺度的覆盖率在覆盖率得分中的占比，其和为 1。

设备单应用的得分，是自动化测试工具的评估体系中的最小评分单元。该评分代表了自动化测试工具对单个设备上的单个应用测试能力的评估，具体计算方式如公式 3.5所示：

$$\begin{aligned} DeviceApplicationScore &= \alpha_1 * CoverageRate + \alpha_2 * MutationRate \\ &+ \alpha_3 * ExceptionRate + \alpha_4 * ComponentRate, \sum_{i=1}^{4}\alpha_i = 1 \end{aligned} \tag{3.5}$$

其中参数均由以上公式计算所得。α_i 为常量，代表不同测试类别在最终得分的比重，其和为 1。

设备得分是自动化测试工具对不同版本设备测试能力的综合评估，首先将 $DeviceApplicationScore$ 按设备分类，求出单个设备测试应用集的平均分。然后

将设备得分按照 SDK 版本以及屏幕在安卓设备中的占比加权，得到设备得分，可以有效体现该工具处理碎片化问题的能力，具体如公式 3.6所示：

$$DeviceScore = \frac{\sum_{j=1}^{m}\left(\frac{\sum_{i=1}^{n} DeviceApplicationScore_{j_i}}{n} \times Partition_{j_sdk_version}\right)}{\sum_{j=1}^{m} Partition_{j_sdk_version}} \times \alpha + \frac{\sum_{j=1}^{m}\left(\frac{\sum_{i=1}^{n} DeviceApplicationScore_{j_i}}{n} \times Partition_{j_screen}\right)}{\sum_{j=1}^{m} Partition_{j_screen}} \times (1-\alpha), \alpha \in (0,1) \tag{3.6}$$

公式中 n 为待测应用集的数量，m 为设备的数量，$Partition_{j_sdk_version}$ 为当前设备 SDK 版本在所有安卓设备中的占比 [38]，$Partition_{j_screen}$ 为当前屏幕属性在所有安卓设备中的占比 [39]，α 为常量。

应用得分是自动化测试工具对应用测试能力的综合评估，首先将不同设备上相同应用的 $DeviceApplicationScore$ 得分求平均，该值即为自动化测试工具在设备集群上测试单个应用的原始分，再将该值求平均即可获得自动化测试工具测试应用集的综合评分。如公式 3.7所示：

$$ApplicationScore = \frac{\sum_{i=1}^{n} \overline{DeviceApplicationScore_i}}{n} \tag{3.7}$$

其中 n 为应用集中应用的数量，$\overline{DeviceApplicationScore_i}$ 即为单个应用在不同设备上测试的平均得分，再将其求平均即可得到结果。

$FinalScore$ 为自动化测试工具的最终得分，体现了自动化测试工具的测试能力。得分由应用得分与设备得分加权组成。如公式 3.8所示：

$$FinalScore = ApplicationScore \times \alpha + DeviceScore \times (1-\alpha), \alpha \in (0,1) \tag{3.8}$$

其中 α 为常量。

3.4　本章小结

针对目前自动测试框架缺乏客观全面评估的场景，本章首先对安卓自动化测试工具的多维评估系统进行了需求分析。其中包括了功能性需求的分析、非功能性需求的分析以及系统用例的设计。随后对总体设计进行了详细剖析，从架构设计开始讲解，通过 4+1 视图对系统进行详细剖析。随后又初步介绍了插桩、覆盖率检测和测试的设计，通过流程图对以上模块进行简要剖析。最后对系统的评估指标进行解释，介绍了相关指标的计算方法。

第四章　安卓自动化测试工具评估系统的详细设计与实现

本章将会对工具的核心部分进行介绍。首先会对核心部分进行概述，随后将从时序图、主要类表、类详情列表、关键类图和关键代码几个方面对其设计实现进行详细介绍。其中核心部分包括：UI 控件覆盖检测插件，此插件通过修改字节码的方式使原应用拥有记录点击事件的能力；源码插桩变异模块，通过插桩可以做到全自动接入覆盖率检测插件以及 Jacoco 插件，并对插桩后的代码进行变异以及编译归档；测试模块，此模块可以异步进行自动化测试工具的测试，并完成环境准备、测试监控、日志收集以及文件收集等活动；测试结果评估模块，通过将测试的日志以及运行时文件分析收集，得到相应测试的能力评价。

4.1　UI 控件覆盖检测插件详细设计与实现

4.1.1　UI 控件覆盖检测插件概述

安卓软件的界面由 UI 控件组成，安卓自动化测试工具的主要行为就是与控件交互。本系统将插桩后的应用提供给自动化测试工具测试，这样就可以获得工具与控件交互的数量。我们需要的是每当操作控件或点击控件时，系统可以记录下相应的控件信息。直接修改源码是不现实的，因为工作量很大而且可能会有遗漏，导致检测结果从数量和质量上均不能满足测试的需要。本文通过字节码插桩实现了对点击事件的代理，可以有效对所有点击事件进行监控。这样只用少量的代码就可以完成对测试工具点击事件的收集。插件不需要应用获取额外的权限同时也不需要对设备进行额外配置，具有较好的易用性。本节将介绍控件覆盖检测插件在修改字节码、记录控件信息等方面的详细设计与实现。

4.1.2　UI 控件覆盖检测插件设计实现

插件可以在应用编译过程中对点击 UI 控件的操作进行插桩，具体操作如时序图 4.1所示。在应用执行编译任务时，插件会调用 Gradle 的 Transform API 里的 transform 方法，该方法用于读写应用的 jar 包以及编译好的 class 文件。插件按照以下逻辑重写了 transform 方法：方法首先会对类文件进行筛选，以去除掉确定不需要插桩的 class 文件，这一步的作用主要是避免编译速度过慢。如果文件可能进行插桩操作，则调用 ModifyHelper 里的 modifyClass 方法对类文件进行进一步判断。在 ModifyHelper 对 class 文件的信息进行读取之后，调用其 accept

方法，使插件中实现的 FeatClassVisiter 可以访问相应的 class 文件，并对其进行修改。在修改前将调用 ClassReader 的 readMethod 方法，用以获取文件的具体方法的信息，再使用 visitMethod 方法获得插件自定义的 FeatMethodVisiter 类，随后即可对该类进行修改。插件对 onMethodEnter 方法进行重写，通过该方法可以在目标方法前插入代码。在重写的 onMethodEnter 方法中会判断调用方法是否来自于控件类以及方法的描述是否与点击操作相同，满足条件的方法会使用 visitMethodInsn 创建指令，以调用我们在 API 中预设的方法。调用会将需要的信息传递到记录控件信息的方法中，在后文中将会详细介绍此方法。

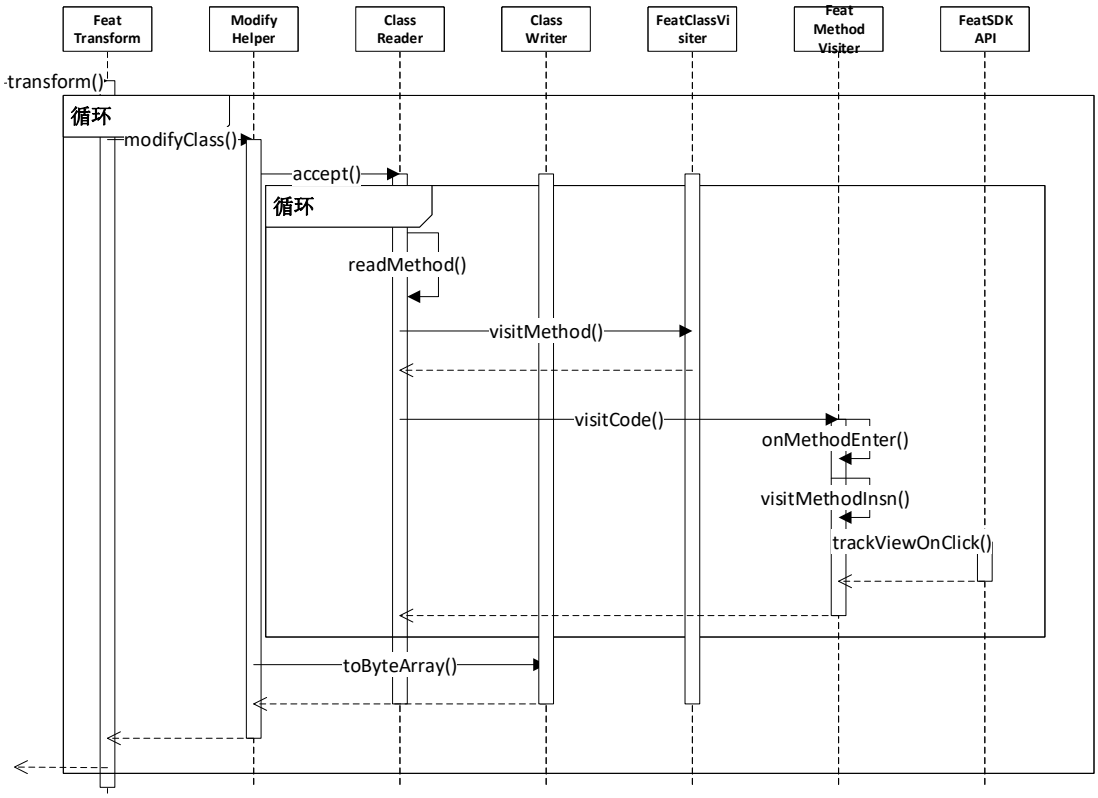

图 4.1: 应用插桩时序图

表4.1是插件主要类，FeatTransform 继承于 Gradle 的 Transform 类，该类用以提供项目中的 class 文件以及项目相关 jar 包中的 class 文件。ModifyHelper 是用来辅助修改 class 文件的工具类，该类负责修改操作的具体逻辑。ClassReader、ClassWriter 以及 ClassVisitor 是 ASM 框架提供给用户用以读取、拜访以及修改字节码文件的类。代码重写了 ClassVisitor，以达到替换 MethodVisitor 为自定义的 FeatMethodVisiter 的目的。插件在 FeatMethodVisiter 类中对具体的点击操作进行筛选埋点。FeatSDKAPI 里则是具体的信息记录逻辑。

表 4.1: UI 控件覆盖检测插件主要类

类名称	作用
FeatTransform	负责项目构建的转换
ModifyHelper	负责修改 jar 包以及 class 文件
ClassReader	负责帮助 ClassVisitor 类访问需要修改的文件
ClassWriter	负责帮助 ClassVisiter 生成 class 文件的二进制文件
FeatClassVisitor	负责修改相应的字节码，完成插桩活动
FeatMethodVisiter	负责修改方法的字节码
FeatSDKAPI	负责提供记录控件信息的方法

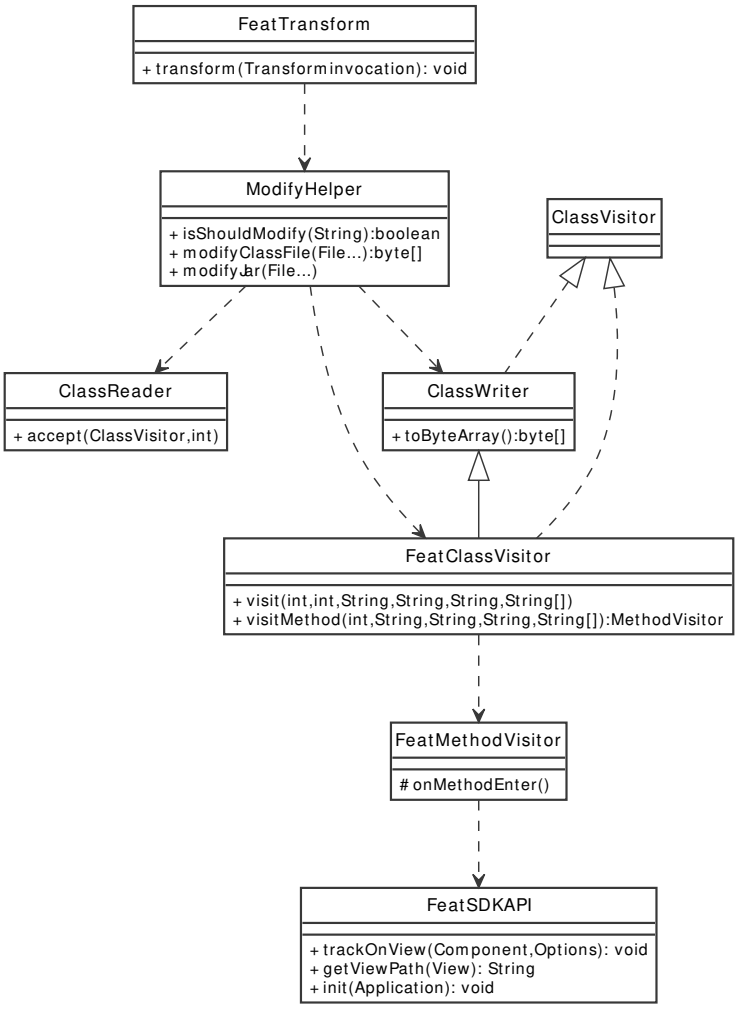

图 4.2: UI 控件覆盖检测插件关键类图

图 4.2是相关类的类图，在编译过程中 FeatTransform 只有唯一实例，ModifyHelper 帮助类均为静态方法，没有状态。对单个 class 文件的每次修改均会新

建 ClassReader、ClassWriter 和 FeatClassVisitor 三个对象，其中 FeatClassVisitor 和 ClassWriter 均继承于 ClassVisitor 类，FeatClassVisitor 中会保留 ClassWriter 的引用，用以生成修改后类的二进制数组。在修改过程中，FeatClassVisitor 会为每个方法新建 FeatMethodVisitor，所以只需要在此类中完成代码的插桩，即可确保所有想要插桩的对象均可完成插桩。

```
public static String getViewPath(View view) {
  if (view == null) return "";
  ViewParent viewParent = tmpView.getParent();
  // 默认 index 为 0
  int index = 0;
  // 如果 viewParent 是viewGroup，找到该类型 view 在 group 中的 index
  if (viewParent != null && viewParent instanceof ViewGroup) {
    for (int i = 0; i < viewParent.getChildCount(); i++) {
      View view = parent.getChildAt(i);
      if (child.getClass().isInstance(view)) {
        if (view == child) {
          return index;
        }
        index++;
      }
    }
  }
  // 递归调用 getViewPath
  // 返回形如 parent1[index]-parent2[index]-...-view[index] 的 ID
  return StrUtil.format("{}-{}[{}]",
    getViewPath(viewParent), view.getClass().getSimpleName(), index);
}
```

图 4.3: FeatSDKAPI 关键代码

插桩完毕之后，每个 UI 控件的点击事件都会调用 FeatSDKAPI 中的 trackOnClick 方法，并会传入控件的信息。插件需要记录控件的点击记录。为了识别不同的控件，本框架使用以下 4.3算法为 view 获取唯一 ID，通过将 View 的 ID 转换为 View Path 来保证测试工具点击的控件不会被重复记录。控件的动态插入以及删除等操作会影响测试的结果，为了减少对测试结果的影响，将会在每个 View 的后面加上 index 的属性，这个 index 属性是在当前 ViewGroup 里相同 Class 的序号。在生成控件的唯一 ID 之后，可以将当前 Activity 以及 View ID 打

印到系统的日志之中，配以 Plugin 的 TAG。将测试过程中的日志进行收集，在测试之后对日志加以分析即可获得测试工具的控件点击情况。

4.2　源码插桩变异模块

4.2.1　源码插桩变异模块概述

源码插桩变异模块可以将应用源码改造成为可用于系统测试的应用。

自动化测试工具的测试效果具体表现为检测应用缺陷的能力。因此系统需要将应用加以改造，使应用拥有分析测试工具测试效果的能力。即在自动化测试工具测试应用的同时，应用可以从多个维度对该工具的操作进行记录。通过对应用记录进行分析，即可获得测试工具的具体表现。为全面评估自动化测试工具的测试表现，需要准备大量不同类型的应用形成应用集。而且安卓应用迭代速度极快，由于手工扩充工作量较大，需要保证系统可以对应用集进行自动扩充。本模块实现了对应用的全自动插桩以及编译，仅需提供应用源码，即可获得可用于测试的相关应用。在插桩后可以对应用源码进行相关变异操作。由于 MDroid+ 只能生成相关变异，并不能提供变异的检测功能。系统对 MDroid+ 的变异模块进行改进，使 MDroid+ 会在生成的变异体之前打印相关信息。简而言之，此模块用于系统测试应用集的准备，只有全面的测试集才可以保证客观的测试结果。因此本模块对保证测试效果以及准确性起到不可或缺的作用。

4.2.2　源码插桩变异模块的设计实现

图 4.4是修改源代码以及配置文件用以接入插件的时序图（本文后续将插件插入结束称之为插桩结束）。在修改文件时，将具体的修改步骤封装为实现 Func 接口的方法传递到 FeatUtil 中，这样在进行修改操作时，首先工具类会调用 checkChangedFile 方法来检测之前此文件是否做过同样的操作，如果该文件没有执行过相同操作，则执行 runChangedFileFunction 以调用传入的方法。在修改结束之后调用 markChangedFile 对文件进行标记。以此保证文件不会被重复修改。随后调用 execGradleTasks 执行插件的上传操作，使插件在应用中可用。具体实现是通过调用 CommandUtil 来调用本地命令行工具，进而执行 Gradle 任务。在执行任务之后需要添加一些配置到配置文件中，通过 appendBeforeContent 方法即可在具体内容之前追加配置。由于插件需要在应用初始化的时候启用，系统采用替换 Application 类的方式。首先通过 FileUtil 的 copy 方法将配置初始化完毕的 Application 类拷贝至应用源代码之中。再通过 changeApplicationClass 修改原应用的配置，将系统的应用类改变为拷贝的类。具体实现是通过 getApplicationFile 获

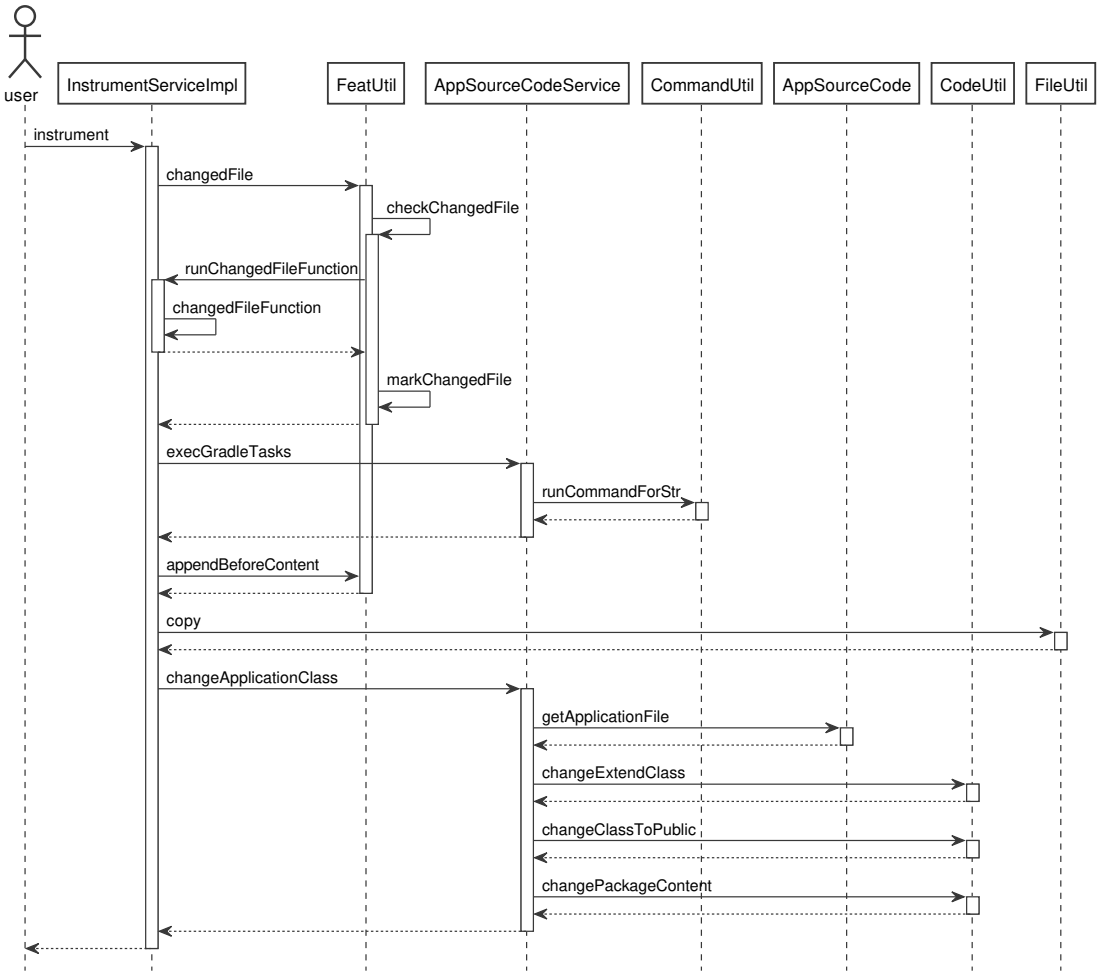

图 4.4: 插件接入时序图

取原工具的应用类文件，随后通过 changeExtendClass 将拷贝的 Application 的父类设置为原应用的 Application 类。还需要将原应用的类通过 changeClassToPublic 变为 public 类，避免继承相关问题。最后通过 changePackageContent 将 package 信息添加到新的应用类之中，并修改 AndroidManifest 文件中应用文件的配置。这样就完成了插件的导入。

图 4.5是变异的具体时序图，变异须在应用进行插桩之后。用户的请求通过 Controller 简单封装，将会调用 AppSourceCodeService 的 createMutation 方法，在进行变异之前会对当前源码进行编译。CommandUtil 通过命令行执行 gradle 的编译命令。在编译结束之后，会对返回的信息进行简单判断。如果编译失败，则停止编译等待管理员接入；如果编译成功，则进行下一步操作。变异前的准备工作分为两步：首先需要将应用生成的应用文件拷贝到系统的文件系统进行存储；随后需要将相关 AppPackage 的信息归档至数据库中。在准备工作完成之后，调用

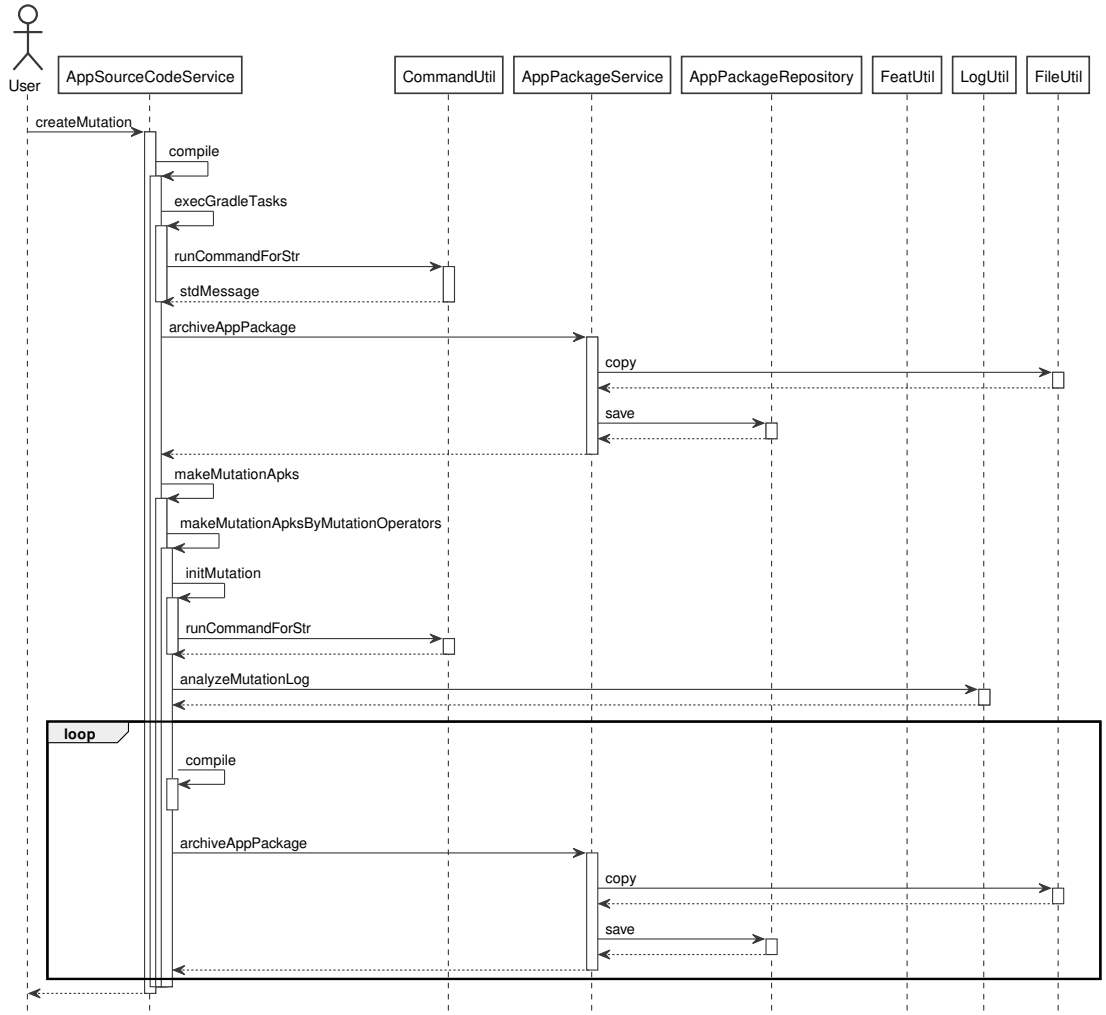

图 4.5: 变异时序图

makeMutationApks 方法生成变异。若用户提供了变异因子，则使用用户提供的变异因子进行变异。若用户没有提供变异因子，则随机选择变异因子进行变异。将变异因子传递进 makeMutationApksByMutationOperators 方法之后，该方法会调用 initMutation 方法，即启用 MDroid+ 工具对源码进行变异。MDroid+ 是命令行工具，所以 initMutation 需要调用 CommandUtil 执行命令以启用该工具。在变异操作执行完毕时将会有相应的日志文件生成。LogUtil 中的 analyzeMutationLog 方法可以对 MDroid+ 的日志进行分析，以获得变异的详细信息。在日志分析完成后，对生成所有的变异源代码进行循环编译操作。具体的执行方式是：首先在系统中拷贝一份应用，随后将应用中 app 模块中的源码以及 AndroidManifest 文件进行替换（因为有些变异是对 AndroidManifest 进行变异），然后对替换后的应用进行编译操作。若该应用编译成功，系统会将生成的应用进行归档。由于部分

编译失败的原因是缓存文件导致的，系统在编译失败的情况下会删除缓存文件再次编译。以上为生成一个应用以及其相关变异应用的具体流程。

表4.2是此模块的主要类，InstrumentServiceImpl 是 InstrumentService 接口类的具体实现，提供了插桩的具体逻辑。AppSourceCodeService 在此模块中主要负责对源码的具体改动，包括对源码的插桩、变异以及编译等操作，在源码编译后需要配合 AppPackageService 类对应用包文件进行归档。而 CodeUtil、LogUtil、CommandUtil、FileUtil 是此模块使用的工具类，在后续模块中同样也被大量使用，CodeUtil 负责代码修改等操作，在此次测试中负责了修改继承类等功能。LogUtil 在该模块中负责变异日志的解析工作。CommadUtil 在该模块中负责启动 Gradle 任务以及启动改进后的 MDroid+ 任务。FileUtil 在本模块中负责将相应文件拷贝到待修改的应用之中。

表 4.2: 插桩变异模块主要类

类名称	**分类**	**作用**
InstrumentService	接口类	提供插桩方法的接口类。
InstrumentServiceImpl	服务类	用来提供插桩的具体逻辑，提供插桩的服务，InstrumentService 的具体实现。与底层 Service 进行交互。
AppSourceCodeService	底层服务类	用于与 AppSourceCode 相关的具体改动，并与 AppPackageService 配合，完成源码的变异归档等错作。
AppPackageService	底层服务类	用于完成对 AppPackage 相关的业务逻辑，在此模块中使用了其归档的功能。
CodeUtil	工具类	用于修改 java 以及 kotlin 代码的工具类。
LogUtil	工具类	用于执行日志相关工作的工具类。
CommandUtil	工具类	用于执行命令以及提供日志的工具类。
FileUtil	工具类	用于执行文件相关操作的工具类。
AppSourceCode	数据类	包含了工具需要的源码的全部信息。
AppPackage	数据类	包含了工具需要应用包的全部信息。
MutationAppPackage	数据类	继承于 AppPackage，增加了变异相关的信息。

表 4.3是 AppSourceCode 的详细描述。表示可进行插桩以及变异的操作的基

本单位，为 AppSourceCodeService 提供对源码改动时需要的信息。

表 4.3: AppSourceCode 类详情列表

字段	含义	类型	备注
id	应用源码编号	String	主键，不为空
appFolderPath	应用路径	String	应用在系统中的路径
mutationChange FolderPath	源码路径	String	变异后的源码需要替换的路径
packageName	包名	String	对应 AndroidManifest 文件中的包名
manifestPath	文件路径	String	应用 AndroidManifest 文件的路径
appModuleName	模块名	String	应用的具体模块名

表4.4是 AppPackage 类的详细信息，提供信息的主要来源是由 aapt 获取的 Apk 的 dump 信息，此外还拥有后续测试操作所需要的信息，如含有对编译其源码的 AppSourceCode 的引用。AppSourceCode 类与 AppPackage 类是一对多的关系，即单个应用源码对应多个应用包，多个应用包中包括插桩应用和变异应用。类 MutationAppPackage 即为变异应用对应的 AppPackage 类，表4.5是 Mutation-AppPackage 类的详细信息表，该类继承于 AppPackage，除 AppPackage 原有的信息以外，此类保存了 MutationOperator 的信息，用以记录该变异应用在变异操作中使用的变异因子。

表 4.4: AppPackage 类详情列表

字段	含义	类型	备注
id	应用编号	String	主键，不为空
apkFilePath	应用路径	String	应用文件在系统中的路径
apkFileMd5	文件 Md5	String	应用文件的 Md5，避免对应用的识别错误
apkInfo	Apk 信息	ApkInfo	通过 aapt 文件获得的 Apk 具体信息
AppSourceCode	源码对象	AppSource-Code	编译该应用的对应源码对象

表 4.5: MutationAppPackage 类详情列表

字段	含义	类型	备注
mutationOperator	变异因子	Mutation-Operator	表明变异应用采用的变异因子

图4.6是该模块的核心类图，此模块的核心逻辑大部分由 AppSourceCodeService 完成，主要包括 Gradle 命令的执行、变异的生成以及对应用文件的修改。AppSourceCodeService 类中有 AppPackageService 的引用，用于对修改后的 AppPackage 实体进行归档，归档后的应用可作为后续测试的待测应用集。服务中的具体操作部分抽象到了工具类之中，通过对 FeatUtil、CommandUtil、LogUtil、FileUtil 以及 CodeUtil 的调用，对具体的操作进行实现。而 AppSourceCodeService 被实现 InstrumentService 接口的 InstrumentServiceImpl 类相关联，这种实现的好处是可以保证该模块后续的可扩展性，仅需实现对应接口即可扩展其它种类的插桩，以服务于后续新的测试项目。

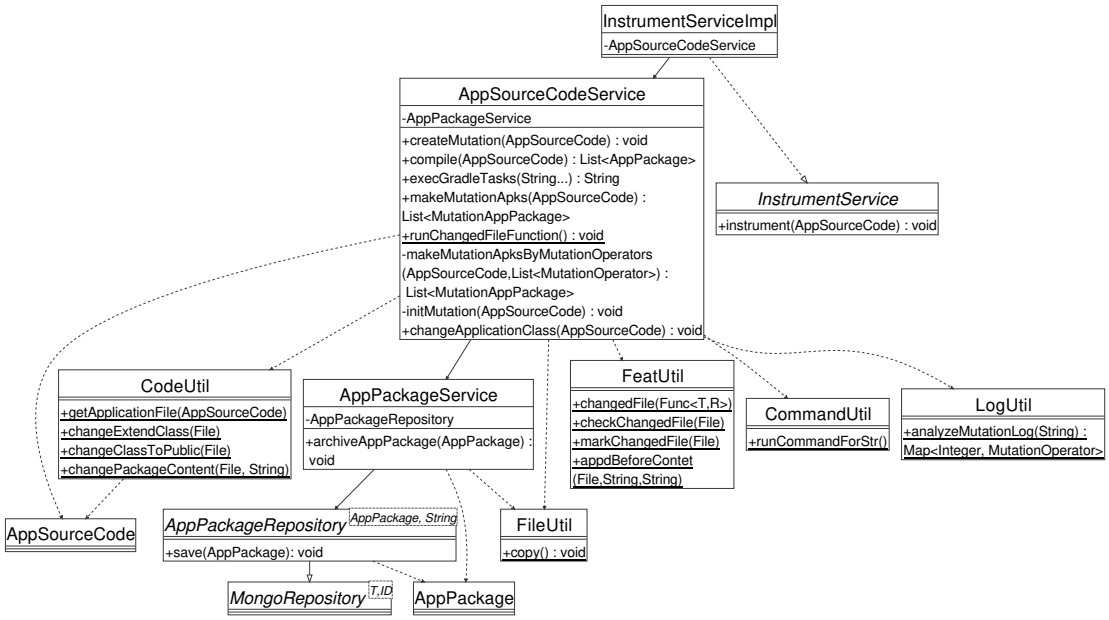

图 4.6: 源码插桩变异模块关键类图

由于篇幅限制，后续只引用了 instrument 方法以及 makeMutationApksByMutationOperators 方法作为本文的关键代码进行展示。其中初始化对象、获取属性以及后续收尾工作为避免代码过于冗余，没有展示出来。图 4.7是 instrument 的具体代码，该代码位于 AppSourceCodeService 类中。该方法通过修改、拷贝配置文件，将 UI 控件覆盖率插件以及 Jacoco 插件引入到待测应用之中，插件会在

编译过程中自动对代码进行埋点，使应用获得 UI 控件覆盖率检测以及代码覆盖率检测的能力。

```
@Override
public void instrument(AppSourceCode appSourceCode) throws
    FEATException {
  // 改变配置文件
  File setFile = FileUtil.file(appFolder, "settings.gradle");
  FeatUtil.changedFile(setFile, voids -> {/*.. 增加Module到项目中*/});
  try {
    // 执行 gradle 构建插件的任务
    appSourceCodeService.execGradleTasks(PROJECT_PLUGIN_MODULE +
        ":uploadArchives");
  } catch (IOException e) {
    // ..将错误包装后增加相应描述
  }
  // 改变配置
  File buildFile = FileUtil.file(appFolder, "build.gradle");
  FeatUtil.changedFile(buildFile, voids -> {/*.. 修改依赖*/});
  // 改变配置
  File appBuildFile = FileUtil.file(appFolder, "app/build.gradle");
  FeatUtil.changedFile(appBuildFile, voids -> {/*.. 修改依赖*/});
  File newApplication = FileUtils.getFile("componentCoverage",
      "FeatBaseApplication.java");
  // 拷贝应用
  newApplication = FileUtil.copy(newApplication, copyFolder);
  // 修改应用的 Application 文件
  appSourceCodeService.changeApplicationClass(newApplication);
}
```

图 4.7: instrument 关键代码

图 4.8的代码同样位于 AppSourceService 类中，该方法可以通过用户提供的变异因子，对选中应用进行变异操作。系统首先会对源码进行拷贝，以防止插桩或变异操作对源代码造成影响。在拷贝结束后使用拷贝的源码进行变异和编译的相关操作。最后对上述操作生成的 MutationAppPackage 进行归档，归档会保存 AppPackage 的相关信息以及应用相对应的原 AppSourceCode 的信息。

```
private List<MutationAppPackage> makeMutationApksByMutationOperators(
  AppSourceCode appSourceCode, List<MutationOperator>
      mutationOperators) {
  initMutation(AppSourceCode); // 初始化变异
  // ... 省略获取新建对象以及获取变量的代码
  // 获取变异log信息
  val mutationOperators = LogUtil.analyzeMutationLog(logFile);
  AppSourceCode copySourceCode = makeCopy(appSourceCode); /*
      拷贝原应用用以变异 */
  // 遍历生成变异的文件夹
  for (File file : mutationCodeFolder) {
    // 如果是文件夹，则内部是具体变异源码
    if (file.isDirectory()) {
      FileUtil.clean(copySourceFolder); // 清理文件夹
      FileUtil.copy(file, copySourceFolder, true); // 拷贝到文件夹
      copySourceCode.cleanApk(); // 清理之前生成的应用
      try {
        // 替换源代码后编译，并将应用归档
        copySourceCode.compile();
        List<AppPackage> apks = copySourceCode.getApks();
        // 将 AppPackage 转换为 MutationAppPackage
        apks.stream().map(apk -> new MutationApk(apk, mutationOperator));
        result.addAll(apks);
        apks.foreach(AppPackageService::archiveAppPackage);
      } catch (Exception e) {/* 增加错误信息，对错误包装*/}
    }
  }
  return result;
}
```

图 4.8: makeMutation 关键代码

4.3　测试模块

4.3.1　测试模块概述

测试模块是本系统的核心模块，本系统使用该模块实现对自动化测试工具的执行、监控、日志收集等功能。该模块可以实现对自动化测试工具的全自动测

试。在保证工具的正确执行的情况下，该模块将大量测试任务分解并进行异步执行，极大的提高了系统执行测试的效率。可以在自动化测试工具停止运行或没有正确测试应用的情况下主动杀死测试工具，有效减少了无效测试时间。而且具有较好的可拓展性，通过面向对象的设计，使测试工具仅需实现相关接口即可接入至系统中。在工具接入系统后即可拥有监控测试以及收集日志的能力。对于已经适配的自动化测试工具，系统可以在测试执行之前对设备环境进行部署；在测试之后也可以有效的收集测试生成的运行时文件，该文件用于后续的评估模块之中。简而言之，该模块提供了高效率、可容错、日志完备的自动化测试工具执行功能，同时可以支持对不同工具的定制化，具有较高的可用性。

4.3.2　测试模块的设计实现

该章节是对测试模块设计实现的详细描述，本模块是该工具的核心模块，在第三章以及对该模块的测试流程以流程图（图3.11）的方式进行了简要介绍。在本章同样会从时序图、主要类表、类详情列表、关键类图和关键代码几个方面，对具体测试进行解析。

为了避免时序图过于复杂，将分配测试任务与异步执行测试的具体步骤分至两张图上，图 4.9是分配测试任务的时序图，图 4.10为运行测试任务的时序图。

本文首先对分配测试任务的时序图进行详细解析。如图 4.9所示，测试由用户发起，前端以 Restful API 的形式向 TestController 发起调用，将测试模式、测试工具、待测 APK 和所选用的设备以参数的形式传递到后端，TestController 类通过调用 TestService 类中的 startTest 方法开始测试。首先通过 AppPackageService 的 getOutputFolder 获取该 APK 测试相关文件的输出文件夹，该方法是按照 APK 文件的 MD5 以及测试工具生成在系统中的唯一路径。随后将当前测试的 APK 文件通过 backupApkToFolder 方法对其进行备份。调用 startTest 的重载，该方法是为单个 APK 分派测试任务的具体方法。通过调用 DeviceService 的 getDeviceExectorService 方法，可以获得该设备的单例线程池。如果当前设备不存在单例线程池，则为该设备新建单例线程池，并存储在 DeviceService 之中。这样可以保证每一个设备同时只存在一个执行测试任务的线程池。如果当前设备存在单例线程池，则直接返回缓存中的单例线程池。随后通过调用 TestService 的 getOutPutFolder 方法，用以获取匹配 APK、测试工具和设备的输出目录，该目录是之前调用 AppPackageService 的 getOutPutFolder 方法所得目录的子目录。随后使用 JDK 中原生的 CompletableFuture 类中的 runAsync 方法，将测试线程封装后加入线程池执行队列中。使用该类的好处是可以在测试完成或者崩溃时自动进行回调，有利于测试完毕之后文件的清理以及收集工作。

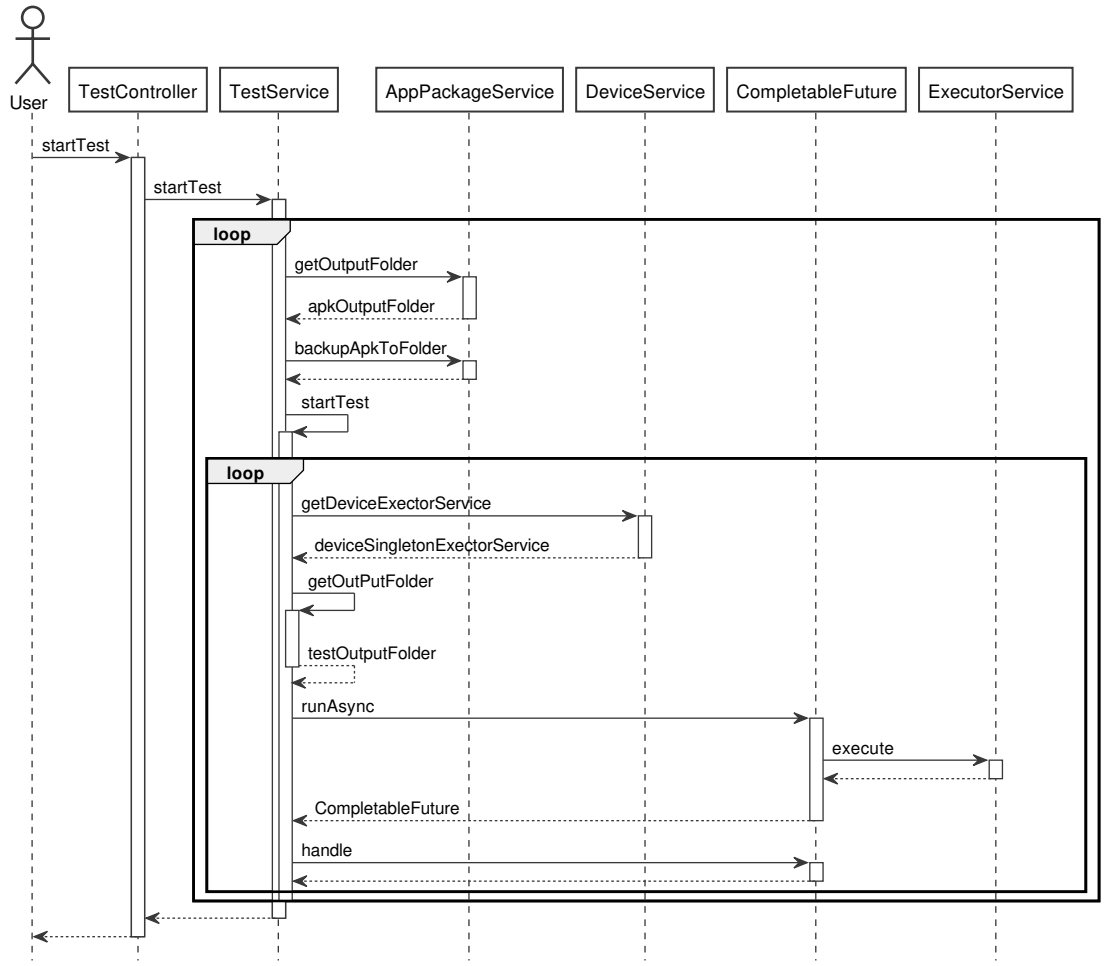

图 4.9: 分配测试任务的时序图

图 4.10是运行测试任务的时序图。由设备的线程池发起测试，即运行线程池中的匿名测试类，在图中使用 AnonymousTestThread 表示其相关操作。在测试之前，不同的测试模式和测试工具都需要进行环境准备、数据清理等操作，匿名测试线程分别调用 TestModeService 以及 TestToolService 的 beforeTest 对测试环境进行准备。beforeTest 一般都是使用 DeviceService 中的 pushFile 以及 delFile 方法对设备中的文件进行改动，使设备拥有测试需要的文件。而 DeviceService 中与文件相关的方法都是借助 CommandUtil 工具类实现的，通过命令行调用 ADB 以实现对手机文件的增删改查等功能。为避免图像过于复杂，在该图中将这些方法的具体时序省略。随后调用 DeviceService 的 installApk 方法，该方法会在安装的同时授予 APK 所有需要的权限。以保证在后续测试过程中应用不会有权限相关的弹窗，这样可以有效避免对自动化测试工具测试的影响。同时保证插桩后应用的运行时文件可以保存到设备中。随后调用 TestToolService 获得 TestThread

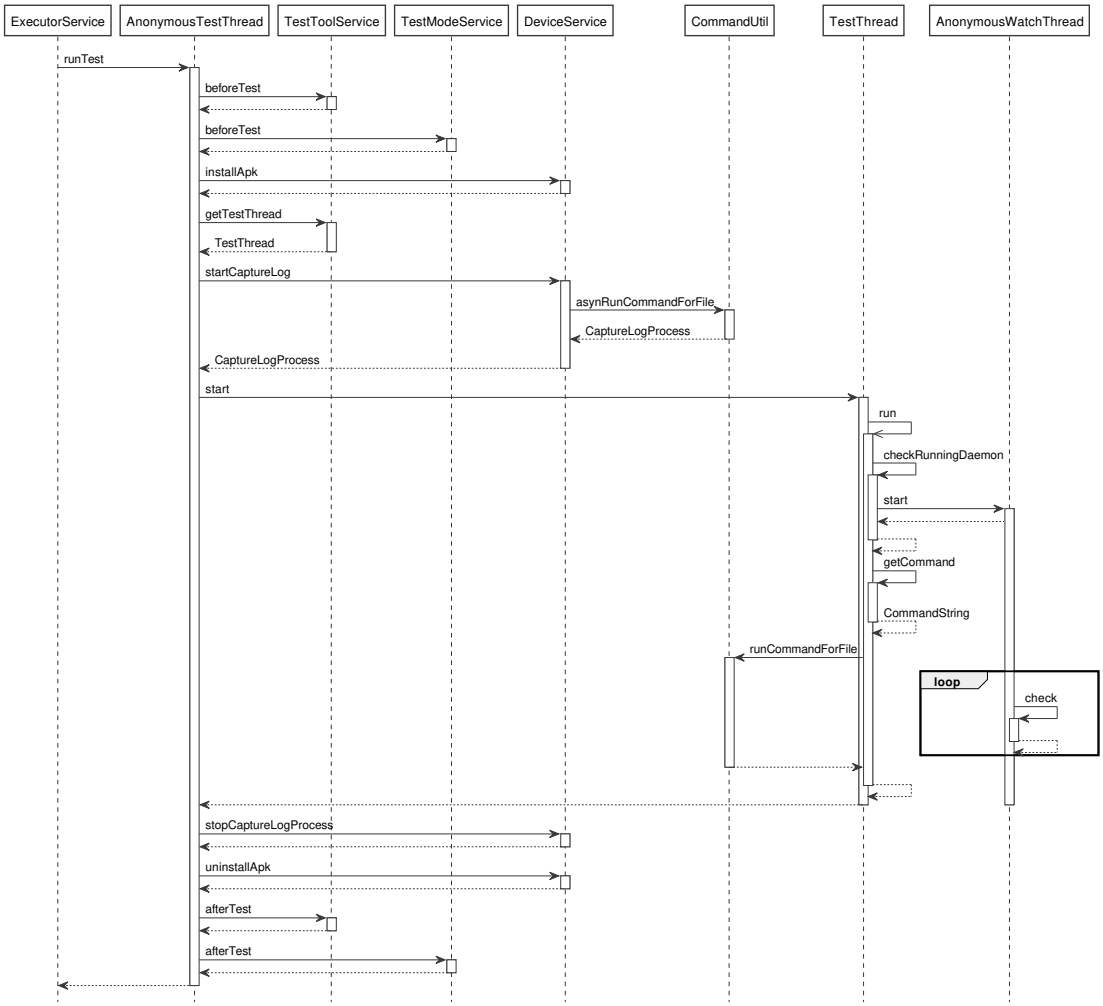

图 4.10: 运行测试任务的时序图

的实例，TestThread 是抽象类，每个测试工具都会实现自己的 TestThread。该类是具体执行测试工具的线程，本章后续将会详细介绍该类。通过 startCaptureLog 方法的调用可以获得异步存储 Logcat 日志的 Process，线程会持有该 Process，在测试执行结束之后将其关闭。自动化测试工具的 TestThread 在完成以上准备工作之后开始执行，首先测试线程会通过 checkRunningDaemon 方法启动监控线程。监控线程可以保证测试线程的正确执行，在测试停止或自动化测试工具没有正确打开待测应用等情况下，监控线程会主动杀死测试线程，以节省测试时间。测试线程首先通过调用该测试工具实现的 getCommand 方法，获得执行测试的命令。然后使用 CommandUtil 的 runCommandForFile，将测试工具的日志存储到 outputFolder 中。由于匿名测试线程被 CompletableFuture 封装，运行结束和被监控线程杀死都会执行后续的方法，首先调用 stopCaptureLogProcess 方法以杀

死记录手机日志的线程；随后调用 uninstallApk 方法，卸载测试的应用；最后调用 TestModeService 和 TestToolService 的 afterTest 方法，对测试的运行时文件进行收集，同时对测试生成的文件进行清理。

表 4.6内是该模块的主要类，其中 TestController 负责接收前端测试相关请求，并调用后端 Service 层的 TestService。TestService、TestToolService 和 TestModeService 为测试提供相应的服务。TestToolService 负责提供测试线程以及环境配置等服务。TestModeService 负责提供后续的评分以及数据收集等服务。TestService 是具体执行测试的业务逻辑，通过该服务可以将测试线程部署到测试的线程池之中，以保证测试的异步高效执行。DeviceService 是底层服务类，用于提供与设备相关的具体服务。TestBaseService 为服务类，实现该接口可以在测试前后实现文件收集或设备环境配置等自定义操作。有些自动化测试工具需要 Appium 的支持，本工具通过 AppiumManager 类在内部封装了包括指定接口新建 Appium 服务以及 Appium 进程的回收等相关操作。

表 4.6: 测试模块主要类

类名称	**分类**	**作用**
TestController	控制器类	对前端提供测试相关的 Restful API。
TestService	服务类	提供测试的具体业务逻辑。
TestToolService	服务接口	测试工具的相关服务接口。该接口的实现类为测试工具提供环境的准备及清理方法，以及可以提供测试工具的测试线程。
TestModeService	服务接口	测试模式的相关服务类。该接口的实现类在本模块中为不同测试模式提供测试前数据清理、测试后的数据手机等功能。
TestBaseService	服务接口	测试相关服务类，实现该接口可以在测试之前或者之后对设备进行操作。
DeviceService	底层服务类	提供设备的相关服务。如设备文件的增删改查、应用的安装删除等。
AppiumManager	管理类	提供 Appium 的管理服务。
TestThread	数据类	提供对测试工具的具体信息。
Device	数据类	提供测试设备的详细属性。

表 4.7是 Device 类的详细数据，其中属性 id 可供 ADB 调用设备时使用，属性 sdkVersion 为设备的 SDK 版本，部分操作在不同的 SDK 版本所执行的命令可能不同，因此将其记录。属性 timeZone 记录时区，用于分析日志时筛选时间用。剩于设备的具体信息用于后续评分模块，在计算碎片化得分时使用。

表 4.7: Device 类详情列表

字段	**含义**	**类型**	**备注**
id	设备编号	String	主键，唯一 ID，不可更改。
sdkVersion	版本号	Integer	应用文件在系统中的路径。
timeZone	系统时区	String	应用时区，用于筛选日志。
deviceModel	设备型号	String	设备的具体型号。
deviceName	设备名称	String	设备用户设置的的名称。
brand	设备品牌	String	
resolution	设备分辨率	String	

表 4.8: TestThread 类详情列表

字段	**含义**	**类型**	**备注**
id	任务编号	String	主键，唯一 ID，用于识别不同任务。
outputFolder	输出文件夹	File	系统输出文件夹，用于保存日志及运行时文件。
device	设备	Device	测试设备的具体信息。
appPackage	app 包	AppPackage	测试应用的具体信息。
testToolLog	测试工具日志	BaseLog	记录测试工具的运行时日志文件信息，以及测试相关信息。
deviceLog	设备日志	DeviceLog	记录设备的运行时日志，相比 BaseLog 增加 Timezon 信息。

表 4.8为 TestThead 类的详细数据，TestThread 类为单次执行自动化测试工具测试单一 AppPackage 的线程单位。每个线程均记录了任务编号、输出文件夹路径、设备详细信息、AppPackage 详细信息以及日志信息。在执行测试过程中需要提供以上信息以完成日志的保存等工作，在后续评估模块中需要对以上信息进行分析以计算具体得分。

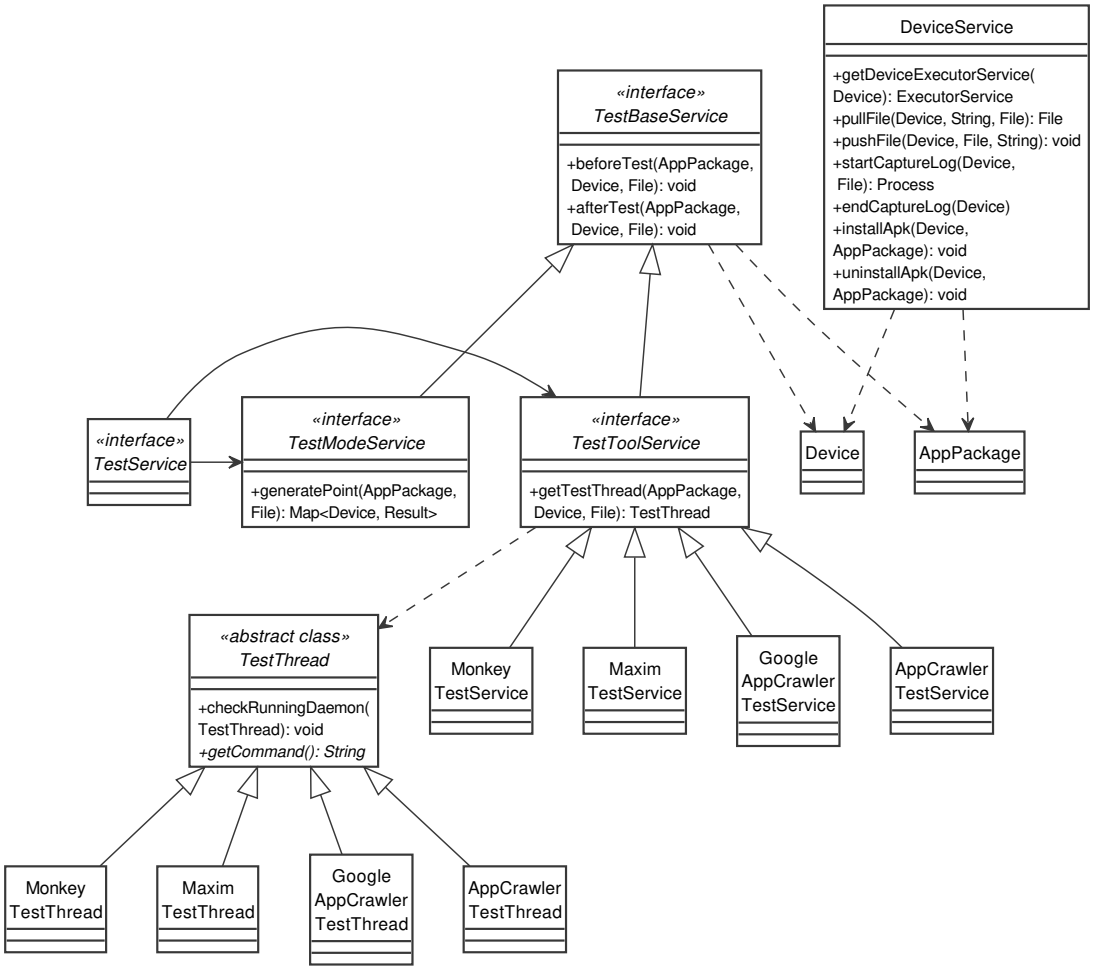

图 4.11: 测试模块关键类图

图4.11是测试模块的核心类图，该模块的主要测试逻辑由 TestService 完成，具体的测试工具执行、监控等工作则依赖抽象类 TestThread 和 TestToolService。如果接入具体的测试工具到该模块中，则需要实现具体的 TestToolService 类以及 TestThread 类。用户只需完成测试工具的执行部分，监控以及日志功能均由工具自动接管。这种设计的好处是可以有效的将自动化测试工具与本工具解耦，用户只需要关心如何完成自动化测试工具的具体执行以及环境配置。增加了系统的可扩展性。目前系统适配了五个自动化测试工具，分别是 Maxim、Monkey、

GoogleAppCrawler、AppCrawler 和 Fastbot。以上工具的适配相关源码可以提供给用户参考，用以后续待测工具集的扩展。TestModeService 为不同的测试模式提供待测 APK 以及评分方法，评分方法将在评估模块进行详细介绍。这种模式的设计可以方便对测试模式的扩展，新的测试模式只需提供相应 APK 以及得分方法就可以直接调用工具集成的所有测试工具。目前系统实现了一种测试模式。TestModeService 以及 TestToolService 均实现了 TestBaseService，通过对该接口的实现可以在测试执行前后进行环境配置以及运行时文件收集等自定义操作，用户通过调用底层服务 DeviceService 的 API，可以实现与设备相关的具体操作。

```
@Override
public void run() {
  super.run();
  // 配置 deviceLog 的时区
  TimeZone timeZone = TimeZone.getTimeZone(device.getTimeZone());
  deviceLog.setLogTimeZone(timeZone);
  // 记录测试开始时间，用于后续筛选Log
  long current = DateUtil.current();
  deviceLog.setStartLogTime(current);
  testToolLog.setStartLogTime(current);
  try {
    checkRunningDaemon(this); // 运行监控线程
    // 运行自动化测试工具
    CommandUtil.runCommandForFile(FileUtil.file(testToolLog.getLogFilePath()),
        getCommand());
  } catch (IOException e) {//... 错误处理
  } catch (InterruptedException e) {//... 错误处理
  }
}
```

图 4.12: TestThread 关键代码

图 4.12是 TestThread 的关键代码，其中省略了部分初始化及错误处理的代码。首先将设备信息记录在 deviceLog 类中，随后将开始时间记录在 Log 类中。运行监控线程后，可以通过 getCommand 方法获取自动化测试工具的运行命令。使用命令行工具运行该命令，同时将日志存储至测试工具的日志文件中。

图 4.13是 TestToolService 的范例，此处是适配 AppCrawler 自动化测试工具的完整服务类。由于市面上的部分工具依赖于 Appium，该工具也是如此。所以

```java
@Service
public class AppCrawlerService implements TestToolService {
  @Autowired AppiumManager appiumManager;
  // 使用 ThreadLocal 保证线程隔离。
  ThreadLocal<Integer> port = new ThreadLocal<>();
  Map<Integer, AppiumDriverLocalService> portMap = new HashMap<>();
  @Override  // 将具体测试参数配置到 AppCrawler 的 Thread 之中。
  public TestThread getTestThead(AppPackage appPackage, Device device,
      File outputFolder) {
    return new AppCrawlerThread(device, appPackage, outputFolder,
        port.get());
  }
  @Override
  public void beforeTest(AppPackage appPackage, Device device, File
      outputFolder) {
    AppiumDriverLocalService server;
    synchronized (appiumManager) { // 保证线程安全
      port.set(NetUtil.getUsableLocalPort()); // 获取随机端口
     // 开启 appium，将 appium 日志持久化
      server = appiumManager.startServer(
             port.get(), device, FileUtil.file(outputFolder,
                APPIUM_LOG_FILE_NAME));
    }
    portMap.put(port.get(), server);
  }
  @Override
  public void afterTest(AppPackage appPackage, Device device, File
      outputFolder) {
    Integer integer = port.get();
    synchronized (appiumManager) { // 关闭 appium
      AppiumDriverLocalService appiumService = portMap.remove(integer);
      if (appiumService != null) appiumService.stop();}
  }
}
```

图 4.13: TestToolService 代码

本系统适配了 Appium 的管理模块。Appium 与具体测试相关，因此将 Appium 管理模块归并到此模块之中。通过 AppiumManager 可以新建 Appium 的实例，在新建 Appium 时可对其端口、udid、日志文件等参数进行配置。测试工具服务类可以在执行测试之前对测试工具的运行环境进行配置，在测试之前匿名测试线程会调用此类的 afterTest 方法。AppCrawlerService 在测试之前为 Appium 找到空闲端口，并为设备提供 Appium 服务，随后将端口号存储在 ThreadLocal 之中，借助 ThreadLocal 类可以保证每个线程在运行时都可以获得该线程对应的 Appium 所用的端口号。将 AppiumDriverLocalService 同时存储在该服务中，以用于测试结束后停止 Appium 服务。开始测试时会运行 getTestThread 方法所提供的线程，该线程类也需要为不同的测试工具提供不同的实现。具体的执行逻辑即为上文中图 4.12展示的逻辑。测试服务类在测试结束之后会调用 beforeTest 方法，可以提供将设备中的运行时文件归档、还原测试环境等服务。由于 beforeTest 方法中创建了 Appium，因此需要在 afterTest 方法中杀死该 Appium。为了确保端口号的正确使用，在停止 Appium 以及删除服务中 Appium 存档时，代码需要使用 synchronized [40] 关键字，以避免多线程时可能产生的端口重用等问题。

4.4　测试结果评估模块

4.4.1　测试结果评估模块概述

该模块也是系统的核心模块之一，使用该模块可以对同种测试工具的单次测试任务或多次测试任务的测试结果进行评估。在系统执行完测试之后，用户提交评估要求，该模块对测试模块的测试的运行时文件进行分析（如 Jacoco 生成的 coverage.ec 文件、设备生成的 Logcat 文件），单次分析可以从异常、代码覆盖率、变异、UI 控件覆盖率四个方面对自动化测试工具得出评价。随后对在不同平台、不同屏幕参数下的测试效果进行综合分析，可以得到自动化测试工具对安卓碎片化的支持情况，最后将所有维度的得分均归档至数据库之中。

4.4.2　测试结果评估模块的设计实现

图 4.14是获取测试结果的时序图。用户通过 Restful API 调用 AnalysisController 的 generateResult 方法，可以通过提供测试 id 生成该次测试的结果。在获取了单个 Apk 在不同 Device 上具体的 TestLog 的集合 Map 后，首先将会获取该测试工具单个 Apk 的最终得分，在 AnalysisService 里面封装了生成 TestApkResult 的 generatePoint 方法，该方法首先会调用 TestModeService 类中的 generatePoint 方法，获得该测试模式的单次测试的评分（指测试工具在单个设备上测试单个

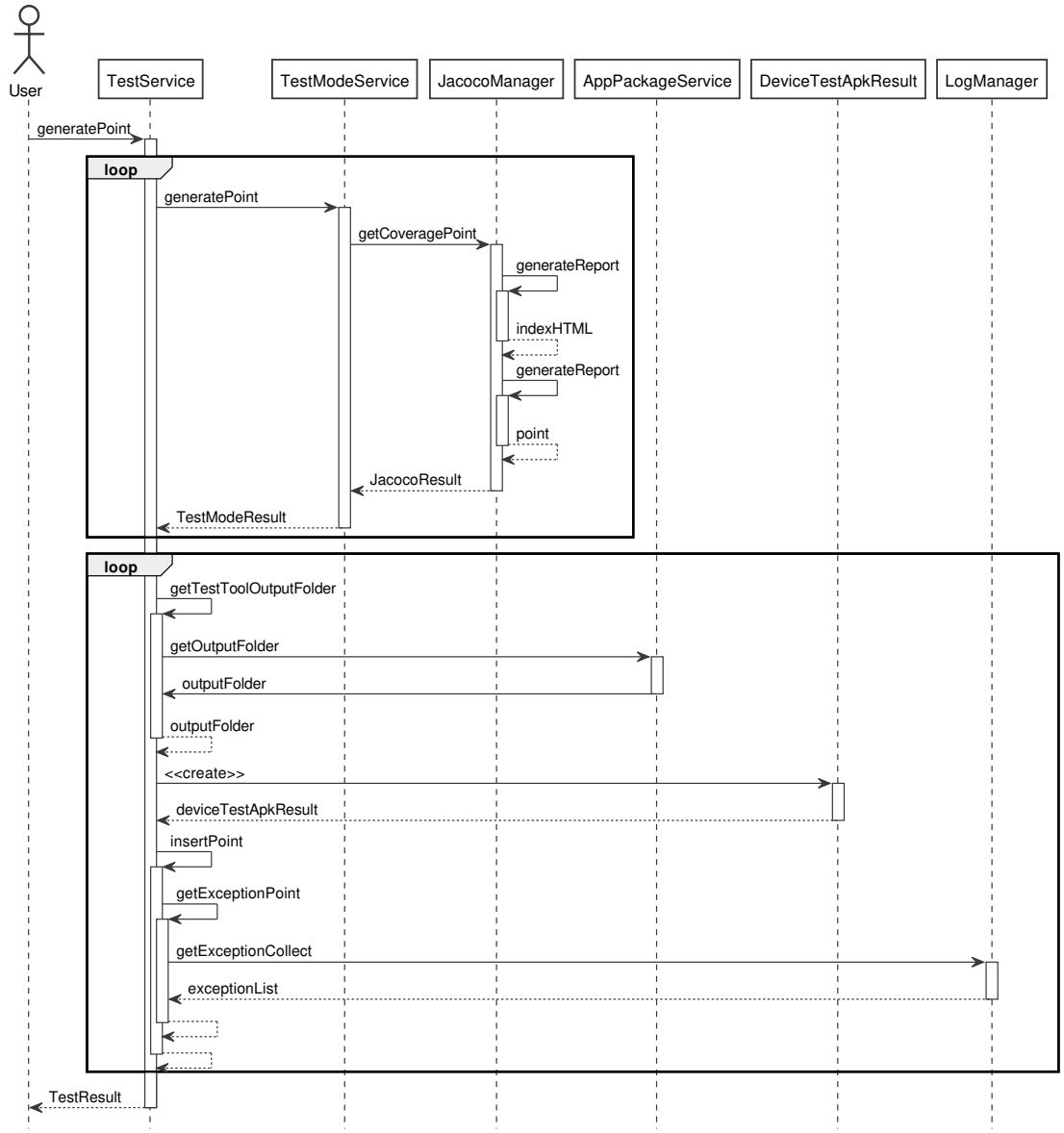

图 4.14: 获取测试结果的时序图

项目的得分)，在本项目中，测试模式的得分为 Jacoco 的覆盖率得分。所以会通过调用 JacocoManager 以生成 Jacoco 的测试报告。Jacoco 测试报告的具体生成是通过分析应用在设备上生成的 coverage.ec 文件得到的，通过该文件以及编译插桩后的 class 文件可以得到覆盖率的具体得分。Jacoco 的报告会生成在输出文件夹里。报告中的得分在 index.html 上面显示，在此处通过 Jsoup 对 html 页面进行分析，然后使用正则表达式对页面进行正则匹配，最后即可获得 Jacoco 包含指令覆盖率、分支覆盖率、圈覆盖率、行覆盖率、方法覆盖率和类覆盖率的完整得分。随后根据用户的具体配置，将分数按照比例加权即可获得代码覆盖率

得分。DeviceTestApkResult 为单个设备上测试单个项目的综合得分，其中包含了测试模式的得分结果。在得到 TestModeResult 的集合之后，需要将其集成到 DeviceTestApkResult 中。集成后即可获得 DeviceTestApkResult 的集合，随后需要将异常得分添加到 DeviceTestModeResult 之中。获取异常得分的具体逻辑是通过调用 AnalysisService 的 getExceptionPoint 方法实现的。该方法通过 LogUtil 的 getExceptionCollect 方法获得所有异常的统计。系统会为每个不同的 AppSourceCode 维护一个自己的异常列表。

表 4.9: 测试结果评估模块主要类

类名称	**分类**	**作用**
AnalysisController	控制器类	对前端提供结果分析相关的 Restful API。
AnalysisService	服务类	提供结果分析的具体业务逻辑。
TestModeService	服务接口	测试模式的相关服务类。该接口的实现类在本模块中为其测试模式提供评分功能，并以 TestModeResult 的形式返回。
JacocoManager	管理类	负责 Jacoco 相关报告的生成。
ExceptionManager	管理类	负责提供异常相关的管理服务。
LogUtil	工具类	提供日志相关的分析处理工作。
DeviceTestApkResult	数据类	提供单个设备对单个 APK 文件的自动测试结果。
TestApkResult	数据类	提供多个设备对单个 AppPackage 的自动测试结果。
TestAppResult	数据类	提供多个设备对同个 AppSourceCode 的多个 AppPackage 的测试结果，其中包含异常的自动测试结果。
TestResult	数据类	提供多个设备对多个 AppPackage 的自动测试结果。
MutationAppResult	数据类	提供同样 AppSourceCode 的 AppPackage 的异常测试结果。
MutationFinalResult	数据类	提供测试工具异常测试的最终的测试结果。
TestModeResult	数据类	提供测试模式的测试结果。

表 4.9是测试结果评估模块的主要类，AnalysisController 用以提供给用户与测试结果相关的 API，由 AnalysisService 来提供与结果相关的具体逻辑。具体测试模式所需的逻辑都封装在 TestModeService 里面。通过管理类 JacocoManager 可以对 Jacoco 的报告生成等底层功能进行封装，使测试与 Jacoco 相关逻辑解耦。LogUtil 可用于对 Logcat 的日志进行解析，以提取出不同的 Exception 类，用以结果的统计中。该模块中最为核心的部分即为数据类的设计，依照不同的测试粒度将不同层级的测试结果封装成不同的数据类，粗粒度的数据类依赖细粒度的数据类结果。向粗粒度数据类中填充细粒度的数据类，粗粒度数据类会自动调整其加权平均的结果。下文将会对相关数据类进行详细介绍。

表 4.10: DeviceTestApkResult 类详情列表

字段	**含义**	**类型**	**备注**
id	结果编号	String	主键，测试结果的唯一 ID。
testThreadId	线程编号	String	当前测试对应的具体线程编号。
device	设备	Device	拥有测试时设备的具体信息。
testToolType	测试工具	TestTool	可以记录当前测试所使用的测试工具信息。
testModeType	测试模式	TestMode	记录当前测试所使用的测试模式信息。
startTime	开始事件	Long	当前测试的开始时间。
finalPoint	最终得分	Double	当前测试得分加权后的最终得分。
exceptionPoint	异常得分	Double	当前测试的通过分析日志得到的异常得分。
testModeResult	测试模式结果	TestMode-Result	当前测试对应测试模式的特定测试结果。

表 4.10是自动化测试工具在单设备上对单 AppPackage 测试的结果。其中包含了测试线程的 id，由于测试线程与 DeviceTestApkResut 是一对一的关系，保存该属性可以避免测试结果的重复计算。该类同时记录了设备、测试工具、测试模式、开始时间等测试相关的信息，可以方便后续结果的加权统计。该类的 testModeResult 为不同测试模块的结果，具体的结果需要实现 testModeResult 类。

使用这种设计可以有效增加测试结果的可扩展性。

表 4.11: TestApkResult 类详情列表

字段	含义	类型	备注
id	结果编号	String	主键，测试结果的唯一 ID。
appPackageId	应用编号	String	当前测试对应的具体应用的编号。
deviceTest-ApkList	设备测试结果集合	List<Device-TestApk-Result>	对应 AppPackage 在不同 Device 上测试的结果列表。
testToolType	测试工具	TestTool	
testModeType	测试模式	TestMode	
finalPoint	最终得分	Double	当前测试得分加权后的最终得分。
averageTest-ModeResult	测试模式结果	TestMode-Result	当前对应测试模式的特定测试结果的平均值。
average-ComponentPoint	UI 控件得分	Double	分析 DeviceTestApkRe-sult 得到的 UI 控件得分的平均值。
average-ExceptionPoint	异常得分	Double	分析 DeviceTestApkRe-sult 得到的异常得分的平均值。

表 4.11是代表在本次测试中，不同设备对同一 AppPackage 的测试结果汇总。该类中拥有单设备测试结果的集合，对单设备测试结果的得分部分进行求和取平均值，可以得到多设备情况下对同一应用的测试得分。通过扩展测试的设备集，即可使得分有效体现自动化测试工具对安卓碎片化的支持。

表 4.12: TestAppResult 类详情列表

字段	含义	类型	备注
id	结果编号	String	主键，测试结果的唯一 ID。
appSource-CodeId	应用源码编号	String	当前测试对应的具体应用源码的编号。

testApk-ResultList	应用测试结果集合	List<Test-ApkResult>	对应 AppSourceCode 的不同 AppPackage（包括变异应用）的结果列表。
testToolType	测试工具	TestTool	
testModeType	测试模式	TestMode	
finalPoint	最终得分	Double	
average-ExceptionPoint	异常得分	Double	
average-ComponentPoint	UI 控件得分	Double	
averageTest-ModeResult	测试模式结果	TestMode-Result	
mutation-AppResult	异常测试结果	Mutation-AppResult	该 AppSourceCode 的异常得分。

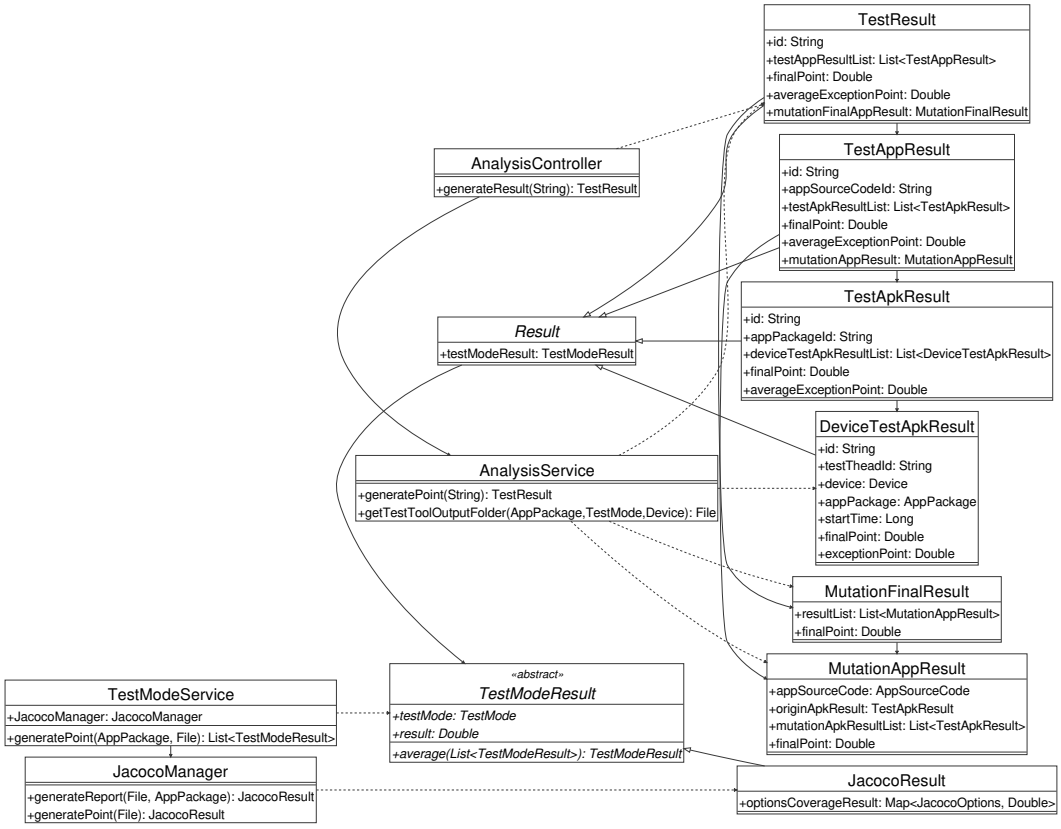

图 4.15: 测试结果评估模块关键类图

表4.12是测试工具对单个 AppSourceCode 测试的结果，若测试应用集中不包含变异应用，则 TestAppResult 与 TestApkResult 结果相同，TestAppResult 相较 TestApkResult 只是增加了对 appSourceCodeId 的记录；在包含变异测试的情况下，该数据类中将会包含 MutationAppResult 类，MutationAppResult 类是变异评估的具体结果，在此类中可以为空。该类由于需要记录应用变异测试的结果以及变异应用的相关信息，因此与异常得分的展示方式不同。以上部分为类的详细介绍，由于 TestResult 与 TestAppResult 的类结构较为重复，所以在此省略。

图 4.15是核心类的类图，其中类 TestResult、TestAppResult、TestApkResult 以及 DeviceTestApkResult 中存在链式引用，TestResult 是测试工具最终的测试结果，其中包含所有的 TestAppResult，TestResult 中的具体得分是所有测试结果的汇总。而这些类都继承于抽象类 Result，为了避免图像过于冗余，数据类省略了部分属性。这些数据类都由 AnalysisService 类生成，一些具有普适性的结果就在该类之中生成，如不需要插桩即可获得的异常得分。该类被 AnalysisController 类调用。而具体测试模块的评分将会通过 TestModeService 的 genereatePoint 方法生成。UI 控件覆盖率的得分通过分析使用 UI 控件覆盖插件的应用运行时日志即可生成，而代码覆盖率则需要引用第三方类库，在使用第三方库生成报告后对报告进行分析才可获得。系统将 Jacoco 的报告生成功能封装到 JacocoManager 之中，JacocoManager 可以通过生成报告分析报告以得 JacocoResult，该结果类即为 TestModeResult 的子类。系统对 MDroid+ 应用进行修改，会在变异位置打印日志，仅需分析设备日志即可获得相应变异得分。

图 4.16是 AnalysisService 中 generatePoint 方法的核心代码，为避免过于冗长，图中省略了获取数据的部分代码。首先对拥有所有被测 AppPackage 对象的队列进行遍历，随后为每个 AppPackage 对象的测试任务生成对应的 TestApkResult 对象。更详细的 DeviceTestApkResult 对象的生成过程封装在 generatePoint 方法之中，后文将会介绍此方法代码的详细逻辑。生成 TestApkResult 对象需要为 generatePoint 方法提供所需要的参数，generatePoint 方法需要测试工具类型、测试模式类型、被测 appPackage 以及键为 Device 值为测试日志的 Map 作为参数。该 Map 可以通过处理 MongoDB 中存储的 testThread 信息获得。generatePoint 方法会返回 TestApkResult 对象，并将其存储到 appResultMap 之中。随后会使用先前通过 AppSourceCode 列表进行汇总的 TestApkResult 列表作为参数，生成对应该 AppSourceCode 对象的 TestAppResult 对象，并将 TestAppResult 对象存储到 HashMap 之中存储，用以后续 TestResult 的生成。最后将该 HashMap 中所有 TestAppResult 结果传入 TestResult 对象之中，即可获得最终的测试结果。测试结果由 DeviceTestApkResult 到 TestApkResult 再到 TestAppResult 逐步汇总而来。

```
// 该 Map 为每个 AppSourceCode 收集所有测试结果
Map<AppSourceCode, List<TestApkResult>> appResultMap;
for (AppPackage appPackage : allAppPackage) {
  if (!appResultMap.containsKey(appSourceCode))
   appResultMap.put(appSourceCode, new ArrayList<>());
  // 获取当前 AppPackage 测试时信息
  List<TestThread> testThreads = testThreadRepository.
    findTestThreadByAppPackage(appPackage);
  // 按照 Device -> Log 的形式一一对应形成 Map
  Map<Device, TestLog> collect = testThreads.stream().collect(
       Collectors.toMap(value -> value.getDeviceLog().getDevice(),
           value -> value.getDeviceLog()));
  // 将 generatePoint 生成的单次结果填入结果集中
  appResultMap.get(appSourceCode).add(generatePoint(testToolType,
      testModeType, appPackage, collect));
}
HashMap<AppSourceCode, TestAppResult>
    appSourceCodeTestAppResultHashMap = new HashMap<>();
// 对单个 AppSourceCode 进行循环处理，以生成 TestAppResult
for (Entry<AppSourceCode, List<TestApkResult>> entry :
    appResultMap.entrySet()) {
  AppSourceCode key = entry.getKey();
  List<TestApkResult> value = entry.getValue();
  appSourceCodeTestAppResultHashMap.put(key, new TestAppResult(key,
      value, testModeType, testToolType));
}
// 生成最终结果
TestResult testResult =
   new TestResult(appSourceCodeTestAppResultHashMap, testModeType,
       testToolType);
```

图 4.16: 多应用 generatePoint 关键代码

图 4.17是获取单个 AppPackage 在设备集群上测试结果的方法，该方法首先为每个设备分别获取其对应测试工具的得分。具体是通过调用 modeServiceMap 获取对应的测试模式的服务类；随后调用其具体的 genereatePoint 方法以获取该次测试的工具得分；在获取测试工具得分后，为每个设备分别创建 DeviceTestApkResult 结果对象；在获得结果对象后再将先前获得的测试模式得分填充其中；最

```
@Override // 该方法为该 AppPackage 获取所有设备层级的得分
public TestApkResult generatePoint(TestTool testToolType, TestMode
    testModeType, AppPackage appPackage, Map<Device, TestLog>
    deviceTestLogMap) {
  // 调用测试模式的方法以获得该模式的结果
  Map<Device, TestModeResult> testModeResults =
      modeServiceMap.get(testModeType).generatePoint(appPackage,
      getTestToolOutputFolder(appPackage, testToolType));
  // 获取所有设备单次执行测试工具测试该 AppPackage 测试结果
  // 并将上面获得的测试模式的结果填充其中
  Map<Device, DeviceTestApkResult> collect =
      testModeResults.entrySet().stream().collect(Collectors.toMap(
      Entry::getKey, entry -> {
        DeviceTestApkResult deviceTestApkResult = new
            DeviceTestApkResult(entry.getKey(), testToolType,
            testModeType, appPackage,
            deviceTestLogMap.get(entry.getKey()).getTestToolLog(),
            deviceTestLogMap.get(entry.getKey()).getDeviceLog());
        deviceTestApkResult.setTestModeResult(entry.getValue());
        return deviceTestApkResult;}));
  for (DeviceTestApkResult it : collect.values()) {
    try {
      it.setExceptionPoint(logUtil.getExceptionPoint(it.getLogcatLog()));
    } catch (IOException | ParseException e) {/* 处理异常 */}
  } // 填充异常得分
  return new TestApkResult(collect, testModeType, testToolType,
      appPackage);
}
```

图 4.17: 单应用 generatePoint 关键代码

后填充测试的异常得分。异常得分是通过 LogUtil 直接获取的，该工具类通过分析运行时设备生成的日志，将异常抽象成为对象，随后进行去重对比，最后得到异常结果得分。随后将所有的 DeviceTestApkResult 填充到 TestApkResult 之中，TestApkResult 使用 Kotlin 的 data 类 [32]，其代码如 4.18所示。该类可以直接通过提供的 DeviceTestApkResult 结果归纳得到平均值。结果类均使用 Kotlin 进行实现，主要原因是因为设计数据的计算，Kotlin 相较 Java 而言封装了较多计算相关方法，可以较为简便的求和、求平均。而 Kotlin 代码可以直接在 JVM 上运行，

同时 Spring 框架也支持 Kotlin 语言，可以较为简便的接入先有框架之中。Kotlin 为用户提供了专用的 data 类，本系统使用 data 类作为结果的存储对象。采用 data 类相较 Java 代码而言可以减少很多代码量，理解起来更为简明扼要。

```
data class TestApkResult(var testApkList: List<DeviceTestApkResult>,
    var testMode: TestMode, var testTool: TestTool, var appPackage:
    AppPackage) {
  // 新建时自动获取得分的平均值
  var finalPoint = testApkList.map { result -> result.finalPoint
      }.average()
  var averageExceptionPoint = testApkList.map { value ->
      value.exceptionPoint }.average()
  var averageTestModeResult: TestModeResult? =
      testMode.result.average(deviceTestApkList.map { value ->
      value?.testModeResult })
}
```

图 4.18: TestApkResult 关键代码

4.5　本章小结

本章将本系统的三个核心模块（源码插桩变异模块、测试模块、测试结果评估模块）以及一个核心插件（UI 控件覆盖率检测插件）进行了详细描述。以上模块均以概述、时序图、核心类、类图、核心代码的顺序进行了阐述，并对关键类进行了详细介绍。

第五章　安卓自动化测试工具评估系统的测试与案例分析

本章主要介绍安卓自动化测试工具多维评估系统的系统测试与具体案例分析，对前文介绍的需求、设计进行测试评估，并对案例进行详细分析，同时结合前端页面对测试流程以及测试结果进行展示。

5.1　系统测试

5.1.1　测试目标

本章对该环境中运行的系统进行有效性测试，为该系统中的模块分别设计测试用例，并在此环境中进行测试，包括测试该系统中插桩插件、执行测试任务、对测试效果进行评估等功能的可用性。对系统进行有效性测试可以保证系统的质量。

5.1.2　测试环境

表 5.1: 系统测试环境

参数	参数详情
云服务器	Azure 云服务器
操作系统	Ubuntu 18.04-LTS 64
CPU	2
内存	8G
数据库	MongoDB 4.0
容器	Docker 0.19.0
JDK 版本	1.8.0_275
Node 版本	14.16.0-LTS
Appium 版本	1.18.1
ADB 版本	1.0.39
安卓设备品牌	三星、华为、小米、VIVO、一加

表 5.1为该系统进行系统测试的环境，系统的前后端均已 Docker 容器化的形式部署在云端之上。由于 Appium 需要依赖 Node 环境，因此对 Node 环境进

行部署。系统所使用的主要软件版本均在表格中明确描述。

5.1.3　功能测试

功能测试是一种确保系统质量的测试手段，通过测试用例对系统进行黑盒检测。本节根据前文中提到的功能性需求进行测试用例的设计与测试。

表5.2中的测试步骤为完成应用插桩变异的全过程。首先由用户对应用进行上传，随后系统根据用户的选择对源码进行修改，具体包括对应用的插桩、根据变异因子进行变异、对变异后的应用源码进行编译等操作，在编译完成之后应用进入待审核的模块，即完成该测试用例的测试。通过执行该测试用例可以实现对该工具的源码插桩变异模块的测试。

表 5.2: 完成应用插桩和变异的测试用例

描述项	**说明**
用例编号	TC1
用例名称	完成应用插桩和变异
用例描述	测试人员在系统中新建一个待插桩的应用，并完成相应操作。
测试步骤	1. 测试人员登录系统，选择上传待插桩应用的源码。 2. 测试人员对上传成功的应用选择插桩。 3. 测试人员选择变异因子，系统开始对源码进行变异，在变异之后分别对其进行编译。
预期结果	1. 待审核应用列表中出现该应用。 2. 管理员可以对该应用进行审核。

表 5.3中的测试步骤是审核上传插桩变异后代码的流程，该测试用例同样为代码插桩以及变异模块的评估。在管理员登录系统之后，可以在审核任务列表中看到并下载应用源码以及打包好的应用文件。在完成审核之后可以对审核任务进行处理，点击审核成功按钮即可将该应用集自动转移到可测应用集之中，同时完成该测试用例的执行。

表 5.3: 审核插桩变异的应用的测试用例

描述项	**说明**
用例编号	TC2
用例名称	审核插桩变异的应用

用例描述	管理员为插桩变异后的应用进行审核，保证其可执行性与测试可用性。
测试步骤	1. 管理人员登陆系统，进入任务审核页面。 2. 审核人员可以下载插桩后的源码以及编译后的应用。 3. 审核人员根据手工测试结果完成审核流程。
预期结果	1. 审核通过，应用将出现在测试可用应用集中。 2. 审核状态改变为审核完成，用户可以新建测试并选取此应用。

表 5.4的测试步骤是执行完整测试任务的流程，该流程的完整过程包括用户的登录、用户对配置的选择、用户开始测试。在配置过程中选择不同的配置，可以对本工具所适配的自动化测试工具进行测试。通过对设备执行情况、运行时日志以及归档数据库的观察分析，可以得知自动化测试工具的运行状况。对以上状态进行分析，可评估本系统的测试模块的可用性，同时可以检测自动化测试工具的适配情况。

表 5.4: 执行测试任务的测试用例

描述项	**说明**
用例编号	TC3
用例名称	执行测试任务
用例描述	用户提交一个对于自动化测试工具的测试任务，系统执行测试之后将测试数据归档。
测试步骤	1. 用户登陆系统，新建测试。 2. 为测试配置测试工具、测试模式、测试设备以及选择测试的应用。 3. 用户选择之后系统开始测试。
预期结果	1. 用户选择的设备开始自动化测试。 2. 所有的待测应用均在所有设备上被自动化测试工具执行过测试。 3. 测试完毕之后任务状态变为已完成。 4. 用户可以在测试之后生成测试结果。

表 5.5是执行测试任务结果生成的测试用例，该测试流程包括用户的登录、用户选择生成测试结果、系统分析自动测试的文件、得出测试结果并将其归档。

用户点击生成测试结果即可开始测试用例的执行。该测试用例可以通过对执行日志的分析得出评估结果。通过该用例的执行可以有效对结果评估模块进行评估，同时可以检测测试模块是否正确执行。

表 5.5: 执行测试任务结果生成的测试用例

描述项	**说明**
用例编号	TC4
用例名称	执行测试结果生成
用例描述	用户对于一个或多个执行中或执行结束的测试任务提交测试结果生成请求，系统自动生成测试的最终得分。
测试步骤	1. 用户登陆系统，打开任务列表。 2. 为测试中或测试结束的应用点击生成结果。 3. 系统通过分析测试时的运行时文件生成测试结果。
预期结果	1. 系统在测试执行完成之后生成测试结果。 2. 为测试生成报告。 3. 生成报告之后可以在任务列表中点击进入报告。

表 5.6是查看测试报告的测试用例具体描述，该测试用例的流程包括测试用户的登录、用户选择查看测试结果。该报告入口在系统分析的任务列表之中，在结果分析执行完毕之后，通过该入口可以看到测试结果的详细报告。系统通过该用例的执行可以对测试结果生成模块的结果展示部分进行评估，同时可以检测评估任务是否顺利执行。

表 5.6: 查看测试报告的测试用例

描述项	**说明**
用例编号	TC5
用例名称	查看测试报告
用例描述	该测试任务生成结果之后，在网站查看测试报告。
测试步骤	1. 用户登陆系统，打开任务列表。 2. 用户点击查看测试结果。
预期结果	1. 打开测试结果页面，可以查看测试的测试应用。 2. 测试生成相应得分。 3. 测试生成结果图像。

5.2　案例分析

为了验证系统在现实环境的可用性，该节设计了一个实验，执行本系统对自动化测试工具进行检测，实验的测试流程如下：

(1) 在 F-Droid 上挑选不同品类的 17 个开源应用；
(2) 对开源应用进行自动插桩，对插桩应用进行审核，保证应用的代码覆盖率以及 UI 控件覆盖率检测的可用性；
(3) 将插桩失败的应用进行手动修复；
(4) 将插桩成功的应用随机选取变异因子后生成变异应用；
(5) 为该次测试选取待测应用集；
(6) 使用本工具适配的 5 个待测自动化测试工具在设备上进行测试，保存测试过程的运行时文件；
(7) 对测试结果进行分析。

表 5.7即为选择应用集的具体信息，为了保证测试足够全面，应用集包含了多个品类中的代表性应用，如开发工具、网络、图像、多媒体、教育、系统等多个品类。17 个应用之中有 16 个自动插桩成功，失败原因经排查与本系统无关。

表 5.7: 应用详情信息

编号	名称	应用类型	应用版本	自动插桩结果
A1	OSMBugs	Development	v1.2.2	成功
A2	MemeTastic	Graphics	v1.6.7	成功
A3	AndStatus	Internet	v57.07	成功
A4	Firefly III Mobile	Money	v3.0.4	成功
A5	Binary Eye	Multimedia	v1.33.0	成功
A6	DuckDuckGo	Internet	v5.71.0	成功
A7	Eventyay Attendee	Connectivity	v0.9.1	成功
A8	Calendar	Tool	v6.11.3	成功
A9	Gallery	Graphics	v6.19.1	成功
A10	GPSTest	Navigation	v3.9.1	成功
A11	Streak Alarm	Tool	v1.4.1	成功
A12	Kakugo	Education	v1.35.0	成功
A13	Authorizer	Security	v0.4.0	成功
A14	Notepad	Writing	v2.3.6	成功
A15	App Launcher	System	v5.2.0	成功
A16	Pretend You’re Xyzzy	Game	v5.0.9	失败
A17	HgLauncher	Theme	v1.4.3	成功

表 5.8: 失败详情信息

编号	名称	失败原因	修复方式	修复后结果
A16	Pretend You’re Xyzzy	JVM 堆空间耗尽	调整编译时 JVM 内存上限	失败

表 5.8为管理员介入后，代码审核修改流程中对应的具体信息。经检测，Jacoco 插件与 UI 控件覆盖检测插件均工作正常。由于代码编译时间成本过高，每个应用选择 5 个变异因子并编译其变异后的源码。以上操作结束后，应用在系统前端的展示页面如图 5.1所示。点击列表中的“是”可查看编译后的应用详情。

图 5.1: 插桩变异应用列表

图 5.2为前端中 OSMBugs 的变异应用展示页面，在页面的列表中展示了应用名称、应用唯一 ID 以及该应用所使用的变异因子。OSMBugs 在选用 5 个变异因子的情况下共生成了 23 个变异应用，该次测试共生成了 1073 个变异应用。为控制测试时间成本，测试对待测应用进行进一步筛选。为每个应用（指未变异的应用）的每个变异因子随机选择 2 个应用，形成一个变异应用集。由于选择的变异因子生成的变异源码有可能编译不成功，部分变异因子生成的变异应用也可能不足 2 个，所以变异应用集有可能不足 10 个。若不足 10 个则随机选择变异应用，直到应用集数目补齐至 10 个或所有变异应用均被选中。该次测试共选择 168 个应用，包括 16 个原应用以及 152 个变异应用。

应用名	应用ID	变异因子
OSMBugs	60499c5657744213d7c5238e	NULL_INTENT
OSMBugs	60499c5e57744213d7c52390	NULL_INTENT
OSMBugs	60499c6557744213d7c52392	NULL_INTENT
OSMBugs	60499c6d57744213d7c52394	NULL_INTENT
OSMBugs	60499c7357744213d7c52396	NULL_INTENT

每页行数: 5　1-5 的 34

图 5.2: OSMBugs 变异应用展示列表

表5.9是本次测试选用的设备，选用了不同尺寸不同型号的手机类型共 6 款，其中每个设备搭载的系统安卓版本不同，各厂商定制的系统版本也不同。可以测试不同测试工具在碎片化方面的表现。

表 5.9: 测试选用设备详情

编号	品牌	型号	安卓版本	系统版本
D1	SAMSUNG	S20	11	OneUI 3.0
D2	SAMSUNG	S20+	10	OneUI 2.5
D3	HUAWEI	Mate 40	10	EMUI 10
D4	Xiaomi	Mi6	9	MIUI 11
D5	OnePlus	OnePlus2	8	LineageOS
D6	Vivo	Xplay6	7	Funtouch OS

本次测试共挑选了 9 个自动化测试工具做为待测工具。表 5.10为未接入的自动化测试工具，虽然系统可以接入表中的测试工具，但以上自动化测试工具对设备要求苛刻，对安卓碎片化问题的支持较差。根据安卓设备当前的保有量占比，可以用于以上自动化测试工具的设备占比均不超过 25%。而且自动化测试工具的可用性较低，其中 Sapienz 开源版、Dynodroid 以及 PUMA 所支持的 SDK 版本过低，当前大部分主流应用对工具要求的 SDK 版本不提供支持，而且工具没有后续的更新计划。所以没有将以上自动化测试工具加入此次测试。

表 5.10: 未接入的自动化测试工具

自动化测试工具	未接入原因
GUIRipper	需要 Root 权限
Dynodroid	只能运行在安卓 2.3 的设备上
PUMA	只能运行在 SDK 25.0 以下
Sapienz	Sapienz 开源版本只能运行在安卓 4.4.2 的模拟器上

表5.11为本次测试选用的自动化测试工具，首先为自动化测试工具实现接口使其接入本系统。选中的工具涵盖了三种测试策略。其中混合策略是将随机策略与模型探索策略相融合。安卓自动化测试多维评估系统可对现有可用的自动化测试工具进行较为全面的测试，体现本系统对不同自动化测试工具的良好托管能力。其中运行参数为每次执行测试工具所添加的参数，所添加的参数旨在保证自动化测试工具可以完成一次完整的测试，不会因为时长配置等问题中止测试。随机策略的测试工具会为其添加配置，使其每次测试所生成随机事件符合规律。

表 5.11: 系统适配的自动化测试工具

编号	自动化测试工具	策略	运行参数
T1	Monkey	随机策略	`-s 100 --throttle 100 -v -v -v 3600`
T2	Google AppCrawler	模型探索策略	`--time-sec 100000 --ui-automator-mode`
T3	AppCrawler	模型探索策略	`--capability noReset=true -t 3600`
T4	Maxim	混合策略	`-s 100 --uiautomatormix 3600`
T5	Fastbot	混合策略	`--agent robot --running-minutes 30 --throttle 800`

在准备工作完成之后，分别在不同设备上运行所有的自动化测试工具，每个自动化测试工具将测试所有的待测应用集。即每个应用在同一个设备上会使

用不同的测试工具共执行 5 次测试。开始任务的页面如图 5.3所示，在该页面上可以选择测试设备、工具以及待测应用。在点击开始任务后，该测试任务将会添加到任务列表中，并在后端以异步的形式执行测试。

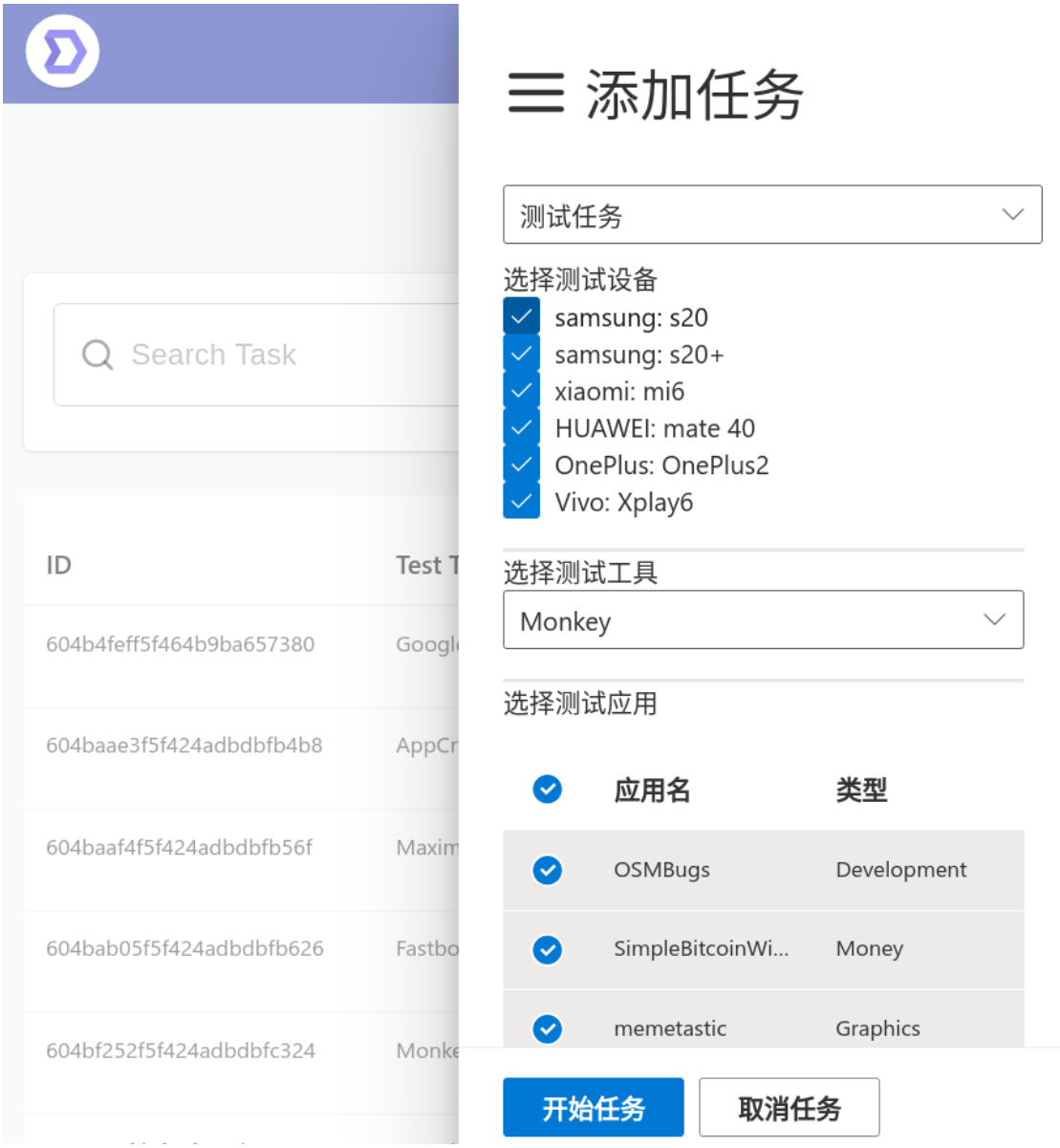

图 5.3: 启动测试任务界面

在测试开始之后以及执行过程中，可以在测试任务列表中看到相应的测试，同时可以在任务详情中看到任务的相关信息。在任务执行结束后，可以在任务详情中得到任务的具体结果。任务详情中的任务相关信息如图 5.4所示。其中包括测试工具、测试执行情况以及包含所有被测应用的表格，在测试状态中，斜杠之前为当前执行完毕的测试任务数，斜杠之后为当前任务中的所有测试任务数。

Task Detail

测试工具： Maxim
测试状态： 180/180

ID	File Name	File MD5	Apk Label	Apk Version	变异因子
6048d0c65774d0...	GPSTest-google-...	53f95e54019c48...	GPSTest	18078	
604a24d164d45...	GPSTest-google-...	2b1ff8b5af72518...	GPSTest	18078	
6048d55e5774d...	GPSTest-google-...	1381d537710ca4...	GPSTest	18078	INVALID_KEY_IN...
6048d18e5774d...	GPSTest-google-...	01660c957de2d1...	GPSTest	18078	INVALID_ACTIVIT...
6048d31b5774d...	GPSTest-google-...	4a74b2faf10135...	GPSTest	18078	DIFFERENT_ACTI...

1

图 5.4: 任务相关信息

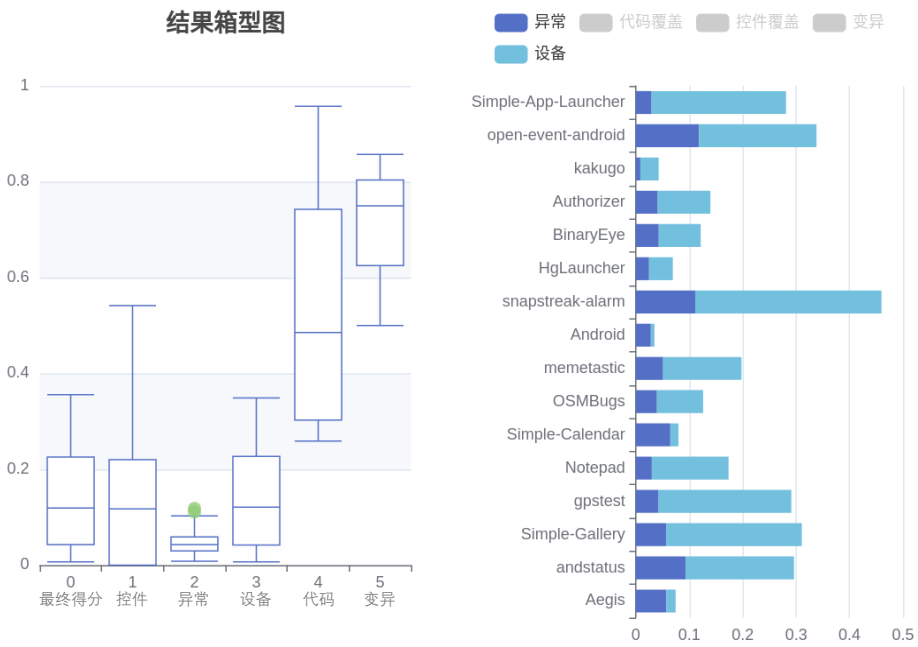

图 5.5: 任务测试结果

任务详情中的任务测试结果如图 5.5所示。在任务执行完毕后，将会在任务详情中展示由 Echarts 渲染的结果图，包括一个箱型图以及一个堆叠条形图，图像展示了本次测试的最终得分情况。其中箱型图从左至右分别是该应用的最终得分、UI 控件得分、异常得分、设备得分（碎片化得分）、代码覆盖率得分以及变异得分。而右侧图可以选中单个得分项对不同应用的得分进行对比，图中即

为选中异常以及设备得分后所展示的结果。

图 5.6为该案例的结果箱型图，该图中数据为不同工具最终得分的平均值，其原始得分由公式 3.7计算得到。可以看出在复杂应用中结果较为离散，且有些得分为 0。其主要是因为工具对碎片化的支持较差，而且因为复杂的应用程序拥有特定触发条件，模型探索策略的测试工具不能准确识别其状态，导致测试效果差别较大。简单应用的测试效果比较集中。同样可得采用随机策略的测试工具相较采用模型探索策略的测试工具其结果拥有较好的稳定性，在不同设备上结果相近。

表 5.12: 案例测试结果

编号	碎片化	UI 控件	代码覆盖率	异常	变异	最终得分
T1	97	78	100	59	100	87
T2	79	85	74	46	70	71
T3	38	5	55	49	19	20
T4	98	80	83	51	68	76
T5	100	100	80	100	75	91

对所有的测试任务执行结果进行收集，对各维度得分进行坐标变换，即将最大值视为 100 分，结果如表 5.12所示。其最终得分为评估标准中五个维度的平均得分，该得分的计算采用系统的默认计算逻辑。在实际使用过程中，本系统可根据用户需求在前端对不同维度的权重进行调整，以区分不同维度在测试中的重要程度。在代码覆盖方面，采用随机策略的工具拥有更为优异的表现，模型探索策略的工具在 UI 控件得分上表现更为优异。由于采用模型探索策略的自动化测试工具虽然可以检测当前页面的状态，但是并不能使页面达到特定的触发条件，而随机策略会在相同页面停留较长时间，可以执行更多操作，更有可能触发特定条件，所以其代码得分较高。在探索新的界面等方面，采用随机探索策略的工具更有优势，其 UI 控件得分较高。

在该案例中，系统可正确分析插桩应用的运行时生成文件以及日志，证明系统可有效对应用进行插桩变异。系统正常使用测试工具进行了 5040 次测试，证明系统可以有效支持对自动化测试工具。Choudhary 等人 [2] 在 2015 年对自动化测试工具进行了测试，在代码覆盖率、异常、碎片化以及可用性四个方面对自动化测试工具进行评估。将案例中 Monkey 的测试结果与 Choudhary 的结果进行对比，案例中代码覆盖率和触发的异常数目与 Choudhary 测试的结果吻合，可以证明本系统的可用性。本系统的评测维度与 Choudhary 相比，有以下几点改进：

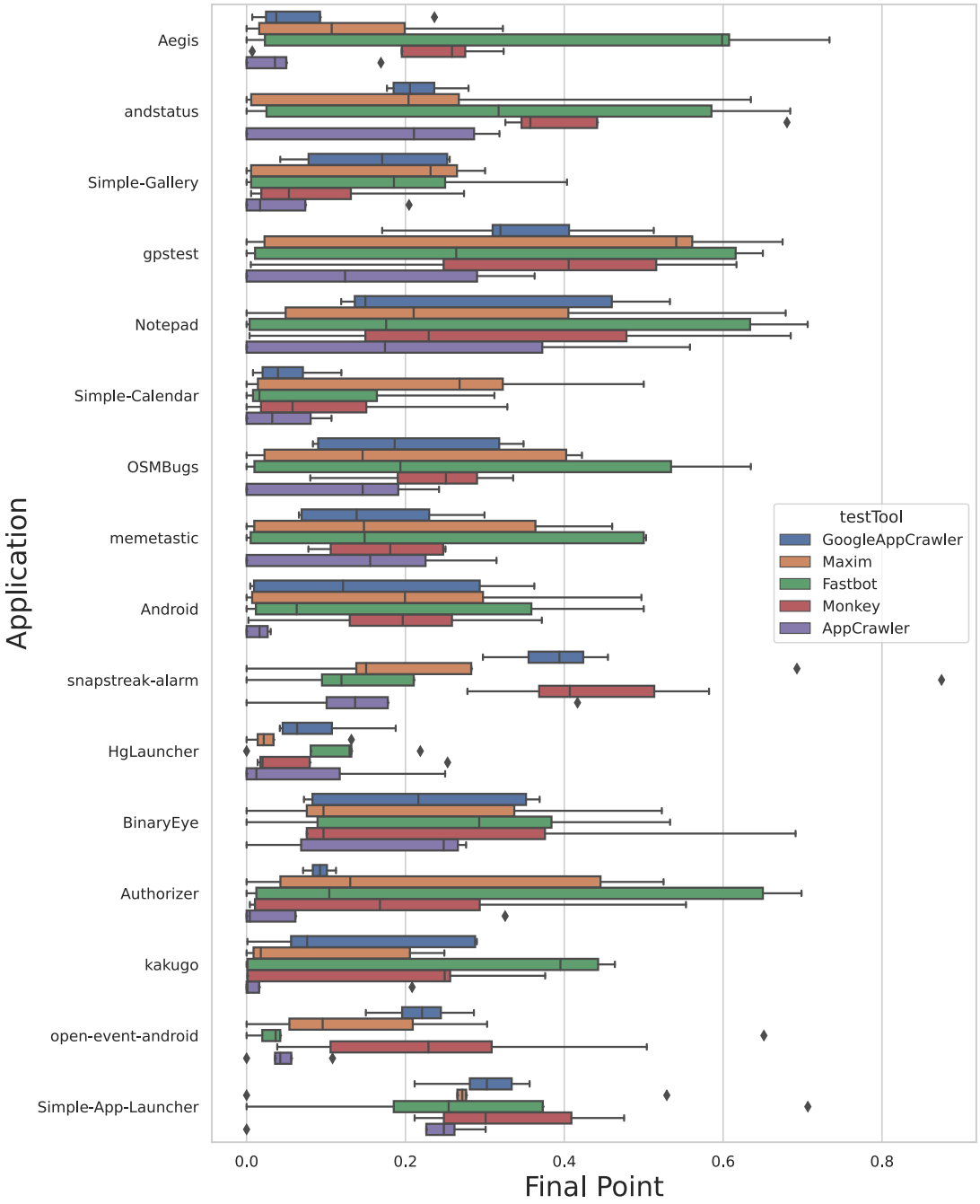
Application
Aegis
andstatus
Simple-Gallery
gpstest
Notepad
Simple-Calendar
OSMBugs
memetastic
Android
snapstreak-alarm
HgLauncher
BinaryEye
Authorizer
kakugo
open-event-android
Simple-App-Launcher
0.0
0.2
0.4
0.6
0.8
Final Point
testTool
GoogleAppCrawler
Maxim
Fastbot
Monkey
AppCrawler

图 5.6: 结果箱型图

系统增加了 UI 控件得分，该得分与代码覆盖率作用相同，均可用于检测自动化测试工具所探索的规模；系统增加了的变异得分，该得分和异常得分作用相同，均可用于检测自动化测试工具检测错误的充分性；本系统的碎片化得分增加了对机型占有率的考量；系统舍弃了人为因素影响较大的可用性评估。以上因素使得该案例的评测结果与 Choudhary 等人的结果相比对工具的评估更有区分度。

5.3　本章小结

本章首先介绍了系统测试，其中包括了对测试目标、测试环境、功能测试的介绍。通过系统测试可以有效保证系统的质量，同时可以验证系统的可用性。随后对测试案例进行分析，详细介绍了测试案例的设计、执行实际结果分析。通过此案例可以充分证明本系统多维分析的有效性和充分性。

第六章　总结与展望

6.1　总结

伴随着移动互联网的蓬勃发展，智能手机成为了日常生活中最为重要的部分。不管是购物、出行、还是疫情的防控，这些都离不开移动互联网和智能手机。随着大众对设备更为广泛频繁的使用，对使用质量的要求也逐步增加。对于软件开发而言，测试变得愈发重要。而移动互联网的开发时限较短，人工测试并不能满足移动应用开发的需求。安卓在智能手机方面的市场占有率比例高达84.7%，但是由于安卓开源的特性导致碎片化问题愈发严重，自动化测试工具变得尤为重要。通过自动化测试工具，可以有效降低企业的测试成本，而且可以在一定程度上减轻个人主观因素对测试的影响。但目前自动测试很多都没有在现实环境中做过充分的测试，导致评估结果不够真实客观。自动化测试工具没有一个可以对其全面客观评估的系统，大部分自动化测试工具声称自己测试效果良好，但是兼容性通常不够好，其评估样本多数不够充分。这不仅增加了用户了解自动化测试工具测试效果的成本，而且对自动测试的推广起到了很大的阻碍。为了解决以上提到的问题，本文设计并实现了一个自动化测试工具的多维评估系统，旨在为自动化测试工具提供统一的评测系统以及客观的评测标准。

在文章架构上本文首先介绍了该项目的背景与自动化测试工具的发展现状，在第二章简要介绍了本系统使用的技术，从与安卓变异相关的 MDroid+，到与手机连接相关的 ADB，再到与插桩相关的 Gradle、ASM、Jacoco，随后简要介绍了自动化测试工具的分类，最后介绍了 Spring Boot 以及将应用容器化的 Docker。

本文从第三章开始详细介绍此系统，首先对系统进行概述。随后对系统的需求进行分析，分别从功能性需求、非功能性需求、系统用例设计以及用例描述方面对其进行介绍。最后对系统的总体设计从以下三个方面进行概要介绍：通过“4+1 视图”对系统模块进行介绍；通过流程图对插桩、覆盖率检测、测试的设计进行介绍；通过公式对系统的评估指标进行介绍。从第四章开始对本系统的详细设计与实现进行详细描述，将系统的 UI 控件覆盖检测插件、源码插桩变异模块、测试模块、测试结果评估模块进行详细介绍，每个部分首先介绍该部分的概述，随后从时序图、主要类、主要类的详情、核心类图、关键代码这几个方面对模块的设计实现进行详细介绍。通过以上两章可以较为详细的了解本工具的设计原理与实现。

在第五章为本工具设计了功能测试，用于对前文中提到的需求进行检测，并

对相应案例进行分析，同时结合前端页面对其测试流程以及结果进行展示。从测试和案例分析的结果可知本系统拥有较好的可用性。

6.2 展望

自动化测试的多维检测系统已经完成了初步的开发，关于自动化测试工具的检测、对自动化测试工具效果的基本度量等核心功能均已实现。伴随着安卓自动测试产业的发展，本工具仍需进行后续的迭代与改善：

(1) 本文目前实现了异常、变异、代码覆盖率、UI 控件覆盖率以及碎片化等五个维度的评估，在后续使用中，可以扩充其它维度对自动化测试工具进行评估。通过不断增加测试维度，可以使测试评估结果更为全面。

(2) 本文已经适配了五个自动化测试工具，虽然对每种自动化测试工具类别都有所涵盖，但是总量不是特别理想。系统已经为自动化测试工具的适配提供了相应接口，后续可以较为简便的对自动化测试工具进行适配。

(3) 因本文已经适配了一些自动化测试工具，后期可以考虑通过对系统改造使其成为自动化测试工具的脚手架，用户上传应用并提供设备即可对应用进行多工具的全面测试。

参考文献

[1] 中共中央网络安全和信息化委员会办公室，第 46 次《中国互联网络发展状况统计报告》，http://www.cac.gov.cn/2020-09/29/c_1602939918747816.htm.

[2] S. R. Choudhary, A. Gorla, A. Orso, Automated test input generation for android: Are we there yet?, CoRR abs/1503.07217.
URL http://arxiv.org/abs/1503.07217

[3] D. Amalfitano, A. R. Fasolino, P. Tramontana, S. D. Carmine, A. M. Memon, Using GUI ripping for automated testing of android applications, in: M. Goedicke, T. Menzies, M. Saeki (Eds.), IEEE/ACM International Conference on Automated Software Engineering, ASE'12, Essen, Germany, September 3-7, 2012, ACM, 2012, pp. 258–261.
URL https://doi.org/10.1145/2351676.2351717

[4] Google, Monkey: a command-line tool that generates pseudo-random streams, https://developer.android.com/studio/test/monkey.html.

[5] 黄延胜, Appcrawler：基于 appuim 的 app 自动遍历工具, https://github.com/seveniruby/AppCrawler.

[6] Bytedance, Z. Zhao, Fastbot-android: a model-based testing tool for modeling gui transitions to discover app stability problems, https://github.com/bytedance/Fastbot_Android.

[7] M. Tokic, G. Palm, Value-difference based exploration: Adaptive control between epsilon-greedy and softmax, in: J. Bach, S. Edelkamp (Eds.), KI 2011: Advances in Artificial Intelligence, 34th Annual German Conference on AI, Berlin, Germany, October 4-7,2011. Proceedings, Vol. 7006 of Lecture Notes in Computer Science, Springer, 2011, pp. 335–346.
URL https://doi.org/10.1007/978-3-642-24455-1_33

[8] 陈翔, 顾庆, 变异测试: 原理, 优化和应用, 计算机科学与探索 6 (12) (2012) 1057–1075.

[9] Y. Jia, M. Harman, An analysis and survey of the development of mutation testing, IEEE Trans. Software Eng. 37 (5) (2011) 649–678.
URL https://doi.org/10.1109/TSE.2010.62

[10] L. Brader, H. F. Hilliker, A. C. Wills, Testing for Continuous Delivery with Visual Studio 2012, Microsoft, 2012.

[11] 朱少民, 软件测试方法和技术, 清华大学出版社有限公司, 2005.

[12] G. Quentin, Automated software testing: Introduction, management and performance, elfriede dustin, jeff rashka and john paul, addison-wesley, 1999 (book review), Softw. Test. Verification Reliab. 9 (4) (1999) 283–284.

[13] Google, Android debug bridge: a versatile command-line tool that lets you communicate with a device, https://developer.android.google.cn/studio/command-line/adb.

[14] M. Hoffmann, B. Janiczak, E. Mandrikov, M. Friedenhagen, Jacoco code coverage tool, https://www.eclemma.org/jacoco.

[15] S. Dashevskyi, O. Gadyatskaya, A. Pilgun, Y. Zhauniarovich, The influence of code coverage metrics on automated testing efficiency in android, in: D. Lie, M. Mannan, M. Backes, X. Wang (Eds.), Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018, ACM, 2018, pp. 2216–2218.
URL https://doi.org/10.1145/3243734.3278524

[16] S. Christou, free java tool that calculates the percentage of code accessed by tests, http://cobertura.github.io/cobertura.

[17] M. J. Shepperd, A critique of cyclomatic complexity as a software metric, Softw. Eng. J. 3 (2) (1988) 30–36.
URL https://doi.org/10.1049/sej.1988.0003

[18] K. Moran, M. Tufano, C. Bernal-Cárdenas, M. L. Vásquez, G. Bavota, C. Vendome, M. D. Penta, D. Poshyvanyk, Mdroid+: a mutation testing framework for android, in: M. Chaudron, I. Crnkovic, M. Chechik, M. Harman (Eds.), Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings, ICSE 2018, Gothenburg, Sweden, May 27 - June 03, 2018, ACM,

2018, pp. 33–36.
URL https://doi.org/10.1145/3183440.3183492

[19] Y. Jia, M. Harman, An analysis and survey of the development of mutation testing, IEEE Trans. Software Eng. 37 (5) (2011) 649–678.
URL https://doi.org/10.1109/TSE.2010.62

[20] R. Just, The major mutation framework: efficient and scalable mutation analysis for java, in: C. S. Pasareanu, D. Marinov (Eds.), International Symposium on Software Testing and Analysis, ISSTA '14, San Jose, CA, USA - July 21 - 26, 2014, ACM, 2014, pp. 433–436.
URL https://doi.org/10.1145/2610384.2628053

[21] H. Coles, Pit: a state of the art mutation testing system, http://pitest.org/.

[22] L. Deng, N. Mirzaei, P. Ammann, J. Offutt, Towards mutation analysis of android apps, in: Eighth IEEE International Conference on Software Testing, Verification and Validation, ICST 2015 Workshops, Graz, Austria, April 13-17, 2015, IEEE Computer Society, 2015, pp. 1–10.
URL https://doi.org/10.1109/ICSTW.2015.7107450

[23] B. Muschko, Gradle in action, Vol. 27, Manning, 2014.

[24] 马思峻, 肖荣, 成江伟, Android 应用性能数据采集探针研究, 计算机应用与软件 34 (7) (2017) 192–197.

[25] E. Bruneton, Asm: Java bytecode manipulation framework, https://asm.ow2.io/.

[26] S. Yamazaki, M. Matsumoto, T. Nakanishi, T. Kitasuka, A. Fukuda, A case study of development of a java bytecode analyzer framework using aspectj, IPSJ Digital Courier 1 (2005) 104–116.

[27] 王灼洲, Android 全埋点解决方案, 机械工业出版社, 2019.

[28] 赵耀宗, 程绍银, 蒋凡, Android 应用程序 gui 遍历的自动化方法, 计算机系统应用 24 (9) (2015) 219–224.

[29] W. Choi, G. C. Necula, K. Sen, Guided GUI testing of android apps with minimal restart and approximate learning, in: A. L. Hosking, P. T. Eugster, C. V. Lopes

(Eds.), Proceedings of the 2013 ACM SIGPLAN International Conference on Object Oriented Programming Systems Languages & Applications, OOPSLA 2013, part of SPLASH 2013, Indianapolis, IN, USA, October 26-31, 2013, ACM, 2013, pp. 623–640.
URL https://doi.org/10.1145/2509136.2509552

[30] Google, App crawler, use the app crawler tool, part of jetpack, to automatically test your app without the need to write or maintain any code., https://developer.android.com/training/testing/crawler.

[31] R. Mahmood, N. Mirzaei, S. Malek, Evodroid: segmented evolutionary testing of android apps, in: S. Cheung, A. Orso, M. D. Storey (Eds.), Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering, (FSE-22), Hong Kong, China, November 16 - 22, 2014, ACM, 2014, pp. 599–609.
URL https://doi.org/10.1145/2635868.2635896

[32] M. Flauzino, J. Veríssimo, R. Terra, E. Cirilo, V. H. S. Durelli, R. S. Durelli, Are you still smelling it?: A comparative study between java and kotlin language, in: Proceedings of the VII Brazilian Symposium on Software Components, Architectures, and Reuse, SBCARS 2018, Sao Carlos, Brazil, September 17-21, 2018, ACM, 2018, pp. 23–32.
URL https://doi.org/10.1145/3267183.3267186

[33] C. Pautasso, O. Zimmermann, F. Leymann, Restful web services vs. ”big”’ web services: making the right architectural decision, in: J. Huai, R. Chen, H. Hon, Y. Liu, W. Ma, A. Tomkins, X. Zhang (Eds.), Proceedings of the 17th International Conference on World Wide Web, WWW 2008, Beijing, China, April 21-25, 2008, ACM, 2008, pp. 805–814.
URL https://doi.org/10.1145/1367497.1367606

[34] R. Goers, G. Gregory, S. Deboy, Log4j–log4j 2 guide-apache log4j 2 (2015).

[35] D. Bernstein, Containers and cloud: From LXC to docker to kubernetes, IEEE Cloud Comput. 1 (3) (2014) 81–84.
URL https://doi.org/10.1109/MCC.2014.51

[36] 鲁涛, 陈杰, 史军, Docker 安全性研究, 计算机技术与发展 28 (006) (2018) 115–120.

[37] P. Kruchten, The 4+1 view model of architecture, IEEE Softw. 12 (6) (1995) 42–50.
URL https://doi.org/10.1109/52.469759

[38] SkylledDev, Distribution data for android, https://androiddistribution.io/#/.

[39] Google, Distribution dashboard: a page that provides information about the relative number of devices that have a certain characteristic, https://developer.android.com/about/dashboards.

[40] A. Saxena, Handling of synchronized data using java/j2ee, International Journal of Managment, IT and Engineering 1 (7) (2011) 182–194.

致　谢

两年的硕士研究生生活即将落幕，在这两年我学到了很多专业知识，同时对软件工程有了更深刻的实践和认识。在此，我要对所有的老师、同学以及亲友致以最诚挚的感谢。

首先，我要感谢陈振宇教授以及房春荣老师，从系统设计、研究思路到实验设计，他们都给了我很多帮助。感谢陈老师让我加入 ISE 实验室，可以在一个优秀奋进的环境中不断提高自己，也感谢房老师对我学术上亲力亲为的指导。

其次，我要感谢钟怡学姐和徐悠然学长对项目的帮助，同时我要感谢程龙、程豪、邓靖琦、梁越勇和张晓波同学，他们都是我想要学习的伟大榜样。

我也要感谢父母和亲友在研究生生涯对我的支持，他们在我懈怠的时候警醒我，在我焦虑的时候安慰我，给予了我前进的动力。

我还要感谢南京大学坐落于南京市，我爱南京。

最后，感谢能在百忙之中审阅论文以及参加答辩的各位评审专家，向你们致敬，感恩有您!