软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2024,35(7):3180−3203 [doi: 10.13328/j.cnki.jos.007107]

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

面向版本演化的 APP 软件缺陷跟踪分析方法*

刘海毅[1,2], 姜 瑛[1,2], 赵泽江[1,2]

[1](云南省人工智能重点实验室(昆明理工大学), 云南 昆明 650504)
[2](昆明理工大学 信息工程与自动化学院, 云南 昆明 650504)
通信作者: 姜瑛, E-mail: jy_910@163.com

摘 要: 移动应用(APP)软件的版本更新速度正在加快, 对软件缺陷的有效分析, 可以帮助开发人员理解和及时修复软件缺陷. 然而, 现有研究的分析对象大多较为单一, 存在信息孤立、零散、质量差等问题, 并且没有充分考虑数据验证及版本失配问题, 分析结果存在较大误差, 导致无效的软件演化. 为了提供更有效的缺陷分析结果, 提出一种面向版本演化的 APP 软件缺陷跟踪分析方法(ASD-TAOVE). 首先, 从多源、异构的 APP 软件数据中抽取 APP 软件缺陷内容并挖掘缺陷事件的因果关系; 接着, 设计了一种 APP 软件缺陷内容验证方法, 基于信息熵结合文本特征和结构特征定量分析缺陷怀疑度, 用于缺陷内容验证并构建 APP 软件缺陷内容异构图; 然后, 为了考虑版本演化带来的影响, 设计了一个 APP 软件缺陷跟踪分析方法, 用于在版本演化中分析缺陷的演化关系, 并将其转化为缺陷/演化元路径; 最后, 通过一个基于深度学习的异构信息网络完成 APP 软件缺陷分析. 针对 4 个研究问题(RQ)的实验结果, 证实了 ASD-TAOVE 方法在面向版本演化过程中对缺陷内容验证与跟踪分析的有效性, 缺陷识别准确率分别提升约 9.9%和 5% (平均 7.5%). 与同类基线方法相比, ASD-TAOVE 方法可分析丰富的 APP 软件数据, 提供有效的缺陷信息.

关键词: APP 软件缺陷; 跟踪分析; 版本演化; 缺陷内容提取; 缺陷内容验证; 异构信息网络
中图法分类号: TP311

中文引用格式: 刘海毅, 姜瑛, 赵泽江. 面向版本演化的 APP 软件缺陷跟踪分析方法. 软件学报, 2024, 35(7): 3180−3203. http://www.jos.org.cn/1000-9825/7107.htm

英文引用格式: Liu HY, Jiang Y, Zhao ZJ. APP Software Defect Tracking and Analysis Method Oriented to Version Evolution. Ruan Jian Xue Bao/Journal of Software, 2024, 35(7): 3180−3203 (in Chinese). http://www.jos.org.cn/1000-9825/7107.htm

APP Software Defect Tracking and Analysis Method Oriented to Version Evolution

LIU Hai-Yi[1,2], JIANG Ying[1,2], ZHAO Ze-Jiang[1,2]

[1](Yunnan Key Laboratory of Artificial Intelligence (Kunming University of Science and Technology), Kunming 650504, China)
[2](Faculty of Information Engineering and Automation (Kunming University of Science and Technology), Kunming 650504, China)

Abstract: The speed of evolution in mobile application (APP) software market is accelerating. Effective analysis of software defects can help developers understand and repair software defects in time. However, the analysis object of existing research is not enough, which leads to isolated, fragmented information, and poor information quality. In addition, because of insufficient consideration of data verification and version mismatch issues, there are some errors in the analysis results, resulting in invalid software evolution. In order to provide more effective defect analysis results, an APP software defect tracking and analysis method oriented to version evolution (ASD-TAOVE) is proposed. First, the content of APP software defects is extracted from multi-source, heterogeneous APP software data,

* 基金项目: 国家自然科学基金(62162038, 61462049, 61063006, 60703116); 国家重点研发计划(2018YFB1003904); 云南省计算机技术应用重点实验室开放基金(2020101)
本文由“面向复杂软件的缺陷检测与修复技术”专题特约编辑张路教授、刘辉教授、姜佳君副研究员、王博博士推荐.
收稿时间: 2023-09-10; 修改时间: 2023-10-30; 采用时间: 2023-12-14; jos 在线出版时间: 2024-01-05
CNKI 网络首发时间: 2024-03-09

and the causal relationship of defect events is discovered. Then, a verification method for APP software defect content is designed, which is based on information entropy combined with text features and structural features to calculate the defect suspicious formula for verification and construction of APP software defect content heterogeneity graph. In order to consider the impact of version evolution, an APP software defect tracking analysis method is designed to analyze the evolution relationship of defects in version evolution. The evolution relationship can be transformed into the defect/evolutionary meta-paths which are useful for defect analysis. Finally, this study designs a heterogeneous information network based on deep learning to complete APP software defect analysis. The experimental results of four research questions (RQ) confirmed the effectiveness of ASD-TAOVE method of defect content verification and tracking analysis in the process of version-oriented evolution, and the accuracy of defect identification increased by about 9.9% and 5% respectively (average 7.5%). Compared with baseline methods, the ASD-TAOVE method can analyze more APP software data and provide effective defect information.

Key words: APP software defect; tracking analysis; version evolution; defect content extraction; defect content verification; heterogeneous information network

在软件演化过程中, 软件缺陷不可避免, 有效的缺陷分析可以支持开发人员更好地理解和修复软件缺陷. 近年来, 移动应用(APP)软件已经成为我们日常生活不可或缺的工具. 随着 APP 软件功能越来越丰富, 用户在使用软件的过程中产生的数据也越来越庞大, 伴随的软件缺陷也越来越复杂. 为了能及时修复软件缺陷, APP 软件的版本迭代和演化更新速度不断加快. 当前研究大多是通过挖掘用户评论来识别软件存在的问题[1−4]. 然而, 用户评论与程序执行语言存在较大的语义和粒度差异, 开发人员无法根据用户评论准确地定位、理解并修复 APP 软件缺陷. 此外, 由于 APP 软件功能模块的组成和调用在软件演化过程中会相互影响和变化, 例如在不同版本的 APP 软件中导致软件卡顿的原因可能为“服务器断开连接”或“图片解码错误”, 若缺乏对软件缺陷有效地跟踪分析, 开发人员难以准确判断导致软件缺陷产生的真实原因以及该缺陷是否被正确修复, 极容易在缺陷修复过程中遗漏缺陷或产生新的缺陷. 上述问题会导致无效的软件演化, 不仅会造成大量的人力资源浪费, 还将降低用户的使用体验, 使得软件失去竞争力.

最近的一项研究[5]尝试解释无效软件演化的原因, 即: 版本失配问题会对传统的缺陷分析方法产生显著影响, 将所有缺陷报告视为同一版本进行缺陷分析会导致错误的分析结果. 文献[1]根据用户反馈揭示了频繁的 APP 软件演化过程中, 在下一版本被成功修复的软件缺陷不足 20%. 移动应用市场的版本演化速度快, 不同 APP 软件的开发规范、功能架构各不相同, 版本失配问题更加突出. 例如: “哔哩哔哩”APP 软件在 V6.17.0 版本调整了“收藏夹”功能模块的布局, 由展示所有“收藏夹”修改为展示“默认收藏夹”, 其“默认收藏”“查看收藏”“取消收藏”等相关功能和组件的调用方式发生变化, 这会影响用户使用相关功能的过程, 从而影响用户的使用体验, 并因此给出反馈. 若不考虑版本演化的影响, 针对该软件旧版的缺陷报告可能包含了旧版本中的功能组件调用和用户操作情况, 开发人员依据这样的缺陷报告设计的测试用例并不适用于新版本的 APP 软件, 从而无法有效修复缺陷. 在软件演化过程中对软件缺陷进行跟踪分析, 是应对版本失配问题的有效方法. 软件缺陷跟踪分析的目的是建立历史版本缺陷之间的关联, 在缺陷分析过程中考虑软件版本演化带来的影响, 并建立缺陷之间的演化关系, 从而支持在版本演化过程中分析和评估软件缺陷. 例如: 对于上述“哔哩哔哩”APP 软件中“收藏夹”功能模块修改的例子, 缺陷跟踪分析的作用之一即是建立不同软件版本间、“收藏夹”相关功能模块用户使用情况的演化关系, 从而向开发人员提供包含演化关系的缺陷报告. 传统的缺陷跟踪分析方法针对缺陷报告, 基于文本语义挖掘(text semantic mining, TSM)方法[6−8]建立历史版本缺陷相关文本之间的关联, 并使用版本更新日志评估缺陷分析效果[1,2,9]. 但 TSM 方法的效果受限于缺陷报告文本的内容和结构, 难以支持对多源、异构的 APP 软件数据进行缺陷跟踪分析. 当前移动应用市场的版本更新速度加快, APP 软件缺陷的类型复杂, 变化速度快, 使得缺陷分析的难度加大, 其中版本失配问题尤为突出. 如何针对 APP 软件进行有效的缺陷跟踪分析, 是一个亟待解决的问题.

现有的缺陷分析方法通常依赖于对软件缺陷报告的分析来识别有缺陷的模块[10−16], 但仅使用缺陷报告中的信息不足以支持开发人员理解和修复软件缺陷[5,6]. 软件开发和用户使用过程中产生的大量软件数据包括源代码、错误报告、日志以及用户评论, 有效分析软件数据可以提高缺陷识别等任务的效率[3,17]. 其中, 用户

评论、用户操作、异常行为日志数据分别从用户体验、操作过程以及 APP 软件行为这 3 个维度描述软件中可能存在的缺陷, 但由于其数据异构性强、内部元素之间的关系复杂、不同 APP 软件之间缺陷类别差异大、类不平衡等问题, 多数现有研究仍使用孤立或零散的信息分析 APP 软件缺陷, 导致难以表示和挖掘 APP 软件数据之间的关联, 从而不能正确地理解软件缺陷.

除了内容和结构差异大之外, APP 软件数据往往还存在噪声和错误, 例如: 因用户错误的操作导致的软件异常行为, 用户为此给出了消极的用户评论, 而用户操作数据和软件运行日志数据并未显示异常, 此时, 该用户评论所反馈的问题不能被认为是软件缺陷. 因此, 在分析 APP 软件数据时应该对缺陷相关内容进行验证, 当前常见的验证方式有两类. (1) 基于相似度的噪声过滤(noise filtering based on similarity, NFBOS)方法[14,18], 用于过滤缺陷报告中与缺陷内容不相关的内容. 这种处理方式依赖对缺陷报告内容特征的提取, 若提取存在误差, 则会错误地过滤正确的样本. (2) 基于缺陷定位(fault localization, FL)的方法[19,20], 可结合信息熵或怀疑度指标定量分析缺陷相关因素之间的关系. 相较于 NFBOS 方法, FL 方法的计算精度更高.

然而, 当前对缺陷相关内容进行验证的方法没有充分考虑在 APP 软件版本演化过程中 APP 软件缺陷内容之间的关系是一个复杂且不断变化的过程, 不同缺陷内容的演化程度不同. 例如: 当前版本存在的软件缺陷在下一版本中可能被修复, 也可能未被修复, 未被修复的原因可能与上一版本相同, 也可能是由于在修复过程中产生了其他原因导致该缺陷未被修复. 因此, 针对 APP 软件版本演化的缺陷跟踪分析有赖于对其缺陷内容的验证和潜在关联的深入挖掘.

综上所述, 若要进一步提升软件缺陷报告的质量, 为开发人员提供丰富、有效的软件缺陷信息, 就需要解决信息孤立、数据噪声大、版本失配等问题. 因此, 本文提出一种面向版本演化的 APP 软件缺陷跟踪分析方法 ASD-TAOVE (APP software defect tracking and analysis method oriented to version evolution). 针对信息孤立问题, ASD-TAOVE 首先从多源、异构的 APP 软件数据中抽取 APP 软件缺陷内容, 并挖掘缺陷事件的因果关系, 以从多个维度补充软件缺陷内容. 针对数据噪声, ASD-TAOVE 包含一个新颖的缺陷内容验证方法, 基于提出的缺陷怀疑度对不同的缺陷内容设计怀疑对象, 结合信息熵量化分析文本特征和结构特征, 再根据获取的缺陷怀疑度结果过滤缺陷内容的噪声和错误, 并构建缺陷内容异构图. 针对版本失配问题, ASD-TAOVE 基于缺陷内容异构图设计 APP 软件缺陷跟踪分析方法, 分析并评估历史版本的缺陷内容, 建立演化关系并获取缺陷/演化元路径. 最后, ASD-TAOVE 通过一个基于深度学习的异构信息网络模型进一步提取并融合缺陷内容与缺陷/演化元路径的特征, 以完成面向版本演化的 APP 软件缺陷分析.

本文的主要贡献总结如下:

(1) 本文设计了一个缺陷内容验证方法, 基于信息熵计算内容怀疑度和关联怀疑度, 有助于挖掘缺陷内容文本的语义和结构关联特征, 能够更加准确地过滤缺陷内容的噪声和错误, 从而构建基于多源 APP 软件数据的缺陷内容异构图.

(2) 本文设计了一个 APP 软件缺陷跟踪分析方法, 可在 APP 软件缺陷内容异构图中建立演化关系并获取缺陷/演化元路径, 在提出的 APP 软件缺陷跟踪模型中进一步提取并融合特征, 有助于在版本演化过程中分析和评估软件缺陷, 获得有效的分析结果.

(3) 本文针对 3 个用户常用的 APP 软件在不同版本中的软件数据进行分组实验, 对 ASD-TAOVE 的性能进行了全面的评估, 实验结果显示 ASD-TAOVE 能够有效分析面向版本演化的缺陷内容, 并为开发人员提供细粒度的缺陷修复建议, 在版本演化中的平均缺陷识别准确率提升约 7.5%.

本文首先阐述 APP 软件缺陷分析的研究背景. 第 1 节分析与本文提出方法相关的研究和应用. 第 2 节介绍 ASD-TAOVE 方法的原理、设计和实现细节. 第 3 节介绍本文实验所用数据集、实验设置和研究问题. 第 4 节针对研究问题设计实验获取实验结果, 对实验结果进行分析并给出相关结论. 第 5 节阐述有效性威胁. 第 6 节总结全文.

1 相关工作

随着版本的快速迭代, APP 软件缺陷的表现、影响用户操作的范围、产生的原因都可能随之发生变化, 这会导致缺陷分析变得困难. 根据面向版本演化过程中 APP 软件缺陷分析任务涉及到的方法, 本节将分别介绍基于文本挖掘的缺陷识别、噪声过滤与缺陷定位、APP 软件缺陷报告重放、面向异构数据分析的缺陷识别、基于深度学习的异构信息网络分析的相关方法.

1.1 基于文本挖掘的缺陷识别

现有的基于文本挖掘的缺陷识别方法可分为两类.

(1) 基于单源软件数据的分析方法[3,4,9]. 钟仁毅等人[9]提出的 ATA 方法基于机器学习的方法, 针对版本更新日志分析版本更新趋势. Wang 等人[3]提出一种语言感知的方法, 从用户评论中提取〈用户操作, APP 软件行为异常〉事件对, 用于帮助理解软件缺陷. 这些方法可以从分析对象中提取与软件缺陷相关的内容, 但分析结果受到数据信息质量差、孤立、零散等问题的影响, 分析效果有限.

(2) 基于多源软件数据的分析方法[2,21,22]. 肖建茂等人[2]设计了一种更新日志与用户评论之间的匹配算法(ARICA)度量文本之间的相似性, 用于分析用户评论在版本更新中的作用. Saidani 等人[21]提出一种 APP 软件缺陷追踪方法(APPTRACKER), 基于版本更新记录和负面的用户评论分析不良的更新.

基于文本分析的方法虽然在一定程度上缓解了信息孤立、零散的问题, 但未充分考虑多源软件数据之间异质性强、存在噪声和错误的问题, 仅使用文本分析方法难以对其进行验证并挖掘其中的潜在关联. 因此, 本文将文本分析方法与其他方法结合起来分析 APP 软件缺陷内容.

1.2 噪声过滤与缺陷定位

缺陷报告中常用的噪声过滤方法是基于相似度的噪声过滤(noise filtering based on similarity, NFBOS)方法[14,18]. 典型方法是 Peters 等人[18]给出的一种噪音数据过滤框架 FARSEC. 该框架使用 TF-IDF 获取与安全相关的缺陷报告关键词, 可用于过滤缺陷报告中与安全关键词相似度较高的样本(即缺陷报告). Jiang 等人[14]基于生成模型识别和安全缺陷报告相关的语义区域, 接着计算每个非安全缺陷报告的异常值分数, 用于过滤掉和安全报告相似并可能是噪音的非安全缺陷报告. NFBOS 方法的处理方式存在误差, 如果提取的词不准确或语义差异较大, 那么可能会错误地过滤掉正确的样本. 基于缺陷定位(fault localization, FL)的方法通过怀疑度指标定量分析缺陷相关因素之间的关系[19,20]. 例如: 姜淑娟等人[19]提出一种基于路径分析和信息熵的错误定位方法 FLPI, 在频谱信息技术的基础上, 基于信息熵分析执行上下文的信息, 并计算可疑语句的怀疑度用于缺陷定位. 类似地, 李铮等人[20]通过收集深度神经网络的神经元输出信息和预测结果作为频谱信息, 提出了针对深度神经网络缺陷定位的怀疑度计算方法, 基于贡献信息计算神经元的怀疑度以找出最有可能存在缺陷的神经元. 考虑到FL可灵活地设计怀疑对象和怀疑度计算方法对不同类型的APP软件数据进行分析, 并且相较于 NFBOS 方法的分析结果更加准确, 因此, 受到 FL 方法的启发, 本文根据不同的 APP 软件缺陷内容设计了不同类型的怀疑对象, 从多个角度综合分析 APP 软件缺陷内容的怀疑度, 用于提高缺陷内容验证的有效性.

1.3 APP软件缺陷报告重放

为了提高移动软件缺陷报告的质量, 部分研究尝试从缺陷报告中重放具体的步骤(detailed reproduction steps, S2Rs). Zhao 等人[23]在 2019 年提出了 ReCDroid 方法, 该方法基于自然语言处理方法, 结合设计的语法规则从缺陷报告中提取能够表示 S2Rs 的事件, 并结合图形用户界面(graph user interface, GUI)确定 S2Rs 序列. Fazzini 等人[24]进一步结合静态 GUI 分析与动态 GUI 分析来分析用户操作的顺序. 在最近的研究中, Zhang 等人[25]为了重现最初错误报告中缺少或不精确的步骤, 将利用自然语言处理技术提取的 S2Rs 与 APP 软件 UI (user interface)元素进行匹配, 并基于强化学习优化其匹配策略, 进一步提升了缺陷报告的质量. 实验表明: 该方法能够较为准确地从 bug 报告中重现 S2Rs, 优于同类基线方法. 缺陷报告重放方法主要的分析对象为缺陷报告, 而高质量的缺陷报告往往难以获取. 相较于缺陷报告, 用户操作数据中包含更加丰富和准确的 S2Rs

和UI元素信息, 可补充有助于开发人员理解软件缺陷的信息. 另一方面, 不同版本中被重放的S2Rs可能指向不同软件缺陷, 因此在分析 S2Rs 时, 还应考虑由版本演化带来的影响.

1.4 面向异构数据分析的缺陷识别

近年来, 一些用于跨项目缺陷检测的方法可针对异构软件数据进行分析[26–29]. 例如: Chen 等人[28]基于Pearson特征选择方法, 结合度量补偿技术实现异构数据集之间的软件缺陷检测; 李伟漳等人[29]提出一种同步语义对齐的方法, 分别学习异构数据集之间的语义表示和语义迁移关系, 通过目标伪标签的匹配进行对齐. 此类方法大多数基于特征选择与关联分析方法对异构数据进行分析, 忽略了版本失配问题, 分析结果存在误差. 此外, 由于未能同时考虑文本特征和离散的结构特征, 导致其关联分析效果受到影响. 因此, 本文针对版本失配问题提出了 APP 软件缺陷跟踪分析方法, 在版本演化过程中定量分析和评估软件缺陷的演化程度. 针对关联分析效果差的问题, 在验证过程中分别提出内容怀疑度和关联怀疑度, 结合文本特征和结构特征共同分析 APP 软件缺陷内容之间的关联, 有助于进一步提升关联分析的效果.

1.5 基于深度学习的异构信息网络分析

基于深度学习的文本挖掘技术有助于提取缺陷内容中深层次的语义特征, 提升缺陷识别效果[13,30–34]. 郑炜等人[13]在 5 个不同规模的安全缺陷报告数据集上进行了实证研究, 结果表明, 经典文本分类方法与注意力相结合的深度学习模型(TextCNN-attention 和 TextRNN-attention)优于传统的机器学习模型. 由于异构信息网络[35]能够提供丰富的结构化信息, 其连通性有助于学习和表示异构数据间复杂的关联关系, 近年来一些研究基于异构信息网络, 结合深度学习建模异构数据的分析过程取得了较好的分析结果[36–38]. Wang 等人[36]基于元路径挖掘节点与邻居节点之间的关系, 并使用分层注意力机制(attention)学习节点与元路径的重要性. 类似地, 石美惠等人[38]通过建立用户-兴趣点-区域之间的异构信息网络来表示异构数据之间的关联关系, 并提取不同的元路径学习实体间的关联表示, 基于注意力机制(attention)[30]计算不同元路径的贡献度, 用于下一个兴趣点的推荐. 实验结果表明, 基于异构信息网络并结合元路径和深度学习的方法可以有效分析异构数据之间的关联. 然而, 由于 APP 软件数据之间的关系较为复杂且含有噪声, 目前鲜有研究使用异构信息网络表示 APP 软件数据之间的关系, 并用于 APP 软件缺陷的分析. 因此, 本文提出了 APP 软件缺陷内容验证方法, 首先对缺陷内容之间的关联关系进行分析, 接着将其转化为 APP 软件缺陷内容异构图, 以进一步用于 APP 软件缺陷分析.

2 ASD-TAOVE 方法

ASD-TAOVE 方法具体分为 3 步, 分析版本演化过程中的 APP 软件缺陷.

(1) APP 软件缺陷内容抽取. 分别从 3 个来源的 APP 软件数据中抽取与软件缺陷相关的内容, 提取日志数据中的缺陷事件和因果关系, 建立缺陷事件因果图.

(2) APP 软件缺陷内容验证. 逐层验证由步骤(1)获取的 APP 软件缺陷内容, 并建立其关联, 获取 APP 软件缺陷内容异构图.

(3) APP 软件缺陷跟踪分析. 对步骤(2)获取的 APP 软件缺陷内容异构图进行演化分析, 建立演化关系, 并基于 APP 软件缺陷跟踪分析模型, 获取分析结果.

ASD-TAOVE 整体框架及应用场景如图 1 所示.

考虑到在 APP 软件缺陷分析环境中, 开发人员获取的数据类型可能不同, ASD-TAOVE 为开发人员提供了灵活的使用方式, 分别为低资源管道和高资源管道. 在低资源管道中, 开发人员可基于加入演化关系的 APP 软件缺陷内容异构图进行数据增强, 将单一类型的数据通过异构关系扩充为信息更加丰富的多源数据, 然后再通过APP软件缺陷跟踪分析模型对单条APP软件缺陷进行分析. 当开发人员拥有较为丰富的数据资源时, 可根据高资源管道使用APP软件缺陷跟踪分析模型对单条APP软件缺陷进行分析, 然后基于APP软件缺陷内容异构图对分析结果进行信息检索, 以获取相关联的缺陷分析结果. 如此, ASD-TAOVE 为开发人员提供

了便捷的使用管道, 使得开发人员无须考虑不同源软件数据的语义鸿沟以及数据类型即可进行 APP 软件缺陷分析, 并获取较为丰富的缺陷信息.

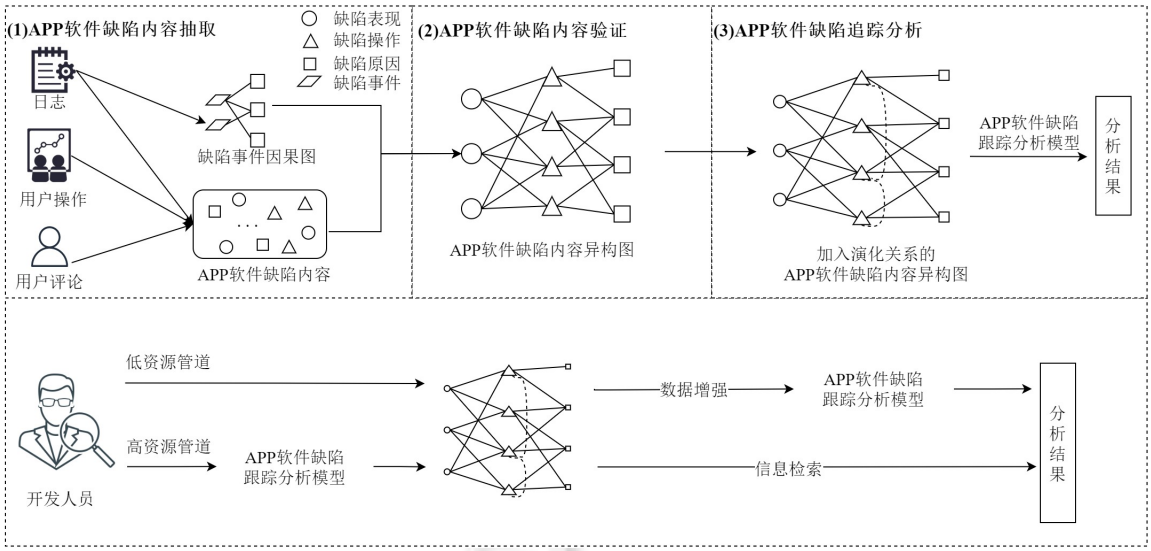

图 1 ASD-TAOVE 整体框架及应用场景

2.1 APP软件缺陷内容抽取

APP 软件用户评论、APP 软件用户操作以及 APP 软件日志数据(以下称为 APP 软件数据)可以分别从用户主观反馈、实际操作行为以及软件运行环境这 3 个角度描述与软件缺陷相关的内容. 现有的研究通常独立地分析 APP 软件数据来分析软件缺陷[4,12,14], 导致缺陷内容是零散并且孤立的, 不足以让开发人员理解和修复软件缺陷. 此外, 以往研究中通常将不同的 APP 软件数据源平等看待, 忽略了不同数据源之间的信息差异, 从而导致分析结果有误. 因此, ASD-TAOVE 方法分为两步抽取 APP 软件缺陷内容, 具体如下.

(1) 预处理 APP 软件数据

我们通过采集工具分别采集了 APP 软件用户评论、APP 软件用户操作以及 APP 软件日志数据, 并分别抽取缺陷内容. 本文使用爬虫工具采集 360 手机应用市场的用户评论, 使用本实验室开发的采集工具(APP 用户行为监测分析软件 V1.0, 软件著作权登记号: 2022SR1600335)采集用户操作及软件日志数据. 为了获取有效的缺陷信息, 本文将缺陷内容定义为一个四元组: 〈缺陷名称,缺陷表现,缺陷操作,缺陷原因〉. 其中: 用户评论中包含的缺陷名称和表现是用户对软件缺陷的主观印象; 缺陷操作描述的是用户实际操作过程中的细节, 反映了产生缺陷的操作过程; 缺陷原因则是基于软件运行日志的客观记录, 揭示了缺陷发生的原因. 针对 APP 软件用户评论, 本文参考文献[39]中的方法提取差评评论中的评价对象以及评价观点作为缺陷名称及缺陷表现. 其次, 针对 APP 软件用户操作和 APP 软件日志数据, 通过采集工具将采集的操作记录和日志记录转化为结构化的数据(具体见第 2.2 节中的示例), 提取在相同时间段内产生的所有操作记录以及日志记录作为缺陷操作以及缺陷原因.

(2) 建立 APP 软件缺陷事件因果图

定义 1(缺陷事件因果关系). 这是指一段时间内产生的与软件缺陷有关的事件 $e1$ 与事件 $e2$ 之间存在的一种关联关系. 若在一段时间内, 缺陷事件 $e1$ 于缺陷事件 $e2$ 之前发生的频率远大于 $e2$ 于 $e1$ 前发生的概率, 则认为缺陷事件 $e1$ 与缺陷事件 $e2$ 之间存在因果关系[40], 其具体表现如公式(1)所示.

$$R_{(e1,e2)}=\frac{C_{(e1,e2)}-C_{(e2,e1)}}{C_{(e1,e2)}+C_{(e2,e1)}}\ (e1,e2\in E_{\mathrm{APP(Activity)}}) \tag{1}$$

其中, $C_{(e1,e2)}$表示缺陷事件 $e1$ 发生于缺陷事件 $e2$ 之前的次数, $C_{(e2,e1)}$表示缺陷事件 $e2$ 发生于缺陷事件 $e1$ 之前

的次数. 当 $C_{(e1,e2)}$较大而 $C_{(e2,e1)}$较小时, 缺陷事件的因果关系强度 $R_{(e1,e2)}$越大. 由于用户在操作 APP 软件的过程中会发生 APP 软件切换、功能转移等行为, 而日志数据通过对“Activity”记录用户在该功能模块的操作以及软件响应信息, 为了使计算结果更准确, 我们将缺陷事件序列的计算周期范围限定于同一“Activity”周期内.

定义 2(缺陷事件因果图). $G(V,E)$是一个有向无环图, 节点 V 是具有因果关系的变量, 包含两类节点, 分别为表示软件功能启动事件的描述类缺陷原因 V_{GO}, 表示错误类、异常类缺陷原因 V_{DC}; 边 $E(V_{GO},V_{DC})$为缺陷事件的因果关系.

基于步骤(1)获取的 APP 软件缺陷内容相互独立地描述软件缺陷, 不同源的缺陷内容之间存在语义鸿沟, 无法对其进行有效的关联性分析. 考虑到日志数据主要是以“Activity”进程为单位记录缺陷原因, 主要包含错误类(error)、异常类(exception)缺陷原因以及描述类(description)缺陷原因. 其中, 描述类缺陷原因包含与“Activity”相关的软件功能启动事件, 此类事件存在两个特点: (1) 与缺陷操作中记录用户使用 APP 软件功能(activity)的内容相关; (2) 与错误类、异常类缺陷原因存在时序上的对应关系. 因此, 我们提出了缺陷事件因果关系(定义 1)与缺陷事件因果图(定义 2)用于分析软件功能启动与软件缺陷原因之间的因果关系, 示例图如图 2 所示.

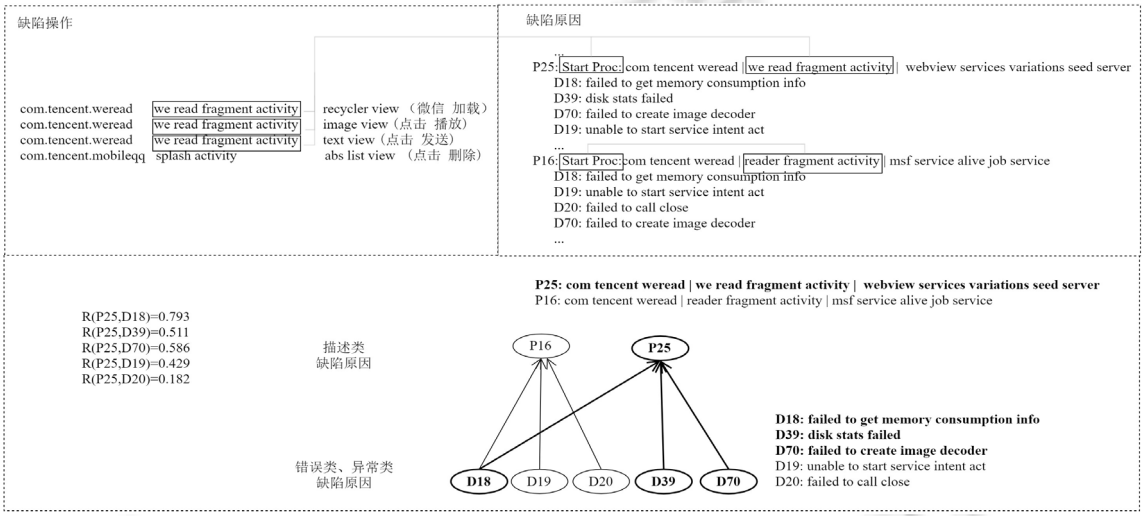

图 2 缺陷事件因果图(示例)

如图 2 所示, 通过对缺陷原因文本进行词法分析可获取包含“Start Proc”关键词的相关内容, 即描述类缺陷原因, 其中包含与“Activity”相关的描述, 将其与缺陷操作中记录 APP 软件功能活动的内容关联, 可作为缺陷原因与缺陷操作内容关联分析的基础. 例如: 图 2 缺陷操作中, “we read fragment activity”与缺陷原因 P25 中记录软件功能的相同内容(P25 中实线相连的方框部分). 文本将描述类原因与错误类、异常类缺陷原因视为两类缺陷事件, 分析其缺陷事件因果关联. 图 2 缺陷原因部分记录了某用户在 1 天内使用微信阅读 APP 软件的缺陷原因内容片段, 其中: P25: “com tencent weread|we read fragment activity|webview services variations seed server”表示与微信阅读 APP 软件的导航功能界面中, 与网络视图相关的服务组件; 而 D18、D39、D70、D19 分别为该进程执行时所记录的若干错误类、异常类缺陷原因, 其内容分别为“failed to get memory consumption info (读取内存消耗失败)”“diskstatsfailed (磁盘错误)”“failed to create image decoder (无法解码图片), unable to start intent intent act (无法启动跳转服务)”. 基于公式(1)可计算上述两类缺陷事件的因果关系强度, 图 2 显示了该用户在 1 天之中的缺陷事件因果关系强度计算结果, 根据计算结果可建立缺陷事件间的因果关系. 由于不同的“Activity”进程中可能存在类似的错误类、异常类缺陷原因, 这使得缺陷事件的因果关联强度计算会具有一定的噪声, 本文将缺陷事件因果关联强度阈值设定为 0.5, 以过滤因果关联强度较低的关系, 例如保留图 2 中错误类、异常类缺陷原因 D18、D39、D70 (加粗部分)与描述类缺陷原因 P25 的因果关系. 如此, 缺陷事

件因果图可以表示软件功能启动事件(activity)与软件缺陷原因之间的潜在因果关联, 可作为进一步挖掘 APP 软件缺陷内容之间的关联基础.

2.2 APP软件缺陷内容验证

定义 3(APP 软件缺陷内容异构图). $GD(O_d,E_d)$, O_d 表示缺陷内容对象, 存在一个实体映射函数$\partial:O_d\rightarrow A$ 以及一个关系映射函数$\varphi:E_d\rightarrow R$. 其中, A 和 R 分别表示缺陷内容类型和关系类型. 缺陷内容类型包括缺陷表现、缺陷操作、缺陷原因, 关系类型包括不同类型缺陷内容之间的关联关系以及同类缺陷内容之间的演化关系.

在第 2.1 节我们抽取了 APP 软件缺陷内容, 并基于缺陷原因构建了缺陷事件因果图. 然而, 由于 APP 软件缺陷内容异质性强、描述粒度差异大, 抽取的缺陷内容中还存在噪声和错误. 不同源缺陷内容之间仍存在较大的鸿沟, 传统的文本分析方法仅从文本特征的角度建立数据之间的关联, 忽略了数据内部的特征, 关联分析效果不理想. 此外, 缺陷事件因果图中只包含了缺陷原因中表示软件功能启动的缺陷事件与表示错误、异常类缺陷事件的关联, 还未建立不同源缺陷内容之间的关联, 开发人员无法通过 APP 软件缺陷内容或缺陷事件因果图获取或扩展准确的、具有关联性的缺陷信息. 例如: 当观察到用户评论“登录时闪退”, 开发人员无法了解用户的实际操作和缺陷产生的原因, 需要通过大量的测试用例来重现缺陷产生的步骤, 从而分析缺陷产生的原因, 这非常消耗人力和物力资源. 因此, 本文提出了 APP 软件缺陷内容的验证方法, 主要包含两个方面: 分析并过滤 APP 软件缺陷内容中的噪声和错误; 建立不同来源的 APP 软件缺陷内容之间的关联.

针对缺陷内容中的噪声和错误, 引入缺陷定位问题中的怀疑度可以结合文本特征和结构特征以验证 APP 软件缺陷内容并表示其中的关联关系. 因此, 本文基于缺陷定位问题, 引入信息熵计算不同缺陷内容的缺陷怀疑度, 并基于计算结果构建 APP 软件缺陷内容异构图 *GD*. 针对缺陷内容的关联性分析, 由于同一版本中 APP 软件缺陷是客观存在的, 其数量和表现形式主要与缺陷内容的语义描述和结构相关, 不因缺陷内容获取的时间差异以及用户偏好差异而显著改变. 因此, 本文主要关注缺陷内容的语义和结构特征来建立不同源缺陷内容之间的关联. 对于某一软件缺陷来说, 相较于缺陷操作与缺陷表现, 从 APP 软件日志中抽取的缺陷原因是对软件缺陷较为客观和准确的描述. 为了利用不同类型的缺陷信息, 本文定义了 APP 软件缺陷内容异构图(定义 3), 并按照缺陷原因→缺陷操作→缺陷表现的顺序挖掘并建立 APP 软件内容之间潜在的关联, 构建 APP 软件缺陷内容异构图. 其中, 为了更好地验证和关联不同的缺陷内容, 使用缺陷怀疑度量化缺陷内容之间的关联, 由内容怀疑度和关联怀疑度两部分组成.

(1) 缺陷怀疑度-内容怀疑度

在错误定位分析过程中, 怀疑度可以用来度量程序或执行路径出错的可能, 怀疑度越高, 出错的可能性越大[19,20]. 对于缺陷操作, 由于用户操作具有随机性, 独立的缺陷操作并不能反映发生缺陷的操作意图, 而当发生缺陷时, 通常会引起缺陷操作的前驱操作和后继操作异常, 例如: 用户在使用“微信阅读”APP 软件时, 因软件卡顿或崩溃导致用户不能使用期望的软件功能. 因此, 异常操作路径可以反映用户使用 APP 软件时的异常信息, 本文将其作为缺陷操作的内容怀疑对象. 对于缺陷表现, 内容怀疑度由两部分组成: 一方面, 缺陷表现的频次越高, 表示越多的用户反馈了该问题, 可以自然地认为该类缺陷表现的怀疑度越高; 另一方面, 由于用户表达习惯、方式、角度的不同, 不同用户可能针对同一类缺陷表现产生不同的评论, 例如: 针对“软件卡顿”, 用户 *A* 产生的用户评论为“卡死了”, 用户 *B* 产生的用户评论为“卡的动不了”, 此类评论表达了类似的缺陷表现, 具有较高的相似度, 为了更准确地分析缺陷表现内容, 本文将缺陷表现的文本相似度与频次共同作为内容怀疑度的怀疑对象. 本文针对缺陷操作和缺陷表现设计的内容怀疑对象见表 1.

表 1 内容怀疑对象

APP 软件缺陷内容	内容怀疑对象	作用	对应公式
缺陷操作	异常操作路径	通过分析操作路径的内容怀疑度可以对缺陷操作进行验证	(2)–(4)
缺陷表现	频次	出现频次较高的缺陷表现被验证为缺陷内容的可能性较高	(5)
	文本相似度	在针对相同 APP 软件的缺陷表现中, 降低由用户表达习惯、方式、角度等因素带来的误差	(5)–(8)

在错误定位任务中, 文献[19]指出: 在错误定位中引入信息熵可以提升错误定位的准确性, 有助于量化错误程序的怀疑度. 类似地, APP 软件缺陷内容验证的目的之一即是定位最有可能表示缺陷的内容, 从而去除噪声和错误. 因此, 本文基于内容怀疑度的怀疑对象, 引入信息熵对缺陷操作和缺陷表现进行验证. 其中, 根据公式(2)–(4)计算缺陷操作的内容怀疑度.

$$CS_{\mathrm{op}}=\frac{H(CF)_{\mathrm{op}}\cdot NF_{\mathrm{op}}}{H(CS)_{\mathrm{op}}\cdot NS_{\mathrm{op}}} \tag{2}$$

$$H(X_{\mathrm{op}})=-P(X_{\mathrm{op}})\cdot Log(P(X_{\mathrm{op}})) \tag{3}$$

$$P(X_{\mathrm{op}})=\frac{NX_{\mathrm{op}}}{\sum X_{\mathrm{op}}},\ X_{\mathrm{op}}\in Path(APP)_{\mathrm{op}} \tag{4}$$

CS_{op} 表示缺陷操作的内容怀疑度得分; NF_{op}, NS_{op} 分别表示异常操作和正确操作发生的次数, 判断依据为该条操作是否引起了异常的操作路径; $H(CF)_{\mathrm{op}}$ 和 $H(CS)_{\mathrm{op}}$ 分别表示异常路径中包含该操作次数的信息熵和正确操作路径中包含该操作次数的信息熵; $H(X_{\mathrm{op}})$和 $P(X_{\mathrm{op}})$分别表示任意操作 X_{op} 发生的信息熵和概率, 其中, X表示由该操作前驱、该操作、该操作后继构成的操作 NF_{op} 路径发生事件, 事件的计算范围为同一 APP 软件中, 表示为 $path(\mathrm{APP})_{\mathrm{op}}$; NX_{op} 表示该事件发生的次数; $\sum X_{\mathrm{op}}$ 表示范围内所有操作路径总数. 通过公式(2)–(4), 可以基于异常操作路径分析缺陷操作的内容怀疑度, 以去除怀疑度较低, 保留怀疑度较高的缺陷操作.

公式(5) –(8)用于计算缺陷表现的内容怀疑度.

$$CS_{\mathrm{rev}}=\frac{H(F)_{\mathrm{rev}}\cdot Sim(F)_{\mathrm{rev}}}{H(S)_{\mathrm{rev}}\cdot Sim(S)_{\mathrm{rev}}} \tag{5}$$

CS_{rev} 表示缺陷表现的内容怀疑度得分; $H(F)_{\mathrm{rev}}$、$H(S)_{\mathrm{rev}}$ 分别表示缺陷表现发生次数的信息熵和相同 APP 软件中其他类缺陷表现发生次数的信息熵; $Sim(F)_{\mathrm{rev}}$、$Sim(S)_{\mathrm{rev}}$ 分别表示缺陷表现与相同 APP 软件中其他类缺陷表现的相似度以及不相似度, 具体计算方法如公式(6)、(7)所示.

$$Sim(F)_{\mathrm{rev}}=\frac{\sum Cosine(\mathrm{rev},\mathrm{rev}_{APP})}{\sum \mathrm{rev}_{\mathrm{APP}}} \tag{6}$$

$$Sim(S)_{\mathrm{rev}}=1-Sim(F)_{\mathrm{rev}} \tag{7}$$

公式(6)表示采用余弦相似度计算缺陷表现 rev 与相同 APP 软件中其他类缺陷表现 $\mathrm{rev}_{\mathrm{APP}}$ 的平均相似度.考虑到缺陷表现的组成为“对象-具体表现”, 例如“电影-不能播放”“视频-打开不了”, 针对同一个对象的具体表现可能不同, 其对应的缺陷内容也不同, 所以本文进一步细化相似度计算方法, 将公式(5)细化为

$$CS_{\mathrm{rev}}=\frac{H(F)_{\mathrm{rev\_ob}}\cdot Sim(F)_{\mathrm{rev\_ob}}}{H(S)_{\mathrm{rev\_ob}}\cdot Sim(S)_{\mathrm{rev\_ob}}}+\frac{H(F)_{\mathrm{rev\_rep}}\cdot Sim(F)_{\mathrm{rev\_rep}}}{H(S)_{\mathrm{rev\_rep}}\cdot Sim(S)_{\mathrm{rev\_rep}}} \tag{8}$$

$H(F)_{\mathrm{rev\_ob}}$, $H(F)_{\mathrm{rev\_rep}}$ 分别表示缺陷表现对象和缺陷具体表现发生次数的信息熵; $Sim(F)_{\mathrm{rev\_ob}}$, $Sim(F)_{\mathrm{rev\_rep}}$ 分别相同 APP 软件中不同类缺陷表示对象和具体表现之间的平均相似度. 通过公式(5)–公式(8), 可以较为精确地计算缺陷表现的内容怀疑度, 以过滤怀疑度较低的缺陷表现.

(2) 缺陷怀疑度-关联怀疑度

虽然基于内容怀疑度可对 APP 软件缺陷内容中的噪声和错误进行验证, 但其无法表示不同源缺陷内容间的关联, 缺陷内容间的一致性难以判断. 因此, 本文提出了关联怀疑度用于挖掘不同缺陷内容之间的潜在关联, 在内容验证的基础上进一步验证其关联性强弱, 以达到综合验证和关联性分析的目的.

对于缺陷原因和缺陷操作, 基于第 2.1 节获取的缺陷事件因果图中描述类缺陷原因是日志数据中与 APP 软件功能启动(activity)相关的描述, 这与缺陷操作中用户使用 APP 软件功能(activity)相关. 由于日志数据和操作数据几乎是同时发生和采集的, 所以二者在同一 APP 软件功能模块中发生的次序差异即可反映其关联性强弱, 相对位置差异越小, 缺陷操作与缺陷原因的关联性越强, 反之则越弱. 因此, 本文以“Activity”划分计算周期, 分别获取描述类缺陷原因与缺陷操作在相同计算周期内的发生次序差异, 即相对位置差异作为缺陷操作和缺陷原因的关联怀疑对象.

对于缺陷操作和缺陷表现, 缺陷表现“对象”是指 APP 软件中存在缺陷的功能, “具体表现”则是对 APP 软件功能以何种方式发生缺陷的描述. 而缺陷操作的内容则是以“功能模块-使用方式”的结构记录用户使用软件的细节. 同一功能模块可向用户提供不同的功能, 例如, “MediaPlayActivity-clickplay”与“MediaPlayActivity-click pause”分别表示在多媒体资源播放功能模块点击播放和在同一功能模块中点击暂停. 考虑到缺陷操作和缺陷表现中分别以“功能模块-使用方式”和“对象-具体表现”的方式体现 APP 软件的功能和使用细节, 因此, 本文分别计算“功能模块”与“对象”以及“使用方式”与“具体表现”的文本相似度作为缺陷操作和缺陷表现的关联怀疑对象, 见表 2.

表 2 关联怀疑对象

APP 软件缺陷内容	关联怀疑对象	作用	对应公式
缺陷操作-缺陷原因	相对位置差异	度量在一个 Activity 操作周期内缺陷操作与缺陷原因的关联关系	(9)、(10)
缺陷表现-缺陷操作	文本相似度	度量缺陷表现与缺陷操作中描述软件功能及使用细节之间的差异	(11)–(13)

基于关联怀疑对象, 本文根据公式(9)、(10)计算缺陷操作与缺陷原因之间的关联怀疑度, 并结合缺陷操作的内容怀疑度构成缺陷操作与缺陷原因之间的缺陷怀疑度.

$$RS_{\text{op-rea}}=\frac{1}{RD} \tag{9}$$

$$DS_{\text{op-rea}}=CS_{\text{op}}\cdot RS_{\text{op-rea}} \tag{10}$$

$RS_{\text{op-rea}}$ 表示缺陷操作与缺陷原因之间的关联怀疑度, RD 表示缺陷操作与缺陷原因在以 Activity 划分的计算周期内的相对位置差异, 关联怀疑度与 RD 成反比, 当相对位置差异越大时, 关联怀疑度越小, 反之则越大; $DS_{\text{op-rea}}$ 表示缺陷操作与缺陷原因之间的缺陷怀疑度, 由缺陷操作的内容怀疑度和缺陷操作与缺陷原因之间的关联怀疑度乘积构成.

公式(11) –(13)用于计算缺陷表现与缺陷操作之间的关联怀疑度, 并结合缺陷表现的内容怀疑度构成缺陷表现与缺陷操作之间的缺陷怀疑度.

$$RS_{\text{rev-op}}=Sim_{\text{rev-op}} \tag{11}$$

$RS_{\text{rev-op}}$ 表示缺陷表现与缺陷操作之间的关联怀疑度, $Sim_{\text{rev-op}}$ 表示缺陷表现与缺陷操作之间的文本相似度. 由于缺陷表现和缺陷操作中分别使用“对象”和“功能模块”表示用户使用的软件功能, 使用“具体表现”和“使用方式”表现使用过程及细节, 因此可将公式(11)可以进一步细化, 如公式(12)所示.

$$Sim_{\text{rev-op}}=Sim_{\text{ob-activity}}+Sim_{\text{rep-use}} \tag{12}$$

$$DS_{\text{rev-op}}=CS_{\text{rev}}\cdot RS_{\text{rev-op}} \tag{13}$$

$Sim_{\text{ob-activity}}$, $Sim_{\text{rep-use}}$ 分别表示“对象”与“功能模块”、“具体表现”与“使用方式”之间的相似度; $DS_{\text{rev-op}}$ 表示缺陷表现与缺陷原因之间的缺陷怀疑度, 由缺陷表现的内容怀疑度和缺陷表现与缺陷原因之间的关联怀疑度乘积构成.

(3) 缺陷怀疑度阈值

为了获取经过验证的 GD, 经过步骤(1)、步骤(2)获取的缺陷怀疑度需要进一步确定阈值, 用以去除 APP 软件缺陷内容中的噪声和关联性较弱的关系. 因此, 缺陷怀疑度的阈值确定问题可以转化为 GD 的剪枝阈值优化问题. 由于 GD 中每个连通分量均可表示有关联的 APP 软件缺陷内容, 在逐层验证 APP 软件缺陷内容的过程中, 每次需要分析并建立两类不同来源的缺陷内容之间的关联. 考虑到同一连通分量中不同来源的缺陷内容从两个角度描述同一类软件缺陷, 可以通过聚类(聚类数量设置为 2)将语义、结构上类似的节点归为同一类. 所以, 本文使用 K-Means 聚类算法对 GD 中的连通分量进行聚类, 基于聚类结果中节点与类中心的距离来评价剪枝的效果. 特别注意的是: 过度剪枝会导致 GD 中存在大量只包含一个节点的连通分量, 在缺陷分析中无法应用该类型节点. 为了考虑 GD 的整体有效性, 本文将损失函数设计为 GD 中连通分量聚类的平均距离, 具体如公式(14)所示.

$$Loss=\frac{\sum disComp(Comp)}{\sum Comp} \tag{14}$$

其中, $\sum disComp(Comp)$ 表示 GD 中所有连通分量聚类后的平均距离之和, $\sum Comp$ 表示 GD 中连通分量的数量. 此外, 由于同一连通分量中的不同缺陷内容是对同一类软件缺陷的描述, 应具有一致性, 因此, 本文进一步加入聚类中心之间的距离项来优化剪枝的阈值. 聚类结果的平均距离计算方法如公式(15)所示.

$$disComp(Comp)=\frac{\sum disClus(i,Cent_1)}{M}+\frac{\sum disClus(j,Cent_2)}{N}+disCent(Cent_1,Cent_2) \tag{15}$$

$disClus(i,Cent_1)$表示第 1 类聚类结果中第 $i(i\in(1,2,\ldots,M))$个节点到该聚类中心的距离(本文使用余弦距离, $disClus(j,Cent_2)$ $(j\in(1,2,\ldots,N)$同理). $disClus(Cent_1,Cent_2)$表示聚类中心之间的距离. 应用公式(14)、(15), 本文最终确定的最优阈值分别为 $BestDS_{\text{op-rea}}$=1.8(对应剪枝阈值为 41.5%), $BestDS_{\text{rev-op}}$=1.7(对应剪枝阈值为 34.5%).

在分别获取了缺陷怀疑度和相关阈值后, ASD-TAOVE 方法可基于定义 3 构建 APP 软件缺陷异构图 $GD(O_d,E_d)$, 示例图如图 3 所示.

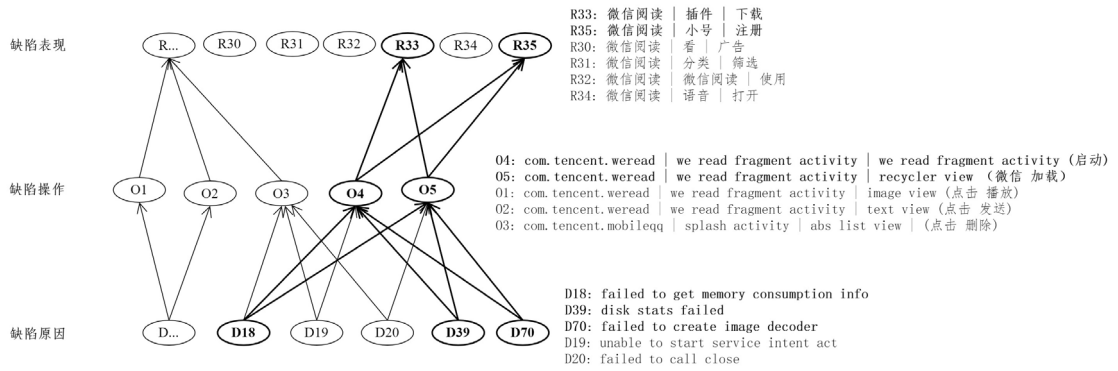

图 3 APP 软件缺陷内容异构图 GD (示例)

相较于简单地将相同的 APP 软件中的不同缺陷内容建立关联, 本文基于功能模块和使用过程细节对 APP 软件内容进行验证, 构建的 APP 软件缺陷内容异构图 GD 可表示更准确和细粒度的关联关系. 例如图 3 中加粗表示的缺陷原因(D18、D39、D70, 与图 2 中的示例相同), 经过验证后发现, 其与加粗部分的缺陷操作 O4: “com.tencent.weread|we read fragment activity|we read fragment activity (微信阅读 APP 软件中读者导航界面启动操作)”以及 O5: “com.tencent.weread|we read fragment activity|recycler view (微信阅读 APP 软件中读者导航界面加载操作)”相关, 而与其他类别的 APP 软件的缺陷操作无关, 例如 O3: “com.tencent.mobileqq|splash activity|abs list view (腾讯 QQ 软件中点击删除广告操作)”. 此外, GD 中还可以区分在相同 APP 软件中使用其不同功能模块的情况, 例如: O1、O2 分别表示微信阅读 APP 软件中读者导航功能模块(与 O4、O5 表示的功能模块相同)中对 imageview (图片组件)、textview (文本组件)的相关操作, 这与加粗示例的缺陷原因(D18、D39、D70)并无关联. 进一步地, 图 3 中显示了与微信阅读 APP 软件相关的缺陷表现类型(R30–R35), 经过验证后发现,缺陷表现 R33 (插件下载)、R35 (注册小号)与启动、加载相关的缺陷操作(O4、O5)相关, 相较于 R33、R35 来说, 其他类缺陷表现(R30–R34)与 O4、O5 的相关性较低. GD 的关联结果是符合用户使用 APP 软件过程的, 当 APP 软件功能模块启动时(O4), 相关资源会被加载(O5), 加载内容可能为插件(R33)、图片等资源, 加载过程失败的原因可能来自内存读取失败(D18)、硬盘读取失败(D39)以及图片解码失败(D70).

构建 GD 的算法流程如算法 1 所示, 输入为通过第 2.1 节获取的 APP 软件缺陷内容 D 以及缺陷事件因果图 G, 在一个初始化为空的异构图 GD 的基础上, 逐层验证并建立 APP 软件缺陷内容之间的关联, 最终根据验证和关联结果构建 APP 软件缺陷内容异构图 GD.

算法 1. APP 软件缺陷内容验证算法.

输入: 由第 2.1 节获取的 APP 软件缺陷内容 D 和缺陷事件因果图 G.

1. 初始化一个空的异构图 GD
2. 从 D 中初始化出缺陷操作类型节点 V_{DO}、缺陷表现类型节点 V_{DR}
3. 从 G 中初始化和描述类缺陷原因节点 V_{GO}、错误类及异常类缺陷原因类节点 V_{DC}、缺陷事件因果关系 $E(V_{GO},V_{DC})$
4. **for** v_i in V_{DO}
5. 　获取缺陷操作的内容怀疑度 CS_{op} //使用公式(2)–(4)计算
6. 　**for** v_j in V_{GO}
7. 　　获取缺陷操作节点与错误类、异常类缺陷原因类节点 v_j 之间的关联怀疑度 $RS_{op\text{-}rea}$ //使用公式(9)计算
8. 　　获取缺陷操作节点与错误类、异常类缺陷原因类节点 v_j 之间的缺陷怀疑度 $DS_{op\text{-}rea}$ //使用公式(10)计算
9. 获取缺陷操作节点与缺陷原因节点之间最优剪枝阈值 $BestDS_{op\text{-}rea}$ //使用公式(14)、(15)计算
10. 根据 $BestDS_{op\text{-}rea}$ 和 $E(V_{GO},V_{DC})$建立 GD 中缺陷操作节点 V_{DO} 和错误类、异常类缺陷原因类节点 V_{DC} 之间的边 $E(V_{DO},V_{DC})$
11. **for** v_k in V_{DR}
12. 　获取缺陷表现节点 v_k 的内容怀疑度 CS_{rev} //使用公式(5)–(8)计算
13. 　**for** v_i in V_{DO}
14. 　　获取缺陷表现节点 v_k 与缺陷操作节点 v_i 之间的关联怀疑度 $RS_{rev\text{-}op}$ //使用公式(11)、(12)计算
15. 　　获取缺陷表现节点 v_k 与缺陷操作节点 v_i 之间的缺陷怀疑度 $DS_{rev\text{-}op}$ //使用公式(13)计算
16. 获取缺陷表现节点与缺陷操作节点之间最优剪枝阈值 $BestDS_{rev\text{-}op}$ //使用公式(14)、(15)计算
17. 根据 $BestDS_{rev\text{-}op}$ 建立 GD 中缺陷操作节点 V_{DR} 和缺陷操作节点 V_{DO} 之间的边 $E(V_{DR},V_{DO})$

输出: 经过验证的 APP 软件缺陷内容异构图 GD.

2.3 APP软件缺陷跟踪分析

应用第 2.2 节的方法, 经过验证后的 APP 软件缺陷内容异构图(GD)包含了 APP 软件不同历史版本中的软件缺陷内容. 然而, APP 软件演化是一个复杂的过程, 相同的缺陷可能在下一版本得到了修复, 也可能未能得到修复, 未能得到修复的软件缺陷的缺陷原因可能与上个版本一致, 即未修复该软件缺陷, 也有可能是由于在演化过程中产生了其他的软件缺陷并间接导致了该软件缺陷表现相同. 大多数现有研究没有将不同历史版本的缺陷内容加以区分或仅利用版本更新日志分析版本演化过程, 导致缺陷分析结果有误, 当前缺乏一个有效的面向版本演化的 APP 软件缺陷跟踪分析框架.

定义 4(元路径). 可形式化为 $Od_{beginning} \xrightarrow{r_1} Od_1 \xrightarrow{r_2} Od_2 \ldots \xrightarrow{r_{n-1}} Od_{n-1} \xrightarrow{r_n} Od_{end}$, 表示起始缺陷内容节点 $Od_{beginning}$ 到终止缺陷内容节点 Od_{end} 之间的关系. 其中, n 为元路径中关系的数量, 任意缺陷内容节点 Od_i 关系 r_i 满足 $Od_i \in A$, $r_i \in R$.

基于上述问题, 本文提出了一个 APP 软件缺陷跟踪分析方法用于在版本演化过程中分析和评估软件缺陷. 方法首先根据设计的演化分析对象提取演化特征, 并基于 GD 建立缺陷操作的演化关系; 接着, 基于元路径(定义 4)的定义从 GD 中获取缺陷/演化元路径; 然后, 提出了一个 APP 软件缺陷跟踪分析模型, 结合深度学习方法分别提取并融合缺陷内容与缺陷/演化元路径的特征; 最终完成 APP 软件缺陷跟踪分析. 具体分为两个阶段.

(1) 缺陷/演化元路径提取

考虑到在版本演化过程中, 软件缺陷是否修复会对用户的使用过程产生直接影响, 缺陷表现及缺陷原因

分别是用户主观的使用反馈和 APP 软件的使用记录, 均与缺陷操作相关. 因此, 本文针对缺陷操作设计演化分析对象. 演化分析对象的作用是为了度量缺陷操作在 APP 软件版本演化过程中的变化程度, 反映了版本演化对用户使用 APP 软件过程产生的影响. 在缺陷操作的演化过程中, 相较于独立的操作, 操作路径能够更清晰地反映用户的操作意图(详见第 2.2 节), 操作路径的变化程度越大, 演化程度越高, 用户越容易发表相关的使用体验. 其次, 演化频次可以用于度量缺陷操作在演化过程中的重要程度, 若一条缺陷操作同时与多条缺陷操作存在演化关系, 则认为该缺陷操作在版本演化中的重要程度较高. 此外, 不同版本中用户操作的比例差异可以度量用户的关注程度: 若该缺陷操作的比例差异较低, 则认为用户在不同版本中对该操作记录对应的功能关注差异较小; 反之, 则认为用户关注差异较大. 本文基于缺陷操作设计的演化关系分析对象见表 3.

表 3 缺陷操作演化关系分析对象

演化分析对象	作用	对应公式
缺陷操作路径的变化程度	度量缺陷操作演化的程度, 变化程度越高, 演化程度越高	(16)、(17)
缺陷操作的演化频次	度量缺陷操作在演化过程中的重要程度, 演化频次越高, 重要程度越高	(16)
缺陷操作的比例差异	度量用户关注程度的变化, 变化越大, 用户关注度变化越大	(16)、(18)

基于公式(16), 可提取缺陷操作的演化特征:

$$Evo_{\mathrm{op}}=path_{\mathrm{level}}\cdot count_{\mathrm{op}}\cdot rate_{\mathrm{change}} \tag{16}$$

Evo_{op} 表示缺陷操作的演化特征; $path_{\mathrm{level}}$、$count_{\mathrm{op}}$、$rate_{\mathrm{change}}$ 分别表示缺陷操作路径的变化程度、缺陷操作的演化频次以及缺陷操作的比例差异, 具体计算公式如下:

$$path_{\mathrm{level}}=\begin{cases}0.01, & op_{\mathrm{pre}}=op'_{\mathrm{pre}}\wedge op_{\mathrm{last}}=op'_{\mathrm{last}}\\ 1, & (op_{\mathrm{pre}}=op'_{\mathrm{pre}}\wedge op_{\mathrm{last}}\neq op'_{\mathrm{last}})\vee(op_{\mathrm{pre}}\neq op'_{\mathrm{pre}}\wedge op_{\mathrm{last}}=op'_{\mathrm{last}})\\ 2, & op_{\mathrm{pre}}\neq op'_{\mathrm{pre}}\wedge op_{\mathrm{last}}\neq op'_{\mathrm{last}}\end{cases} \tag{17}$$

$$rate_{\mathrm{change}}=rate(op_{\mathrm{pre}})-rate(op_{\mathrm{last}}) \tag{18}$$

公式(17)中的 op_{pre}、op_{last} 分别表示上一版本中目标类操作的前驱和后继操作, op'_{pre}、op'_{last} 分别表示下一版本中目标类操作的前驱和后继类操作. 当目标类操作的前驱和后继操作在版本演化过程中均未发生变化时, $path_{\mathrm{level}}$ 为 0.01 (避免 $path_{\mathrm{level}}$ 为 0); 当前驱或后继其中之一发生变化时, $path_{\mathrm{level}}$ 的变化程度为 1; 当前驱和后继操作皆在版本演化过程中发生变化时, $path_{\mathrm{level}}$ 的变化程度为 2. 公式(18)中, 缺陷操作的比例差异 $rate_{\mathrm{change}}$ 表示为版本演化前后目标类操作在相同 APP 软件的功能模块中被用户操作的比例差异. 缺陷操作比例的具体计算方法如公式(19)所示.

$$rate_{\mathrm{op}}=\frac{Cpath_{\mathrm{Activity}}}{Cpath_{\mathrm{APP}}} \tag{19}$$

$Cpath_{\mathrm{Activity}}$ 表示该缺陷操作在 APP 软件功能模块(activity)中构成的操作路径次数, $Cpath_{\mathrm{APP}}$ 表示该缺陷操作在 APP 软件所有功能模块中构成的操作路径次数.

在获取了缺陷操作的演化特征后, 本文基于第 2.2 节获取的 APP 软件缺陷内容异构图(*GD*), 建立 *GD* 中缺陷操作的演化关系, 示例如图 4 所示.

根据本文提出的演化关系分析方法, 可建立如图 4 中黑色虚线所示的演化关系, 相较于只关注软件功能的增加或消失趋势, 本文建立的演化关系可表示在版本演化过程中 APP 软件的不同功能之间的互相影响情况, 有助于在版本演化过程中分析缺陷信息, 发现缺陷造成的潜在影响. 例如: 在用户操作数据中, 我们发现用户在使用某一版本的微信阅读 APP 软件阅读小说时的操作路径为(O4、O5、O1), 即启动、加载、播放. 在发生了一次版本演化后, 用户使用该功能的操作路径演化为(O5、O1、O5), 即加载、播放、加载, 这反映出用户在点击播放相关资源后, 软件可能出现卡顿或加载失败的问题, 此时可判断在版本演化过程中 O1 与 O4、O5 之间存在演化关系. 此信息可为软件开发团队明确可能受版本演化影响的软件功能模块范围. 为了获取更加准确和丰富的缺陷信息, 基于加入演化关系的 APP 软件缺陷内容异构图, 本文抽取两类元路径: 第 1 类为缺陷元路径, 用于表示不同 APP 软件缺陷内容在版本演化过程中的关联关系, 其组成形式为 DR-DO-DC-DO

或 DC-DO-DR-DO; 第 2 类为演化元路径, 用于表示缺陷操作的演化及具有演化关系的 APP 软件缺陷内容在版本演化过程中的关联关系, 其组成形式为 DO-DO-DC-DO-DR, DR-DO-DO-DC-DO-DR 等. 不同类型、组成的元路径所表示的关联关系类型和贡献各不相同, 可为缺陷分析提供不同的信息增益. 例如: 在演化元路径中, DO-DO-DC-DO-DR 表示缺陷操作演化关系及演化相关的缺陷内容的关联关系, 而 DR-DO-DO-DC-DO-DR 则可以表示缺陷表现和缺陷操作对缺陷操作的影响以及缺陷操作演化与相关缺陷内容的关联关系.

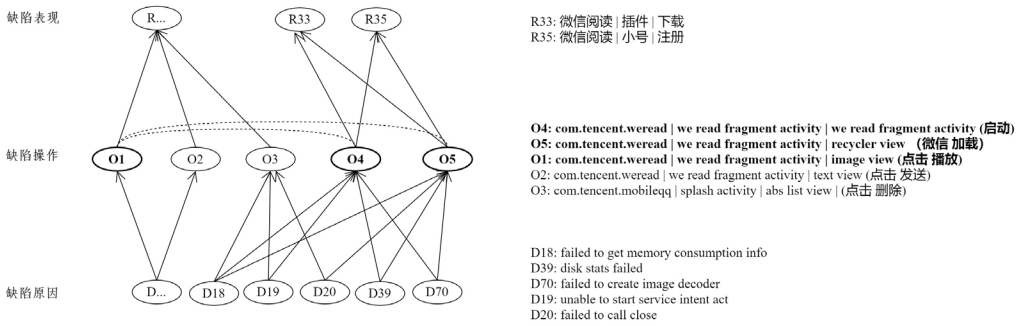

图 4 加入演化关系的 APP 软件缺陷内容异构图(示例)

(2) APP 软件缺陷跟踪分析模型

为了进一步挖掘在版本演化过程中 APP 软件缺陷内容之间互相影响的复杂关系, 设计了一个 APP 软件缺陷跟踪分析模型用于提取缺陷内容和缺陷/演化元路径的特征, 将其融合后用于分析在版本演化过程中的 APP 软件缺陷, 模型的整体流程如图 5 所示.

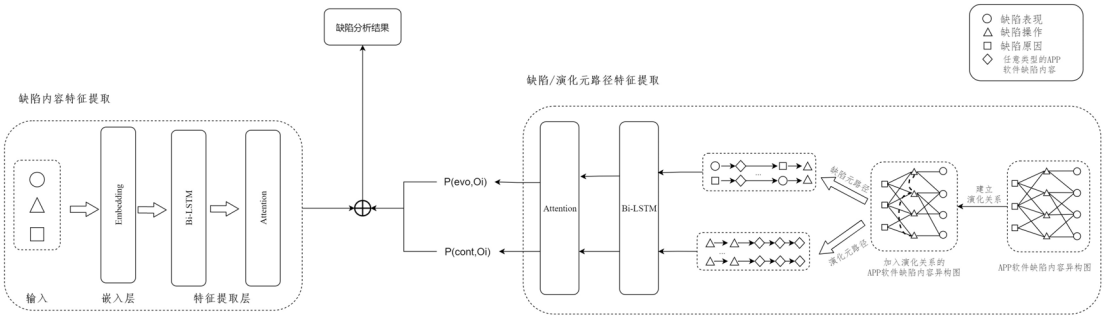

图 5 APP 软件缺陷跟踪分析模型框架

APP 软件缺陷跟踪分析模型主要包含缺陷内容特征提取和缺陷/演化元路径特征提取两部分.

- 缺陷内容特征提取: 首先, 由于 APP 软件缺陷内容是关系复杂的多源异构数据, 而 Metapath2Vec[41] 方法可以对异构信息网络中的信息进行有效地学习和表示, 所以本文在嵌入层使用 Metapath2Vec 提取 APP 软件缺陷内容异构图的顶点序列, 并获取其节点的向量表示(缺陷表现 DR, 缺陷操作 DO, 缺陷原因 DC); 接着, 深度学习模型长短期记忆网络(long short term memory, LSTM)与注意力机制(attention)可以提取缺陷内容深层次的语义和结构特征. 考虑到大部分的 APP 软件缺陷内容是由文本构成的, 并且不同的缺陷内容对分析结果的影响不同, 所以本文将 APP 软件缺陷内容视为自然语言序列, 使用双向 LSTM (Bi-LSTM)模型提取其语义特征, 使用 Attention 计算缺陷内容的贡献值完成缺陷内容的特征提取.
- 缺陷/演化元路径特征提取:由于不同类型的元路径的长度、组成元素不同, 其表示的缺陷内容关联类型和贡献度不同, 所以本文将元路径的节点序列视为自然语言序列, 基于 Bi-LSTM 模型学习元路径的关联表示, 并基于 Attention 提取其贡献度, 最终与缺陷内容特征融合完成 APP 软件缺陷跟踪分析.

本文特别讨论了元路径的类别、数量、生成方式(随机生成或按演化强度排序生成)对缺陷分析的影响, 详见第 4 节针对 RQ3 的实验结果分析.

APP 软件缺陷跟踪分析模型实现了一个面向版本演化的 APP 软件缺陷分析的通用框架, 可从 APP 软件缺陷内容中无监督地提取有效的演化特征, 并使用缺陷/演化元路径的方式表示缺陷内容之间的关联; 接着, 基于异构信息网络和深度学习模型进一步提取特征; 最终, 将 APP 软件缺陷内容特征和缺陷/演化元路径特征融合完成缺陷跟踪分析. 根据对缺陷操作的演化分析可获取下一版本的 APP 软件中与缺陷相关的缺陷操作内容, 并基于第 2.2 节构建的 *GD* 获取相关的缺陷表现、缺陷原因以组成更丰富、有效的缺陷报告, 辅助开发人员快速、准确地理解和修复软件缺陷.

3 实验设计

考虑到提升 APP 软件缺陷报告的质量需要考虑信息孤立、数据噪声大、版本失配这 3 个方面的问题, 为了全面评估本文提出的 ASD-TAOVE 方法的有效性, 本文设计了如下 4 个研究问题(research question, RQ).

- RQ1: 本文第 2.2 节提出的 APP 软件缺陷内容验证方法是否有效并优于其他验证方法?
- RQ2: 本文第 2.3 节提出的 APP 软件缺陷跟踪分析方法如何影响缺陷分析效果?
- RQ3: APP 软件缺陷跟踪分析方法中相关的演化关系变量如何影响缺陷分析的效果?
- RQ4: ASD-TAOVE 方法是否能够为开发人员提供丰富、有效的缺陷修复建议?

其中: RQ1 与 RQ2 分别可以评估 ASD-TAOVE 方法在处理数据噪声和版本失配问题的性能; 由于异构信息网络中元路径的类型、生成方式、数量等因素均会影响演化关系, 从而影响缺陷分析的效果, 因此本文设计了 RQ3 分析相关因素在缺陷分析时的作用; 最后, 信息孤立问题主要表现为信息粒度粗以及在版本演化过程中软件缺陷识别效果差两方面, 因此本文设计 RQ4, 从这两个方面评估 ASD-TAOVE 在缺陷修复建议的有效性.

3.1 实验设置

本文针对 RQ1–RQ4 设置了 4 组实验, 实验设置见表 4.

表 4 针对不同研究问题的实验设置

实验组别	研究问题	实验设计
第 1 组	RQ1	对比本文提出的 APP 软件缺陷内容验证方法与不使用任何验证方法以及其他典型验证方法时的缺陷识别准确率
第 2 组	RQ2	对比本文提出的 APP 软件缺陷跟踪分析方法提取的不同类型的信息作为信息增益与不使用任何跟踪分析方法时的缺陷识别准确率
第 3 组	RQ3	分析不同生成方式、数量的元路径对缺陷识别准确率的影响
第 4 组	RQ4	对比基线方法与提出的 ASD-TAOVE 方法在缺陷修复建议内容上的不同

由于在版本演化过程中, 同一缺陷内容可能引起不同的软件缺陷, 从而生成不同的缺陷报告, 因此本文将训练过程设计为多标签的分类任务, 使用交叉熵损失函数和 Adam 算法优化神经网络的训练过程, 训练集和测试集的比例为 7.5∶2.5. 为了使实验具有一定的代表性, 本文将两类深度学习模型(TextCNN-attention, Bi-LSTM-attention)[13]与实验涉及的方法相结合进行讨论.

本文使用 *Sigmoid* 函数将分类结果转化为 0–1 之间的概率, 使用准确率(*Acc*)来评价训练过程, 具体计算过程如公式(20)所示.

$$Acc=\frac{\sum flag(Pred,GT)}{Total} \tag{20}$$

其中, *Pred* 表示经过 *Sigmoid* 函数计算后的预测概率; *Total* 表示测试数据集的数量; $\sum flag(Pred,GT)$ 用于判断分类结果是否正确, 本文定义为: 当分类结果中至少包含 1 个与 *GT* 一致时, 即认为分类正确.

需要特别注意的是: 在 RQ4 中, 本文将对比 ASD-TAOVE 与基线方法在生成缺陷修复建议时的有效性.

由于部分基线方法可能重复识别缺陷内容, 例如 ARICA 方法根据用户评论的分类、长度、情感倾向和主题推荐在版本演化中重要的评论, 而不同的推荐评论可能反馈相同的软件缺陷, 若推荐评论反馈的软件缺陷大量重复, 则不能为开发人员提供有效的修复建议. 因此, 除了准确率外, 本文使用召回率(*recall*)以及无重复召回率(*unirecall*)来评价缺陷修复建议内容的有效性, 如公式(21)、(22)所示.

$$Recall = \frac{TP}{TP + FN} \tag{21}$$

$$UniRecall = \frac{Uni(TP)}{TP + FN} \tag{22}$$

公式(21)中, *TP* 为预测正确的样本数量, 表示当前版本生成的缺陷修复建议与下一版本的软件缺陷一致; *FN* 为预测错误的负样本数量, 表示当前版本生成的缺陷修复建议与下一版本的软件缺陷不一致的情况. 召回率可以从一定程度上评估针对当前版本生成的缺陷修复建议的有效性.

公式(22)中, *Uni*(*TP*)为去重后 *TP*. 去重召回率可从缺陷修复中包含的缺陷类型角度评估缺陷修复建议的有效性. 召回率和无重复召回率表现了方法在 APP 软件版本演化过程中识别软件缺陷的能力.

3.2 训练数据

本文使用爬虫工具从 APP 软件应用市场(360 手机助手)采集了 200 万条用户评论. 此外, 本文开发了用户操作和日志数据采集工具(APP 用户行为监测分析软件 V1.0, 软件著作权登记号: 2022SR1600335), 在 APP 软件使用过程中收集了 50 万条用户操作数据和 50 万条软件日志数据, 随机选择了 300 000 条数据(用户评论、用户操作、软件运行日志数据数量一致). 由于不同 APP 软件、不同版本的数据量差异较大, 我们从中选择了数据量较为均衡的 3 个 APP 软件(微信阅读、哔哩哔哩、腾讯视频)的数据, 并根据其版本进行划分. 其中, 考虑到更新频繁的版本之间差异较小, 因此本文根据二级版本代号进行划分, 例如 3.1.0–3.1.x 之间视为同一版本. 为在版本演化过程中分析软件缺陷, 本文使用 3 个 APP 软件在 3 个版本中的软件数据, 见表 5.

表 5 APP 软件版本及数据使用情况

APP 软件版本	用户评论	用户操作	软件运行日志
微信阅读 V5.0	950	40 000	150 000
微信阅读 V6.0	1 160	45 000	180 000
微信阅读 V7.0	100	–	–
哔哩哔哩 V7.1	990	12 000	10 000
哔哩哔哩 V7.2	1 230	16 000	13 000
哔哩哔哩 V7.4	100	–	–
抖音 V15.0	2 000	140 000	180 000
抖音 V18.0	1 770	85 000	120 000
抖音 V20.0	130	–	–

考虑到用户在使用 APP 软件过程中并不是独立地使用单独的 APP 软件, 而会与第三方软件进行交互, 因此对于用户操作数据和软件运行日志数据, 本文使用的是用户在使用目标 APP 软件时的全部相关数据. 在分析版本演化过程中的软件缺陷时, 使用每个 APP 软件前两个版本的数据分别作为训练集和测试集, 例如: “微信阅读 V5.0”软件数据作为训练集, “微信阅读 V6.0”软件数据作为测试集.

特别地, 针对 RQ4, 为了评估缺陷修复建议的有效性, 本文使用每个 APP 软件较近 2 个版本的软件数据作为训练集, 对该 APP 软件在第 3 个版本中的用户评论进行人工标注, 获得其中由用户反馈的软件缺陷作为测试集. 例如: 对于“抖音”APP 软件, “抖音 V15.0”与“抖音 V18.0”的所有 APP 软件数据作为训练集, “抖音 V20.0”的用户评论数据经过人工标注后的结果作为测试集. 由于 RQ4 的实验中均采用人工标注用户评论中包含的软件缺陷的方式进行评估, 无须用户操作数据和软件运行日志数据, 因此, 本文并未给出第3个版本的相关数据.

本文基于不同版本的缺陷操作获取 Ground Truth (*GT*). 具体来说, 我们获取当前版本及下一版本软件的缺陷操作, 并使用 K-Means 对所有缺陷操作进行聚类. 如此, 相似的缺陷操作将会被归并到同类中. 接着, 分

析聚类结果中操作路径的版本分布情况并进行人工提取. 当缺陷操作路径仅在当前版本发生, 而在下一版本不发生, 或仅在下一版本发生, 而在当前版本未发生时, 则认为该缺陷操作路径发生了演化, 标注发生演化的缺陷操作路径上包含的缺陷操作为 *GT*. 最后, 本文基于 Metapath2Vec 提取 APP 软件缺陷内容异构图的顶点序列并对其进行预训练, 获取其节点的向量表示.

4 结果分析

4.1 RQ1的实验结果分析

- RQ1: 本文第 2.2 节提出的 APP 软件缺陷内容验证方法是否有效并优于其他验证方法?

为了分析 APP 软件缺陷内容验证方法对多源、异构的 APP 软件缺陷内容的提取及噪声过滤效果, 本文将与不进行 APP 软件缺陷内容验证以及噪声过滤的基线方法在缺陷识别任务中的准确率进行比较. 噪声过滤的基线方法选择 FARSEC 方法[18], 该方法将文本挖掘技术与传统分类算法相结合用于噪音数据过滤. 本文将分别分析不进行 APP 软件缺陷内容验证、FARSEC 方法与本文提出的 APP 软件缺陷内容验证方法结合两类深度学习模型时的缺陷分析效果. 其中, 不进行 APP 软件缺陷内容验证时, 相同 APP 软件的不同缺陷内容直接构成 APP 软件缺陷内容异构图, 不进行剪枝; 使用缺陷怀疑度的方法时, 则根据本文于第 2.2 节提出的缺陷怀疑度对 APP 软件缺陷内容进行验证, 并对噪声和关联性较弱的节点和边进行剪枝. 此外, 第 1 组实验均在未考虑 APP 软件跟踪分析的情况下进行, 实验结果见表 6.

表 6 不同验证方法与两类深度学习模型结合的缺陷识别准确率(%)

扩展的深度学习模型	准确率		
	不进行 APP 软件缺陷内容验证	FARSEC	本文的方法(缺陷怀疑度)
TextCNN-attention	73.74	72.12	**82.86**
Bi-LSTM-attention	74.74	72.29	**83.93**

如表 6 所示, 本文提出的 APP 软件缺陷内容验证方法在两类深度学习模型中均达到了最高的缺陷识别准确率, 其中, 使用Bi-LSTM-attention时达到最高的识别准确率. 当不进行缺陷内容验证时, 缺陷内容仅由APP软件类别而被关联起来, 忽略了软件缺陷内容中存在的噪声和错误, 关联分析较为简单, 导致分析效果有限. 而 FARSEC 方法在使用 TF-IDF 方法过滤缺陷内容中相似度较低的部分时, 没有充分考虑缺陷内容之间的语义和结构特征, 未能挖掘缺陷内容之间的潜在关联, 导致其提取到的特征质量不佳. 本文提出的方法基于怀疑度的计算, 进一步将怀疑度分为内容怀疑度和关联怀疑度, 分别挖掘缺陷内容的内部特征和关联特征, 降低了由于类不平衡问题带来的干扰, 有助于较为准确地分析缺陷内容中的噪声和错误, 并针对共同的缺陷类别建立其关联. 此外, 使用 Bi-LSTM-attention 模型可以有效地挖掘 APP 软件缺陷内容的语义特征和不同缺陷内容的贡献度.

实验结果表明: ASD-TAOVE 中的 APP 软件缺陷内容验证方法可基于提出的缺陷怀疑度有效过滤 APP 软件缺陷内容中的噪声, 相较于不进行 APP 软件缺陷内容验证与基线的噪声过滤方法 FARSEC, 缺陷识别准确率分别提升 9.12%和 10.74%. 此外, 扩展的深度学习模型比较结果显示: Bi-LSTM-attention 在缺陷识别任务中的表现更优, 平均准确率可提升 0.74%.

4.2 RQ2的实验结果分析

- RQ2: 本文第 2.3 节提出的 APP 软件缺陷跟踪分析方法如何影响缺陷分析效果?

由于本文提出的 APP 软件缺陷跟踪分析方法将提取的演化关系转化为缺陷/演化元路径, 该元路径是以信息增益的方式对缺陷跟踪分析起作用的, 因此, 本文针对 RQ2 设计实验, 对比在不同信息增益方式时缺陷分析的效果, 以验证 APP 软件缺陷跟踪分析的效果. 其中, APP 软件缺陷内容的表示将分为两类, 以验证本文提取的演化关系在不同的表示方法中的有效性: 第 1 类为开源的词嵌入模型表示[42], 第 2 类为使用 Metapath2Vec 预训练的节点表示. 在无信息增益时, 仅使用 APP 软件缺陷内容进行缺陷分析; 在使用元路径

进行信息增益时, 均选择 3 条进行实验, 在使用演化元路径时, 则选择演化程度最高的前 3 条进行实验. 实验结果见表 7.

表 7 不同的信息增益类型与两类深度学习模型结合的缺陷识别准确率(%)

表示方式	信息增益类别	准确率	
		TextCNN-attention	Bi-LSTM-attention
词嵌入	无信息增益	78.40	78.74
	演化元路径	82.47	83.11
Metapath2Vec	无信息增益	82.86	83.93
	缺陷元路径	83.72	84.92
	演化元路径	86.66	87.49
	缺陷元路径+演化元路径	**87.58**	**88.9**

如表 7 所示, 相较于不使用任何信息增益的情况, 无论是使用传统的词嵌入模型还是 Metapath2Vec 与训练的节点表示, 本文获取的演化关系均能显著提升在面向版本演化时的缺陷识别准确率(词嵌入约 5%, Metapath2Vec 约 4%). 这表明, 本文提出的 APP 软件缺陷跟踪分析方法能够有效获取并分析演化关系. 在使用 Metapath2Vec 的情况下, 相较于无信息增益, 缺陷元路径在不同类型的深度模型中带来的准确率并不显著(约为 1%). 缺陷识别准确率在使用 Bi-LSTM-attention 模型, 同时使用缺陷元路径和演化元路径时达到最高, 表明缺陷元路径和演化元路径均有助于在版本演化过程中分析软件缺陷. 此外, 本文提出的面向版本演化的 APP 软件缺陷分析模型可以分别提取 APP 软件缺陷内容的特征、贡献度以及缺陷/演化元路径的特征、贡献度, 并将其融合完成缺陷分析.

实验结果表明: ASD-TAOVE 中关注的演化元路径是有效的信息增益, 可结合不同的嵌入方式提升缺陷分析的效果, 当结合缺陷、演化元路径和 Metapath2Vec 嵌入时识别效果达到最佳, 相较于使用传统词嵌入方式并不加入任何信息增益时, 缺陷识别准确率最大可提升 9.18%. 这表明, ASD-TAOVE 可通过建立演化关系并将其转化为缺陷/演化元路径的方式提升面向版本演化的 APP 缺陷分析效果.

4.3 RQ3的实验结果分析

- RQ3: APP 软件缺陷跟踪分析方法中相关的演化关系变量如何影响缺陷分析的效果?

由于演化关系可以以不同类型、不同数量的元路径的方式对缺陷分析效果产生影响, 为进一步讨论演化关系对缺陷分析的影响, 第 3 组实验将对比在不同生成方式及元路径数量情况下的缺陷识别准确率. 为保证客观性, 第 3 组实验均使用 APP 软件缺陷跟踪模型. 当使用随机生成的方式产生元路径时, 从候选元路径中随机抽取 K 条元路径进行实验; 当以排序方式产生元路径时, 根据候选元路径的演化强度进行排序, 选择前 K 条元路径进行实验. 实验结果见表 8.

表 8 使用不同生成方式和数量的元路径时的缺陷识别准确率(%)

生成方式	准确率			
	K=1	K=2	K=3	K=4
随机	82.47	81.10	77.46	75.79
排序	**85.68**	**86.36**	**87.42**	**87.49**

当随机选择元路径时, 随着使用的元路径数量增加, 缺陷识别准确率下降. 这是因为随机选择的演化元路径可能与当前缺陷内容相关度较低, 包含的演化关系存在噪声和错误, 使得准确率降低; 相反地, 当使用本文提出的 APP 软件缺陷跟踪分析方法时, 不同缺陷类别的演化程度以及不同缺陷内容的缺陷怀疑度被定量分析, 从而可以对演化关系进行排序和筛选, 选择其中较为准确的元路径. 因此, 当基于演化强度对元路径进行排序并选择前 K 条元路径并且 K 满足条件 $K\leqslant 3$ 时, 缺陷识别准确率随着使用数量的增加而提高.

针对 RQ3 的实验结果表明, 元路径的生成方式和数量的确会影响缺陷分析的效果. 当排序生成的元路径数量小于 3 条($K\leqslant 3$)时, 每增加 1 条元路径, 缺陷识别准确率平均提升 0.87%; 当该数量大于 3 条($K>3$)时, 准确率提升仅为 0.07%. 由于增加演化元路径的数量会增加模型的训练成本, 为综合考虑训练成本和缺陷分析

效果, 选择排序生成 3 条(K=3)元路径时较为合适.

4.4 RQ4的实验结果分析

RQ4: ASD-TAOVE 方法是否能够为开发人员提供丰富、有效的缺陷修复建议?

除了验证 ASD-TAOVE 方法在缺陷识别准确率的表现, 本文还关注方法是否能够为开发人员提供高质量的缺陷报告. 因此, 本文针对 RQ4 对比了 ASD-TAOVE 方法与基线方法的分析粒度以及包含的缺陷修复建议内容, 以进一步评估方法的有效性. 选择的基线方法如下, 对比结果见表 9.

(1) ARICA[2]方法基于用户评论的分类、主题、情感倾向等属性推荐在版本演化过程中重要的评论, 并基于软件更新日志评估推荐的用户评论的有效性.

(2) ATA 方法[9]利用机器学习算法, 以版本更新记录为对象, 自动分析 APP 软件的开发和更新趋势.

(3) Arab[3]方法基于语义感知, 从 APP 软件评论文本中提取有缺陷的功能, 包括用户操作和 APP 异常行为, 并挖掘两者之间的关联关系.

(4) APPTRACKER[21]方法可跟踪 APP 软件中不良版本的更新, 即, 是否在下一版本中包含的负面评论更多.

(5) Caspar[43]方法可从 APP 用户评论中提取有序事件, 事件可分类为用户操作问题与应用程序问题, 并将两类问题组成软件缺陷内容, 形成缺陷报告.

表 9 SD-TAOVE 方法与基线方法的分析粒度与缺陷修复建议内容

方法	分析对象				缺陷修复建议内容	粒度
	用户评论	用户操作	APP 软件行为	更新日志		
ARICA	√	–	–	√	推荐评论	★★
ATA	–	–	–	√	版本更新趋势	★
APPTRACKER	√	–	–	√	版本更新趋势	★
Arab	√	–	–	–	用户操作、异常行为	★★★
Caspar	√	–	–	–	用户操作、异常行为	★★★
ASD-TAOVE	√	√	√	–	版本演化中的缺陷表现、缺陷操作、缺陷原因	★★★★★

如表 9 所示, 根据方法使用的数据和缺陷修复建议内容, 可将上述基线方法分为两类. 其中, ARICA、ATA、APPTRACKER 方法基于用户评论及更新日志分析软件缺陷, 获取缺陷的修复建议内容包括推荐评论、版本更新趋势. 这类修复建议的粒度较粗, 不足以帮助开发人员理解和修复缺陷. 例如, 开发人员无法从 ARICA 方法推荐的用户评论或由 ATA 方法获取的版本更新趋势了解缺陷产生的具体步骤和原因. 相较于上述方法, Arab、Caspar 方法基于自然语言处理技术, 从用户评论中提取与缺陷相关的用户操作和异常行为, 提高了修复建议内容的粒度. 然而, 文献[43]中也指出: 用户评论的质量参差不齐, 信息杂乱、信息孤立等问题会导致方法的效果受到影响. 此外, Arab、Caspar 方法在分析软件缺陷时忽略了版本演化过程中缺陷内容之间的影响关系. 相较于上述基线方法, 本文提出的 ASD-TAOVE 方法基于多源、异构的 APP 软件数据分析软件缺陷, 对 APP 软件缺陷内容进行验证和关联性分析, 并考虑版本演化过程中缺陷内容的演化关系, 可对缺陷信息进行筛选和互补, 从而可为开发人员提供细粒度、准确并具有演化关系的缺陷修复建议内容.

除了评估缺陷修复建议的粒度, 为了进一步评估 ASD-TAOVE 方法生成的缺陷修复建议内容的有效性, 本文从上述两类基线方法中分别选择典型方法进行实验, 对比在版本演化过程中不同方法识别软件缺陷的效果. 其中, ARICA 方法可获取较为明确的推荐用户评论, 而 ATA、APPTRACKER 方法获取的内容为版本更新趋势. 本文选择更易于统一评估标准的 ARICA 方法. 而对于 Arab 和 Caspar 方法, 本文选择较为新颖的 Arab 方法进行对比. 需要特别说明的是: ARICA 方法使用版本更新日志来评估推荐评论的有效性, 而文献[1]指出, 版本更新日志中对用户反馈的软件缺陷响应比例不足 20%, 这表明版本更新日志仅对少量的软件缺陷进行了回应, 无法全面评估缺陷修复内容的有效性. 为统一评估标准, 本文人工标记测试数据集中用户评论反馈的软件缺陷, 并将其作为缺陷修复建议内容有效性的评估基准. 此外, 由于 ARICA 与 Arab 方法识别的软件缺陷

数量受限于用户评论的质量和数量, 而 ASD-TAOVE 方法可通过演化关系和关联性分析从 APP 软件缺陷内容异构图中识别更多包含软件缺陷的用户评论, 我们直觉上认为, ASD-TAOVE 方法可从有限数量的用户评论中识别更多有效的软件缺陷. 因此, 我们在实验中记录了不同方法在相同的训练数据中获取的缺陷修复建议数量(number of suggestion, *NumOfSug*). 考虑到 ARICA 方法可根据重要性推荐指定数量的用户评论(文献[2]中该数量为 10), 为充分比较实验结果, 本文为 ARICA 方法设定了不同的推荐评论数量(10, 20, 30)进行对比, 实验结果见表 10.

表 10 缺陷修复建议内容有效性评估实验结果

APP	方法	*NumOfSug*	*Recall* (%)	*UniRecall* (%)
抖音	ARICA	10	12.04	8.70
		20	12.56	9.23
		30	13.42	9.23
	Arab	**151**	6.89	6.89
	ASD-TAOVE	50	**14.09**	**14.09**
微信读书	ARICA	10	53.33	16.67
		20	56.62	18.11
		30	57.58	19.23
	Arab	**106**	7.46	7.46
	ASD-TAOVE	35	**84.67**	**25.00**
哔哩哔哩	ARICA	10	11.29	5.71
		20	13.67	6.59
		30	15.01	6.59
	Arab	**174**	6.43	6.43
	ASD-TAOVE	71	**23.86**	**14.49**

在表 10 显示的实验结果中, ASD-TAOVE 方法在 3 个 APP 数据集的各个指标均取得了最优的表现. 其中, ARICA 方法基于用户评论的类别、长度、情感等属性进行评论推荐, 这使得同类别中较长的差评评论较容易被推荐, 而被推荐的评论可能反馈相同软件缺陷内容, 这使得 ARICA 方法的无重复召回率较低. Arab 方法基于用户评论的语义及相关属性识别用户操作和异常行为, 该方法容易受到评论质量的影响. 我们发现, 超过 50%的评论中不包含用户操作和异常行为标签, 这会影响 Arab 方法的识别效果. 此外, Arab 方法虽然可以根据用户评论训练集发现较多类型的缺陷类别修复建议数量, 但忽略了版本演化过程中缺陷内容的变化, 导致其召回率和无重复召回率均取得了较差的结果. 相较于 ARICA 和 Arab 方法, ASD-TAOVE 对多个来源的 APP 软件数据进行了验证, 并在版本演化过程中分析其关联关系, 使得 ASD-TAOVE 方法在版本演化过程中识别软件缺陷的能力更强.

表 10 所示实验结果除了能够表示不同方法在 APP 版本软件演化过程中识别软件缺陷的能力, 还可以分析不同 APP 软件的缺陷修复效果. 以 ASD-TAOVE 方法的实验结果为例: 对于“抖音”APP 软件来说, 存在 14.09%用户反馈的软件缺陷尚未得到修复; 而对于“微信读书”APP 软件来说, 该比例达到了 84.67%. 这表明“抖音”APP 软件开发团队对于用户反馈的软件缺陷的响应速度更快, 缺陷修复能力更强.

综上所述, 相较于基线方法, ASD-TAOVE 方法既能够较全面地识别在版本演化过程中未修复的软件缺陷, 还可针对这些待修复的软件缺陷分析细粒度的缺陷修复建议, 帮助开发人员快速理解和修复软件缺陷.

5 有效性威胁

在数据标注阶段, 本文通过人工标注的方式获取不同版本 APP 软件用户操作的演化关系, 标注过程可能存在误差, 但本文的第一、第三作者对 200 条人工标注的数据进行抽样检查后未发现标注错误, 其结果可作为数据集可靠性的参考.

在数据预处理阶段, 本文分别使用 jieba (版本: 0.42.1)以及 nltk (版本: 3.7)分词工具对数据集中的中文和英文文本进行分词和词性标注. 由于 APP 软件文本差异性大, 这些工具的分词结果存在一定的误差, 但它们均被广泛使用和长期维护, 因此具有一定的可靠性基础. 此外, 本文通过构建 APP 软件功能词典来降低分词工具带来的误差, 后续工作需要人工介入继续丰富该词典, 以进一步提升数据预处理的可靠性.

在方法验证阶段, 本文选取 APP 软件版本较为均衡的数据集, 避免了数据不均衡问题, 并在丰富的基线方法和机器学习模型中进行方法评估, 因此性能数据相对可靠.

然而, 本次实验仅在 3 个 APP 软件的 3 个不同版本数据集上进行探究, APP 软件开发团队的更新风格差异可能会导致同一 APP 软件在不同版本的数据集以及不同 APP 软件的数据集中存在噪音, 因此, 方法的有效性有待在更丰富的 APP 软件类型、版本数据集上进行验证.

6 总 结

本文提出一种面向版本演化的 APP 软件缺陷跟踪分析方法 ASD-TAOVE. 方法首先从多源、异构的 APP 软件数据中抽取 APP 软件缺陷内容, 并基于事件关系首先建立了异常日志数据中缺陷事件的因果关联, 接着设计了一个缺陷内容验证方法, 基于 APP 软件数据特点设计不同的怀疑对象, 结合信息熵定量分析 APP 软件缺陷内容并构建缺陷内容异构图. 之后, 设计了一个 APP 软件缺陷跟踪分析方法, 在版本演化过程中分析和评估缺陷内容, 根据演化分析对象分析缺陷操作的演化程度, 建立演化关系并转化为缺陷/演化元路径. 最后, 通过一个基于深度学习的异构信息网络模型(APP 软件缺陷跟踪分析模型)提取缺陷内容和缺陷/演化元路径的特征并融合, 完成面向版本的 APP 软件缺陷跟踪分析.

本文设计了 4 个研究问题(RQ)对 ASD-TAOVE 方法的各个子过程的有效性进行验证和讨论. 首先, APP 软件缺陷内容验证方法从多个来源的 APP 软件数据中提取的 APP 软件缺陷内容是丰富及有效的, 应用提出的缺陷怀疑度可对提取的 APP 软件缺陷内容进行定量分析, 挖掘其中潜在的关联关系, 使得缺陷内容之间可以互相验证及补充以过滤噪声和错误, 平均缺陷识别准确率提升约 9.9%; 其次, APP 软件缺陷跟踪分析方法可以分析和评估历史版本的缺陷演化信息, 获取的演化关系在 APP 软件缺陷内容异构图的基础上可进一步转化为元路径以表示缺陷内容节点及其相关的其他缺陷内容信息, 使得缺陷识别准确率进一步提升(约 5%). 相较于同类的基线方法, ASD-TAOVE 对多个来源的 APP 软件数据进行了验证, 并在版本演化过程中分析其关联关系, 既能够较全面地识别在版本演化过程中未修复的软件缺陷, 又可为未修复的软件缺陷分析细粒度的缺陷修复建议, 有效缓解了信息孤立、数据质量差、版本失配等问题, 提升了缺陷报告的质量.

考虑到 APP 软件数据不同版本的数据质量以及不平衡问题, 本文的目前工作还存在局限: (1) 本文目前验证并讨论了方法在 APP 软件缺陷内容数据集较为均衡情况下的表现, 方法在不均衡的数据集下的鲁棒性及适用范围还有待进一步验证; (2) 本文提出了一种面向版本演化的 APP 软件缺陷分析通用框架, 以缺陷操作为例设计相关演化关系分析对象并量化演化关系, 然而还存在用户偏好演化等更丰富的演化关系需进一步探索. 下一步, 我们将探索方法在大规模 APP 软件数据中的鲁棒性, 并探索不同类型的演化关系, 进一步明确 APP 软件演化的度量方法, 为开发人员提供全面的演化建议.

References:

[1] Qian Y, Cao EY, Deng WJ, Yuan H. Mining the participatory role of massive user reviews in the update design of APP software. Systems Engineering-theory & Practice, 2021, 41(3): 554−564 (in Chinese with English abstract). [doi: 10.12011/SETP2019-1136]

[2] Xiao JM, Chen SZ, Feng ZY, Liu PL, Xue X. An automatic analysis of user reviews method for APP evolution and maintenance. Chinese Journal of Computers, 2020, 43(11): 2184−2202 (in Chinese with English abstract). [doi: 10.11897/SP.J.1016.2020.02184]

[3] Wang YW, Wang JJ, Shi L, Wang Q. Semantic-aware and fine-grained App review bug mining approach. Ruan Jian Xue Bao/Journal of Software, 2023, 34(4): 1613−1629 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/6697.htm [doi: 10.13328/j.cnki.jos.006697]

[4] Duan WJ, Jiang Y. Defect recognition of APP software based on user feedback. Computer Science, 2020, 47(6): 44−50 (in Chinese with English abstract). [doi: 10.11896/jsjkx.191100133]

[5] Zhou HC, Guo ZQ, Mei YQ, Li YH, Chen L, Zhou YM. Watch out for version mismtaching and data leakage! A case study of their influence in bug report based bug localization models. Ruan Jian Xue Bao/Journal of Software, 2023, 34(5): 2196−2217 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/6401.htm [doi: 10.13328/j.cnki.jos.006401]

[6] Wen M, Wu RX, Cheung SC. Locus: Locating bugs from software changes. In: Proc. of the 31st IEEE/ACM Int'l Conf. on Automated Software Engineering. Singapore: ACM, 2016. 262−273. [doi: 10.1145/2970276.2970359]

[7] Lam AN, Nguyen AT, Nguyen HA, Nguyen TN. Bug localization with combination of deep learning and information retrieval. In: Proc. of the 25th IEEE/ACM Int'l Conf. on Program Comprehension. Buenos Aires: IEEE, 2017. 218−299. [doi: 10.1109/ICPC.2017.24]

[8] Xiao Y, Keung J. Improving bug localization with character-level convolutional neural network and recurrent neural network. In: Proc. of the 25th Asia-Pacific Software Engineering Conf. Nara: IEEE, 2018. 703−704. [doi: 10.1109/APSEC.2018.00097]

[9] Zhong RY, Wang C, Liang P, Luo Z. Automatic trend analysis of mobile app updates based on app changelogs. Journal of Computer Research and Development, 2021, 58(4): 763−776 (in Chinese with English abstract). [doi: 10.7544/issn1000-1239.2021.20200756]

[10] Siers MJ, Islam MZ. Software defect prediction using a cost sensitive decision forest and voting, and a potential solution to the class imbalance problem. Information Systems, 2015, 51: 62−71. [doi: 10.1016/j.is.2015.02.006]

[11] Wu F, Jing XY, Dong X, Cao JC, Xu MW, Zhang HY, Ying S, Xu BW. Cross-project and within-project semi-supervised software defect prediction problems study using a unified solution. In: Proc. of the 39th IEEE/ACM Int'l Conf. on Software Engineering Companion (ICSE-C). IEEE, 2017. 195−197. [doi: 10.1109/ICSE-C.2017.72]

[12] Wijayasekara D, Manic M, Wright JL, McQueen M. Mining bug databases for unidentified software vulnerabilities. In: Proc. of the 5th Int'l Conf. on Human System Interactions. IEEE, 2012. 89−96. [doi: 10.1109/HSI.2012.22]

[13] Zheng W, Chen JZ, Wu XX, Chen X, Xia X. Empirical studies on deep-learning-based security bug report prediction methods. Ruan Jian Xue Bao/Journal of Software, 2020, 31(5): 1294−1313 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5954.htm [doi: 10.13328/j.cnki.jos.005954]

[14] Jiang Y, Mu CG, Su XH, Wang TT. Security bug report detection via noise filtering and deep learning. Chinese Journal of Computers, 2022, 45(8): 1794−1813 (in Chinese with English abstract). [doi: 10.11897/SP.J.1016.2022.01794]

[15] Chen JF, Wang XL, Cai SH, Xu JP, Chen JY, Chen HB. A software defect prediction method with metric compensation based on featureselection and transfer learning. Frontiers of Information Technology & Electronic Engineering, 2022, 23(5): 715−732. [doi: 10.1631/fitee.2100468]

[16] Wan Z, Xia X, Hassan AE, Lo D, Yin JW, Yang XH. Perceptions, expectations, and challenges in defect prediction. IEEE Trans. on Software Engineering, 2018, 46(11): 1241−1266. [doi: 10.1109/TSE.2018.2877678]

[17] Zhang T, Sun XB, Zheng ZB, Li G. Intelligent analysis for software data: Research and applications. Frontiers of Information Technology & Electronic Engineering, 2022, 23(5): 661−666. [doi: 10.1631/fitee.2230000]

[18] Peters F, Tun TT, Yu Y, Nuseibeh B. Text filtering and ranking for security bug report prediction. IEEE Trans. on Software Engineering, 2017, 45(6): 615−631. [doi: 10.1109/TSE.2017.2787653]

[19] Jiang SJ, Zhang X, Wang RC, Huang Y, Zhang YM, Xue M. Fault localization approach based on path analysis and information entropy. Ruan Jian Xue Bao/Journal of Software, 2021, 32(7): 2166−2182 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/6262.htm [doi: 10.13328/j.cnki.jos.006262]

[20] Li Z, Cui ZQ, Chen X, Wang RC, Liu JB, Zheng LW. Deep-SBFL: Spectrum-based fault localization approach for deep neural networks. Ruan Jian Xue Bao/Journal of Software, 2023, 34(5): 2231−2250 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/6403.htm [doi: 10.13328/j.cnki.jos.006403]

[21] Saidani I, Ouni A, Ahasanuzzaman M, Hassan S, MKaouer MW, Hassan AE. Tracking bad updates in mobile apps: A search-based approach. Empirical Software Engineering, 2022, 27(4): 81. [doi: 10.1007/s10664-022-10125-6]

[22] Zhou W, Wang Y, Gao CY, Yang F. Emerging topic identification from app reviews via adaptive online biterm topic modeling. Frontiers of Information Technology & Electronic Engineering, 2022, 23(5): 678−692. [doi: 10.1631/fitee.2100465]

[23] Zhao Y, Yu TT, Su T, Liu Y, Zheng W, Zhang JZ, Halfond WGJ. ReCDroid: Automatically reproducing Android application crashes from bug reports. In: Proc. of the 41st IEEE/ACM Int'l Conf. on Software Engineering (ICSE). Montreal, 2019. 128−139. [doi: 10.1109/ICSE.2019.00030]

[24] Fazzini M, Moran K, Bernal-Cardenas C, Wendland T, Orso A, Poshyvanyk D. Enhancing mobile app bug reporting via real-time understanding of reproduction steps. IEEE Trans. on Software Engineering, 2022, 49(3): 1246−1272. [doi: 10.1109/tse.2022.3174028]

[25] Zhang ZX, Winn R, Zhao Y, Yu TT, Halfond WGJ. Automatically reproducing Android bug reports using natural language processing and reinforcement learning. In: Proc. of the 32nd ACM SIGSOFT Int'l Symp. on Software Testing and Analysis. 2023. 411−422. [doi: 10.1145/3597926.3598066]

[26] Li Z, Jing XY, Zhu X, Zhang HY, Xu BW, Ying S. Heterogeneous defect prediction with two-stage ensemble learning. Automated Software Engineering, 2019, 26: 599−651. [doi: 10.1007/s10515-019-00259-1]

[27] Xu Z, Ye S, Zhang T, Xia Z, Pang S, Wang Y, Tang YT. MVSE: Effort-aware heterogeneous defect prediction via multiple-view spectral embedding. In: Proc. of the 19th IEEE Int'l Conf. on Software Quality, Reliability and Security (QRS). IEEE, 2019. 10−17. [doi: 10.1109/QRS.2019.00015]

[28] Chen J, Wang X, Cai S, Xu JP, Chen JY, Chen HB. A software defect prediction method with metric compensation based on feature selection and transfer learning. Frontiers of Information Technology & Electronic Engineering, 2022, 23(5): 715−731. [doi: 10.1631/fitee.2100468]

[29] Li WW, Chen X, Zhang HW, Huang ZQ, Jia XY. Heterogeneous defect prediction based on simultaneous semantic alignment. Ruan Jian Xue Bao/Journal of Software, 2023, 34(6): 2669−2689 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/6495.htm [doi: 10.13328/j.cnki.jos.006495]

[30] Bahdanau D, Cho K, Bengio Y. Neural machine translation by jointly learning to align and translate. arXiv:1409.0473, 2014.

[31] Zhang Y, Wallace B. A sensitivity analysis of (and practitioners' guide to) convolutional neural networks for sentence classification. arXiv:1510.03820, 2015.

[32] Liu P, Qiu X, Huang X. Recurrent neural network for text classification with multi-task learning. arXiv:1605.05101, 2016.

[33] Greff K, Srivastava RK, Koutník J, Steunebrink BR, Schmidhuber J. LSTM: A search space odyssey. IEEE Trans. on Neural Networks and Learning Systems, 2016, 28(10): 2222−2232. [doi: 10.1109/tnnls.2016.2582924]

[34] Mani S, Sankaran A, Aralikatte R. Deeptriage: Exploring the effectiveness of deep learning for bug triaging. In: Proc. of the ACM India Joint Int'l Conf. on Data Science and Management of Data. 2019. 171−179. [doi: 10.1145/3297001.3297023]

[35] Shi C, Li YT, Zhang JW, Sun YZ, Philip SY. A survey of heterogeneous information network analysis. IEEE Trans. on Knowledge and Data Engineering, 2016, 29(1): 17−37. [doi: 10.1109/TKDE.2016.2598561]

[36] Wang X, Ji H, Shi C, Wang B, Ye YF, Cui P, Yu PS. Heterogeneous graph attention network. In: Proc. of the World Wide Web Conf. 2019. 2022−2032.

[37] Wang X, Liu N, Han H, Shi C. Self-supervised heterogeneous graph neural network with co-contrastive learning. In: Proc. of the 27th ACM SIGKDD Conf. on Knowledge Discovery & Data Mining. 2021. 1726−1736. [doi: 10.1145/3447548.3467415]

[38] Shi MH, Shen DR, Cou Y, Nie TZ, Yu G. Next point-of-interest recommendation approach with global and local feature fusion. Ruan Jian Xue Bao/Journal of Software, 2023, 34(2): 786−801 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/6712.htm [doi: 10.13328/j.cnki.jos.006712]

[39] Hu TY, Jiang Y. Mining of user's comments reflecting usage feedback for APP software. Ruan Jian Xue Bao/Journal of Software, 2019, 30(10): 3168−3185 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5794.htm [doi: 10.13328/j.cnki.jos.005794]

[40] Wang L, Li QS, Lü WQ, Zhang H, Li H. Self-adaptation analysis method for recognition quality assurance using event relationships. Ruan Jian Xue Bao/Journal of Software, 2021, 32(7): 1978−1998 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/6268.htm [doi: 10.13328/j.cnki.jos.006268]

[41] Dong Y, Chawla NV, Swami A. metapath2vec: Scalable representation learning for heterogeneous networks. In: Proc. of the 23rd ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining. 2017. 135−144. [doi: 10.1145/3097983.3098036]

[42] Song Y, Shi S, Li J, Zhang HS. Directional skip-gram: Explicitly distinguishing left and right context for word embeddings. In: Proc. of the 2018 Conf. of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Vol.2 (Short Papers). 2018. 175−180. [doi: 10.18653/v1/N18-2028]

[43] Guo H, Singh MP. Caspar: Extracting and synthesizing user stories of problems from app reviews. In: Proc. of the 42nd ACM/IEEE Int'l Conf. on Software Engineering. 2020. 628−640. [doi: 10.1145/3377811.3380924]

附中文参考文献:

[1] 钱宇, 曹恩叶, 邓文君, 袁华. 海量用户评论在 APP 更新设计中的参与作用挖掘. 系统工程理论与实践, 2021, 41(3): 554−564. [doi: 10.12011/SETP2019-1136]

[2] 肖建茂, 陈世展, 冯志勇, 刘朋立, 薛霄. 一种基于用户评论自动分析的 APP 维护和演化方法. 计算机学报, 2020, 43(11): 2184−2202. [doi: 10.11897/SP.J.1016.2020.02184]

[3] 王亚文, 王俊杰, 石琳, 王青. 一种语义感知的细粒度App评论缺陷挖掘方法. 软件学报, 2023, 34(4): 1613−1629. http://www.jos.org.cn/1000-9825/6697.htm [doi: 10.13328/j.cnki.jos.006697]

[4] 段文静, 姜瑛. 基于用户反馈的 APP 软件缺陷识别. 计算机科学, 2020, 47(6): 44−50. [doi: 10.11896/jsjkx.191100133]

[5] 周慧聪, 郭肇强, 梅元清, 李言辉, 陈林, 周毓明. 版本失配和数据泄露对基于缺陷报告的缺陷定位模型的影响. 软件学报, 2023, 34(5): 2196−2217. http://www.jos.org.cn/1000-9825/6401.htm [doi: 10.13328/j.cnki.jos.006401]

[9] 钟仁毅, 王翀, 梁鹏, 罗忠. 基于版本更新日志的移动应用演化趋势自动分析. 计算机研究与发展, 2021, 58(4): 763−776. [doi: 10.7544/issn1000-1239.2021.20200756]

[13] 郑炜, 陈军正, 吴潇雪, 陈翔, 夏鑫. 基于深度学习的安全缺陷报告预测方法实证研究. 软件学报, 2020, 31(5): 1294−1313. http://www.jos.org.cn/1000-9825/5954.htm [doi: 10.13328/j.cnki.jos.005954]

[14] 蒋远, 牟辰光, 苏小红, 王甜甜. 噪音过滤和深度学习相结合的安全缺陷报告识别. 计算机学报, 2022, 45(8): 1794−1813. [doi: 10.11897/SP.J.1016.2022.01794]

[19] 姜淑娟, 张旭, 王荣存, 黄颖, 张艳梅, 薛猛. 基于路径分析和信息熵的错误定位方法. 软件学报, 2021, 32(7): 2166−2182. http://www.jos.org.cn/1000-9825/6262.htm [doi: 10.13328/j.cnki.jos.006262]

[20] 李铮, 崔展齐, 陈翔, 王荣存, 刘建宾, 郑丽伟. Deep-SBFL: 基于频谱的深度神经网络缺陷定位方法. 软件学报, 2023, 34(5): 2231−2250. http://www.jos.org.cn/1000-9825/6403.htm [doi: 10.13328/j.cnki.jos.006403]

[29] 李伟漳, 陈翔, 张恒伟, 黄志球, 贾修一. 一种基于同步语义对齐的异构缺陷预测方法. 软件学报, 2023, 34(6): 2669−2689. http://www.jos.org.cn/1000-9825/6495.htm [doi: 10.13328/j.cnki.jos.006495]

[38] 石美惠, 申德荣, 寇月, 聂铁铮, 于戈. 融合全局和局部特征的下一个兴趣点推荐方法. 软件学报, 2023, 34(2): 786−801. http://www.jos.org.cn/1000-9825/6712.htm [doi: 10.13328/j.cnki.jos.006712]

[39] 胡甜媛, 姜瑛. 体现使用反馈的APP软件用户评论挖掘. 软件学报, 2019, 30(10): 3168−3185. http://www.jos.org.cn/1000-9825/5794.htm [doi: 10.13328/j.cnki.jos.005794]

[40] 王璐, 李青山, 吕文琪, 张河, 李昊. 基于事件关系保障识别质量的自适应分析方法. 软件学报, 2021, 32(7): 1978−1998. http://www.jos.org.cn/1000-9825/6268.htm [doi: 10.13328/j.cnki.jos.006268]

刘海毅(1997－), 男, 博士生, CCF 学生会员, 主要研究领域为智能软件工程, 软件质量保障与测试.

赵泽江(1998－), 女, 硕士生, CCF 学生会员, 主要研究领域为智能软件工程, 软件质量保障与测试.

姜瑛(1974－), 女, 博士, 教授, 博士生导师, CCF 杰出会员, 主要研究领域为软件质量保证与测试, 云计算, 大数据分析, 智能软件工程.