软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2020,31(12):3671−3684 [doi: 10.13328/j.cnki.jos.005873]

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

一种手绘制导的移动应用界面测试方法*

成浩亮[1,2], 汤恩义[1,2], 玉淳舟[1,2], 张初成[1,2], 陈 鑫[1,3], 王林章[1,3], 卜 磊[1,3], 李宣东[1,3]

[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(南京大学 软件学院,江苏 南京 210093)

3(南京大学 计算机科学与技术系,江苏 南京 210023)

通讯作者: 汤恩义, E-mail: eytang@nju.edu.cn

摘 要: 软件测试在提高移动应用的安全性和可靠性方面扮演着重要角色.然而,目前主流的移动应用界面测试技术存在着许多不足:人工编写脚本和录制回放技术需要消耗大量的人力成本,自动化测试在移动应用界面测试的应用场景上受到了诸多限制.针对这些问题,提出一种基于手绘制导的移动应用界面测试方法.该方法通过设计一种简单直观且具有较强表达能力的手绘语言来帮助测试者轻松表达其测试意图,测试者仅需在待测应用的界面图像上做简单绘制,就能生成对应的测试模型,并以此为基础生成界面测试所需的测试用例.以近年来在相关文献中已经用作移动应用界面测试的评估用例集为基准来评估该方法的测试效果.评估结果表明:在提供很少人力成本的条件下,手绘图形所表达的用户测试意图在制导移动应用界面测试上能起到非常关键的作用.

关键词: 手绘制导测试生成;模型驱动的自动化测试;移动应用;图形界面测试

中图法分类号: TP311

中文引用格式: 成浩亮,汤恩义,玉淳舟,张初成,陈鑫,王林章,卜磊,李宣东.一种手绘制导的移动应用界面测试方法.软件学报, 2020,31(12):3671−3684. http://www.jos.org.cn/1000-9825/5873.htm

英文引用格式: Cheng HL, Tang EY, Yu CZ, Zhang CC, Chen X, Wang LZ, Bu L, Li XD. Approach of sketch-guided GUI testing for mobile app. Ruan Jian Xue Bao/Journal of Software, 2020,31(12):3671−3684 (in Chinese). http://www.jos.org.cn/1000-9825/5873.htm

Approach of Sketch-guided GUI Testing for Mobile App

CHENG Hao-Liang[1,2], TANG En-Yi[1,2], YU Chun-Zhou[1,2], ZHANG Chu-Cheng[1,2], CHEN Xin[1,3], WANG Lin-Zhang[1,3], BU Lei[1,3], LI Xuan-Dong[1,3]

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

[2](Software Institute, Nanjing University, Nanjing 210093, China)

[3](Department of Computer Science and Technology, Nanjing University, Nanjing 210023, China)

Abstract: Software testing plays an important role in improving the security and reliability of mobile applications. However, there are many shortcomings in the current mainstream testing technology for mobile application. For example, manual scripting and recording playback techniques require a lot of labor costs, while automatic testing solutions have been limited when they are applied in mobile applications. This article presents a sketch-guided GUI test generation approach for testing mobile applications. It provides a simple but expressive sketch language to help testers specify their testing purposes easily. Testers just need to draw a few simple strokes on the corresponding screenshots of the application under test, then the proposed approach recognizes these strokes automatically and converts them into the test model. Finally, test cases are generated based on the model. The effects of proposed sketch-guided testing technique are

* 基金项目: 国家重点研发计划(2017YFB1001801); 国家自然科学基金(61772260, 61402222, 61632015, 61690204)

Foundation item: National Key Research and Development Program of China (2017YFB1001801); National Natural Science Foundation of China (61772260, 61402222, 61632015, 61690204)

收稿时间: 2018-12-11; 修改时间: 2019-05-08; 采用时间: 2019-07-08

evaluated based on the test cases, which were used in the literature of mobile application testing in recent years. The results show that the proposed approach is significantly effective with little labor cost in GUI testing of mobile applications.

Key words: sketch guided test generation; model based test automation; mobile applications; graphical user interface testing

随着移动设备的普及,移动应用程序获得了广泛使用,截至 2019 年 6 月,Google Play 上的移动应用已超过 270 万[1].目前,这些移动应用程序已普及到教育、医疗、金融、通信等各行各业,深刻影响着人们的生产和生活.如何对这些应用程序进行有效测试,以保障其安全性和可靠性,是软件工程领域的一个重要研究课题.

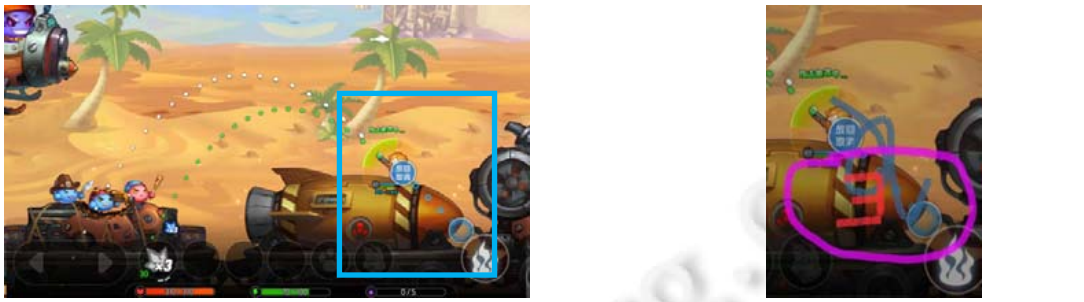
现有移动应用程序常以丰富的图形界面元素作为其人机交互基础,以达到使用简便直观的目的.但与此同时,丰富的图形界面元素增加了软件测试的难度.现有的图形界面测试方法以测试脚本为载体来描述应用程序运行过程中图形界面所接收的事件序列.按照脚本的生成方式,主要分为两类:第 1 类方法以人工直接编写测试脚本来完成图形界面的软件测试,该方法对测试人员的专业性要求较高,且在测试脚本数量较多的情况下需要大量的人力成本;第 2 类方法采用录制回放的方式,在测试者使用软件的过程中记录下图形界面的事件序列,并以此来构建测试脚本,用于对应软件事件序列的回放测试.尽管该方法降低了对测试人员的专业性要求,但由于每次应用程序的执行过程仅能录制与回放程序中的一条执行路径,测试者须反复运行与录制该应用程序,以达到较高的测试覆盖率.对于现实中界面复杂的应用程序测试,这样的录制过程仍然需要大量的人力成本.

近年来,为了减轻测试人员在测试过程中繁重的手工操作,研究人员提出了多种针对图形化界面的测试用例自动生成技术[2−5].这些技术本质上都在试图自动探索待测应用程序中所有可能的事件组合,并记录相应的测试事件序列.但由于实际应用程序的图形化界面往往层次较多且逻辑复杂,所以其相应的搜索空间很大,相应的测试事件序列生成会存在组合爆炸等问题.以 Memon 等学者[6,7]提出的模型驱动图形界面自动化测试生成方法为例,它首先根据待测应用程序来构建一张事件流图(event flow graph,简称 EFG),再通过遍历图上的所有可达事件路径生成相应的测试脚本.而在实际运行过程中,对于搜索空间较大的复杂应用程序,构建事件流图模型的过程非常耗时,且由于生成的事件流图复杂,其组合路径和对应的事件序列数量巨大.因此,根据用户意图提供一种削减其搜索空间而获得有效测试用例的方法在这一背景下显得非常必要.Choi 等学者[5]进一步提出了一种基于机器学习算法的图形界面自动化测试生成技术,称为 SwiftHand,该方案以尽可能少的程序重启执行次数作为机器学习的目标,以达到在短时间内实现较高测试覆盖这一目的.然而,有些程序错误可能会隐藏在复杂程序界面的组合逻辑中,仅仅从测试用例搜索空间中获取程序重启次数少的测试脚本,并不一定能完全满足移动应用程序的界面测试要求.

本文通过提出一种基于手绘的图形界面测试用例生成方法来解决这些问题.在本文中,我们定义了一种简单直观且具有较强表达能力的手绘测试语言.测试人员仅需按照该语言定义的语法,在待测应用程序的界面图像上做简单绘制,即可准确表达其测试意图.本文框架会将这些绘制的元素图形自动转换成测试模型,并高效生成符合要求的测试用例.由于用户的手绘图形明确定义了当前的测试意图,因此本文方法不需要对所有可能的事件组合进行遍历,从而使得生成的测试用例更为有效且具有针对性.

图 1 展示了手机游戏弹弹堂的一个测试场景示例.在该场景下,用户希望右侧导弹以不同的角度和力度打出时,测试对应的界面元素是否存在错误.尽管该场景的界面动作并不复杂,但不同的角度和力度的组合数量很多,手工测试方法难以满足该场景的测试要求.全自动的测试脚本生成由于缺少用户的测试意图信息,会尝试遍历界面中所有控件的可能事件组合(例如背景树叶拖动事件、火车上的其他界面控件等等),从而造成搜索空间过大,难以完成有效测试.本文定义了一种简单直观的手绘测试语言来帮助测试者表达其测试意图.在该测试场景下,用户仅需寥寥几笔绘制 3 个元素图形(如图 1(b)所示),即可完整表达其测试意图.首先,用户将绘制一个蓝色的动作元素用于声明一个拖拽动作,表示当前希望测试导弹的拖拽;其次,用户绘制一个紫色的范围元素,表示拖拽动作的终止位置将在紫色区域选定的范围内;最终,用户将绘制一个红色的存在量词,表示测试是否存在不满足测试目标的动作序列.一般来说,用户需要额外指定本次测试的测试目标.如不指定,默认测试目标为是否发生软件错误(如程序崩溃等).本文框架将用户手绘的图形转换成抽象事件流图(abstract event flow graph,简称 AEFG).作为测试模型,抽象事件流图中刻画了满足测试意图的所有动作序列,在本示例中即为满足范围要求

的拖拽动作集.最终,我们的框架将动作序列从模型中导出,生成用户需要的界面测试用例.



(a) 弹弹堂游戏的测试界面 (b) 测试导弹在不同角度和力度发射的手绘图形

Fig.1 A test scenario example of bouncing ball

图 1 一个弹弹堂游戏的测试场景示例

在本文中,我们以近年来在相关文献中已用作移动应用界面测试的评估用例集为基准,来评估手绘图形界面测试方法的测试效果.评估结果表明:对于大多数移动应用程序,用户仅需进行 8 分钟左右手绘人力成本,就能使界面测试效果得到明显提升,手绘图形所表达的用户测试意图在制导移动应用界面测试上能起到非常关键的作用.

1 手绘制导测试框架

本文的应用程序界面测试框架主要包含两个部分:手绘前端用于帮助用户在待测应用界面图像上绘制图形,并按照手绘测试语言的语法进行识别,使之能准确刻画用户的测试意图;测试生成后端用于在此基础上建立测试模型并生成对应的测试用例.在实施测试时,本文框架还提供了交互式反馈机制,即,用户可以根据生成用例的测试效果,进一步补充新的手绘图形,使对应的测试意图更为准确.

1.1 手绘前端

手绘前端用于识别测试人员在应用界面图像上的手绘图形,然后根据识别结果生成布局文件.布局文件中包含了按照手绘测试语言所定义的各项参数,以及对应界面的控件信息.为了方便测试人员进行手绘并规范测试人员的手绘图形,本文的手绘测试语言定义了 4 种基本元素图形,分别是:(1) 用于表示各种基本测试动作的动作元素图形;(2) 用于申明动作元素作用范围的范围元素图形;(3) 用于表示动作范围约束的量词元素图形;(4) 用于以一定逻辑关系连接基本动作步的逻辑连接符元素图形.

动作元素图形定义了测试中的单个动作,常见的测试动作有单击、双击(在某些设备上相当于长按)、拖拽等等(如图 2 所示),用户可以根据实际需要对动作元素图形进行扩展.

(a) 单击 (b) 双击或长按 (c) 拖拽

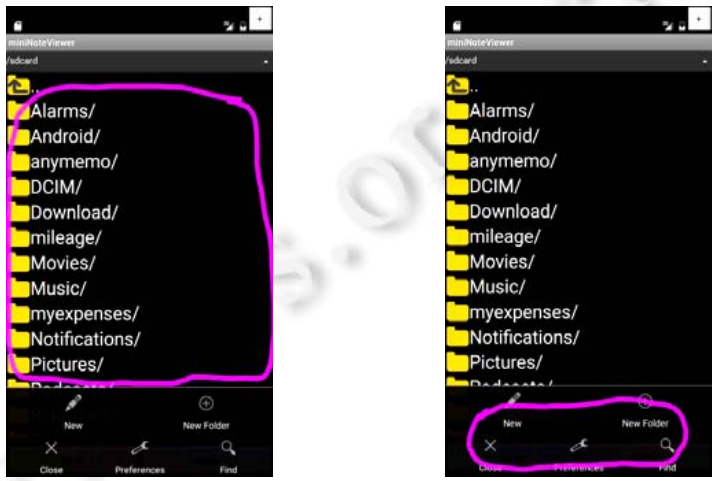
Fig.2 Example of action elements

图 2 动作元素图形示例

另外,动作元素还需定义必要的参数,包括触发这个动作的触发点坐标,这个坐标就是图 2 中每个动作元素近似圆的部分的中心点坐标(对于双击元素而言,只取第 1 个近似圆的部分的中心点),以及这个坐标对应的控件

信息.例如对于图 2(c)的拖拽动作元素,它定义的参数包括拖拽起点、拖拽终点以及位于拖拽起点上的控件的信息.

范围元素图形用于申明测试动作的作用范围.它一般由一个封闭的圈状结构构成(如图 3 所示),在圈范围内及位于圈轨迹上的所有控件都是测试动作的作用范围.例如在图 3(b)中,范围元素圈出了菜单项中的 5 个按钮,表明动作元素将会作用于这 5 个按钮上.在一些难以获得控件信息的游戏类应用中,范围元素将直接圈出测试动作的作用像素范围,此时将以像素坐标作为测试基准.



(a) (b)

Fig.3 Example of range elements

图 3 范围元素图形示例

量词元素图形须与范围元素图形同时使用,以表示测试动作在该范围内的数量约束.在本文的手绘测试语言中,我们定义了 4 种典型的量词元素,包括存在量词、全称量词、递归存在量词以及递归全称量词,其对应的手绘量词元素图形分别如图 4(a)~图 4(d)所示.

(a) 存在量词 (b) 全称量词 (c) 递归存在量词 (d) 递归全称量词

Fig.4 Sketch of quantifier elements

图 4 量词元素图形

存在量词表示仅需在该范围内找到一个满足测试目标的测试动作,全称量词表示须反复验证是否该范围内的动作均能满足测试目标.为了加强量词的表达能力,在本文的手绘测试语言中,我们进一步引入了递归存在量词和递归全称量词.这些递归量词以对应范围元素所圈定的测试动作作为起始动作,递归地探索更深层次的动作序列组合,如果由该动作开始的所有动作序列均满足测试目标,那么它符合递归全称量词;同理,如果至少存在一个由该动作开始的动作序列满足测试目标,那么它就符合递归存在量词.以图 3(a)为例,假设测试人员绘制的是单击元素和递归存在量词,并以图 3(a)中的范围元素表明单击元素的作用范围,这就代表测试人员希望

单击范围内的每个列表项,并递归地单击新出现的界面中随机元素,直到能够产生一个满足测试目标的事件.

为了进一步简化用户的绘制过程,本文的手绘测试语言进一步引入了逻辑连接符元素图形,它们主要用于按照一定的逻辑连接动作集合,以生成更为复杂的逻辑动作步.本文的手绘测试语言定义了 3 种逻辑连接符,分别是逻辑连接符“与”、“或”和“非”,其元素图形如图 5(a)~图 5(c)所示.逻辑连接符“与”所连接的多个动作必须同时触发;逻辑连接符“或”所连接的动作只需其中之一能够满足测试目标就可以了;而逻辑连接符“非”用于从动作序列集中扣除特定的测试动作序列.

(a) 逻辑连接符“与” (b) 逻辑连接符“或” (c) 逻辑连接符“非”

Fig.5 Sketch of connector elements

图 5 逻辑连接符元素图形

在手绘元素基础上,本文进一步基于上下文无关文法来精确定义手绘语言的语法,具体如下:

手绘图 → 动作序列集
动作序列集 → 动作序列 | 动作序列集,动作序列
动作序列 → 逻辑动作步| 动作序列;逻辑动作步
| 动作序列:动作序列集
逻辑动作步 → 基本动作步 | 基本动作步 + **逻辑连接符元素** + 逻辑动作步
基本动作步 → **动作元素**| **动作元素** + **范围元素** + **量词元素**

Fig.6 Grammar rulers of sketching language

图 6 手绘测试语言的语法规则

测试人员在待测应用的界面截图上首先绘制一个动作元素来构建基本动作步,这样的基本动作步只有一个动作.他们也可以绘制动作+范围+量词来构成基本动作步,此时的基本动作步由一个可能发生的测试动作集来定义.测试人员可以进一步将基本动作步用逻辑连接符元素图形连接起来,形成逻辑动作步.一个逻辑动作步定义了测试人员在待测应用的一个特定状态下希望进行的测试步骤.待测应用程序在经过这样的一个逻辑动作步之后会跳转到下一执行状态,因此,多个逻辑动作步由‘;’依次衔接即构成一个动作序列,而多个动作序列进一步形成动作序列集,并最终成为用户表达其测试意图的手绘图.

在测试语言的语法中,‘+’‘;’‘,’‘:’分别表示不同含义的衔接符,它们由测试平台支持的用户手势或测试界面组件来实现.用户绘制完一个图形后,通过手势或点击一个衔接符来连接下一个手绘图形,以表达手绘图形间的关系.这里:符号‘+’用于衔接同一个动作步(包括基本动作步和逻辑动作步)内的手绘图形;符号‘;’用于衔接同一个动作序列内的多个动作步;符号‘,’用于衔接整个测试集中的不同动作序列;符号‘:’表示分叉,当有一批测试动作序列以相同的子序列开头时,用户可先绘制这一共有的子序列,再由‘:’分叉出各动作序列后续的不同部分.

例如,当用户以“动作元素+范围元素”定义 A_1,A_2 等测试动作集时,用户可以按照测试语言的语法绘制如下序列:

$$A_1+\forall\neg a_1;a_2;\ldots;A_2+\forall \tag{1}$$

由于 A_1,A_2 由“动作元素+范围元素”定义,$A_1+\forall$ 和 $A_2+\forall$ 代表了由动作元素、范围元素和全称量词元素$\forall$定义的基本动作步;a_1,a_2 分别表示只由一个动作元素构成的基本动作步;‘¬’代表了逻辑连接符“非”,它将逻辑动作步 $A_1+\forall$ 和基本动作步 a_1 连接成一个逻辑动作步;最后,符号‘;’衔接了该动作序列中的各逻辑动作步.

接下来,我们描述该测试语言的语义.手绘图形在语义上定义了测试所需覆盖的所有动作序列所构成的动作序列集,该集合是以下列方式来定义的.

- 首先,在语法上,由分号或者冒号分隔的各个逻辑动作步在语义上定义了动作序列中的一个动作.为了便于区分测试语言的语法元素,我们假定实际测试的动作序列为 $s_1;s_2;s_3;...;s_n$,其中,s_i 是该次测试序列中的第 i 个动作;
- 其次,对于每一个逻辑动作步,由手绘语法上的"动作元素+范围元素"在语义定义动作集(例如公式(1)中动作集的 A_1,A_2),由量词定义基本动作步内各动作集的数量关系,由逻辑连接符定义基本动作步之间的逻辑关系.

最终,公式(1)所定义的动作序列集如下:

$$\{s_1;s_2;s_3;...;s_n|(\forall s_1\in A_1\wedge s_1\neq a_1)\wedge(s_2=a_2)\wedge\ldots\wedge(\forall s_n\in A_2)\} \tag{2}$$

用户在按照测试语言语法完成手绘图形后,本文框架的手绘前端会将图形记录在布局文件 L 内,并将其传递给后端生成测试用例.布局文件 L 中记录了已经经过形状识别并按照图 6 语法建立的图形描述序列 l、测试目标 O_e 以及待测程序的界面控件信息 W.这里的 O_e 和 W 都以映射的形式来存储:O_e 将语法上的每一条序列映射到一个测试目标判断条件上,而 W 则将程序界面上的坐标值映射到界面控件的标识符上.

1.2 测试生成后端

测试生成后端由手绘前端输出的布局文件来构建对应测试模型,并依据模型产生测试脚本.从框架的实用性考虑,我们希望构建对于大型软件来说也足够简单的测试模型,模型中仅包含满足用户测试意图的动作序列.本文设计抽象事件流图模型来满足这些需求.与已有的事件流图模型[6,7]中每一个节点都是一个具体测试动作不同,抽象事件流图模型中的每一个节点 v 代表一个逻辑动作步.由于模型中的逻辑动作步表示一个经过抽象的事件集,因此该模型在表达形式上更为简洁.在抽象事件流图模型中,连接两个不同抽象事件节点 v_1,v_2 的边表示事件的连续触发关系.具体来说,当模型中存在边(v_1,v_2)时,用户希望能够测试节点 v_2 所定义的任意事件 e_2 紧跟着节点 v_1 对应事件 e_1 的触发序列.我们将抽象事件流图 G 形式化定义为一个四元组$\langle V,E,V_0,O\rangle$,其中,

1. V 代表抽象事件流图 G 的节点集,其中的每个节点 $v\in V$ 都是由测试者手绘生成的一个逻辑动作步构成的;
2. $E\subseteq V*V$ 代表抽象事件节点之间的有向边集.对于边$(v_1,v_2)\in V$ 而言,且$\forall e_1\in v_1,\forall e_2\in v_2$,则事件 e_2 是可以在事件 e_1 发生之后立即触发.且作为模型的一部分,用户希望先后发生这两个事件的序列能及时被测试到;
3. $V_0\subseteq V$ 代表起始节点集,即允许测试该应用的第 1 个逻辑动作步所构成的集合;
4. O 是一个代表测试目标(oracle)的映射.它将抽象事件流图 G 中的每一条事件序列映射到一个用于判断测试目标是否达成的判定条件上:如果测试完该序列后判定条件得到满足,则测试通过;否则,测试不通过.如果有抽象事件流图中的事件序列未在 O 中定义,则对应的判定条件为缺省条件,即程序是否崩溃.

算法 1 描述了本文从手绘布局文件 L 中构建抽象事件流图模型 G 的过程.手绘布局文件 L 由三元组$\langle l,W,O_e\rangle$定义,其中:l 是按照图 6 的语法定义的图形描述序列;W 保存了每一个界面控件的坐标信息,它将坐标值映射到控件标识上;而 O_e 记录了用户完成手绘过程中定义的测试目标,即,将 l 定义的描述图形序列映射到判断测试目标是否达成的判定条件上.

算法 1. 抽象事件流图的建模算法.

输入:$L\langle l,W,O_e\rangle$;

输出:$G\langle V,E,V_0,O\rangle$.

```
1:  while s∈logicStep(L.l) do
2:      t_v←vertexGenerate(s)
3:      while p_t∈pointCoodinate(t_v) do
```

4: $t_v \leftarrow t_v[p_t \mapsto L.W(p_t)]$ //将坐标点转化为对应的控件
5: **end while**
6: $G.V \leftarrow G.V \cup \{t_v\}$
7: $V(s) \leftarrow t_v$ //缓存,以便从动作步 s 找到图中的对应节点
8: **end while**
9: **while** $(s_1;s_2) \in subSequence(L.l)$ **do**
10: $G.E \leftarrow G.E \cup \{(V(s_1),V(s_2))\}$
11: **end while**
12: **while** $(et:etset) \in subSequence(L.l) \wedge et' \in etset$ **do**
13: $s_1 \leftarrow lastStep(et)$ //找出动作序列 et 中的最后一个逻辑动作步
14: $s_2 \leftarrow firstStep(et')$ //找出动作序列 et' 中的第 1 个逻辑动作步
15: $G.E \leftarrow G.E \cup \{(V(s_1),V(s_2))\}$ //将这两个逻辑动作步连接形成的边加入到边集中
16: **end while**
17: **while** $et \in eventTrace(L.l)$ **do**
18: $s \leftarrow firstStep(et)$
19: $G.V_0 \leftarrow G.V_0 \cup \{V(s)\}$ //初始动作步对应节点加入到抽象事件流图的起始节点集 $G.V_0$
20: **end while**
21: $G.O \leftarrow oracleExtract(L,V)$ //提取 L 中的测试目标
22: **return** G

算法 1 由布局文件 L 中记录的信息构建抽象事件流图 G,其中,第 1 行~第 8 行构造抽象事件流图的节点集 $G.V$.第 1 行的 *logicStep* 函数依次从图形序列中获得逻辑动作步 s,并由函数 *vertexGenerate* 生成抽象事件流图的节点 t_v.在第 3 行~第 5 行将节点中记录的坐标转为控件信息后,由第 6 行加入节点集 $G.V$.第 7 行临时记录了映射 V,它将逻辑动作步 s 映射到对应的节点上.第 9 行~第 16 行进一步构造抽象事件流图的边集 $G.E$.函数 *subSequence*(·)的作用是将‘,’相连的动作序列集映射为单个的动作序列.第 9 行~第 11 行将 l 中绘制的动作步衔接(即由‘;’相连的动作步)加入到抽象事件流图的边集 $G.E$ 中,第 12 行~第 16 行进一步将 l 中绘制的动作步分叉(即由‘:’相连的动作序列和动作序列集)加入到抽象事件流图的边集 $G.E$ 中.第 17 行~第 20 行进一步将 l 中每一个初始动作步的对应节点加入到抽象事件流图的起始节点集 $G.V_0$.最终在第 21 行由 *oracleExtract* 函数来构造抽象事件流图 G 中的测试目标映射 $G.O$.

算法 2 描述了 *oracleExtract* 函数的内部细节,该函数将布局文件记录的测试目标 $L.O_e$ 解析并提取出来,找到其中每一个判定条件在模型 G 中的对应路径,并将其对应关系写入到 $G.O$ 中.

该算法通过第 1 行~第 17 行的大循环,每次处理布局文件 $L.l$ 中按测试语言语法定义的一条事件序列 et,将该序列在模型 G 中所有对应路径形成的路径集 P 找到(第 2 行~第 14 行),并在第 15 行将 $L.O_e$ 中关于 et 的判定条件写入到 O 中对应的每一条路径上.

算法 2. *oracleExtract* 函数:测试目标提取算法.

输入:$L\langle l,W,O_e\rangle,V$;
输出:O //测试目标.

1: **while** $et \in eventTrace(L.l)$ **do**
2: $p \leftarrow V(firstStep(et))$
3: $P \leftarrow P \cup \{p\}$ //路径 p 的起始节点加入到集合 P 中
4: **while** $p \in P$ **do**
5: **if** $(s_1;s_2) \in subSequence(et) \wedge V(s_1) == lastVertex(p)$ **then**
6: $p \leftarrow addVertex(p,V(s_2))$ //将 s_2 对应节点追加到路径 p 的末尾

```
      else if (et′:etset)∈subSequence(et)∧V(lastStep(et′))==lastVertex(p) then
        P←P−{p}
        while et″∈etset do
          p′←addVertex(p,V(firstStep(et″)))
          P←P∪{p′}
        end while
      end if
    end while
    ∀p∈P,O(p)←L.O_e(et)
    P←∅
end while
return O
```

在完成抽象事件流图模型 G 的构建后,测试生成后端将按照模型驱动的界面测试生成算法来生成用于界面测试的用例.即:对模型 G 做完整遍历或者等概率均匀分布的随机路径遍历,并由遍历中的每一条路径生成一个用于界面测试的运行脚本,最终由界面测试框架来执行脚本和比对测试结果.这一过程与已有的模型驱动测试生成方法[8−10]类似,因此这里不再进一步详细赘述了.

在运行脚本的过程中,测试人员可能会发现真实的测试运行效果与期望效果不符或者测试人员产生了新的测试需求,因此在测试生成后端,引入了交互式反馈来帮助用户完善测试.在测试人员手绘过程中,模型生成和脚本生成时,各自额外维护一份日志文件.在手绘时,日志文件不仅简单明了地记录测试人员在每个状态下绘制的逻辑动作步,还记录了测试人员绘制的各个逻辑动作步的次序.这就提供了充分的信息便于测试人员快速定位在哪个逻辑动作步出现了问题.在建模过程中维护的日志文件,主要记录了模型的各个节点与布局文件中语法符号的对应关系.测试脚本生成过程中维护的日志信息则记录了每一条测试语句与模型的各个节点的对应关系.

2 实验评估

在已有的移动应用界面测试文献中,Choi[5]和 Choudhary[11]构建了一套基准评估用例集,用来评估不同移动应用界面测试方法的实施效果.本文亦以该用例集为基准来评估我们手绘界面测试方法的效果.该用例集的应用程序全部来自于 F-Droid 开源应用市场,具体信息见表 1,其中,最小的应用程序为 music note,含 1 345 条字节码指令和 245 个程序分支;最大的应用程序为 anymemo,含 72 145 条字节码指令和 4 954 个程序分支.

Table 1 Details of benchmark apps

表 1 基准用例集的详细信息

应用名称	类别	字节码指令数	方法数量	分支数量
musicnote	education game	1 345	72	245
whohas	lent item tracker	2 366	146	464
explorer	JVM status lookup	6 034	252	885
weight	weight tracker	4 283	222	813
tippy	tip calculator	4 475	303	1 090
myexpense	finance manager	11 472	482	1 948
mininote	text viewer	13 892	680	2 489
mileage	car management	44 631	2 583	3 761
anymemo	falsh card	72 145	832	4 954
sanity	system management	21 946	1 415	5 723

通过与已知移动应用界面测试技术进行对比实验,本文将试图回答以下 3 个重要的研究问题:

问题 1:手绘所表达的测试意图对提升测试效果能起到多大作用?

问题 2:使用本文框架进行移动应用程序的测试时,需要多少人力?

问题 3:测试语言中的逻辑连接符和测试框架的交互式反馈对测试有多大帮助?

针对这 3 个研究问题,我们设计了 3 组不同实验.实验的手绘前端运行于 Android 23.0.0 平台,测试生成后端运行于一台 13 英寸的 MacBook Pro,CPU 是 Intel Core i5 2.7GHz,内存为 8G.我们招募了 12 名志愿者来完成手绘测试的对比实验.其中有 6 名志愿者有一定的软件测试经验,而另外 6 名为没有经验的在校学生.我们对志愿者进行了统一培训,保证其能理解并熟悉我们的手绘测试框架,并在实验时对他们进行了两两分组,每组由一位有经验的测试人员和一位在校学生组成.

2.1 研究问题1:手绘的有效性

在第 1 个实验中,我们对比了文献中移动应用界面测试的现有前沿技术和本文方法的测试效果,我们以 30 分钟时长内达到的测试覆盖率为评价指标来进行评估.我们对比了 onkey[12],AndroidRipper[8],MobiGUITAR[9]以及 SwiftHand[13]这 4 种前沿的测试框架,其中, Monkey 是目前最流行的随机测试框架,AndroidRipper 和 MobiGUITAR 是前沿的基于模型移动测试框架,而 SwiftHand 是最新的机器学习制导的移动测试框架.对于我们的手绘制导方案,这 30 分钟测试时长中的 10 分钟用于手绘(包括 8 分钟初始手绘和 2 分钟交互式反馈),20 分钟用于框架运行,对于用于对比的自动化测试框架,对应的 30 分钟全部用于框架运行.

表 2 给出了本文方法与对比方法在基准用例集上的实验结果,由于对比方法实现接口和部署依赖的限制,我们在 Monkey,AndroidRipper 和 MobiGUITAR 上收集了语句覆盖率,而在 SwiftHand 上收集了分支覆盖率.对于本文手绘制导的测试方法,我们同时收集了语句覆盖率和分支覆盖率.从测试结果观察:本文手绘制导的移动应用测试方法在语句覆盖率上平均比 Monkey 高出 18.86 个百分点,比 AndroidRipper 高出 55.90 个百分点,比 MobiGUITAR 也高出了 48.41 个百分点.在分支覆盖率上,本文手绘制导的移动应用测试方法比 SwiftHand 平均高出 6.12 个百分点.由此可知,手绘制导在移动应用界面的测试效果上能够起到非常关键的作用.

Table 2 Coverage of sketch-guided testing and other techniques

表 2 本文手绘制导测试方法与对比方法在基准用例集上的测试覆盖率

待测应用	语句覆盖率(%)				分支覆盖率(%)	
	Monkey	AndroidRipper	MobiGUITAR	手绘制导	SwiftHand	手绘制导
musicnote	46.85	4.01	29.83	84.12	62.30	73.63
whohas	59.35	16.31	32.89	78.35	54.40	61.39
explorer	69.36	2.52	16.49	69.40	62.94	53.42
weight	45.07	14.97	17.32	70.06	51.45	62.02
tippy	78.62	7.68	13.35	88.97	61.50	65.82
myexpense	42.57	12.82	9.48	62.04	34.40	45.42
mininote	29.15	4.25	6.39	41.89	24.20	28.14
mileage	30.60	9.52	14.68	59.79	24.50	39.64
anymemo	25.51	2.80	3.99	51.97	36.20	41.67
sanity	22.17	3.99	9.36	31.33	16.69	18.69
平均值	44.93	7.89	15.38	63.79	42.86	48.98

进一步深入分析表 2 对应的测试结果可以观察到:用例集中像 sanity 这样的应用程序,无论哪个测试方法都不能达到很高的覆盖率.原因在于程序中存在大量图形界面无法触发的功能模块,包括传感器的相关功能、GPS 定位功能等.这些功能导致了实验中 4 种前沿的图形化界面测试方法,与本文提出的手绘制导测试方法均不能通过界面测试来覆盖相关代码.在未来的工作中,我们考虑在手绘测试中引入对传感器、GPS 等部件的控制来解决这一问题.尽管如此,我们的手绘制导测试方法对于绝大多数应用程序已经能够获得显著的测试效果.

2.2 研究问题2:手绘测试的人力消耗

为了进一步分析手绘测试带来的人力消耗,我们对实验一进行了优化,让测试人员以不同的时间消耗来进行手绘,并观察手绘时间对测试结果的影响.在该实验中,我们要求测试人员分别提供 2 分钟内、4 分钟内、6 分钟内、7 分钟内以及 8 分钟的手绘图形,并以不同的图形驱动本文测试框架,测试效果见表 3、表 4.

Table 3 Statement coverage under different sketching time (%)

表 3 以不同时长手绘时取得的语句覆盖率 (%)

待测应用	2 分钟	4 分钟	6 分钟	7 分钟	8 分钟	2 分钟反馈后
musicnote	58.35	**83.37**	83.96	84.09	84.12	84.12
whohas	45.02	**78.22**	78.26	78.32	78.35	78.35
explorer	**61.11**	61.19	61.19	61.20	61.28	69.4
weight	**55.32**	55.59	55.68	55.70	55.70	70.06
tippy	46.44	**76.64**	76.8	76.88	76.93	88.97
myexpense	7.45	19.15	31.02	**53.33**	56.17	62.04
mininote	**37.76**	38.04	38.06	38.08	38.09	41.89
mileage	24.87	27.05	35.47	47.03	**55.52**	59.79
anymemo	6.73	22.85	33.31	40.91	**43.78**	51.97
sanity	6.02	10.84	16.33	**19.54**	19.85	31.33

Table 4 Branch coverage under different sketching time (%)

表 4 以不同时长手绘时取得的分支覆盖率 (%)

待测应用	2 分钟	4 分钟	6 分钟	7 分钟	8 分钟	2 分钟反馈后
musicnote	52.75	**72.73**	73.31	73.61	73.63	73.63
whohas	25.32	**60.83**	61.28	61.36	61.39	61.39
explorer	**45.39**	47.30	47.31	47.36	47.37	53.42
weight	**42.40**	42.45	42.48	42.48	42.48	62.02
tippy	16.84	**41.44**	44.30	44.36	44.39	65.82
myexpense	6.04	9.83	18.08	**36.45**	38.64	45.42
mininote	**25.49**	25.68	25.77	25.77	25.80	28.14
mileage	16.83	20.07	25.68	32.37	**37.05**	39.64
anymemo	4.95	17.18	23.57	29.79	**33.31**	41.67
sanity	2.17	5.39	9.96	**11.45**	11.68	18.69

表 3、表 4 给出了不同时间消耗下的测试覆盖率,在表格中,我们将覆盖率达到稳定值的位置用粗体标出.从数据结果来看,表 3 的行覆盖率和表 4 的分支覆盖率能够得到类似的结论,即简单移动应用仅需 2 分钟~4 分钟的手绘就可以达到稳定的测试覆盖率,即使较为复杂的移动应用,8 分钟的手绘也足以使测试达到稳定的高覆盖率结果.由此可知:由于我们精心设计了具有一定表达能力的手绘测试语言,使得实际用本文框架进行手绘制导测试的人力成本大大降低,8 分钟左右的手绘即能满足大多数移动应用的测试需要.

2.3 研究问题3:测试语言相关机制的作用

我们设计了一个对照实验来分析本文手绘测试语言的相关机制在测试中的作用.这里,我们主要关注本文测试框架所提供的两项特殊机制:1) 手绘测试语言中的逻辑连接符;2) 初步测试后的交互式反馈.在实验过程中,我们随机对一半测试人员所用的测试框架关闭了逻辑连接符的识别功能,且有一半的测试人员没有交互式反馈的过程.这两个随机因素相互独立,最终获得的实验结果如下.

Table 5 Compared result of sketch-guided testing with/without logical connector elements and the feedback stage

表 5 是否使用逻辑连接符和交互式反馈条件下的覆盖率对照结果

待测应用	分支数量	无交互式反馈(语句覆盖率/分支覆盖率)(%)		有交互式反馈(语句覆盖率/分支覆盖率)(%)	
		不使用逻辑连接符	使用逻辑连接符	不使用逻辑连接符	使用逻辑连接符
musicnote	245	58.10/37.36	84.12/73.63	58.10/37.36	84.12/73.63
whohas	464	78.35/61.39	78.35/61.39	78.35/61.39	78.35/61.39
explorer	885	36.74/28.68	61.28/47.37	52.41/30.47	69.40/53.42
weight	813	51.71/37.22	55.70/42.48	64.73/51.88	70.06/62.02
tippy	1 090	72.02/40.24	76.93/44.39	83.86/42.35	88.97/65.82
myexpense	1 948	56.17/38.64	56.17/38.64	62.04/45.42	62.04/45.42
mininote	2 489	31.68/19.85	38.09/25.80	35.56/22.78	41.89/28.14
mileage	3 761	55.52/37.05	55.52/37.05	59.79/39.64	59.79/39.64
anymemo	4 954	39.06/22.18	43.78/33.31	45.79/28.26	51.97/41.67
sanity	5 723	16.41/8.59	19.85/11.68	27.76/15.35	31.33/18.69

由表 5 的实验结果可知,大多数应用在采用逻辑连接符和交互式反馈的条件下能得到更高的覆盖率.但也

有少量例外:对于 whohas,myexpense 和 mileage,逻辑连接符并不能起到提高测试效果的作用.原因在于:这些应用界面控件的差异较大,测试人员很难通过逻辑连接符将这些程序测试控件的内部逻辑有效地连接起来.对于其他大部分应用程序来说,逻辑连接符能够有效提高测试效果.交互式反馈的测试效果和程序的分支数有密切关系,对于像 music note 和 whohas 这样分支数较少的简单程序,初始的手绘图形已经能够一次性地将用户的测试意图定义明确,因而交互式反馈并不能进一步帮助用户提高测试效果;而对于分支数较多的复杂应用来说,例如 anymemo,sanity 等等,从表 5 的实验结果可以明显看到,交互式反馈能够帮助用户更清晰地定义测试意图,从而使得测试覆盖率得到明显提升.

综上所述,逻辑连接符和交互式反馈在大多数条件下能够有效提高测试效果,对于具有一定逻辑关系的界面控件进行测试时,逻辑连接符的提升效果明显;而对于界面复杂的应用程序进行测试时,交互式反馈能够帮助测试人员清晰地理解和定义其测试意图,从而使测试更为有效.

3 相关工作

本文提出一种手绘制导的移动应用图形界面测试生成方法,它将用户手绘描述的测试意图转换成测试模型,再由模型来生成图形界面的测试用例.这里,我们将从移动应用程序界面测试、模型驱动的测试方法以及手绘技术在软件工程领域的其他应用这 3 个角度介绍一些近年来与本文技术密切相关的工作.

- 移动应用程序的界面测试

由于智能手机与平板电脑日渐普及,近年来涌现出许多与移动应用程序界面测试相关的技术.Yang[4],Clapp[14]以及 Machiry[15]等学者在移动应用程序界面上分别尝试了不同的随机测试策略,以提高其测试效果.Arzt[16],Barros[17],Brutschy[18],Mirzaei[19],Yang[20]等学者引入了静态程序分析方法,使得移动应用程序界面测试的效果得到了进一步提升.Mehlitz 等学者[21]基于模型对移动应用的界面测试进行验证;Mirzaei 等学者[22]基于符号执行来生成移动应用程序的测试用例.Choi 等学者[5]进一步给出了 Swifthand 框架,它采用机器学习算法来自动生成移动应用程序的测试脚本.与所有已有的移动应用测试方法不同,本文的测试方法由用户通过手绘来进行制导.由我们的实验评估可知,这一制导方式成本不高,且效果显著.

- 模型驱动的测试方法

近年来,模型驱动的测试方法成为了界面测试领域的主流方法.该方法将程序的图形界面构建成测试模型,并依据模型生成合法的事件序列,从而能够系统性地生成测试用例.可用的测试模型包括有限状态机模型[23]、交互序列模型[24]、活动图[25]、窗口转换图[26]、语义模型[27,28]、UML 状态图模型等等.Takala 等学者[29]提出了一种人工构造测试模型的方法.Amalfitano 等学者[30]在此基础上进一步设计一种动态爬取图形界面执行事件的测试模型构造技术.Baek 等学者[2]提出了一种多层次的界面测试比较准则,供用户在多个抽象级别上选择不同的测试模型.Memon 等学者定义了更为常用的事件流图测试模型[6,7],并在后来的研究中进一步由使用-观察-运动范式[31]和测试回归技术[32,33]改进了该模型[34].基于这些研究,Nguyen 等学者开发了界面测试工具 GUITAR[10].并由此分支出两个不同的移动应用界面测试工具 AndroidRipper[8]和 MobiGUITAR[9],这两个工具采用了不同的模型遍历策略.本文由手绘来生成测试模型,既能够继承已有模型驱动测试方法的优势,又能够低成本地产生更为有效的测试模型.

- 手绘技术在相关领域的应用

手绘技术作为一种直观的人机交互方式,也被用于软件工程领域内的各种其他需求上,包括图形界面的原型设计与构建[35,36]、UML 生成[37]及自动建模技术[38]等.Coyette[35]和 Lin[36]设计的工具可以允许用户在软件开发过程中手绘界面原型,然后根据手绘信息自动构建应用程序界面,从而使界面程序的开发更为便捷.Grundy 设计的 MaramaMTE[37]工具能够利用用户的手绘信息来生成 UML 图.Wüest 等人设计的 FlexiSketch[38]工具进一步通过手绘来辅助软件设计.与这些现有的工作不同,本文首次将手绘技术引入到移动应用的界面测试上,为软件测试领域探索新途径和新方法.

4 结束语

随着移动设备的普及,移动应用程序获得了广泛使用.然而,移动应用的界面和逻辑功能日益复杂,增加了软件测试的难度.目前,主流的移动应用界面测试技术存在着许多不足:人工编写脚本和录制回放技术需要消耗大量的人力成本,自动化测试在移动应用界面测试的应用场景上受到了诸多限制.针对这些问题,本文提出了一种基于手绘制导的移动应用界面测试方法.该方法定义了一种简单直观且具有较强表达能力的手绘测试语言.测试人员仅需按照该语言定义的语法,在待测应用程序的界面图像上作简单绘制,即可准确表达其测试意图.本文框架会将这些绘制的元素图形自动转换成测试模型,并高效生成符合要求的测试用例.由于用户的手绘图形明确定义了当前的测试意图,因此本文方法不需要对所有可能的事件组合进行遍历,从而使得生成的测试用例更为有效且具有针对性.我们以近年来在相关文献中已经用作移动应用界面测试的评估用例集为基准来评估本文方法的测试效果,评估结果表明:在提供很少人力成本的条件下,手绘图形所表达的用户测试意图在制导移动应用界面测试上能够起到关键作用.

References:

[1] Statista. Number of available applications in the google play store from December 2009 to June 2019|Statista. 2019. https://www.statista.com/statistics/266210/number-of-available-applications-in-the-google-play-store/

[2] Baek Y, Bae D. Automated model-based Android GUI testing using multi-level GUI comparison criteria. In: Proc. of the 31st IEEE/ACM Int'l Conf. on Automated Software Engineering. New York: Academic Press, 2016. 238−249.

[3] Imparato G. A combined technique of GUI ripping and input perturbation testing for Android apps. In: Proc. of the 37th Int'l Conf. on Software Engineering. New York: IEEE Press, 2015. 760−762.

[4] Yang W, Prasad MR, Xie T. A grey-box approach for automated GUI model generation of mobile applications. In: Proc. of the 16th Int'l Conf. on Fundamental Approaches to Software Engineering. Berlin: Springer-Verlag, 2013. 250−265.

[5] Choi W, Necula G, Sen K. Guided GUI testing of Android apps with minimal restart and approximate learning. In: Proc. of the 2013 ACM SIGPLAN Int'l Conf. on Object Oriented Programming Systems Languages & Applications. New York: Academic Press, 2013. 623−640.

[6] Memon AM. An event-flow model of GUI-based applications for testing. Software Testing, Verification and Reliability, 2007,17(3): 137−157.

[7] Memon AM, Pollack ME, Soffa ML. Hierarchical GUI test case generation using automated planning. IEEE Trans. on Software Engineering, 2001,27(2):144−155.

[8] Amalfitano D, Fasolino AR, Tramontana P, De Carmine S, Memon AM. Using GUI ripping for automated testing of Android applications. In: Proc. of the 27th IEEE/ACM Int'l Conf. on Automated Software Engineering. New York: Academic Press, 2012. 258−261.

[9] Amalfitano D, Fasolino AR, Tramontana P, Ta BD, Memon AM. MobiGUITAR: Automated model-based testing of mobile apps. IEEE Trans. on Software Engineering, 2015,32(5):53−59.

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

[11] Choudhary SR, Gorla A, Orso A. Automated test input generation for Android: Are we there yet? In: Proc. of the 2015 30th IEEE/ACM Int'l Conf. on Automated Software Engineering. New York: Academic Press, 2015. 429−440.

[12] Monkey. UI/Application exerciser monkey. 2018. https://developer.android.com/studio/test/monkey

[13] SwiftHand. GitHub-Wtchoi/SwiftHand: Automated testing tool for android applications. 2015. https://github.com/wtchoi/SwiftHand

[14] Clapp L, Bastani O, Anand S, *et al*. Minimizing GUI event traces. In: Proc. of the 2016 24th ACM SIGSOFT Int'l Symp. on Foundations of Software Engineering. New York: Academic Press, 2016. 422−434.

[15] Machiry A, Tahiliani R, Naik M. Dynodroid: An input generation system for Android apps. In: Proc. of the 2013 9th Joint Meeting on Foundations of Software Engineering. New York: Academic Press, 2013. 224−234.

[16] Arzt S, Bodden E. StubDroid: Automatic inference of precise data-flow summaries for the Android framework. In: Proc. of the 38th Int'l Conf. on Software Engineering. New York: Academic Press, 2016. 725−735.

[17] Barros P, Just R, Millstein S, Vines P, Dietl W, Amorim Md, Ernst MD. Static analysis of implicit control flow: Resolving java reflection and Android intents. In: Proc. of the 2015 30th IEEE/ACM Int'l Conf. on Automated Software Engineering. New York: Academic Press, 2015. 669−679.

[18] Brutschy L, Ferrara P, Müller P. Static analysis for independent app developers. In: Proc. of the 2014 ACM Int'l Conf. on Object Oriented Programming Systems Languages & Applications. New York: Academic Press, 2014. 847−860.

[19] Mirzaei N, Garcia J, Bagheri H, Sadeghi A, Malek S. Reducing combinatorics in GUI testing of Android applications. In: Proc. of the 38th Int'l Conf. on Software Engineering. New York: Academic Press, 2016. 559−570.

[20] Yang S, Yan D, Wu H, Wang Y, Rountev A. Static control-flow analysis of user-driven callbacks in Android applications. In: Proc. of the 37th Int'l Conf. on Software Engineering. Vol.1. New York: IEEE Press, 2015. 89−99.

[21] Mehlitz P, Tkachuk O, Ujma M. JPF-AWT: Model checking GUI applications. In: Proc. of the 2011 26th IEEE/ACM Int'l Conf. on Automated Software Engineering. New York: Academic Press, 2011. 584−587.

[22] Mirzaei N, Malek S, Păsăreanu CS, *et al*. Testing android apps through symbolic execution. ACM Sigsoft Software Engineering Notes, 2012,37(6):1−5.

[23] Shehady RK, Siewiorek DP. A method to automate user interface testing using variable finite state machines. In: Proc. of IEEE 27th Int'l Symp. on Fault Tolerant Computing. New York: Academic Press, 1997. 80−88.

[24] White L, Almezen H. Generating test cases for GUI responsibilities using complete interaction sequences. In: Proc. of 11th Int'l Symp. on Software Reliability Engineering. Washington: IEEE Computer Society, 2000. 110−121.

[25] Azim T, Neamtiu I. Targeted and depth-first exploration for systematic testing of Android apps. In: Proc. of the 2013 ACM SIGPLAN Int'l Conf. on Object Oriented Programming Systems Languages & Applications. New York: Academic Press, 2013. 641−660.

[26] Yang S, Zhang H, Wu H, Wang Y, Yan D, Rountev A. Static window transition graphs for Android. In: Proc. of the 2015 30th IEEE/ACM Int'l Conf. on Automated Software Engineerin. New York: Academic Press, 2015. 658−668.

[27] Krishnaswami NR, Benton N. A semantic model for graphical user interfaces. ACM Sigsoft Software Engineering Notes, 2011, 46(9):45−57.

[28] Navarro PLM, Ruiz DS, Pérez GM. A proposal for automatic testing of GUIs based on annotated use cases. In: Proc. of the Advances in Software Engineering. 2010.

[29] Takala T, Katara M, Harty J. Experiences of system-level model- based GUI testing of an Android application. In: Proc. of the 2011 4th IEEE Int'l Conf. on Software Testing, Verification and Validation. Washington: IEEE Computer Society, 2011. 377−386.

[30] Amalfitano D, Fasolino AR, Tramontana P. A GUI crawling-based technique for Android mobile application testing. In: Proc. of the 2011 IEEE 4th Int'l Conf. on Software Testing, Verification and Validation Workshops. Washington: IEEE Computer Society, 2011. 252−261.

[31] Nguyen B, Memon A. An observe-model-exercise paradigm to test event-driven systems with undetermined input spaces. IEEE Trans. on Software Engineering, 2014,40(3):216−234.

[32] Grundy J, Hosking J. Supporting generic sketching-based input of diagrams in a domain-specific visual language meta-tool. In: Proc. of the 29th Int'l Conf. on Software Engineering. Washington: IEEE Computer Society, 2007. 282−291.

[33] Memon AM. Automatically repairing event sequence-based GUI test suites for regression testing. ACM Trans. on Software Engineering and Methodology, 2008,18(2):Article 4.

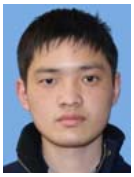
[34] Memon AM, Nguyen BN. Advances in automated model-based system testing of software applications with a GUI front-end. Advances in Computers, 2010,80:121−162.

[35] Coyette A, Vanderdonckt J. A sketching tool for designing anyuser, anyplatform, anywhere user interfaces. In: Proc. of the 2005 IFIP TC13 Int'l Conf. on Human-Computer Interaction. Berlin: Springer-Verlag, 2005. 550−564.

[36] Lin J. A visual language for a sketch-based UI prototyping tool. In: Proc. of the Extended Abstracts on Human Factors in Computing Systems (CHI'99). New York: Academic Press, 1999. 298−299.

[37] Grundy J, Hosking J. Supporting generic sketching-based input of diagrams in a domain-specific visual language meta-tool. In: Proc. of the 29th Int'l Conf. on Software Engineering. Washington: IEEE Computer Society, 2007. 282−291.

[38] Wüest D, Seyff N, Glinz M. Flexisketch: A mobile sketching tool for software modeling. In: Proc. of the Int'l Conf. on Mobile Computing, Applications, and Services. Berlin: Springer-Verlag, 2012. 225−244.



成浩亮(1994－),男,硕士,主要研究领域为软件工程,新型软件测试方法.

汤恩义(1982－),男,博士,副教授,CCF 专业会员,主要研究领域为软件工程,新型软件测试方法,程序分析方法.

玉淳舟(1996－),男,硕士生,主要研究领域为软件工程,新型软件测试方法.

张初成(1992－),男,硕士,主要研究领域为软件工程,新型软件测试方法,程序分析方法.

陈鑫(1975－),男,博士,副教授,CCF 专业会员,主要研究领域为软件工程,软件测试,验证技术.

王林章(1973－),男,博士,教授,博士生导师,CCF 杰出会员,主要研究领域为模型驱动的软件测试与验证,安全测试,软件测试自动化.

卜磊(1983－),男,博士,副教授,博士生导师,CCF 杰出会员,主要研究领域为形式化方法,实时混成系统,复杂系统测试和验证.

李宣东(1963－),男,博士,教授,博士生导师,CCF 会士,主要研究领域为软件工程,软件建模与分析,软件测试与验证.