计算机研究与发展
Journal of Computer Research and Development
DOI:10.7544/issn1000-1239.20210105
59(6):1370-1387,2022

# 安卓移动应用兼容性测试综述

郑 炜[1,5,6] 唐 辉[1] 陈 翔[2,3] 张满青[1] 夏 鑫[4]
[1](西北工业大学软件学院 西安 710072)
[2](南通大学信息科学技术学院 江苏南通 226019)
[3](信息安全国家重点实验室(中国科学院信息工程研究所) 北京 100093)
[4](蒙纳士大学信息技术学院 澳大利亚墨尔本 3800)
[5](空天地海一体化大数据应用技术国家工程实验室(西北工业大学) 西安 710072)
[6](大数据存储与管理工业和信息化部重点实验室(西北工业大学) 西安 710072)
(wzheng@nwpu.edu.cn)

# State-of-the-Art Survey of Compatibility Test for Android Mobile Application

Zheng Wei[1,5,6], Tang Hui[1], Chen Xiang[2,3], Zhang Manqing[1], and Xia Xin[4]
[1](*School of Software*, *Northwestern Polytechnical University*, *Xi'an* 710072)
[2](*School of Information Science and Technology*, *Nantong University*, *Nantong*, *Jiangsu* 226019)
[3](*State Key Laboratory of Information Security* (*Institute of Information Engineering*, *Chinese Academy of Sciences*), *Beijing* 100093)
[4](*Faculty of Information Technology*, *Monash University*, *Melbourne*, *Australia* 3800)
[5](*National Engineering Laboratory for Integrated Aero-Space-Ground-Ocean Big Data Application Technology* (*Northwestern Polytechnical University*), *Xi'an* 710072)
[6](*Key Laboratory of Big Data Storage and Management* (*Northwestern Polytechnical University*), *Ministry of Industry and Information Technology*, *Xi'an* 710072)

**Abstract** Mobile application compatibility failure refers to a kind of software defect caused by application running results inconsistent with expected results in different environments or internal changes. Due to the highly open-source nature of the Android platform, mobile application compatibility failures frequently occur on the Android platform. In the most serious cases, this kind of failure can lead to program crashes. On the one hand, it will affect the user experience. On the other hand, its sudden occurrence will bring huge losses to users. The combinations of different device models and versions of the Android OS make it impossible for developers to test their applications thoroughly. In the context of the extremely serious fragmentation of the Android ecosystem, how to effectively deal with the compatibility faults has become a hot research issue in software quality assurance. Starting from the three aspects of Android mobile application compatibility fault (i.e., fault

**收稿日期**:2021-02-02;**修回日期**:2021-09-07
**基金项目**:国家重点研发计划项目(2020YFC0833105Z1);国家自然科学基金专项项目(62141208);陕西省工业科技攻关项目(2015GY073);陕西省重点研发项目(2021GY-041);信息安全国家重点实验室(中国科学院信息工程研究所)开放课题(2020-MS-07);西北工业大学硕士研究生创意创新种子基金项目(CX2020246)
This work was supported by the National Key Research and Development Program of China(2020YFC0833105Z1), the Special Funds of the National Natural Science Foundation of China(62141208), the Industrial Science and Technology Plan of Shaanxi Province (2015GY073), the Key Research and Development Program of Shaanxi Province (2021GY-041), the State Key Laboratory of Information Security(Institute of Information Engineering, Chinese Academy of Sciences) Open Topic (2020-MS-07), and the Seed Foundation of Innovation and Creation for Graduate Students in Northwestern Polytechnical University (CX2020246).
**通信作者**:陈翔(xchencs@ntu.edu.cn)

analysis, fault detection, fault location, and repair), we briefly introduce the development history of Android mobile application compatibility test and the main challenges faced in this field. In addition, we also review and summarize the practical exploration and theoretical achievements in recent years. Finally, we discuss future work, which can provide guidelines for other researchers to study compatibility testing for Android mobile application.

**Key words** Android mobile application; compatibility test; fragmentation; software failure; fault location; fault detection; fault repair

**摘 要** 安卓移动应用兼容性故障是指应用程序在不同的环境或内部状态发生变化时,实际结果与预期结果不相符合而导致的一类软件缺陷.安卓平台的高度开源的特性,使得安卓平台下移动应用的兼容性故障频繁发生.这类软件故障在最严重的时候,甚至可以导致程序崩溃,程序崩溃一方面会影响到用户体验,另一方面因其突发性也会对用户带来难以估量的损失.因不同设备型号和安卓操作系统版本所组成的大量组合,使得开发人员无法对其应用程序进行充分的测试.在安卓生态系统碎片化异常严重的开发背景下,如何有效地应对兼容性问题成为当前软件质量保障领域的一个热门研究问题.从安卓移动应用兼容性故障的分析、检测、定位和修复3个方面出发,简要介绍了安卓移动应用兼容性故障的发展历程及该领域所面临的主要挑战,并回顾和总结了近些年来该综述主题的实践探索和理论成果.最后,对该领域的未来工作进行了展望,以期为兼容性测试研究人员提供有价值的参考.

**关键词** 安卓移动应用;兼容性测试;碎片化;软件故障;故障定位;故障检测;故障修复

**中图法分类号** TP311

安卓(Android)是由Google公司领导的开放手机联盟(Open Handset Alliance, OHA)开发的一款开放式操作系统.作为全球最受欢迎的移动平台,目前安卓已经拥有超过80%的市场份额和10亿台活跃设备[1-2].与此同时,安卓应用程序的数量也正在以惊人的速度增长,据统计,每月在Google Play上新发布的应用程序数量已多达35 000个[3].与其他应用领域的软件一样,移动应用也面临着各种质量保障问题,除了常见的与软件安全性和可靠性相关问题[4-7]之外,频繁发生的移动应用兼容性问题也越来越受到研究人员的关注.

安卓碎片化(Android fragmentation)是指大量支持安卓的硬件设备、安卓API及安卓操作系统的快速发展,导致了应用程序出现大量可能的运行环境.当前安卓生态系统严重的碎片化现象,已成为导致移动应用兼容性问题频发的罪魁祸首.兼容性问题是软件故障的一种表现形式,也可称作兼容性故障,轻度的兼容性问题可能表现为程序中图片显示不全、控件错位、屏幕显示异常等,而严重的兼容性问题可能会导致程序安装或卸载失败、应用运行时突然崩溃等,从而为用户酿成不可挽回的损失.对于普通用户来说,针对兼容性问题,主要有2种解决方法:1)等待应用程序发行方更新升级程序以适配新的安卓系统;2)将系统回退到程序能够兼容的旧系统版本.因兼容性问题导致的低可用性,会导致用户体验变差,并对程序的使用失去信心.已有实践表明:开发方经常收到用户的兼容性问题投诉,并随后不得不耗费高昂的代价去处理这些问题[8-10].不难看出,生态系统的用户群体规模越大,因兼容性问题造成的负面影响就越大.

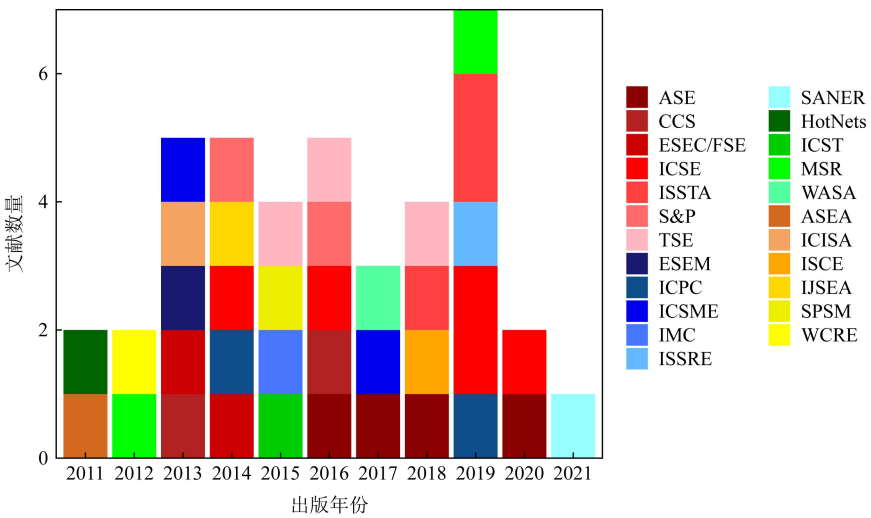
在安卓生态系统碎片化异常严重的开发背景下,处理兼容性问题已经成为具有挑战性的研究问题[11-12].这也给应用程序的开发人员带来了一定的负担,他们需要确保开发的应用程序能够在多种操作系统版本的不同移动设备型号上提供兼容策略,因此依赖于大量的编码和测试工作.实际上,开发人员充分测试他们的应用程序以涵盖设备型号和操作系统版本的所有组合并不可行.与此同时,由于安卓操作系统和API的频繁迭代和更改,使得开发人员必须在短时间内对应用程序进行更新,以确保应用程序在新的API和操作系统版本中正常运行,这也是一个繁琐耗时且容易出错的过程.因此,当前亟需一个高效低成本的自动化的检测、定位和修复工具来解决繁杂的兼容性问题.

为了对该研究问题进行系统的分析、总结和比较,我们首先在IEEE, ACM, Springer, Elsevier,

CNKI 等论文数据库中进行检索,检索时主要采用的英文关键字包括“Android fragmentation”“mobile applications”“compatibility”“software failure”“Android API”等,然后通过阅读论文的标题、摘要和引言对其进行人工审查,排除掉与安卓兼容性问题无关的论文后,再对所筛选的论文的相关工作进行分析和作者已发表论文清单进行分析,并以此来补充与综述主题相关的遗漏文献.

重复上述步骤多次,我们选择出与兼容性问题直接相关的高质量文献共 40 篇(截至 2021 年 7 月),图 1 以论文发表年份为横坐标,以年份发表的论文总数为纵坐标,对论文发表的时间分布进行了汇总.

不难看出,该研究问题是当前安卓系统测试领域的一个热点问题,尤其近 6 年发表的论文数占到该综述主题总发表论文数的 55%.此外,我们对处在 CCF 列表中的论文发表源进行了统计,可以看出,相关文献大部分发表在软件工程领域的核心期刊与会议上,比如 ICSE 会议 4 篇、ISSTA 会议 3 篇、ASE 会议 3 篇、TSE 期刊 3 篇等.




注:文献统计截至2021年7月

Fig. 1 Statistics of papers published in different publication years

图 1 不同年份发表论文统计图

图 2 展示了本文的综述过程.在文献检索阶段,首先选取合适的检索关键词,以计算机学科相关的论文数据库为检索源,同时在标题、摘要、关键词和索引中进行初步检索,对所得文献进行人工内容筛

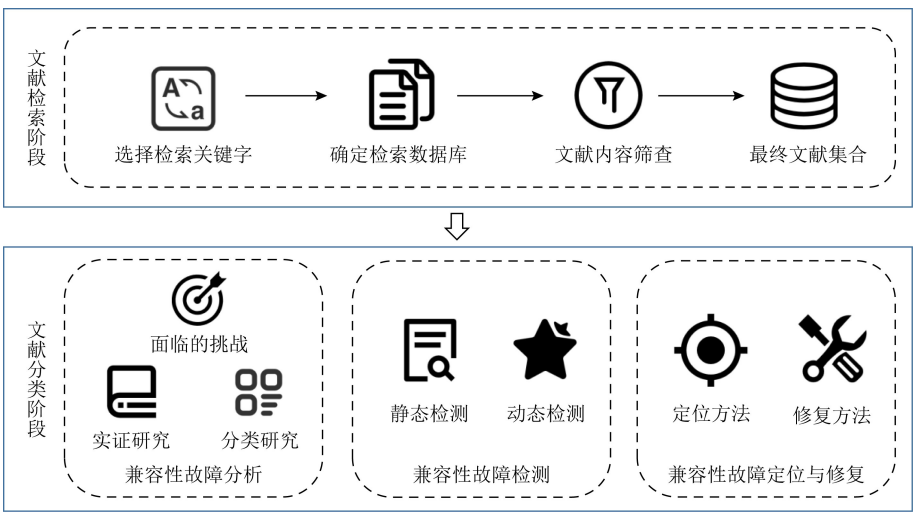



Fig. 2 The framework of Android mobile application compatibility testing

图 2 安卓移动应用兼容性测试的框架图

查,剔除与本综述课题无关的文献,然后得到最终文献集合.在文献分类阶段,我们从移动应用兼容性分析、移动应用兼容性检测、移动应用兼容性定位与修复3个大类对所得文献进行归类.本文将实证研究和分类方法归于兼容性分析研究工作中,探究兼容性问题的表现形式、根本原因、分类依据以及当前所面临的挑战;以发现程序中潜在的兼容性问题为目标,归纳整理文献中提出的兼容性故障检测工具和方法,并根据技术特性作出静态检测和动态检测的进一步分类;以定位程序中兼容性故障的根源以及修复兼容性故障为目标,整理文献中提出的兼容性故障定位和修复方法,并归纳出定位和修复阶段常用的技术框架.最后对不同子类的文献进行总结、评价和比较,探究安卓移动应用兼容性测试领域未来的研究方向.

## 1 基本概念及术语

### 1.1 安卓碎片化

安卓碎片化是指随着大量支持安卓的硬件设备、安卓API及安卓操作系统的快速发展,以及原始设备制造商在其设备上部署的安卓操作系统的定制版本的存在,导致了应用程序存在大量可能的运行环境.安卓生态系统高度碎片化的特性,使开发者在应用程序开发过程中面临十分严峻的挑战,也使其成为一个开放性研究问题[13-17].根据2019年6月腾讯移动分析统计显示,Android 6.0版本以下的市场占比仍然高达26.7%,其中较老的Android 4.4版本竟仍占有5.9%的份额.此统计分析结果表明当前安卓系统版本的碎片化问题已经十分严重[18].

根据Ham等人[19-21]的研究,可以将安卓碎片划分为操作系统碎片和设备碎片,其中设备碎片又可以进一步细分为硬件碎片和API碎片.具体来说:操作系统碎片表示安卓平台版本的频繁更新而导致的碎片.开发商通过降低旧版本终端设备的比例,并对安卓版本进行更新,使得操作系统的碎片化问题已经得到初步缓解[22].而当前比较严重的是设备碎片化问题,其中硬件碎片化是指不同终端设备之间具有不同的硬件规格,尤其以分辨率问题最为突出;API碎片化是指由基本安卓API的演化而导致的各类差异化问题,API碎片化主要体现在API级别上,在每个版本的安卓平台更新时,Google都会为安卓版本设置一个API级别,例如安卓1.0版本所对应的API级别为1,其后每个版本的安卓系统都会对API等级进行升级,即以整数的形式往后累加.截止到2020年6月为止,安卓生态系统中的API级别范围已经拓展到1~28级[23].

### 1.2 应用程序兼容性

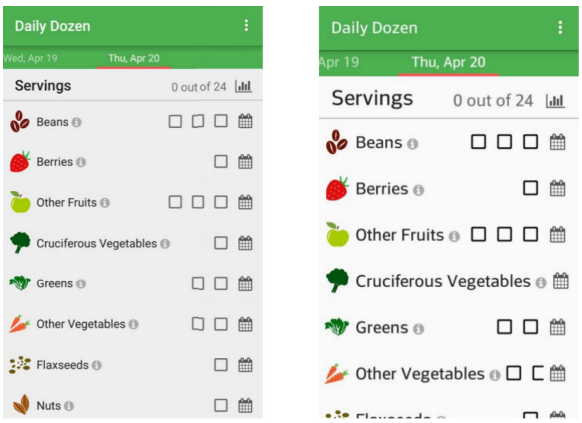
很多已有研究工作分析了因安卓碎片化导致的各种功能性和非功能性问题[24-26],而其中最为显著的就是应用程序的兼容性问题.移动应用程序兼容性(mobile app compatibility)是指应用程序能否在不同的环境或内部系统变化中保持一致的行为[27].安卓应用程序的兼容性可以进一步细分为向前兼容(forward compatibility)和向后兼容(backward compatibility).其中,向前兼容是指在安卓平台的较低版本环境中实现的应用程序可以在较高版本的环境中运行;而向后兼容,又称为回溯兼容,是指最新的应用程序仍然能够访问和处理旧版本中的数据.其中向后兼容主要体现在对安卓API的持续维护上,开发商应尽量保证安卓API只增不减来维持移动应用的向后兼容性.在后续分析中我们会进一步对兼容性的分类进行阐述.

兼容性问题常常会导致应用程序安装或运行失败、运行卡顿或崩溃、功能异常、UI异常等,从而破坏程序正常运行的能力,导致程序实际结果与预期结果不符,因而从本质上属于一种软件缺陷.移动应用兼容性问题的产生原因复杂多样.安卓平台自身版本更迭的需求以及安卓API的不断演化,使得应用程序很难保证向后兼容.系统版本的迭代只是兼容性问题的一个线性增长因素,真正导致其复杂性的根源在于不同终端设备型号的"大杂烩"现象,在各大设备厂商激烈的市场竞争下,来自不同制造商甚至来自同一制造商的设备品牌和型号层出不穷,各种智能终端设备与安卓的适配直接导致版本数量呈指数级增长,并使得移动应用的兼容性很难得到保障[28].除此之外,对第三方库函数的调用也是导致应用程序不兼容的原因之一,开发者通过调用第三方库的资源对应用程序的功能进行扩充,因此第三方库在应用程序中的高流行率也为兼容性问题的出现埋下了隐患[29-30].

图3给出了一个与跨平台相关的兼容性问题示例[31].DAILY DOZEN[32]是一款用户下载超过5万次的移动应用程序,如果将该应用程序分别运行在LG品牌的2个不同型号G3和Optimus L70的手机终端上,由于2个设备的屏幕分辨率不同,在运行同一个DAILY DOZEN应用程序时,在LG Optimus L70设备上可以发现2个易被视觉感知的兼容性问题:

1) 与“Cruciferous Vegetables”标签相关联的复选框元素不可见,使得用户无法勾选此项.这种兼容性故障是由一个与主活动(MainActivity)相关联的布局文件错误引起的.

2) 与“Other Vegetables”标签关联的最右边的复选框元素显示不全,这是由屏幕分辨率或像素密度不同而导致的兼容性故障.




(a) LG G3 (b) LG Optimus L70

Fig. 3 Examples of cross-platform related compatibility issues

图 3 与跨平台相关的兼容性问题示例

在该示例中需要注意的是,在 LG Optimus L70 设备上虽然没有显示“Nuts”标签,但是由于滚动列表的存在,这类现象并不能视为兼容性问题.除此之外,应用程序在开发过程中可供测试的设备数量是有限的,因此可能很难发现这 2 类兼容性问题,这也从侧面反映了兼容性测试面临的研究挑战.

### 1.3 兼容方法

为了预防安卓碎片化带来的兼容性问题,开发者在程序开发初期就十分注重兼容程序的编写[33].以参数设置为例,开发者通过使用 minSdkVersion 和 maxSdkVersion 参数来设置安卓的最低和最高 SDK 版本,以此来限制 API 级别并提升应用程序的兼容能力.如果程序不指定 minSdkVersion 参数的取值或指定错误的 minSdkVersion 参数取值均可能会导致向后兼容问题,使应用程序代码中的一些 API 在较旧版本的安卓中不能被使用,进而导致程序崩溃;如果程序不指定 maxSdkVersion 参数的取值,则会导致潜在的向前兼容问题,因为程序中使用的 API 方法可能在最新版本的安卓平台中不再被支持.

2011 年,安卓平台推出了安卓支持库(Android support library)[23],以处理应用程序中日趋严重的兼容性问题.安卓支持库集成了程序开发中特定的框架组件和 UI 功能元素,这些库主要分为 2 组:兼容库和组件库.其中兼容库主要用来缓解版本演化而带来的兼容性问题,主要包括 v4 兼容库和 v7 兼容库.库中提供了新 SDK 版本的后台移植功能,它为不同 SDK 版本上的接口子集提供包装器,应用程序可以调用支持库中的包装器而不是直接调用 SDK 中定义的 API 函数,使得在新 SDK 版本上开发的应用程序可以在旧 SDK 版本上运行,而无需修改.

此外,Google 公司在 2014 年还进一步推出了安卓兼容性计划(Android compatibility program)[34],该计划旨在让整个安卓社区(包括用户、开发者和设备制造商)受益,通过制定完善的兼容性标准以最大限度地降低与兼容性相关的成本和开销.安卓兼容性计划包括 3 个关键的组成部分:

1) 安卓开放项目源代码;

2) 兼容性定义文档(compatibility definition document, CDD);

3) 兼容性测试套件(compatibility test suite, CTS).

其中 CDD[35]是描述安卓兼容性策略的文档,提供了安卓源代码的使用框架,使其可以用作引用其他内容(如 SDK API 文档)的纲要,促进兼容性系统的实现.CTS 是一个测试安卓 API 兼容性的自动化测试套件,提供了单元测试和功能测试的测试用例,并使用 CTS 验证程序进行补充,以尽早发现兼容性问题,并确保软件在整个开发过程中保持兼容性.

然而,安卓兼容性计划并没有完全解决安卓碎片化和兼容性问题.官方并没有说明通过 CTS 测试的标准,同时商业设备也很难达到 100%的测试覆盖率.最重要的是,CTS 是一个检查安卓设备兼容性的程序,在对应用程序进行开发时,无法使用 CTS 的结构进行测试.

## 2 移动应用兼容性分析

鉴于移动应用兼容性问题的严重性和复杂性,近年来针对移动应用兼容性分析的研究成果不断涌现,主要体现在对移动应用兼容性问题的实证研究和分类研究上.兼容性问题的实证研究具有鲜明的直接经验特征,研究人员通过手工收集数据资料,对提出的理论假设进行实验验证,从而对导致兼容性故障的根本原因进行总结.兼容性问题的分类研究是指研究者对各种兼容性故障进行汇总,并按照某项

指标进行故障分类,以解决同一类别故障重复报告的问题,从而提高开发人员处理兼容性故障的效率.

当前移动应用兼容性问题所面临的主要挑战有3个方面:

1) 安卓碎片化问题严重.安卓操作系统版本的快速迭代、安卓API的不断演化以及不同终端设备型号的更替,加剧了安卓生态系统的碎片化问题,版本数量的爆炸式增长,使得移动应用的兼容性难以得到保障[11-12],潜在的兼容性故障也很难被开发者发现,从而对故障的检测、定位和修复提出了严峻的挑战.

2) 手工测试具有局限性.在移动应用开发过程中,开发人员经常需要将程序移植到不同的型号设备上进行兼容性测试,该移植过程费时费力且容易出错.当发行方经过设备购置、反复测试和问题修复等多个阶段完成对应用的测试,并将应用投放到市场,用户仍会持续反馈应用在开发时遗漏的各种兼容性问题,开发者必须对其快速做出反应,如此繁重的工作量和高昂的代价显然是难以接受的[36-37].

3) 第三方库的大量使用所带来的隐患.大多数安卓应用程序需要依赖第三方库来实现增值功能,例如广告、Google Play服务等.然而,随着时间的推移,这些第三方库与调用第三方库的相关应用功能协同演化,当这些第三方库发布新版本时,开发人员需要对与其相关的应用功能进行相关兼容性测试,以确保能够与旧版本库提供一致的用户体验[29-30].第三方库在应用程序中的大量使用为兼容性问题的出现埋下了隐患.

### 2.1 实证研究

初期研究人员针对安卓应用程序与API演化关系的探索,拉开了兼容性问题研究的序幕.Linaresvasquez等人[38-40]对7 097个安卓应用程序展开了实证分析,他们发现更受欢迎的应用程序通常倾向于使用不易更新的API,经常使用易出错和易更新的API会对应用程序造成负面影响,并导致应用程序出现故障或崩溃.McDonnell等人[41]对安卓API的稳定性和使用率展开了实证研究,他们的研究结果表明,安卓正在以平均每月115个API更新的速度快速发展,然而与快速演化的API相比,开发人员采用最新API版本的平均时间要比API演化时间滞后很多,其平均滞后时间大约为16个月.

Wei等人[42-43]对在开源安卓应用程序中收集的191个兼容性问题进行了大规模实证研究,将安卓碎片化导致的应用程序兼容性问题称为FIC(fragmentation-induced compatibility)问题,并总结出了导致FIC问题的5个主要根源,其中最突出的是安卓API的演化和错误的硬件驱动实现,并提出了FicFinder工具以检测应用程序中的兼容性问题.Wei等人的研究表明,在不同的软硬件环境下,确保不同型号设备上应用程序的兼容性面临严峻挑战.随后He等人[44]对11个不同的安卓版本和4 936个安卓应用程序展开了大规模实证研究,他们的研究表明,导致安卓应用程序兼容性问题的主要原因是因安卓版本更新而导致的API剧烈变化,以及安卓支持库的支持能力有限.这2类问题十分普遍,因此大约92%的应用程序都需要通过编写特定的代码来处理兼容性问题,而开发员最常见的做法(大约78.4%)是通过直接将变量 *android.os.Build.VERSION.SDK_INT* 与常量值进行比较来检查底层系统的SDK版本.Wu等人[45]对2.4万个应用程序进行了研究分析,发现开发人员经常不会对目标SDK版本和最低SDK版本进行预先声明,为程序的兼容性故障埋下了隐患.上述实证研究工作为深入研究因安卓碎片化导致的兼容性问题提供了有益指导.

Mutchler等人[46]将过时安卓API级别产生的应用程序问题定义为目标碎片问题(target fragmentation problem).他们提出了"疏忽性过时"(negligent outdatedness)和"过时"(outdatedness)2个度量指标来衡量程序中API级别的滞后程度以及目标碎片问题的严重程度.通过对1 232 696个开源安卓应用程序进行量化分析,他们发现目标碎片问题主要是由开发人员的疏忽造成的,且会使程序受到兼容性故障的威胁.这一工作有助于开发者意识到过时API级别可能导致的后果,并且重新检查和更改这种缺陷设计,从而减少安卓应用程序中出现兼容性问题的概率.

Huang等人[36]对安卓开发者的API参考文档[47]、安卓API差异报告[48]、安卓开源项目[49]和50个开源安卓应用的100个回调兼容性问题进行了实证研究,他们发现回调API的演化会引起回调API调用协议(callback API invocation protocol)的改变,并将回调API的演化细分为API可达性更改和API行为更改2种方式,前者改变回调API的可达性,而后者改变回调API的调用条件和行为,进而导致程序控制流信息不一致并引发回调兼容性问题.

Xia等人[50]对大约30万个安卓市场应用程序进行了大规模实证研究,旨在揭示开发人员在处理

由 API 演化导致的兼容性问题的具体修复策略.他们发现只有 38.4%的不兼容 API 提供了官方的替代实现,当谷歌提供 API 替代推荐时,开发人员很可能会用推荐 API 来替代不兼容的 API 调用,但推荐 API 所包含的过多参数往往使替代操作变得棘手;当谷歌没有给出替代推荐时,开发人员往往只能根据个人经验对不兼容 API 进行修复,但修复效果十分依赖于开发者自身的专业水平.上述研究中对各类 API 演化进行了深度探索,为解决各类 API 演化导致的兼容性问题提供了契机.

### 2.2 分类研究

兼容性问题一般可以按照兼容特性分为向前兼容问题和向后兼容问题,但这种分类方式较为粗略,无法体现出兼容性问题的产生原因和表现程度,同时也使得一些相似的兼容性错误报告被反复提交.因此,针对兼容性问题的分类研究就显得尤为重要.

Wei 等人[42-43]将 FIC 问题划分为设备无关问题和设备相关问题.如果 FIC 问题可以在运行特定安卓版本的任何设备模型上均能触发,那么该问题就属于设备无关问题;如果 FIC 问题只能在运行特定 API 级别的特定设备型号上才能触发,那么该问题就属于设备相关问题.与设备无关问题相比,由于增加了设备型号这一维度,针对设备相关问题的搜索空间规模也会大幅度增长,并使得设备相关问题更加难以检测.

Huang 等人[36]和 Malinda 等人[51]分别研究了回调 API 兼容性问题和运行时权限兼容性问题,其中回调 API 兼容性问题是指回调 API 的演化而导致的回调 API 调用协议条件和行为的改变,进而引发的回调兼容性故障,而运行时权限兼容性问题是指自安卓 6.0 引入运行时权限机制以来,不同 API 级别的系统不能正确执行被调用 API 的权限检查,或者用户手动撤销了某些 API 的调用权限,进而因为权限机制不适配而导致程序功能失效或运行崩溃等问题.分别以回调 API 演化和权限机制不适配的产生原因作为划分依据,为兼容性问题在故障原因层面的精细划分提供了参考.

Cai 等人[37]以应用程序能否在某个版本的安卓平台上成功安装和运行为依据,将兼容性问题分为安装时不兼容和运行时不兼容两种情况.如果应用程序能够成功安装到至少一个版本的安卓平台,而在另一个版本的安卓平台安装失败,则将其视为安装时兼容性问题;如果应用程序可以在至少一个版本的安卓平台上成功运行指定的时间段且不产生任何错误信息,而在另一个版本的安卓平台无法运行,则将其视为运行时兼容性问题.通过将这 2 类程序兼容性问题进行区分,可以方便研究人员更加深入地对相应兼容性问题的作用范围和阶段进行分析.

对兼容性故障进行分类管理,总结在应用程序开发过程中不同兼容性问题出现的频度,有利于开发人员制定合理高效的软件管理策略,提高软件质量和软件组织生产能力.我们总结并拓展了 7 类不同维度下的兼容性故障分类方法:

1) 兼容特性.兼容特性是指应用程序能否在不同的环境或内部系统变化中保持一致的行为.该分类方法系统地将兼容性问题分为前向兼容问题和后向兼容问题,两者几乎囊括了软件所有的兼容性故障类别,是目前最为常用的宏观分类方法.

2) 硬件相关.硬件相关是指通过更换不同硬件设备型号或组件,来判断程序兼容性故障是否与特定设备型号或组件相关的方法.该分类方法可以有效减小与硬件相关兼容性故障的搜索空间,提高故障检测效率.

3) 产生阶段.产生阶段是指对应用程序在开发、测试、维护等不同阶段产生的兼容性故障进行归类的方法.该方法有利于开发人员在应用程序开发的不同阶段制定不同的故障规避策略.

4) 产生原因.产生原因是指依据兼容性故障产生的根本原因对故障进行分类的方法.该方法是一种精细可靠的分类策略,但分析应用程序兼容性故障根源的过程是复杂且耗时的,时间成本和人力成本相对较高.

5) 故障表现.故障表现是指对故障的不同表现形式进行分类的方法.兼容性故障常常表现为程序安装失败、运行崩溃、功能异常、UI 异常等,依据该方法有利于找寻各类故障表现之间的内在联系.

6) 修复难度.修复难度是指在兼容性故障的修复过程中,根据修复的成本和难度的不同,从而对故障进行修复评估并归类的方法.该方法旨在指导开发人员知悉各类故障的修复难度及其对应的修复成本,尽可能减少多种兼容性故障所带来的损失.

7) 风险程度.风险程度是指以放弃修复兼容性故障所带来的不同程度的损失和后果为依据对故障进行分类的方法.根据不同的风险评估方法,对兼容性故障的危险性进行分类,优先修复风险程度大、危险性高的故障.

表 1 对提及的兼容性故障分类研究所属的分类方法进行了总结,高效的分类方法有利于开发人员

实施多种有效的故障分类方法和管理策略,然而对于不同类型应用程序来说,其兼容性故障产生的原因具有一定的复杂性,并导致开发人员难以筛选出最优的故障分类策略.此外,自动化故障分类工具和分类技术的研究仍处于起步阶段,自动化分类工具的严重匮乏导致故障分类策略的选取完全依赖于专业人士的手工分析,并造成分类效率比较低下.同时,由于各类兼容性故障报告缺乏行之有效的系统化管理,使得当前故障分类管理工作的规范化程度低且极易出错.

**Table 1　Summary of the Dimensions of the Compatibility Fault Classification Research**

**表 1　兼容性故障分类研究所属维度总结**

| 分类研究 | 所属维度 |
|---|---|
| 文献[42-43] | 硬件相关 |
| 文献[36] | 产生原因 |
| 文献[51] | 产生原因 |
| 文献[37] | 产生阶段、故障表现 |

## 3　移动应用兼容性检测

本节以检测过程中是否需要运行移动应用程序为依据,将程序兼容性检测技术划分为静态检测和动态检测 2 种方法.其中静态检测方法是指在不运行程序的情况下,利用控制流分析、数据流分析、符号执行等技术完成对程序开发文档、源代码或字节码的审查和分析,从而检测出各类软件缺陷或兼容性问题.动态检测方法是指通过运行被测程序,比对运行结果与预期结果的差异,并对程序运行效率和健壮性进行分析,通常由构造测试用例、执行程序和分析执行结果 3 个步骤组成.

### 3.1　静态检测方法

当前已有的静态检测方法主要有 4 种,分别是基于规则的方法、基于控制流分析的方法、基于数据流分析的方法以及基于多技术融合的方法.

1) 基于规则的方法.Scalabrino 等人[52]提出了一种自动检测安卓应用程序中 API 兼容性问题的工具 ACRYL.ACRYL 是一种基于数据驱动的方法,它依赖于开发人员在大量应用程序中已经定义的条件 API 用法(conditional API usages, CAUs).ACRYL 以应用程序的 APK 文件作为输入,首先使用 DEX2JAR 将其转换为 jar,并利用 WALA 库来分析获得的 Java 字节码,通过分析应用程序中的每个方法,对包含检查 SDK_INT 字段值的条件语句的方法以及检查结果中具有返回值的方法进行标记,提取出对应的 CAUs;然后 ACRYL 便可以使用它们来推断检测规则,并根据从中学习规则的应用程序的数量为每个规则分配一个置信级别;最后,这些规则可用于检测给定应用程序中的可疑 API 调用.然而,由于研究人员只对开源程序进行规则学习,因此这些规则可能无法适用于一些商业软件.此外,ACRYL 的有效性也会受到规则时效性的影响,从而很容易过时.

2) 基于控制流分析的方法.FIC 问题在检测时需要对设备型号、API 级别和被调用 API 的每个组合进行动态检索,为了缩小 FIC 问题的检索空间,Wei 等人[53]在后续研究中对与设备相关的兼容性问题进行了分析,并开发了 PIVOT 检测工具.为了学习 API 调用与设备型号之间的关联性,PIVOT 通过定义 API-Device 标识符来对关联性进行标识,并定义了一种关系抽取器对应用程序的调用图和每个方法的控制流图进行组合,完成过程间控制流图(inter-procedural control flow graph)的构建,然后对与设备参数识别相关的 API 函数进行识别并计算 API-Device 的关联程度,最后采用 Nguyen 等人[54]提出的排序组件对 API-Device 关联程度进行排序并输出优先排序结果.他们成功利用 PIVOT 对开源安卓应用程序进行了检测,并发现 10 个未被报告过的兼容性问题.该研究提供了检测设备相关的兼容性问题的方法,但是无法对与设备无关的问题进行检测,而且 PIVOT 需要对 API-Device 关联程度的排序结果进行人工验证,因此需要依赖一定的专业知识.

Huang 等人[36]提出了基于协议不一致性图(protocol inconsistency graph)的静态检测工具 Cider,用来检测回调 API 演化引发的兼容性问题.协议不一致性图借鉴了回调控制流图(callback control flow graph)[55]的表示方法,回调控制流图由一个包含回调节点和辅助节点的有向图表示.其中回调节点表示拒绝回调 API 的调用,而辅助节点表示拒绝回调 API 的前后 2 点,即控制流的分支和连接点.Cider 通过捕获安卓应用程序运行时产生的控制流信息,并以此来构建不同 API 级别下的协议不一致性图,通过遍历和比对协议不一致性图即可发现程序中的回调兼容性问题.他们基于 Soot[56]框架实现了 Cider 工具,随后基于 GitHub 的开源项目评估了 Cider 工具的可用性.Malinda 等人[51]对应用

程序运行时权限兼容性问题进行了研究,提出了基于静态控制流分析的 ARPDroid 技术来检测权限兼容性问题.ARPDroid 以要分析的应用程序和一个 API 权限映射作为输入,其中 API 权限映射提供了 API 权限依赖表并收集了每个 API 所依赖的权限信息,然后利用 Soot 框架对程序权限使用的关键点执行静态控制流分析.根据控制流分析的结果,ARPDroid 先后对 targetSdkVersion 参数和 API 权限-负责-调用方(permission-responsible caller, PRC)进行检查,并给出不兼容错误报告.上述研究工作都采用了静态控制流分析的方法对特定的兼容性问题进行检测,但也同时存在检测类型较为单一的问题,且 ARPDroid 对 Soot 功能组件具有依赖性.

3) 基于数据流分析的方法.He 等人[44]提出了基于过程间上下文敏感数据流分析的 IctApiFinder 技术,用来检测由 API 演化导致的程序兼容性问题.他们首先给出了不兼容 API 用法的 3 个必要条件:

① 有一个 SDK 版本的 API 级别大于或等于应用程序中声明的 minSdkVersion 值;

② 应用程序在该 SDK 版本上使用 API;

③ 使用的 API 不包含在该特定版本的 SDK 中.

其中第 2 个条件的判断具有一定的挑战性,IctApiFinder 通过在 Heros[57]上实现过程间数据流求解器(inter-procedural data flow solver),并利用 Doop[58]框架从安卓 SDK 中提取 API 文件并对每个 SDK 版本的 API 信息进行加载,从而精确计算出每一个 API 可访问的 SDK 版本,并以此为依据来检测兼容性问题.通过与安卓 Lint[59]性能的比较,证实了 IctApiFinder 可以有效地减少检测的误报率且具有更大的故障检测范围.与 PIVOT 一样,IctApiFinder 的主要不足在于检测结果必须要经过人工验证,对领域知识具有一定的依赖性.

Li 等人[33]提出了一种自动检测技术 CiD,旨在检测应用程序中潜在的 API 兼容性问题.CiD 技术由 3 个模块组成:

① API 生命周期建模(API lifecycle modeling, ALM)模块.该模块通过挖掘安卓框架的修订历史来构建 API 生命周期模型.

② API 调用提取(API usage extraction, AUE)模块.该模块通过构造条件调用图(conditional call graph, CCG)来对 API 函数进行反向数据流分析,以验证程序是否在与 API 级别相关的条件中调用了 API 方法.

③ API 兼容性分析(API compatibility analysis, ACA)模块.该模块将 API 生命周期模型与目标 API 级别的信息进行匹配,并与 CCG 中 API 的调用条件进行比较,将有问题的 API 调用进行标记,以发现与 API 调用相关的潜在兼容性问题.

CiD 的检测精度和自动化程度较高,但它的局限性也很明显,与 IctApiFinder 一样具有检测故障类型较为单一的特点,使其无法对 API 调用错误以外的兼容性问题进行检测.

Tarek 等人[60]提出了一种自动化检测工具 ACID,该工具旨在利用安卓应用程序源代码的 API 差异和反向数据流分析,以检测由 API 演化所导致的兼容性问题.具体来说,ACID 以谷歌正式发布的安卓 API 差异报告和目标应用程序代码作为输入,并由 4 个功能组件对输入进行处理:

① API 差异分析器.对输入的应用程序代码提取其 API 级别,并根据 API 差异报告识别可疑的 API 调用.

② API 调用定位器.利用自研的词法分析器对应用程序代码进行解析,获取所有的 API 调用情况,并借助反向数据流分析检查每个 API 所属的 API 级别.此外,该组件还可以分析类层次结构,来查找回调 API 使用情况.

③ API 调用兼容性问题检测器.该组件通过使用 API 调用定位器输出的 API 调用情况和 API 差异分析器输出的可疑 API 方法,来检测 API 调用兼容性问题.

④ 回调 API 兼容性问题检测器.该组件首先对 API 调用定位器中生成的类层次结构进行解析,并检查每个 API 类的继承和方法覆盖情况.如果其中有任何覆盖方法被 API 差异分析器所标识,则将其被标记为回调 API 兼容性问题.

此外,与 Cider 相比,ACID 具有更高的检测精度;与 CiD 相比,ACID 具备更少的时间和内存消耗,因而具有更高的检测效率.ACID 因其卓越的检测性能,从而成为目前基于数据流分析方法的典型代表.

4) 基于多技术融合的方法.Wei 等人[42-43]收集并分析了 191 个兼容性问题案例,他们发现大多数 FIC 问题都是由于程序在有问题的软硬件环境中错误地使用安卓 API 而引发的,因此,他们将每个软硬件环境和对应的安卓 API 建模为一个特定的 API 上下文对(API-context pair, acPair),并提出了自动化检测工具 FicFinder.他们在已有案例中选择了 25 个最有可能在应用程序中重复出现的 API

上下文对作为 acPairs 列表,在检测时 FicFinder 以 acPairs 和应用程序的 Java 字节码或 APK 文件作为输入,利用程序依赖图和调用图对程序执行后向切片,根据程序语句依赖关系将语句不断加入到切片中,直到切片收敛为止.最后,遍历切片并根据 acPairs 中的每个 acPair 规则对其软硬件配置和 API 调用情况进行检查,最终输出应用程序中可能存在的 FIC 问题.他们在 Soot 上实现了 FicFinder,并成功检测出 14 个未被报告过的兼容性缺陷,以此验证了 FicFinder 的有效性.然而,FicFinder 在构建 acPairs 的过程中十分依赖于研究人员的手工分析,且覆盖范围较小,也极易过时.

根据以上 4 种方法,我们总结出了移动应用兼容性问题静态检测的一般技术框架,如图 4 所示:

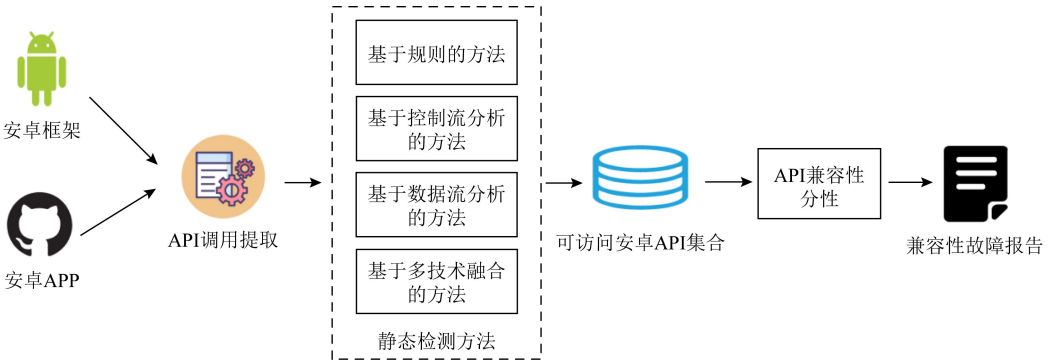



Fig. 4　The framework of mobile application compatibility issues static detection

图 4　移动应用兼容性问题静态检测框架图

### 3.2　动态检测方法

Fazzini 等人[31]提出了 DIFFDROID,这是一种自动化动态检测技术,旨在通过结合输入测试用例生成、用户界面(user interface, UI)行为建模和差异测试来自动识别移动应用程序中的跨平台不一致行为(cross-platform inconsistencies, CPIs). DIFFDROID 将应用程序作为输入,并执行 4 个主要步骤:

1) DIFFDROID 自动为应用程序生成大量的输入测试用例;

2) 将测试用例输入到运行在特定兼容设备的应用程序中,并构建应用程序的 UI 行为模型;

3) 将相同测试用例分别输入到一组其他型号的设备上,并构建应用程序的 UI 行为模型;

4) 将不同设备上构建的 UI 模型与特定兼容设备上构建的 UI 模型进行比较,并向程序开发人员报告排序后的差异.

研究人员在 5 个基准程序和 130 多个设备平台上对 DIFFDROID 进行了评估,结果表明,DIFFDROID 能够有效地识别应用程序上的 CPIs,且误报率较低.DIFFDROID 的局限性在于只能检测因设备型号差异导致的 UI 兼容性问题,而且检测报告必须要经过人工验证,与此同时,由于其测试的应用程序和设备数量有限,很难做到对所有型号设备的全覆盖.

Ki 等人[27]提出了 Mimic 系统来执行自动化 UI 兼容性检测.研究人员定义了一个编程模型来配置具有多个设备型号、安卓版本和应用程序版本的测试环境,并利用运行时(runtime)提供了一个执行 Mimic 脚本的环境.与 DIFFDROID 相比,Mimic 允许使用随机测试和顺序测试等多种测试策略,并支持多设备平台的并行测试,从而大幅度降低了软件测试的时间成本,并减轻了开发人员的开发负担,在一定程度上弥补了 DIFFDROID 存在的缺陷,是当前动态兼容性检测方法中最具代表性的方法,我们总结了 Mimic 的工作流程,如图 5 所示.Mimic 系统首先以一组应用程序和一个脚本作为输入,随后在运行时上执行脚本,并在连接的设备集上执行 UI 测试.Mimic 使用 OpenCV 中的直方图差异[61]、模板匹配[62]和特征匹配[63]3 种方法作为图像处理机制,以此来量化不同 UI 之间的视觉差异.研究人员

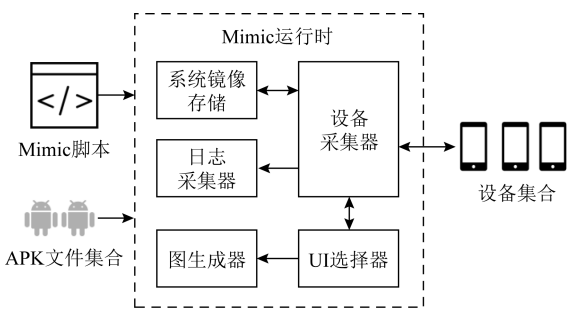



Fig. 5　The working process of Mimic

图 5　Mimic 工作流程图

在向前兼容性、向后兼容性等 4 个测试场景下对 Mimic 进行了性能评估，验证了 Mimic 进行 UI 兼容性问题测试的有效性.然而，由于 Mimic 不测试传感器输入(比如游戏)和系统事件(比如报警弹窗)等触发的 UI 界面更改，因此该系统能够检测的应用程序类型是有限的.

### 3.3 不同类型检测方法的对比

与静态检测方法相比，动态检测方法的优势在于检测准确率高、误报率低，但是随之而来的是其算法研究难度也会大幅提高，制定测试策略十分复杂且缺少实用性高的算法.表 2 从 2 种不同类型的方法中选出具有代表性的检测方法，并进行了对比.

**Table 2 Comparison Among Typical Methods of Mobile Application Compatibility Detection**

**表 2 典型的移动应用兼容性检测方法比较**

| 检测方法 | 工具 | 输入 | 兼容性问题类型 | 核心技术 | 技术优点 | 局限性 |
|---|---|---|---|---|---|---|
| 静态检测 | ACRYL | APK 文件 | 安卓 API 演化问题 | 条件 API 用法 | 规则易于制定，检测效率高 | 规则易过时，检测类型单一 |
| | PIVOT | API-Device 关联标识 | 特定于设备的 FIC 问题 | 静态控制流图，过程间数据流求解器 | 缩小 FIC 问题的检索空间 | 人工验证结果，检测类型单一 |
| | Cider | APK 文件，协议不一致性图列表，关键 API 列表 | 回调 API 演化问题 | 静态控制流图，协议不一致性图 | 检测精度较高，误报率低 | 检测类型单一 |
| | ARPDroid | API 权限映射，APK 文件 | API 权限问题 | 静态控制流分析 | 检测精度较高，自动化程度高 | 检测类型单一 |
| | IctApiFinder | APK 文件 | 安卓操作系统演化问题 | 过程间上下文敏感数据流求解器 | 支持多类型故障检测，误报率低 | 人工验证结果 |
| | CiD | 安卓框架，APK 文件 | 安卓 API 演化问题 | 反向数据流分析 | 检测精度较高，自动化程度高 | 检测类型单一 |
| | ACID | 安卓 API 差异报告，APK 文件 | 安卓 API 演化问题 | API 差异和反向数据流分析 | 检测精度和效率高，自动化程度高 | 检测类型单一 |
| | FicFinder | Java 字节码或 APK 文件，API 上下文对列表 | FIC 问题 | API 上下文对，程序依赖图和调用图 | 规则易于制定 | 规则易过时，覆盖范围小 |
| 动态检测 | DIFFDROID | 基准设备，测试下的应用程序 | 跨平台不一致问题 | UI 行为差异检测器 | 测试用例自动生成，误报率较低 | 人工验证结果，设备型号覆盖不全 |
| | Mimic | Mimic 脚本，APK 文件集合 | 跨平台不一致问题 | UI 行为差异检测器 | 支持多种测试策略，测试成本低且效率高 | 应用程序类型检测不全 |

## 4 移动应用兼容性定位与修复

故障的定位与随后的修复是解决移动应用兼容性问题中 2 项关键性工作，故障定位旨在发现兼容性故障产生的根源所在，而故障修复旨在生成正确有效的兼容性补丁.相对于兼容性故障检测来说，故障的定位和修复由于难度更大，因此更具研究挑战性，现有文献对这 2 方面的研究较少且处于起步阶段，基于现有的研究工作，我们对兼容性故障定位与修复的通用技术流程进行了总结，并提出了移动应用兼容性问题定位与修复框架，如图 6 所示.现有方法一般以目标应用程序的 APK 文件和重现故障的测试用例为输入，在对测试用例执行结果进行计算和分析的基础上发现可疑交互数据块，从而完成对兼容性故障的定位，基于位置信息等依据对 API 调用情况进行排序，利用 API 迁移编辑、API 调用更新等方法获取故障修复策略集合，最后完成对所有

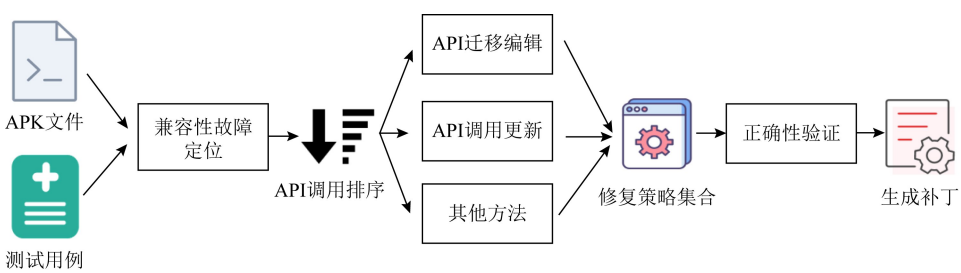



Fig. 6 The framework of mobile application compatibility issues location and repair

图 6 移动应用兼容性问题定位与修复框架图

修复策略的正确性验证,从而生成最终的故障修复补丁.

故障定位技术可以用来定位兼容性故障产生的位置,而不是检测兼容性故障的症状.Wong 等人[64]总结了基于频谱的故障定位(spectrum-based fault localization, SBFL)技术,这种方法搜集测试用例的代码覆盖信息和执行结果,并基于可疑值计算公式(suspicious formula)算出每个语句含有缺陷的概率,最后按照概率值从大到小依次审查语句,直至定位到真正缺陷语句为止,从而实现程序故障定位.这种技术的优点是适用范围较广,可以定位由不同原因导致的多种兼容性故障,但由于其计算的是整个程序中所有语句的故障可疑值,因此缺乏有效的语句筛选机制,并容易忽略各程序语句之间存在的依赖关系,从而导致故障定位效率和精度受到限制.

研究兼容性问题的最终目的是能够高效和低成本地解决此类问题,因此对兼容性故障的修复研究是必不可少的.测试工具仅能够为开发人员提供一些简单的手工修复建议,亟待研发更为高效的自动化兼容性故障修复方法.当前已有的兼容性故障修复方法主要有 2 种,分别是基于 API 调用更新的方法和基于 API 迁移编辑的方法.

1) 基于 API 调用更新的方法.Mobilio 等人[65-66]提出了针对 API 演化而导致的兼容性故障的修复方法 FILO,旨在自动向开发人员报告兼容性故障的可能修复位置.FILO 的输入包括一个重现故障的测试用例和 2 个安卓访问环境,其中 2 个安卓环境分别使用兼容 API 和不兼容 API 来运行应用程序.FILO 的工作主要包括 3 个阶段:

① 测试执行阶段.运行重现故障的测试用例,并从 2 个可用的安卓环境中收集应用程序与安卓 API 之间的交互.

② 异常检测阶段.通过比较收集到的交互差异来识别具有可疑交互的程序块.

③ 修复位置识别阶段.识别可以实施修复的方法位置,并将其对应的支撑证据按照位置信息进行排序.

FILO 将输出的排序结果报告给开发人员,使其可以根据报告对导致兼容性故障的 API 方法进行更改.FILO 的优点在于其不需要执行测试用例,因此修复成本较低,并可以直接对由于系统级交互出错而导致的故障提出修复建议.FILO 的局限性在于无法对不兼容 API 进行自动替换或更新,仍然依赖开发者的手工修复.

Malinda 等人[51]提出的 ARPDroid 工具可以定位并自动修复应用程序存在的 API 权限兼容性故障.在定位阶段,ARPDroid 通过利用静态控制流分析,构造程序的 API 调用图,并从虚拟主方法开始对该调用图执行前向遍历,每当遇到与权限相关的 API 时,对其执行向后遍历直至到达 API 的权限负责调用方(PRC).因此只需要对调用图执行向后的深度优先搜索即可对 PRC 进行定位,从而找到导致权限兼容性问题的 API 位置.

在修复阶段,ARPDroid 首先由检测算法产生使用不兼容 PRC 的列表和每个不兼容 PRC 中不兼容 API 的调用点,然后利用其定义的 2 个修复验证规则对调用点进行验证,当验证失败时,分 2 种情况进行修复:如果之前没有插入 API 所依赖的权限调用,则使用修复算法自动插入;如果该类是内部类,则替换为更高级的祖先类,从而有效解决了程序中 API 权限不兼容的问题,同时,由于 ARPDroid 可以实现修复结果的自动化验证,因此其很好地节省了故障修复的时间成本.

Fazzini 等人[67]提出了一种修复技术 AppEvolve,用来修复 API 演化导致的兼容性问题.在定位阶段,AppEvolve 可以通过程序间数据流求解器计算目标应用程序可以执行的 API 版本集,然后通过程序内分析以识别旧的 API 用法,并检查其中的方法调用是否可以在新版本的 API 上执行.如果是这种情况,则说明此类 API 调用需要更新,并将它们与相应 API 调用的位置一起存储在 API 调用报告中.在修复阶段,AppEvolve 的输入包括要更新的目标应用程序和有关 API 更改的信息,然后执行 4 个步骤:

① 分析目标应用程序以识别由 API 更改引起的代码;

② 搜索现有的代码库以获取对新版本 API 的更新示例;

③ 分析、排序并将上一步中的更新示例转换为通用补丁;

④ 将生成的补丁按排名顺序应用到目标应用程序,同时执行差异测试以验证更新.

AppEvolve 是在提取其他应用程序现有的更新示例的基础上提出的,其运行效率主要取决于 AppEvolve 对更新示例的搜索速度,研究人员在 15 个应用程序上对 AppEvolve 进行了评估,其能够更新 85%的 API 更改,并自动验证 68%的更新,用户研究证实了该修复方法的有效性.AppEvolve 的缺点在于其只能修复特定函数内 API 的更新,不能

处理跨越多个函数的更新.此外,与 SBFL 技术相比,ARPDroid 与 AppEvolve 在定位阶段所采用的故障定位方法都在一定程度上提高了定位精度,然而,两者也同时受限于特定故障类型,无法对不同原因导致的多类型兼容性故障进行有效定位.

2) 基于 API 迁移编辑的方法.Xu 等人[68]提出了 Meditor 工具,旨在使用自动 API 代码迁移方法来修复 API/库演化导致的向后兼容性问题.Meditor 分 2 个阶段执行代码迁移过程:在第 1 阶段,Meditor 对给定库 $L$ 的版本标识出使用库 $L$ 的开源项目,并在项目的版本历史记录中查找所有与迁移相关的(migration-related, MR)提交,对于每个 MR 提交,Meditor 通过语法程序差异(syntactic program differencing)[69]和程序依赖性分析来识别 MR 代码更改并对其进行分组,在通过抽象具体的标识符用法来概括每组 MR 更改之后,Meditor 派生出 API 迁移更改方法并将它们保存到数据库中.在第 2 阶段,Meditor 定义了一个使用 $L$ 的应用程序 $P$,用来枚举数据库中的所有更改方法以决定哪个方法更适用于 $P$,如果更改方法和 $P$ 之间的上下文信息匹配成功,那么 Meditor 将应用这些更改并生成迁移版本程序 $P'$.Meditor 便可以帮助缺乏经验的开发人员找到其他开发人员曾应用的 API 迁移更改,并以此来更改应用程序中的代码,从而修复 API 演化带来的兼容性问题.Meditor 具有很好的故障修复效果,但由于其算法实现难度大且修复结果依赖于人工验证,因此修复成本较高.

我们根据文献中所提及的 FILO, ARPDroid, AppEvolve 和 Meditor 等 4 种兼容性故障修复方法,在表 3 中对这些方法进行了对比.

**Table 3 Comparison Among Typical Methods of Mobile Application Compatibility Fault Repair**

**表 3 典型的移动应用兼容性故障修复方法比较**

| 工具 | 输入 | 兼容性问题类型 | 核心技术 | 技术优点 | 局限性 |
|---|---|---|---|---|---|
| FILO | 测试用例,2 个安卓访问环境 | 安卓 API 演化问题 | API 调用手动更新 | 修复成本低 | 依赖人工修复,修复类型单一 |
| ARPDroid | API 权限映射,APK 文件 | API 权限问题 | API 调用自动更新 | 自动化验证结果,时间成本低 | 修复类型单一 |
| AppEvolve | 版本 API 更改信息,APK 文件 | 安卓 API 演化问题 | API 调用自动更新 | 自动化程度高,修复效率高 | 修复类型单一,无法跨函数修复 |
| Meditor | API/库版本信息,APK 文件 | API/库演化向后不兼容问题 | API 代码自动迁移编辑,静态程序分析 | 自动化程度高,修复精度高 | 人工验证结果,修复成本高 |

## 5 未来展望

针对我们提到的移动应用兼容性问题各阶段的研究工作,本节围绕数据集构建、故障分类方法、故障检测、故障定位与修复、其他技术拓展 5 个方面,对移动应用兼容性问题的未来研究方向进行了展望.

### 5.1 数据集构建

移动应用兼容性故障数据集的构建是开展该领域研究的基础,如何构建大规模高质量的数据集,并建立数据集的自适应机制,是该领域未来很有价值的研究方向之一.

1) 构建大规模、高质量的数据集.在移动应用兼容性问题的检测、定位和修复各环节开发过程中,评测数据集对过程所研发的工具或技术的性能评测至关重要,评测数据集可以给出工具性能的直观评估结果,方便开发人员对工具进行指定功能的改进.当前研究人员大多从 Google Play 的免费开源移动应用项目中获取评测数据,获取的程序文件数量少且覆盖功能类型不全,使得数据集的规模小且无法对工具进行全面的评估.此外,一些研究者由于担心数据集的隐私问题被攻击者利用,从而不愿公开自己的数据集,造成很多实证研究难以重现,是阻碍其他研究人员开展后续工作的关键.因此,构建一个大规模、高质量的兼容性评测数据集就显得尤为重要,研究人员可以在一些开源项目的托管网站(例如 GitHub, SourceForge 等)中拓展数据来源渠道,同时尝试与商业企业合作,扩充商业软件与付费程序,对各类硬件设备信息和第三方库版本数据进行采集,通过制定合理的数据管理机制和隐私保护机制来构建一个开放、安全、可维护的移动应用兼容性评测数据集.

2) 建立数据集自适应机制.安卓碎片化现象使得所构建的兼容性评测数据集具有明显的时效性,硬件设备和安卓平台迭代速度极快,各种移动应用程序版本数量的爆炸式增长对数据集提出了迫切的更新需求.在对数据集进行横向信息扩充的同时,还要针对同一应用程序和操作系统按照演化时间进行

纵向的版本扩充,自适应机制的研发可以有效缓解数据集易过时的问题,在兼容性技术发展的过程中实现数据集各版本数据的动态更新.

### 5.2　故障分类方法

在移动应用兼容性故障的分类方法研究中,如何探索可靠的分类依据,并研发高效的分类工具,也是该领域未来需要拓展的重要研究方向.

1) 探索可靠的分类依据.在已有的移动应用兼容性问题的分类研究中,一般只按照兼容特性分为向前兼容问题和向后兼容问题,而这种粗略的分类方式无法体现兼容性问题的产生原因和表现程度,因此亟需更加精细的分类依据.在后续研究中,可以尝试将兼容性问题按照其产生的根本原因进行分类,然后根据特定原因的产生机制进行更精细的子类划分,或者尝试以兼容性问题的表现形式进行划分,总结出不同表现形式所对应的故障产生类型,以此来获得更加准确的分类依据.

2) 研发高效的分类技术和工具.当前兼容性故障报告大多依靠人工分类,效率低下且容易出错,亟需研发一个高效的自动化分类工具或分类技术,使其能够自动对兼容性故障报告进行合理归类,并给出自动分类依据,或者针对研究人员给出的不同分类依据,完成兼容性故障报告的定向分类和汇总.此外,尝试对兼容性故障报告进行系统化管理,利用自动化管理系统完成对故障报告的格式统一和有序管理.

### 5.3　故障检测

在移动应用兼容性故障的检测方法研究中,如何研发多类型故障检测工具,实现基于二进制代码的检测方法,能够对测试设备进行高效筛选,并对检测结果进行自动化验证,是该领域未来的重点研究方向之一.

1) 研发多类型故障检测工具.已有研究中所提出的兼容性故障检测工具都具有检测故障类型单一的问题,安卓碎片化和第三方库的频繁更新使得现有工具很难对兼容性故障类型进行全面覆盖.其中静态检测方法的误报率高,而动态检测中测试策略的制定十分复杂,且缺少实用性高的检测算法.因此,可以考虑将静态检测方法和动态检测方法进行融合,在拓展工具检测范围的同时,综合考虑运行时间等因素,尝试以最小代价实现多技术的并行化故障检测.

2) 基于二进制代码的检测.当前兼容性故障检测工具多数都是基于免费开源应用开发的,利用控制流图、数据流图等方法对应用程序的源代码进行分析,以此完成对兼容性故障的静态检测.然而,由于很多商业软件不会将源代码对外界公开,因此只将安卓开源应用程序作为研究对象,可能会使工具无法推广到商业闭源应用程序中.当前研究人员在二进制代码级别上开展的兼容性研究十分匮乏,对于商业闭源应用程序,利用其二进制代码执行兼容性故障检测,是未来很有价值的研究方向之一.

3) 兼容性测试设备高效筛选方法.安卓设备的严重碎片化直接导致各类智能终端设备与安卓适配的组合呈指数级增长,在各大设备厂商激烈的市场竞争下,来自不同制造商甚至来自同一制造商的设备品牌和型号层出不穷,提高了企业中兼容性测试的时间成本、财力成本和人力成本,同时也加重了测试人员的负担.如何研发出更为高效的设备终端筛选方法,优化设备相关兼容性测试的覆盖范围,是未来移动应用兼容性测试领域亟待研究的重要课题.

4) 自动验证故障检测结果.现有工具执行兼容性故障检测后的结果一般要经过人工交叉验证,对领域知识具有一定依赖性,且很难保证验证结果的准确率,缺乏高效可行的自动化验证方法.如何利用现有的测试验证技术开发一款自动化验证工具,并完成兼容性故障检测结果的自动验证,同样是未来十分有意义的研究课题.

### 5.4　故障定位与修复

在移动应用兼容性故障的定位与修复方法研究中,如何研发更为精确的故障定位方法和多类型故障修复工具,并探索跨文件/跨函数的修复方法,也是该领域未来的重要研究方向之一.

1) 研发更精确的故障定位方法.兼容性故障定位是一种高效自动的应用程序兼容调试技术,也是兼容性修复工作能够顺利实施的前提.移动应用兼容性故障产生原因的复杂性,使得故障定位的搜索空间十分庞大,导致定位的时间成本过高,且定位结果可能不准确.未来可以尝试减小故障定位的搜索空间,研发更为精确的定位方法,并对比不同故障定位方法对兼容性问题修复的影响效果.

2) 研发多类型故障修复工具.与兼容性故障检测的局限性类似,当前研究人员提出的兼容性故障修复工具覆盖的故障类型不全,且无法保证修复补丁的有效性,修复后也必须要经过人工检验.当前修复的主要方法是将定位到的过时安卓 API 进行更新,旨在解决安卓 API 演化导致的兼容性故障,而对于第三方库中 API 的演化、硬件型号不匹配、

分辨率不统一等导致的兼容性故障,目前还没有有效的修复方法.因此,如何研发一个可以修复多类型兼容性故障的工具,对所定位的故障进行统一修复,是未来研究中亟待解决的难题.

3) 探究跨文件/跨函数的修复方法.当前兼容性故障的修复方法局限于同一个文件内部或函数内部的 API 更改,无法跨越多个文件或多个函数进行故障修复,导致同一修复方式在多个文件重复进行,严重影响兼容性故障的修复效率.探索跨文件/跨函数的兼容性故障修复技术,可以提高故障修复速度,有效缓解修复效率低下的问题.

### 5.5 其他技术拓展

近年来深度学习算法和人工智能领域的蓬勃发展,为软件测试与安全领域的研究带来了契机,深度学习算法在漏洞检测、应用程序分类、故障修复等诸多研究[70-74]中都有着十分出色的表现.将深度学习技术拓展到安卓移动应用兼容性测试领域中,可能会为当前面临的很多棘手问题找到最适合的解决方案.

在兼容性故障的分类研究中,可以利用机器学习算法完成兼容性故障的分类,从而获得一个有效的故障分类工具;在兼容性故障的检测中,结合自然语言处理算法,利用程序数据流和控制流分析得到程序的上下文信息,将程序中有无兼容性故障转化为深度学习算法中的二分类问题,从而获得兼容性故障检测模型,过滤掉检测结果中的误报和漏报,进而提高故障检测的自动化程度和精度;在兼容性故障的定位与修复中,可以利用深度学习算法获得更细粒度的故障定位,并对兼容性故障修复方法作为网络模型输入,以期获得一个自动化补丁生成工具.总体来说,深度学习和自然语言处理算法的应用,在安卓移动应用兼容性测试领域的未来研究中具有十分广阔的前景.

## 6 总 结

在安卓碎片化现象严重的今天,移动应用兼容性问题越来越受到研究人员的关注,本文从安卓移动应用兼容性故障的分析、检测、定位和修复等方面出发,简要回顾和总结了近年来相关课题的实践探索和理论成果.在移动应用兼容性分析中,本文指出了兼容性故障分析面临的三大挑战,总结了研究人员在兼容性问题的实证研究和分类研究中所探索到的故障原因及其发展规律.在移动应用兼容性检测中,我们以检测过程中是否运行移动应用程序为依据,将程序兼容性检测技术划分为静态检测和动态检测 2 种方法,并对静态检测方法进一步划分为基于规则的、基于控制流分析的、基于数据流分析的和基于多技术融合的方法,并对现有检测工具进行了评价和比较.在移动应用兼容性定位和修复中,我们对现有的兼容性故障定位和修复技术进行了总结,并归纳出定位和修复阶段常用的技术框架.最后从 5 个方面展望了安卓移动应用兼容性测试领域的未来研究趋势.

**作者贡献声明**:郑炜负责提出论文整体研究思路、全文框架设计和最终审核;唐辉负责文献调研、内容设计、论文撰写和最后版本修订;陈翔负责部分文献调研和全文修订;张满青负责论文插图设计;夏鑫负责调研分析和全文修订.

## 参 考 文 献

[1] Lunden I. Android breaks 1b mark for 2014, 81% of all 1.3b smartphones shipped [OL]. (2015-01-29) [2021-07-11]. https://techcrunch.com/2015/01/29/android-breaks-1b-mark-for-2014-81-of-the-1-3b-smartphones-shipped-in-total

[2] Rosoff M. The research firm that once thought Microsoft would beat the iPhone has given up on windows phone [OL]. (2015-12-08) [2021-07-12]. https://www.businessinsider.com/idc-smartphone-os-market-share-2015-12

[3] F-Droid. Amaze file manager [OL]. (2016-10-04) [2021-07-12]. https://f-droid.org/en/packages/com.amaze.filemanager/X

[4] Jenkins J, Cai Haipeng. Leveraging historical versions of Android apps for efficient and precise taint analysis [C] //Proc of the 15th Int Conf on Mining Software Repositories (MSR'18). New York: ACM, 2018: 265-269

[5] Cai Haipeng, Ryder B G. Understanding Android application programming and security: A dynamic study [C] //Proc of the 33rd Int Conf on Software Maintenance and Evolution (ICSME). Piscataway, NJ: IEEE, 2017: 364-375

[6] Cai Haipeng, Jenkins J. Towards sustainable Android malware detection [C] //Proc of the 40th Int Conf on Software Engineering (ICSE'18). New York: ACM, 2018: 350-351

[7] Fu Xiaoqin, Cai Haipeng. On the deterioration of learning-based malware detectors for Android [C] //Proc of the 41st Int Conf on Software Engineering (ICSE'19). Piscataway, NJ: IEEE, 2019: 272-273

[8] Han Dan, Zhang Chenlei, Fan Xiaochao, et al. Understanding Android fragmentation with topic analysis of vendor-specific bugs [C] //Proc of the 19th Working Conf on Reverse Engineering. Piscataway, NJ: IEEE, 2012: 83-92

[9] Khalid H, Nagappan M, Shihab E, et al. Prioritizing the devices to test your app on: A case study of Android game apps [C] //Proc of the 22nd ACM SIGSOFT Int Symp on Foundations of Software Engineering (FSE 2014). New York: ACM, 2014: 610-620

[10] Lu Xuan, Liu Xuanzhe, Li Huoran, et al. PRADA: Prioritizing Android devices for apps by mining large-scale usage data [C] //Proc of the 38th Int Conf on Software Engineering (ICSE'16). Piscataway, NJ: IEEE, 2016: 3-13

[11] Joorabchi M E, Mesbah A, Kruchten P. Real challenges in mobile app development [C] //Proc of the 7th Int Symp on Empirical Software Engineering and Measurement. Piscataway, NJ: IEEE, 2013: 15-24

[12] Kochhar P S, Thung F, Nagappan N, et al. Understanding the test automation culture of app developers [C/OL] //Proc of the 8th Int Conf on Software Testing, Verification and Validation (ICST). Piscataway, NJ: IEEE, (2015-04-13) [2021-07-16]. https://core.ac.uk/reader/35455720

[13] Zhou Xiaoyong, Lee Y, Zhang Nan, et al. The peril of fragmentation: Security hazards in Android device driver customizations [C] //Proc of the 35th Symp on Security and Privacy. Piscataway, NJ: IEEE, 2014: 409-423

[14] Choudhary S R, Gorla A, Orso A. Automated test input generation for Android: Are we there yet? [C] //Proc of the 30th IEEE/ACM Int Conf on Automated Software Engineering (ASE). Piscataway, NJ: IEEE, 2015: 429-440

[15] Linaresvasquez M, Moran K, Poshyvanyk D. Continuous, evolutionary and large-scale: A new perspective for automated mobile app testing [C] //Proc of the 33rd Int Conf on Software Maintenance and Evolution (ICSME). Piscataway, NJ: IEEE, 2017: 399-410

[16] Thomas D R, Beresford A R, Rice A. Security metrics for the Android ecosystem [C] //Proc of the 5th Annual ACM CCS Workshop on Security and Privacy in Smartphones and Mobile Devices (SPSM'15). New York: ACM, 2015: 87-98

[17] Wu Lei, Grace M, Zhou Yajin, et al. The impact of vendor customizations on Android security [C] //Proc of the 20th ACM SIGSAC Conf on Computer & Communications Security (CCS'13). New York: ACM, 2013: 623-634

[18] CSDN. Android mobile phone system version distribution in 2019 [OL]. (2019-06-11) [2021-07-13]. https://blog.csdn.net/Coo123_/article/details/91445527

[19] Ham H K, Park Y B. Designing knowledge base mobile application compatibility test system for Android fragmentation [J]. International Journal of Software Engineering and Its Applications, 2014, 8(1): 303-314

[20] Park J, Park Y B, Ham H K. Fragmentation problem in Android [C/OL] //Proc of the 4th Int Conf on Information Science and Applications (ICISA). Piscataway, NJ: IEEE, (2013-06-24) [2021-07-16]. https://ieeexplore.ieee.org/document/6579465

[21] Ham H K, Park Y B. Mobile application compatibility test system design for Android fragmentation [C] //Proc of the 4th Int Conf on Advanced Software Engineering and Its Applications. Berlin: Springer, 2011: 314-320

[22] Tran M. Around the world with Google Developer Day 2011 [OL]. (2011-12-05) [2021-07-16]. https://developers.googleblog.com/2011/12/around-world-with-google-developer-day.html

[23] Android Developers. Android support library [OL]. (2020-05-29) [2021-07-16]. https://developer.android.google.cn/topic/libraries/support-library

[24] Li Huoran, Lu Xuan, Liu Xuanzhe, et al. Characterizing smartphone usage patterns from millions of Android users [C] //Proc of the 15th Internet Measurement Conf (IMC'15). New York: ACM, 2015: 459-472

[25] Pathak A, Hu Y C, Zhang Ming. Bootstrapping energy debugging on smartphones: A first look at energy bugs in mobile devices [C/OL] //Proc of the 10th ACM Workshop on Hot Topics in Networks (HotNets-X). New York: ACM, (2011-11-14) [2021-07-20]. https://dl.acm.org/doi/10.1145/2070562.2070567

[26] Liu Yepang, XuChang, Cheung S C. Characterizing and detecting performance bugs for smartphone applications [C] //Proc of the 36th Int Conf on Software Engineering (ICSE'14). New York: ACM, 2014: 1013-1024

[27] Ki T, Park C M, Dantu K, et al. Mimic: UI compatibility testing system for Android apps [C] //Proc of the 41st Int Conf on Software Engineering (ICSE'19). Piscataway, NJ: IEEE, 2019: 246-256

[28] Sinha V S, Mani S, Gupta M. MINCE: Mining change history of Android project [C] //Proc of the 9th IEEE Working Conf on Mining Software Repositories (MSR). Piscataway, NJ: IEEE, 2012: 132-135

[29] Backes M, Bugiel S, Derr E. Reliable third-party library detection in Android and its security applications [C] //Proc of the 23rd ACM SIGSAC Conf on Computer and Communications Security (CCS'16). New York: ACM, 2016: 356-367

[30] Ma Ziang, Wang Haoyu, Guo Yao, et al. LibRadar: Fast and accurate detection of third-party libraries in Android apps [C] //Proc of the 38th Int Conf on Software Engineering Companion (ICSE-C). Piscataway, NJ: IEEE, 2016: 653-656

[31] Fazzini M, Orso A. Automated cross-platform inconsistency detection for mobile apps [C] //Proc of the 32nd IEEE/ACM Int Conf on Automated Software Engineering (ASE 2017). New York: ACM, 2017: 308-318

[32] Google Play. Greger's daily dozen [OL]. (2021-05-23) [2021-07-20]. https://play.google.com/store/apps/details?id=org.nutritionfacts.dailydozen

[33] Li Li, Bissyande T F, Wang Haoyu, et al. CiD: Automating the detection of API-related compatibility issues in Android apps [C] //Proc of the 27th ACM SIGSOFT Int Symp on Software Testing and Analysis (ISSTA 2018). New York: ACM, 2018: 153-163

[34] Android. Google's Android compatibility program [OL]. (2019-10-02) [2021-07-20]. https://source.android.google.cn/compatibility/overview

[35] Android. Compatibility definition document [OL]. (2020-04-29)[2021-07-20]. https://source.android.google.cn/compatibility/cdd

[36] Huang Huaxun, Wei Lili, Liu Yepang, et al. Understanding and detecting callback compatibility issues for Android applications [C] //Proc of the 33rd ACM/IEEE Int Conf on Automated Software Engineering (ASE 2018). New York: ACM, 2018: 532-542

[37] Cai Haipeng, Zhang Ziyi, Li Li, et al. A large-scale study of application incompatibilities in Android [C] //Proc of the 28th ACM SIGSOFT Int Symp on Software Testing and Analysis (ISSTA 2019). New York: ACM, 2019: 216-227

[38] Linaresvasquez M, Bavota G, Bernalcardenas C, et al. API change and fault proneness: A threat to the success of Android apps [C] //Proc of the 9th Joint Meeting on Foundations of Software Engineering (ESEC/FSE 2013). New York: ACM, 2013: 477-487

[39] Linaresvasquez M, Bavota G, Penta M D, et al. How do API changes trigger stack overflow discussions? A study on the Android SDK [C] //Proc of the 22nd Int Conf on Program Comprehension (ICPC 2014). New York: ACM, 2014: 83-94

[40] Bavota G, Linaresvasquez M, Bernalcardenas C, et al. The impact of API change and fault-proneness on the user ratings of Android apps [J]. IEEE Transactions on Software Engineering, 2015, 41(4): 384-407

[41] McDonnell T, Ray B, Kim M. An empirical study of API stability and adoption in the Android ecosystem [C] //Proc of the 29th IEEE Int Conf on Software Maintenance (ICSM'13). Piscataway, NJ: IEEE, 2013: 70-79

[42] Wei Lili, Liu Yepang, Cheung S C. Taming Android fragmentation: Characterizing and detecting compatibility issues for Android apps [C] //Proc of the 31st IEEE/ACM Int Conf on Automated Software Engineering (ASE 2016). New York: ACM, 2016: 226-237

[43] Wei Lili, Liu Yepang, Cheung S C, et al. Understanding and detecting fragmentation-induced compatibility issues for Android apps [J]. IEEE Transactions on Software Engineering, 2018, 46(11): 1176-1199

[44] He Dongjie, Li Lian, Wang Lei, et al. Understanding and detecting evolution-induced compatibility issues in Android apps [C] //Proc of the 33rd ACM/IEEE Int Conf on Automated Software Engineering (ASE 2018). New York: ACM, 2018: 167-177

[45] Wu Daoyuan, Liu Ximing, Xu Jiayun, et al. Measuring the declared SDK versions and their consistency with API calls in Android apps [C] //Proc of the 12th Int Conf on Wireless Algorithms Systems and Applications. Berlin: Springer, 2017: 678-690

[46] Mutchler P, Safaei Y, Doupe A, et al. Target fragmentation in Android apps [C] //Proc of the 37th IEEE Security and Privacy Workshops (SPW). Piscataway, NJ: IEEE, 2016: 204-213

[47] Android Developers. Android API reference [OL]. (2020-12-15)[2021-07-20]. https://developer.android.com/reference/

[48] Android Developers. Android API differences report [OL]. (2013-10-29)[2021-07-20]. https://developer.android.com/sdk/api_diff/19/changes.html

[49] Android Developers. Android for developers [OL]. (2020-12-15)[2021-07-20]. https://developer.android.com/

[50] Xia Hao, Zhang Yuan, Zhou Yingtian, et al. How Android developers handle evolution-induced API compatibility issues: A large-scale study [C] //Proc of the 42nd Int Conf on Software Engineering (ICSE'20). New York: ACM, 2020: 886-898

[51] Malinda D, Cai Haipeng, Jenkins J. Automated detection and repair of incompatible uses of runtime permissions in Android apps [C] //Proc of the 5th Int Conf on Mobile Software Engineering and Systems. New York: ACM, 2018: 67-71

[52] Scalabrino S, Bavota G, Linaresvasquez M, et al. Data-driven solutions to detect API compatibility issues in Android: An empirical study [C] //Proc of the 16th Int Conf on Mining Software Repositories (MSR'19). New York: ACM, 2019: 288-298

[53] Wei Lili, Liu Yepang, Cheung S C. PIVOT: Learning API-device correlations to facilitate Android compatibility issue detection [C] //Proc of the 41st Int Conf on Software Engineering (ICSE'19). Piscataway, NJ: IEEE, 2019: 878-888

[54] Nguyen H A, Dyer R, Nguyen T N, et al. Mining preconditions of APIs in large-scale code corpus [C] //Proc of the 22nd ACM SIGSOFT Int Symp on Foundations of Software Engineering (FSE 2014). New York: ACM, 2014: 166-177

[55] Yang Shengqian, Yan Dacong, Wu Haowei, et al. Static control-flow analysis of user-driven callbacks in Android applications [C] //Proc of the 37th Int Conf on Software Engineering (ICSE'15). Piscataway, NJ: IEEE, 2015: 89-99

[56] GitHub. What is Soot? [OL]. (2020-10-15)[2021-07-16]. https://github.com/soot-oss/soot

[57] Bodden E. Inter-procedural data-flow analysis with IFDS/IDE and Soot [C] //Proc of the 1st ACM SIGPLAN Int Workshop on State of the Art in Java Program Analysis (SOAP'12). New York: ACM, 2012: 3-8

[58] Smaragdakis Y, Bravenboer M. Using datalog for fast and easy program analysis [C] //Proc of the 1st Int Conf on Datalog in Academia and Industry. Berlin: Springer, 2010: 245-251

[59] Android. Lint API check [OL]. (2013-08-30)[2021-07-20]. http://tools.android.com/recent/lintapicheck

[60] Tarek M, Meiru C, Guowei Y. Android compatibility issue detection using API differences [C] //Proc of the 28th IEEE Int Conf on Software Analysis, Evolution and Reengineering (SANER). Piscataway, NJ: IEEE, 2021: 480-490

[61] OpenCV. Histogram comparison [OL]. (2013-01-02)[2021-07-22]. http://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_histograms/py_histogram_begins/py_histogram_begins.html

[62] OpenCV. Template matching [OL]. (2013-01-02)[2021-07-22]. http://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_template_matching/py_template_matching.html

[63] OpenCV. Feature matching [OL]. (2013-01-02)[2021-07-22]. http://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_feature2d/py_matcher/py_matcher.html

[64] Wong W E, Gao Ruizhi, Li Yihao, et al. A survey on software fault localization [J]. IEEE Transactions on Software Engineering, 2016, 42(8): 707-740

[65] Mobilio M, Rignaelli O, Micucci D, et al. FILO: FIx-LOcus recommendation for problems caused by Android framework upgrade [C] //Proc of the 30th Int Symp on Software Reliability Engineering (ISSRE). Piscataway, NJ: IEEE, 2019: 358-368

[66] Mobilio M, Rignaelli O, Micucci D, et al. FILO: FIx-LOcus localization for backward incompatibilities caused by Android framework upgrade [C] //Proc of the 35th IEEE/ACM Int Conf on Automated Software Engineering (ASE'20). New York: ACM, 2020: 1292-129

[67] Fazzini M, Xin Qi, Orso A. Automated API-usage update for Android apps [C] //Proc of the 28th ACM SIGSOFT Int Symp on Software Testing and Analysis (ISSTA 2019). New York: ACM, 2019: 204-215

[68] Xu Shengzhe, Dong Ziqi, Meng Na. Meditor: Inference and application of API migration edits [C] //Proc of the 27th Int Conf on Program Comprehension (ICPC). Piscataway, NJ: IEEE, 2019: 335-346

[69] Fluri B, Wursch M, Pinzger M, et al. Change distilling: Tree differencing for fine-grained source code change extraction [J]. IEEE Transactions on Software Engineering, 2007, 33(11): 725-743

[70] Li Zhen, Zou Deqing, Xu Shouhuai, et al. VulDeePecker: A deep learning-based system for vulnerability detection [C/OL] //Proc of the 25th Network and Distributed System Security Symp. Reston, VA: ISOC. (2018-01-05) [2021-07-22]. https://arxiv.org/abs/1801.01681

[71] Russell R L, Kim L, Hamilton L, et al. Automated vulnerability detection in source code using deep representation learning [C] //Proc of the 17th IEEE Int Conf on Machine Learning and Applications (ICMLA). Piscataway, NJ: IEEE, 2018: 757-762

[72] Lin Guanjun, Zhang Jun, Luo Wei, et al. POSTER: Vulnerability discovery with function representation learning from unlabeled projects [C] //Proc of the 24th ACM SIGSAC Conf on Computer and Communications Security. New York: ACM, 2017: 2539-2541

[73] White M, Tufano M, Martinez M, et al. Sorting and transforming program repair ingredients via deep learning code similarities [C] //Proc of the 26th Int Conf on Software Analysis, Evolution and Reengineering (SANER). Piscataway, NJ: IEEE, 2019: 479-490

[74] Shabtai A, Fledel Y, Elovici Y. Automated static code analysis for classifying Android applications using machine learning [C] //Proc of the 3rd Int Conf on Computational Intelligence and Security. Piscataway, NJ: IEEE, 2010: 329-333

**Zheng Wei**, born in 1975. PhD, associate professor. Senior member of CCF. His main research interests include software testing, software security, and software warehouse mining.

**郑　炜**,1975 年生.博士,副教授,CCF 高级会员.主要研究方向为软件测试、软件安全、软件仓库挖掘.

**Tang Hui**, born in 1997. Master candidate. Student member of CCF. His main research interests include software testing, data mining, and machine learning.

**唐　辉**,1997 年生.硕士研究生,CCF 学生会员.主要研究方向为软件测试、数据挖掘、机器学习.

**Chen Xiang**, born in 1980. PhD, associate professor. Senior member of CCF. His main research interest include software testing and vulnerability mining.

**陈　翔**,1980 年生.博士,副教授,CCF 高级会员.主要研究方向为软件测试、漏洞挖掘.

**Zhang Manqing**, born in 1997. Master candidate. Student member of CCF. His main research interests include software testing, information security, and deep learning.

**张满青**,1997 年生.硕士研究生,CCF 学生会员.主要研究方向为软件测试、信息安全、深度学习.

**Xia Xin**, born in 1986. PhD, lecturer. ARC DECRA Fellow. His main research interests include data mining, software engineering, and machine learning.

**夏　鑫**,1986 年生.博士,讲师,ARC DECRA 研究员.主要研究方向为数据挖掘、软件工程、机器学习.