软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software, 2023,34(4):1630−1649 [doi: 10.13328/j.cnki.jos.006701]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

面向安卓自动化测试工具综合评估*

钟 怡[1,2], 石孟雨[1,2], 房春荣[1,2], 赵志宏[1,2], 陈振宇[1,2]

[1](计算机软件新技术国家重点实验室(南京大学), 江苏 南京 210023)

[2](深圳研究院(南京大学), 广东 深圳 518063)

通信作者: 陈振宇, E-mail: zychen@nju.edu.cn

摘 要: 自动化测试工具是安卓应用质量保障的主要手段. 随着安卓版本多样性、底层硬件差异性(碎片化)以及逻辑复杂性增加, 自动化测试迎来新的挑战. 为解决这些问题, 近年来, 业界开发出大量自动化测试工具. 但是现有工具数量多, 并且测试重点多样, 测试人员选择工具时存在一定的困扰. 为帮助测试人员选择最佳测试工具, 实现对自动化测试工具的统一评估, 提出了面向安卓自动化测试工具多特征综合评估方法(comprehensive evaluation of Android automated testing, CEAT), 并将其实现为便于测试人员使用的平台. CEAT 在引入测试领域广泛接受的 3 个评估指标, 即代码覆盖率、异常检出率、融合多版本兼容度得分的基础上, 进一步基于变异测试的思想引入变异杀死率, 并从用户体验出发引入 UI 控件覆盖率. 以上 5 个指标构成 CEAT 整个体系, 从而实现安卓自动化测试工具的综合多维评估. 为验证 CEAT 的效果, 生成了 1 089 个变异应用的待测应用集, 在包含 6 个移动设备的真机集群中部署实验, 对 5 个自动化测试工具适配并执行 5 040 次测试任务. 最终结果表明: i) 5 个指标从不同角度对自动化测试工具进行评估, 更加多维地反映不同工具的测试效果, 验证了 CEAT 的有效性; ii) CEAT 支持测试人员为 5 个指标分配不同的权重, 根据实际测试需求得到综合评估结果, 具有一定的灵活性; iii) CEAT 可自动改造 APP 获得变异应用, 同时为工具设置特定平台用于测试, 操作具备简单性. CEAT 可以有效地根据不同测试需求为测试人员选择最佳的安卓自动化测试工具提供参考依据.

关键词: 自动化测试; 碎片化; 变异测试; UI 控件覆盖率; 多维评估

中图法分类号: TP311

中文引用格式: 钟怡, 石孟雨, 房春荣, 赵志宏, 陈振宇. 面向安卓自动化测试工具综合评估. 软件学报, 2023, 34(4): 1630−1649. http://www.jos.org.cn/1000-9825/6701.htm

英文引用格式: Zhong Y, Shi MY, Fang CR, Zhao ZH, Chen ZY. Towards Comprehensive Evaluation for Android Automated Testing Tools. Ruan Jian Xue Bao/Journal of Software, 2023, 34(4): 1630−1649 (in Chinese). http://www.jos.org.cn/1000-9825/6701.htm

Towards Comprehensive Evaluation for Android Automated Testing Tools

ZHONG Yi[1,2], SHI Meng-Yu[1,2], FANG Chun-Rong[1,2], ZHAO Zhi-Hong[1,2], CHEN Zhen-Yu[1,2]

[1](State Key Laboratory for Novel Software Technology (Nanjing University), Nanjing 210023, China)

[2](Shenzhen Institute (Nanjing University), Shenzhen 518063, China)

Abstract: Automated testing tools are the primary means of quality assurance for Android applications. With the increase in Android version diversity, underlying hardware variability (fragmentation), and logical complexity, automated testing faces new challenges. Numerous automated testing tools have been developed in recent years to address the above issues. However, there are vast tools with various testing focuses, making it hard for testers to choose the right one. To help testers select the best tool for testing and achieve a unified evaluation for automated testing tools, a multi-characteristic comprehensive evaluation of the Android automated testing (CEAT) method is proposed and an easy-to-use platform is implemented for testers. CEAT introduces three widely accepted evaluation metrics:

* 基金项目: 深圳市科技创新委员会(CJGJZD20200617103001003); 国家自然科学基金(62141215)
收稿时间: 2021-12-19; 修改时间: 2022-03-12; 采用时间: 2022-05-05

code coverage, exception detection rate, fusion multi-version compatibility score, and further introduces mutation kill rate based on the mutation testing concept, and UI control widget coverage from the perspective of the user. The five metrics constitute the whole CEAT system, thus realizing a comprehensive multi-dimensional evaluation of Android automated testing tool. To verify the effectiveness of CEAT, a set of 1,089 mutated applications is generated for testing, the experiments are deployed in a real-world cluster containing six mobile devices, and 5,040 test tasks are executed for the testing tools. The results suggest that: (i) the five indicators evaluate the automated testing tools from different perspectives, reflecting the testing performance of different tools in a more multi-dimensional way and validating the effectiveness of CEAT; (ii) CEAT supports testers to assign different weights to the five metrics and obtain comprehensive evaluation results depending on the practical testing requirements, which has certain flexibility; (iii) CEAT automatically reconstructs the APP to obtain mutant APPs and set a specific platform for testing the tool, making it convenient to operate. CEAT effectively provides a reference for testers to select the best Android automated testing tool according to different testing requirements.

Key words: automated testing; fragmentation; mutation testing; UI control widget coverage; multi-dimensional evaluation

每月在 Google Play 上发布的安卓应用数量已超过 35 000[1]. 海量安卓应用以及功能需求与应用场景差异大, 给软件质量保障带来挑战[2,3]. 安卓应用更新迭代快, 同时, 开发过程中面临如跨平台和碎片化等诸多新挑战[4]. 为降低测试成本、提高测试效率并满足测试需求, 自动测试技术成为发展趋势.

当前涌现了若干自动化测试工具, 如 Dynodroid[5], ORBIT[6], SwiftHand[7], PUMA[8], Monkey, App Crawler (https://github.com/ seveniruby/AppCrawler), Fastbot (https://github.com/bytedance/Fastbot_Android)等. 面向安卓应用的自动化测试工具推陈出新, 最近 6 年涌现的工具数高达 103 款[9]. 这些测试工具各有侧重, 如 GUI 界面测试、代码覆盖、异常检出、安全性等. 因此, 测试人员难以选择合适的工具, 或在选择测试工具时耗时费力.有效地对自动化测试工具进行综合评估, 成为一个亟待解决的问题.

从工具测评角度来看, 评估指标的选择是影响自动化测试工具评估的一个主要因素. 目前, 软件测试社区已提出各种评估指标, 例如代码覆盖率[10]、易用性[11]、碎片化导致的兼容性[12]、漏洞检测[13]等. 不同的指标测试重点不同, 反映应用质量的角度也不一样, 评估结果也随着指标的不同而有所差异. 常出现的现象是: 不同测试案例在不同指标下评估结果不一致, 依据何种指标来评估存在一定争议. 根据不同的指标, 向测试人员推荐的工具也随之不同, 指标不一致时, 评估结果显然是不合理的. Mao 等人[14]将他们的多目标自动化测试技术 Sapienz 应用在 Monkey 和 Dynodroid 进行比较, 发现 Monkey 的覆盖率高于 Dynodroid; 而 Nariman 等人的研究[15]却得到了相反的结果. 这种由于指标不一致导致评估出现问题的现象, 阻碍了测试人员对自动化测试工具的选择.

现有工作还存在一些问题:

(1) 缺乏客观全面的统一评估标准. 大部分工具的介绍都将重点放在自身现有优势上, 对于工具中存在的一些缺陷和弊端避而不谈. 对测试人员而言, 没有工具之间横向的衡量以及统一的评估方式, 无法快速借鉴现有的评估结果选择合适的工具;

(2) 指标间的信息交叉给拟合带来一定阻碍, 传统的加权计算不能客观地反映其综合测试效果;

(3) 随着技术的不断更新迭代, 安卓应用各个方面必须快速发展(交互性、界面复杂性、应用智能度、安全防护性能等), 以保持其吸引力并适应新设备的特点, 因此以往的测试结果不再适用. 然而, 对自动化测试工具评估呈现增长趋势, 相关研究热度持续升高[16,17]. 并且当前研究多使用安卓仿真机或使用少量安卓机, 测试结果缺乏真实性.

Choudhary 等人[11]在 2015 年研究提出的代码覆盖率、异常检出率、在多平台工作的能力和易用性成为业界普遍认可的指标.

- 代码覆盖率提供具有可行数据的合理、客观、行业标准的度量, 将海量异构信息整合为一个数据, 为开发人员的工作带来极大的便利;
- 异常检出率是检测应用现有故障的一个衡量指标, 故障检测在提高应用质量上占有极大的比重, 因此它是业界评估自动化测试工具普遍认可的指标之一;
- 安卓的开源性导致严重的碎片化问题, 碎片化是指大量的安卓硬件载体、API 以及运载系统等问题导

致安卓应用有更多的运行环境, 这种碎片化现象已经成为兼容性问题频发的主要来源[1]. 本文在原有多平台工作能力评估基础上加以改进, 提出一个设备得分公式, 引入设备占比率和屏幕大小等信息, 用以体现自动化测试工具单个设备测试能力的评估;

- 易用性从用户体验角度出发, 评估自动化测试工具使用时适配等情况. 但易用性对改善安卓应用质量没有实质性影响并且人为因素影响较大, 不将其作为我们的评估指标.

此外, 我们发现应用中真实故障数量有限. 由于故障基数太小导致无法有效评估自动化测试工具, 因此无法判定工具的异常检出效果. 为加强评估自动化测试工具缺陷检验能力, 本文选择变异杀死率作为一项评估指标. 此外, 应用大多由控件组成, 并且安卓应用源码并非全部公布, 缺乏可获取的源码会限制测试场景. 而 UI 控件测试则不需要源码, 从而打破了源码限制. 此外, 用户对安卓应用的体验主要来自页面控件交互点击, 并未直接与底层代码接触. 而 UI 控件覆盖率则是用以检测安卓应用运行过程中, 自动化测试工具是否可以触发所有控件, 从用户体验角度出发, 对应用控件的可用性进行测试. 为测试 UI 控件能够被正常触发的效果, 受到代码覆盖率思想的启发, 本文引入 UI 控件覆盖率作为一项扩充指标.

针对问题(2), 本文将实验部署在 6 种不同品牌移动设备上, 并使用 17 个真实世界的安卓应用经变异后形成 1 089 个待测应用集. 并选取其中 168 个应用完成 5 040 轮测试. 在真机集群上对自动化测试工具进行评估, 为测试人员提供更真实有效的评估结果. CEAT 支持动态扩充待测应用集(根据提供的源码自动改造原 APP 生成变异 APP 以及对 UI 控件覆盖率检测插件进行应用), 以增强评估结果的实时性, 适应不断更新迭代的技术.

本文提出一种面向安卓自动化测试工具综合评估方法 CEAT, 在异常检出率、代码覆盖率、多平台工作能力基础上, 新增变异杀死率、UI 控件覆盖率共五大指标对自动化测试工具进行综合评估. 该评估可总结如下: (1) 一个用于安卓自动化测试工具综合评估方法, 适用于各种类型待测应用集, 为便于测试人员使用, 进一步开发一个评估平台(后面用 CEAT 评估平台用以代指该综合评估平台); (2) 允许多款自动化测试工具访问 CEAT 评估平台, 并支持动态扩充 APP; (3)通过代码覆盖率、异常检出率、变异杀死率、UI 控件覆盖率以及多平台工作的能力, 从设备和应用角度出发完成各项指标之间的拟合, 生成综合评估报告来对自动化测试工具进行评估, 为测试人员提供单指标以及多维指标评估结果.

本文的主要贡献总结如下:

- 多角度筛选并定义评估指标. 本文调研并汇总业界自动化测试工具评估指标使用情况, 抛弃人为影响较大的易用性、对提高应用质量无直接影响的并发性、性能、能源效率以及常被单独研究的安全性等. 从不同角度出发, 最终确定并重新定义异常检出率、代码覆盖率、多平台工作能力、变异杀死率和 UI 控件覆盖率这 5 个指标作为基础评估项;
- 提供统一全面的自动化测试工具评估. 通过研究业界自动化测试工具评估指标发现: 各指标并非相互独立, 而是彼此之间存在信息交叉, 相互影响. 为克服传统加权计算无法客观反映综合测试效果, CEAT 从设备和应用角度出发进行评估获得 *DeviceScore* 和 *ApplicationScore*, 完成 5 个指标的加权拟合, 用于不同工具的横向比较, 并为后续开发改进提供思路;
- 实现 CEAT 评估平台. 该平台实现随机选择或人工指定变异算子这两种方式对源码进行变异, 从而获取待测应用集. 平台可自行调节各标准的权重, 便于研究人员根据需求查看个性化工具评估效果. 本文筛选可运行的变异应用, 形成包含 1 089 个变异版本的待测应用集;
- 实验结果真实全面. 本文在 6 个真实设备上共执行了 5 040 次自动测试工具的测试任务. 实验结果显示, CEAT 评估平台获得的代码覆盖率和触发的异常数目与 Choudhary 测试的结果吻合, 证明了 CEAT 的可用性. 相较于其他研究单一的代码和故障检测维度而言, CEAT 增加了 UI 控件覆盖率, 并借用变异测试丰富待测应用集, 加强错误检测能力评估, 相对传统测试更加多维.

本文首先描述目前安卓市场以及自动化测试工具现状, 针对评估自动化测试工具指标选择问题以及技术更新迭代问题等痛点分析讨论, 得出多维评估和动态扩充的重要性. 第 1 节介绍自动化测试工具相关工作. 第 2 节介绍 CEAT 相关背景技术. 第 3 节讨论本文提出的多特征评估自动化测试方法 CEAT, 详细阐述各步骤具

体操作, 并介绍生成测试报告的各个指标定义以及综合评估公式. 第 4 节对实验设置进行介绍并分析实验结果. 第 5 节进行讨论, 指出 CEAT 中有效性威胁. 最后进行总结, 并给出未来研究方向.

1 相关工作

1.1 自动化测试

自动化测试是提高移动应用程序质量和降低测试成本的有效解决方案[9]. 可将自动化测试工具分为 3 类, 分别是随机探索策略、模型探索策略和系统探索策略. 随机探索策略采用随机方式为安卓应用生成 UI 事件, 产生测试输入. 因此, 系统不需要进行额外的分析计算工作, 可以高效地产生 UI 随机事件. Monkey 是采用随机探索策略的自动测试工具的典型代表, 模拟生成用户的伪随机事件流(点击、触摸或手势). 模型探索策略思想来源于 Web 爬虫和 GUI 测试, 这种工具使用待测应用程序的 GUI 模型[7]生成事件, 系统探索应用程序的各种行为. 工具动态为待测应用产生精确的测试模型. 采用此策略的工具主要有 SiHan 开发的 AppCrawler、Google Jetpack 套件的 App Crawler (https://developer.Android.com/training/testing/crawler)以及字节跳动的 Fastbot 等. EvoDroid 等人[18]研究使用符号执行以及遗传算法探索难以探索的目标, 并采用启发式规则进化, 以最大化代码覆盖率. 但由于此类工具的复杂度较高, 导致工具的伸缩性较差, 目前在工业界还不能够成熟应用. Andre 提出了一个源码级别操作的安卓应用工具 COSMO[19], 引入很小的开销并不改变应用执行轨迹从而完成代码覆盖. Morena 等人提出了突变测试工具 SuMo[20], 从最新的 Solidity 文档和众所周知的变异测试工具开始设计, 实现一组 44 个变异操作符.

针对不断发展的自动化测试工具, 如何评估并帮助测试人员选择合适的工具至关重要. 测试社区不断涌现对自动化测试工具进行评估的研究. Shauvik 等人对现有安卓系统的主要测试输入生成工具进行了全面的比较[11]. Camila Silva 等人通过检查安卓 XML 文件中 ImageView 元素的 content Description 属性发现缺失的替代文本, 据此设计可访问性评估工具来检查移动应用程序是否满足可访问性标准, 并开发了评估工具的原型[21].

1.2 代码覆盖率

代码覆盖率[22]是常用的衡量测试彻底性的一项指标, 反映应用中内容被覆盖的比例, 将大量信息量化为一个数值. 已有不少研究提出提高代码覆盖率, 最常见的就是改进测试用例生成办法[23]. Tengeri 等人[24]研究了异常密度和代码覆盖率之间的关系, 他们仅通过代码覆盖反映套件质量的缺陷密度, 而没有给出综合评估. Takahashi 等人[25]以代码覆盖率为指标, 研究基于模型的扩展测试. 测试社区常用于判定代码覆盖率的工具有 JaCoCo, Emma, Cobertura 等.

JaCoCo 是测量 Java 代码覆盖率且提供相关报告的开源工具包, 具有以下优点.

(1) 易用性. Gradle 直接嵌入对 JaCoCo 的支持, 只需在配置文件中启用插件即可, 易用性更强;

(2) 兼容性. 业界大部分应用的 JDK 版本还停留在版本 8, 而 Emma 并不支持 JDK8, 导致可用性不高;

(3) 性能. 与 Emma, Cobertura 相比, JaCoCo 性能更优越. 安卓编译本身对性能要求较高, 因为系统待测应用集的准备过程涉及到大量编译操作, 性能差别对实验准备过程造成较大影响;

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

1.3 变异测试

变异测试是通过模拟故障, 强制生成有效测试的一种方法[26]. 通过改变源代码细节, 产生一个与先前类似且满足语法要求的语句, 以达到模拟缺陷的目的. 业界对更可靠的变异检测方法及工具的需求正在增长. Yue Jia 的研究成果[27]表明: 变异测试可以有效地对测试效果进行衡量, 且当前的变异测试发展迅速, 变异工具已经十分成熟. 此外, 一些变异工具不仅在实验室中被证明有效, 在工业项目中也同样得到证明, 例如 PIT 工具(PIT homepage, http://pitest.org/, last visited: 2021-6-13). Ahmad 等人使用突变测试评估安卓应用[28]. 以上研究均表明, 变异测试可以有效地对测试效果进行衡量.

变异项目是 MDroid+[29]由威廉玛丽学院的 SEMERU 与安第斯大学、桑尼奥大学以及卢加诺大学于 2018 年共同合作完成的. MDroid+研究过去的测试经验得出 38 个特定变异算子, 自动生成变异算子到目标应用中, 并用于分析测试用例. MDroid+有以下优点.

(1) 适应性. 添加了对安卓特定的支持, 多平台可用;

(2) 可用性. 可编译成功的变异占比更高, 支持多线程产生变异;

(3) 变异质量. 不可执行的变异占比低, 在进行有限次数测试的情况下, 选用此工具可以获得覆盖率更高的测试数据.

1.4 异常检测

提高安卓应用质量是自动化测试工具的最终测试目的, 安卓应用中存在的故障数量直接反映其质量. 代码覆盖率从另一个角度讲, 可以协助开发人员预测安卓应用中存在的异常. 例如: 低测试覆盖率会带来某些影响, 比如故障没有被覆盖, 那么就可能不被发现. 因此在对其评估时, 分析其故障检测能力十分有必要. 已有很多研究将确定质量好坏的依据放在故障检测结果上, 质量的优劣性与故障密度成正比关系[30,31]. 例如: 为预估系统中的实际故障, Tengeri 等人使用从各自的缺陷报告系统计算的缺陷密度度量(作为每个系统大小的确认缺陷数量)[24]. Yaohui 等人[32]从混合测试出发, 结合模糊测试以及导向性随机测试寻找应用中更多的异常, 以寻找异常数量作为评估指标. 考虑到测试的最终目标即为暴露现有故障, 因此我们选择故障检出率作为评估指标.

2 背景和动机

为更清楚大分析各评估指标, 本文调查了业界评估自动化测试技术时的指标使用情况. 首先, 从业界普遍认可的两篇综述中汇总目前评估自动化测试使用最广的 10 项指标[11,33]; 然后, 在 DBLP (https://dblp.uni-trier.de/)检索网站上搜索关键字. 考虑到技术更新, 我们将检索论文年份约束到 2014 年~2021 年共 8 年, 检索时采用英文关键字包括“survey”“code coverage”“GUI Android testing”“bug testing, fault testing”等. 具体判断步骤如下: (1) 检索关键字, 根据论文题目判断该论文是否相关; (2) 当存在争议时, 阅读摘要以及引言部分对其进行人工排查, 同时结合论文发表会议以及期刊信息判断是否相关. 对汇总的 10 项指标相关英文关键字进行检索, 重复上述步骤, 从检索关键字得到的 875 篇论文中找到 381 篇相关论文. 此外, 还可能出现题目中虽包括关键字, 但是内容却与软件无关, 我们将这类论文排除不进行统计, 最终得到表 1 所示的汇总结果.

表 1 2014 年~2021 年软件测试相关指标研究汇总

指标名称 / 检索关键字	代码覆盖率	UI 控件覆盖率	故障检出率	兼容性	易用性	变异测试	并发性	安全性	性能	能源效率
survey	2		28	7	1	1	7	20	10	6
code coverage	85	2	9	2		2		1		1
GUI Android testing		28								
bug testing, fault testing			62			8				
cross-platform testing, fragment testing, cross-platform Android, fragment android				27						
ease of use tool					2					
mutation testing						98			2	1
concurrency Android			1				5			
security Android testing								5		
performance testing Android									4	
energy testing Android										3
共计	**87**	**31**	**100**	**36**	3	**109**	12	27	16	11

根据论文指标调查情况, 按照论文中涉及的指标数量对论文进行分类统计, 得到表 2. 我们发现: 业界评

估自动化测试工具性能或者用来改进测试性能时, 往往通过单指标对其进行评估. 仅仅是单独查看在某一项指标下的实验结果, 并没有将各维度的结果进行融合.

表 2 论文涉及指标个数统计

指标	单指标	双指标	三指标	四指标	五指标
数量	340	34	5	1	1

随着安卓应用规模及复杂性不断扩大, 当时选择的测试应用集已经无法有效反映自动化测试工具的效果. 真实安卓应用实验成本较高[11], 当前研究对自身进行评估大都只使用少量安卓真机, 或直接使用安卓仿真器. 测试并没有在大规模真机的环境下执行, 导致测试结果可能与真实环境中的效果不一致. 为了保证测试的效率, 测试社区已经研究出大量的自动化测试工具. 在不同的研究中, 通过某一项指标对自动化测试工具进行横向比较时, 可能会出现评估结果冲突的现象. 在这种情况下, 测试人员依据不同的研究结果, 在选择自动化测试工具时会出现分歧. 依据单指标评估鲁棒性较低, 例如不同的移动应用、实验设备等, 对结果产生较大的影响.

3 综合评估 CEAT

本节我们将详细介绍多特征综合评估自动化测试框架: CEAT. 采用添加变异测试模拟特定的待测缺陷, 用以评估自动化测试工具对特定缺陷检出的能力; 并引入 UI 控件覆盖率, 加强对安卓控件使用情况能力评估. 从用户体验出发, 检测自动化测试工具在 UI 测试中对 UI 控件的覆盖的能力. 针对安卓严重的碎片化问题, 引入对版本、机型等兼容度的衡量[34]. 最终在真机集群上对自动测试工具进行多维综合评估, 为测试人员选择自动化测试工具提供参考. 基于此评估方法实现的 CEAT 评估平台相关源码公开发布在 GitHub, 网址为 https://github.com/tykuyh/FEAT.

3.1 整体设计

本文提出的 CEAT 方法框架如图 1 所示. 该方法的研究主要基于 5 个测试评估指标, 对自动化测试工具各个维度进行评估. 整个评估过程包括 3 部分: 确定指标、CEAT 平台评估和模型评估. 指标确定是整个 CEAT 评估的基础, 通过统计业界对各指标研究情况并分析指标之间的交叉信息, 然后从不同角度出发确定评估指标. CEAT 平台评估通过对源码进行改造, 完成变异获取待测应用集, 通过待评估工具的测试获得评估数据. 最后参照 CEAT 拟合公式, 根据收集到的数据计算出最终评估结果. 以数据形式显示各个工具在各指标下的测试效果.

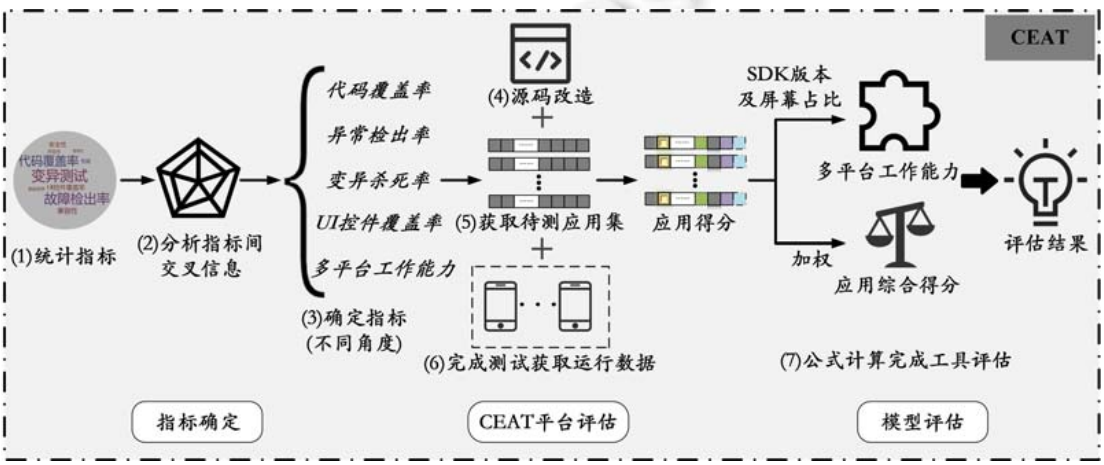

图 1 CEAT 方法框架

CEAT 主要分为以下 7 个步骤.

- 步骤 1: 统计指标. 调研并统计业界在评估自动化测试工具时使用的指标情况, 统计各指标频度;
- 步骤 2: 分析指标间交叉信息. 根据调研结果分析指标之间的相互影响, 通过分析发现各个指标之间并非完全独立而是存在信息交叉, 彼此之间相互影响;
- 步骤 3: 确定指标(不同角度). 从不同角度出发确定综合评估指标, 根据评估需求对各个指标如何计算进行定义;
- 步骤 4: 源码改造. 提供真实应用源代码, 通过在应用程序上插入探针采集 CEAT 所需的代码、UI 控件等信息, 并帮助后续完成变异操作;
- 步骤 5: 获取待测应用集. 生成变异创建真实应用程序的突变版本获取待测应用集(AUT), 并人工审核并筛选可运行的变异 APP, 为实验提供测试数据;
- 步骤 6: 完成测试获取运行数据. 在真机上, 分别对 AUT 执行自动化测试, 同时收集运行时相关数据;
- 步骤 7: 公式计算完成工具评估. 分析自动化测试工具运行得到的数据并计算各标准得分, 通过综合评估模型的拟合公式计算综合评估结果.

CEAT 中比较重要的两部分就是测试数据准备以及根据提出的模型拟合公式对收集数据进行分析计算. 为使测试结果更加真实, 以下载量为依据, 选择各类别的应用程序作为原始待测应用集. 变异测试的目的是通过分析变异 APK 和原始 APK 的检测结果, 获得变异杀死率. 为便于分析, 将 CEAT 的上述 7 个步骤用 3 个阶段表示: 分析并确定指标阶段(阶段 I)、CEAT 平台评估阶段(阶段 II)、CEAT 评估模型拟合公式计算阶段(阶段 III). 其中, 阶段 I 涉及上述步骤 1–步骤 3, 阶段 II 包含步骤 4–步骤 6, 阶段 III 即为步骤 7. 下面我们详细介绍这 3 个阶段.

- 阶段 I. 此阶段为准备工作, 调查并统计业界研究人员在评估自动化测试工具指标选用情况, 并对相关研究进行分析. 通过调研发现: 指标彼此之间存在信息交叉, 相互影响. 例如代码覆盖率, 代表覆盖范围越大, 异常检出率也就越高. 这些交叉信息给指标之间的拟合带来挑战. 之后, 从不同角度出发, 研究人员、用户、硬件兼容性等确定评估指标;
- 阶段 II. 此阶段将阶段 I 得到的指标作为基础指标项, 然后通过开发的 CEAT 评估平台对各自动化测试工具进行综合评估. 该阶段需要向 APP 源码中插入探针(插桩)并完成变异, 将变异后可运行的 APP 作为待测应用集. CEAT 平台实现随机自动/手动选择变异因子, 极大方便研究人员的使用. 插桩后的应用为后续数据收集提供便利, 方便获取覆盖信息、UI 控件以及变异杀死等数据. 然后引入待评估工具, 对待测应用集进行测试, 从而收集运行时的信息. 该阶段选择 6 台不同品牌的安卓真机作为载体, 然后分别对待测应用集中的原应用和变异应用进行测试, 并在运行过程中, 对真机以及自动测试工具进行相关的数据收集;
- 阶段 III. 本阶段根据阶段 II 收集到的数据信息, 按照第 3.3 节的评分模型对各指标进行计算. 阶段 II 完成应用预处理(插入探针), 通过对自动测试工具及 CEAT 系统进行监控, 收集应用运行时生成的文件、设备日志和被测工具的日志信息. 这些数据会存储为原始数据, 作为评分模型的输入. 根据 CEAT 评估模型公式计算 5 个指标得分, 同时支持用户根据具体需求赋予不同指标不同的权重. 为克服指标之间的信息交叉, 彼此之间存在的影响, CEAT 评估模型从设备和应用这两个角度出发, 计算设备得分(多平台工作能力)和应用得分, 然后完成加权拟合. 根据 CEAT 评估模型中拟合公式得到最终评估结果, 最后平台将测试结果进行可视化显示.

CEAT 整个评估流程即通过工具对待测应用集进行测试, 获取运行以及测试信息, 最终根据模型拟合公式计算各指标得分. 以数据形式对自动化测试工具进行直观评估, 将大量信息拟合为综合评估结果. 通过获取安卓应用测试效果以及安卓设备的不同特征, 根据单指标下的评估公式以及综合评估公式进行计算, 最终将大量信息浓缩为 5 个单指标评估结果及综合评估结果共 6 类数据. 不仅单指标可以进行横向对比, 而且可以结合不同的指标查看评估结果. 在没有额外条件限制下, 选用平均值作为工具评估结果可视为最佳选择,

因此, CEAT 将综合评估结果中 5 个指标权重初始化为同一数值. 考虑到不同安卓应用测试社区有着不同的测试需求, 支持测试人员根据自身需求为不同的指标赋予不同的权重, 计算符合各自特性的自动化测试工具的综合评分.

3.2 CEAT五大指标评估模型

下面详细介绍 5 个指标各自的公式以及综合评估公式, 并给出具体定义. 覆盖率是衡量软件性能的重要指标之一, 是评估软件测试程度的一个重要手段. 公式(1)为软件测试社区计算覆盖率的通用公式:

$$Coveragerate=\left(\frac{item\_covered}{item\_total}\right)\times 100\% \tag{1}$$

公式(1)中, 假设要对 *item* 的覆盖情况进行计算, 可以将本文需要计算的控件和代码代替 *item* 来计算控件覆盖率与代码覆盖率.

定义 1(代码覆盖得分). 为测试基本工具的代码覆盖率, 在测试时兼顾传统自动化测试指标, 引入代码覆盖率作为基础指标之一. JaCoCo 简单易用且兼容性好, 因此本文选择该工具来查看自动化测试工具的指令覆盖率、分支覆盖率、圈复杂度、行覆盖率、方法覆盖率、类覆盖率等相关信息. 通过数据文件可以生成应用的代码覆盖率报告, 计算覆盖率报告中的数据, 如公式(2)所示:

$$CoverageRate=\sum_{i=1}^{n}\left(\beta_i\times\frac{Meta_i-Missed_i}{Meta_i}\right)\times 100\%,\ \sum_{i=1}^{n}\beta_i=1 \tag{2}$$

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

定义 2(异常检出得分). 异常检出得分即评估自动化测试工具对待测应用中现有故障的检测效果. 故障检测借鉴 Choudhary 等人对当时的自动测试工具研究指标, 通过对现有故障进行检测暴露应用中现有故障. 通过对比测试查找的异常数目与系统记录的异常数目, 得到其异常得分, 如公式(3)所示:

$$ExceptionRate=\frac{Exception_{found}}{Exception_{most\_found}}\times 100\% \tag{3}$$

定义 3(变异得分). 通过分析检测杀死变异的比例获得变异原始分, 作为多维测试中较为重要的评估项. 变异是本工作的一个亮点, 用户可手动选择变异算子完成相应变异功能. 变异完成后, 需在执行变异文件时记录相关的日志信息, 作为后续评估主要依据. 具体计算方法见公式(4):

$$MutantRate=\frac{Mu\tan t_{killed}}{Mu\tan t_{total}}\times 100\% \tag{4}$$

在该公式中, $Mutant_{killed}$ 是自动测试工具杀死变异的数目, $Mutant_{total}$ 是测试变异应用总数. 通过该公式, 可以计算工具变异得分.

定义 4(UI 控件覆盖得分). UI 控件覆盖是从用户角度出发, 面向用户直接接触的应用控件进行应用质量的测试. 对于当前部分应用不开源的问题, UI 控件覆盖也给出解决思路. 此外, 根据分析源码难以获得当前应用所有控件信息, 应用源码不报错并不能和控件点击正常划等号, 比如一些遮盖现象. 因此, 考虑用户真实使用情况, 引入 UI 控件覆盖作为扩充的测试评估指标. 通过在应用编译代码过程中使用 ASM 框架对 class 文件进行修改来实现插桩, 对点击事件进行监控. UI 控件覆盖率可以用来考察自动测试工具在 UI 测试中对控件覆盖的能力, 系统同样会记录测试工具对该应用测试时所点击的 UI 控件信息. 通过对比测试点击的控件数目与系统记录的控件数目, 可以得到控件得分, 如公式(5):

$$ComponentRate=\frac{Component_{found}}{Component_{most\_found}}\times 100\% \tag{5}$$

在本公式中: $Component_{found}$ 为该次测试中自动测试工具所点击的控件数量; 而 $Component_{most\_found}$ 与异常

得分计算方式相同, 为测试中所有测试工具检测控件的最大值. 通过该公式的计算, 可以获得控件覆盖的原始分.

定义 5(设备得分). 海量的市场以及安卓系统的开源性, 导致安卓设备的碎片化问题. 碎片化为安卓应用开发造成了极大的隐性开销, 其中, 界面显示受到碎片化影响尤为严重. 例如: 由于碎片化造成的兼容性故障, 导致应用出现图片溢出、控件错位以及程序突然跳出崩溃等[1]. 这些问题给阻碍了开发方进一步对程序进行扩展, 开发方不得不耗费大量人力财力处理此类问题以应对用户的反馈[35−37]. 针对以上问题, 作为基本的自动化测试工具衡量指标, 引入设备在多平台工作等兼容度相关问题评估工具. 设备中单个应用的得分, 是自动化测试工具的评估体系中一个评分单元. 该评分代表工具对单设备上的单应用测试能力的评估, 具体计算方式如公式(6)所示:

$$DeviceApplicationScore=\alpha_1*CoverageRate+\alpha_2*MutationRate+\alpha_3*ExceptionRate+\alpha_4*ComponentRate \tag{6}$$

其中参数均由公式(2) −公式(5)计算所得. α_i 为常量, 代表不同测试类别在最终得分的比重, $\sum_{i=1}^{4}\alpha_i=1$. 在测试人员没有任何侧重点的情况下, 设备应用得分为 4 个基本指标得分的均值, 也就是这里的α_i 默认取 0.25. 此外, CEAT 支持根据测试需求, 有所侧重地对各项指标的占比进行修改, 即调整评估模型的拟合公式中各指标的权重.

设备得分是自动测试工具在不同版本设备上测试能力的综合评估, 首先将 *DeviceApplicationScore* 按设备分类, 求出单个设备测试应用集的平均分; 然后将设备得分按照 SDK 版本以及屏幕在安卓设备中的占比加权, 得到设备得分, 可以有效体现自动测试工具处理碎片化等兼容度问题的能力. 具体如公式(7)所示:

$$\begin{aligned} DeviceScore=&\frac{\sum_{j=1}^{m}\left(\frac{\sum_{i=1}^{n}DeviceApplicationScore_{j\_i}}{n}\times Partition_{j\_sdk\_version}\right)}{\sum_{j=1}^{m}Partition_{j\_sdk\_version}}\times\alpha+\\ &\frac{\sum_{j=1}^{m}\left(\frac{\sum_{i=1}^{n}DeviceApplicationScore_{j\_i}}{n}\times Partition_{j\_screen}\right)}{\sum_{j=1}^{m}Partition_{j\_screen}}\times(1-\alpha),\ \alpha\in(0,1) \end{aligned} \tag{7}$$

公式中: n 为待测应用集的数量; m 为设备的数量; $Partition_{j\_sdk\_version}$ 为当前设备 SDK 版本在所有安卓设备中的占比; $Partition_{j\_screen}$ 为当前屏幕属性在所有安卓设备中的占比; α为常量, 代表 SDK 和屏幕版本在设备得分中的比重.

定义 6(CEAT 综合评估模型). 单独的评分仅能从自动化测试工具的单个维度对其进行评估. CEAT 通过提出一个综合评估公式, 实现融合多项指标的评估模型, 获得自动化测试工具的综合评估结果. 应用得分是自动测试工具对应用测试能力的综合评估. 首先将不同设备上相同应用的 *DeviceApplicationScore* 得分加和后求平均值, 该值即为自动化测试工具在设备集群上测试单个应用的原始得分, 将其平均结果作为工具在该测试应用集上的综合评分. 如公式(8)所示:

$$ApplicationScore=\frac{\sum_{i=1}^{n}\overline{DeviceApplicationScore_i}}{n} \tag{8}$$

其中, n 为待测应用集中的应用数量; $\overline{DeviceApplicationScore_i}$ 即为单个应用在不同设备上测试的平均得分. 再将其在所有设备上的得分平均化, 即可得到结果.

FinalScore 为自动化测试工具的最终得分, 体现了工具的综合测试能力. 该得分是由 *ApplicationScore* 与 *Devicescore* 加权组成, 如公式(9)所示:

$$FinalScore=ApplicationScore\times\alpha+Devicescore\times(1-\alpha),\ \alpha\in(0,1) \tag{9}$$

其中, α为常数, 测试人员可以根据自己需要的测试重点灵活为其分配相应的权重.

通过 CEAT 提出的多特征综合评估模型拟合公式, 计算最终的 *FinalScore*, 提供多维度的评估结果, 以更

可靠的方式为测试人员选择工具提供依据.

3.3 CEAT平台详细设计

本节以实现的 CEAT 评估平台为例, 详细介绍 3 个阶段中的一些主要实现.

源码改造. CEAT 评估平台需要获取 5 个指标的信息, 包括对现有故障进行检测的异常检出率, 行覆盖、分支覆盖、指令覆盖等自动化测试工具下的覆盖信息、SDK 版本以及屏幕相关反映碎片化的内容, 变异杀死率、UI 控件覆盖信息等结果, 并获得相应数据. 为能够分类获取各指标所需的信息, 我们通过插桩埋点来收集相关日志信息, 对工具进行分析评估. 执行插桩操作可以解决以下几个问题. JaCoCo 插件使用 Offline 方式对代码进行埋点, 该方式需要在系统中存储 coverage.ec 文件, 通过对该文件进行分析, 可以得到代码覆盖率报告. 由于有些应用不涉及文件的读写操作, 所以需要修改应用的文件读写权限. 生成运行时文件以及开启对点击事件的监控均需初始化, 并修改应用初始化的相应文件. 实现覆盖率测试需要插件以及模块的引入, 对应用源码进行相应的配置;

生成变异应用. 本文的核心之一是检测自动化测试工具发现安卓应用变异的能力, 通过代码修改, 迫使应用存在漏洞或失败. 然后, 通过变异应用和原应用之间自动化测试的结果进行对比分析, 获取杀死变异的数量, 计算变异得分. 检测自动化测试工具变异杀死率作为评估工具的一个指标, 使用目前较为成熟的变异工具 MDroid+完成待测应用的变异工作[29]. MDroid+有 38 个变异算子, 不同类型的变异算子可以进行不同类别的测试. 引入的变异算子越多, 测试范围就越大, 覆盖率也会随之上升. CEAT 评估平台随机选择不同的变异算子为原应用创建变异版本, 也支持测试人员自由选择变异算子. 若用户提供变异算子, 则使用提供的变异算子进行变异; 若未提供, 则随机生成. 由于对应用内部源码进行修改, 测试人员缺乏对安卓应用深层度的实现方法以及结构安排等知识, 对我们的工作造成一定困扰. 因此在对应用源码进行变异操作后并筛选, 将可运行的变异应用作为原始实验数据;

执行测试. 对应用程序预处理后, 完成原应用相关变异操作. 然后对待测应用数据集进行测试, 通过获取相应测试信息以及工具运行时信息评估工具. 为保障获取评估模型中各指标计算需要的信息, 需要 CEAT 评估平台启动测试服务, 通过接口保证 CEAT 可以正确执行并监测自动化测试工具. 开始测试之后, 首先获取将要进行测试的设备列表, 列表中的安卓设备包括后续测试将要使用的 SDK 版本、序列号等信息; 随后, 以设备为单位, 为每个设备准备一个异步测试的队列, 队列中的每个元素是一次完整的测试任务, 包括准备测试环境、启动监控线程、启动日志记录、启动测试工具以及收集运行时文件. CEAT 实现自动获取测试需要的运行时日志文件数据用以评估.

4 实 验

本节基于测试获得各指标下的得分, 最终通过 CEAT 公式计算得到综合评估结果.

4.1 研究问题

在本节中, 我们调查了 CEAT 的具体表现, 通过回答以下 3 个研究问题讨论其有效性.

- RQ1: 5 项评估指标的关联性及其优缺点?
- RQ2: 多维综合评估方法的优势所在?
- RQ3: 与现有其他评估方法的优势所在?

4.2 数据集

在自动化测试工具方面, 拟采用测试常用工具, 如 Monkey, Dynodroid, Sapienz 等. 但部分工具存在兼容性和可用性问题, 在表 3 中列出这些技术所存在的问题, 最终没有选择这些工具作为实验对象.

表 3 自动化测试技术中存在的问题

技术名称	问题类型	问题
Sapienz	安卓 SDK、兼容性	公共版本的 Sapienz 只能支持安卓 4.4.2 模拟器
A3E	多设备、兼容性	A3E 未提供选择运行哪些设备的接口，不支持在多设备环境下运行
PUMA	多设备、兼容性	PUMA 只能使用低于 25.0.0 的 SDK 构建工具来运行，并且不能同时在具有不同 SDK 版本的设备上运行
SwiftHand	基础设施、兼容性	SwiftHand 需要分支级别的 Apk 检测，但它只能在其演示应用程序上运行．已停止维护
Dynodroid	安卓 SDK、兼容性	Dynodroid 只能在安卓 2.3 设备上运行
GUIRipper	根权限	GUIRipper 需要 root 权限才能访问中间文件．无法在无根的物理设备上运行

由于安卓应用是事件驱动的，输入通常以事件的形式出现，它可以模拟用户交互(UI 事件)，如点击、滚动和文本输入，或系统事件，如新接收的短信通知．当前，大部分自动化测试工具都是按照不同策略自动生成相关输入完成对应用的测试．为兼顾不同测试输入策略下的自动化测试工具，在每种策略下，选择最具有代表性的作为实验对象．最终选择以下 5 款适配的工具，见表 4.

其中，Monkey 可以随机创建触摸和滑动等行为，本文实验设置了 3 个参数：seed(100)，events number(3600)和 throttle(100)．AppCrawler 是一款基于深度优先遍历策略的安卓爬虫，可以通过命令行调用来启动，然后设置其执行时间．Maxim(https://github.com/zhangzhao4444/Maxim)可以采用脱机/非脱机这两种运行方法，将两个 jar 文件 push 到手机的根目录即可．Fastbot 通过接口接入 CEAT，设置参数见表 4.

表 4 适配的自动化测试工具

编号	工具名称	测试策略	详细介绍(运行参数)
T1	Monkey	随机探索策略	由名为 GithMonkey 的 Shell 脚本来启动执行，采用随机策略，不能自定义．向系统发送伪随机的用户事件流(如按键输入、触摸屏输入、手势输入等)(-s 100 --throttle 100 -v -v -v 3600)
T2	Google AppCrawler	模型探索策略	类似 Monkey,调起应用程序并执行各种动作(点击、输入、滑动等)，查看各种情况下应用程序状态．当应用崩溃或达到指定的超时时间终止运行(--time-sec 100000 --ui-automator-mode)
T3	App Crawler	模型探索策略	与 Web 爬虫思想相同，基于 Appium 实现测试框架，通过 resource-id, content-desc, text 和 index 属性来确定唯一控件．在支持自动爬取的同时，还支持丰富的自定义配置(--capability noReset=true -t 3600)
T4	Fastbot	混合策略	对 GUI 建模，结合机器学习与强化学习增强对模型的探索．采用 n 步资格迹强化学习，首先在 APP 遍历时构建状态转义模型，遍历的同时更新 n 步价值收益，最后使用 Epsilon-Greedy[38]算法决策(-s 100 --uiautomatormix)
T5	Maxim	混合策略	Maxim 是基于 Monkey 的二次开发，除保留原生 Monkey 已有功能外，可深度遍历控件，可自定义黑白名单，可设定执行时长，增加防睡眠/防假死机制、防跳出/防误点状态栏及下拉状态栏等(--agent robot --running-minutes 30 --throttle 800)

为避免测试的偶然性，扩展测试多样性，参照下载量大小选择真实环境中 17 款不同应用类型的程序作为测试示例．此外，待测应用集不仅包括原应用，还包含经变异算子处理后可以运行的变异应用，共计 1 089 个待测应用作为基本的测试数据集．表 5 为 17 个真实应用的相关信息，我们选择 15 种不同类型的安卓应用，最大程度覆盖市场上的应用类型．为使实验数据更具代表性，选择下载量在 1 万以上的应用作为本实验的待测应用．其中，1 个应用由于系统环境因素导致编译失败，其余 16 个均编译成功．对失败应用进行修复时发现，失败原因是 JVM 堆空间耗尽，最终还是修复失败．然后对 16 个编译成功的应用自动改造 APP，生成变异 APP，从而形成实验的待测应用集．

表 5 待测应用信息

编号	名称	应用类型	应用版本	自动插桩结果	应用下载量
A1	OSMBugs	Development	V1.2.2	成功	1 万+
A2	Meme Tastic	Graphics	v1.6.7	成功	5 万+
A3	AndStatus	Internet	V57.0.7	成功	5 万+
A4	Firefly III Mobile	Money	V3.0.4	成功	5 万+
A5	Binary Eye	Multimedia	V1.33.0	成功	300 万+
A6	DuckDuckGo	Internet	V5.71.0	成功	1 万+
A7	Eventyay Attendee	Connectivity	v0.9.1	成功	1 万+
A8	Calendar	Tool	v6.11.3	成功	75 万+
A9	Gallery	Graphics	V6.19.1	成功	2 万+
A10	GPSTest	Navigation	v3.9.1	成功	279 万+
A11	Streak Alarm	Tool	v1.4.1	成功	1 万+
A12	Kakugo	Education	v1.35.0	成功	5 万+
A13	Authorizer	Security	v0.4.0	成功	50 万+
A14	Notepad	Writing	v2.3.6	成功	1 万+
A15	App Launcher	System	v5.2.0	成功	1 万+
A16	Pretend You’re Xyzzy	Game	v5.0.9	失败	50 万+
A17	HgLauncher	Theme	v1.4.3	成功	1 万+

变异是获得待测应用集的主要方式, 在整个评估模型中占据很大比重. CEAT 评估平台内将 OSMBugs 的变异应用展示应用名称、应用唯一 ID 以及该应用所使用的变异算子等信息进行可视化. 最终, OSMBugs 在选用 5 个变异算子的情况下共生成了 23 个变异应用, 该次测试共生成了 1 073 个变异应用. 为控制测试时间成本, 测试对待测应用进行进一步筛选. 为每个应用(指未变异的应用)的每个变异算子随机选择 2 个应用, 形成一个变异应用集. 由于选择的变异算子生成的变异源码有可能编译不成功, 部分变异算子生成的变异应用也可能不足 2 个, 所以变异应用集有可能不足 10 个. 若不足 10 个时, 则随机选择变异应用, 直到应用集数目补齐至 10 个或所有变异应用均被选中. 本次测试共选择 168 个应用作为我们的案例研究, 包括 16 个原应用以及 152 个变异应用. 在 6 个设备上共执行 5 040 次自动化测试工具的测试任务.

4.3 实验设置

实验环境为双核 CPU,8G RAM, 部署在 MongoDB 4.0 数据库上. 实验使用 Azure 云服务器, 操作系统为 Ubuntu 18.04-LTS 64. 由于自动化测试工具的环境搭建较为困难, 实验选用 Docker 作为服务端的容器, 方便部署和后续更新. Docker 使我们可以在虚拟的容器环境里开发以及部署程序, 意味着程序可以在不同机器上以同样的形态运行, 但不需要额外的配置. 为了获取更多的日志信息, 收集并分析各个指标下的覆盖情况, 借助 Spring 接入的 Log4j2 日志框架来帮助系统进行日志记录以及异常追踪. 此外, JDK 版本为 1.8.0_275, Node 版本 14.16.0-LTS, Appium 版本 1.18.1, ADB (Android debug bridge)版本 1.0.39.

实验选取 F-Droid 上的 17 个真实应用作为实验基础, 对 5 个广泛应用的自动化测试工具(Monkey, Google AppCrawler, App Crawler, Fastbot, Maxim, 详细信息见表 4)进行测试. 根据本文提出的各个指标以及综合评估指标公式, 对测试应用得到的数据分析, 获得测试工具的具体得分. Choudhary 等人当时选择了 60 多个应用进行测试, 但是由于安卓应用迭代较快, 其选择的应用集对当前的应用而言没有普适性. CEAT 评估平台支持动态扩充待测应用集, 用户只需提供源码即可对待测应用集进行扩充, 同时, 可动态对其源码进行变异, 以获得更多的待测应用. 除系统环境因素导致 1 个应用编译失败外, 其他应用均可成功执行变异. 包括变异应用在内, 共生成 1 089 个应用的待测应用集合. 对以 Monkey 为首的 5 款自动化测试工具, 在 6 个设备上共执行 5 040 次自动测试工具的测试任务. 当前, 自动测试工具对自身进行评估大都只使用少量安卓真机, 或直接使用安卓仿真器. 测试并没有在大规模的真机环境下执行, 导致测试结果可能与真实环境中的效果不一致. 由于安卓仿真机和真机环境仍然存在一定的区别, 因此为获得更加真实的测试数据, CEAT 在真机集群上执行待测工具. 实验选择的真实安卓移动手机详细信息见表 6.

表 6 测试安卓手机信息

ID	安卓设备品牌	模型	屏幕大小	版本信息	系统版本
1	SAMSUNG	S20+	5.1	10	OneUI 3.0
2	SAMSUNG	S20	5.7	11	OneUI 2.5
3	HUAWEI	Mate40	6.57	10	EMUI 10
4	小米	NOTE9 PRO	6.67	9	MIUI 11
5	OnePlus	OnePlus2	6.55	8	LineageOS
6	Vivo	IQOONEO5	6.62	7.0	Funtouch OS

4.4 实验结果与分析

(1) RQ1 分析

通过对检索到的 381 篇相关论文指标进行汇总，见表 1，我们发现测试社区在评估工具时选用最多的是代码覆盖率、故障检出率以及变异测试. 于是, CEAT 评估模型将以上 3 项作为基本得分. 此外, UI 控件覆盖率、兼容性以及安全性也占有较高比重. 选择其中的 UI 控件覆盖率以及兼容性融入到 CEAT 综合评估得分. 虽然安全性指标在业界也得到一定的重视，但是安全性在业界进行研究时大多单独考虑. CEAT 的 5 个指标即为是业界普遍使用的评估软件测试技术的指标，体现 CEAT 评估的价值以及普遍适用度.

- 代码覆盖率: 代码覆盖率作为最常使用的一项指标，直观地表示应用中代码被覆盖的程度，用来衡量测试充分性. 高覆盖的测试可以帮助揭示更多软件嵌入的错误，并将海量异构信息量化为覆盖率的值. Brader 提出: 一个测试覆盖率高的程序，与测试覆盖率低的程序相比，未检测软件 Bug 的机率更低[39]. 因此，我们将代码覆盖率作为五大指标之一，用来对自动化测试技术进行评估;
- 异常检出率: 自动化测试工具的主要目标即为暴露现有故障，故障检出就是检查每个工具在一定时间内可以检查出应用程序中有多少故障. 除了运行时异常情况可以发现外，安卓自动化测试工具无法识别故障. 并且故障检测情况可以反映应用现有问题，对于提高应用质量、及时向开发人员提出建设性修改意见意义重大. 因此，我们在参考 Choudhary 的研究成果后，采用原有的异常检出得分作为其中一项指标;
- 多平台工作兼容度能力: 由于操作系统版本以及设备的多样性，安卓碎片化问题越来越尖锐[1]. 碎片化为安卓应用开发造成了极大的隐性开销，因碎片化问题导致的兼容性错误频发，因此碎片化逐渐成为自动化测试工具的一个评估方向[40]. 在开发过程中，相关人员必须不断处理碎片化问题，对硬件以及不同版本进行适配，完成自己的开发. 软件兼容程度的好坏直接影响应用的竞争力，因此，CEAT 增加对机型占有率的评估，比如屏幕大小等. 此外，继承之前研究中的框架版本兼容以及覆盖率情况，最终得到一个新的设备得分，用以评估应用多平台工作的兼容度能力;
- 变异杀死率: 原有异常检出只能对应用中现有的故障进行检出测试，一般已经上市的 App 异常信息有限，限制了自动化测试工具对缺陷检测能力的衡量. 为加强对工具检测缺陷能力的衡量，我们引入了变异测试模拟应用缺陷. 为测试工具检测出某一特定的故障，引入了变异测试，通过改变源码而达到模拟特定缺陷的目的. 系统拥有可扩展的待测应用集，其中包括插桩后的原应用以及对应的变异应用，用户只需提供应用源码方可对应用集进行扩充. 因此，只需提供一个真实环境的应用，通过不同变异算子，可以不断扩充待测应用集. 变异测试可以有效地对测试效果进行衡量，且当前的变异测试发展迅速，变异工具已经十分成熟，已经可以用于工作环境之中. 最终引入变异杀死率作为评估指标，用以评估自动化测试工具检测不同缺陷的能力;
- UI 控件覆盖率: 目前，安卓应用大多是由控件组成，并且分析源码难以获得当前应用中所有控件的信息. 在实际使用过程中，用户更多的体验效果是通过点击 UI 控件反映的，代码上的覆盖并不能真实反映用户真实体验情况. 比如控件上会出现一种遮盖信息，也就是 UI 自动化，可能会碰到某目标控件元素被遮挡的情况. 这种情况下，代码是可以正常运行并被检测，但用户却无法正常使用. 控件覆盖率就是检测安卓应用运行过程中自动化测试工具是否可以触发所有控件的能力. 借助代码覆盖率的思想，我们引入了 UI 控件覆盖率作为 CEAT 扩充指标. 将 UI 控件覆盖率作为一项评估指标，通

过发现自动化测试工具可以点击到的 UI 控件信息, 考察其在 UI 测试中控件覆盖的能力.

5 个指标从不同角度对自动化测试工具进行评估. 其中: UI 控件覆盖与代码覆盖从理念上看是同样的作用, 都是从覆盖面上检测自动化测试工具能够遍历搜索的情况, 但是 UI 控件点击情况可以检测应用功能的完备性, 从用户体验角度出发, 检测工具可以点击到 UI 控件的能力, 因此属于另一维度的评估. 而检测变异的能力是对除应用现有故障外, 模拟特定故障检测能力的补充. 使用不同的变异算子, 不仅可以快速形成测试数据, 并且可以有针对性地对特定缺陷模拟, 检测自动化测试工具对特定逻辑错误检测能力. 考虑到应用载体的多样性, 我们对平台工作能力进行改进. 通过引入 SDK 版本以及屏幕在安卓设备中的占比, 形成新的设备得分, 加上在该设备上的各个评分结果计算, 从而表示自动化测试工具对不同版本设备测试能力的综合评估. 工具代码覆盖情况、用户点击 UI 控件真实使用情况以及故障检测情况综合分析, 对整个自动化测试进行全方位的评估.

(2) RQ2 分析

根据表 2 中分类统计结果, 发现在检索到的 381 篇文章里, 使用单指标评估自动化测试工具的论文占比 89%. 但我们发现: 业界已经有人意识到多指标评估的重要性, 开始结合多个指标用来检验工具或者使用多个指标衡量实验结果. 但是统计结果显示: 涉及双指标仅占比 9%; 三指标占比更低, 仅为 2%. 通过对比多指标评估论文内容以及实验部署, 我们发现: 即使论文涉及多个指标, 也是仅针对单个指标进行实验, 并没有将指标进行融合.

与单指标相比, CEAT 更加客观, 单指标评估存在一定的局限性, 如果不同的测试工具使用不同的指标得到评估结果, 只能从单维度检测自动化测试工具的测试效果. 但 CEAT 从 5 个维度对其进行评估, 涉及代码覆盖率、异常检出能力、融合多版本兼容度、变异杀死率以及 UI 控件覆盖率. 特别是融合多版本兼容度, 增加对机型、屏幕大小等信息完成评估. 最终的综合评分将 5 项指标测试结果融合起来, 兼顾所有指标下的评估得分, 评估结果更加可靠权威. 同时, CEAT 支持测试人员根据客观测试需求对权重灵活分配. 例如: 如果对兼容性问题不加考虑, 即可将相关的设备得分权重降为 0, 仅查看其余 4 项评估得分. 完成测试后, 工具最终评估结果以图形化界面显示, 并支持自行选择单项指标的评估得分.

以测试工具 Maxim 为例, 将其完成 180 个测试任务后的结构以可视化界面展示. 图 2 为 CEAT 评估平台选择异常以及设备得分后所展示的结果. 经由 Echarts 渲染后的结果图, 包括一个箱型图以及一个堆叠条形图. 图中显示本次测试的最终得分情况, 箱型图从左至右分别是该应用的最终得分、UI 控件得分、异常得分、设备得分(多平台工作能力)、代码覆盖率得分以及变异得分. 而右侧图可以选中单个得分项, 对不同应用的得分进行对比.

在单指标评估存在争议时, 多维评估是评估自动化测试工具测试效果权威的代表. 表 7 为实验得到的 5 个不同案例评估结果(案例 T1–T5 即为选择的 5 款自动化测试工具). 从我们的实验结果看, 也有一些单指标评估结果冲突的情况发生. 对于不同的工具, 无法判断哪一个更好. 如在代码覆盖率指标下, 工具表现依次为 T1>T5>T4>T2>T3; 而从异常检出效果看, 表现为 T5>T1>T4>T3>T2. 不同工具测试效果各有侧重点, 对于案例 T1 而言, 测试效果比较顺序是代码覆盖率>兼容度>UI 控件, 而 T2 则是 UI 控件>兼容度>代码覆盖率. CEAT 综合评估融合 5 项指标得分, 综合性评估结果可以给出最可靠的推荐. 将实验结果与 Choudhary 等人的研究进行对比分析, 以 Monkey 测试结果为例, 案例中代码覆盖率和触发的异常数目与 Choudhary 测试的结果吻合, 可以证明 CEAT 的有效性.

在评估时考虑因素越全面, 就越能体现自动化测试工具的综合能力. CEAT 平台为测试人员选择工具提供可靠有效的评估结果, 既可以从单个维度, 也可以从多个维度综合选择. 不仅在评估指标上做到统一, 更是将评估范围扩大. 此外, CEAT 在综合评估上对权重实现动态修改功能, 可根据用户需求对不同维度的权重进行调整, 以区分不同维度在测试中的重要程度. 结合业界普遍认可的指标, 以一种数据可视化的形式, 直观地给出工具评估结果(如图 2 所示).

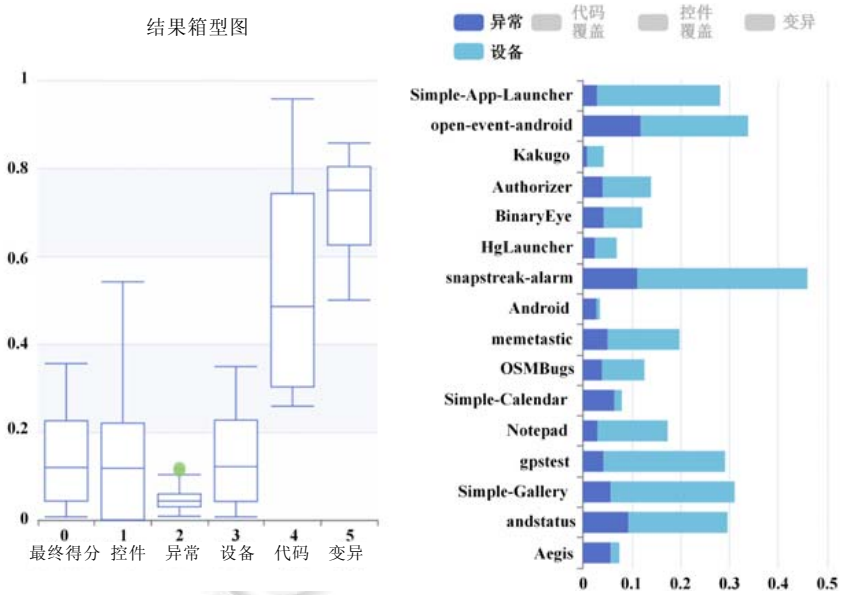

图 2 任务测试结果

表 7 案例测试结果

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

(3) RQ3 分析

Choudhary 等人对自动化测试工具进行了测试, 在代码覆盖率、异常检出、多平台工作能力以及可用性这 4 个方面对工具进行评估. 而我们对其中多平台工作评估进行了改进, 通过引入机型和屏幕等信息形成一个设备得分, 同时增加 UI 控件覆盖率、变异杀死率共 5 个指标对工具进行评估. 对比而言, CEAT 评估更加多维全面. 为显示评估结果的分散情况, 用箱型图表示 CEAT 评估模型下各测试工具综合得分情况, 如图 3 所示. CEAT 的评测维度与 Choudhary 相比有以下几点改进(见表 7): 增加了 UI 控件得分, 从用户体验角度出发,可用于检测工具对 UI 控件覆盖的能力; 增加了变异得分, 通过引入杀死异常的能力, 用于深层次评估工具检测错误的能力, 同时丰富了待测应用集, 节约人力配置的同时, 获得了大量评估结果; CEAT 中的设备得分增加了对机型占有率的评估, 引入了屏幕属性以及当前设备 JDK 版本占比, 形成工具在单个设备上单个应用测试能力的评估, 参照公式(6)–公式(8), 相较传统测试更加客观; Choudhary 于 2015 年以当时的应用为实验对象, 采用 3 种不同 SDK 版本的仿真器(没有物理设备)完成测试; 而 CEAT 选择最新的安卓应用进行实验, 并且在真机集群上完成, 实验结果更加真实可信.

实验结果箱型图以 16 款成功插桩的应用为分类标准, 展示 5 款工具对各应用(包括其变异应用)5 040 轮次的测试评估结果. 如图 3 所示为 5 个自动化测试工具测试在成功插桩的 16 款真实应用(此处用真实应用名字表示其与通过该应用自动变异得到的若干个待测应用)下得到的综合评估结果箱型图. 该图中数据为不同工具最终得分的平均值, 其原始得分由公式(8)计算得到. 可以看出: 在复杂应用中结果较为离散, 且有些得分为 0. 原因是工具对碎片化的支持较差, 兼容度低, 而且因为复杂的应用程序拥有特定触发条件, 模型探索策略的测试工具不能准确识别其状态, 导致测试效果差别较大. 简单应用的测试效果比较集中. 此外, 本文还发现: 采用随机策略的测试工具相较采用模型探索策略的测试工具, 其结果拥有较好的稳定性, 在不同设备上结果相近.

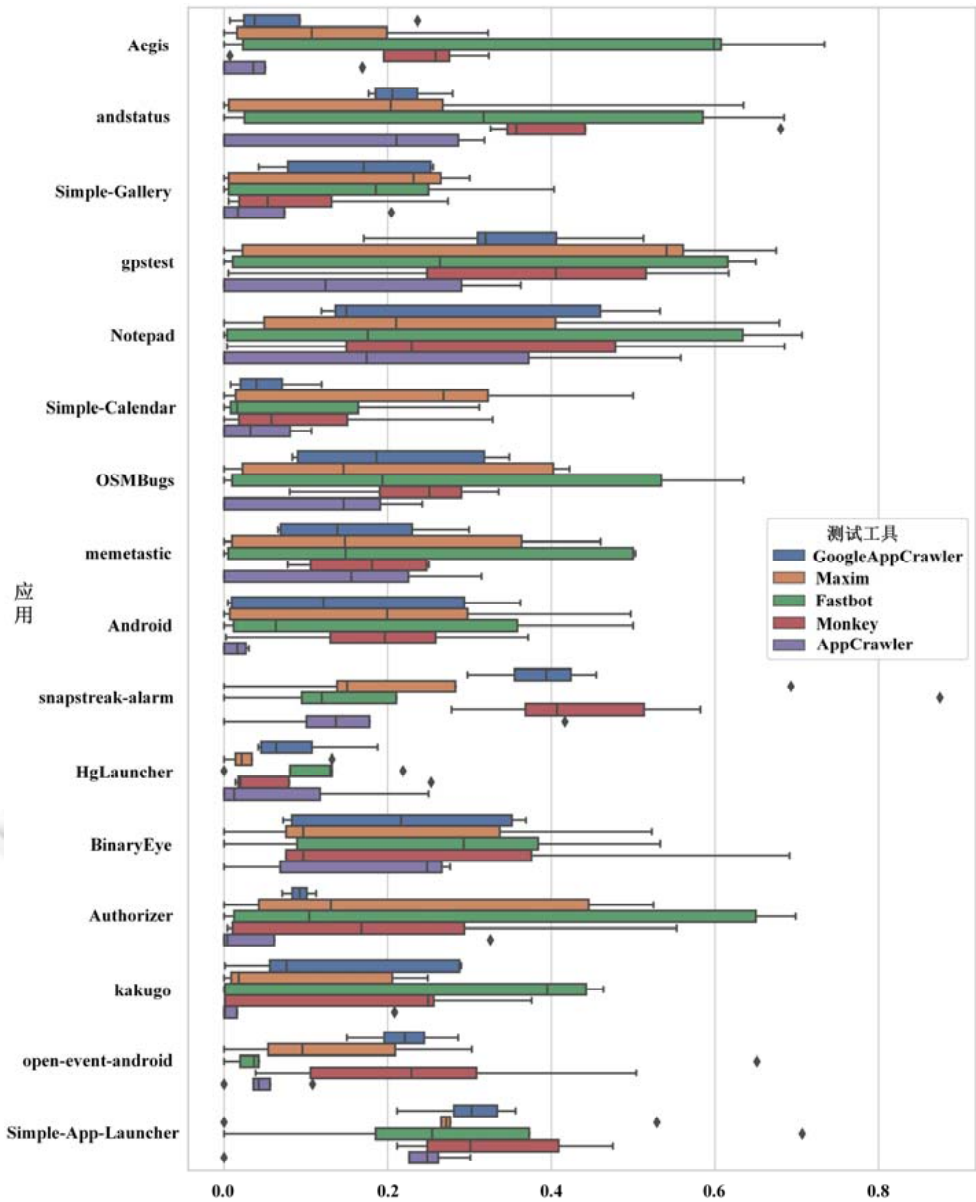

图 3 综合评估结果箱型图

从箱型图数据分布可以看出: Fastbot 测试结果比较分散, 综合评估得分不稳定. 虽然它能获得对安卓应用的最佳测试情况, 但是也可能出现测试效果最差的情况, 即测试效果不稳定. 此外, 在 11 个应用中得到的最佳测试效果都是通过 Fastbot 工具获得的, 也就是说, 使用 Fastbot 可能会获得最佳测试结果. 如果对于一些希望通过多次测试找到最佳综合测试效果的工作人员来说, Fastbot 不失为一个选择. 从平均情况来看, 16 个应用下, 其中, Monkey 评估结果在 10 款应用中平均效果最好, 从中位数可以分析得到这个结果. 并且 Monkey 测试结果比较集中, 箱型图长度普遍较短, 也就是测试比较稳定. 如果追求测试稳定又对其他指标没有特殊要求的情况下, 可以选择 Monkey 工具. 在各指标下, 各工具的短板开始暴露, 优势也各有不同. 但是测试结果很明显, 业界普遍认为测试能力较强的 Monkey, Fastbot 即使指标不一样, 测试效果仍然是最好的, 可以保持较稳定的评估结果.

5 实验结论及威胁分析

5.1 实验结论

(1) CEAT 弥补了单指标评估结果冲突的不足.

从表 7 给出的实验结果看, 5 款自动化测试工具存在单指标冲突的情况. 如在代码覆盖率指标下, T1 所示 Monkey 的测试效果优于 T5 所示的 Maxim; 而从异常检出效果看, 则反之. 从代码覆盖维度来看, 采用随机策略的工具如 Monkey 拥有更为优异的表现, 而基于模型探索策略的工具如 Google AppCrawler 在 UI 控件得分上表现更为优异. 通过日志文件可以发现: 模型探索策略类工具虽然可以检测当前页面状态, 但并不能使页面达到特定的触发条件; 而随机策略类工具会在相同页面停留较长时间, 从而执行更多操作, 更有可能触发特定条件, 因此其代码覆盖维度测试效果较好. 在探索新界面时, 模型探索策略类工具可以将代表事件的结点模拟为流程图用以生成 GUI 测试集, 因此其在 UI 控件维度更有优势. 根据案例 T5 的测试结果显示: 由于混合策略集成了各策略的优势, 因此基于混合策略的工具 Maxim 在多个指标下均有较好的测试效果. 本文从多维度进行评估, 提供统一全面的评估结果, 弥补单指标冲突的不足. 由于不同的自动化测试工具在各维度下测试效果的差异性, CEAT 可根据不同需求为研究人员推荐工具提供参考依据.

(2) 发现不同应用下工具测试效果存在差异.

根据图 3 的箱型图发现, 相同工具在不同应用下测试效果存在较大差异. 如对应用 snapstreak-alarm 进行测试后, Monkey 获得的综合评估效果较好; 而在应用 HgLauncher 下, Fastbot 评估效果较好, Monkey 较差. 一方面是由于自动化测试工具本身的问题, 例如从表 7 中可以看到, T3 兼容度得分较低, 测试效果受到不同的环境等影响; 另一方面则是由于应用本身复杂度不同. 功能复杂的应用有特定的触发条件, 例如滑动、滚动、区域内触发以及控件连接触发等, 基于模型的探索策略受到限制, 不能准确识别其状态造成测试效果差别较大; 而对于功能简单的应用而言, 不同的自动化测试工具的测试效果大致相同. 这一发现方便研究人员根据不同类型应用选择适合工具, 也为未来的研究工作提供了方向.

5.2 威胁分析

获取待测数据集时, 需要重编译变异后的源码. 由于经过变异算子处理后一些应用源码被修改, 从而受到影响, 导致出现大量变异应用 APK 不可运行. 针对此类问题, 我们通过人工审核其可用性, 然后将可运行应用添加到待测应用集. 为获取大量可运行变异应用 APK, 我们借助变异算子对应用进行了大量变异. 考虑到人工成本以及效率问题, 选取 17 个开源应用进行自动插桩并完成变异操作. 除系统环境因素导致 1 个应用编译失败外, 其他应用均成功执行. 包括变异应用在内, 共形成 1 089 个可运行的待测应用集. 待测应用集是从 16 个成功插桩的真实应用变异而来, 并非为真实应用. 测试应用数据数量也受到一定影响, 对实验的有效性存在一定威胁.

此外, 设备的数量以及种类比较少, 并且测试时间比较久. 经调研, 云平台真机测试时间一般最多只能设置 1h, 超时需重新处理. 并且测试一个应用理想状态是 5 min 左右, 因此完成 1 089 个待测应用集, 5 个测试工具需要在每个真机上进行测试, 需要花费大量时间. 由于变异的数量以及种类较多, 完成一个应用测试所花费时间随之增加. 在进行实验时, 本文使用随机策略尽可能选用多种变异算子对待测应用进行变异处理, 获取了丰富的变异应用. 其中存在变异无法运行等情况, 部分变异算子完成的变异应用不足两个, 最终某一应用的变异应用集合不足 10 个. 针对此类情况, 继续随机变异. 实验最终选择 168 个应用, 包括 16 个原始应用和 152 个变异应用. 兼顾不同的变异算子以及各个应用变异 APK 数量, 分别在 6 台真机设备上完成 5 040 轮次测试(5 个自动化测试工具), 有效降低了测试时间.

6 总结与展望

本文提出一种安卓自动化测试工具综合评估 CEAT, 提供包含代码覆盖率、检测真实故障的异常检出率以

及体现软件兼容度的碎片化问题等多个维度的评估指标. 通过 CEAT 提出的评估模型获得代码覆盖得分、异常检出得分、变异得分、UI 控件覆盖得分以及设备得分, 并通过拟合公式获得多维评估结果. 该评估结果可为测试社区解决单指标结果冲突提供思路. 评估模型支持按照测试需求调整 5 个指标在综合评估所占权重, 从而获得个性化综合评估结果, 按照测试重点, 为测试人员推荐最合适的工具. 相较于以往研究, 评估指标更加多维全面, 并且将单指标下的评估结果通过 CEAT 评估模型拟合公式进行融合, 提供更加客观权威的评估结果. 将实验部署在 6 个涉及不同品牌和 SDK 版本的真机上, 以 17 个真实应用为基础, 对 5 款自动化测试工具进行案例分析, 参考 Choudhary 的研究结果, 验证了 CEAT 的有效性. 多维度的评估可以更好地发现自动化测试工具在各个指标下的优势和短板, 从融合机型硬件设备兼容度以及覆盖率、故障检测等综合评估, 为改进自动化测试提供参考依据.

伴随着安卓自动测试产业的发展, CEAT 仍需进行后续的迭代与改善. 本文未来工作重点将放在以下 3 个方向: 通过不断扩充并调整评估指标, 提升测试评估的合理性; 探索效果更优的拟合公式, 从而实现指标更加客观的表达, 致力于完成统一的自动化测试工具评估综合指标; 改进 CEAT 工具的兼容性, 不断引入自动化测试工具进行综合评估, 完成工具之间的横向比较, 为测试人员提供有价值的参考依据.

References:

[1] Zheng W, Tang H, Chen X, *et al*. State-of-the-art survey of compatibility test for Android mobile application. Computer Research and Developmen, 2022, 59(6): 1370−1387 (in Chinese with English abstract).

[2] Segura S, Durn A, Snchez AB, *et al*. Automated metamorphic testing of variability analysis tools. Software Testing Verification & Reliability, 2015, 25(2): 138−163.

[3] Nguyen BN, Robbins B, Banerjee I, *et al*. GUITAR: An innovative tool for automated testing of GUI-driven software. Automated Software Engineering, 2014, 21(1): 65−105.

[4] Joorabchi ME, Mesbah A, Kruchten P. Real challenges in mobile app development. In: Proc. of the IEEE Int'l Symp. on Empirical Software Engineering and Measurement. IEEE, 2013. 15−24.

[5] Machiry A, Tahiliani R, Naik M. Dynodroid: An input generation system for Android apps. In: Proc. of the ESEC/SIGSOFT FSE 2013. 224−234.

[6] Yang W, Prasad MR, Xie T. A grey-box approach for automated GUI-model generation of mobile applications. In: Proc. of the FASE. 2013. 250−265.

[7] Choi W, Necula GC, Sen K. Guided GUI testing of Android apps with minimal restart and approximate learning. In: Proc. of the OOPSLA. 2013. 623−640.

[8] Hao S, Liu B, Nath S, *et al*. PUMA: Programmable UI-automation for large-scale dynamic analysis of mobile apps. In: Proc. of the MobiSys. 2014. 204−217.

[9] Li L, Bissyandé TF, Papadakis M, *et al*. Static analysis of android apps: A systematic literature review. Information and Software Technology, 2017, 88: 67−95.

[10] Xu YR. Design and Implementation of iterative Android application automation test system [Ph.D. Thesis]. Nanjing: Nanjing University, 2020 (in Chinese with English abstract).

[11] Choudhary SR, Gorla A, Orso A. Automated test input generation for Android: Are we there yet? In: Proc. of the ASE. 2015. 429−440.

[12] Huang JF. AppACTS: Mobile app automated compatibility testing service. In: Proc. of the Mobile Cloud. 2014. 85−90.

[13] Qin JW. Research on key technologies for vulnerability detection in the Android platform [Ph.D. Thesis]. Beijing: Beijing University of Posts and Telecommunications, 2021 (in Chinese with English abstract).

[14] Mao K, Harman M, Jia Y. Sapienz: Multi-objective automated testing for Android applications. In: Proc. of the ISSTA. ACM, 2016. 94−105.

[15] Mirzaei N, Bagheri H, Mahmood R, *et al*. SIG-Droid: Automated system input generation for Android applications. In: Proc. of the ISSRE. IEEE, 2015. 461−471.

[16] Palma F, Realista N, Serrão C, *et al*. Automated security testing of Android applications for secure mobile development. In: Proc. of the ICST Workshops. 2020. 222−231.

[17] Salehnamadi N, Alshayban A, Lin JW, *et al*. Latte: Use-case and assistive-service driven automated accessibility testing framework for Android. In: Proc. of ACM CHI Conf. 2021 Online, 2021, 274: 1−11. https://dl.acm.org/doi/10.1145/3411764.3445455

[18] Mahmood R, Mirzaei N, Malek S. EvoDroid: Segmented evolutionary testing of Android apps. In: Proc. of the SIGSOFT FSE. 2014. 599−609.

[19] Romdhana A, Ceccato M, Georgiu GC, *et al*. COSMO: Code coverage made easier for Android. In: Proc. of the ICST. 2021. 417−423.

[20] Barboni M, Morichetta A, Polini A. SuMo: A mutation testing strategy for solidity smart contracts. In: Proc. of the AST@ICSE. 2021. 50−59.

[21] Silva C, Eler MM, Fraser G. A survey on the tool support for the automatic evaluation of mobile accessibility. In: Proc. of the DSAI. 2018. 286−293.

[22] Li JJ. Prioritize code for testing to improve code coverage of complex software. In: Proc. of the ISSRE. 2005. 75−84.

[23] Jiang B, Zhang YY, Chan WK, *et al*. Which factor impacts GUI traversal-based test case generation technique most? A controlled experiment on Android applications. In: Proc. of the QRS. 2017. 21−31.

[24] Tengeri D, Vidács L, Beszédes Á, *et al*. Relating code coverage, mutation score and test suite reducibility to defect density. In: Proc. of the ICST Workshops. 2016. 174−179.

[25] Takahashi J, Kakuda Y. Extended model-based testing toward high code coverage rate. In: Proc. of the Software Quality (ECSQ). 2002. 310−320.

[26] Offutt AJ, Lee A, Rothermel G, *et al*. An experimental determination of sufficient mutant operators. ACM Trans. on Software Engineering and Methodology, 1996, 5(2): 99−118.

[27] Jia Y, Harman M. An analysis and survey of the development of mutation testing. IEEE Trans. on Software Engineering, 2011, 37(5): 649−678.

[28] Saifan AA, Alzyoud AA. Mutation testing to evaluate Android applications. Int'l Journal of Open Source Software and Processes, 2020, 11(1): 23−40.

[29] Moran K, Tufano M, Bernal-Cárdenas C, *et al*. MDroid+: A mutation testing framework for Android. In: Proc. of the ICSE (Companion Volume). 2018. 33−36.

[30] Su T, Wang J, Su ZD. Benchmarking automated GUI testing for Android against real-world bugs. In: Proc. of the ESEC/SIGSOFT FSE. 2021. 119−130.

[31] Lima I, Silva J, Miranda B, *et al*. Exposing bugs in JavaScript engines through test transplantation and differential testing. Software Quality Journal, 2021, 29(1): 129−158.

[32] Chen YH, Li P, Xu J, *et al*. SAVIOR: Towards bug-driven hybrid testing. In: Proc. of the IEEE Symp. on Security and Privacy. 2020. 1580−1596.

[33] Kong PF, Li L, Gao J, *et al*. Automated testing of Android apps: A systematic literature review. IEEE Trans. on Reliability, 2019, 68(1): 45−66.

[34] Nguyen-Vu L, Ahn J, Jung S. Android fragmentation in malware detection. Computers & Security, 2019, 87: 101573.

[35] Han D, Zhang CL, Fan XC, *et al*. Understanding Android fragmentation with topic analysis of vendor-specific bugs. In: Proc. of the 19th Working Conf. on Reverse Engineering. Piscataway: IEEE, 2012. 83−92.

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

[37] Lu X, Liu XZ, Li HR, *et al*. PRADA: Prioritizing Android devices for apps by mining large-scale usage data. In: Proc. of the 38th Int'l Conf. on Software Engineering (ICSE 2016). Piscataway: IEEE, 2016. 3−13.

[38] Tokic M, Palm G. Value-difference based exploration: Adaptive control between epsilon-greedy and softmax. In: Bach J, Edelkamp S, eds. Proc. of the 34th Annual German Conf. on AI (KI 2011). LNCS 7006, Berlin: Springer, 2011. 335−346.

[39] Brader L, Hilliker HF, Wills AC. Testing for Continuous Delivery with Visual Studio 2012. Microsoft, 2012.

[40] Wei LL, Liu YP, Cheung SC. Taming Android fragmentation: Characterizing and detecting compatibility issues for Android apps. In: Proc. of the ASE. 2016. 226−237. http://dx.doi.org/10.1145/2970276.297031

附中文参考文献:

[1] 郑炜, 唐辉, 陈翔, 等. 安卓移动应用兼容性测试综述. 计算机研究与发展, 2022, 59(6): 1370−1387.

[10] 徐悠然. 迭代式安卓应用自动化测试系统的设计与实现 [博士学位论文]. 南京: 南京大学, 2020.

[13] 秦佳伟. 安卓平台的漏洞检测关键技术研究 [博士学位论文]. 北京: 北京邮电大学, 2021.

钟怡(1987－), 女, 博士生, 主要研究领域为移动应用测试.

石孟雨(1998－), 女, 硕士生, 主要研究领域为移动应用测试.

房春荣(1986－), 男, 博士, CCF 专业会员, 主要研究领域为代码大数据, 软件测试.

赵志宏(1975－), 男, 博士, 教授, 博士生导师, 主要研究领域为信息系统工程.

陈振宇(1978－), 男, 博士, 教授, 博士生导师, CCF 高级会员, 主要研究领域为智能软件工程.