南京大学

研 究 生 毕 业 论 文

（申 请 硕 士 学 位）

论 文 题 目 基于动静态程序分析的

安卓应用众包测试用例生成技术

作 者 姓 名 郭超

专 业 方 向 软件工程领域

研 究 方 向 软件工程

指 导 教 师 陈振宇 教授

2022 年 **5** 月 **20** 日

学　　号：**MG1932003**

论文答辩日期：**2022** 年 **5** 月 **20** 日

指 导 教 师：　　（签字）

Crowdsourced Testing Case Generation Technology for Android Applications Based on Dynamic and Static Program Analysis

by

Chao Guo

Supervised by

Professor **Zhenyu Chen**

A dissertation submitted to
the graduate school of Nanjing University
in partial fulfilment of the requirements for the degree of
MASTER
in
Software Engineering

Software Institute
Nanjing University

May 20, 2022

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：基于动静态程序分析的安卓应用众包测试用例生成技术

软件工程领域 专业 2019 级硕士生姓名：郭超

指导教师（姓名、职称）：陈振宇 教授

摘　要

Android 应用以快速迭代的优势迅速占据移动应用市场，但是严重的碎片化问题和复杂的使用场景为其测试带来巨大挑战。自动化测试和众包测试成为 Android 测试的重要方法。自动化测试支持在设备上反复测试，可以降低人工测试成本。然而，自动化测试难以实现测试路径全覆盖，且通常只能检测崩溃问题，无法测试页面布局缺陷和用户体验等其他问题。为弥补自动化测试的不足，众包测试成为一种新的移动应用测试模式，其通过互联网招募未知大众执行测试任务，能够获取丰富的测试场景，一定程度上解决 Android 碎片化问题。然而，传统的众包测试存在测试人员专业性不足等问题，需要生成测试用例，辅助测试人员高效完成测试。

本文提出了一种基于动静态程序分析的 Android 应用众包测试用例生成技术。使用慕测平台的自动化测试工具进行自动化分析，获取测试过程中产生的操作事件序列和应用截图信息，构建带有应用截图的测试行为列表，为其生成测试操作描述。使用经过优化的 GATOR 工具获得窗口跳转信息，结合映射信息构建包含控件真实名称的测试行为列表，并生成测试操作描述。基于动静态分析获取的测试行为列表构建窗口跳转模型，从模型中获取自动化测试未覆盖的测试行为。使用最短路径算法，选择模型中从应用起始窗口到自动化未覆盖测试行为的最短测试行为路径，基于测试行为中的测试操作描述形成测试用例的步骤描述，使用应用截图辅助步骤说明，引导测试人员完成众包测试。

本文在六个应用中分别采用三组实验进行验证。实验结果证明，本技术相较于传统的众包测试和基于自动化测试的众包测试分别可以提升 21.8%、14.2% 的测试路径覆盖率，静态分析对提升路径覆率起到显著作用。此外，通过分析在相同时间内发现异常的数量，证明本技术可以增加 24.7% 的异常发现数量。

关键词： 众包测试，自动化测试，静态分析，测试用例，测试引导

南京大学研究生毕业论文英文摘要首页用纸

THESIS: Crowdsourced Testing Case Generation Technology for Android Applications Based on Dynamic and Static Program Analysis

SPECIALIZATION: Software Engineering

POSTGRADUATE: Chao Guo

MENTOR: Professor **Zhenyu Chen**

Abstract

Android applications quickly occupy the mobile application market with the rapid and iterative development model. However, it presents many challenges for software testing due to the serious fragmentation issues and diverse usage environments. To ensure the quality of Android applications, a large number of techniques have been proposed to assist in testing tasks, including automated testing and crowdsourced testing. Automated testing techniques automatically simulate program operation and execute test cases with little to no human involvement. However, automated testing is difficult to achieve full coverage of the test path, and usually can only detect crashes, but cannot test other problems such as page layout defects and user experience. To alleviate the above problems, crowdsourced testing has become a new working paradigm. It recruits capable users to perform testing tasks through the Internet, which can obtain rich testing scenarios and solve the fragmentation problem to a certain extent. However, the traditional crowdsourced testing has problems such as the lack of testers' professionalism, so it is necessary to generate test cases to assist the testers to complete the test efficiently.

This paper proposes a crowdsourced testing case generation technology for Android applications based on dynamic and static program analysis. First, we use the automated testing tool provided by MoocTest platform to perform automated testing, which can record application logs, operation event sequences and screenshots, and build a test behavior list with the screenshots. Second, this paper utilizes the optimized GATOR, which is a widely used static analysis tool, to obtain window transition information. Here, we combine the mapping information to build a test behavior list and generate

operation descriptions. Third, we build a window transition model based on the list of test behaviors obtained from dynamic and static analysis, and obtain test behaviors not covered by automated testing. Finally, we use the shortest path algorithm to guide testers to explore testing behaviors that are not covered by automated testing tools. In particular, we provide detailed test step descriptions and corresponding screenshots to help testers understand the test task.

To validate the approach, we use three sets of experiments with 6 Android applications. The experimental results show that this technology can improve the test path coverage by 21.8% and 14.2% respectively compared to the traditional crowdsourced testing and the crowdsourced testing based on automated testing. Static analysis plays a significant role in improving the path coverage. Furthermore, by analyzing the number of anomalies found in the same time, it is demonstrated that this technique can increase the number of anomalies found by 24.7%.

keywords: Crowdsourced testing, Automated testing, Static analysis, Test cases, Test guidance

目　　录

插图清单

附表清单

第一章　绪论

1.1　研究背景和意义

随着科学技术的进步，移动互联网进入了高速发展的快车道。根据 newzoo[①]的报告可知，2021 年全球共有 39 亿个智能手机用户，46 亿部活跃智能手机。纵观全球移动操作系统市场份额，根据 StatCounter[②]统计，截止 2022 年 3 月 Google 公司开发的 Android 操作系统以占比 71.7% 的压倒性优势领先于苹果的 iOS 系统和其他移动操作系统。据工信部统计，国内市场 APP 数量高达 252 万款，移动应用数量的下载次数已超过两万亿次[③]。

Google 公司将 Android 操作系统进行开源，由于缺乏统一规范，各设备制造厂商均可对操作系统进行定制化开发，于是涌现出众多的平台厂商以及各异的系统版本，这导致 Android 系统碎片化问题日益严峻 [1]。为迎合用户的需求，市面上存在不同屏幕尺寸以及不同网络环境的设备，这对应用质量带来了极大挑战。此外，在应用开发层面，频繁的迭代周期以及复杂的使用场景也使得 Android 测试变得更加困难。

软件测试是软件工程学科不可分割的一部分，它通常占总开发成本的 50% 以上 [2]。相较于传统的 Web 应用，移动应用迭代速度更快、兼容性问题更突出，因此移动应用测试难度更大。由于厂商众多、版本各异、设备多样，导致 Android 碎片化问题日益严峻，使得未经充分测试的应用容易暴露出问题 [3]。不稳定的应用程序会严重影响用户体验，导致应用的评级下降，最终损害应用程序开发人员及其组织的声誉 [4, 5]。有研究发现，53% 的用户会避免使用出现过崩溃现象的应用 [6]。Android 应用程序的下载量每年都呈现急剧增长的趋势，但是在 2019 年仍有 17% 的应用被认为是低质量应用[④]。由此可见，对 Android 应用进行充分测试显得至关重要。

自动化测试和众包测试成为 Android 测试的重要方法。自动化测试可以在

[①] https://newzoo.com/
[②] https://gs.statcounter.com/
[③] https://www.miit.gov.cn/gxsj/tjfx/hlw/art/2022/art_b0299e5b207946f9b7206e752e727e66.html
[④] https://www.appbrain.com/stats/number-of-android-apps

多台设备上进行反复测试，能够解决手工测试费时耗力的问题，很大程度上降低了人工测试成本。然而，自动化测试工具很难覆盖所有测试路径，Chen 等人[7] 通过调研发现，业界主流自动化测试技术的覆盖率通常可达 50% 至 60%。此外，自动化测试工具一般只能发现应用崩溃问题，无法检测页面布局以及功能是否存在缺陷。众包测试通过互联网招募真实用户执行测试任务，能够获取丰富的测试场景，一定程度上可以解决 Android 碎片化问题。众包测试通过人工参与为检测应用页面布局和功能问题提供了可能，弥补了自动化测试的不足。测试用例生成是众包测试的关键步骤之一，然而，人工生成测试用例要求用例生成者对待测产品具备一定的领域知识，往往会耗费大量的人力和时间。并且，传统的众包测试人员存在专业性不足等问题，需要为众包测试人员生成测试用例，辅助高效完成测试。因此，如何为众包测试人员快速生成测试用例显得极其重要。

1.2 相关研究现状

1.2.1 Android 自动化测试

Android 应用程序不同于传统软件，它是基于事件驱动的图形用户界面（GUI）程序。这种 GUI 事件驱动的特性给开发者带来了不小的挑战 [8]，与传统的桌面和服务器应用程序相比，移动应用程序的测试更具有难度。众多的设备厂商以及各异的操作系统导致 Android 系统碎片化问题日益明显，加之频繁的迭代周期和复杂的应用场景使得测试时间变得愈发紧凑，这些都给 Android 应用测试带来了困难。传统的手工测试在这些挑战面前早已难堪重负，手工测试不仅耗时费力，而且在进行回归测试时容易出错。

在此背景下，自动化测试技术应运而生。Android 应用程序可以感知上下文，能够对来自用户和系统交互的不同输入做出反应 [9, 10]。基于 Android 的自动化测试通过在移动设备上产生用户交互事件和系统事件，以达到模拟用户操作的效果。通过遍历页面组件并探索应用程序状态，记录应用程序的堆栈信息，以检测是否触发应用程序的异常或错误 [11]。自动化测试支持同时在多台设备上反复执行测试工作，Baek 等人证明 Android 自动化测试是对传统测试手段的一种有效补充 [12]。采用自动化测试的方式代替传统的手工测试可以避免重复工作，防止在回归测试时由于人工操作带来的错误，可以节省大量成本，并且提高测试效率。

对 Android 自动化测试的研究一直是学术界的比较关心的话题。Hao 等人 [13] 提出了一个灵活的 UI 自动化框架 PUMA，用于实现各种基于状态的测试策略，动态生成应用程序的状态模型。Alzaylaee 等人 [14] 提出了一个深度学习系统 DL-Droid，通过生成有状态的输入进而检测恶意应用程序，实验证明生成输入对于动态分析的重要性。Moran 等人 [15] 将静态和动态分析相结合，提出了 CrashScope 技术，CrashScope 使用系统化的输入生成算法对应用程序进行探索，并采用自然语言的格式生成具有明确步骤的崩溃报告。

测试工具对应用探索的范围越广，就越有可能发现潜在的崩溃 [16]。Choudhary 等人 [17] 分别从代码覆盖率、故障检测能力、易用性以及框架兼容性四个方面对主流的 Android 自动化测试框架进行评估，研究表明自动化测试工具对应用程序的代码覆盖率未能达到 100%。自动化测试工具对应用中的窗口、控件或者对象状态无法实现全覆盖，存在遗漏某些测试路径的情况。此外，自动化测试工具未能发现页面布局缺陷和用户体验问题，无法校验应用的功能是否正常，相较于人工测试还存在不足的地方。

在众包测试过程中，通过众包工人进行人工测试，获得众包工人的真实反馈，可以很好地解决自动化测试工具无法识别应用页面布局和校验应用功能逻辑的问题。将自动化测试工具未覆盖的测试路径形成众包测试用例，辅助众包工人执行测试任务。自动化测试工具的应用截图可以补充众包测试用例信息，众包测试可以弥补自动化测试工具的不足，两者优势互补，实现更加全面、高效的 Android 应用测试。

1.2.2 众包测试

“众包”一词最开始是由 Howe[18] 在 2006 年提出，描述了一种新型的在线工作者分布式问题解决模式。众包是指将任务通过互联网的方式外包给个人或者某个组织，他们通过线上平台的方式自愿承担任务，在完成任务后可以获得一定的报酬。Sterling 提出成千上万用户在成千上万种不同条件下测试过的软件称为最好的软件 [19]。近些年，越来越多的研究人员把众包与软件测试相结合，提出了众包测试的概念。众包测试被应用于许多软件测试活动，如 QoE（quality of experience）测试 [20, 21]、可用性测试 [22, 23]、功能测试 [24]、兼容性测试 [25] 和性能测试 [26] 等。

众包测试是软件测试在众包领域的应用，利用众包的真实性和平台的有效性完成测试任务 [25]。在众包测试过程中主要包括三个参与者：任务请求者、

众包工人和众包测试平台。如图1-1所示是众包测试的流程，可以分为以下几个步骤：

（1）任务请求者在众包测试平台中提交待测应用并发布测试任务。

（2）众包工人在众包测试平台中自愿领取测试任务，完成测试任务后提交测试报告至平台。

（3）众包测试平台的报告审核人员对测试报告进行审核并整理，最后向任务请求者交付最终的测试结果。

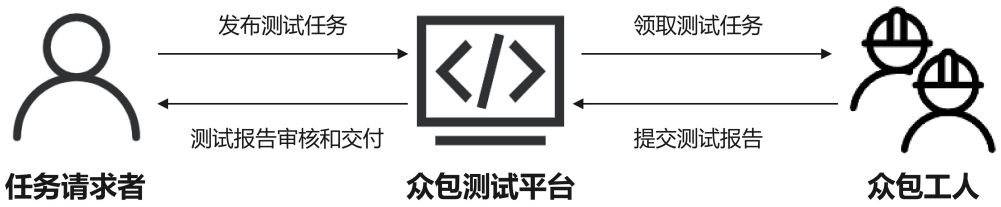


图 1-1: 众包测试流程

众包测试可以在短时间内获得使用不同手机型号、处在不同测试环境、掌握不同语言、拥有不同习惯的众包工人对产品使用的真实反馈。中小型企业或者个体开发者为了对开发的 Android 应用进行全面测试，考虑到市场上 Android 设备种类繁多并且更新速度极快，在有限的预算下拥有较为全面的 Android 设备比较困难。于是，许多中小型企业和个体开发者使用众包测试解决设备不充分的问题。Zhang 等人 [27] 通过对实验室的移动应用测试和众包测试的移动应用测试进行比较，说明移动应用众包测试更具优势。移动应用众包测试主要具有以下优点：

（1）测试成本低：移动应用众包测试的参与者不一定是专业测试人员，所收取的费用更低。此外，众包工人通过使用自己的移动设备完成测试任务，可以降低移动设备的成本以及网络费用的开销。

（2）测试环境丰富：移动应用众包测试可以获得使用不同设备、来自不同地区众包工人的真实反馈，这为检验应用与设备、应用与用户所在位置等问题提供了丰富的验证环境。

（3）测试规模大：移动应用众包测试通过在线招募的方式，可以快速吸引大量众包工人参与。

（4）测试结果全面：移动应用众包测试通过众包工人检测应用页面布局和功能问题，相较于移动应用自动化测试拥有更加全面的测试结果。

众包测试通过众包工人完成测试任务，可以模拟真实应用场景以及真实用

户表现，所需的测试周期较短，并且测试成本较低 [28]。但是众包测试模式本身的一些固有问题会弱化移动应用众包测试的优势，众包测试模式的固有问题主要有以下几点：

（1）众包工人专业素养较低：由于众包测试平台对众包工人的招募没有要求，大多数众包工人可能不具备专业的软件测试知识，与专业软件测试人员相比测试效率更低，这会影响众包测试的结果。

（2）测试报告整合困难：众包测试过程中众包工人会提交大量重复的测试报告，并且由于众包工人水平参差不齐，导致测试报告中编写的测试用例差异较大，这极大地增加了平台审核人员的报告整合压力。

传统的众包测试存在众包工人专业性不足的问题，并且众包工人之间缺乏协作，需要通过生成测试用例辅助众包工人高效完成测试。然而，人工生成测试用例要求用例生成者对待测产品具备一定的领域知识，并且会耗费大量的人力和时间。测试用例对于众包测试过程十分重要，有效的、易于理解测试用例可以提升众包测试的质量和效率。

1.3 本文主要工作

本文结合 Android 自动化测试和众包测试的问题，提出了基于动静态程序分析的 Android 应用众包测试用例生成技术。通过自动化测试工具对应用进行自动化测试，获取操作事件序列和应用截图，构建带有应用截图的测试行为列表，并生成测试操作描述。同时在不执行应用程序的情况下，采用静态分析工具获得应用资源文件和窗口跳转信息，构建包含控件真实名称的测试行为列表，并生成测试操作描述。基于动静态分析获取的测试行为列表构建窗口跳转模型，从模型中获得自动化测试未覆盖的测试行为。采用最短路径思想对窗口跳转模型进行测试路径选择，计算从应用起始窗口到未覆盖测试行为的最短测试行为路径，基于测试行为中的测试操作描述生成测试用例的步骤描述，使用应用截图辅助用例的步骤说明，引导众包工人完成众包测试任务并探索异常。

为验证此技术的有效性，我们选取六款 Android 应用程序，分别开展传统的众包测试、只包含自动化测试的众包测试以及使用该技术的众包测试实验。我们通过对比不同实验中应用的测试路径覆盖率，得出该技术可以有效提升待测应用的测试路径覆盖率，结果还表明引入静态分析的必要性。通过在相同时间内发现的异常数量进行对比，验证了基于动静态程序分析的众包测试能够有

效提高众包测试的质量。

总结相关工作，本文的贡献主要有:

（1）提出一种众包测试用例生成技术，引导众包工人完成众包测试任务并探索异常。

（2）实时推荐测试用例，基于用户的测试反馈对测试用例进行推荐。

（3）设立三组对比实验，分别从测试路径覆盖率和众包测试质量的角度，利用真实数据评估该技术的有效性。

1.4 本文组织结构

本文的组织结构如下:

第一章绪论。介绍该技术的研究背景和研究意义，对 Android 自动化测试、众包测试的研究现状进行阐述，说明本文的主要工作和贡献。

第二章研究背景。介绍自动化测试工具、静态分析工具和众包测试，并通过工具比对说明工具选择的原因。

第三章相关工作。介绍和本文研究主题相关的研究工作，包括测试用例生成和测试路径选择算法。

第四章测试用例生成技术。从自动化测试、静态分析、模型构建和用例生成四方面对该技术的设计和实现原理进行详细介绍。

第五章实验设计与分析。介绍本文的实验设计与分析，通过设定实验目的，利用六款应用程序分别完成三组实验，通过实验对该技术进行评估，并对实验结果进行深入分析。

第六章总结与展望。对本文工作进行总结，并对未来工作进行展望。

第二章　研究背景

2.1　Android 自动化测试工具

自动化测试是一个将以人为驱动的测试行为转换为机器执行的过程。由于 Android 应用程序基于事件驱动，输入通常以事件的形式出现，自动化测试工具可以通过模拟点击、滑动、文本输入的用户交互事件，也可以模拟如屏幕旋转、收到短信通知等系统事件对应用进行测试。Android 自动化测试工具使用不同策略生成这些输入，主要可以分为以下几类:

基于随机策略的自动化测试工具。最具代表性的是 Monkey[①]工具，Monkey 是谷歌内置于 Android 平台的官方测试工具，相较于其他自动化测试工具更易获得广泛应用。Monkey 通过自动生成随机事件序列，其中包含基本的用户交互事件，分析操作事件所产生的日志文件，以检测应用运行中产生的错误。但 Monkey 只采用命令的方式发送系统随机事件流，以黑盒测试的方式完成测试过程，并且如果跳转到其他非测试项界面中，可能导致测试不受控制。Machiry 等人 [29] 提出了一种基于伪随机策略的开源测试工具 Dynodroid，相较于 Monkey 的完全随机策略，Dynodroid 采用的随机策略更具优势，它可以基于事件上下文或者事件执行频率产生操作事件，但对于最近最少使用的事件会出现偏差。

基于模型策略的自动化测试工具。此类工具常常与 GUI 相结合，其核心思想通常是根据应用页面变化，动态构建应用程序事件与页面状态之间的 GUI 变更模型。应用程序的状态会随着 GUI 事件的执行而不断变化 [30]。Amalfitano 等人 [31] 提出了一种采用深度优先遍历策略探索应用程序不同状态的技术 GUIRipper，当探索过程中未检测到新状态时，它会重新进行探索。然而，安装 GUIRipper 比较困难，并且只支持 Windows 环境。随后，Amalfitano 等人 [32] 在 GUIRipper 基础上提出了 MobiGUItar 框架，MobiGUItar 可以从应用程序中获得状态机模型，通过模拟应用程序的 GUI 状态，不断更新 GUI 状态并执行事件。Yang 等人 [33] 将 GUIRipper 和静态分析相结合提出了 ORBIT

[①]https://developer.android.com/studio/test/monkey

工具，ORBIT 可以避免产生无关的 UI 事件，通过对静态分析结果进行处理，获得特定页面和特定 UI 事件之间的关联，以此辅助自动化测试，很遗憾的是 ORBIT 并未开源。Azim 等人 [34] 提出了一种基于页面进行抽象的测试工具 A3E-Depth-First，但是 A3E-Depth-First 忽略了页面内部组件状态，因此可能会错过一些非常容易实现的操作。

基于探索策略的自动化测试工具。Mahmood 等人 [35] 首次提出了基于探索的 Android 测试框架 EvoDroid，它扩展了接口模型和调用图模型，但 EvoDroid 框架的实现并未进行公开。Mao 等人 [36] 提出了 Sapienz 技术，使用基于多目标搜索策略对应用进行探索，通过优化测试序列以缩短测试序列的长度。Sapienz 旨在最大限度地提高代码覆盖率，尽可能地发现故障。Van 等人 [37] 提出了 JPF-Android，通过扩展 Java 模型检查工具 Java Path Finder（JPF）以支持 Android 应用程序。JPF-Android 旨在探索应用程序中的所有路径，但是 JPF-Android 的局限性严重限制了它的实际应用。

本文采用的是 Mooctest [①] 自动化测试工具，该工具为了达到对应用更加全面和充分的测试效果，使用深度优先的遍历策略对应用进行测试，通过点击、滑动、文本输入等方式触发用户交互事件和系统监听事件。相较于应用广泛的 Monkey、Dynodroid 等工具，此工具提供人工测试脚本为辅助，允许在测试过程中主动使用人工测试脚本中的输入参数，辅助执行输入事件，尽可能地遍历应用中的组件，以提高应用程序的组件覆盖率。并且，Mooctest 自动化测试工具可在 Windows、Liunx、Macos 等操作系统上稳定运行。

2.2 Android 静态分析工具

静态分析是指不执行程序，将程序的源码作为输入，检查语句序列、代码结构和变量值在不同函数调用中的处理结果 [38]。Android 应用程序是基于事件机制，因此和传统的 Web 应用程序不同。Android 应用程序开发过程中需要声明窗口、控件以及事件监听处理函数，通过对应用程序的源码进行分析，可以获得程序的逻辑结构，采用语法分析、词法分析、数据流分析或控制流分析等技术遍历程序路径，对程序属性进行检查，检查代码的规范性、安全性等指标。

[①] www.mooctest.net

2.2.1 Android 源码获取工具

Android 静态分析的关键在于获取应用程序源码，源码是进行静态分析工作的基础。目前市面上已有较多成熟的 Android 静态源码获取工具，例如 ApkTool [①]、Dex2Jar [②]、jadx [③]。

● ApkTool：ApkTool 是一款主要用于对应用程序进行逆向工程的二进制工具，可以将应用程序资源解码成近乎原始的形式，并支持编译、签名等功能，对资源文件修改后可重新构建 APK。

● Dex2jar：Dex2jar 是一款将 dex 文件转换成 jar 文件的工具，转换原理是基于 asm 库将 dex 文件中的 Dalvik 字节码还原成可阅读的 Java 代码。然而，通过 Dex2jar 工具反编译出的应用程序资源存在缺失的情况。

● Jadx：Jadx 是一款功能更加强大，使用更加简单的反编译工具，可以从 Dex 和 APK 文件中获取 Java 源码，支持命令行操作，同时提供了图形用户界面。Jadx 与 Dex2jar 相比提供了更加简单的操作方式，但是采用 Jadx 对应用进行反编译比较占用内存。

本文中，我们希望通过静态源码分析工具获取应用的资源文件，通过对资源文件进行分析获取应用中的控件信息，由于 Dex2jar 工具可能对应用程序资源文件获取不完整，Jadx 工具对内存要求更高，因此我们采用 ApkTool 对应用进行反编译处理，获取应用的资源文件。

2.2.2 Android 静态分析工具

相较于 Android 自动化测试，Android 静态分析的难度更大，在这方面的研究相对更少，实际落地到工业界使用的产品或者工具并不多，但正因为静态分析具有挑战，使得静态分析工作的开展具有重大意义。Android 静态分析在评估应用的安全性、污点分析等领域具有重要作用。

Android 框架模型定义了非常多的回调函数，用于组件创建、设备状态更改等。Yang 等人 [39] 基于用户事件驱动的组件以及相关回调序列（生命周期回调和事件处理程序回调），提出了一个捕捉回调序列的程序表示，通过使用回调方法的上下文构建回调控制流图。随后，Yang 等人 [40] 基于先前的工作开发

[①]https://ibotpeaches.github.io/Apktool/
[②]https://www.kali.org/tools/dex2jar/
[③]https://github.com/skylot/jadx

了 GATOR[①]工具，并通过遍历分析上下文相关的程序控制流路径构建窗口转换图（WTG），WTG 是表示 Android 应用程序 GUI 的行为模型，展示了 GUI 窗口序列及其相关事件的回调情况，模型中的节点代表窗口，边代表窗口之间的转换，窗口之间的转换是由用户界面线程执行的回调进行触发。这是第一个对当前活动窗口堆栈建模以及堆栈中回调效果的研究。Arzt 等人 [41] 提出了污点分析框架 FlowDroid，FlowDroid 通过创建一个人工 main 方法模拟回调效果，但是未考虑事件处理程序，并且只能在单个组件中执行污点分析。Li 等人 [42] 则在 FlowDroid 的基础上进行改进，提出了可以跨越多个组件的静态污点分析器 IccTA，用于检测 Android 应用程序中组件之间的隐私泄漏。

Wang 等人 [43] 使用六款开源的 Android 应用程序对 FlowDroid，IccTA 和 GATOR 三个静态分析工具的回调序列覆盖率进行评估，结果证明 GATOR 工具的回调序列覆盖率最高，但是这三个静态分析工具对六款应用程序的回调序列覆盖率均无法达到 100%。究其原因是因为 FlowDroid 和 IccTA 没有以最通用的形式表示跨组件的回调，并且没有对未声明在 XML 清单文件中的类进行分析，GATOR 在对控制流分析时未考虑隐含的意图。因此，为了尽可能覆盖回调序列，本文中我们选用 GATOR 工具对应用进行静态分析。Yang 等人提出的 GATOR 可检测出应用程序 87.5% 的路径 [40]，虽然 GATOR 会遗漏部分测试路径，但是在实验中通过测试人员进行探索可以发现新的测试路径，基于 GATOR 的分析结果可以满足我们的需求。

自动化测试和静态分析各有优劣。静态分析方法相较于自动化测试不需要执行应用程序，所建立的模型更加全面，但是静态分析无法获得某些小部件在特定状态下是否被禁用等信息，然而根据自动化测试生成的模型可以提供这些信息。此外，对于只有在复杂的运行条件下才可能出现的行为，静态分析可以捕获到，但是自动化测试难以触发这些行为。最好的方式是将自动化测试和静态方式相结合，利用两者的优势对劣势进行互补，获得更加可靠的分析结果。Yang 等人 [33] 和 Azim 等人 [34] 采用了动静态相结合的方式，基于静态分析获得的信息，使自动化测试更加有效和完整。

在本文中，我们采用动静态程序分析相结合的方式，在静态程序分析中使用 ApkTool 和 GATOR 工具，利用动静态程序分析进行优势互补，为 Android 应用生成测试用例，引导众包工人在众包测试过程中完成测试任务。

[①]http://web.cse.ohio-state.edu/presto/software/gator/

2.3 众包测试

现如今，软件产品的更新迭代速度快，特别是在移动应用领域。移动应用复杂的使用场景和频繁的迭代周期，使得软件测试的时间被急剧压缩。在软件测试中，软件开发者希望能在短时间内通过软件测试获得大量真实有效的反馈，根据反馈的问题尽快修复产品的缺陷，提高软件产品质量以提供更好的用户体验。然而，不管是招募还是训练测试人员通常都需要较大成本，这使得很难获得大量专业的测试人员 [44]。因此，如何低成本地快速获得大量用户的真实反馈是当前软件测试面临的困难之一。

众包在软件测试上的应用可以解决软件测试的这一难题。众包软件测试，简称众包测试，是众包技术在软件开发生命周期测试阶段的应用。在众包测试过程中，任务发起者通过众包测试平台招募众包工人执行测试任务，众包工人根据个人喜好自由完成测试任务并提交测试报告，众包测试平台对测试报告进行审核并整理，最后向任务请求者交付最终的测试结果。根据测试报告中问题的严重程度，众包工人将会获得相应的奖励。

众包测试已在 QoE 测试、GUI 测试、可用性测试等方面逐渐被应用。Chen 等人 [45] 首次提出将众包技术应用于 QoE 测试，替代传统以人工测试为主的 QoE 测试。Komarov 等人 [46] 从多个方面比较实验室测试和众包 GUI 测试，实验结果显示两者相差无几，以此证明了众包 GUI 测试可以实现对传统实验室测试的有效补充。Liu 等人 [47] 通过对比传统实验室测试和众包在可用性测试的应用，研究表明众包可用性测试更易获得具有不同背景测试人员的测试数据，可以显著降低测试成本，但在质量方面传统实验室测试略胜一筹。

众包测试与传统实验室测试相比具有优势，如果需要对 Android 应用进行兼容性测试，测试在不同平台设备上应用的运行情况，仅通过传统实验室测试几乎无法完成 Android 应用的兼容性测试，受限于经济成本或测试资源，很难拥有较为全面的设备。通过众包测试可以很容易解决此问题，一群来自不同地区的众包工人使用自己的设备在真实环境中对应用进行测试，再将测试结果反馈给任务请求者，这可以在短时间获得测试结果反馈。

然而，众包测试平台对众包工人的招募没有要求，众包工人在平台上完成认证后，即可在平台中接收测试任务。在测试过程中，决定测试结果的关键因素是测试人员的专业知识。相较于专业测试人员，部分众包工人可能不具备软件测试技巧，对待测应用缺乏相应的领域知识。此外，众包工人可能不懂如何

编写测试用例，这些都会对测试结果产生影响。在传统的众包测试过程中，众包工人通过编写测试用例以检测应用是否存在异常。平台会获得大量包含冗余测试用例的测试报告，并且由于测试用例的质量参差不齐，使得整合报告的难度大大提升。

基于众包测试存在的问题，越来越多的专家学者开始关注对移动应用众包测试的改进和优化。Maria 等人 [29] 基于众包工人对应用程序的操作，收集应用程序数据，通过复制异常的上下文场景，以帮助开发人员定位异常的原因。Wang 等人 [48] 提出了根据众包测试报告的文本描述，采用局部的主动分类对众包测试报告中的真实故障进行分类。Liu 等人 [49] 提出使用截图信息帮助开发人员理解众包测试报告，基于众包测试报告中的截图，采用图像理解技术为截图生成描述性关键字。Hao 等人 [50] 将文本描述和屏幕截图相结合，通过计算两者的相似性，提出了一种众包测试报告处理技术 CTRAS，实现对众包测试报告的融合。Chen 等人 [51] 提出了一种测试报告增强工具 TRAF，TRAF 使用自然语言处理技术对众包测试报告进行预处理，并提出了三种策略分别增加环境、输入和被检查测试报告的描述，通过合并重复的测试报告进而提高测试报告质量。

现有研究大多关注于众包报告的整合处理方面，然而从流程进行优化，可以很大程度减轻报告整合的审核工作。因此，本文提出了一种众包测试流程的优化方法，希望从众包测试用例生成方面，为众包工人提供易于理解的众包测试用例，辅助众包工人执行测试任务，实现高效的移动应用众包测试。

2.4 本章小结

本章主要介绍本文的研究背景，包括 Android 自动化测试工具、Android 静态分析工具和众包测试的研究。通过对以往工作的梳理和分析，我们发现，目前还没有研究人员将自动化测试和静态分析相结合用于为众包测试人员生成测试用例。因此，我们创新性地提出通过自动生成测试用例，引导众包测试流程，实现人机协同完成众包测试。

第三章　相关工作

3.1　测试用例生成

测试用例生成对智力要求较高，也是众多测试活动中的关键步骤之一，因为它对整个测试过程的有效性和效率具有很大影响 [52–54]。测试用例可以有效反映软件产品的事件流，通过执行测试用例可以检验软件的功能、性能等是否存在问题。由此可见，测试用例的生成会直接影响测试结果，继而对软件的功能和用户体验产生重要影响。在软件测试中，所需的总成本、时间和工作量将取决于测试用例的总数 [55]。然而，测试用例生成是劳动密集型的任务，如果通过人工的方式生成测试用例，这将会耗费巨大的资源，并且人工生成测试用例要求用例生成者对待测应用具备一定的领域知识。在测试用例生成方面研究，主要包含以下几类：

基于符号执行的测试用例生成。符号执行技术早在 70 年代中期就由 King[56] 提出，King 提出使用符号值而不是具体值作为程序输入，并采用输入符号代替程序变量值进行表示。Cadar 等人 [57] 提出了一种符号执行工具 KLEE，KLEE 使用各种约束求解优化，简洁地表示程序状态，通过自动生成测试用例以便达到高代码覆盖。Notzli 等人 [58] 介绍了一种使用符号执行为 P4 程序自动生成测试用例的工具 p4pktgen，p4pktgen 生成的测试用例由测试包、表条目和预期路径组成，通过测试用例验证 P4 程序在设备上的行为是否符合预期。

基于模型测试的测试用例生成。模型测试是一种轻量级的形式化方法，它使用软件系统的模型进而推导测试用例。Miguel 等人 [59] 提出了一种基于 GUI 模型的 GUI 测试用例生成方法，GUI 模型由代表 Android 应用程序活动的状态以及 GUI 事件触发状态之间的转换组成，通过合并手动测试方法来实现更好的代码覆盖率。Choi 等人 [60] 提出了一种为 GUI 应用程序自动生成测试用例的 SwiftHand 框架，通过动态构建 GUI 的有限状态机模型，以达到减少重启时间并且提高测试覆盖率的目的。

基于搜索测试的测试用例生成。搜索测试使用搜索技术进行测试，尽可

能检测待测应用中的错误。Fraser 等人 [61] 提出了一个基于搜索算法的测试用例生成工具 EVOSUITE，它可以为 Java 编程语言编写的类自动生成单元测试用例，EVOSUITE 通过使用遗传算法产生测试用例集，尽可能地提高代码覆盖率。Yasin 等人 [62] 提出了一种基于 Q-Learning 技术生成 GUI 测试用例的方法 DroidbotX，DroidbotX 采用探索策略获得最小化事件的执行，进而最大化指令、方法和窗口的覆盖范围，从而提高覆盖率和崩溃检测。Arcuri[63] 提出了一种为 RESTful API web 服务自动生成测试用例的工具 EvoMaster，EvoMaster 采用一种全自动白盒测试方法，使用搜索算法自动生成测试用例以实现高代码覆盖率。

基于随机测试的测试用例生成。随机测试是通过产生随机的、独立的输入进而测试程序，将输出结果与软件规格进行比较，以验证测试输出是否正确。Gay 等人 [64] 使用随机测试生成方法，直接为多个覆盖标准生成测试输入，与相同大小的纯随机测试用例相比，生成的测试用例和随机测试用例都相对更少，并且同时保证了覆盖率。Haoyin[65] 提出了一个 SmartMonkey 工具，SmartMonkey 通过使用随机测试方法生成由用户和系统事件组成的测试用例，旨在提高故障检测能力。

目前测试用例生成的研究主要着眼于两个方面。一是通过为程序生成测试用例，旨在提升代码覆盖率，尽可能地检测程序代码是否存在故障。二是通过为自动化测试生成测试用例，旨在提升自动化测试对待测应用的路径覆盖率，尽可能地检测待测应用是否存在问题。很少有研究专注于为众包测试生成测试用例，进而实现对众包工人的测试引导。本文提出将文本描述和应用截图相结合，为众包工人提供易于理解的众包测试用例，旨在提升众包测试的效率和质量。

3.2 测试路径选择

所有应用页面在 Android 应用程序中都是通过 Activity 组件进行关联，在 Android 应用中通过点击事件或其他操作事件所触发的页面跳转，实质上就是 Activity 组件之间的转换。本文基于程序分析为应用程序构建窗口跳转模型，此模型通过有向图的方式表示 Activity 组件和 Activity 组件之间的跳转关系，其中图的节点是各个 Activity 组件对象，边是跳转关系。在窗口跳转模型中，代表跳转关系的边中包含具体的测试操作说明，以此展示窗口之间是如何实现跳

转的。本文生成的测试用例中包含一组测试步骤，用于引导众包工人进行测试任务，每个测试步骤都对应窗口跳转模型中的边。因此，为了生成测试用例，需要对窗口跳转模型进行测试路径选择，通过选择应用起始窗口到达指定窗口的最短测试路径，以此构建测试用例的步骤列表。

获取图中两个节点之间的最短路径其实就是求解最短路径的问题。最短路径是指从图中的某个节点到达另外一个节点所经过边的权重和最小的一条路径，Dijkstra（迪杰斯特拉）算法和 Floyd（弗洛伊德）算法都是常见的最短路径算法。

3.2.1 Dijkstra

Dijkstra 算法使用广度优先搜索策略，主要用于解决赋权图的单源最短路径问题。Dijkstra 算法基于贪心策略，其特点是以某个节点为中心向外层扩展，直到扩展所有节点。

算法实现思路分以下几步：

（1）指定一个中心节点 A，表示以 A 为中心向外层扩展。

（2）定义两个节点集合 S 和 U，S 代表已求出最短路径的节点集合（含从 A 到各节点的最短路径距离），U 代表未求出最短路径的点（含从 A 到各节点的距离）。

（3）初始时 S 中只含有中心节点 A，U 中是除 A 以外的其他节点。

（4）将 U 中距离 A 最近的节点取出，并加入到 S 中。

（5）接着更新 U 中节点以及其对应的最短路径距离。

（6）循环以上两个步骤，直到遍历结束所有节点。

如图 3-1所示是 Dijstra 算法示例图结构，我们通过示例对 Dijstra 算法进行说明。在此示例中，选用 A 为中心节点，计算从中心节点 A 到达各节点的最短距离。我们定义 $dis[x]$ 代表从节点 A 到节点 x 的最短距离，$path[x]$ 代表从节点 A 到节点 x 的最短路径中经过的中间节点。Dijstra 算法思路如下：

第 1 步：节点 B、D 与节点 A 直接相连，因此 $dis[B] = 40$，$dis[D] = 20$，其余节点没有与节点 A 相连，则最短距离是无穷大。可以对集合 S 和集合 U 进行初始化，$S = \{A(0)\}, U = \{B(40), D(20), C(\infty), E(\infty)\}$，括号中的数值代表节点 A 到该节点的最短距离。

第 2 步：节点 D 是 U 中距离 A 最近的节点，将 D 从 U 中移除并加入到 S 中，则 $S = \{A(0), D(20)\}$，$U = \{B(40), C(\infty), E(\infty)\}$。

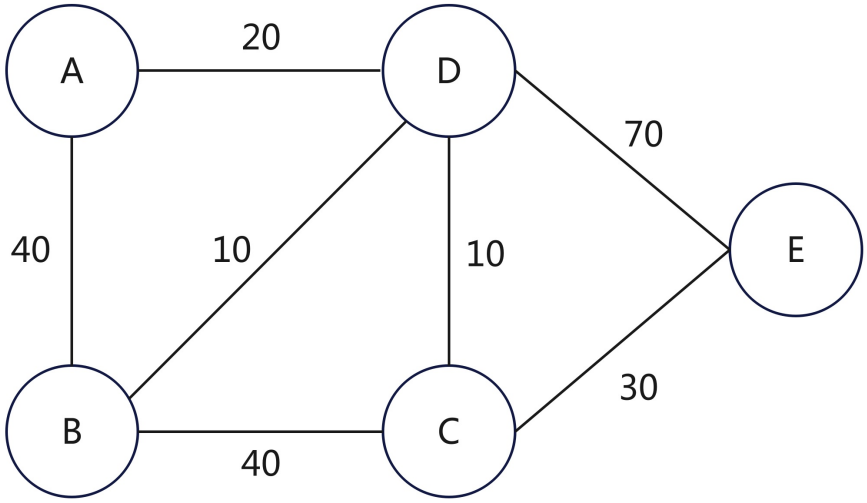


图 3-1: Dijstra 算法示例图结构

第 3 步：更新 U 中节点对应的最短距离。将 D 加入到 S 中后，获得与 D 相连的节点分别有 B、C 和 E（A 已在集合 S 中因此不考虑），$dis[B]$ 的值可以用 $dis[D]+dis[D-B]$ 表示，此时 $dis[D]+dis[D-B]$ 的值为 30 小于 U 中 $dis[B]$ 的值 40，因此更新 U 中 B 的值，并且将 $path[B]$ 赋值为 D。同理，U 中 C 和 E 的值分别更新为 30 和 90，$path[C]$ 和 $path[E]$ 都为 D，则 $U=\{B(30),C(30),E(90)\}$。

第 4 步：节点 B 和 C 都是 U 中距离 A 最近的节点，对于包含两个距离最短的节点，任取一个加入到 S 中，我们将 B 加入到 S 中，则 $S=\{A(0),D(20),B(30)\}$，$U=\{C(30),E(90)\}$。

第 5 步：更新 U 中节点对应的最短距离。将 B 加入到 S 中后，获得与 B 相连的节点 C（A、D 已在集合 S 中因此不考虑），$dis[C]$ 的值可以用 $dis[B]+dis[B-C]$ 表示，但 $dis[B]+dis[B-C]$ 的值为 70 大于 U 中的 $dis[C]$ 的值 30，因此不对 U 进行更新，$U=\{B(30),C(30),E(90)\}$。

第 6 步：节点 C 是 U 中距离 A 最近的节点，将 C 从 U 中取出并加入到 S 中，则 $S=\{A(0),D(20),B(30),C(30)\}$，$U=\{E(90)\}$。

第 7 步：更新 U 中节点对应的最短距离。将 C 加入到 S 中后，获得与 C 相连的节点 E（B、D 已在集合 S 中因此不考虑），$dis[E]$ 的值可以用 $dis[C]+dis[C-E]$ 表示，此时 $dis[C]+dis[C-E]$ 的值为 60 小于 U 中的 $dis[E]$ 的值 90，因此更新 U 中 E 的值，并且将 $path[E]$ 赋值为 C，则 $U=\{E(60)\}$。

第 8 步：节点 E 是 U 中距离 A 最近的节点，将 E 从 U 中取出并加入到 S 中，则 $S=\{A(0),D(20),B(30),C(30),E(60)\}$，$U\{\}$。此时 U 已为空集，遍历

结束。

通过以上步骤，获得从节点 A 到达 B、C、D、E 的最短距离分别是 20、30、30、60，*path*[*x*] 中记录 A 到其他节点的最短路径中经过的中间节点。

3.2.2 Floyd

Floyd 算法是用于解决任意两点之间最短路径问题的算法，采用了贪心和动态规划策略进行问题求解。Floyd 算法思路如下：

我们定义两个矩阵 Dis 和 Path，矩阵 Dis 是距离矩阵，用于存放任意两个节点之间的最短路径距离值，*Dis*[*i*][*j*] 表示节点 i 到节点 j 的距离。矩阵 Path 是路径矩阵，用于记录任意两个节点之间的最短路径，*Path*[*i*][*j*] 表示节点 i 到节点 j 经过的中间节点。对于每一个节点 k 检查 *Dis*[*i*][*k*] + *Dis*[*k*][*j*] 是否小于 *Dis*[*i*][*j*]，如果小于则说明从节点 i 到节点 k 再到节点 j 的路径距离，比节点 i 直接到节点 j 的路径距离更短，那么使用 *Dis*[*i*][*k*] + *Dis*[*k*][*j*] 替换 *Dis*[*i*][*j*] 的值，并将 k 赋值给 *Path*[*i*][*j*]。遍历结束所有节点，得到的 *Dis*[*i*][*j*] 是从 i 到 j 最短路径的距离，*Path*[*i*][*j*] 是从 i 到 j 经过的中间节点。

3.2.3 算法选择

Dijkstra 算法适用于处理单源最短路径问题，Dijkstra 算法每次从未求出最短路径的点中取出距离中心节点最近的点，通过这个点更新剩下未求节点的最短路径的距离，时间复杂度为 $O\left(n^2\right)$，缺点是不能处理存在负权的图。Floyd 算法是处理多源最短路径，时间复杂度为 $O\left(n^3\right)$，可以处理存在负权的图。本文通过为 Android 应用构建窗口跳转模型，模型中两节点之间的边表示窗口跳转关系，不存在负向边。根据应用起始窗口到达未覆盖路径所在窗口的最短路径形成测试用例，最短路径的求解可以看做是单源最短路径问题，采用 Dijkstra 算法更适合。因此，本文采用更适用我们技术场景的 Dijkstra 算法，作为测试路径选择的算法。

3.3 本章小结

本章主要介绍本文的相关工作，测试用例生成和测试路径选择的研究。通过对以往研究分析可以发现，基本没有研究专注于为众包测试生成测试用例，

以便为众包工人提供测试引导。在测试路径选择的算法选择上，通过对 Dijkstra 和 Floyd 算法进行分析比对，我们采用更适用当前技术场景的 Dijkstra 算法。

第四章　测试用例生成技术

这一章将介绍基于动静态程序分析的 Android 应用众包测试用例生成技术的详细设计。整个框架的设计思想如图4-1所示，结合动静态程序分析的测试行为构建应用的窗口跳转模型，基于窗口跳转模型进行路径选择，选择从应用起始窗口到自动化测试未覆盖的测试行为所在窗口的最短测试路径，结合未覆盖的测试行为生成测试用例，将测试用例分配给众包工人，引导众包工人探索新异常，以人机协同的方式协助众包工人完成测试任务，实现低成本、高效率的众包测试。

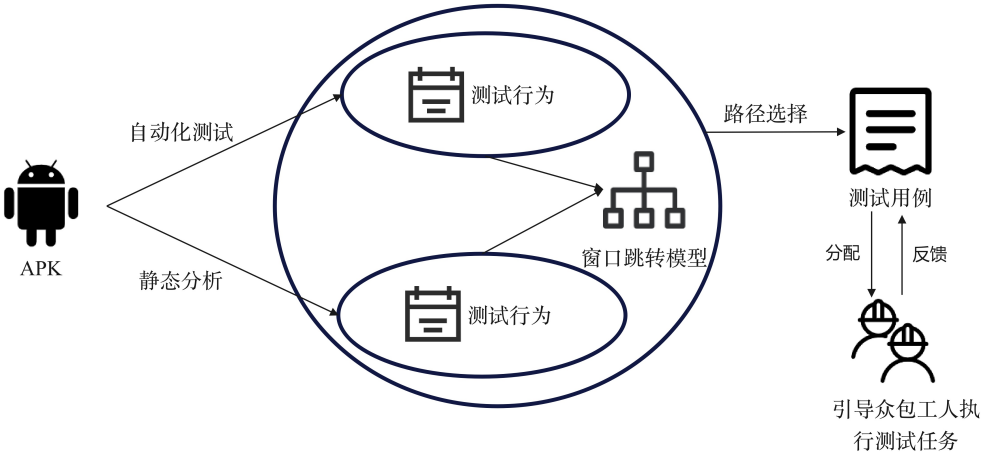


图 4-1: 测试用例生成技术的设计

在传统的众包测试过程中，需要众包工人生成测试用例，人工生成测试用例受限于众包工人的专业水平。本文提出了一种新颖的众包测试用例生成技术，具体设计如下：在动态程序分析中，采用 Mooctest 自动化测试工具对 APK 进行测试，测试结果中包含日志信息和应用截图等，提取操作事件并构建包含应用截图的测试行为列表，为测试行为生成测试操作描述。在静态分析中，先采用 ApkTool 工具对 APK 进行反编译处理，获取应用的资源文件，通过分析并建立控件名称和控件真实名称映射。然后，采用经过优化的 GATOR 工具获取应用的窗口跳转信息，将控件窗口跳转信息中的控件名称匹配对应的控件真实名称，构建包含控件真实名称的测试行为列表，为测试行为生成测试操作描

述。完善自动化测试和静态分析的测试行为列表，并获得包含应用截图和控件真实名称的全面测试行为列表。基于此列表构建应用的窗口跳转模型，并获得自动化测试过程中未覆盖的测试行为，这些是待执行的测试行为。采用 Dijkstra 算法进行测试路径选择，选择从应用起始窗口到达待执行测试行为所在窗口的最短测试路径，通过最短测试路径和待执行测试行为的测试操作描述，形成测试用例中的步骤描述，结合应用截图辅助步骤说明，以此形成测试用例。将测试用例分配给众包工人，引导众包工人对应用进行测试。图4-2是此技术的具体框架图。

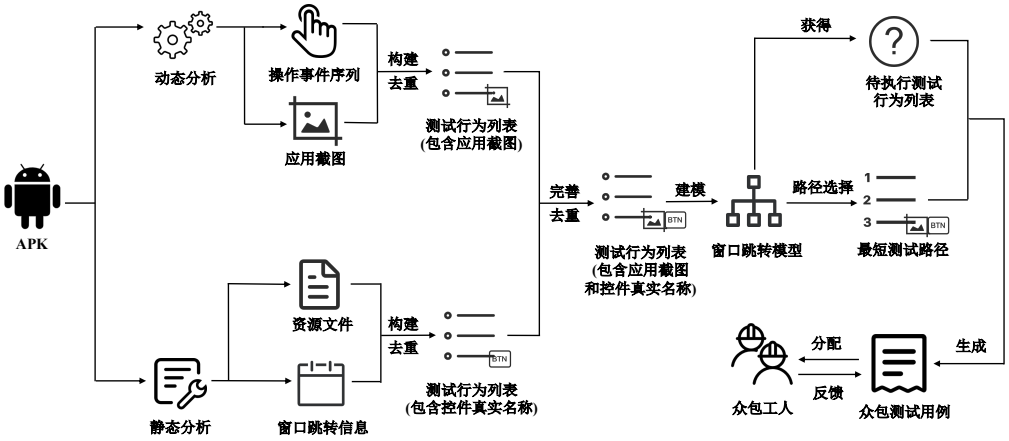


图 4-2: 基于动静态程序分析的 Android 应用众包测试用例生成技术框架

该技术最终集成在慕测众测平台中，根据任务请求者上传的 APK，自动为待测应用生成众包测试用例，引导众包工人对自动化测试未覆盖的测试路径进行探索。该技术通过对传统众包测试机制进行优化，以提高众包测试的效率和质量。

4.1 自动化测试

Activity 是 Android 应用程序的关键组件之一，是应用窗口的载体。Android 应用程序通过调用 Activity 生命周期中的回调方法对 Activity 实例进行启动和销毁。Dialog 和 Menu 是 Activity 中常见的形式，本文对应用程序中 Activity，Dialog 和 Menu 的实体类统一用“窗口”表示，窗口在应用中以页面的形式进行展示。

MoocTest 自动化测试工具在测试过程中会记录日志信息和应用截图。通过

读取自动化测试结果文件中的日志数据，从中提取操作事件信息，对操作事件进行遍历并构建测试行为列表 TestAction_Auto。由于自动化测试工具可对多台设备进行测试，不同设备的测试结果中可能拥有相同的测试行为，因此需要对 TestAction_Auto 去重，去重后为每个测试行为生成测试操作描述信息。自动化测试工具会保存测试过程的应用截图，通过遍历测试行为匹配对应的应用截图。图4-3展示了自动化测试模块处理流程，下面将详细介绍自动化测试模块的分析过程。

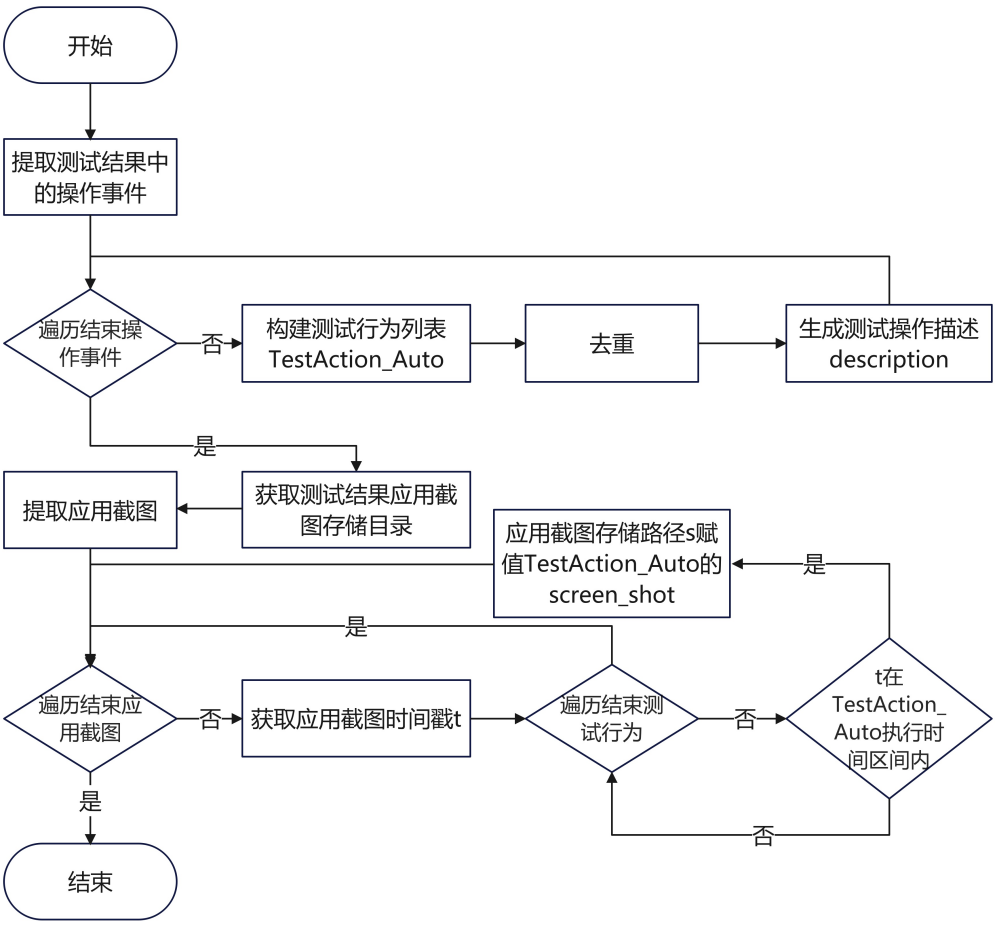


图 4-3: 自动化测试模块流程图

提取操作事件。本文选用 MoocTest 自动化测试工具，通过模拟用户交互事件和系统监听事件对应用进行自动化测试，该工具支持借助人工测试脚本的方式丰富输入事件的输入值，并基于深度优先遍历算法尽可能地遍历应用组件。自动化测试过程中会产生大量的日志文件，文件内容复杂繁多，其中包含测试过程中的操作事件信息，通过人工很难获得有效的测试信息，因此需要通过程序提取操作事件。

如图4-4是自动化测试工具在应用 Budget Watch 上产生的操作事件示例，其中“activityBeforeAction”和“activityAfterAction”代表起始窗口和目标窗口，“timeBeforeAction”和“timeAfterAction”记录操作事件发生前后的时间，“message”存储操作事件信息，“type”则输出操作事件类型。图4-4中的示例表示在某个时间区间内，通过在应用 Budget Watch 的 IntroActivity 窗口上点击 id 为 skip 的控件，应用将会跳转到 MainActivity 窗口。为众包工人提供控件真实名称（应用窗口上显示的控件名称）有助于完成测试任务，但是从自动化测试结果数据中无法获得控件真实名称，可以通过静态分析解决此问题，这在4.2节中会进行说明。


```
{
    "activityBeforeAction": ".intro.IntroActivity",
    "activityAfterAction": ".MainActivity",
    "timeBeforeAction": "2022-03-04 14:21:25,943",
    "timeAfterAction": "2022-03-04 14:21:26,314",
    "message": "Click widget protect.budgetwatch:id/skip, bounds: '[24,1729][324,1920]'",
    "type": "click"
}
```


图 4-4: 操作事件格式

构建测试行为列表。由于自动化测试和静态分析的结果数据格式相差甚远，为了便于模型构建以及测试用例生成，需要使用统一的数据结构将自动化测试和静态分析的结果数据进行结构化处理。

我们定义 TestAction 表示测试行为集合，使用 TestAction 统一自动化测试和静态分析的结果数据的格式。TestAction 的存储结构如表4-1所示。activity_source 和 activity_target 代表起始窗口和目标窗口，对应操作事件中的 activityBeforeAction 和 activityAfterAction，time_before_action 和 time_after_action 记录测试行为执行前后的时间，对应操作事件中的 timeBeforeAction 和 timeAfterAction。event_type 表示测试行为的事件类型，对应操作事件中的 type，message 存储事件信息，对应操作事件中的 message，description 保存测试操作描述，screen_shot 记录应用截图，widget_real_name 代表控件真实名称，widget_type 表示控件类型，operation_object 代表操作对象。数据存储中设置 data_type 字段，用于区分测试行为的类型，其中 1 代表从自动化测试中获得的测试行为，2 代表从静态分析中获得的测试行为，3 代表从自动化测试和静态分析均可获得的测试行为。

表 4-1: 测试行为 TestAction 的具体存储结构

字段	类型	含义
id	BIGINT	唯一标识
activity_source	varchar	起始窗口
activity_target	varchar	目标窗口
time_before_action	varchar	执行前的时间
time_after_action	varchar	执行后的时间
event_type	varchar	事件类型
operation_object	varchar	操作对象
widget_id	varchar	控件唯一标识
widget_name	varchar	控件名称
widget_real_name	varchar	控件真实名称
widget_type	varchar	控件类型
description	varchar	测试操作描述
message	varchar	事件信息
screen_shot	varchar	应用截图
edge_type	varchar	边类型
callbacks	varchar	回调信息
data_type	INT	类型

从自动化测试的结果文件中提取操作事件序列，将操作事件按照 TestAction 的定义，构建测试行为进行存储，并对 TestAction 中的 data_type 属性标识为 1，以此获得自动化测试的测试行为列表 TestAction_Auto。

由于使用多设备进行自动化测试，测试结果中包含冗余的数据，我们设定 TestAction_Auto 的起始窗口 activity_source、目标窗口 activity_target、和事件信息 message 完全一致，则认为是重复的 TestAction_Auto。使用这三个属性对 TestAction_Auto 进行去重，可以提取有效的测试行为，为后续处理提供可靠保证。

生成测试操作描述。通过前面的工作，已获得测试行为列表 TestAction_Auto，TestAction_Auto 中包含 message 事件信息属性。表4-2展示了 TestAction_Auto 中 message 属性的类型，可以对 message 进行分析，生成 TestAction_Auto 的测试操作描述 description，description 属性对测试用例的生成起着关键性作用，是构成测试用例步骤描述的基础。

Android 应用程序事件可以分为以下两类：

（1）系统事件：系统事件作用于设备硬件端，主要包含屏幕旋转、通过返回键返回上级窗口、按 Home 键返回手机桌面、短信通知、应用通知等

事件。

（2）用户交互事件：用户交互事件作用于当前屏幕的窗口组件，通常包括在 GUI 元素（如按钮、图像或文本块）中点击、滑动或输入文本等事件。

表 4-2: TestAction_Auto 的 message 类型

编号	部分 message 事件信息	部分 description 测试操作描述
1	**Click Return button because this page has done**	点击返回键返回上级页面
2	**Click Home button has tries more than 3 times**	按 Home 键回到手机桌面
3	**Click Confirm button for Android Alert**	点击弹框中确认/允许等相近含义按钮
4	**Click Cancel button for Android Alert**	点击弹框中取消/拒绝等相近含义按钮
5	**Click widget protect.budgetwatch:id/skip**	点击控件 id 为 skip 的控件
6	**Click widget //android.widget.CheckedTextView contains(@text,'CSV')**	点击控件内容为 CSV 的控件
7	**Click widget //android.widget.LinearLayout contains(@index'0')**	点击列表中第 1 个元素
8	**Input android:id/input use value test**	在控件 id 为 input 的控件中输入值 test
9	**startX: 810, startY: 960, endX: 270, endY: 960**	从右往左滑动页面
10	**startX: 270, startY: 960, endX: 810, endY: 960**	从左往右滑动页面
11	**startX: 810, startY: 960, endX: 810, endY: 560**	从下往上滑动页面
12	**startX: 810, startY: 560, endX: 810, endY: 960**	从上往下滑动页面

表4-2中编号 1-7 均是点击操作，此类 TestAction 的事件类型 event_type 用“click”表示，其中编号 1-4 通过 message 内容较为容易生成测试操作描述 description。编号 1 和 2 属于系统事件，编号 3 和 4 表示对弹框中的窗口组件进行操作，属于用户交互事件。编号 5-7 的 message 内容以“Click widget”开头，表示对控件进行点击操作，均属于用户交互事件，事件内容较为复杂。这三种 message 中包含事件操作的控件信息，需要对 message 内容进行处理才能获得具体的控件 id、控件名称以及列表元素索引，然后生成对应的 description。编号 8 中 message 内容以“Input”开头，表示进行文本输入操作，此类 TestAction 的事件类型 event_type 用“input”表示，message 中包含控件 id 以及输入值等信息，生成的 description 表示在指定控件中输入指定值。编号 9-12 是对页面进行滑动操作，此类 TestAction 的事件类型 event_type 用“swipe”表示，message 中展示了页面滑动范围，对滑动范围进行分析，可以生成包含页面滑动方向描述的 description。

图4-5展示了 TestAction_Auto 的测试操作描述 description 生成流程图。遍历自动化测试构建的测试行为列表，校验事件类型 event_type 是否为

“click”，基于表4-2的类型生成测试操作描述 description。值得注意的是，表4-2中编号 5-7 需要使用正则匹配才可获得对应的控件 id、控件名称和列表元素索引，控件 id 和控件名称不一定是控件的真实名称，在后续会为测试操作描述中的控件提供真实的名称展示。对于事件类型 event_type 不是“click”的测试行为，则校验是否为“input”或者“swipe”，并根据表4-2生成测试操作描述 description。最后校验测试行为是否发生了窗口跳转，对于发生窗口跳转的测试行为需要在 description 补充跳转信息。

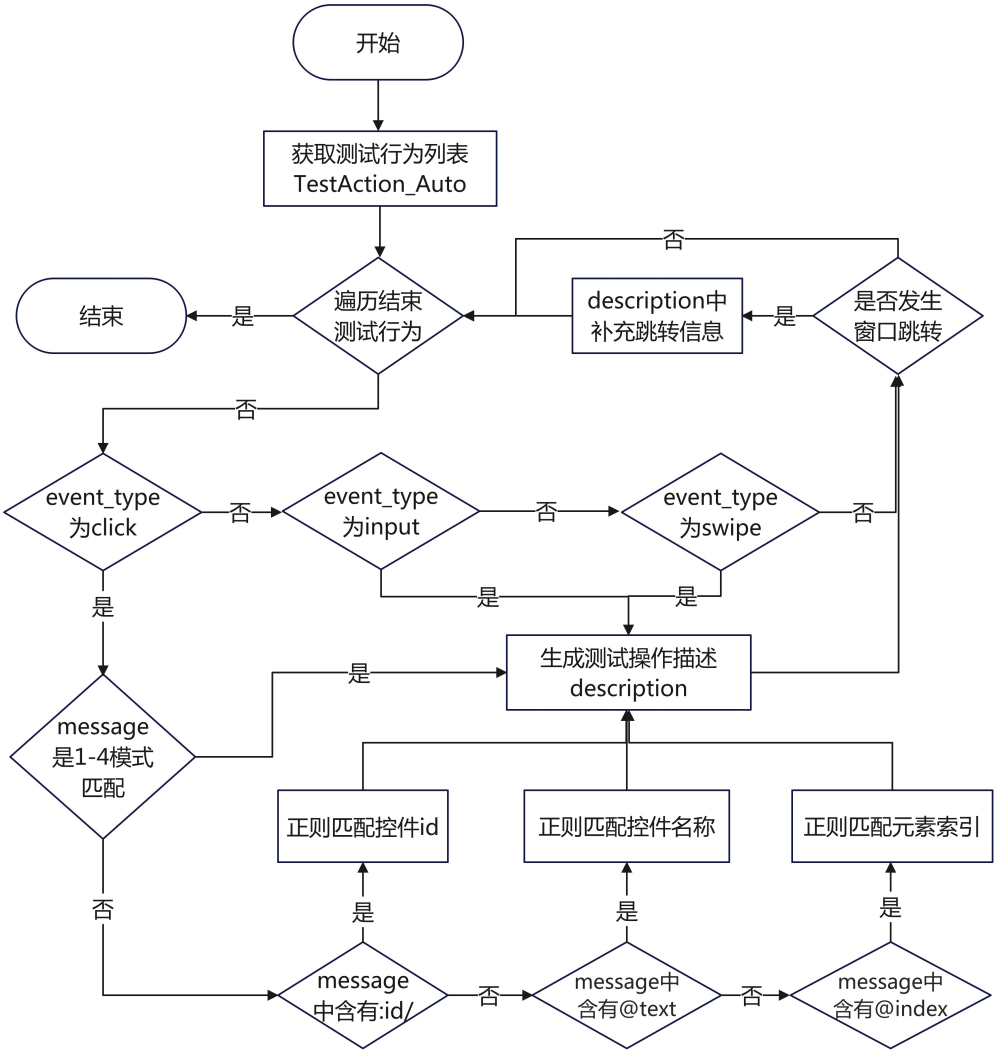
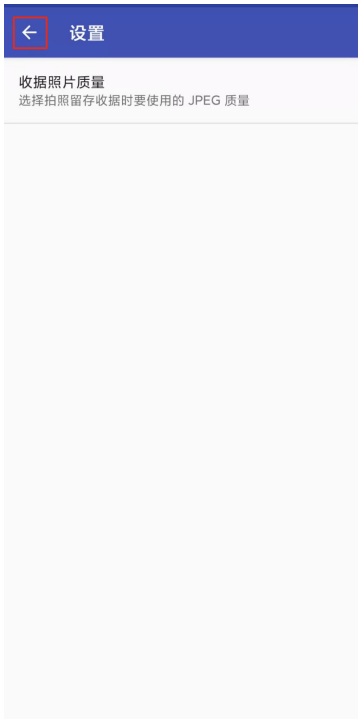
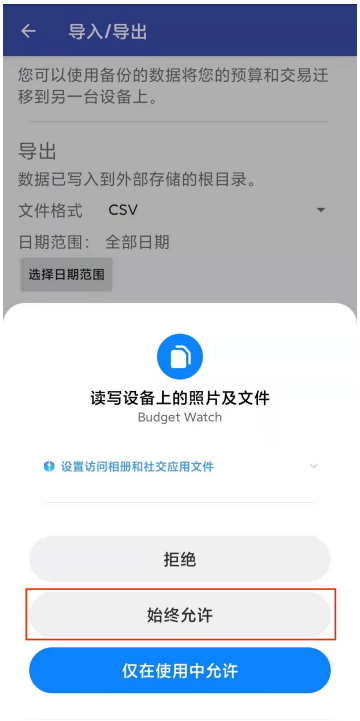


图 4-5: TestAction_Auto 的测试操作描述 description 生成流程图

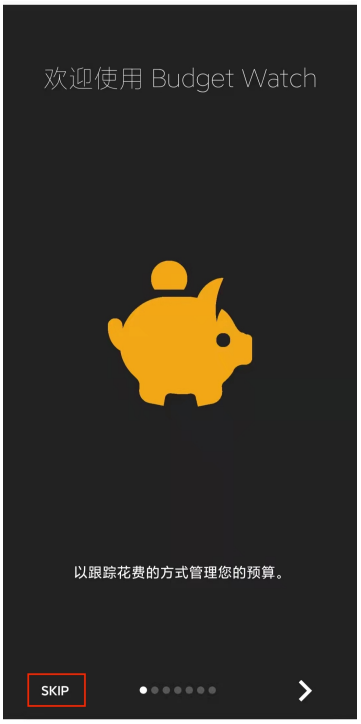
匹配应用截图。在众包测试过程中，为了便于开发者快速定位产品问题，通常要求众包工人书写测试用例时提交应用截图。由此可见，应用截图对于测试用例具有重要意义。因此，该技术提出使用应用截图辅助测试步骤说明。



(a) 点击返回键返回上级窗口

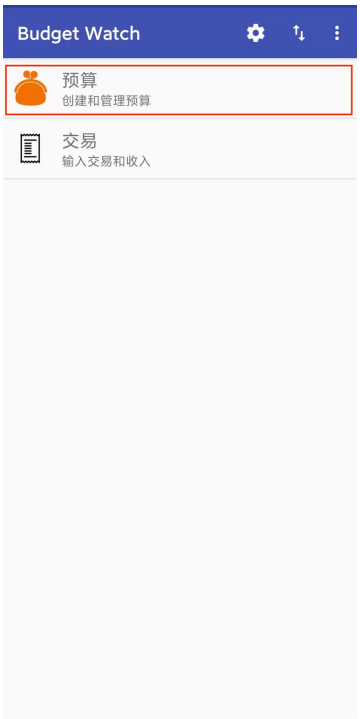


(b) 点击弹框中确认/允许等相近含义按钮



(c) 点击控件 id 为 skip 的控件

(d) 点击控件内容为 CSV 的控件



(e) 点击列表中第 1 个元素

(f) 在控件 id 为 input 的控件中输入值 test

图 4-6: 不同测试行为对应的应用截图

在自动化测试过程中，MoocTest 自动化测试工具不仅会记录操作事件，还会保存应用截图。但是应用截图和操作事件之间存在时延，为了保证应用截图和测试行为匹配的准确性，允许有 1200ms 的时延。

本文定义应用截图列表 P，对于任意应用截图，可以表示为P_i=<s_i, n_i, t_i>，s_i 表示应用截图文件的存储路径，n_i 代表应用截图文件名，t_i 记录应用截图的时间戳。自动化测试结果中包含应用截图文件，这些文件是以设备编号和应用截图时间戳命名，通过对应用截图文件进行遍历，从应用截图文件的文件名 n_i 中提取应用截图时间戳 t_i，将应用截图文件上传至云端进行存储，并将存储路径赋值给 s_i，以此形成应用截图 P_i。对于任意测试行为$TestAction_Auto_j$，在 P 中必定存在应用截图P_i，满足 $t_i \in [time\_before\_action_j - 1200, time\_after\_action_j + 1200]$，从而建立$P_i$ 与$TestAction_Auto_j$ 的映射关系，并将P_i 中的存储路径s_i 赋值给$TestAction_Auto_j$ 的$screen_shot_j$。

充分利用应用截图可以直观地向众包工人展示测试行为所操作的页面，应用截图可以辅助测试用例中的测试步骤说明，有助于引导众包工人完成测试任务。图4-6展示应用程序“Budget Watch”和“记乐部”中部分测试行为对应的应用截图。

4.2 静态分析

Android 应用程序的静态程序分析是指将程序的源码作为输入，在不执行程序的情况下检查代码结构、语句序列以及变量值在不同函数调用中的处理进而产生分析结果。采用语法分析、词法分析、数据流分析或控制流分析等技术遍历程序路径，对程序属性进行检查。

在静态分析模块中采用两款工具对 Android 应用进行静态分析，先采用 ApkTool 工具对 APK 进行反编译处理，获取应用资源文件，通过对应用资源文件进行分析获得应用的控件信息，从控件信息中生成控件名称和控件真实名称的映射关系。随后，采用经过优化的 GATOR 工具对 APK 进行静态分析，GATOR 工具通过遍历并分析上下文相关的程序控制流路径，输出窗口跳转信息。通过对窗口跳转信息进行分析，构建静态分析的测试行为列表，对测试行为列表去重，并按照一定的规则为测试行为生成测试操作描述，以此获得包含控件真实名称的测试行为列表。图4-7是静态分析模块流程图，主要包含以下内容。

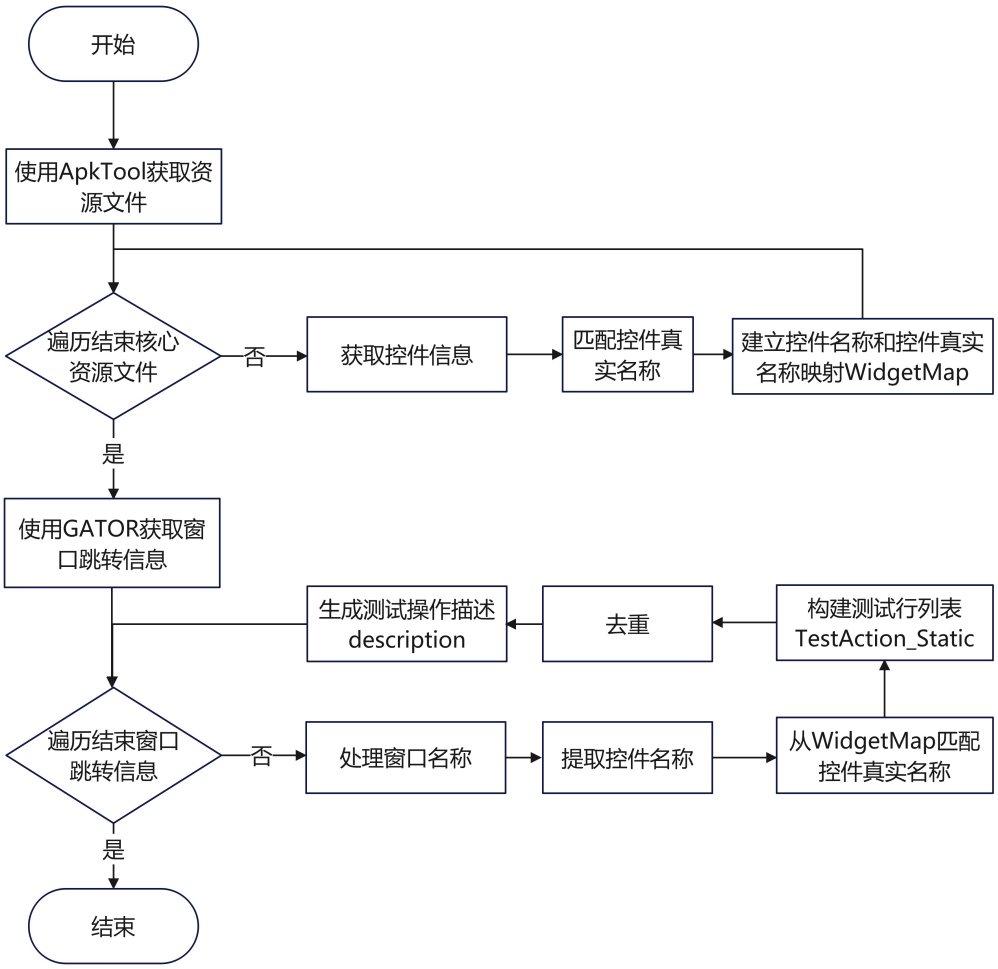


图 4-7: 静态分析模块流程图

反编译 APK。Android 操作系统和传统的计算机操作系统类似，也有其特有的运行环境，在 Android 操作系统中支持 APK 格式的程序安装包，不支持其他操作系统的可执行文件。APK 中包含应用源码和资源文件等信息，但是在应用程序代码转换成 APK 格式文件时需要经过编译、压缩和打包等阶段，我们无法直接从 APK 中获取资源文件。因此，需要对 APK 进行反编译处理，进而获取应用的资源文件等信息。

本文选用 ApkTool 工具对 APK 进行反编译，ApkTool 是一款二进制 Android 应用程序工具，主要用于对应用程序进行逆向工程。ApkTool 可以将应用程序资源解码成近乎原始的形式，并支持对资源进行修改后重新构建 APK。使用 ApkTool 对待测应用进行反编译处理，获得应用配置文件和资源文件，如 AndroidManifest.xml 文件和 res 文件夹下的资源文件。AndroidManifest.xml 是 Android 应用的全局配置文件，提供了关于该应用的重要信息，例如包名和

应用中的窗口声明等信息。res 文件夹主要用来存放资源信息，包含 menu、layout、values 等文件夹，这些文件夹中包含大量控件信息。

表4-3展示了 res 文件夹中的部分核心资源文件内容示例。res 文件夹中存有大量以 values 开头的文件夹，如 values、values-zh-rCN 等，文件夹中的 strings.xml 文件定义了在开发中使用的字符串变量，其中 values-zh-rCN/strings.xml 展示了字符串变量的中文含义，用来实现国际化。此外，在 res/menu 菜单文件夹、res/layout 和 res/layout-v17 布局文件夹中也包含很多 xml 文件，这些 xml 文件中也记录了控件信息，layout-v17 文件夹中存储了在 API 17+ 设备上产生的结果。

表 4-3: res 文件夹中的部分核心资源文件内容示例

示例	文件	备注
<string name="save">Save</string> <string name="account">Account</string> <string name="skip_button">SKIP</string>	res/values/strings.xml	控件变量名及其含义
<string name="save"> 保存 </string> <string name="account"> 账户 </string>	res/values-zh-rCN/ strings.xml	控件变量名及其中文含义
<item android:id="@id/action_save" android:title="@string/save"/>	res/menu/edit_menu.xml	含 android:title 属性
<Button android:id="@id/skip" android:text="@string/skip_button"/>	res/layout/intro_layout.xml	含 android:text 属性
<TextView android:labelFor="@id/account" android:text="@string/account"/>	res/layout-v17/transaction _view_activity.xml	含 android:labelFor 属性

建立 WidgetMap 映射。WidgetMap 是控件名称和控件真实名称的映射，生成 WidgetMap 映射的操作流程如图4-8所示。

首先校验是否存在 values-zh-rCN 文件夹，如果存在则读取 values-zh-rCN 文件夹中的 strings.xml 文件，此文件包含开发中使用的字符串变量名及其所对应的中文含义，如表4-3第二行所示，从而建立变量名及其含义的映射 NameMap。部分应用的反编译结果中不包含 values-zh-rCN 文件夹，对于此类情况则直接读取 values/strings.xml 文件，此文件中的数据比 values-zh-rCN/strings.xml 的内容更加全面，对比表4-3前两行，values/strings.xml 中多有包含变量名为“skip_button”和其所对应的含义“SKIP”。基于上述操作建立或补充 NameMap 映射。

然后遍历 menu、layout 和 layout-v17 文件夹中的文件，res 文件夹中的 menu

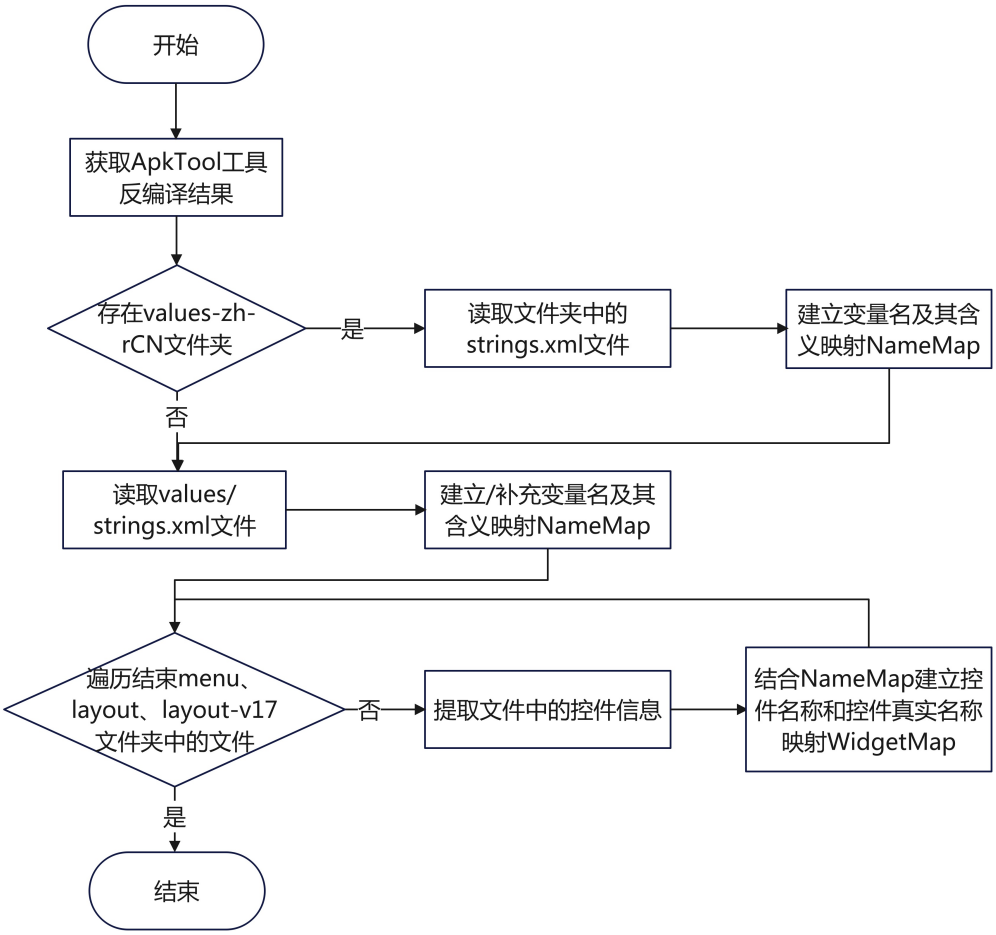


图 4-8: 建立控件名称和控件真实名称 WidgetMap 映射的流程图

菜单文件夹和 layout 布局文件夹中记录了控件信息，从中提取文件中的控件信息。由表 4-3后三个示例所示，根据 android:id 和 android:labelFor 属性可以获取控件名称，例如“action_save”、“skip”、“account”。根据 android:text 和 android:title 属性可以获取控件变量名，例如“save”、“skip_button”、“account”，进而建立控件名称和控件变量名的对应关系。基于控件变量名从 NameMap 中匹配对应的控件真实名称，从而建立控件名称和控件真实名称的映射 WidgetMap。例如控件名称为“action_save”的控件，其控件变量名是“save”，对应的控件真实名称是“保存”，因此，将控件名称“action_save”和控件真实名称“保存”的对应关系存储到 WidgetMap 中。

获取窗口跳转信息。通过对 GATOR 工具进行优化，获取完整的窗口跳转信息。GATOR 是一款开源工具，可以对 APK 进行静态分析。GATOR 通过遍历并分析上下文相关的程序控制流路径，获得触发堆栈变化的回调以及由它们

触发的回调，基于这些回调信息生成窗口跳转信息，窗口跳转信息展示了 GUI 窗口序列及其相关事件的回调情况。

在使用 GATOR 对 APK 进行静态分析过程中，发现 GATOR 对多个 APK 都出现执行失败的现象，并且 GATOR 输出的窗口跳转信息中不包含窗口跳转之间的事件信息。通过阅读并分析 GATOR 源码，对 GATOR 工具进行优化，主要有以下几点优化：

（1）修复 GATOR 工具对部分 APK 执行失败的问题。

（2）将输出结果中新增事件信息（表 4-4中的 Event 属性）。

（3）对经过优化的 GATOR 进行重新构建。

表 4-4展示了经过优化的 GATOR 工具输出的窗口跳转信息示例，窗口跳转信息中包含窗口信息、事件信息、回调信息和栈操作信息等。对于新增的 Event 属性可以定义为 $Event=<c, idNode, pid, id>$。其中，c 代表事件操作对象类型（android.view.MenuItem），idNode 表示控件信息（WID[2131689727|action_save]898），pid 记录父节点 id（1200），id 存储事件 id（3663）。idNode 可用 tag + “[” + i + “|” + name + “]” + id 表示，其中 tag 代表标签（WID），i 表示整形序号（2131689727），name 记录控件名称（action_save），id 存储控件 id（898）。GATOR 输出的窗口跳转信息数据中还包含回调信息和栈操作信息，这些信息可以展示事件在代码层的执行过程。

表 4-4: GATOR 工具对应用“Budget Watch”输出的部分窗口跳转信息

属性	含义	示例	备注
Source Window	起始窗口	OptionsMenu[protect.budgetwatch.TransactionViewActivity]120	
Target Window	目标窗口	ACT[protect.budgetwatch.TransactionViewActivity]1206	
Event	事件	MenuItemINFL[android.view.MenuItem,WID[2131689727|action_save]898,1200]3663	新增
Event Type	事件类型	click	
Event Callbacks	事件回调	<protect.budgetwatch.TransactionViewActivity: boolean onOptionsItemSelected(android.view.MenuItem)>	
Lifecycle Callbacks	生命周期回调	<protect.budgetwatch.TransactionViewActivity:void onCreate(android.os.Bundle)> <protect.budgetwatch.TransactionViewActivity:void onResume()>	
Stack Operations	栈操作	POP Menu[protect.budgetwatch.TransactionViewActivity]1200 PUSH ACT[protect.budgetwatch.TransactionViewActivity]1206	

对于任意一个$Event_i$，其中的c_i 代表事件操作对象的类型。如表 4-5展示了常见的几种事件操作对象类型，这些都是 android.view.View 的子类，通过

操作对象类型可以为测试操作描述增加操作对象类型说明，例如表4-5中的android.view.MenuItem 类型表示在菜单栏中进行操作，android.widget.ListView 类型代表对列表进行操作。

表 4-5: 事件操作对象类型

类型	含义
android.view.MenuItem	菜单栏
android.widget.ListView	列表
android.widget.EditText	文本编辑
android.widget.Button	按钮
android.widget.TextView	文本

构建测试行为列表。通过 GATOR 可以获得 Android 应用的窗口跳转信息，接下来根据窗口跳转信息构建测试行为列表 TestAction_Static。下面将介绍构建 TestAction_Static 的具体步骤:

（1）处理窗口信息：窗口跳转信息中包含 Source Window 和 Target Window 两个属性，由于这两个属性与自动化测试的测试行为列表 TestAction_Auto 中的起始窗口 activity_source 和目标窗口 activity_target 格式不一致，为了在4.3节中构建 Android 应用的窗口跳转模型，需要将两者格式进行统一。

如表4-4中的 Source Window 和 Target Window，可以通过正则表达式提取此类的窗口信息（TransactionViewActivity）。然而，Source Window 和 Target Window 可能是弹框，例如 DIALOG[android.app.AlertDialog]3240, alloc: <protect.budgetwatch.Import ExportActivity$1: void onClick(android.view.View)>，此类数据对于用户而言不具备直观的展示，需要对此类数据进行处理，我们采用窗口信息、弹框标志和弹框 id 进行拼接处理，获得 ImportExportActivity-Dialog-3240，以此形成弹框类型窗口的 Source Window 和 Target Window。通过对窗口跳转信息中的 Source Window 和 Target Window 属性进行处理，基于处理后的 Source Window 和 Target Window 形成 TestAction_Static 的 activity_source 和 activity_target 两个属性。

（2）构建 TestAction_Static：遍历窗口跳转信息，基于窗口跳转信息的 Event 属性，使用 c 字段可以生成 TestAction_Static 的 operation_object 操作对象，使用 idNode 字段的 id 属性可以生成 TestAction_Static 的 widget_id 控件 id，使用 idNode 字段的 name 属性可以生成 TestAction_Static 的 widget_name 控件名称，并将 widget_name 从 WidgetMap 中匹配对应的控件真实名称，以

此生成 TestAction_Static 的 widget_real_name 控件真实名称。根据窗口跳转信息的 Event Type 属性生成 TestAction_Static 的 event_type 事件类型，并对 TestAction_Static 中的 data_type 属性标识为 2。以此构建测试行为列表 TestAction_Static。

（3）去重 TestAction_Static：通过以上两步可以获得 TestAction_Static，我们认为如果 TestAction_Static 的起始窗口 activity_source、目标窗口 activity_target、和控件名称 widget_name 完全一致，则可认定是重复的 TestAction_Static，使用以上三个属性对 TestAction_Static 进行去重。

表 4-6: 窗口跳转信息的事件类型 Event Type 分类

名称	含义	是否采用
click	点击	√
long_click	长按	√
swipe	滑动	√
drag	拖拽	√
focus_change	修改文本框信息	√
select	选择	√
dialog_cancel	关闭弹框	√
dialog_dismiss	忽略弹框	√
item_click	点击列表	√
item_long_click	长按列表	√
item_selected	选择列表	√
open_options_menu	打开选择菜单	√
implicit_back_event	返回事件	√
implicit_home_event	Home 事件	√
implicit_launch_event	启动页事件	√
implicit_power_event	功率事件	×
implicit_rotate_event	旋转事件	×

表4-6中展示了窗口跳转信息中 Event Type 属性的分类，如 focus_change 表示进行文本框输入，item_click 代表对列表进行点击，dialog_cancel 表示关闭弹框。表4-6中存在未对控件操作的事件类型，如 implicit_rotate_event 和 implicit_power_event。由于未对控件进行操作，这两类事件类型对应的测试行为中不包含相应的控件信息。在窗口跳转信息中存在较多这两类的数据，然而它们对于检测应用程序起到的作用较弱，并且基于自动化测试构建的测试行为列表 TestAction_Auto 中存在少量这两类型的测试行为。如果将这两类作为未覆盖的测试行为，以此生成测试用例，那么在测试用例中会存在大量这两类型的

操作，但是实际对于检验应用的效果较弱。因此，在本文中我们不考虑这两类事件类型的窗口跳转信息，不构建相应的测试行为。

生成测试操作描述。现已基本完成静态分析的测试行为列表的构建，但是静态分析的测试行为列表 TestAction_Static 中缺少相应的测试操作描述 description，测试操作描述是构成测试用例步骤描述的基础。因此，下一步是为 TestAction_Static 生成测试操作描述 description。

对于表4-6中 implicit_back_event 返回事件、implicit_launch_event 启动页事件等简单的事件类型，这些可以不依托控件信息生成测试操作描述。对于具有控件操作的事件，需要根据事件类型、事件操作对象类型、控件信息生成测试操作描述。基于 TestAction_Static 的事件类型 event_type、操作对象 operation_object 和控件真实名称 widget_real_name 生成测试操作描述。如果发生了窗口跳转，则补充窗口跳转信息说明，以增强测试操作描述的可读性。如表4-4展示的窗口跳转信息所示，生成的 TestAction_Static 的属性 event_type、operation_object 和 widget_real_name 分别为：“click”、“android.view.MenuItem”和“保存”，“click”和“android.view.MenuItem”对应的含义分别是“点击”、“菜单栏”。因此，生成的测试操作描述 description 可以表示为：点击菜单栏中的“保存”按钮。由于此示例并未发生窗口跳转，因此不存在窗口跳转说明描述。

表 4-7: 应用程序“Budget Watch”的部分测试操作描述示例

窗口跳转信息的 Event 属性	部分 description 测试操作描述
MenuItemINFL[android.view.MenuItem,WID[2131689727|action_save]898,1200]3663	点击菜单栏中的“保存”按钮
INFL[android.widget.ListView,WID[2131689670|list]844,3651]3653	点击列表
INFL[android.widget.EditText,WID[2131689709|accountEdit]875,3437]3459	在标识为“账户”的文本框中输入内容（可使用正负值、小数、符号、文字等进行测试）
INFL[android.widget.Button,WID[2131689718|captureButton]912,3471]3473	点击“拍照留存”按钮

表4-7展示了在应用程序“Budget Watch”上生成的部分测试操作描述示例。根据窗口跳转信息的 Event 属性，形成 TestAction_Static 的操作对象 operation_object 和控件真实名称 widget_real_name，结合事件类型 event_type 生成对应的测试操作描述 description，表中未展示 description 中的窗口跳转说明。

4.3 模型构建

通过前面的工作我们已获得包含应用截图的测试行为列表 TestAction_Auto，以及包含控件真实名称的测试行为列表 TestAction_Static。但是 TestAction_Auto 和 TestAction_Static 对于核心属性存在数据缺失的情况，如 TestAction_Auto 的测试操作描述 description 属性中只包含控件名称，缺少控件真实名称信息，仅根据控件名称难以定位到具体的控件。此外，TestAction_Static 缺少应用截图 screen_shot 属性值。这些属性的缺失或不完整会影响测试用例的有效性。

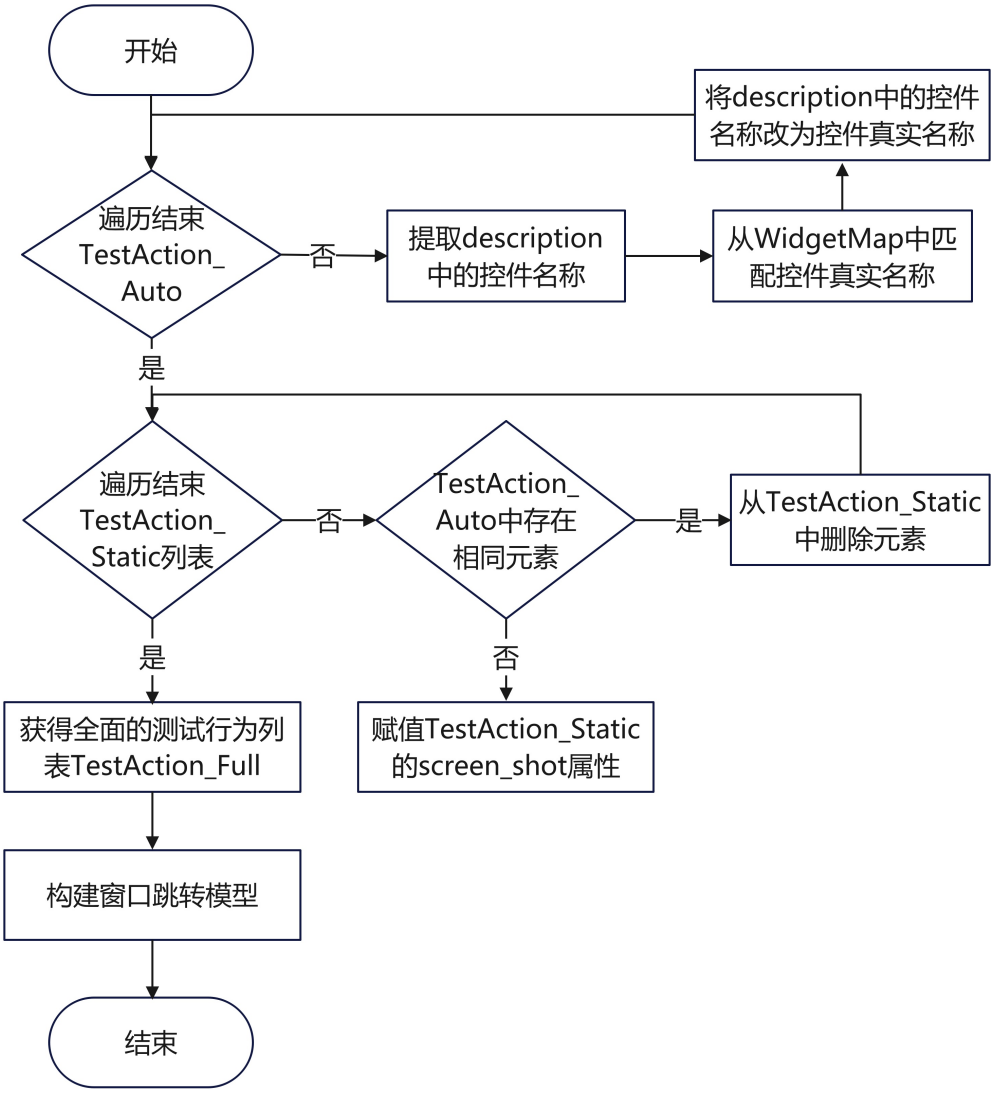


图 4-9: 模型构建模块流程图

为了构建 Android 应用的窗口跳转模型，需要完善测试行为列表 TestAction_Auto 和 TestAction_Static，然后形成全面的测试行为列表 TestAction_Full，最后基于 TestAction_Full 构建 Android 应用的窗口跳转模型。如图4-9是模型构建流程图，模型构建模块主要包括以下内容。

完善测试行为列表。在自动化测试过程中，自动化测试工具通过模拟用户交互事件和系统监听事件对应用进行 GUI 测试，未对应用源码进行分析，无法获得控件真实名称。因此 TestAction_Auto 的 description 属性中只含控件名称，缺少对应的控件真实名称。在4.2节，我们已经建立控件名称和控件真实名称的映射关系 WidgetMap。对于任一$TestAction_Auto_i$，将$TestAction_Auto_i$ 的 widget_name 从 WidgetMap 中匹配对应的 value，赋值$TestAction_Auto_i$ 的控件真实名称 widget_real_name，并修改$TestAction_Auto_i$ 的测试操作描述 description 属性，将 description 中的控件名称用控件真实名称替换。

由于静态分析工具不运行应用程序，只对源码进行分析，无法获得应用运行时的应用截图，因此 TestAction_Static 的 screen_shot 应用截图属性值缺失。自动化测试工具在运行过程中会记录应用截图，在4.1节，我们已为 TestAction_Auto 中的 screen_shot 属性添加了对应的应用截图信息。

比对 TestAction_Auto 和 TestAction_Static，将起始窗口 activity_source、目标窗口 activity_target 和控件名称 widget_name 作为唯一标识，对任一测试行为$TestAction_Static_j$，如果存在相等的$TestAction_Auto_i$，说明自动化测试和静态分析检测出相同的测试行为，将$TestAction_Static_j$ 从 TestAction_Static 中删除，只保留具有应用截图的$TestAction_Auto_i$，并将$TestAction_Auto_i$ 的 data_type 设置为 3。如果 TestAction_Auto 中不存在与$TestAction_Static_j$ 匹配的测试行为，我们获得与$TestAction_Static_j$ 的起始窗口 source_activity 属性相等的$TestAction_Auto_i$，将两者的 screen_shot 属性进行对应。由于自动化测试工具无法覆盖应用的所有测试路径，因此存在自动化测试过程中未覆盖$TestAction_Static_j$ 中的起始窗口 source_activity，则无法匹配 source_activity 的应用截图，那么$TestAction_Static_j$ 的 screen_shot 属性就是空值。

遍历结束 TestAction_Static 和 TestAction_Auto 后，TestAction_Static 表示存在于静态分析中但不存在于自动化测试中的测试行为列表。将 TestAction_Auto 和 TestAction_Static 进行合并，可获得既包含控件真实名称，又具有应用截图的测试行为列表 TestAction_Full，TestAction_Full 是应用全面的测试行为列表。

构建窗口跳转模型。有了全面的测试行为列表 TestAction_Full，接下来

基于 TestAction_Full 构建窗口跳转模型。窗口跳转模型是一种表示窗口之间跳转关系的模型，通过窗口跳转进行建模，可为获得高效的测试用例提供保证。对于任一$TestAction_Full_i$，通过$source_activity_i$和$target_activity_i$可获得窗口跳转关系，如果$source_activity_i$和$target_activity_i$不相等，说明测试行为$TestAction_Full_i$会引起应用窗口发生改变，反之说明应用窗口未发生改变，可能是对应用进行文本输入等测试行为。

本文通过有向图结构存储窗口跳转模型。我们定义 G=<A, $TestAction\_Full$>表示窗口跳转模型，其中 G 中的节点集合用 A 表示，代表 Android 应用的所有窗口。G 中的边集合用 TestAction_Full 表示，代表 Android 应用中检测出的所有测试行为。对于 TestAction_Full 中任意一条边$TestAction_Full_{ij}$，与其相连的两个节点分别为 A_i 和 A_j。在 A_i 和 A_i 两个节点中间，可能存在多条有向边$TestAction_Full_{ij}$= {$TestAction\_Full_1$, $TestAction\_Full_2$, $TestAction\_Full_3$, ...}，但集合 TestAction_Full 中每条边代表的测试行为信息都不同。此外，对于未发生窗口跳转的测试行为，在图中以一个环的形式进行展现。

图4-10是应用程序“Budget Watch”基于 TestAction_Full 构建的部分窗口跳转模型。由于窗口之间的跳转可能存在多种测试行为进行触发，并且一个窗口上也可能存在多种测试行为执行但未发生窗口跳转，为了便于展示，图4-10中从多个跳转方式中选取一条边进行表示。

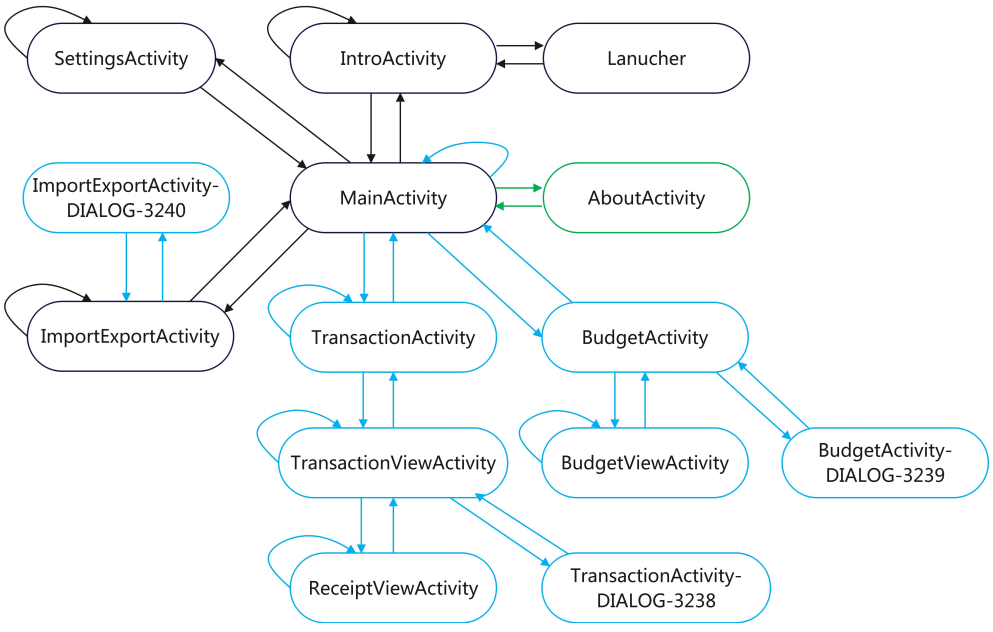


图 4-10: 应用程序“Budget Watch”的部分窗口跳转模型

图4-10中绿色边表示 TestAction_Full 中 data_type 为 1 的测试行为，蓝色边表示 TestAction_Full 中 data_type 为 2 的测试行为，黑色边表示 TestAction_Full 中 data_type 为 3 的测试行为。因此绿色节点和边分别代表通过自动化测试获得的窗口和测试行为，蓝色节点和边分别表示通过静态分析补充的窗口和测试行为，黑色节点和边分别代表自动化测试和静态分析均可获得的窗口和测试行为。相较于自动化测试，静态分析能够分析出更多的应用窗口和测试行为，使得窗口跳转模型更加丰富。

该技术构建的窗口跳转模型比自动化测试和静态分析单独构建的模型更全面。从图4-10可以看出，窗口跳转模型呈树形结构，用户可以从起始窗口出发访问应用程序的不同窗口。

如图4-11、4-12、4-13、4-14、4-15所示，通过采用此技术分别构建了“Calculator”、“我吃药了吗”、“Clear List”、“记乐部”、“2048 精简加强版”应用程序的部分窗口跳转模型。如果两窗口之间同时存在黑色和蓝色的边，则说明自动化测试和静态分析均可获得这两窗口之间的测试行为，但是静态分析存在对这两个窗口跳转的测试行为补充。

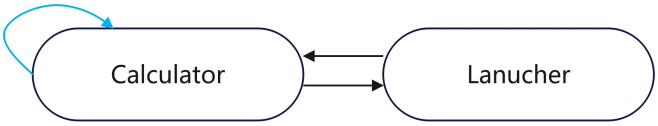


图 4-11: 应用程序“Calculator”的部分窗口跳转模型

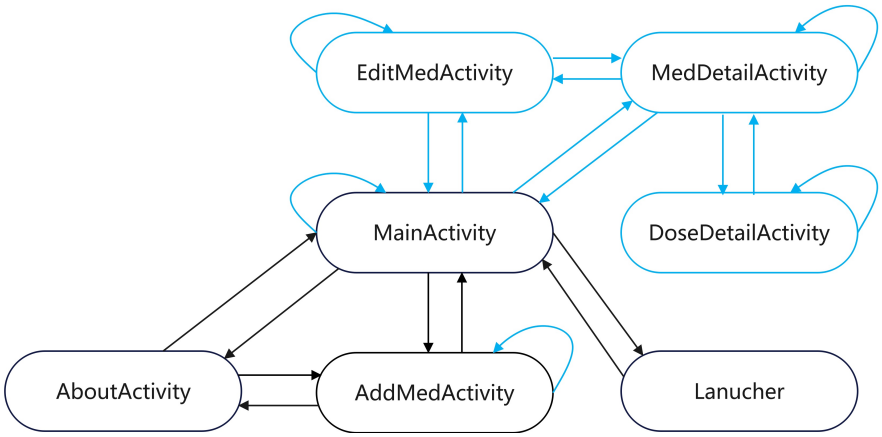


图 4-12: 应用程序“我吃药了吗”的部分窗口跳转模型

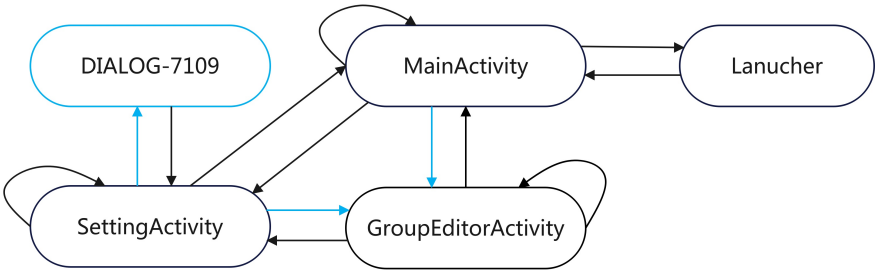


图 4-13: 应用程序“Clear List”的部分窗口跳转模型

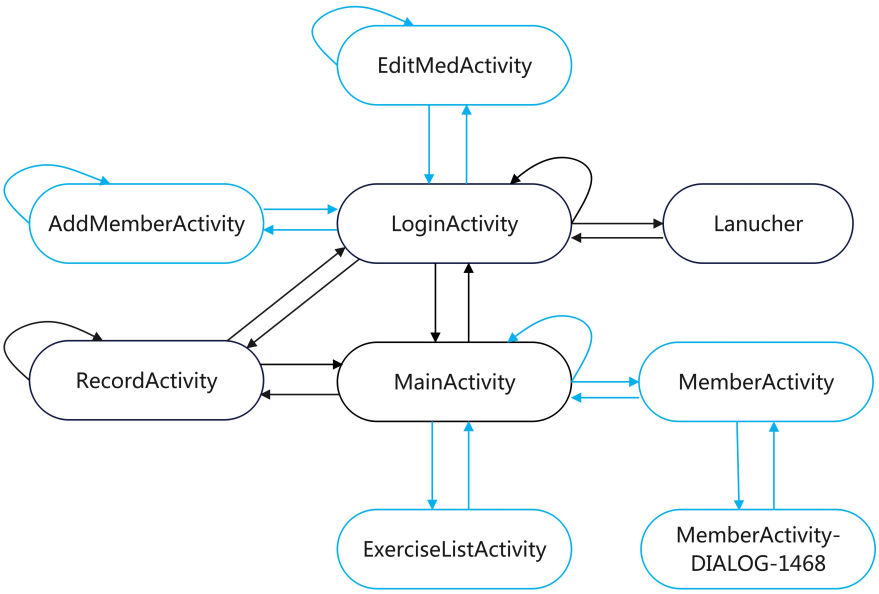


图 4-14: 应用程序“记乐部”的部分窗口跳转模型

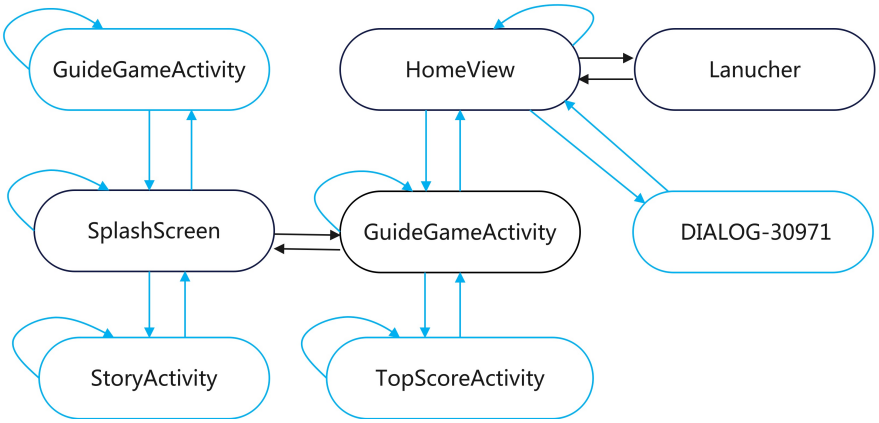


图 4-15: 应用程序“2048 精简加强版”的部分窗口跳转模型

4.4 用例生成

我们已获得 Android 应用的窗口跳转模型，基于窗口跳转模型形成测试行为序列，根据序列中测试行为的测试操作描述和应用截图信息，生成测试用例。将测试用例提供给众包工人，引导众包工人完成测试任务。

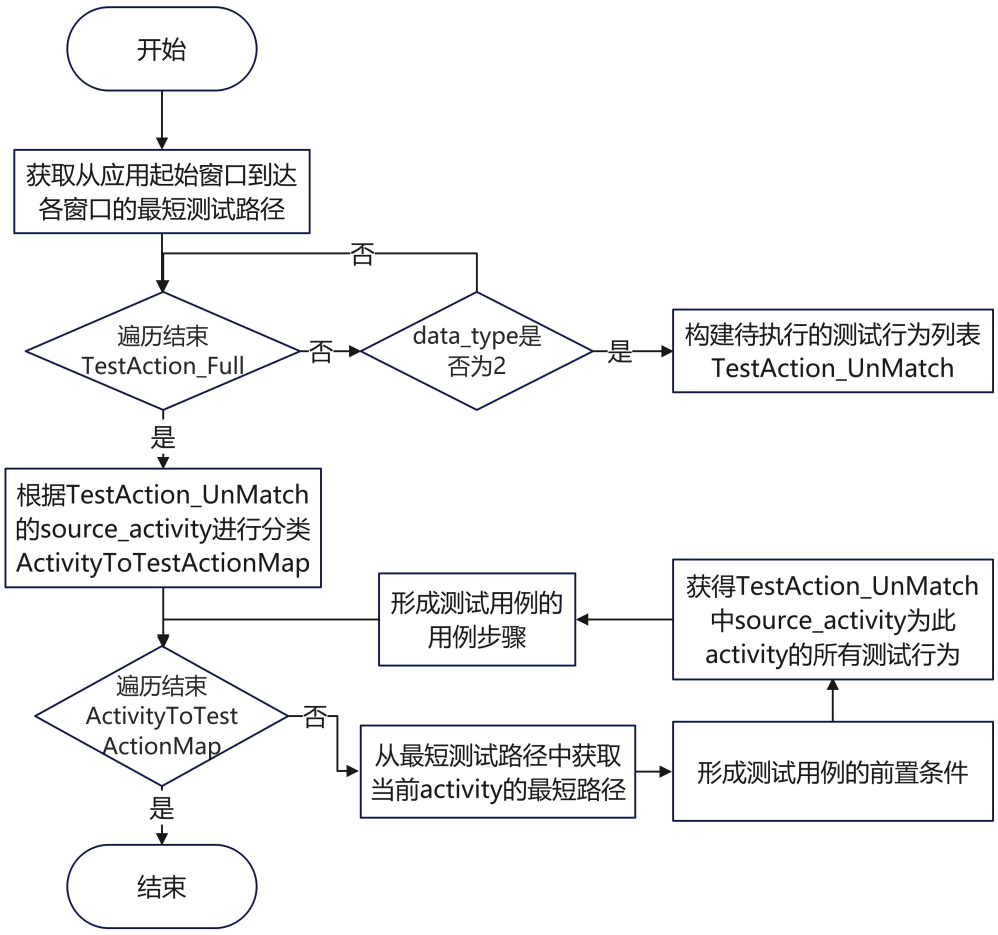


图 4-16: 用例生成模块流程图

用例生成模块处理流程如图4-16所示。通过测试路径选择，获得从应用起始窗口到达各窗口的最短测试路径，从 TestAction_Full 中获取自动化测试未覆盖的测试行为列表，构建待执行的测试行为列表 TestAction_UnMatch。根据 source_activity 属性对 TestAction_UnMatch 进行分类，获得从应用起始窗口到达 source_activity 的最短测试路径，结合 source_activity 上的所有待执行测试行为，形成测试行为序列，基于序列中测试操作描述和应用截图生成测试用例。测试用例生成模块主要包含以下内容：

测试路径选择。根据窗口跳转模型 G，采用 Dijstra 算法计算从应用的起始窗口到达各窗口的最短操作路径。Dijstra 算法适用于求解有向图中单源点到其他节点的最短路径，其主要特点是每次都选用离源点最近的、未计算出最短路径的节点用于迭代计算，不断更新选取节点到其邻接节点的路径长度。保证每次迭代获得的节点到源点的路径长度都是最短的，采用贪心的思想使得每一次迭代获得的最优解就是整体的最优解。

算法 4.1 计算最短测试路径

```
1:  G =< A, TestAction_Full >, start_activity
2:  dist[], prev[]
3:  S ← {start_activity}
4:  U ← {}
5:  dist[start_activity] ← 0, prev[start_activity] ← −1
6:  while 遍历结束 A do
7:      if a ∈ A and a ≠ start_activity then
8:          if a is adjacent to start_activity then
9:              dist[a] ← 1
10:             prev[a] ← a
11:         else
12:             dist[a] ← ∞
13:             prev[a] ← −1
14:         end if
15:         U.add(a)
16:     end if
17: end while
18: while U is not empty do
19:     u ← vertex in U with min dist[u]
20:     U.remove(u)
21:     S.add(u)
22:     for each adjacent a of u do
23:         sum ← dist[u] + 1
24:         if sum < dist[a] then
25:             dist[a] ← sum
26:             prev[a] ← u
27:         end if
28:     end for
29: end while
30: return dist, prev
```

算法 4.1所示是测试路径选择算法的伪代码，采用的是最短路径算法实现测试路径选择。其中 G 代表应用的窗口跳转模型，TestAction_Full 表示应用的全

面测试行为列表。最短测试路径计算时的源点用 start_activity 表示，在本文中使用应用的起始窗口作为源点。dist 和 prev 是两个数组，dist 表示从源点到各节点的最短测试路径距离，prev 代表从源点到各节点的最短测试路径中所需经过的中间节点。由于 G 中的节点代表应用窗口，因此任意两个节点之间的距离都是 1。S 和 U 都是节点集合，都是 A 的子集，其中 S 表示已求出最短测试路径的节点集合，U 代表未求出最短测试路径的节点集合。

首先对 S、U、dist、prev 进行初始化，初始时 S 中只包含 start_activity，U 为空集，因此 *dist*[*start_activity*] 的值为 0，*prev*[*start_activity*] 的值为-1（行 3-5）。遍历 G 的节点集合 A，如果节点 a 与源点 start_activity 相连，则 *dist*[*a*] 设置为 1，*prev*[*a*] 赋值为 a，否则 *dist*[*a*] 为 ∞，*prev*[*a*] 初始化为-1，并将节点 a 加入到集合 U 中（行 6-17）。遍历节点集合 U，将 U 集合中 dist 值最小的节点 u 取出，并加入到 S 集合中（行 18-21），对于 u 的每个邻接节点 a，如果 *dist*[*a*] 的值大于 *dist*[*u*]+1（节点之间的权重为 1），说明通过 start_activity 先到 u 再到 a 的距离比 start_activity 直接到 a 更近，更新 *dsit*[*a*] 的值为 *dist*[*u*] + 1，并且为 *prev*[*a*] 赋值为 u，表明由 start_activity 到 a 的最短测试路径需要经过中间节点 u（行 22-28）。最后该算法返回 dist 和 prev，从 prev 中可以获得从 start_activity 到达各窗口节点的最短测试路径。

测试用例生成。通过遍历窗口跳转模型完成测试路径选择，接下来基于选择的测试路径生成测试用例。我们定义测试用例 TestCase，可以表示为 *TestCase=<name, TestAction_Pre, TestAction_Next>*，name 代表测试用例的名称，TestAction_Pre 表示测试用例中前置条件的测试行为列表，TestAction_Next 记录测试用例中用例步骤的测试行为列表。TestAction_Pre 和 TestAction_Next 都是 TestAction_Full 的子集。

本文生成的测试用例是基于指定窗口上存在的所有自动化测试未覆盖的测试行为，结合最短测试路径，形成特定窗口的测试用例。因此 TestCase 的 TestAction_Pre 是指从起始窗口到达待测窗口的最短测试路径，TestAction_Next 是指在待测窗口上所有待执行的测试行为。

将窗口跳转模型中 TestAction_Full 的 data_type 字段为 2 的测试行为加入到 TestAction_UnMatch 测试行为列表中，TestAction_UnMatch 表示自动化测试过程中未覆盖的测试行为列表，存放待执行的测试行为。使用 ActivityToTestActionMap 存储 TestAction_UnMatch 根据起始窗口 source_activity 属性的分类结果，ActivityToTestActionMap 是一个键值对，其中的 key 是 source_activity

属性值，value 是测试行为列表，value 用于存储 TestAction_UnMatch 中指定 source_activity 上的所有待执行测试行为。

前边已经获得了从应用起始窗口到达各窗口的最短操作路径，遍历 ActivityToTestActionMap，对于每一个 source_activity，从最短操作路径中匹配到各 source_activity 的最短测试路径，用于构建 TestCase 中的 TestAction_Pre 以形成测试用例的前置条件。将 source_activity 作为 key 从 ActivityToTestActionMap 匹配对应的待执行测试行为列表，用于构建 TestCase 中的 TestAction_Next 以形成测试用例的用例步骤。通过提取 TestAction_Pre 和 TestAction_Next 中测试行为的测试操作 description 属性，分别生成前置条件和用例步骤的步骤描述列表，并获取 TestAction_Pre 和 TestAction_Next 中的应用截图 screen_shot 属性，分别生成前置条件和用例步骤的步骤截图列表。

为了便于众包工人对单个页面上自动化测试未覆盖的测试行为进行执行，我们将应用中单个页面上所有待执行的测试行为生成一个测试用例提供给众包工人进行探索。那么待执行测试行为的执行顺序对测试用例的步骤顺序起着关键性作用，这将会对测试过程的引导产生影响。通过分析发现，GATOR 工具输出结果中的 Event 属性中包含控件的 id 标识（如表4-4所示），GATOR 根据控件所在页面的顺序进行 id 标识。因此，我们基于测试行为中的 widget_id 属性对 TestAction_Next 进行排序，以此构建待执行测试行为列表，通过此列表中的 description 属性生成测试用例的用例步骤描述。

图4-17中展示了基于应用程序“Budget Watch”生成的一个测试用例示例。测试用例包含用例名称、前置条件和用例步骤，前置条件和用例步骤包括两部分，第一部分是步骤描述，通过分步骤展示测试过程，提供了易于理解的步骤描述，用例中的每个步骤都对应一个测试行为。对于需要操作具体控件的步骤会在步骤描述中提示控件真实名称，并且对于发生跳转的操作在步骤描述中也会进行提示。第二部分是应用截图，用于辅助步骤描述。通过前文可知，自动化测试工具对未覆盖的页面会出现应用截图缺失的情况，因此在测试用例中对于缺失页面的应用截图未进行展示。由图4-17可见，此测试用例中只包含前置条件中前两步的应用截图，自动化测试工具未探索到 TransactionActivity 和 TransactionViewActivity 页面，因此在前置条件的步骤 3 以及用例步骤 1-11 中不存在对应的应用截图。

我们通过在手机上安装“Budget Watch”应用程序，执行图4-17展示的测试用例，定位自动化测试未探索到的 TransactionActivity 和 TransactionViewAc-

图 4-17: 基于应用程序“Budget Watch”生成的测试用例示例一

tivity 两个页面，如图 4-18展示了测试用例中前置条件的步骤 3 以及用例步骤的步骤 1-11 的应用截图。

由图 4-18(a)可见，在前置步骤的步骤 3 中描述了此步骤操作具体控件的真实名称是“添加”，并且说明了控件所处位置是在“菜单栏”中，虽然此控件在应用中实际是以图标的方式存在，但通过长按图标可以显示控件真实名称，以此定位到前置步骤中步骤 3 所对应的控件。由图 4-18(b)可知，在用例步骤的步骤 1-6 中可以通过步骤描述匹配对应的文本框和控件，步骤描述中提示可以采用多种方式对输入值进行测试，以此提示众包工人对待测应用进行更充分的测试。值得一提的是，用例步骤的步骤 6 中提示了对“拍照留存”按钮进行点击，当点击此按钮并完成拍照后，页面的“拍照留存”按钮会变成“查看”和“更新”两个按钮（如图 4-18(c)所示），通过测试用例步骤很好地展示了控件的变化。

(a) 前置条件的步骤 3 (b) 用例步骤的步骤 1-6 (c) 用例步骤的步骤 7-11

图 4-18: 自动化测试工具对应用程序“Budget Watch”的部分未覆盖页面的应用截图

图 4-19: 基于应用程序“Budget Watch”生成的测试用例示例二

图4-19展示了应用程序“Budget Watch”的另一个测试用例示例。对于页面上存在多个测试行为均可跳转至同一页面，本文提出的技术可以识别此类跳转，并通过测试步骤展示。该技术提出采用分点展示具有相同效果的测试步骤，提供给众包工人进行选择，众包工人可以自行选择某一测试步骤执行，也可以对所提示的步骤分别执行，以便探索通过不同控件跳转至同一页面，是否会出现不同效果。

众包工人根据步骤描述和应用截图执行完成测试用例时，可以点击图4-17和图4-19右上角的“提交完成”按钮结束此测试用例，通过提交测试用例反馈众包工人执行情况，该技术会根据众包工人执行用例的情况为众包工人推荐其他测试用例。测试用例推荐基于测试用例已完成的次数，为众包工人推荐执行次数最少的测试用例。

4.5 本章小结

本章作为本文的核心章节，详细描述该技术的框架与设计。将动静态程序分析技术相结合，构建应用的窗口跳转模型，从模型中进行路径选择，获取从应用起始窗口到达各窗口的最短测试路径。从窗口跳转模型中获得自动化测试未覆盖的测试行为，构建待执行测试行为列表，基于最短测试路径和待执行的测试行为列表生成测试用例。测试用例中包含测试步骤描述以及应用截图，引导众包工人探索新异常，以人机协同的方式协助众包工人完成测试任务，实现低成本、高效率的众包测试。

第五章　实验设计与分析

5.1　实验目的

本文提出以下几个问题:

RQ1: 此技术能否有效提升众包测试的测试路径覆盖率？

RQ2: 此技术能否有效提高众包测试的测试质量？

RQ1 的提出是为了验证此技术在众包引导机制中的重要性，通过研究不使用此技术为测试人员提供自动化测试未覆盖路径的测试用例，分析测试人员是否可以很好地完成对待测应用的探索。并通过分析采用此技术与采用自动化测试作为引导对比测试路径覆盖率，以此研究使用静态分析的有效性。因此，为了验证 RQ1，我们根据测试人员对待测应用的测试路径覆盖率，对比传统众包测试、基于自动化测试的众包测试与基于动静态程序分析的众包测试。

RQ2 的提出是为了探索此技术对众包测试质量的提升，通过研究在使用和不使用该技术，测试人员发现的不重复异常的数量，分析此技术的众包测试质量的效果。为了回答 RQ2，我们从相同时间内测试人员发现的不重复异常数量，对比传统众包测试与基于动静态程序分析的众包测试。

5.2　实验设计

5.2.1　实验应用选择

表 5-1中展示了实验中使用的 6 个 Android 应用，覆盖了金融、工具、生活、生产力和游戏 5 个不同应用类别，其中包含 2 个商业应用和 4 个开源应用。开源应用是从 F-Droid [①]平台中选取的热门应用。通过尽可能地覆盖不同种类以及开源闭源的应用，探索此技术在不同类型应用上的实际效果，以此对 RQ1 和 RQ2 进行更加充分的研究。

[①]https://search.f-droid.org/

表 5-1: 实验使用的应用列表

应用名	版本	种类	介绍	是否开源
Budget Watch	0.21.4	金融	帮助管理个人预算	√
Calculator	5.4	工具	计算换算一体计算器	√
Clear List	1.5.6	生产力	管理和提示待办事项	√
我吃药了吗	1.5.3	生活	管理和跟踪药物服用	√
记乐部	1.0	工具	帮助记录俱乐部开销	×
2048 精简加强版	3.48	游戏	休闲益智类游戏	×

5.2.2 实验机型选择

此技术需要使用 MoocTest 自动化测试工具对待测应用进行自动化测试，表 5-2列出了自动化测试过程中使用的 20 台 Android 测试设备。选用的测试设备涵盖 10 个主流品牌，通过对市面上多个品牌的设备进行测试，以获得更加全面的自动化测试结果。

表 5-2: 自动化测试使用的设备列表

编号	品牌	型号
1	华为	HUAWEI TAG-TL00
2	华为	HUAWEI CAZ-AL10
3	华为	HUAWEI TIT-CL00
4	华为	HUAWEI CRR-CL00
5	华为	HUAWEI VNS-DL00
6	荣耀	ATH-CL00
7	三星	SM-A7000
8	小米	MI2
9	小米	MI NOTE LTE
10	小米	MI MAX
11	魅族	MX5
12	魅族	U20
13	魅族	Y685C
14	乐视	Letv X501
15	乐视	Le X620
16	乐视	Le X621
17	360 奇酷	8692-M02
18	一加	ONEPLUS A3000
19	美图	Meitu M4
20	OPPO	OPPO R7

本实验邀请了研一至研三的 15 名学生参与实验，他们具有一定的测试经验，但是对待测应用并不了解，他们所提交的测试报告具有较高的准确性。在实验过程中，对每个待测应用设置三组实验：传统众包测试、基于自动化测试的众包测试和基于动静态程序分析的众包测试。每场实验有 5 名测试人员参与，在实验过程中，需要在 Android 设备上安装待测应用进行测试，实验中为测试人员提供当前国内市场占有率较高的 5 款 Android 设备，作为实验的测试设备，实验中各机型如表 5-3所示。

表 5-3: 实验使用的设备列表

编号	品牌	型号
1	魅族	18pro
2	小米	MI11
3	小米	Redmi K30s
4	小米	Redmi K40
5	一加	ONEPLUS9 Pro

5.2.3 实验步骤

实验开展依托于慕测众测平台，慕测众测平台是由南京慕测信息科技有限公司开发的一款众包测试服务平台。任务请求者在慕测众测平台新建众测任务，填写任务的基本信息，上传待测应用的 APK，并通过选择是否进行用例生成以考虑是否采用此技术自动生成测试用例。对于使用该技术进行用例生成的任务，在任务新建后，需要耗费十分钟左右完成测试用例生成，因为该技术会调用自动化测试工具对应用程序进行测试。

当任务发布后，测试人员可以在平台中接收任务。对于采用此技术的众测任务，在测试人员接收任务后会显示测试用例列表，测试人员根据提供的测试用例对应用进行测试。测试过程中如果发现应用存在异常，可以在平台中上传测试报告。图 5-1展示任务请求者创建任务的操作截图，图 5-2展示测试人员创建报告的操作截图。

实验分三组，第一组使用传统众包测试，测试人员自行探索待测应用中的页面和测试路径；第二组采用基于自动化测试生成的测试用例，引导测试人员对待测应用进行探索；第三组采用基于动静态程序分析生成的测试用例，引导测试人员对自动化测试未覆盖的页面和测试路径进行探索。对实验使用的每个应用均创建三个测试任务，分别对应设置的三个实验。实验过程中，三组实验

图 5-1: 慕测众测平台任务请求者创建任务截图

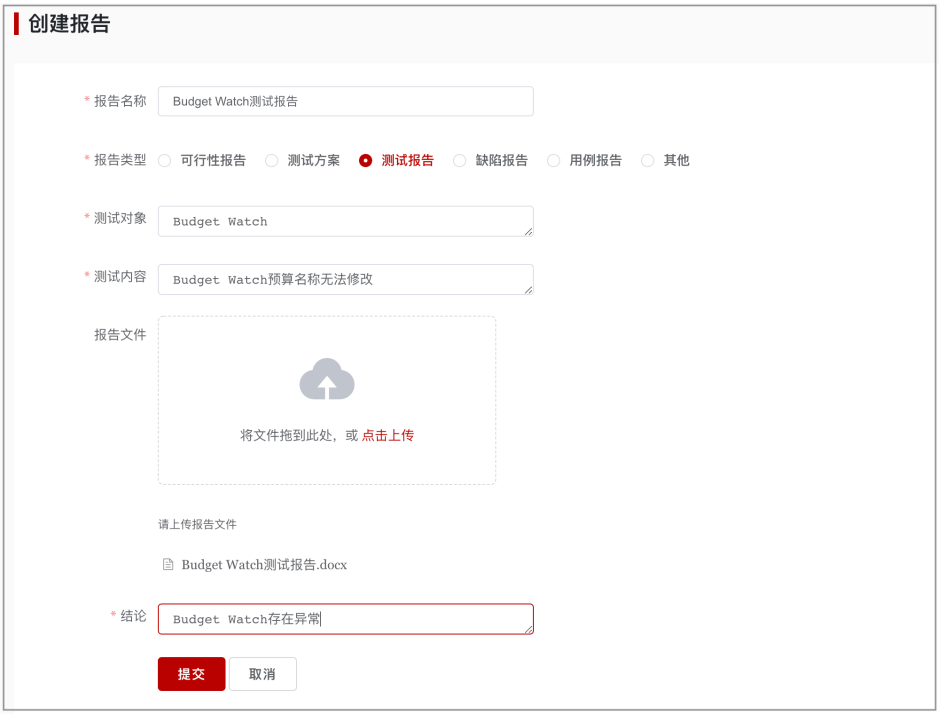


图 5-2: 慕测众测平台测试人员创建报告截图

分批进行，每组实验的测试人员同时在线执行测试任务，每人对单个测试任务的测试时长限定在 20 分钟以内，测试人员可以自由选择任务的执行顺序。

表 5-4: 实验收集的常见异常类型

类型	代表异常	说明
输入意图异常	java.lang.NullPointerException java.lang.IllegalArgumentException java.lang.ArrayIndexOutOfBoundsException java.lang.ClassNotFoundException java.lang.ClassCastException	通过输入产生的异常
资源加载异常	android.content.pm.PackageManager NameNotFoundException java.lang.FileNotFoundException java.lang.FileNotFoundException java.lang.IllegalAccessException android.system.ErrnoException java.lang.NoSuchMethodException	由于资源数据导致的异常
其他异常	KeyDisPatchTimeout BroadcastTimeout ServiceTimeout java.io.IOException ExecutionException NetworkError	例如 ANR 异常、线程间异常、网络通信异常等
用户体验异常		应用在页面布局上出现异常，如页面内容显示过小或过大、页面内容分布不合理等
功能异常		应用在功能上出现异常，如某页面访问失败或者结果不符合预期等

为了回答 RQ2，我们通过 Logcat 工具获取设备日志，Logcat 工具可以获得设备报错时的堆栈轨迹，用于转储系统消息日志，每个日志都包含一个优先级，我们使用 Logcat 工具收集应用程序在测试过程中产生的 ERROR 和 FATAL 级别日志信息，并从中分析测试所触发的异常。此外，我们对测试人员在测试报告中反馈的异常进行汇总，这些异常属于用户可感知异常。实验中收集的常见异常类型如表5-4所示，前三种是 Android 应用程序常见的异常分类，通过 Logcat 工具可以捕获，后两种是从测试报告中获得的用户可感知异常。

5.3 实验结果预处理

5.3.1 异常数据统计

每场实验开始前，我们会在测试设备上重新安装待测应用，并将测试设备连接电脑。在实验过程中，通过 Logcat 工具捕获应用运行时出现的异常，结合测试人员在慕测众测平台上提交的测试报告，收集测试人员在待测应用上所发现的异常。对发现的异常进行去重，统计实验中待测应用程序的不重复异常平均数据如表 5-5所示。

表 5-5: 不重复异常平均统计数据

实验类型	Budget Watch	Calculator	Clear List	我吃药了吗	记乐部	2048 精简加强版
实验 |C|	16.6	6.2	10.4	16.8	16.4	12.2
实验 |D|	19.2	7.2	11.8	18	17.6	12.6
实验 |T|	24.2	8.8	16.4	23.4	22.4	15.6

表中 |C| 代表传统众包测试，|D| 代表只包含自动化测试的众包测试，实验 |D| 中具有基于自动化测试生成的测试用例，对于自动化测试未覆盖的测试路径仍采用传统众包测试的模式，|T| 则代表该技术下的众包测试。由表可知，从六款应用程序中均可以获得较多不重复异常。其中，在三组实验中，测试人员所发现的不重复异常数量在应用程序“Calculator”上相差不大，但是在应用程序“Budget Watch”中差异比较明显。

5.3.2 异常数据分析

为了更好地分析实验结果，我们根据表 5-4中定义的异常类型，获得在不同实验中待测应用程序的异常分布情况，如表 5-6展示了实验 |C|、实验 |D|、实验 |T| 中，测试人员在待测应用上发现不同异常的平均数量。

表 5-6展示在实验 |C|、实验 |D|、实验 |T| 中，测试人员在六个待测应用上所发现异常的平均数量，按照“输入意图异常”、“资源加载异常”、“其他异常”、“用户体验异常”、“功能异常”类型对异常进行分类。结果显示，除了应用程序“Calculator”不存在“输入意图异常”和“资源加载异常”，以及应用程序“记乐部”不存在“输入意图异常”，每个待测应用中基本都发现了不同类型的异常。

表 5-6: 实验异常分布情况

	输入意图异常	资源加载异常	其他异常	用户体验异常	功能异常
Budget Watch|C|	3.6	3.6	5.4	1.4	2.6
Budget Watch|D|	4	4.2	6.2	1.6	3.2
Budget Watch|T|	6	5.2	7	2.4	3.6
Calculator|C|	0	0	2.8	2	1.4
Calculator|D|	0	0	2.8	2.4	2
Calculator|T|	0	0	3.6	3.2	2
Clear List|C|	2.4	1.4	3.2	1	2.4
Clear List|D|	2.8	1.6	3.6	1.4	2.4
Clear List|T|	4.2	2.2	4.6	1.8	3.6
我吃药了吗 |C|	2.4	1.8	1.6	3	8
我吃药了吗 |D|	3	2.2	1.6	2.6	8.6
我吃药了吗 |T|	4.2	2.6	2.4	4	10.2
记乐部 |C|	2	0	1.2	2.2	11
记乐部 |D|	1.6	0	1.2	2.6	12.2
记乐部 |T|	3.2	0	1.6	3.2	14.4
2048 精简加强版 |C|	1	4	1.8	3	2.4
2048 精简加强版 |D|	1.4	3.2	2.2	3	2.8
2048 精简加强版 |T|	1.6	4.6	2.6	3.6	3.2

为了更直观地对比不同异常在不同实验中的分布情况，我们通过折线图的方式展示待测应用在实验中的异常类型分布情况。待测应用在实验中的异常类型分布如图5-3所示，图中也对自动化测试工具产生的异常按照异常分类进行统计，将自动化测试工具发现的异常与三组实验发现的异常进行对比。

通过图5-3展示的折线变化趋势，我们可以发现以下两种现象：

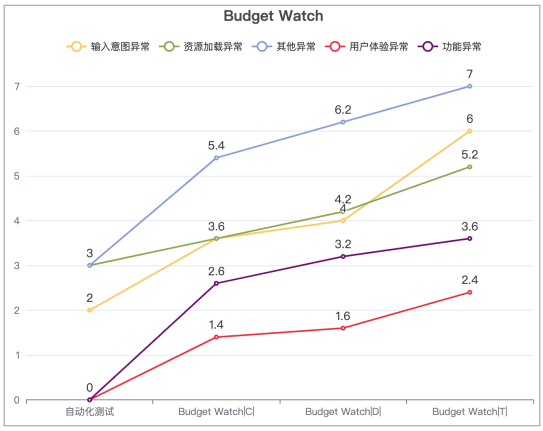
现象 1：输入意图异常在自动化测试工具中出现次数较少，但在众包测试实验中出现次数较多。

现象 2：用户体验异常和功能异常在自动化测试工具中未出现，但在众包测试实验中出现次数较多。

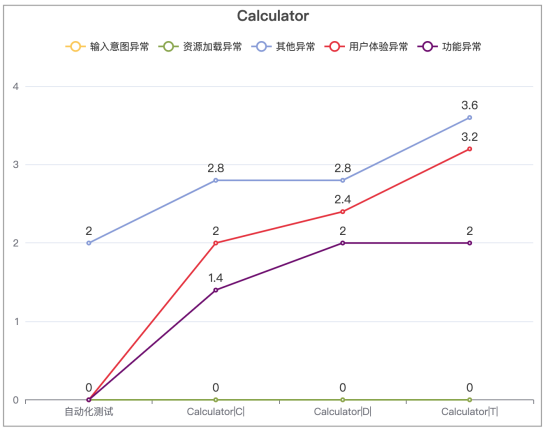
我们通过对自动化测试工具的测试过程进行分析，并对测试人员的测试过程进行了解，解释了这两个变化趋势产生的原因。

（1）现象 1 产生的原因：MoocTest 自动化测试工具支持提供人工测试脚本，允许在测试过程中主动寻找人工测试脚本中的输入参数，辅助执行输入事件。但是在人工脚本中，输入参数设置有限。然而在众包测试实验中，测试人员会对输入内容进行丰富地测试，使得输入内容形式多样，以此探索待测应用

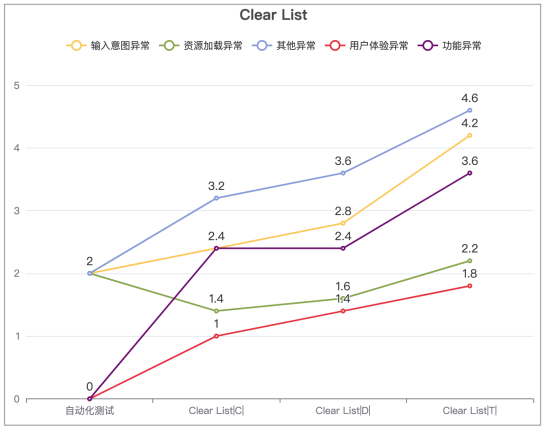
的运行情况，正是这些复杂多样的输入使得应用产生更多的输入意图异常。



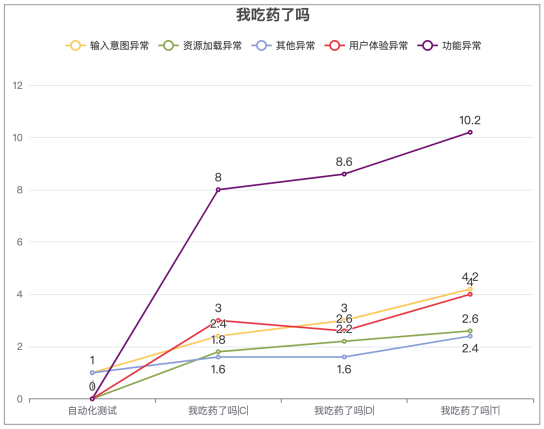
(a) “Budget Watch”在实验中的异常类型分布



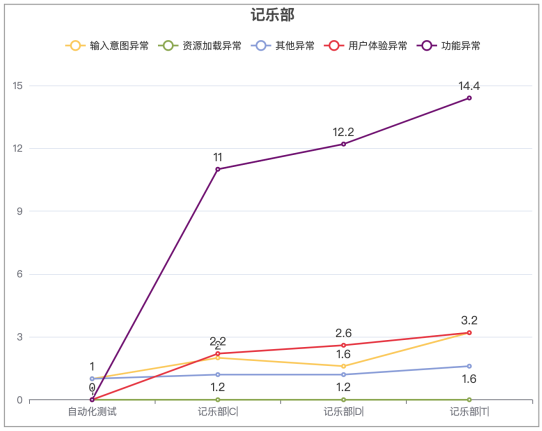
(b) “Calculator”在实验中的异常类型分布



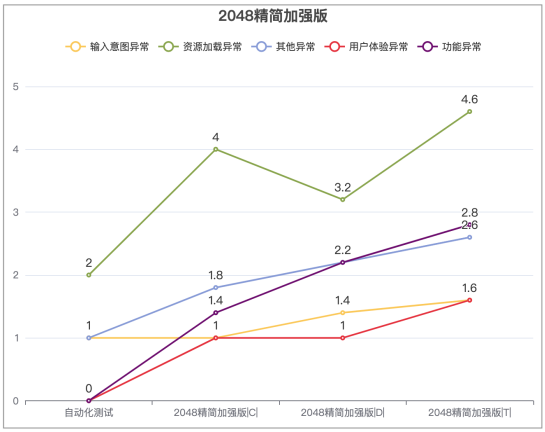
(c) “Clear List”在实验中的异常类型分布



(d) “我吃药了吗”在实验中的异常类型分布



(e) “记乐部”在实验中的异常类型分布



(f) “2048 精简加强版”在实验中的异常类型分布

图 5-3: 待测应用在实验中的异常类型分布图

（2）现象 2 产生的原因：MoocTest 自动化测试工具通过遍历应用组件探

索应用程序状态，记录应用程序的堆栈信息，检测是否触发应用程序异常，但是 MoocTest 自动化测试工具无法识别应用的页面布局是否合理，未能获得用户体验异常。此外，MoocTest 自动化测试工具不能校验应用的功能是否存在异常，无法获得应用的功能异常。然而，在众包测试实验中依托测试人员的参与，可以通过测试人员观察应用的页面布局是否合理，判断应用的功能是否正常，以此反馈应用的用户体验异常和功能异常。

通过以上分析，我们可以发现，无论是传统的众包测试还是包含测试用例引导的众包测试，使用众包测试都可以得到全面的测试结果。因此，我们可以得出：众包测试相较于自动化测试可以获得更加全面的测试结果，众包测试是一种有效的移动应用测试方式。

5.4 实验结果分析

5.4.1 测试路径覆盖率

实验一是为了验证此技术的有效性，我们通过收集测试人员在实验过程中对待测应用的操作，进而分析测试人员在测试过程中对待测应用的测试路径覆盖。为了回答 RQ1，我们统计了自动化测试工具对待测应用的窗口覆盖和测试路径覆盖的情况，如表5-7所示。其中 $Cover_{activity}$ 表示自动化测试过程中覆盖的窗口数量，$All_{activity}$ 表示在实验中检测出的所有窗口数量，$CoverActivity_{path}$ 表示自动化测试过程中对已覆盖窗口中已执行的测试路径数量，$AllCoverActivity_{path}$ 表示已覆盖窗口中检测出的所有测试路径数量。

表 5-7: 自动化测试工具覆盖情况

应用名	$Cover_{activity}/All_{activity}$	$CoverActivity_{path}/AllCoverActivity_{path}$
Budget Watch	0.5	0.9
Calculator	1	0.4
Clear List	0.6	0.7
我吃药了吗	0.5	0.5
记乐部	0.6	0.8
2048 精简加强版	0.6	0.8

由表5-7可知自动化测试工具对待测应用的窗口覆盖率无法达到百分之百，有些窗口会被自动化测试工具遗漏，导致测试不充分。即使是对于已覆盖

的窗口，自动化测试工具也无法覆盖所有测试路径。由此可见，对自动化测试过程中未覆盖的测试路径进行探索是非常有必要的。

此技术通过将自动化测试未覆盖的测试路径，生成待测应用的测试用例，测试用例中包含测试步骤说明，通过执行测试用例中的测试步骤，以此提升测试过程中测试路径覆盖率。表5-8展示了此技术为实验 |T| 测试人员生成的待测应用用例数量和测试用例中包含的待测路径数量。由表5-8可知，六款应用的平均待测路径数量为 24.3，存在较多自动化测试未覆盖的测试路径。

表 5-8: 应用的测试用例数量和测试用例中包含的待测路径数量

应用名	测试用例数量	待测路径数量
Budget Watch	8	27
Calculator	1	26
Clear List	3	16
我吃药了吗	5	22
记乐部	6	11
2048 精简加强版	18	44

我们通过收集三组实验过程中测试人员对待测应用的测试路径覆盖情况，分别获得三组实验的测试路径覆盖数量为 P|C|、P|D|、P|T|，对 P|C|、P|D|、P|T| 去重后获得应用的测试路径总数量 P|S|。不同实验的测试路径覆盖情况如表5-9所示。

表 5-9: 不同实验的测试路径覆盖情况

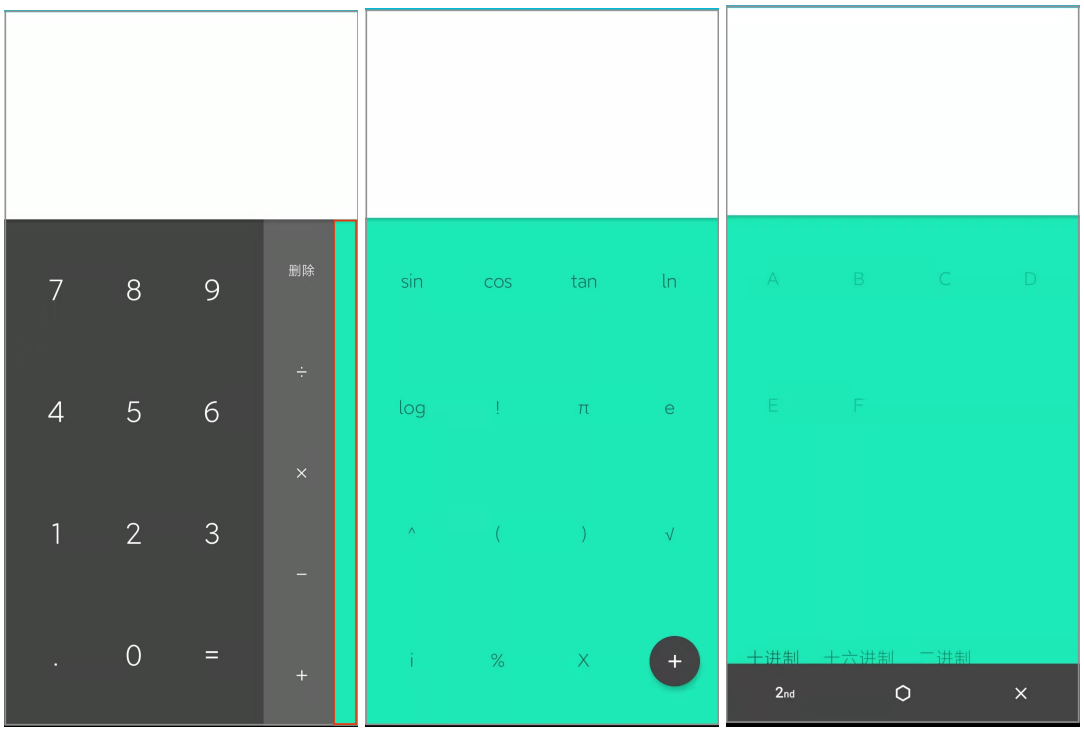
应用名	P|C|/P|S|	P|D|/P|S|	P|T|/P|S|	(P|T|-P|C|)/P|S|	(P|T|-P|D|)/P|S|
Budget Watch	32.8/45	37.6/45	42/45	20.4%	9.8%
Calculator	36.6/58	38/58	55.6/58	32.8%	30.3%
Clear List	21.6/31	23.8/31	26.8/31	16.8%	9.7%
我吃药了吗	31.4/42	35.6/42	38.6/42	17.1%	7.1%
记乐部	33.6/48	37.8/48	44.4/48	22.5%	13.8%
2048 精简加强版	68/97	74.6/97	88.8/97	21.4%	13.6%

我们统计和分析实验中的测试路径覆盖情况，实验结果显示，本文提出的技术相较于传统的众包测试平均提升了 21.8% 的测试路径覆盖率，相较于基于自动化测试的众包测试平均提升了 14.2% 的测试路径覆盖率。

通过比较不同应用之间的测试路径覆盖率，我们发现实验 |T| 中“Calculator”应用的测试路径覆盖率，相较于实验 |C| 和实验 |D| 都获得了显著提升。我

们通过与测试人员进行沟通，分析产生原因:

“Calculator”应用是一款计算器应用，除了 Laucher 页只存在一个 Activity，所有的操作都在一个页面上进行，如图 5-4所示是“Calculator”应用的应用截图。通过与这几组实验的测试人员沟通之后发现，“Calculator”应用页面虽然简单，但是存在隐藏布局，需要通过滑动绿色框（如图 5-4(a)中右侧）才可以显示隐藏布局。如图 5-4(b)和图 5-4(c)展示了“Calculator”应用的隐藏布局，隐藏布局中存在大量可操作的控件。



(a) “Calculator”应用首页 (b) “Calculator”应用隐藏布局 (c) “Calculator”应用隐藏布局

图 5-4: “Calculator”应用的应用截图

此技术对“Calculator”应用程序生成的测试用例示例如图 5-5所示。虽然用例步骤的步骤描述中未说明需要滑动绿色框才可显示应用的隐藏布局，但提示了对隐藏布局中控件操作的测试步骤。通过与测试人员进行沟通可知，实验 |T| 的测试人员一致认为用例步骤提示的控件肯定存在应用中，于是会对应用进行不断探索，直到执行完成测试用例中的测试步骤。然而在自动化测试过程中，没有检测到隐藏布局的控件，因此在实验 |D| 的测试用例中缺少相应的测试步骤描述。实验 |D| 和实验 |C| 中由于缺乏对隐藏布局中控件操作的提示，大部分测试人员在 5-4(a)中操作结束后，认为已经完成了测试任务，便不再对应用进行探索，因此忽略了应用存在的隐藏布局，遗漏了许多测试路径。

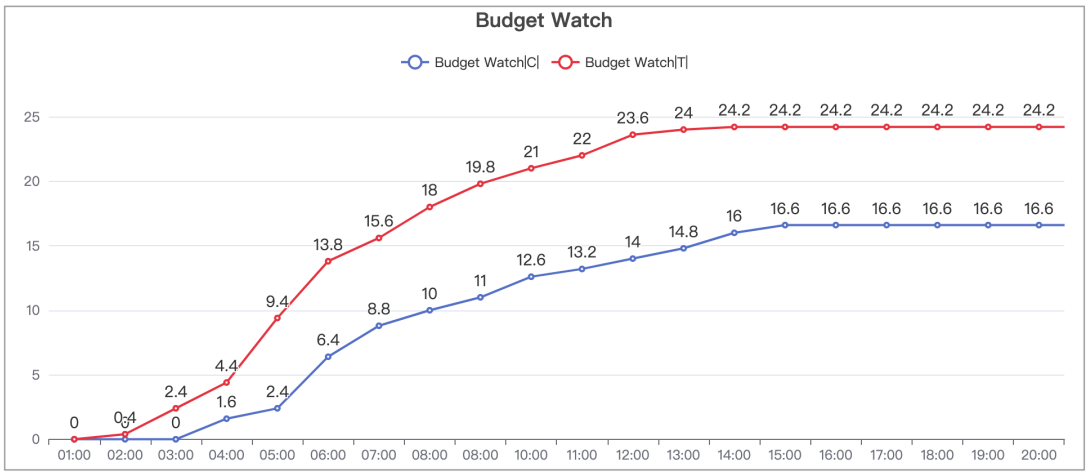
(a) “Calculator”应用的测试用例前半部分 (b) “Calculator”应用的测试用例后半部分

图 5-5: “Calculator”应用的测试用例

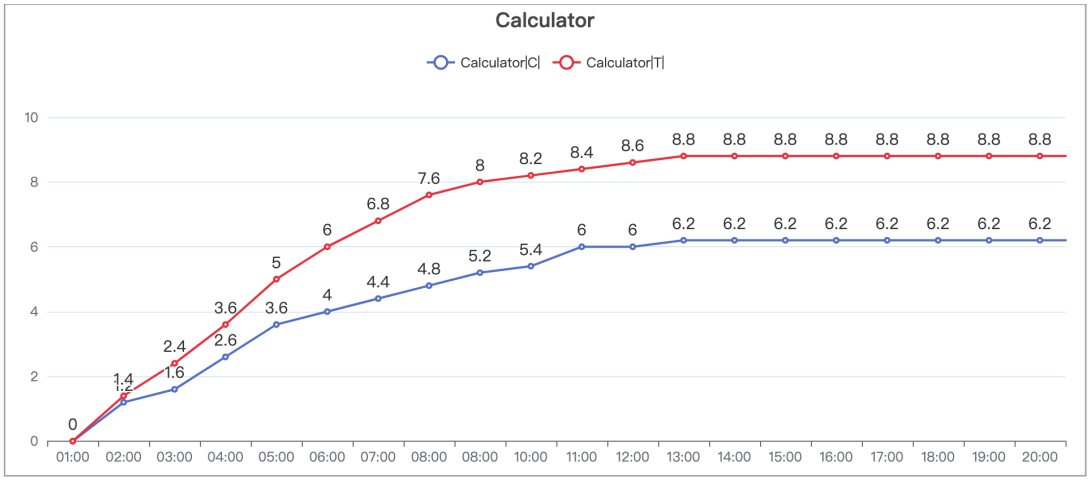
基于以上对测试路径覆盖率的分析，我们可以得出使用本文提出的测试用例生成技术，结合动静态程序分析获取应用较为全面的测试路径，并使用易于理解的步骤描述和清晰直观的应用截图，向测试人员展示自动化测试未覆盖路径的操作，可以激发测试人员在测试过程中对应用的探索，减少测试人员对应用控件的遗漏，提高对待测应用的测试路径覆盖率。此外，通过实验我们可以发现本文提出的技术相较于传统的众包测试平均提升了 21.8% 的测试路径覆盖率，相较于自动化测试的众包测试平均提升了 14.2% 的测试路径覆盖率，由此可见此技术中加入静态分析是有效的。

5.4.2 测试质量评估

为了研究此技术是否能够提高众包测试的质量，我们限定实验中对单个应用的测试时长不超过 20 分钟。在实验过程中，收集 Logcat 工具捕获的应用日志信息，对日志等级为 ERROR 和 FATAL 的异常进行统计，并且汇总测试人员在慕测众测平台上提交的测试报告，对于测试报告中提出的异常进行统计。我们将 Logcat 工具收集的异常和测试人员在测试报告中展示的异常统称为测试人员发现的异常，通过对相同时间内在应用上发现的异常进行统计，异常数越多则代表该实验的众包测试质量越高。



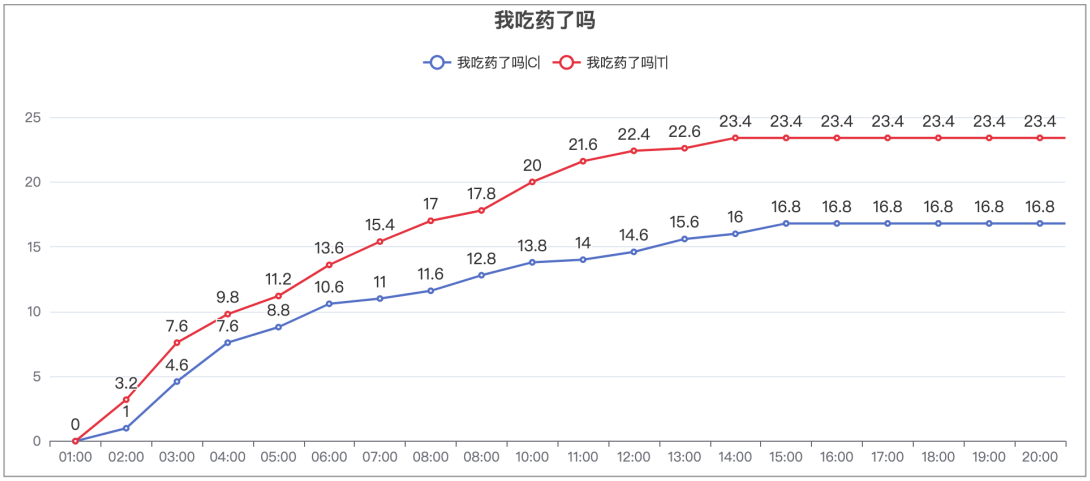
(a) “Budget Watch”应用的异常数量随时间变化图



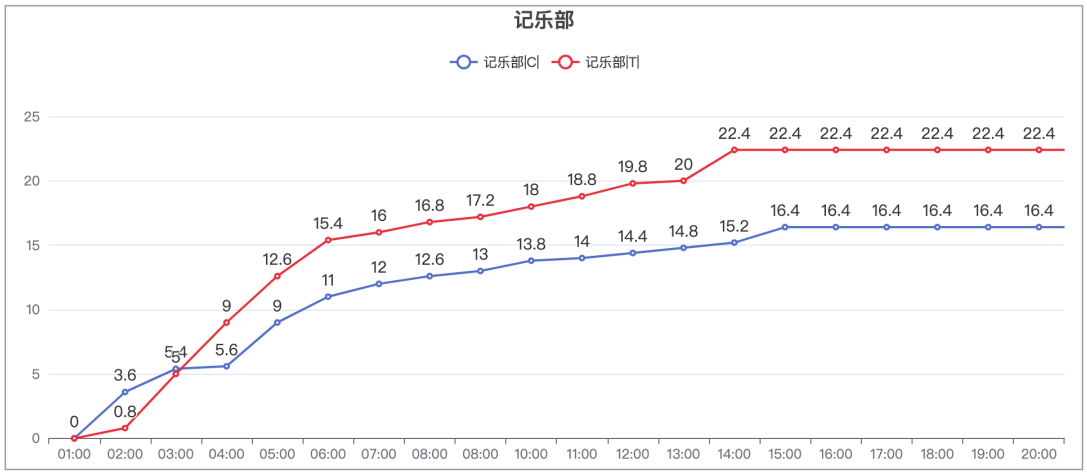
(b) “Calculator”应用的异常数量随时间变化图

(c) “Clear List”应用的异常数量随时间变化图

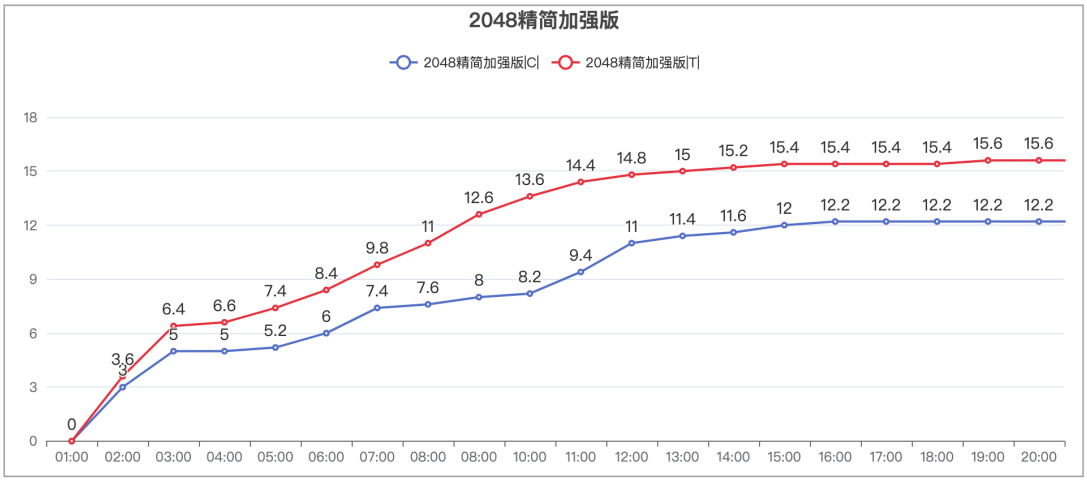
图 5-6: 前三款应用的异常数量随时间变化图



(a) “我吃药了吗”应用的异常数量随时间变化图



(b) “记乐部”应用的异常数量随时间变化图



(c) “2048 精简加强版”应用的异常数量随时间变化图

图 5-7: 后三款应用的异常数量随时间变化图

在实验过程中，我们每隔一分钟统计测试人员在实验 |C| 和实验 |T| 中发现的不重复异常的平均数量，如图 5-6和图 5-7所是六个应用的异常数量随时间变化的情况。从整体趋势可知，基于动静态程序分析的众包测试实验 |T| 比传统的众包测试实验 |C| 基本能够更快、更多地发现应用异常，并且实验 |T| 基本更快结束测试任务。此外，我们可以发现以下两种现象：

现象 1：图 5-6(b)所示应用程序“记乐部”的异常变化折线图中，实验 |C| 发现的异常数量在前三分钟高于实验 |T|。

现象 2：图 5-7(c)所示应用程序“2048 精简加强版”的异常变化折线图中，实验 |T| 测试花费的时长高于实验 |C|。

我们通过调查实验过程中测试人员的测试场景，分析这两种现象产生的原因。

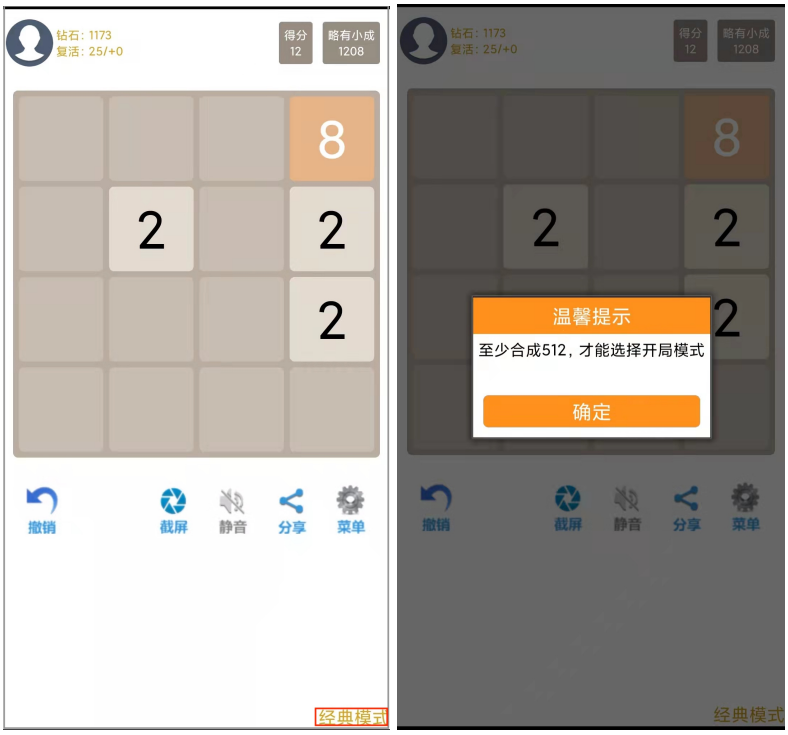
图 5-8: “记乐部”应用的部分测试用例

（1）现象 1 产生的原因：这是由于“记乐部”应用的第一个页面是登录页面，如图 5-8所示是实验 |T| 中该技术生成的部分测试用例，测试用例中提示了登录页面中“会员名”和“密码”的输入值，实验 |T| 中的测试人员可以很快完成登录，进入应用的主页面。然而在实验 |C| 中，由于没有提示“会员名”和“密码”的输入值，测试人员会对输入内容进行测试，多次尝试仍然登录失败后，会点击“注册”按钮跳转到注册页面，注册页面如图 5-9所示，测试人员会在注册页面进行输入值的测试。在实验 |C| 中测试人员对登录页面和注册页面的测试输入操作触发了应用程序的异常，导致前两分钟异常数量突增。实验 |T| 的测试人员虽然根据测试用例成功完成登录操作，但在后续也会对登录页面和注册页面的输入值进行边界值测试。

图 5-9: “记乐部”应用的注册页面

（2）现象 2 产生的原因：应用程序“2048 精简加强版”是一款益智游戏，在游戏过程中，如图 5-10(a)所示存在“经典模式”按钮，点击此按钮会弹框显示如图 5-10(b)的内容，提示测试人员需要在游戏过程中实现合成 512 才可选择开局模式。通过了解可知，达到合成 512 的目标需要很长时间，因此实验 |T| 和实验 |C| 的测试人员都选择先完成其他的测试后，最后再进行合成 512 的探索。由图 5-7(c)可知，实验 |T| 在 13 分钟后曲线趋于一条直线，根据测试人员的测试场景发现，此时实验 |T| 的测试人员基本完成其他模块的测试，正在进

行游戏以便合成 512。然而，实验 |C| 的测试人员在 15 分钟后才开始对合成 512 进行探索。由此可见，不考虑后期对测试合成 512 的探索，实际上实验 |T| 比实验 |C| 更早完成对其他模块的测试。



(a) “2048 精简加强版”应用截图一 (b) “2048 精简加强版”应用截图二

图 5-10: “2048 精简加强版”应用截图

通过对实验数据进行统计，分别获得实验 |C| 和实验 |T| 中发现的不重复异常数量为 E|C|、E|T|，对 E|C|、E|T| 去重后获得应用的异常总数量 E|S|。不同实验的异常比例情况如表 5-10所示。

表 5-10: 不同实验的异常发现情况

应用名	E|C|/E|S|	E|T|/E|S|	(E|T|-E|C|)/E|S|
Budget Watch	16.6/28.2	24.2/28.2	27.0%
Calculator	6.2/10.8	8.8/10.8	24.1%
Clear List	10.4/19.4	16.4/19.4	30.9%
我吃药了吗	16.8/27.4	23.4/27.4	24.1%
记乐部	16.4/26.4	22.4/26.4	22.7%
2048 精简加强版	12.2/17.6	15.6/17.6	19.3%

通过表 5-10我们发现，在对不同应用的实验中，实验 |T| 比实验 |C| 均发现更多异常。通过计算可得实验 |T| 比实验 |C| 发现的异常数量提升了 24.7%。

由此可见，相较于传统的众包测试，此技术不仅可以更快发现异常，而且基本更快完成测试任务，在异常发现数量上比传统的众包测试提升了 24.7%。综上所述，此技术能够提高众包测试的质量。

5.5 有效性威胁

本节我们讨论可能影响实验结果的因素：

（1）GATOR 工具的覆盖率。研究表明，由于 GATOR 工具对控制流分析时未考虑到隐含的意图，对应用程序的路径覆盖率只能达到 87.5%。虽然 GATOR 工具对应用的测试路径无法实现全覆盖，但是在实验过程中，通过测试人员对应用进行探索，能够发现 GATOR 未检测出的测试路径，这可以减轻这种威胁。

（2）测试用例的前置步骤。实验中的测试用例是基于起始页面到其他页面的最短测试路径生成的，测试用例中的前置步骤均是从应用起始页面开始执行测试操作，测试人员重复执行前序步骤可能会影响实验效果。然而，在实验过程中，测试人员通过执行测试用例可掌握应用的使用流程，可以在不重复执行前置步骤的情况下，完成对测试用例的执行。

5.6 本章小结

本章我们提出了两个研究问题，通过三个实验对研究问题进行探讨，分别采用传统众包测试、基于自动化测试的众包测试以及使用该技术的众包测试开展实验。我们通过分析测试人员测试过程中对应用的测试路径覆盖情况，以此验证该技术是否可以提高测试路径覆盖率。结果表明该技术相较于传统众包测试可以提升 21.8% 的测试路径覆盖率，相较于基于自动化测试的众包测试提升了 14.2% 的测试路径覆盖率，由此可见在此技术中采用静态分析的有效性，并且能够得出此技术可以有效提高众包测试的测试路径覆盖。此外，收集测试人员在测试过程中提交的测试报告以及设备执行情况，我们通过在相同时间内发现异常的数量，基于测试效率验证该技术是否可以提升传统众包测试的质量。对于实验结果的分析，我们发现使用此技术相较于传统的众包测试可以提高 24.7% 的异常发现数量，并且基本都更快完成测试任务，由此可见本文提出的技术可以提高传统众包测试的质量。

第六章　总结与展望

6.1　总结

移动应用已然成为人们生活中重要组成部分，人们在享受移动应用带来的新奇世界之时，也对应用的质量提出了更高的要求。Android 系统碎片化问题以及频繁的迭代周期使得传统的手工测试难堪重负，更多开发和测试人员采用 Android 自动化测试和众包测试解决此问题。然而，Android 自动化测试对应用的路径无法实现全覆盖，并且未能检测应用页面布局以及应用功能问题。传统的众包测试存在众包测试人员专业性不足等问题，需要为众包测试人员生成测试用例。然而，人工生成测试用例需要掌握待测应用的领域知识，并且往往会耗费大量的人力和时间。因此，如何为众包测试人员快速生成测试用例，优化众包测试机制，提升众包测试质量，成为亟待解决的问题。

本文创新性地提出一种测试用例生成技术。将动静态程序分析相结合，自动为众包测试人员生成测试用例，辅助众包测试人员高效完成测试。使用 MoocTest 自动化测试工具对应用进行自动化测试，获取测试结果中的操作事件和应用截图，构建包含应用截图的测试行为列表，并生成测试步骤描述。在静态分析中，采用两款静态分析工具对应用进行分析，构建包含控件真实名称的测试行为列表，并生成测试步骤描述。基于动静态分析获取的测试行为列表构建应用的窗口跳转模型，获得自动化测试过程中未覆盖的测试行为。采用 Dijkstra 算法进行测试路径选择，选择模型中从应用起始窗口到自动化未覆盖测试行为的最短测试行为路径，基于测试行为中的测试操作描述形成测试用例的步骤描述，结合应用截图信息辅助步骤说明，以此形成测试用例。生成的测试用例以易于理解的文本描述以及清晰明了的应用截图，向众包测试人员提供测试引导。我们在六个应用程序上进行实验，结果表明该技术相较于传统众包测试和基于自动化测试的众包测试，分别可以提升 21.8% 和 14.2% 的测试路径覆盖率。此外，结果表明使用此技术相较于传统的众包测试可以增加 24.7% 的异常发现数量，并且基本都更快完成测试任务。分别从路径覆盖率和测试质量两个方面，证明该技术能够有效地提升众包测试的质量。

6.2 展望

本文提出的技术未来可以朝着以下几点进行改进:

（1）该技术实现了对 Android 应用的众包测试用例生成，然而 iOS 应用也占据了较大的移动应用市场，后续可以考虑实现对 iOS 应用的众包测试用例生成，以便支持更大的移动应用市场。

（2）该技术根据测试用例已执行次数为测试人员推荐测试用例，没有实时关注众包测试人员执行完成测试用例后所处页面。为了能够更加准确地进行测试用例推荐，可以考虑基于测试人员所在页面，实时推荐最邻近页面的测试用例。

（3）该技术基于起始页面到其他页面的最短测试路径形成测试用例，测试用例中的前置步骤均是从应用起始页面开始执行测试操作。为了避免测试人员重复执行前序步骤，可以考虑基于测试人员所处页面作为起始页面，生成测试用例。

（4）该技术生成的测试用例中，可以提示控件的真实名称。然而有些控件在应用中是以图标的形式展示。为了生成更加有效的测试用例，可以识别应用的图标控件，在测试用例中提示控件的图标，辅助测试人员更加高效地完成测试。

致　　谢

论文即将撰写完毕，突然意识到我的研究生时代即将落幕，这不由地有丝感慨和惆怅。在此之前时常憧憬毕业后的生活，可当要毕业时，又有太多不舍和感伤。我向所有关心、爱护和帮助过我的人表示最诚挚的感谢。

首先，我要感谢我的指导老师陈振宇老师，陈振宇老师治学严谨，思路开阔，为人幽默风趣，在研究生阶段给予了我非常多锻炼的机会。在论文开题阶段和论文写作阶段，陈振宇老师都提出了很多宝贵的建议，让我能够及时调整和修改。其次我要感谢黄勇老师，黄勇老师时常为我解答技术难题，让我在专业技能上获得很大提升。我还要感谢何铁科老师和房春荣老师在学术论文上对我的指导，让我能够顺利发表学术论文。此外，感谢李玉莹学姐在毕业设计和毕业论文上对我提供的帮助，一起讨论毕业设计实现方案，并且耐心指导我修改毕业论文。

其次，感谢南京大学，虽然研究生阶段大部分时间都处于疫情笼罩之下，但是正因为有南京大学正确的方针政策，这才守护了全校师生的安全。在此也特别感谢南京大学软件学院，感谢学院全体教职工的尽心尽责。

此外，由衷地感谢 iSE 实验室的每一位同学，感谢你们对我生活和学习上的帮助，让我度过了美好的研究生时光。特别感谢孙加辉、曹可凡、张朱佩田同学，感谢你们和我一起学习、一起玩耍的日子。感谢王瑾师妹在工作上的支持，感谢贺璐师妹在运动上的相伴，感谢葛修婷师姐在专业上的帮助。也要感谢我的室友储学远，感谢陪伴我度过了研究生最后一年。

最后，感谢我的父母和家人，感谢父母对我的谆谆教诲，教会我如何为人处世。感谢我的女朋友，她的鼓励和支持也给予了我很大的动力。

参考文献

[1] WEI L, LIU Y, CHEUNG S. Taming Android fragmentation: characterizing and detecting compatibility issues for Android apps[C/OL] // LO D, APEL S, KHURSHID S. Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering, ASE 2016, Singapore, September 3-7, 2016. [S.l.]: ACM, 2016: 226 – 237.
https://doi.org/10.1145/2970276.2970312.

[2] ANAND S, BURKE E K, CHEN T Y, et al. An orchestrated survey of methodologies for automated software test case generation[J/OL]. J. Syst. Softw., 2013, 86(8): 1978 – 2001.
https://doi.org/10.1016/j.jss.2013.02.061.

[3] LI L. Mining AndroZoo: A Retrospect[C/OL] // 2017 IEEE International Conference on Software Maintenance and Evolution, ICSME 2017, Shanghai, China, September 17-22, 2017. [S.l.]: IEEE Computer Society, 2017: 675 – 680.
https://doi.org/10.1109/ICSME.2017.49.

[4] WANG H, LI H, LI L, et al. Why are Android apps removed from Google Play?: a large-scale empirical study[C/OL] // ZAIDMAN A, KAMEI Y, HILL E. Proceedings of the 15th International Conference on Mining Software Repositories, MSR 2018, Gothenburg, Sweden, May 28-29, 2018. [S.l.]: ACM, 2018: 231 – 242.
https://doi.org/10.1145/3196398.3196412.

[5] KOCHHAR P S, THUNG F, NAGAPPAN N, et al. Understanding the Test Automation Culture of App Developers[C/OL] // 8th IEEE International Conference on Software Testing, Verification and Validation, ICST 2015, Graz, Austria, April 13-17, 2015. [S.l.]: IEEE Computer Society, 2015: 1 – 10.
https://doi.org/10.1109/ICST.2015.7102609.

[6] PACKARD H. Failing to meet mobile app user expectations: a mobile user survey[J]. Tech. rep., 2015.

[7] CHEN N, KIM S. Puzzle-based automatic testing: bringing humans into the loop by solving puzzles[C/OL] // GOEDICKE M, MENZIES T, SAEKI M. IEEE/ACM International Conference on Automated Software Engineering, ASE'12, Essen, Germany, September 3-7, 2012. [S.l.]: ACM, 2012: 140 – 149. https://doi.org/10.1145/2351676.2351697.

[8] JOORABCHI M E, MESBAH A, KRUCHTEN P. Real Challenges in Mobile App Development[C/OL] // 2013 ACM / IEEE International Symposium on Empirical Software Engineering and Measurement, Baltimore, Maryland, USA, October 10-11, 2013. [S.l.]: IEEE Computer Society, 2013: 15 – 24. https://doi.org/10.1109/ESEM.2013.9.

[9] YU S, TAKADA S. Mobile application test case generation focusing on external events[C/OL] // FLYNN L, SILLITTI A. Proceedings of the 1st International Workshop on Mobile Development, Mobile!@SPLASH 2016, Amsterdam, Netherlands, October 31, 2016. [S.l.]: ACM, 2016: 41 – 42. https://doi.org/10.1145/3001854.3001864.

[10] RUBINOV K, BARESI L. What Are We Missing When Testing Our Android Apps?[J/OL]. Computer, 2018, 51(4): 60 – 68. https://doi.org/10.1109/MC.2018.2141024.

[11] USMAN M, IQBAL M Z, KHAN M U. An automated model-based approach for unit-level performance test generation of mobile applications[J/OL]. J. Softw. Evol. Process., 2020, 32(1). https://doi.org/10.1002/smr.2215.

[12] BAEK Y M, BAE D. Automated model-based Android GUI testing using multi-level GUI comparison criteria[C/OL] // LO D, APEL S, KHURSHID S. Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering, ASE 2016, Singapore, September 3-7, 2016. [S.l.]: ACM, 2016: 238 – 249. https://doi.org/10.1145/2970276.2970313.

[13] HAO S, LIU B, NATH S, et al. PUMA: programmable UI-automation for large-scale dynamic analysis of mobile apps[C/OL] // CAMPBELL A T, KOTZ D, COX L P, et al. The 12th Annual International Conference on Mobile Systems, Applications, and Services, MobiSys'14, Bretton Woods, NH, USA, June 16-19, 2014. [S.l.] : ACM, 2014 : 204 – 217.
https://doi.org/10.1145/2594368.2594390.

[14] ALZAYLAEE M K, YERIMA S Y, SEZER S. DL-Droid: Deep learning based android malware detection using real devices[J/OL]. Comput. Secur., 2020, 89.
https://doi.org/10.1016/j.cose.2019.101663.

[15] MORAN K, VÁSQUEZ M L, BERNAL-CÁRDENAS C, et al. Automatically Discovering, Reporting and Reproducing Android Application Crashes[C/OL] // 2016 IEEE International Conference on Software Testing, Verification and Validation, ICST 2016, Chicago, IL, USA, April 11-15, 2016. [S.l.] : IEEE Computer Society, 2016 : 33 – 44.
https://doi.org/10.1109/ICST.2016.34.

[16] DASHEVSKYI S, GADYATSKAYA O, PILGUN A, et al. The Influence of Code Coverage Metrics on Automated Testing Efficiency in Android[C/OL] // LIE D, MANNAN M, BACKES M, et al. Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018. [S.l.] : ACM, 2018 : 2216 – 2218.
https://doi.org/10.1145/3243734.3278524.

[17] CHOUDHARY S R, GORLA A, ORSO A. Automated Test Input Generation for Android: Are We There Yet? (E)[C/OL] // COHEN M B, GRUNSKE L, WHALEN M. 30th IEEE/ACM International Conference on Automated Software Engineering, ASE 2015, Lincoln, NE, USA, November 9-13, 2015. [S.l.] : IEEE Computer Society, 2015 : 429 – 440.
https://doi.org/10.1109/ASE.2015.89.

[18] HOWE J, OTHERS. The rise of crowdsourcing[J]. Wired magazine, 2006, 14(6) : 1 – 4.

[19] STERLING B. The hacker crackdown: Law and disorder on the electronic frontier[M]. [S.l.]: Open Road Media, 2020.

[20] GARDLO B, EGGER S, SEUFERT M, et al. Crowdsourcing 2.0: Enhancing execution speed and reliability of web-based QoE testing[C/OL] // IEEE International Conference on Communications, ICC 2014, Sydney, Australia, June 10-14, 2014. [S.l.]: IEEE, 2014: 1070 – 1075.
https://doi.org/10.1109/ICC.2014.6883463.

[21] HOSSFELD T, KEIMEL C, TIMMERER C. Crowdsourcing Quality-of-Experience Assessments[J/OL]. Computer, 2014, 47(9): 98 – 102.
https://doi.org/10.1109/MC.2014.245.

[22] GOMIDE V H, VALLE P A, FERREIRA J O, et al. Affective crowdsourcing applied to usability testing[J]. International Journal of Computer Scienceand Information Technologies, 2014, 5(1): 575 – 579.

[23] KHAN A I, AL-KHANJARI Z, SARRAB M. Crowd sourced testing through end users for Mobile Learning application in the context of Bring Your Own Device[C] // 2016 IEEE 7th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON). 2016: 1 – 6.

[24] MALONE D, DUNNE J. Social dogfood: A framework to minimise clouc field defects through crowd sourced testing[C] // 2017 28th Irish Signals and Systems Conference (ISSC). 2017: 1 – 6.

[25] WU G, CAO Y, CHEN W, et al. AppCheck: A Crowdsourced Testing Service for Android Applications[C/OL] // ALTINTAS I, CHEN S. 2017 IEEE International Conference on Web Services, ICWS 2017, Honolulu, HI, USA, June 25-30, 2017. [S.l.]: IEEE, 2017: 253 – 260.
https://doi.org/10.1109/ICWS.2017.40.

[26] MUSSON R, RICHARDS J, FISHER D, et al. Leveraging the Crowd: How 48, 000 Users Helped Improve Lync Performance[J/OL]. IEEE Softw., 2013, 30(4): 38 – 45.
https://doi.org/10.1109/MS.2013.67.

[27] ZHANG T, GAO J Z, CHENG J. Crowdsourced Testing Services for Mobile Apps[C/OL] // 2017 IEEE Symposium on Service-Oriented System Engineering, SOSE 2017, San Francisco, CA, USA, April 6-9, 2017. [S.l.] : IEEE Computer Society, 2017 : 75 – 80.
https://doi.org/10.1109/SOSE.2017.28.

[28] LATOZA T D, van der HOEK A. Crowdsourcing in Software Engineering: Models, Motivations, and Challenges[J/OL]. IEEE Softw., 2016, 33(1) : 74 – 80.
https://doi.org/10.1109/MS.2016.12.

[29] MACHIRY A, TAHILIANI R, NAIK M. Dynodroid: an input generation system for Android apps[C/OL] // MEYER B, BARESI L, MEZINI M. Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering, ESEC/FSE'13, Saint Petersburg, Russian Federation, August 18-26, 2013. [S.l.] : ACM, 2013 : 224 – 234.
https://doi.org/10.1145/2491411.2491450.

[30] VÁSQUEZ M L, WHITE M, BERNAL-CÁRDENAS C, et al. Mining Android App Usages for Generating Actionable GUI-Based Execution Scenarios[C/OL] // PENTA M D, PINZGER M, ROBBES R. 12th IEEE/ACM Working Conference on Mining Software Repositories, MSR 2015, Florence, Italy, May 16-17, 2015. [S.l.] : IEEE Computer Society, 2015 : 111 – 122.
https://doi.org/10.1109/MSR.2015.18.

[31] AMALFITANO D, FASOLINO A R, TRAMONTANA P, et al. Using GUI ripping for automated testing of Android applications[C/OL] // GOEDICKE M, MENZIES T, SAEKI M. IEEE/ACM International Conference on Automated Software Engineering, ASE'12, Essen, Germany, September 3-7, 2012. [S.l.] : ACM, 2012 : 258 – 261.
https://doi.org/10.1145/2351676.2351717.

[32] AMALFITANO D, FASOLINO A R, TRAMONTANA P, et al. MobiGUITAR: Automated Model-Based Testing of Mobile Apps[J/OL]. IEEE Softw., 2015, 32(5) : 53 – 59.
https://doi.org/10.1109/MS.2014.55.

[33] YANG W, PRASAD M R, XIE T. A Grey-Box Approach for Automated GUI-Model Generation of Mobile Applications[C/OL] // CORTELLESSA V, VARRÓ D. Lecture Notes in Computer Science, Vol 7793 : Fundamental Approaches to Software Engineering - 16th International Conference, FASE 2013, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2013, Rome, Italy, March 16-24, 2013. Proceedings. [S.l.] : Springer, 2013 : 250 – 265.
https://doi.org/10.1007/978-3-642-37057-1_19.

[34] AZIM T, NEAMTIU I. Targeted and depth-first exploration for systematic testing of android apps[C/OL] // HOSKING A L, EUGSTER P T, LOPES C V. 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. [S.l.] : ACM, 2013 : 641 – 660.
https://doi.org/10.1145/2509136.2509549.

[35] MAHMOOD R, MIRZAEI N, MALEK S. EvoDroid: segmented evolutionary testing of Android apps[C/OL] // CHEUNG S, ORSO A, STOREY M D. Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering, (FSE-22), Hong Kong, China, November 16 - 22, 2014. [S.l.] : ACM, 2014 : 599 – 609.
https://doi.org/10.1145/2635868.2635896.

[36] MAO K, HARMAN M, JIA Y. Sapienz: multi-objective automated testing for Android applications[C/OL] // ZELLER A, ROYCHOUDHURY A. Proceedings of the 25th International Symposium on Software Testing and Analysis, ISSTA 2016, Saarbrücken, Germany, July 18-20, 2016. [S.l.] : ACM, 2016 : 94 – 105.
https://doi.org/10.1145/2931037.2931054.

[37] van der MERWE H, van der MERWE B, VISSER W. Execution and property specifications for JPF-android[J/OL]. ACM SIGSOFT Softw. Eng. Notes, 2014, 39(1) : 1 – 5.
https://doi.org/10.1145/2557833.2560576.

[38] LI L, BISSYANDÉ T F, PAPADAKIS M, et al. Static analysis of android apps: A systematic literature review[J/OL]. Inf. Softw. Technol., 2017, 88 : 67 – 95. https://doi.org/10.1016/j.infsof.2017.04.001.

[39] YANG S, YAN D, WU H, et al. Static Control-Flow Analysis of User-Driven Callbacks in Android Applications[C/OL] // BERTOLINO A, CANFORA G, ELBAUM S G. 37th IEEE/ACM International Conference on Software Engineering, ICSE 2015, Florence, Italy, May 16-24, 2015, Volume 1. [S.l.] : IEEE Computer Society, 2015 : 89 – 99. https://doi.org/10.1109/ICSE.2015.31.

[40] YANG S, WU H, ZHANG H, et al. Static window transition graphs for Android[J/OL]. Autom. Softw. Eng., 2018, 25(4) : 833 – 873. https://doi.org/10.1007/s10515-018-0237-6.

[41] ARZT S, RASTHOFER S, FRITZ C, et al. FlowDroid: precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for Android apps[C/OL] // O'BOYLE M F P, PINGALI K. ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI '14, Edinburgh, United Kingdom - June 09 - 11, 2014. [S.l.] : ACM, 2014 : 259 – 269. https://doi.org/10.1145/2594291.2594299.

[42] LI L, BARTEL A, BISSYANDÉ T F, et al. IccTA: Detecting Inter-Component Privacy Leaks in Android Apps[C/OL] // BERTOLINO A, CANFORA G, ELBAUM S G. 37th IEEE/ACM International Conference on Software Engineering, ICSE 2015, Florence, Italy, May 16-24, 2015, Volume 1. [S.l.] : IEEE Computer Society, 2015 : 280 – 291. https://doi.org/10.1109/ICSE.2015.48.

[43] WANG Y, ZHANG H, ROUNTEV A. On the unsoundness of static analysis for Android GUIs[C/OL] // ZHANG C, RIVAL X. Proceedings of the 5th ACM SIGPLAN International Workshop on State Of the Art in Program Analysis, SOAP@PLDI 2016, Santa Barbara, CA, USA, June 14, 2016. [S.l.] : ACM, 2016 : 18 – 23. https://doi.org/10.1145/2931021.2931026.

[44] 章晓芳, 冯洋, 刘峘, et al. 众包软件测试技术研究进展 [J]. 软件学报, 2018, 29(1): 69–88.

[45] CHEN K, WU C, CHANG Y, et al. A crowdsourceable QoE evaluation framework for multimedia content[C/OL] // GAO W, RUI Y, HANJALIC A, et al. Proceedings of the 17th International Conference on Multimedia 2009, Vancouver, British Columbia, Canada, October 19-24, 2009. [S.l.]: ACM, 2009: 491–500. https://doi.org/10.1145/1631272.1631339.

[46] KOMAROV S, REINECKE K, GAJOS K Z. Crowdsourcing performance evaluations of user interfaces[C/OL] // MACKAY W E, BREWSTER S A, BØDKER S. 2013 ACM SIGCHI Conference on Human Factors in Computing Systems, CHI '13, Paris, France, April 27 - May 2, 2013. [S.l.]: ACM, 2013: 207–216. https://doi.org/10.1145/2470654.2470684.

[47] LIU D, BIAS R G, LEASE M, et al. Crowdsourcing for usability testing[C/OL] // Proc. Assoc. Inf. Sci. Technol., Vol 49: Information, Interaction, Innovation: Celebrating the Past, Constructing the Present and Creating the Future - Proceedings of the 75th ASIS&T Annual Meeting, ASIST 2012, Baltimore, MD, USA, October 26-30, 2012. [S.l.]: Wiley, 2012: 1–10. https://doi.org/10.1002/meet.14504901100.

[48] WANG J, WANG S, CUI Q, et al. Local-based active classification of test report to assist crowdsourced testing[C/OL] // LO D, APEL S, KHURSHID S. Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering, ASE 2016, Singapore, September 3-7, 2016. [S.l.]: ACM, 2016: 190–201. https://doi.org/10.1145/2970276.2970300.

[49] LIU D, ZHANG X, FENG Y, et al. Generating descriptions for screenshots to assist crowdsourced testing[C/OL] // OLIVETO R, PENTA M D, SHEPHERD D C. 25th International Conference on Software Analysis, Evolution and Reengineering, SANER 2018, Campobasso, Italy, March 20-23, 2018. [S.l.]: IEEE Computer Society, 2018: 492–496. https://doi.org/10.1109/SANER.2018.8330246.

[50] HAO R, FENG Y, JONES J A, et al. CTRAS: crowdsourced test report aggregation and summarization[C/OL] // ATLEE J M, BULTAN T, WHITTLE J. Proceedings of the 41st International Conference on Software Engineering, ICSE 2019, Montreal, QC, Canada, May 25-31, 2019. [S.l.] : IEEE / ACM, 2019 : 900 – 910.
https://doi.org/10.1109/ICSE.2019.00096.

[51] CHEN X, JIANG H, CHEN Z, et al. Automatic test report augmentation to assist crowdsourced testing[J/OL]. Frontiers Comput. Sci., 2019, 13(5) : 943 – 959.
https://doi.org/10.1007/s11704-018-7308-5.

[52] ZHU H, HALL P A V, MAY J H R. Software Unit Test Coverage and Adequacy[J/OL]. ACM Comput. Surv., 1997, 29(4) : 366 – 427.
https://doi.org/10.1145/267580.267590.

[53] PEZZÈ M, YOUNG M. Software testing and analysis - process, principles and techniques[M]. [S.l.] : Wiley, 2007.

[54] BERTOLINO A. Software Testing Research: Achievements, Challenges, Dreams[C/OL] // BRIAND L C, WOLF A L. International Conference on Software Engineering, ISCE 2007, Workshop on the Future of Software Engineering, FOSE 2007, May 23-25, 2007, Minneapolis, MN, USA. [S.l.] : IEEE Computer Society, 2007 : 85 – 103.
https://doi.org/10.1109/FOSE.2007.25.

[55] MEMON A M. GUI Testing: Pitfalls and Process[J/OL]. Computer, 2002, 35(8) : 87 – 88.
https://doi.org/10.1109/MC.2002.1023795.

[56] KING J C. A new approach to program testing[C/OL] // SHOOMAN M L, YEH R T. Proceedings of the International Conference on Reliable Software 1975, Los Angeles, California, USA, April 21-23, 1975. [S.l.] : ACM, 1975 : 228 – 233.
https://doi.org/10.1145/800027.808444.

[57] CADAR C, DUNBAR D, ENGLER D R. KLEE: Unassisted and Automatic Generation of High-Coverage Tests for Complex Systems Programs[C/OL]

// DRAVES R, van RENESSE R. 8th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2008, December 8-10, 2008, San Diego, California, USA, Proceedings. [S.l.] : USENIX Association, 2008 : 209 – 224.
http://www.usenix.org/events/osdi08/tech/full_papers/cadar/cadar.pdf.

[58] NÖTZLI A, KHAN J, FINGERHUT A, et al. p4pktgen: Automated Test Case Generation for P4 Programs[C/OL] // Proceedings of the Symposium on SDN Research, SOSR 2018, Los Angeles, CA, USA, March 28-29, 2018. [S.l.] : ACM, 2018 : 5:1 – 5:7.
https://doi.org/10.1145/3185467.3185497.

[59] MIGUEL J L S, TAKADA S. GUI and usage model-based test case generation for Android applications with change analysis[C/OL] // FLYNN L, SILLITTI A. Proceedings of the 1st International Workshop on Mobile Development, Mobile!@SPLASH 2016, Amsterdam, Netherlands, October 31, 2016. [S.l.] : ACM, 2016 : 43 – 44.
https://doi.org/10.1145/3001854.3001865.

[60] CHOI W, NECULA G C, SEN K. Guided GUI testing of android apps with minimal restart and approximate learning[C/OL] // HOSKING A L, EUGSTER P T, LOPES C V. 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. [S.l.] : ACM, 2013 : 623 – 640.
https://doi.org/10.1145/2509136.2509552.

[61] FRASER G, ARCURI A. EvoSuite: On the Challenges of Test Case Generation in the Real World[C/OL] // Sixth IEEE International Conference on Software Testing, Verification and Validation, ICST 2013, Luxembourg, Luxembourg, March 18-22, 2013. [S.l.] : IEEE Computer Society, 2013 : 362 – 369.
https://doi.org/10.1109/ICST.2013.51.

[62] YASIN H N, HAMID S H A, YUSOF R J R. DroidbotX: Test Case Generation Tool for Android Applications Using Q-Learning[J/OL]. Symmetry, 2021, 13(2) :

310.
https://doi.org/10.3390/sym13020310.

[63] ARCURI A. RESTful API Automated Test Case Generation with EvoMaster[J/OL]. ACM Trans. Softw. Eng. Methodol., 2019, 28(1): 3:1 – 3:37.
https://doi.org/10.1145/3293455.

[64] GAY G, STAATS M, WHALEN M W, et al. The Risks of Coverage-Directed Test Case Generation[J/OL]. IEEE Trans. Software Eng., 2015, 41(8): 803 – 819.
https://doi.org/10.1109/TSE.2015.2421011.

[65] HAOYIN L. Automatic android application GUI testing—A random walk approach[C] // 2017 International Conference on Wireless Communications, Signal Processing and Networking (WiSPNET). 2017: 72 – 76.

简历与科研成果

基本信息

郭超，男，汉族，1997 年 7 月出生，江西省赣州市人。

教育背景

2019 年 9 月 — 2022 年 6 月 南京大学软件学院 硕士

2015 年 9 月 — 2019 年 6 月 江西财经大学软件与互联网工程学院 本科

攻读硕士学位期间完成的学术成果

1. **Guo C**, He T, Yuan W, et al, “Crowdsourced requirements generation for automatic testing via knowledge graph,” in *Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis*, July. 2020.
2. 黄万民, 陈振宇, 张晋桂, **郭超**, 杨鹏, 郑海涛, 林怡坤,“测试用例生成方法、装置、存储介质和电子设备”, 专利申请号:202210029446.1,2022.

攻读硕士学位期间参与的科研课题

1. 国家重点研发计划：信息产品及科技服务集成化众测服务平台研发与应用 (2018YFB1403400)（课题年限 2019 — 2022），负责众测服务平台的研发。