软件学报 ISSN 1000-9825, CODEN RUXUEW
2024,35(11):5040−5064 [doi: 10.13328/j.cnki.jos.007043] [CSTR: 32375.14.jos.007043]
©中国科学院软件研究所版权所有.
E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

移动应用程序中用户界面显示缺陷检测的经验研究*

刘 哲[1,2,3], 王俊杰[1,2,3], 陈春阳[5], 车 行[1,2,3], 苏宇辉[1,2,3], 王 青[1,2,3,4]

[1](中国科学院 软件研究所 互联网软件技术实验室, 北京 100190)
[2](中国科学院大学, 北京 100049)
[3](中国科学院 软件研究所 智能博弈重点实验室, 北京 100190)
[4](中国科学院 软件研究所 天基综合信息系统重点实验室, 北京 100190)
[5](Monash University, Melbourne 3800, Australia)
通信作者: 王俊杰, E-mail: junjie@iscas.ac.cn; 王青, E-mail: wq@iscas.ac.cn

摘 要: 应用程序图形用户界面 (graphical user interface, GUI/UI) 为应用程序与其终端用户提供了一座可视化的桥梁, 用户可以通过交互操作使用应用程序. 随着移动应用程序的发展, 兼具美学与交互设计的图形用户界面也变得越来越复杂, 用户也更加关注应用程序的可访问性和可用性. 然而图形用户界面的复杂性也对其设计与实现带来巨大的挑战. 由于用户对于移动设备的自定义设置以及不同的设备型号和屏幕分辨率导致用户界面显示问题频繁发生. 例如由于软件或硬件兼容性, 在不同设备上进行界面渲染时总会出现文本交叠、组件遮挡、图像丢失等显示问题. 它们对应用程序的可用性和可访问性产生负面影响, 导致用户体验不佳. 不幸的是, 对于移动应用程序用户界面显示问题的成因知之甚少. 为了应对这一挑战, 收集来自百度众测平台上的 6729 张具有用户界面显示缺陷的应用程序截图和 GitHub 中 1016 个缺陷报告提供的应用程序截图, 采用主题分析方法识别出 9 类用户界面显示缺陷, 然后对 GitHub 中 1016 个缺陷报告和其对应的缺陷代码进行分析, 总结出用户界面显示缺陷本质成因. 研究发现: (1) 在众测数据集中用户界面显示缺陷截图占总截图的 62.1%; (2) 导致界面显示缺陷的原因中字体的缩放设置与组件的自适应设置不适配所占的比例较大; (3) 界面的布局设置会导致界面显示缺陷产生; (4) 硬件加速未开启会影响界面的正常显示.

关键词: 经验研究; 移动应用程序; 界面显示缺陷; 界面显示缺陷成因; 界面显示缺陷检测
中图法分类号: TP311

中文引用格式: 刘哲, 王俊杰, 陈春阳, 车行, 苏宇辉, 王青. 移动应用程序中用户界面显示缺陷检测的经验研究. 软件学报, 2024, 35(11): 5040–5064. http://www.jos.org.cn/1000-9825/7043.htm

英文引用格式: Liu Z, Wang JJ, Chen CY, Che X, Su YH, Wang Q. Empirical Study on UI Display Issue Detection in Mobile Applications. Ruan Jian Xue Bao/Journal of Software, 2024, 35(11): 5040–5064 (in Chinese). http://www.jos.org.cn/1000-9825/7043.htm

Empirical Study on UI Display Issue Detection in Mobile Applications

LIU Zhe[1,2,3], WANG Jun-Jie[1,2,3], CHEN Chun-Yang[5], CHE Xing[1,2,3], SU Yu-Hui[1,2,3], WANG Qing[1,2,3,4]

[1](Laboratory for Internet Software Technologies, Institute of Software, Chinese Academy of Sciences, Beijing 100190, China)
[2](University of Chinese Academy of Sciences, Beijing 100049, China)
[3](State Key Laboratory of Intelligent Game, Institute of Software, Chinese Academy of Sciences, Beijing 100190, China)
[4](Science & Technology on Integrated Information System Laboratory, Institute of Software, Chinese Academy of Sciences, Beijing 100190, China)

* 基金项目: 国家重点研发计划 (2018YFB1403400); 国家自然科学基金 (62072442, 62002348)
收稿时间: 2022-10-12; 修改时间: 2023-01-27, 2023-04-02, 2023-05-10, 2023-06-12; 采用时间: 2023-08-28; jos 在线出版时间: 2024-01-17
CNKI 网络首发时间: 2024-01-08

[5](Monash University, Melbourne 3800, Australia)

Abstract: The graphical user interface (GUI/UI) provides a visual bridge between the application and its end users, and users can use the application through interactive operations. With the development of mobile applications, GUI, which combines aesthetics and interaction design, has become more and more complex, and users are increasingly concerned about the accessibility and availability of applications. However, the complexity of GUI also brings great challenges to its design and implementation. Due to user-defined settings for mobile devices and different device models and screen resolutions, UI display issues frequently occur. For example, due to software or hardware compatibility, when rendering interfaces on different devices, there will always be display issues such as text overlap, component masking, and image loss. They have a negative impact on the availability and accessibility of applications, resulting in poor user experience. Unfortunately, little is known about the causes of UI display issues of mobile applications. In order to cope with this challenge, this study collects 6729 screenshots of applications with UI display issues from Baidu crowdtesting platform and 1016 screenshots of applications provided by issue reports in GitHub and identifies nine types of UI display issues using the theme analysis method. Through the analysis of 1061 UI issue reports from GitHub and the corresponding defective code, the essence and causes of UI display issues are summarized. The research found that (1) 62.1% of the total screenshots in crowdtesting dataset are defective screenshots displayed on the UI; (2) the reason for the UI display issues is that the font scaling setting does not match the adaptive setting of components to a great extent; (3) the layout setting of the interface will lead to display issues; (4) If the hardware acceleration is not turned on, the normal display of the interface will be affected.

Key words: empirical study; mobile application; UI display issue; root causes of UI display issues; UI display issue detection

随着移动设备的普及与互联网的发展, 移动应用程序在我们的日常生活中发挥着越来越重要的作用. 移动应用程序为消费者提供了与智能设备通信、交互、导航、娱乐、购物和金融等服务. 图形用户界面 (graphical user interface, GUI/UI) 在几乎所有的现代桌面软件和移动应用程序中扮演着重要角色. 它在应用程序和最终用户之间搭建了一个可视化的桥梁, 他们可以通过该桥梁进行交互. 开发人员设计的用户界面需要适当的用户交互、信息架构和 GUI 视觉效果. 因此, 良好的图形用户界面设计使应用程序易于使用、实用且高效, 这对应用程序的成功以及其用户的忠诚度有很大影响. 例如, 计算机用户认为苹果的 Macintosh OS 操作系统比微软的 Windows 操作系统有更好的图形用户界面, 他们的正面看法几乎是 Windows 用户的两倍, 导致品牌忠诚度增加 20%[1–4].

然而, 随着移动设备性能的提高和用户对图形用户界面的审美要求, 图形用户界面设计中越来越多复杂的视觉效果 (密集媒体嵌入、动画、灯光、浮动和阴影) 给开发人员的实现带来了巨大挑战. 由于移动应用程序的重要性, 确保其能正确显示信息并在用户使用过程中无明显缺陷是至关重要的. 作为用户和软件之间的桥梁, 图形用户界面对于应用程序的可访问性至关重要. 随着大量支持 Android 的硬件设备、Android API 及 Android 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 Android 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. Android 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[5,6]. 根据 2023 年 1 月份 Appbrain 的统计数据显示, Android 9.0 版本以下的应用程序的市场占比仍然高达 26.2%. 此统计分析结果表明当前 Android 系统版本的碎片化问题已经十分严重. 现有研究将 Android 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. 操作系统碎片表示 Android 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 Android 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. 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. 不同的设备与分辨率导致的 Android 碎片化问题也是产生用户界面显示缺陷的重要原因之一. 如图 1 所示, 许多显示问题[6–12], 例如文本重叠、缺少图像、组件遮挡等问题频繁出现, 这对 Android 应用程序的流畅使用产生了负面影响, 导致用户体验非常糟糕并造成相应的用户损失. 我们发现大多数图形用户界面显示缺陷是由不同设备中的不同系统设置引起的, 特别是对于 Android 操作系统. 研究表明有超过 10 个主要版本的 Android 操作系统运行在 24000 多种不同型号且具有不同的屏幕分辨率的设备上[2]. 尽管应用程序仍然可以与这些显示缺陷一起运行, 但它们会对应用程序的流畅使用产生负面影响, 降低应用程序的可访问性和可用性.

目前, 为了确保图形用户界面显示的正确性, 公司必须招募许多测试人员进行应用程序图形用户界面测试或

© 中国科学院软件研究所 http://www.jos.org.cn

利用众包测试[13]. 尽管测试人员可以发现这些图形用户界面显示缺陷, 但这种机制仍然存在两个问题. 首先, 它需要大量的人力, 因为测试人员必须通过不同的交互方式手动探索数十个页面, 还需要检查不同操作系统版本和设备上的用户界面显示, 以及不同分辨率或屏幕大小. 其次, 图形用户界面显示中的一些错误, 尤其是相对较小的错误, 如文本重叠、组件遮挡等缺陷, 通常被测试人员忽略. 为了克服这些问题, 一些应用程序开发团队采用了快速应用程序开发 (rapid application development, RAD)[14–17], 其重点是通过频繁迭代和持续反馈快速开发应用程序. 开发人员利用用户的反馈来检测图形用户界面显示缺陷, 但这是一种被动的缺陷修复方式, 可能已经伤害了应用程序的用户, 导致市场份额损失. 因此, 与从用户处获取反馈来保障用户界面的正常显示相比, 我们需要一种更主动的机制, 可以在应用发布前检查图形用户界面显示, 自动发现图形用户界面潜在的问题, 并提醒开发人员修复这些问题.

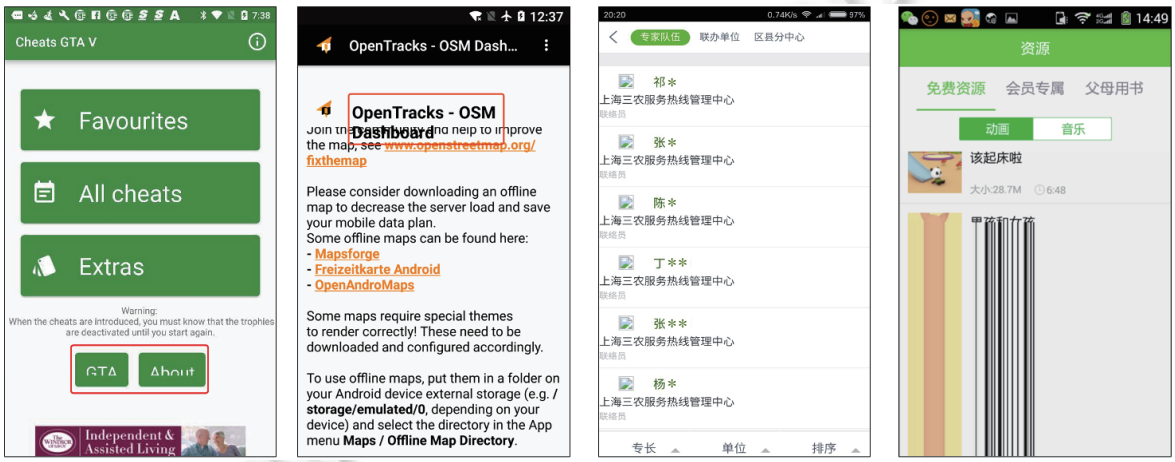


图 1 常见的用户界面显示缺陷

先前的研究和调查表明检测、定位并修复一个已知用户界面显示缺陷平均需要两天时间, 这既耗时又费力. 开发人员必须理解用户界面显示缺陷, 在应用程序上复现缺陷, 在源代码中定位其根本原因, 并推测可能修复缺陷的策略. 这些任务的难度也因软件复杂性、遗留代码的代码不一致性和代码量而加剧, 尤其是在高度碎片化的前端代码中更为突出[5,6]. 具体而言, 应用程序可以通过平均渲染 10 个 XML 文件生成单个用户页面, 这使得很难确定页面上有缺陷组件的可疑 XML 文件. 此外, 用户界面显示缺陷涉及组件的多样性以及图形用户界面的多样化设计风格使情况变得更糟. 考虑到检测、定位与修复用户界面显示缺陷需要大量的人力成本和时间成本, 需要对应用程序的用户界面显示缺陷的种类和成因进行分析, 以帮助测试人员和开发人员检测并修复应用程序渲染过程的界面显示缺陷.

本文旨在对移动应用程序用户界面显示缺陷进行经验研究. 为开发人员检测、定位与修复用户界面显示缺陷提供参考和建议, 进而提高移动应用程序的图形用户界面质量, 降低缺陷修复和维护成本. 具体而言本文试图回答 3 个研究问题.

RQ1: 用户界面显示缺陷的常见类型有哪些?

RQ2: 用户界面显示缺陷的本质成因是什么?

RQ3: 用户界面显示缺陷呈现哪些视觉特征?

回答以上的 3 个研究问题可以帮助开发人员了解用户界面显示缺陷的成因, 为用户界面显示缺陷的检测、定位与修复提供指导; 也可以帮助现有自动化缺陷检测工具对用户界面显示缺陷进行检测, 提升工具的测试完备性.

为了回答上述研究问题. 本文选取了国内流行并得到广泛应用的百度众包测试平台, 收集了该平台上 536 个移动应用程序众测任务的 10 836 个非重复屏幕截图, 人工标注了这些屏幕截图存在的用户界面显示缺陷. 结果显示, 众测报告的屏幕截图中有 6 729 张截图 (62.1%) 存在界面显示缺陷, 这会严重影响用户体验, 并降低应用程序

的声誉. 本文将这些界面显示缺陷截图数据进行了共享, 以便研究者对其进行进一步的使用和研究. 我们利用主题分析方法对众测数据集中应用程序界面显示缺陷与 GitHub 数据集中 1016 个用户界面显示缺陷截图进行分析分类, 共分为 3 个大类别 (交叠与遮挡缺陷、丢失与错误信息缺陷、显示与加载失败缺陷) 和 9 个子类别 (组件遮挡缺陷、文字交叠缺陷、文字缺失缺陷、组件溢出缺陷、图片丢失缺陷、空值问题缺陷、文本乱码缺陷、花屏显示缺陷、加载失败缺陷). 此外, 我们还选取了被广泛应用的开源项目托管平台 GitHub 的开源 Android 应用程序与他们的缺陷报告 (issue report). 我们通过自动关键字过滤, 从 1560 个流行开源 Android 应用程序的 18601 个缺陷报告中检索 2636 个潜在的用户界面显示缺陷报告. 然后, 我们手动验证每个自动收集的报告, 并获取 1016 个用户界面显示缺陷, 包括详细完整的用户界面显示缺陷描述、对应用户界面显示缺陷的相关屏幕截图以及缺陷代码. 然后, 我们使用该数据集对用户界面显示缺陷的根本成因进行了国际上首次完整的经验研究, 理解缺陷的特征. 本文的发现主要包括: (1) 众测数据集的用户界面显示缺陷截图占总截图的 62.1%; (2) 导致界面显示缺陷的原因中字体的缩放设置与组件的自适应设置不适配所占的比例较大; (3) 界面的布局设置会导致界面显示缺陷产生; (4) 硬件加速未开启会影响界面的正常显示.

本文的主要贡献如下.

(1) 收集了共计 7745 张存在用户界面显示缺陷的截图和对应的用户界面显示缺陷报告, 数据集可用于后续的研究工作 (https://pan.baidu.com/s/1DlyEiMYdIsLBwCTTjIB24Q?pwd=1111 公开了本文经验研究的数据集, 增强后的缺陷截图与示例 APP 等).

(2) 采用主题分析方法, 分析并总结了 3 个大类 9 个子类的用户界面显示缺陷, 以及各类别用户界面显示缺陷的分布情况.

(3) 通过分析缺陷报告与对应的缺陷代码. 总结了用户界面显示缺陷的本质成因.

(4) 分析了用户界面显示缺陷的视觉特征, 包括显式特征和隐式特征, 可用于增强自动化测试工具.

本文第 1 节介绍 Android 图形用户界面以及界面显示缺陷的相关背景. 第 2 节介绍本文所构建并使用的数据集与分析方法. 第 3 节介绍本文数据分析的结果以及我们对界面显示缺陷的分类结果. 第 4 节介绍相关工作. 第 5 节总结全文和展望未来工作.

1 研究背景

本节主要介绍用户界面显示缺陷涉及的关键概念, 包括 Android 图形用户界面与其包含的组件, Android 视图层次结构文件的相关概念.

1.1 Android 图形用户界面

Android 图形用户界面包含用户可查看并与之互动的所有内容. 为了帮助开发人员灵活地操作 Android 视图并构建 Android 图形用户界面, Android 软件开发工具包 (SDK) 允许开发人员使用视图和视图组对象在 Android 源代码中构建图形用户界面, 并且提供了丰富多样的预构建界面组件 (结构化布局对象, 界面控件等). 如图 2 所示, 视图对象通常被称为“组件” (例如, ImageView、TextView、Button、EditText), 而视图组对象通常被称作“布局”, 它提供了各种布局结构 (例如, LinearLayout、RelativeLayout、ConstraintLayout、MotionLayout)[6,7].

常见的 Android 组件如下.

1) ImageView: 通常用于显示图像资源, 例如 Bitmap 或 Drawable 资源. ImageView 可以通过设置对应的属性值对图像应用色调和处理图像缩放.

2) TextView: 通常用于向用户显示文本的用户界面元素, 可以通过自定义设置属性调整文本的显示大小与系统默认的按钮背景设置样式.

3) EditText: 通常用于输入和修改文本的用户界面元素. 可以通过设置 inputType 属性确定输入内容的类型.

常见的 Android 布局如下.

1) LinearLayout: 通常用于将其子项组织成单个水平或垂直行的布局. LinearLayout 的所有子视图依次堆叠,

© 中国科学院软件研究所 http://www.jos.org.cn

因此无论子视图有多宽, 垂直列表每行均只有一个子视图, 水平列表将只有一行高 (最高子视图的高度加上内边距). LinearLayout 会考虑子视图之间的边距以及每个子视图的对齐方式 (右对齐、居中对齐或左对齐).

2) RelativeLayout: 通常用于指定子对象相对于彼此 (子 A 在子 B 左侧) 或相对于父对象 (与父对象顶部对齐) 的位置. 每个视图的位置可以指定为相对于同级元素的位置 (例如, 在另一个视图的左侧或下方) 或相对于父级 RelativeLayout 区域的位置 (例如在底部、左侧或中心对齐).

3) ConstraintLayout: 通常可以让开发人员使用扁平视图层次结构 (无嵌套视图组) 创建复杂的大型布局. 它与 RelativeLayout 相似, 其中所有的视图均根据同级视图与父布局之间的关系进行布局, 但其灵活性要高于 RelativeLayout. 每个约束条件均表示与其他视图、父布局或隐形引导线之间连接或对齐方式. 每个约束条件均定义了视图在竖轴或者横轴上的位置; 因此每个视图在每个轴上都必须有至少一个约束条件, 但通常情况下会需要更多约束条件.

4) MotionLayout: 通常用于帮助开发人员管理应用中的组件动画, 旨在移动与用户交互的界面元素 (例如按钮和标题栏), 以及对其调整大小和添加动画效果.

图 2 Android 图形用户界面, 组件与布局示例

如图 3 所示, 开发人员可通过在 XML 文件中声明界面元素. Android 提供对应组件和布局的简明 XML 词汇, 也可以使用 Android Studio 的 Layout Editor, 并采用拖放界面来构建 XML 布局. 开发人员可以通过 XML 文件将应用外观代码与控制其行为的代码分开, 并且 XML 文件还有助于为不同屏幕尺寸和屏幕方向提供不同布局

© 中国科学院软件研究所 http://www.jos.org.cn

(Google 开发者文档在支持不同的屏幕尺寸的章节中深入阐述了此内容). 在 Android XML 文件的源代码中的每个组件和布局都有其特定 XML 属性, 用于设置其边界、样式和引用的外部资源等. 常见的组件属性有 textSize、layout_width、layout_height 等. 开发人员可以使用以下某种常量 (wrap_content、match_parent) 来设置宽度 (layout_width) 或高度 (layout_height) , 有助于确保应用能在各类尺寸的设备屏幕上正确显示. 其中 wrap_content 用于将组件大小调整为内容所需的尺寸, match_parent 用于将组件大小设置为尽可能采用其父组件所允许的最大尺寸. 每当运行 Android 应用程序, 移动设备会通过 Android 渲染来实现屏幕上 Android 图形用户界面的显示.

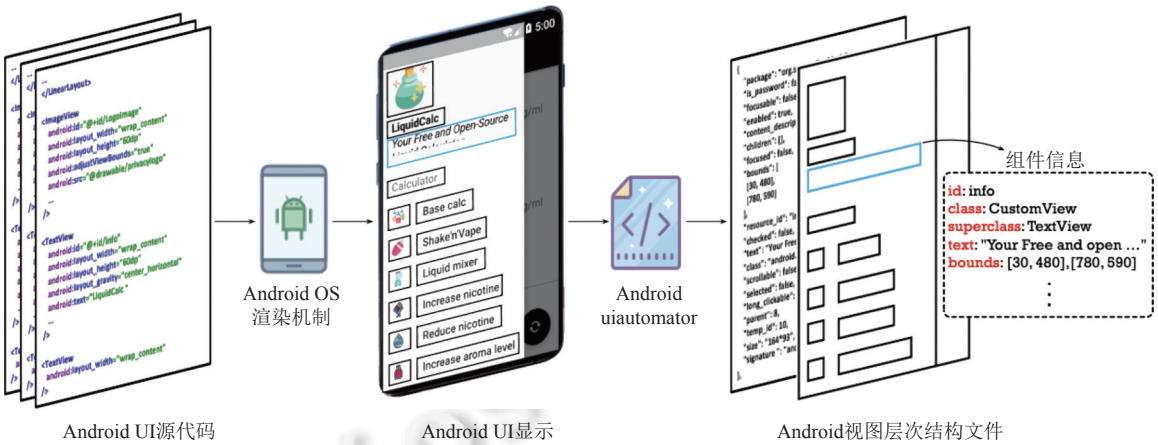


图 3 Android 图形用户界面与图层次结构文件示例

1.2 Android 视图层次结构文件

在运行过程中, 开发人员可以通过 Android 调试桥 (ADB)[15,16]命令的“uiautomator dump”获取与当前 UI 页面 (截图) 对应的视图层次结构文件 [8,9]. 如图 3 所示, 视图层次结构文件包括组件信息 (坐标信息、ID、组件类型、文本描述等), 以及当前 UI 页面上的布局信息. 在图形用户界面渲染期间, 可以将屏幕视为画布进行绘制. 在视图树中, 父视图绘制在其子视图之前, 子视图根据其顺序值绘制在其父视图之上. 将首先绘制顺序较低的子级. 对于具有相同顺序的对象, 将根据其预顺序遍历序列绘制它们. 与源代码不同, 视图层次结构文件获取渲染后显示在屏幕上的所有组件 (某些组件可以从多个源代码文件动态生成), 并包括屏幕中这些组件的坐标. 开发人员可以根据视图层次结构文件中各个组件的坐标信息、ID、组件类型、文本描述信息来定位到源代码中的每个组件. 因此, 在本文中我们使用视图层次结构文件作为源代码和 Android UI 之间的桥梁, 用于分析用户界面显示缺陷的成因.

2 数据与分析方法

本文主要围绕以下 3 个研究问题展开.

1) RQ1: 在移动应用程序中, 常见的用户界面显示缺陷的类型有哪些? 研究用户界面显示缺陷的缺陷类型以及这些类型的分布情况. 可以帮助测试人员了解界面显示缺陷的类型, 为自动化测试工具检测缺陷提供样例.

2) RQ2: 用户界面显示缺陷的成因是什么? 哪些原因导致用户界面显示缺陷的产生? 研究不同类型的用户界面显示缺陷的本质成因与不同原因的分布情况. 可以帮助开发人员规避某些行为, 从而提升代码质量.

3) RQ3: 用户界面显示缺陷呈现哪些视觉特征? 研究界面显示缺陷的视觉特征, 可以帮助测试人员或自动化测试工具更好的检测用户界面显示缺陷.

2.1 数据收集

- 数据来源

本文的数据集分为两部分.

第 1 部分数据是来自百度众测平台上的 536 个不重复的 APP. 百度众测平台是中国最大的众测平台之一, 百度众测平台的众测工人需要在执行平台提供的测试任务后提交测试报告[17,18]. 考虑到目前没有公开的 Android 应用程序界面显示缺陷的数据集, 因此我们采用百度众测平台提供的测试报告作为我们的数据集. 该数据集包含百度众测平台在 2015 年 1 月–2017 年 1 月期间的发布的 536 个 Android 移动应用程序众测任务. 这些应用程序均为流行应用并属于不同的类别, 如新闻、娱乐、金融、购物、社交、开发、医疗等. 在平台发布的每项众测任务中, 众测人员都会提交数百份的测试报告, 其中描述了移动应用程序的测试是如何进行的, 在什么样的设备和环境下进行的测试, 在测试过程中发生了什么, 以及附带的测试截图. 我们之所以使用此数据集, 是因为它既包含大量的移动应用程序的屏幕截图, 也包含相应的 bug 描述和复现步骤, 这有助于搜索和分析用户界面显示缺陷. 此数据集中包含 10836 个独特的移动应用程序 GUI 屏幕截图, 有助于我们分析用户界面显示缺陷的种类与分布情况.

第 2 部分数据是来自广泛应用的开源项目托管平台 GitHub 上的开源 Android 应用程序与他们的缺陷报告 (issue report). 我们选取了 2017 年 1 月–2021 年 1 月的 1560 个流行开源 Android 应用程序. 这些应用程序拥有大量的活跃软件开发者参与项目的开发与维护, 它们的公共问题跟踪系统中 (issue) 包含 18601 个缺陷报告. 这些缺陷报告和其对应的源代码有助于我们分析用户界面显示缺陷的成因与分布情况.

● 收集数据

如图 4 所示, 由于我们的实验数据集来自两个不同的部分 (即众测数据集, GitHub 数据集), 因此我们针对这两个数据集设置了不同的数据收集方法. 首先, 针对众测数据集, 我们对测试报告进行筛选并获取其对应的应用程序运行界面屏幕截图. 具体包含如下步骤.

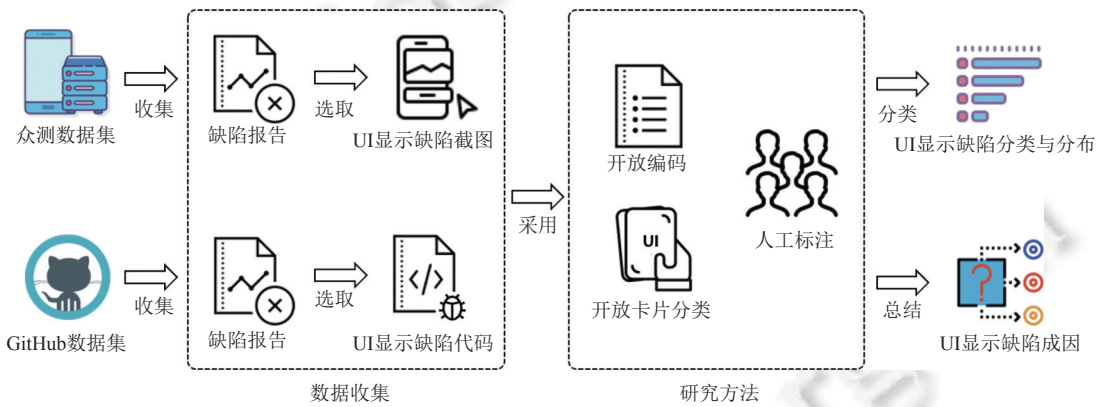


图 4 数据收集与分析方法框架

1) 使用百度众测提供的 API 收集百度众测平台在 2015 年 1 月–2017 年 1 月期间的发布的 536 个 Android 移动应用程序众测任务. 每个测试任务包含以下属性, 测试任务的 id, 测试任务的描述, 测试任务的参与人数. 针对每个测试任务, 我们可以获取众测人员提交的测报告, 测试报告包括测试人员 id、测试报告 id、bug 详情、bug 复现步骤描述、屏幕截图的链接.

2) 我们首先根据众测平台上常见的用户界面显示缺陷报告, 总结了可以描述界面显示缺陷的关键词. 具体来说, 我们首先从平台上的 10836 个缺陷报告中随机筛选出 2000 个缺陷报告以保证数据的代表性. 3 位作者分别对其进分析并根据截图的缺陷特征判断其是否属于用户界面显示缺陷. 如果作者之间存在分歧, 则通过讨论并最终达成共识. 针对每个属于界面显示缺陷的缺陷报告, 3 位作者分别总结 3 个以上的关键词, 并按照关键词的含义与相似度将所有关键词进行合并和组合. 所总结的关键词如下: 用户界面显示、UI、GUI、界面显示、显示问题、显示 bug、显示失败、失败显示、界面显示缺陷、显示缺陷、交错、遮挡、前后遮挡、组件遮挡、交错、组件交错、重叠、覆盖、失败、交叠、错位、消失、超出、乱码、缺失、加载、卡顿、白屏、黑屏、花屏. 根据这些用户界面显示缺陷相关的关键词, 我们使用关键词匹配的方法判断测试报告的 bug 详情和 bug 复现步骤描述中是

© 中国科学院软件研究所 http://www.jos.org.cn

否含有关键词, 自动地从 10836 个测试报告中筛选出 8111 个存在关键词的报告.

3) 然后我们设计了一个网络爬虫系统, 自动化地从百度众测服务器上下载 8111 个测试报告对应的应用程序运行截图.

4) 对于这些 GUI 屏幕截图, 3 位作者分别手动检查每个屏幕截图以及缺陷报告中的相应描述以判断其是否存在界面显示缺陷. 针对存在异议的报告和截图, 3 位作者会进行讨论以确定是否同意对方的结果并更改自己的分析结果. 只有 3 个作者标记一致的 GUI 屏幕截图才被视为存在界面显示缺陷. 共确定 6729 个 GUI 屏幕截图, 其中用户界面显示缺陷占所有缺陷报告的 62.1% (6729/10836). 这一结果表明, 用户界面显示缺陷在众包测试期间是移动应用程序缺陷中不可忽视的一部分, 应需要投入更多的关注以提高应用程序质量.

5) 该步骤获得的 6729 个用户界面显示缺陷截图也构成了我们经验研究的众测数据集. 请注意, 目前没有关于移动应用程序用户界面显示缺陷的公开数据集. 虽然百度众测平台提供的数据为 2017 年前的应用程序, 但其可以帮我们了解用户界面显示缺陷的类型. 我们也在下文的 GitHub 数据集上进一步分析了 2017 年后新应用程序上用户界面显示缺陷的分布情况.

其次, 针对 GitHub 数据集, 我们对开源的 Android 应用程序的公共问题跟踪系统模块 (issue) 和提交请求模块 (pull request) 进行筛选, 选取与用户界面显示相关的缺陷报告 (issue report) 并获取其对应的应用程序运行界面屏幕截图和缺陷代码. 具体包含如下步骤.

1) 我们采用以下标准进行应用程序选择: 在 Google Play 中下载超过 1000 次 (流行) , 在 GitHub 上拥有完整的公共问题跟踪系统 (可跟踪), 3 年以上的开发历史 (值得信赖), 100 多个代码提交 (维护良好), 以及超过 10k 行的代码 (大规模). 我们最终选择了从 2017 年 1 月–2021 年 1 月的 1560 个流行开源 Android 应用程序.

2) 对于选定的应用程序, 我们会自动抓取其最新版本及其相应的缺陷报告, 即总共 16201 份报告. 我们首先从 30 个应用程序中随机选择 939 个缺陷报告和屏幕截图, 并检查其缺陷描述. 结合我们第 1 个数据集提取的可以描述界面显示缺陷的关键词, 我们总结了一组英文的关键词作为显示问题的关键字 (GitHub 的缺陷报告以英文的为主) , 例如: screen、GUI、UI、display、rendering、photo、image、picture、icon、fuzzed、overlap、occlusion、show、showing、blurred.

3) 我们使用这些关键字从剩余的 15262 个已爬取的报告中自动筛选潜在的用户界面显示缺陷报告, 并仅返回具有 3 个或更多关键字的报告, 以避免出现不相关的报告. 我们获得了 6796 个具有潜在的用户界面显示缺陷的候选缺陷报告.

4) 然后, 3 位作者手动检查上述返回的候选缺陷报告, 依据测试报告的缺陷描述的含义与关键词并结合缺陷截图的特征判断是否存在界面显示缺陷, 以确保报告真正与用户界面显示缺陷相关. 具体来说, 每个缺陷报告对应了缺陷修复过程中开发人员合并的 pull request (已完成修复), 通过缺陷报告和修复补丁的自然语言描述可以确定缺陷的信息, 缺陷的代码, 以及修复方法. 第 4 位作者重新检查结果, 直到达成共识. 经过这样的检查, 我们收集了 2636 个具有用户界面显示缺陷的缺陷报告, 其中 1016 个缺陷报告有详细完整的屏幕截图, 描述了用户界面显示缺陷, 并且可以链接到缺陷代码 (pull request).

5) 该步骤获得的这些 1016 个完整的用户界面显示缺陷报告构成了我们经验研究的 GitHub 数据集. 为了进一步分析 2017–2021 年用户界面显示缺陷的分布情况, 我们将 1016 个缺陷报告中的 1016 张用户界面显示缺陷截图用于 RQ1 的分析. 请注意, 由于某些用户界面显示缺陷的描述可能不包含 (或包含很少) 关键字, 因此我们的数据集并不详尽. 尽管如此, 该数据集的规模与其他研究中手工构建的缺陷数据集相比是较大的.

2.2 研究方法

● 主题分析法

对于 RQ1, 我们需要对众测数据集中的用户界面显示缺陷进行分类, 以研究用户界面显示缺陷的缺陷类型以及这些类型的分布情况. 为了进一步分析缺陷在新 APP 上的分布情况, 我们使用了 GitHub 数据集中的界面显示缺陷截图对众测数据集的分析结果进行验证. 对于 RQ2 我们需要根据缺陷类型对 GitHub 数据集中的界面显示缺

© 中国科学院软件研究所 http://www.jos.org.cn

陷进行成因分类与分析. 考虑到要对数据集进行人工的分类与分析, 我们采用了主题分析[19−25]方法来对用户界面显示缺陷的截图与成因进行分析, 主要包括以下步骤.

1) 开放编码 (open coding)[17]: 对于我们获取的应用程序缺陷截图和缺陷代码. 开放编码主要通过对其进行概念化并以适当方式将概念重新抽象、提升和综合为类属. 开放编码是通过对现象进行仔细的研究, 以便进行命名和分类的分析工作, 即分析、检视数据并对其进行概念化的归纳、比较的过程. 开放编码的目的是从截图和缺陷代码中发现概念类属, 对类属加以命名确定类属的属性.

2) 开放卡片分类 (open card sorting)[26]: 开放卡片分类方法主要是帮助研究人员如何理解和分类信息. 参与者会将主题归为他们人员有意义的类别并进行标记, 即将相似含义的卡片放在一起, 相似含义的卡片组成一个集合, 这样就得到不同的集合, 最后对这些集合进行命名. 本文第 1 作者和第 2 作者各自独立完成了数据的编码和集合命名. 本文使用 Fleiss Kappa[20]来衡量两个作者之间的意见一致性, 当两个标记者分类结果的 Kappa 值越接近 1 则表明两个作者基本上同意对方的分类, 对于存在分歧的分类通过讨论并达成共识.

● 人工标注法

根据主题分析方法, 我们采用人工标注的方法对数据集进行分类和标注. 具体而言, 对于众测数据集, 我们需要标注各用户界面显示缺陷的类别. 在手动检查过程中, 我们注意到存在不同类型的用户界面显示缺陷, 对这些缺陷进行分类将有助于相关方法的设计和评估. 第 1 和第 4 作者 (均拥有 3 年以上 Android 开发经验的) 按照开放编码过程手动检查它. 我们分析缺陷屏幕截图并对用户界面显示缺陷进行分类. 具体来说, 每个参与标注工作的作者都会仔细检查缺陷屏幕截图. 我们将相似的代码归为一类, 分组过程是迭代的. 具体来说, 我们经常在类别之间来回移动. 在两位提交人之间没有达成协议的情况下, 第 2 作者充当仲裁人, 讨论并解决冲突. 根据统计结果显示, 两个标记者分类结果的 Kappa 值为 0.863, 有 5807 张图片作者标记的是一致的, 这表明两个作者基本上同意对方的分类. 我们遵循以上的标注程序直到所有作者达成一致. 我们将这些用户界面显示缺陷分为 3 大类, 包括交叠与遮挡缺陷、丢失与错误信息缺陷、显示与加载失败缺陷. 进而分为 9 个子类别, 包括组件遮挡缺陷、文字交叠缺陷、文字缺失缺陷、组件溢出缺陷、图片丢失缺陷、空值问题缺陷、文本乱码缺陷、花屏显示缺陷、加载失败缺陷等. 缺陷的详细信息将在第 3 节中进行介绍.

对于 GitHub 数据集, 第 1 和第 4 作者共同参与了这个过程. 遵循开放编码协议, 我们分析了获取的用户界面显示缺陷的缺陷报告, 并对用户界面显示缺陷的根本原因进行分类和分析. 具体来说, 每个参与标注工作的作者都会仔细检查缺陷报告的内容, 包括缺陷报告标题、缺陷的详细描述和缺陷相关讨论、测试人员提供的屏幕截图和缺陷相关的代码. 然后, 每个参与标注工作的作者为用户界面显示缺陷的根本原因构建一个分类法. 我们将相似的代码归为一类, 分组过程是迭代的. 具体而言, 我们不断在类别/子类别之间来回移动, 并发布报告以改进分类. 在两位参与标注工作的作者之间没有达成协议的情况下, 第 2 和第 3 作者充当仲裁人来讨论和解决冲突. 具体而言, 对于分配了不同标签的缺陷报告, 4 位作者重新确定了其根本原因和缺陷代码, 并确定是否需要新的类别/子类别. 根据统计结果显示, 两个标记者分类结果的 Kappa 值为 0.855, 有 869 个缺陷报告两位作者标记一致, 这表明两个作者基本上同意对方的分类. 我们遵循程序, 直到所有作者达成一致. 第 3 节详细介绍了缺陷根本原因的派生分类.

3 结果及分析

本文使用第 2.2 节的数据分析方法, 总结出常见的用户界面显示缺陷包括 3 个大类别 (交叠与遮挡缺陷、丢失与错误信息缺陷、显示与加载失败缺陷) 和 9 个子类别 (组件遮挡缺陷、文字交叠缺陷、文字缺失缺陷、组件溢出缺陷、图片丢失缺陷、空值问题缺陷、文本乱码缺陷、花屏显示缺陷、加载失败缺陷).

3.1 RQ1: 常见的用户界面显示缺陷的类型有哪些?

在人工标注众测数据集的过程中, 我们注意到存在不同类型的用户界面显示缺陷, 对这些界面显示缺陷进行分类将有助于相关缺陷检测方法的设计和评估. 如图 5 所示, 我们将这些用户界面显示缺陷分为 3 大类, 包括交叠与遮挡缺陷、丢失与错误信息缺陷、显示与加载失败缺陷. 用户界面显示缺陷的种类与其在众测数据集的分布情

© 中国科学院软件研究所 http://www.jos.org.cn

况如表 1 所示, 交叠与遮挡类缺陷在数据集中占的比例最大, 48% 的缺陷截图表现为 UI 组件之间出现交叠或者遮挡; 该类缺陷的 3 个子类, 组件遮挡缺陷, 文字交叠缺陷, 文字缺失缺陷, 组件溢出缺陷分别占 22%、13%、8%、6%. 其次数据集中有 32% 的截图存在丢失与错误信息缺陷, 主要体现在图片和文本的缺失或乱码; 该缺陷存在 3 个子类, 图片丢失缺陷, 空值问题缺陷, 文本乱码缺陷分别占 15%、10%、7%. 有 20% 的缺陷截图存在显示与加载失败缺陷, 分别体现为花屏显示缺陷, 加载失败缺陷, 分别占 13%、7%.

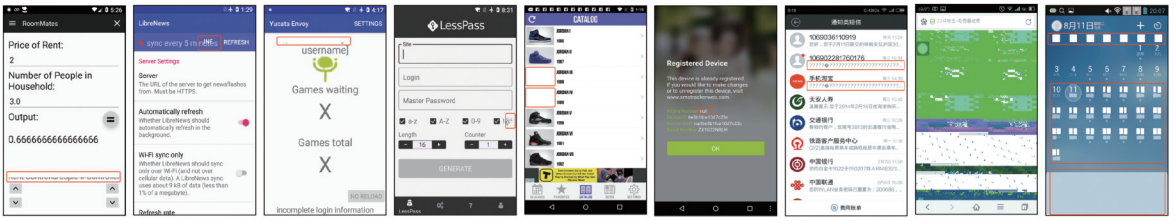


图 5 用户界面显示缺陷样例

表 1 用户界面显示缺陷的种类与其在众测数据集的分布情况

类别		数量	比例 (%)
交叠与遮挡缺陷	组件遮挡缺陷	1 447	22
	文字交叠缺陷	848	13
	文字缺失缺陷	532	8
	组件溢出缺陷	397	6
丢失与错误信息缺陷	图片丢失缺陷	1 030	15
	空值问题缺陷	686	10
	文本乱码缺陷	458	7
显示与加载失败缺陷	花屏显示缺陷	861	13
	加载失败缺陷	471	7

本文针对这 3 个大类和 9 个子类的用户界面显示缺陷进行详细的展示与说明, 详细信息如下.

● 交叠与遮挡缺陷

根据众测数据集的统计结果显示有 48% 的缺陷截图属于交叠与遮挡缺陷类型. 如图 6 所示, 这些缺陷主要表现为组件之间出现相互遮挡、文本之间相互交叠、文本或者组件出现缺失的情况, 影响用户对移动应用程序功能的理解并降低应用程序的可用性和可访问性.

(1) 组件遮挡缺陷

由于不同设备的屏幕分辨率和用户自定义设置不同, 有时会出现组件的布局设置不兼容导致各组件之间相互遮挡. 如图 6(a) 所示, 该缺陷通常表现为文本信息或组件被其他组件遮挡, 它通常与 TextView, Button 或 EditText 等 UI 组件一起出现. 此类型缺陷在用户界面显示缺陷中所占的比例最大, 出现的频率最高. 产生主要原因如下: 元素高度设置不当, 或设置较大字体时引发的自适应问题.

(2) 文字交叠缺陷

在设计应用程序 UI 的过程中, 开发人员有时会选取绝对布局来设置各个 UI 组件的位置使其固定排列在开发人员设置位置. 如图 6(b) 所示, 该缺陷通常表现为两段文本相互重叠, 它通常与 TextView 组件一起出现. 此类型缺陷可能是由于不同设备型号之间的自适应问题造成的. 例如, 在屏幕较小的设备型号中使用较大字体时, 可能会触发此错误并影响后续用户的操作. 注意, 对于文本重叠类别, 两段文本彼此交叠在一起. 而对于组件遮挡, 一个组件覆盖另一个组件的一部分.

© 中国科学院软件研究所 http://www.jos.org.cn

(a) 组件遮挡缺陷 (b) 文字交叠缺陷 (c) 文字缺失缺陷 (d) 组件溢出缺陷

图 6 交叠与遮挡缺陷样例

(3) 文字缺失缺陷

由于移动设备为用户提供了显示大小和字体大小的自定义设置, 当用户设置最大显示和最大字体时可能会导致文本超出组件的范围进而无法显示. 如图 6(c) 所示, 该缺陷通常表现为文本信息部分缺失, 它通常与 TextView 组件一起出现. 此类型缺陷严重影响用户对组件想表达语义的理解, 影响用户体验感. 产生主要原因如下: 组件高度和字体的自适应设置不当.

(4) 组件溢出缺陷

与文字缺失出现的位置不同, 组件溢出缺陷不是出现在组件内部, 而是由于布局设置的原因导致组件溢出界面显示区域. 如图 6(d) 所示, 该缺陷通常表现为组件超出界面显示的区域导致组件显示不全, 它通常与 TextView, Button 或 EditText 等 UI 组件一起出现. 此类型缺陷往往会影响用户的正常使用, 导致部分组件无法进行交互操作. 产生原因是应用程序界面布局设置导致用户设置较大显示尺寸时引发的自适应问题.

● 丢失与错误信息缺陷

根据众测数据集的统计结果显示有 32% 的缺陷截图属于丢失与错误信息缺陷类型. 如图 7 所示, 这些缺陷主要表现为图像显示组件或文本显示组件存在信息缺失或者显示乱码的情况, 影响用户获取页面上 UI 组件所提供的信息.

(a) 图片丢失缺陷 (b) 空值问题缺陷 (c) 文本乱码缺陷

图 7 丢失与错误信息缺陷样例

(1) 图片丢失缺陷

与自适应设置所导致的交叠与遮挡缺陷不同, 图片丢失缺陷主要发生在图像显示组件 ImageView 中. 如图 7(a) 所示, 图像未能正确显示在开发人员设计的图像显示区域, 显示为空白或图像显示失败的图标. 可能的原因如下: 错误的图像路径或布局位置、由于权限导致的配置文件加载失败、图像过大、网络连接、代码逻辑或图片错误等.

(2) 空值问题缺陷

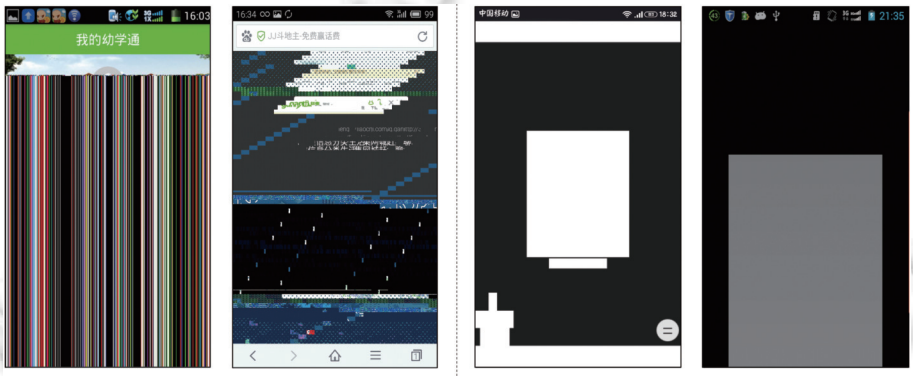
与文字缺失缺陷不同, 空值问题缺陷主要是无法读取数值导致. 如图 7(b) 所示, 文本显示组件没有显示正确的信息, 而是在相应的区域显示了 NULL. 这类错误通常发生在 TextView 中. 主要原因如下: 参数设置或数据库读取问题, 文本长度超过阈值等.

(3) 文本乱码缺陷

由于移动应用程序存在多语言的设置, 不同的语言尤其对应的编码格式, 例如 GBK, UTF-8 等. 如图 7(c) 所示, 该缺陷通常变现为组件的文本出现乱码的情况, 它通常出现在 TextView, Button 等 UI 组件. 产生主要原因是由于不同的编码格式导致的显示乱码问题.

● 显示与加载失败缺陷

根据众测数据集的统计结果显示有 20% 的缺陷截图属于显示与加载失败缺陷类型. 如图 8 所示, 这些缺陷主要表现为 Android 界面元素显示出现花屏问题或者无法加载.



(a) 花屏显示缺陷 (b) 加载失败缺陷

图 8 显示与加载失败缺陷样例

(1) 花屏显示缺陷

由于不同设备的配置存在差异, 部分需要渲染的组件往往需要设备硬件进行渲染加速. 如图 8(a) 所示, 该缺陷通常表现为屏幕显示为花屏. 此缺陷产生的原因可能是硬件设备硬件图像渲染性能存在缺陷, 或者某些 CPU 或 GPU 要求的功能未使用硬件加速设置.

(2) 加载失败缺陷

与花屏显示缺陷不同, 该主要体现为白屏, 灰屏或黑屏. 如图 8 (b) 所示, 该缺陷通常表现为由于信息加载失败导致的白屏, 黑屏或显示加载失败. 此类型缺陷产生主要原因如下: 由于网络问题或者数据库连接问题导致的页面元素加载失败的问题.

下面分析用户界面显示缺陷在新 APP 的分布情况, 我们从众测数据集的分析结果中获得了 3 个大类和 9 个子类的用户界面显示缺陷分类以及在他们在数据集的分布情况. 为了进一步了解用户界面显示缺陷在 2017 年后新 APP 上的出现情况, 我们根据缺陷类型统计了从 2017–2021 年的 GitHub 数据集中用户界面显示缺陷的分布情况. 如表 2 所示, 统计结果显示用户界面显示缺陷在 GitHub 数据集中新 APP 的缺陷分布的情况与众测数据集类似. 其中交叠与遮挡类缺陷在 GitHub 数据集中依然占有最大的比例, 69% 的缺陷截图表现为 UI 组件之间出现交

叠或者遮挡. 该类缺陷的 3 个子类, 组件遮挡缺陷, 文字交叠缺陷, 文字缺失缺陷, 组件溢出缺陷分别占 29%、18%、12%、10%. 其次数据集中有 27% 的截图存在丢失与错误信息缺陷, 主要体现在图片和文本的缺失或乱码. 该缺陷存在 3 个子类, 图片丢失缺陷, 空值问题缺陷, 文本乱码缺陷分别占 15%、8%、4%. 有 4% 的缺陷截图存在显示与加载失败缺陷, 分别体现为花屏显示缺陷, 加载失败缺陷, 分别占 2%、2%.

表 2 用户界面显示缺陷的种类与其在 GitHub 数据集的分布情况

类别		数量	比例 (%)
交叠与遮挡缺陷	组件遮挡缺陷	295	29
	文字交叠缺陷	183	18
	文字缺失缺陷	121	12
	组件溢出缺陷	102	10
丢失与错误信息缺陷	图片丢失缺陷	152	15
	空值问题缺陷	81	8
	文本乱码缺陷	41	4
显示与加载失败缺陷	花屏显示缺陷	24	2
	加载失败缺陷	16	2

相较于 2017 年前众测数据集中流行的 APP, 2017 年后 GitHub 数据集的开源 APP 的缺陷报告中依然存在用户界面显示缺陷, 并且交叠与遮挡缺陷的比例有所提升 (69% vs. 48%). 通过进一步分析产生缺陷的 APP, 我们发现 Android 操作系统版本增多以及 APP 文本数据的增多加剧了交叠与遮挡缺陷的产生. 同时我们发现开源 APP 的开发者较少关注不同屏幕分辨率与 Android 操作系统组合情况下的用户界面显示, 从缺陷报告中可以观察到开发人员在用户提交缺陷报告之前没有考虑到会出现交叠与遮挡缺陷的情况, 进一步证明了本文经验研究的必要性. 针对丢失与错误信息缺陷和显示与加载失败缺陷, 统计结果显示以上两类缺陷比例有所下降, 我们进一步对存在缺陷的 APP 进行分析并总结了以下 3 个原因: (1) GitHub 数据集的开源 APP 使用的 GUI 组件类型较为单一, 以 TextView 为主, 较少涉及图像与视觉效果相关的组件. (2) GitHub 数据集的开源 APP 中游戏类 APP (使用大量图像渲染组件) 较少, 这些开源的 APP 较少使用复杂的图像渲染技术, 因此触发显示与加载失败缺陷的情况有所减少. (3) GitHub 数据集的开源 APP 相较于众测数据集中的流行 APP, 其功能较为单一并且 GUI 页面的数量相对较少, 进而导致以上两类界面显示缺陷的分布有所降低.

以上关于 2017–2021 年 GitHub 数据集的用户界面显示缺陷分布情况进一步验证了我们总结的 3 个大类和 9 个子类的用户界面显示缺陷在新 APP 上仍然适用, 并且缺陷报告的回复也显示开发人员在设计 GUI 的过程中较少关注用户界面显示的相关问题.

总结: 通过 RQ1 对于众测数据的分析, 我们总结了 3 个大类和 9 个子类的用户界面显示缺陷以及它们的分布情况. 其中交叠与遮挡缺陷所占的比例最大, 主要表现为组件之间的相互遮挡或溢出屏幕和文本出现交叠或者丢失的情况. 通常与用户自定义设置系统显示大小与字体大小、多样的设备型号与操作系统版本、不同的设备屏幕分辨率等因素有关. 其次是丢失与错误信息缺陷, 主要表现为图像丢失或文本获取失败与乱码的情况. 通常与数据库等资源链接相关. 最后是显示与加载失败缺陷. 主要表现为设备出现花屏或 UI 显示加载失败等情况, 主要与设备的性能等因素有关. 我们进一步在 GitHub 数据集上 2017 年后的新 APP 上验证了缺陷的分类与分布情况. 接下来我们将在 RQ2 进一步分析用户界面显示的本质成因与缺陷代码.

3.2 RQ2: 用户界面显示缺陷的成因是什么?

根据第 3.1 节中总结的 3 种类型用户界面显示缺陷 (交叠与遮挡缺陷、丢失与错误信息缺陷、显示与加载失败缺陷) 以及他们的特点, 我们进一步运用第 2.2 节中的人工标注方法从 GitHub 数据集中的缺陷报告与其对应的缺陷代码, 总结了如表 3 所示 3 种类型的用户界面显示缺陷的根本原因. 接下来我将分别对这 3 类缺陷的成因进行介绍.

© 中国科学院软件研究所 http://www.jos.org.cn

表 3 用户界面显示缺陷的成因分析与分布情况

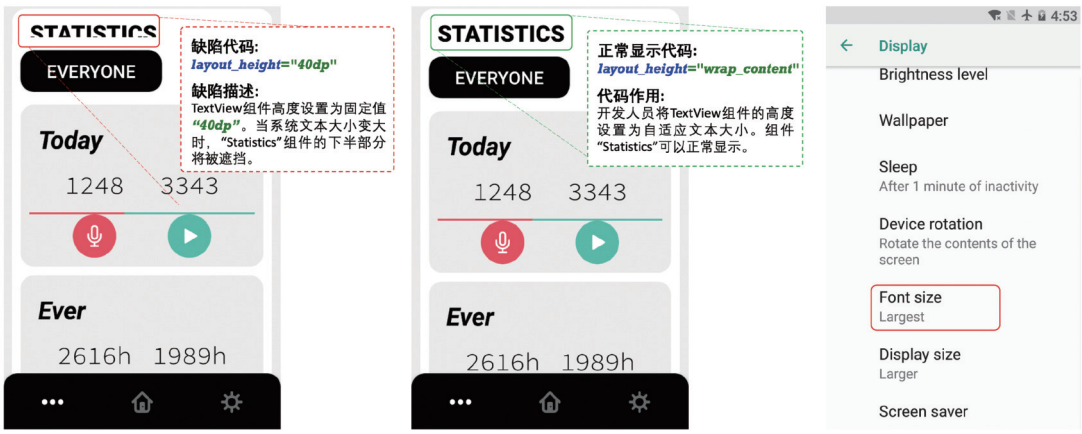
类别		数量	比例 (%)
交叠与遮挡缺陷	组件未进行自适应设置	284	28
	绝对布局未设置页面滚动	183	18
	字体像素密度设置不当	132	13
	窗口小部件版本过期	61	6
	动态组件生成的代码逻辑错误	39	4
丢失与错误信息缺陷	外部资源错误	152	15
	过期的API/SDK版本	41	4
	编码问题	43	4
	条件语句问题	22	2
	内部数据库读取问题	19	2
显示与加载失败缺陷	硬件加速未开启	40	4

● 交叠与遮挡缺陷的产生原因

统计结果显示有 69% 的缺陷报告属于交叠与遮挡缺陷类型. 这些缺陷可以通过将系统的字体大小 (font size) 或系统的显示大小 (display size) 设置为最大 (largest) 来进行复现. 此类个性化设置主要针对用户希望应用程序的内容显示满足他们的设置需求. 交叠与遮挡缺陷主要出现在 TextView、EditText 和 Button 等用于显示文本的组件中. 我们总结了 5 类根本原因, 接下来将进行详细介绍.

(1) 组件未进行自适应设置

统计数据显示, 有 28% 的交叠与遮挡缺陷是由应用程序的 UI 组件边界大小未进行自适应设置引起的 (例如, TextView, EditText), 分为两个子类别. 首先, 如果 UI 组件应用固定边界设置, 则它无法适应变化的文本大小 (例如, 当用户自定义设置 font size 为 Largest). 具体来说, 开发人员将 TextView 的 layout_height (组件高度) 属性设置为固定值 (layout_height = “30dp”). 当文本大小因用户自定义设置或更改屏幕大小而增加时, 文本将超过组件的高度, 从而导致遮挡或交叠. 如图 9 所示, 应用程序设置了固定的 layout_height 属性, 当调整了系统的文本字体大小 (font size) 后所导致的用户界面显示缺陷. 常见的缺陷修复方法是将 layout_height 属性设置为 wrap_content (控件大小自适应文本).



(a) 存在界面显示缺陷的截图 (b) 正常显示的截图 (c) 系统字体大小设置

图 9 组件未进行自适应设置示例

其次, 除了 UI 组件的 layout_height 属性为设置自适应文字, 组件内文本的对齐设置的自适应也会导致界面显示缺陷的产生, 例如, TextView 组件中的 gravity (组件内容位置) 属性没有设置为居中 (例如设置为 gravity =

“right”), 或者 layout_marginTop 属性 (组件内文本与组件边框的距离) 设置的较小 (layout_marginTop = “10dp”), 这也可能导致当系统文本字体大小 (font size) 更改时出现交叠与遮挡缺陷. 常见的缺陷修复方法是将 gravity 属性设置为 center (居中) 或将 layout_marginTop 值调整超过 15 dp.

最后, Android 也为开发人员提供了可以根据组件大小调整文本字体大小的属性 autoSizeTextType, 如果将其关闭 (autoSizeTextType = “none”) , 文本将不会自动调整, 进而导致交叠与遮挡缺陷的产生. 常见的缺陷修复方法是将 autoSizeTextType 属性设置为 uniform (均匀缩放) .

典型的缺陷代码例子如下.

GPStest	#467:	*android:layout_height* = “30dp”
Instead	#213:	*android:gravity* = “right”
Peertube	#115:	*android:layout_marginTop* = “10dp”
TrailSense	#29:	*android:autoSizeTextType* = “none”

(2) 绝对布局未设置页面滚动

统计数据显示, 有 18% 的交叠与遮挡缺陷是由于应用程序布局方式采用了绝对布局设置并没有设置内容可随页面滚动引起的, 例如应用程序采用绝对布局方式 LinearLayout, 并且没有设置滚动视图并添加滚动条 ScrollView/HorizontalScrollView. 如图 10 所示, 当用户设置系统的显示大小 (display size) 设置为最大 (largest) 且页面进行滚动时, 应用程序当前的 UI 页面底部的 UI 组件和内容很容易被遮挡, 尤其是当 UI 页面设计为承载密集组件时. 如图 10 所示, 应用程序未添加滚动条导致下半部分内容被裁减. 常见的缺陷修复方法是将 LinearLayout 调整为 ScrollView (滚动式) .

(a) 存在界面显示缺陷的截图 (b) 正常显示的截图 (c) 系统字体大小设置

图 10 绝对布局未设置页面滚动

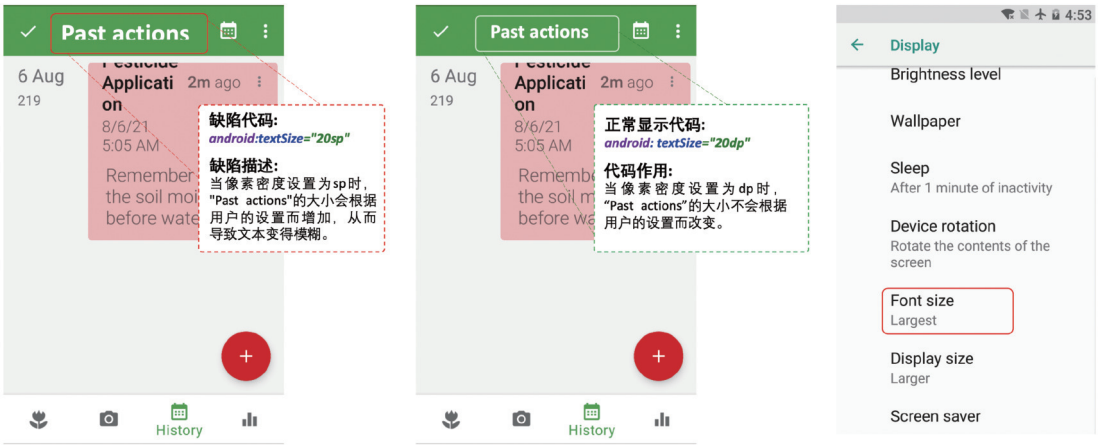
典型的缺陷代码例子如下.

RSAndroid	#102:	- *<Linearlayout ...> ... </Linearlayout>*
		+ *<ScrollView ...>*
		+ *<Linearlayout ...> ... </Linearlayout>*
		+ *</ScrollView>*

(3) 字体像素密度设置不当

统计数据显示, 有 13% 的交叠与遮挡缺陷是由于文本显示组件的像素密度设置不当引起的. 以 TextView 为

例, 其控制字体大小的属性 textSize 通常会选用像素密度单位 dp 和 sp. 其中 dp (device independent pixels, 设备独立像素) 是一个虚拟像素, 它将自动适应不同的像素密度设备, 而 sp (scale-independent pixels, 缩放独立像素) 将根据用户自定义的文本大小调整大小. 例如, 如图 11 所示, 当应用程序使用 sp 时, 如果用户设置较大的字体大小 (font size 大小为 largest) 文本大小会会随着系统字体的增加而增加, 进而导致文本很容易超过 UI 组件 TextView 的边界造成交叠与遮挡缺陷. 常见的缺陷修复方法是将 sp 调整为 dp.



(a) 存在界面显示缺陷的截图 (b) 正常显示的截图 (c) 系统字体大小设置

图 11 字体像素密度设置不当

典型的缺陷代码例子如下.

Look4Sat	#23:	*android:textSize* = “30sp”

(4) 窗口小部件版本过期

统计数据显示, 有 6% 的交叠与遮挡缺陷是由于应用程序采用了过期的窗口展示小部件. 开发人员经常使用 GitHub 上开源的小部件 (例如 simpledialogfragments) 来丰富视觉效果, 但某些小部件的旧版本可能无法适应市场上的新设备, 并导致 UI 显示问题. 常见的缺陷修复方法是升级窗口小部件的版本.

典型的缺陷代码例子如下.

PDroidRt	#138:	*github.eltos:simpledialogfragments:v*3.3
EinkBro	#112:	*DialogFragment v*1.2
OSMTracker	#72:	*com.github.AppIntro:AppIntro*:4.2.3

(5) 动态组件生成的代码逻辑错误

统计数据显示, 有 4% 的交叠与遮挡缺陷是由于应用程序采用了动态生成组件, 其生成的循环或条件语句存在代码逻辑错误. 当开发人员以动态方式生成组件以适应不同的应用场景时, 代码逻辑错误会导致生成的组件过多, 组件之间会发生重叠. 此类缺陷需要调整代码逻辑并进行修复.

典型的缺陷代码例子如下.

Look4Sat	#23:	for 循环语句逻辑错误
Altlaunch	#75:	if 判断语句逻辑错误

- 丢失与错误信息缺陷的原因

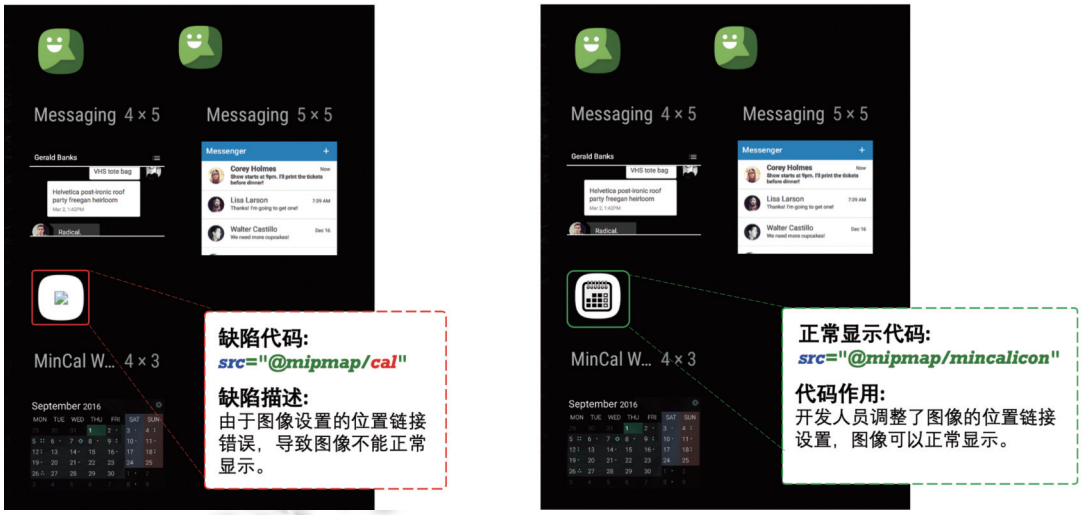
我们观察到 27% 的缺陷报告描述了丢失与错误信息缺陷. 这些缺陷出现在图像显示组件 ImageView 和文本

© 中国科学院软件研究所 http://www.jos.org.cn

显示组件 TextView 组件中. 此类缺陷主要由于资源链接引用错误等原因, 我们总结了导致缺陷的 5 类原因, 接下来将进行详细的介绍.

(1) 外部资源错误

统计数据显示, 有 15% 的丢失与错误信息缺陷是由于应用程序引用了错误的外部资源, 例如图像的位置链接、URL 网址链接等. 如果应用程序由于资源位置不正确而无法加载正确的信息或引用的图像资源本身存在错误无法显示, 则会出现图像丢失或信息缺失的缺陷. 如图 12 所示, 应用程序引用了错误的图像链接导致显示失败. 常见的缺陷修复方法是设置正确的外部链接.



(a) 存在界面显示缺陷的截图 (b) 正常显示的截图

图 12 外部资源错误导致的缺陷

典型的缺陷代码例子如下:

bitcoin	#47:	*android:icon* = “xxxxx”
Instead	#213:	*android:roundIcon* = “xxxxxx”
Peertube	#115:	*Imagepath* = “xxxxxx”

(2) 过期的 API/SDK 版本

统计数据显示, 有 4% 的丢失与错误信息缺陷是由于应用程序采用了过时的 API/SDK 版本. 为了获得更好的视觉效果, 开发人员倾向于使用第三方 API 或 SDK 来帮助显示图像. 例如开发人员经常使用的开源图像处理与显示库 OpenCV. 然而, 随着更高版本的 Android OS 上市, 一些过时的 API/SDK 将在图像渲染过程中导致显示失败. 常见的缺陷修复方法是升级 API 和 SDK 的版本.

典型的缺陷代码例子如下.

cetool	#81:	*opencv* 3.3.1

(3) 编码问题

统计数据显示, 有 4% 的丢失与错误信息缺陷是由于应用程序的文本编码错误和特殊字符处理不当引起的, 通常出现在文本显示组件 TextView 中. 例如, 开发人员使用可以显示中文的本地编码 GBK. 然而, 当该设备将其语言设置为法语时, 文本就会出现乱码的情况. 此外, 还有一些特殊字符需要转义字符“%”才能正确显示. 缺少转义字符可能会导致显示问题. 常见的缺陷修复方法是设置正确的编码格式或添加转义字符.

© 中国科学院软件研究所 http://www.jos.org.cn

典型的缺陷代码例子如下.

UMLCla	#23:	*String.getBytes*(“GBK”)
slimlaun	#19:	“%”未添加转义字符

(4) 条件语句问题

统计数据显示, 有 2% 的丢失与错误信息缺陷是由于应用程序中需要读取信息的条件语句中的代码逻辑错误. 由于判断条件出现错误, 组件的信息获取无法正常工作. 此类缺陷需要调整代码逻辑并进行修复.

典型的缺陷代码例子如下.

Planisph	#53:	if 判断语句逻辑错误导致读取失败

(5) 内部数据库读取问题

统计数据显示, 有 2% 的丢失与错误信息缺陷是由于应用程序的内部数据库读取失败导致的. 大多数应用程序均采用了内部数据集用于存放应用应用程序的相关信息, 当文本组件访问数据库时, 如果连接失败则无法获获取内容, 进而呈现空值. 此类缺陷需要调整数据库读取语句或检查数据库链接以进行修复.

● 显示与加载失败缺陷

我们观察到 4% 的缺陷报告描述了显示与加载失败缺陷. 这些缺陷主要由于应用程序的硬件设备的性能差异导致的, 接下来将进行详细的介绍.

(1) 硬件加速未开启

统计数据显示, 有 4% 的显示与加载失败缺陷是由于应用程序的 Android 硬件加速没有开启. 许多应用程序可以通过使用设备 GPU 获得良好的图形渲染流畅性, 这需要开启硬件加速属性 (hardwareAccelerated) . 当它关闭时, 如下面的示例代码所示, 设备屏幕很容易在应用程序的用户界面渲染过程中出现卡顿, 黑屏或者白屏的情况, 导致屏幕无法正常使用. 常见的缺陷修复方法是开启硬件加速设置.

典型的缺陷代码例子如下.

Foreca	#12:	*<application android:hardwareAccelerated*=“false”>
DSAA	#155:	*<activity android:hardwareAccelerated*=“false” />

总结: 通过 RQ2 对于 GitHub 数据的分析, 我们总结了 3 个类用户界面显示缺陷的本质成因以及它们的分布情况. 其中交叠与遮挡缺陷所占的比例最大, 主要是由不适应或不灵活的边界和布局设置引起的. 其次是丢失与错误信息缺陷, 主要是因为外部资源错误. 此外, 这两个问题都可能是由应用程序的小部件或 API/SDK 的过时版本以及代码逻辑错误 (主要是在条件或循环语句中) 引起的. 最后是显示与加载失败缺陷. 主要是由于硬件加速配置造成的.

3.3 RQ3: 用户界面显示缺陷呈现哪些特征?

根据我们对于 RQ1 的研究结果我们发现, 用户界面显示缺陷存在一定的视觉特征. 如图 5 中各个类型的用户界面显示缺陷所示, 每种类型的用户界面显示缺陷产生于应用程序界面渲染的过程中并在界面上会产生缺陷区域. 根据我们对于 RQ2 的研究结果我们发现, 部分用户界面显示缺陷的触发存在一定的条件. 例如, 交叠与遮挡缺陷往往在用户自定义系统显示大小或文字字体大小的情况下触发, 在默认设置下触发的较少. 显示与加载失败缺陷通常受到设备型号, 操作系统版本, 设备硬件性能等因素影响. 由于应用程序用户界面渲染是一个动态的过程, 较难通过静态分析技术来检测用户界面显示缺陷. 考虑到用户界面显示缺陷在截图中存在缺陷区域并存在视觉特征, 则可以通过计算机视觉的方法[9,14,27]对用户界面显示缺陷进行检测. 接下来我们将根据 RQ1 与 RQ2 中获取的缺陷截图数据, 按照类别总结用户界面显示缺陷的视觉特征. 表 4 展示了各类别缺陷的视觉特征以及分布情况.

表 4 用户界面显示缺陷的视觉特征与其分布情况

缺陷类别	特征类型	数量	比例 (%)	缺陷检测方法
交叠与遮挡缺陷	显式视觉特征	2 788	36	检测组件交叠与遮挡的区域
	隐式视觉特征	1 162	15	对比正常显示大小的截图
丢失与错误信息缺陷	加载失败icon	775	10	检测加载失败的icon
	显示空白	465	6	检测空白区域的边界
	文本为null	465	6	分析view hierarchy检测null
	null的变体	310	4	检测null的变体 (NULL, nullnull)
	文本为乱码	465	6	静态分析view hierarchy检测乱码
显示与加载失败缺陷	大面积色块	620	8	检测截图中色块的区域
	加载竖线	308	4	检测截图中是否存在加载竖线
	白屏显示	155	2	检测截图中的大块白色区域
	灰屏显示	155	2	检测截图中的大块灰色区域
	黑屏显示	77	1	检测截图中的大块黑色区域

● 交叠与遮挡缺陷特征

交叠与遮挡缺陷可以分为有显式视觉特征和隐式视觉特征两大类, 首先显式视觉特征 (36%), 主要包括组件遮挡缺陷和文字交叠缺陷, 他们均有可以检测的视觉特征. 例如组件遮挡缺陷通常会存在组件中文字内容被裁减, 文字的上半部分或下半部分被遮挡, 可以通过检测被裁减的文字进行获取缺陷组件; 文字交叠缺陷通常会存在文本显示组件之间互相交叠, 有较为明显的交叠区域, 可以通过分析两组件之间是否交叠区域进行检测. 其次是隐式视觉特征 (15%), 主要包括文字缺失缺陷和组件溢出缺陷, 他们通常会存在文字或者组件的缺失. 现有用户界面显示缺陷检测工具 OwlEye 难以对其进行检测, 因为组件或者文字的缺失不会在截图上留下信息, 但是分析文本的语意可以发现存在文本缺失的情况. 对于他们的检测方法可以通过对比默认设置下用户界面显示的截图与增加系统显示大小或文字字体大小情况下的应用程序截图是否存在文字或者组件的缺失进行检测.

针对以上交叠与遮挡缺陷的特征, 我们进一步给出两种可行的缺陷检测方法. (1) 采用基于深度学习的目标检测方法, 检测缺陷交叠区域与遮挡的特征. 该缺陷出现在 TextView、EditText、Button 或 ImageView 中, 并且缺陷出现的位置是随机的. 考虑到该部分缺陷区域的像素值有明显的变化, 深度学习模型可以根据像素的变化情况检测到缺陷的特征区域. 对于模型的训练可以生成一个颜色块, 其宽度和背景颜色与组件相同, 但高度较小, 然后将其随机覆盖组件的一部分来模拟缺陷, 或者采用文本覆盖 TextView 来模拟缺陷, 进而自动化生成训练数据. (2) 采用静态分析 view hierarchy file 的方法, 通过对比正常显示情况下的 GUI 页面与用户自定义设置后的 GUI 页面检测隐式的交叠与遮挡缺陷. 具体来说, 许多应用程序的 GUI 仅设计用于字体或显示大小的默认设置, 当从默认设置放大到更大的字体或显示大小时会导致 GUI 不一致进而产生交叠与遮挡缺陷. 因此可以采用对比默认显示比例下的 GUI 页面和用户自定义显示比例下对应的 GUI 页面的方法, 将每个视图与其对应的视图进行映射以更好地进行后续缩放检测, 然后分别从视图和单个视图之间的角度进行检测. 首先检测视图间不一致, 检测 GUI 页面中的两个视图之间是否存在任何重叠/裁剪, 并检查两个页面的重叠/裁剪状态是否保持一致. 然后是视图内不一致性检测, 检查每个视图及其对应视图的不一致性来判断是否存在缺陷.

● 丢失与错误信息缺陷特征

丢失与错误信息缺陷拥有可以检测的显式视觉特征 (32%). 具体来说, 图片丢失缺陷通常体现为图像显示组件无法正确显示图像内容, 取而代之的是图像加载失败的 icon (10%) 或者空白 (6%), 其中图像加载失败的 icon 较为固定, 通过我们 RQ1 中的研究结果, 常见的 icon 有 8 种, 因此可以通过检测截图中图像加载失败的 icon 来检测缺陷; 空值问题缺陷通常体现为文本显示组件显示的值为 null (6%) 或 null 的变体 (4%) (例如 NULL, nullnull 等), 可以通过静态分析当前页面的 view hierarchy file 并检测文本显示组件 TextView 是否显示为 null. 文本乱码缺陷通常会表现为文本显示组件输出乱码 (6%) (例如□, δ̄, ?等常见乱码), 可以通过静态分析当前页面的 view hierarchy file 并检测文本显示组件是否存在乱码符号.

© 中国科学院软件研究所 http://www.jos.org.cn

针对以上丢失与错误信息缺陷的特征, 我们进一步给出两种可行的缺陷检测方法. (1) 采用基于深度学习的目标检测方法, 检测图像加载失败的 icon 或者空白区域. 由于该部分缺陷区域的有较为固定的视觉特征 (图像加载失败的 icon 或空白区域), 深度学习模型可以检测到这些图像相关的特征. 这些图像加载失败的 icon 与真实世界的图像截然不同, 这些表示缺陷图像的 icon 很少在应用程序的 GUI 页面中使用 (大多数 GUI 设计与开发人员认为这些 icon 是表示为存在缺陷的 icon), 因此检测结果大多是存在用户界面显示缺陷. 对于模型的训练可以通过收集常见的图像加载失败的 icon 生成训练数据, 然后用一个随机选择的图像加载失败的 icon 覆盖原始图像显示区域, 并将其背景颜色设置为原始图像的颜色. (2) 采用静态分析 view hierarchy file 的方法, 使用关键词匹配技术来检测 GUI 页面上的 TextView 组件是否存在 null 或 null 的变体. 并且遍历 view hierarchy file 检测其是否存在乱码字符.

● 显示与加载失败缺陷特征

显示与加载失败缺陷拥有可以检测的显式视觉特征. 具体来说, 花屏显示缺陷通常体现在屏幕显示部分出现大面积色块并且色块无序排布 (8%) , 或者文本显示组件引出较长的加载竖线 (4%) , 可以通过检测屏幕截图中的花屏色块或加载竖线进行缺陷检测. 加载失败缺陷通常体现为屏幕上的组件或布局呈现白屏 (2%) , 灰屏 (2%) 或黑屏 (1%) 的情况, 可以通过检测屏幕上是否存在组件为白屏, 灰屏或黑屏来检测缺陷.

针对以上显示与加载失败缺陷的特征, 我们进一步给出如下可行的缺陷检测方法. 采用基于深度学习的目标检测方法, 检测花屏区域或白屏区域. 由于该部分缺陷区域的与其周围组件存在较为明显的像素差异, 深度学习模型可以检测到这些缺陷图像相关的特征. 对于模型的训练可以通过收集常见的花屏问题的图片, 然后用一个随机选择的花屏图片覆盖应用程序的图像区域, 进而自动生成大量的训练数据.

总结: 通过 RQ3 对于用户界面显示缺陷截图数据的分析, 我们总结了 3 个类用户界面显示缺陷的视觉特征. 其中交叠与遮挡缺陷主要存在显式的视觉特征和隐式的视觉特征. 丢失与错误信息缺陷与显示与加载失败缺陷有较为明显的显式视觉特征. 显示视觉特征可以通过分析缺陷在截图上的区域进行缺陷检测, 隐式特征可以通过对比正常显示下的截图与用户自定义设置之后的截图之间的差异进行检测.

3.4 结果启示

本节讨论 RQ1, RQ2, RQ3 的研究结果给测试人员, 开发人员和自动化测试工具带来的启示.

● 测试人员

RQ1 总结了 3 大类以及 9 个子类的用户界面显示缺陷, 研究结果可以为测试人员提供一些常见的界面显示缺陷. 测试人员在测试过程中应关注由于用户自定义设置而导致的遮挡与交叠缺陷, 此类缺陷会影响应用程序的可访问性和可用性.

● 开发人员

RQ2 总结了 3 大类用户界面显示缺陷的本质成因, 研究结果可以为开发人员提供一些 UI 设计过程中的参考意见. 具体来说, 在 UI 设计过程中, 开发人员应尽量设置组件的边界为自适应, 减少设置固定的组件大小或文字大小, 这也符合 Google 官方给出的 Android 开发建议[28–30]. 其次在使用绝对布局时, 尽可能地为布局设置滚动条, 以便用户在使用过程中可以通过滑动屏幕访问信息. 在编码格式方面尽可能注意不同语言对应的编码格式. 最后针对部分对硬件要求过高的应用程序可以通过开启硬件加速来接住硬件设备提升应用程序的 UI 渲染.

● 自动化缺陷修复工具

根据 RQ2 总结的 3 大类用户界面显示缺陷的本质成因以及修复代码, 我们观察到用户界面显示缺陷存在一定的缺陷修复模式. RQ2 的经验研究结果可以为自动化缺陷修复工具的设计提供参考意见. 具体来说, 针对交叠与遮挡缺陷, 当缺陷产生原因是未开启自适应设置时, 则可以设计自动缺陷修复脚本来修改组件的自适应属性为文字自适应组件并完成修复. 针对丢失与错误信息缺陷特征, 当缺陷产生的原因是编码格式错误, 则可以设计自动缺陷修复脚本来更改代码的编码属性为 UTF-8. 针对显示与加载失败缺陷特征, 可以设计自动化缺陷修复脚本开启硬件加速.

● 自动化测试工具

RQ3 总结了 3 大类用户界面显示缺陷的特征, 包括一些显式特征和隐式特征. 对于拥有显式特征的用户界面

© 中国科学院软件研究所 http://www.jos.org.cn

显示缺陷, 自动化测试工具可以通过检测缺陷区域来进行缺陷检测并定位缺陷组件[27]. 对于拥有隐式特征的用户界面显示缺陷, 自动化测试工具可以通过对比系统默认设置下的界面组件与文本和用户自定义设置下的组件与文本的不同进行检测, 判断是否出现文字缺失缺陷和组件溢出缺陷.

3.5 有效性影响因素分析

● 内部有效性

本文中对于用户界面显示缺陷的分类过程是人工进行的, 为了避免分类过程存在偏差, 每位作者都认真检查了缺陷报告的缺陷描述, 报告中描述用户界面显示缺陷的关键词以及缺陷截图. 同时人工分类过程是由本文的第 1 位和第 4 位作者 (均拥有 3 年以上 Android 开发经验), 但是他们并不是 Android 领域专家或 UI 设计师, 如果分类过程由领域专家来进行, 结果可能会不同.

● 外部有效性

本文选取了百度众测平台上的测试报告与 GitHub 网站上的开源 Android 项目, 以上两个数据源无法代表所有类型的应用程序. 本文研究的用户界面显示缺陷的本质成因中, 选用的为开源的应用程序, 虽然大多数在 Google Play 应用商店上发行过, 但是与部分闭源的应用程序存在差异, 因此本文的结论并不一定适用于闭源的应用程序, 未来会从闭源的应用程序角度出发进一步分析导致用户界面显示缺陷产生的本质原因. 此外, 由于百度数据集的闭源性也导致我们无法直接从应用程序的源代码来分析缺陷产生的原因, 因此我们需要引入 GitHub 数据集来进行成因的分析.

4 相关工作

本节简要介绍了用户界面显示缺陷的相关研究. 应用程序的图形用户界面 (GUI) 为用户和终端设备提供了一个可视化的桥梁, 方便用户使用应用程序[20,26−34]. 如何保证应用程序图形用户界面的质量也成为近年来的一个热点研究[7−10,12,27,33,35]. 因此, 许多研究人员提出了基于图像特征的图形用户界面检索方法, 协助开发人员或设计师进行图形用户界面搜索[36−40], 进而自动生成相应的 UI 代码来辅助 UI 设计师设计应用程序的图形用户界面[41−48]. Chen 等人[49]设计了一种基于深度神经网络的方法, 用于对视觉和文本信息进行编码, 以恢复现有 UI 示例的缺失标记, 从而更容易通过文本查询找到它们. Chen 等人[50]研究了基于深度学习的对象检测方法在检测 UI 组件方面的局限性和有效设计. 此外, 他们还设计了一种从粗到细的自顶向下的新策略, 并将其与成熟的 GUI 文本深度学习模型相结合. Moran 等人[51,52]通过比较图像相似性与计算机视觉技术, 检查实现的 GUI 是否违反了原始 UI 设计. 他们的后续工作进一步检测并总结了不断发展的移动应用程序中的 GUI 变化. 与这些工作不同, 我们的工作重点是分析与的应用程序的用户界面显示缺陷, 以帮助开发人员提高应用程序质量.

为了确保应用程序的图形用户界面的质量, 研究人员尝试使用静态分析方法[53−55]来识别代码中的 bug、风格错误和可疑构造, 其中 Android Lint 报告了 260 多种不同类型的 Android 错误, 包括正确性、性能、安全性、可用性和可访问性. StyleLint 帮助开发人员避免错误并在样式中强制执行约定[56]. Gao 等人[10]、Li 等人[11]和 Luo 等人[57]分析了 UI 呈现中可能出现的问题, 并开发了自动检测这些问题的方法. Nayebi 等人[58]发现, 移动设备的不同分辨率给 Android 应用程序的设计和实现带来了挑战. Liu 等人[12]提出了用户界面显示缺陷的自动化检测方法 OwlEye, 通过卷积神经网络来检测用户界面显示缺陷. OwlEye 主要针对 5 类显示缺陷的检测, 本文主要开展 9 类用户界面显示缺陷的种类与本质成因的经验研究, 经验研究的结论也可以进一步提升 OwlEye 的检测范围与精度.

此外, 由于 Web 应用程序也可以在具有各种视窗宽度的设备上运行, 因此 Web 领域也有研究尝试保证应用程序的 UI 质量[59−66]. Kondratova 等人[59]探索了世界各地不同国家和文化的用户界面设计偏好. WebSee[60,61]和 FieryEye[62]利用计算机视觉和图像处理技术将浏览器呈现的测试网页与 Oracle 图像进行比较, 以找出视觉差异. Mahajan 等人[63]研究了网页中移动友好问题的自动检测. Mahajan 等人[64]还研究了由于网站在不同浏览器之间呈现不一致而导致的布局跨浏览器问题. 与 Web 应用程序不同, Android 应用程序由于更复杂的渲染机制, 多样设备与分辨率导致其产生更多类型的用户界面显示缺陷显示问题. 因此, 本文研究并分析了现有移动应用的成因与特征.

© 中国科学院软件研究所 http://www.jos.org.cn

5 总 结

移动应用程序的图形用户界面构建了应用程序与用户间交互的可视化桥梁, 复杂的界面设计以及多样的移动设备与用户自定义设置也对界面显示的正确性带来巨大的挑战. 本文研究了移动应用程序的用户界面显示缺陷的类型, 成因以及特征. 收集了来自众测平台的 6729 个用户界面显示缺陷截图和 GitHub 的应用程序缺陷报告提供的 1016 个用户界面显示缺陷截图, 并用主题分析方法识别出 3 大类和 9 个子类用户界面显示缺陷. 然后根据从 GitHub 收集的 1016 个缺陷报告与代码分析了用户界面显示缺陷的本质原因. 根据研究中的发现总结出缺陷的显式视觉特征和隐式视觉特征可以为自动化检测缺陷提供指导. 未来工作可以进一步研究用户界面显示缺陷的修复技术并开发相应的工具, 此外未来的研究工作可以进一步探究动态渲染阶段的渲染缺陷.

References:

[1] Jansen BJ. The graphical user interface. ACM SIGCHI Bulletin, 1998, 30(2): 22–26. [doi: 10.1145/279044.279051]

[2] Wei LL, Liu YP, Cheung SC. Taming Android fragmentation: Characterizing and detecting compatibility issues for Android APPs. In: Proc. of the 31st IEEE/ACM Int'l Conf. on Automated Software Engineering. Singapore: ACM, 2016. 226–237. [doi: 10.1145/2970276.2970312]

[3] Di Geronimo L, Braz L, Fregnan E, Palomba F, Bacchelli A. UI dark patterns and where to find them: A study on mobile applications and user perception. In: Proc. of the 2020 CHI Conf. on Human Factors in Computing Systems. Honolulu: ACM, 2020. 1–14. [doi: 10.1145/3313831.3376600]

[4] Li L, Bissyandé TF, Papadakis M, Rasthofer S, Bartel A, Octeau D, Klein J, Traon L. Static analysis of Android APPs: A systematic literature review. Information and Software Technology, 2017, 88: 67–95. [doi: 10.1016/j.infsof.2017.04.001]

[5] Choudhary SR, Gorla A, Orso A. Automated test input generation for Android: Are we there yet?(E). In: Proc. of the 30th IEEE/ACM Int'l Conf. on Automated Software Engineering (ASE). Lincoln: IEEE, 2015. 429–440. [doi: 10.1109/ASE.2015.89]

[6] Ham HK, Park YB. Designing knowledge base mobile application compatibility test system for Android fragmentation. Int'l Journal of Software Engineering and Its Applications, 2014, 8(1): 303–314. [doi: 10.14257/ijseia.2014.8.1.26]

[7] Chen K, Li YF, Chen YF, Fan CJ, Hu ZP, Yang W. GLIB: Towards automated test oracle for graphically-rich applications. In: Proc. of the 29th ACM Joint Meeting on European Software Engineering Conf. and Symp. on the Foundations of Software Engineering. Athens: ACM, 2021. 1093–1104. [doi: 10.1145/3468264.3468586]

[8] Chen S, Fan LL, Chen CY, Su T, Li WH, Liu Y, Xu LH. StoryDroid: Automated generation of storyboard for Android APPs. In: Proc.of the 41st IEEE/ACM Int'l Conf. on Software Engineering (ICSE). Montreal: IEEE, 2019. 596–607. [doi: 10.1109/ICSE.2019.00070]

[9] Chen S, Fan LL, Chen CY, Xue MH, Liu Y, Xu LH. GUI-squatting attack: Automated generation of Android phishing APPs. IEEE Trans. on Dependable and Secure Computing, 2021, 18(6): 2551–2568. [doi: 10.1109/TDSC.2019.2956035]

[10] Gao Y, Luo Y, Chen DQ, Huang HC, Dong W, Xia MY, Liu X, Bu JJ. Every pixel counts: Fine-grained UI rendering analysis for mobile applications. In: Proc. of the 2017 IEEE Conf. on Computer Communications. Atlanta: IEEE, 2017. 1–9. [doi: 10.1109/INFOCOM.2017.8057023]

[11] Li WJ, Jiang YY, Xu C, Liu YP, Ma XX, Lü J. Characterizing and detecting inefficient image displaying issues in Android APPs. In: Proc. of the 26th IEEE Int'l Conf. on Software Analysis, Evolution and Reengineering (SANER). Hangzhou: IEEE, 2019. 355–365. [doi: 10.1109/SANER.2019.8668030]

[12] Liu Z, Chen CY, Wang JJ, Huang YK, Hu J, Wang Q. Owl eyes: Spotting UI display issues via visual understanding. In: Proc. of the 35th IEEE/ACM Int'l Conf. on Automated Software Engineering (ASE). Melbourne: IEEE, 2020. 398–409.

[13] Wang JJ, Yang Y, Wang S, Hu YZ, Wang DD, Wang Q. Context-aware in-process crowdworker recommendation. In: Proc. of the 42nd ACM/IEEE Int'l Conf. on Software Engineering. Seoul: ACM, 2020. 1535–1546. [doi: 10.1145/3377811.3380380]

[14] Martin J. Rapid Application Development. New York: Macmillan Publishing Co. Inc., 1991.

[15] Liu JY, Yu JK. Research on development of Android applications. In: Proc. of the 4th Int'l Conf. on Intelligent Networks and Intelligent Systems. Kuming: IEEE, 2011. 69–72. [doi: 10.1109/ICINIS.2011.40]

[16] Moskala M, Wojda I. Android Development with Kotlin. Birmingham: Packt Publishing Ltd., 2017.

[17] Seaman CB. Qualitative methods in empirical studies of software engineering. IEEE Trans. on Software Engineering, 1999, 25(4): 557–572. [doi: 10.1109/32.799955]

[18] Liu Z, Chen CY, Wang JJ, Che X, Huang YK, Hu J, Wang Q. Fill in the blank: Context-aware automated text input generation for mobile GUI testing. In: Proc. of the 45th IEEE/ACM Int'l Conf. on Software Engineering (ICSE). IEEE. 2023. 1355–1367. [doi: 10.1109/ICSE

© 中国科学院软件研究所 http://www.jos.org.cn

48619.2023.00119]

[19] Braun V, Clarke V. Using thematic analysis in psychology. Qualitative Research in Psychology, 2006, 3(2): 77–101. [doi: 10.1191/1478088706qp063oa]

[20] Rigby PC, German DM, Storey MA. Open source software peer review practices: A case study of the Apache server. In: Proc. of the 30th Int'l Conf. on Software Engineering. Leipzig: ACM, 2008. 541–550. [doi: 10.1145/1368088.1368162]

[21] Wang QY, Wan ZY, Li SP, Xia X. Empirical study on restored code changes in code review. Ruan Jian Xue Bao/Journal of Software, 2022, 33(7): 2581–2598 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/6312.htm [doi: 10.13328/j.cnki.jos.006312]

[22] Zheng W, Tang H, Chen X, Zhang MQ, Xia X. State-of-the-art survey of compatibility test for Android mobile application. Journal of Computer Research and Development, 2022, 59(6): 1370–1387 (in Chinese with English abstract). [doi: 10.7544/issn1000-1239.20210105]

[23] Jiang J, Lv JF, Zhang L. Topic analysis on Chinese programming question and answer websites. Ruan Jian Xue Bao/Journal of Software, 2020, 31(4): 1143–1161 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5987.htm [doi: 10.13328/j.cnki.jos.005987]

[24] Zhang L, Pu MY, Liu YJ, Tian JH, Yue T, Jiang J. Investigation of empirical researches in software engineering. Ruan Jian Xue Bao/Journal of Software, 2018, 29(5): 1422–1450 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5520.htm [doi: 10.13328/j.cnki.jos.005520]

[25] Li C, Jiang YY, Xu C. GUI event-based record and replay technologies for Android APPs: A survey. Ruan Jian Xue Bao/Journal of Software, 2022, 33(5): 1612–1634 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/6551.htm [doi: 10.13328/j.cnki.jos.006551]

[26] Spencer D. Card Sorting: Designing Usable Categories. New York: Rosenfeld Media, 2009.

[27] Meier R. Professional Android 4 Application Development. Hoboken: John Wiley & Sons, 2012.

[28] Liu Z, Chen CY, Wang JJ, Huang YK, Hu J, Wang Q. Nighthawk: Fully automated localizing UI display issues via visual understanding. IEEE Trans. on Software Engineering, 2023, 49(1): 403–418. [doi: 10.1109/TSE.2022.3150876]

[29] Griffiths D, Griffiths D. Head First Android Development: A Brain-friendly Guide. 2nd ed., Sebastopol: O'Reilly Media, Inc., 2017.

[30] Goadrich MH, Rogers MP. Smart smartphone development: IOS versus Android. In: Proc. of the 42nd ACM Technical Symp. on Computer Science Education. Dallas: ACM, 2011. 607–612. [doi: 10.1145/1953163.1953330]

[31] Stephanidis C. User Interfaces for All: Concepts, Methods, and Tools. Boca Raton: CRC Press, 2000. [doi: 10.1201/b12453]

[32] Hearst MA. User interfaces and visualization. In: Baeza-Yates R, ed. Modern Information Retrieval. New York: ACM, 1999. 257–324.

[33] Reiss SP. Seeking the user interface. In: Proc. of the 29th ACM/IEEE Int'l Conf. on Automated Software Engineering. Vasteras: ACM, 2014. 103–114. [doi: 10.1145/2642937.2642976]

[34] Morris J. Android User Interface Development. Birmingham: Packt Publishing, 2011.

[35] Kuznetsov K, Avdiienko V, Gorla A, Zeller A. Analyzing the user interface of Android APPs. In: Proc. of the 5th Int'l Conf. on Mobile Software Engineering and Systems. Gothenburg: ACM, 2018. 84–87. [doi: 10.1145/3197231.3197232]

[36] Behrang F, Reiss SP, Orso A. GUIfetch: Supporting APP design and development through GUI search. In: Proc. of the 5th Int'l Conf. on Mobile Software Engineering and Systems. Gothenburg: ACM, 2018. 236–246. [doi: 10.1145/3197231.3197244]

[37] Chen JS, Chen CY, Xing ZC, Xia X, Zhu LM, Grundy J, Wang JS. Wireframe-based UI design search through image autoencoder. ACM Trans. on Software Engineering and Methodology, 2020, 29(3): 19. [doi: 10.1145/3391613]

[38] Chen CY, Feng SD, Xing ZC, Liu LD, Zhao SD, Wang JS. Gallery DC. Design search and knowledge discovery through auto-created GUI component gallery. Proc. of the ACM on Human-computer Interaction, 2019, 3(CSCW): 180. [doi: 10.1145/3359282]

[39] Zhao DH, Xing ZC, Chen CY, Xia X, Li GQ. ActionNet: Vision-based workflow action recognition from programming screencasts. In: Proc. of the 41st IEEE/ACM Int'l Conf. on Software Engineering (ICSE). Montreal: IEEE, 2019. 350–361. [doi: 10.1109/ICSE.2019.00049]

[40] AYang B, Xing ZC, Xia X, Chen CY, Ye DH, Li SP. UIS-hunter: Detecting UI design smells in Android APPs. In: Proc. of the 43rd IEEE/ACM Int'l Conf. on Software Engineering: Companion (ICSE-Companion). Madrid: IEEE, 2021. 89–92. [doi: 10.1109/ICSE-Companion52605.2021.00043]

[41] Chen CY, Su T, Meng GZ, Xing ZC, Liu Y. From UI design image to GUI skeleton: A neural machine translator to bootstrap mobile GUI implementation. In: Proc. of the 40th Int'l Conf. on Software Engineering. Gothenburg: ACM, 2018. 665–676. [doi: 10.1145/3180155.3180240]

[42] Nguyen TA, Csallner C. Reverse engineering mobile application user interfaces with REMAUI (T). In: Proc. of the 30th IEEE/ACM Int'l Conf. on Automated Software Engineering (ASE). Lincoln: IEEE, 2015. 248–259. [doi: 10.1109/ASE.2015.32]

[43] Zhao DH, Xing ZC, Chen CY, Xu XW, Zhu LM, Li GQ, Wang JS. Seenomaly: Vision-based linting of GUI animation effects against

© 中国科学院软件研究所 http://www.jos.org.cn

design—Don't guidelines. In: Proc. of the 42nd IEEE/ACM Int'l Conf. on Software Engineering (ICSE). Seoul: IEEE, 2020. 1286–1297.

[44] Yang B, Xing ZC, Xia X, Chen CY, Ye DH, Li SP. Don't do that! Hunting down visual design smells in complex UIs against design guidelines. In: Proc. of the 43rd IEEE/ACM Int'l Conf. on Software Engineering (ICSE). Madrid: IEEE, 2021. 761–772. [doi: 10.1109/ICSE43902.2021.00075]

[45] Moran K, Bernal-Cárdenas C, Curcio M, Bonett R, Poshyvanyk D. Machine learning-based prototyping of graphical user interfaces for mobile APPs. IEEE Trans. on Software Engineering, 2020, 46(2): 196–221. [doi: 10.1109/TSE.2018.2844788]

[46] Chen QY, Chen CY, Hassan S, Xing ZC, Xia X, Hassan AE. How should I improve the UI of my APP? A study of user reviews of popular APPs in the Google Play. ACM Trans. on Software Engineering and Methodology, 2021, 30(3): 37. [doi: 10.1145/3447808]

[47] Feng SD, Ma SY, Yu JZ, Chen CY, Zhou TT, Zhen YK. Auto-icon: An automated code generation tool for icon designs assisting in UI development. In: Proc. of the 26th Int'l Conf. on Intelligent User Interfaces. College Station: ACM, 2021. 59–69. [doi: 10.1145/3397481.3450671]

[48] Zhao TM, Chen CY, Liu YN, Zhu XD. GUIGAN: Learning to generate GUI designs using generative adversarial networks. In: Proc. of the 43rd IEEE/ACM Int'l Conf. on Software Engineering (ICSE). Madrid: IEEE, 2021. 748–760. [doi: 10.1109/ICSE43902.2021.00074]

[49] Chen CY, Feng SD, Liu ZY, Xing ZC, Zhao SD. From lost to found: Discover missing UI design semantics through recovering missing tags. Proc. of the ACM on Human-computer Interaction, 2020, 4(CSCW2): 123. [doi: 10.1145/3415194]

[50] Chen JS, Xie ML, Xing ZC, Chen CY, Xu XW, Zhu LM, Li GQ. Object detection for graphical user interface: Old fashioned or deep learning or a combination? In: Proc. of the 28th ACM Joint Meeting on European Software Engineering Conf. and Symp. on the Foundations of Software Engineering. ACM, 2020. 1202–1214. [doi: 10.1145/3368089.3409691]

[51] Moran K, Li BY, Bernal-Cárdenas C, Jelf D, Poshyvanyk D. Automated reporting of GUI design violations for mobile APPs. In: Proc. of the 40th Int'l Conf. on Software Engineering. Gothenburg: ACM, 2018. 165–175. [doi: 10.1145/3180155.3180246]

[52] Moran K, Watson C, Hoskins J, Purnell G, Poshyvanyk D. Detecting and summarizing GUI changes in evolving mobile APPs. In: Proc. of the 33rd ACM/IEEE Int'l Conf. on Automated Software Engineering. Montpellier: IEEE, 2018. 543–553. [doi: 10.1145/3238147.3238203]

[53] Yan JW, Liu H, Pan LJ, Yan J, Zhang J, Liang B. Multiple-entry testing of Android applications by constructing activity launching contexts. In: Proc. of the 42nd IEEE/ACM Int'l Conf. on Software Engineering (ICSE). Seoul: IEEE, 2020. 457–468. [doi: 10.1145/3377811.3380347]

[54] Tian C, Xia CL, Duan ZH. Poster: Android inter-component communication analysis with intent revision. In: Proc. of the 40th IEEE/ACM Int'l Conf. on Software Engineering: Companion (ICSE-Companion). Gothenburg: IEEE, 2018. 254–255. [doi: 10.1145/3183440.3194957]

[55] Chen X, Gu Q, Liu WS, Liu SL, Ni C. Survey of static software defect prediction. Ruan Jian Xue Bao/Journal of Software, 2016, 27(1): 1–25 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/4923.htm [doi: 10.13328/j.cnki.jos.004923]

[56] Chen S, Chen CY, Fan LL, Fan MM, Zhan X, Liu Y. Accessible or not? An empirical investigation of Android APP accessibility. IEEE Trans. on Software Engineering, 2022, 48(10): 3954–3968. [doi: 10.1109/TSE.2021.3108162]

[57] Luo CX, Yang ZH, Wang P, Wang Y, Xu W, Nevatia R, Yuille A. Every pixel counts++: Joint learning of geometry and motion with 3D holistic understanding. IEEE Trans. on Pattern Analysis and Machine Intelligence, 2019, 42(10): 2624–2641. [doi: 10.1109/TPAMI.2019.2930258]

[58] Nayebi F, Desharnais JM, Abran A. The state of the art of mobile application usability evaluation. In: Proc. of the 25th IEEE Canadian Conf. on Electrical and Computer Engineering (CCECE). Montreal: IEEE, 2012. 1–4. [doi: 10.1109/CCECE.2012.6334930]

[59] Kondratova I, Goldfarb I. Culturally appropriate Web user interface design study: Research methodology and results. In: Blanchard EG, Allard D, eds. Handbook of Research on Culturally-aware Information Technology: Perspectives and Models. Hershey: IGI Global, 2011. 316–336. [doi: 10.4018/978-1-61520-883-8.ch014]

[60] Mahajan S, Halfond WGJ. Detection and localization of HTML presentation failures using computer vision-based techniques. In: Proc. of the 8th IEEE Int'l Conf. on Software Testing, Verification and Validation (ICST). Graz: IEEE, 2015. 1–10. [doi: 10.1109/ICST.2015.7102586]

[61] Mahajan S, Halfond WGJ. WebSee: A tool for debugging HTML presentation failures. In: Proc. of the 8th IEEE Int'l Conf. on Software Testing, Verification and Validation (ICST). Graz: IEEE, 2015. 1–8. [doi: 10.1109/ICST.2015.7102638]

[62] Mahajan S, Li BL, Behnamghader P, Halfond WGJ. Using visual symptoms for debugging presentation failures in Web applications. In: Proc. of the 2016 IEEE Int'l Conf. on Software Testing, Verification and Validation (ICST). Chicago: IEEE, 2016. 191–201. [doi: 10.1109/ICST.2016.35]

[63] Mahajan S, Gadde KB, Pasala A, Halfond WGJ. Detecting and localizing visual inconsistencies in Web applications. In: Proc. of the 23rd

© 中国科学院软件研究所 http://www.jos.org.cn

Asia-Pacific Software Engineering Conf. (APSEC). Hamilton: IEEE, 2016. 361–364. [doi: 10.1109/APSEC.2016.060]

[64] Mahajan S, Alameer A, McMinn P, Halfond WGJ. XFix: An automated tool for the repair of layout cross browser issues. In: Proc. of the 26th ACM SIGSOFT Int'l Symp. on Software Testing and Analysis. Santa Barbara: ACM, 2017. 368–371. [doi: 10.1145/3092703.3098223]

[65] Althomali I, Kapfhammer GM, McMinn P. Automatic visual verification of layout failures in responsively designed Web pages. In: Proc. of the 12th IEEE Conf. on Software Testing, Validation and Verification (ICST). Xi'an: IEEE, 2019. 183–193. [doi: 10.1109/ICST.2019.00027]

[66] Mahajan S, Alameer A, McMinn P, Halfond WGJ. Effective automated repair of internationalization presentation failures in Web applications using style similarity clustering and search-based techniques. Software Testing, Verification and Reliability, 2021, 31(1–2): e1746. [doi: 10.1002/stvr.1746]

附中文参考文献:

[21] 王青叶, 万志远, 李善平, 夏鑫. 代码审查中代码变更恢复的经验研究. 软件学报, 2022, 33(7): 2581–2598. http://www.jos.org.cn/1000-9825/6312.htm [doi: 10.13328/j.cnki.jos.006312]

[22] 郑炜, 唐辉, 陈翔, 张满青, 夏鑫. 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. [doi: 10.7544/issn1000-1239.20210105]

[23] 蒋竞, 吕江枫, 张莉. 中文软件问答社区主题分析研究. 软件学报, 2020, 31(4): 1143–1161. http://www.jos.org.cn/1000-9825/5987.htm [doi: 10.13328/j.cnki.jos.005987]

[24] 张莉, 蒲梦媛, 刘奕君, 田家豪, 岳涛, 蒋竞. 对软件工程中经验研究的调查. 软件学报, 2018, 29(5): 1422–1450. http://www.jos.org.cn/1000-9825/5520.htm [doi: 10.13328/j.cnki.jos.005520]

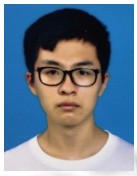
[25] 李聪, 蒋炎岩, 许畅. 基于 GUI 事件的安卓应用录制重放关键技术综述. 软件学报, 2022, 33(5): 1612–1634. http://www.jos.org.cn/1000-9825/6551.htm [doi: 10.13328/j.cnki.jos.006551]

[55] 陈翔, 顾庆, 刘望舒, 刘树龙, 倪超. 静态软件缺陷预测方法研究. 软件学报, 2016, 27(1): 1–25. http://www.jos.org.cn/1000-9825/4923.htm [doi: 10.13328/j.cnki.jos.004923]

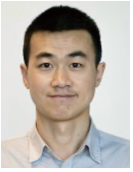
刘哲(1996－), 男, 博士, 助理研究员, CCF 学生会员, 主要研究领域为 Android 自动化测试, 用户界面显示缺陷研究, 软件工程.

车行(1999－), 男, 硕士生, 主要研究领域为软件工程, 软件测试.

王俊杰(1987－), 女, 博士, 研究员, CCF 专业会员, 主要研究领域为智能化软件工程, 软件测试, 数据挖掘.



苏宇辉(1998－), 男, 硕士, 主要研究领域为软件工程, 软件测试.



陈春阳(1987－), 男, 博士, 副教授, 主要研究领域为软件工程, 软件测试, 人机交互.

王青(1964－), 女, 博士, 研究员, 博士生导师, CCF 杰出会员, 主要研究领域为软件工程, 经验软件工程, 软件测试.

© 中国科学院软件研究所 http://www.jos.org.cn