软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2020,31(1):20−46 [doi: 10.13328/j.cnki.jos.005887]

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

缺陷理解研究:现状、问题与发展*

李晓卓[1,2], 贺也平[1,2,3], 马恒太[1]

[1](中国科学院 软件研究所 基础软件国家工程研究中心,北京 100190)

[2](中国科学院大学,北京 100049)

[3](计算机科学国家重点实验室(中国科学院 软件研究所),北京 100190)

通讯作者: 李晓卓, E-mail: xiaozhuo@nfs.iscas.ac.cn; 贺也平, E-mail: yeping@iscas.ac.cn

摘 要: 缺陷理解是针对缺陷本身及衍生产物建立完整认知过程的研究.由于调试的连贯性及完美缺陷理解假设不合理性,深入分析缺陷传播过程及缺陷间关系、研究关键信息检测及理解信息表达方式,提取及表达面向缺陷研究不同场景不同需求下的可理解性信息,最终达到优化缺陷分析结果及辅助调试各过程缺陷知识复用及积累的目的成为必然.由定位与修复之间存在的知识割裂问题出发,思考缺陷理解研究的本质,明确缺陷处理过程中缺乏知识挖掘及互用问题.通过工程实例分析及文献成果梳理,提炼总结缺陷理解领域的研究方向及技术方法,探讨缺陷理解研究中的特点及难点,思考缺陷理解研究中存在的问题及未来的研究方向,对缺陷理解的研究趋势进行了展望.

关键词: 软件调试;程序分析;软件缺陷;缺陷理解

中图法分类号: TP311

中文引用格式: 李晓卓,贺也平,马恒太.缺陷理解研究:现状、问题与发展.软件学报,2020,31(1):20–46. http://www.jos.org.cn/1000-9825/5887.htm

英文引用格式: Li XZ, He YP, Ma HT. Defect comprehension research: Present, problem and prospect. Ruan Jian Xue Bao/Journal of Software, 2020,31(1):20–46 (in Chinese). http://www.jos.org.cn/1000-9825/5887.htm

Defect Comprehension Research: Present, Problem and Prospect

LI Xiao-Zhuo[1,2], HE Ye-Ping[1,2,3], MA Heng-Tai[1]

[1](National Engineering Research Center of Fundamental Software, Institute of Software, Chinese Academy of Science, Beijing 100190, China)

[2](University of Chinese Academy of Sciences, Beijing 100049, China)

[3](State Key Laboratory of Computer Science (Institute of Software, Chinese Academy of Science), Beijing 100190, China)

Abstract: Defect comprehension is the study of establishing a complete cognitive process for the defect itself and its derivatives. Because of the coherence of debugging and the incomprehensibility of information under the assumption of perfect bug understanding, through analysis of defect propagation process and the relationship between defects, research on key information detection and understanding information expression, extraction, and expression of understandable information for different scenarios and different needs of defect research, ultimately achieving the goal of optimizing defect analysis results and assisting the reuse and accumulation of defect knowledge in debugging process becomes inevitable. This study starts from the problem of knowledge fragmentation between location and repair, considers the essence of defect understanding research, clarifies the lack of knowledge mining and interoperability in the process of defect handling. Through the summary of engineering examples and the analysis of literature results, the research directions and technical methods in the field of defect comprehension are summarized, the characteristics and difficulties in the study of defect

* 基金项目: 核高基国家科技重大专项(2014ZX01029101-002); 中国科学院战略性先导科技专项(XDA-Y01-01)

Foundation item: CHB National Science and Technology Major Project of China (2014ZX01029101-002); Strategic Priority Research Program of Chinese Academy of Sciences (XDA-Y01-01)

收稿时间: 2019-03-20; 修改时间: 2019-05-02, 2019-07-21; 采用时间: 2019-08-22; jos 在线出版时间: 2019-11-06

CNKI 网络优先出版: 2019-11-06 11:49:22, http://kns.cnki.net/kcms/detail/11.2560.TP.20191106.1148.010.html

comprehension are discussed, the existing problems in the study and the future research directions are considered, and the research trend of defect comprehension is prospected.

Key words: software debugging; program analysis; software bug; defect comprehension

缺陷研究一直是程序分析领域的热点问题,软件调试是缺陷生命周期分析过程中重要的研究方向[1],涵盖了缺陷定位、缺陷修复及补丁验证过程[2].各过程间具有紧密的联系,研究各过程间相互辅助性关系,是缺陷分析领域所面临的重要问题.其中,缺陷定位与缺陷修复间的知识割裂关系最为显著且最易被忽视.因此,本文主要讨论调试过程中缺陷定位与修复间的缺陷理解过程.

缺陷定位技术兴起于 2002 年[3],多数缺陷定位研究基于完美缺陷理解(perfect bug understanding)[4]假设,即:假设程序员具有针对该系统充分的背景知识,对于系统架构、文件关联、源代码编写习惯具有较强的认识,将可能引发缺陷的单条语句作为缺陷定位结果,缺乏位置与缺陷之间的因果关系解释,面对关联关系、数据处理或传播过程较为复杂的缺陷难以达到辅助缺陷修复的目的.同时,缺陷自动修复研究将明确的缺陷位置作为输入,反向促进了缺陷定位技术向细粒度方向进行深入研究,忽略了调试过程中的不确定性以及复杂缺陷的研究问题,导致修复人员普遍对定位结果不信任[5,6].该结论被多篇文献[7−13]经过修复对比实验证实:缺陷定位技术难以对应人工缺陷修复的要求.同时,在保证精度的前提下,对于定位粒度的追求导致定位精度逐渐趋于瓶颈,在保持适用面的前提下,方法精度难以逾越上限,导致粒度无法满足需求并且精度达到饱和,难以满足自动缺陷修复的要求.另一方面,目前缺乏细粒度缺陷修复知识在定位研究中的应用,基于历史数据的缺陷定位研究中多数研究针对语义相似度进行分析[14−16],由于真实项目中候选集爆炸问题,难以找到合适的入手点实现基于细粒度修复信息的定位研究.因此,缺陷定位与缺陷修复间存在结果难以互用且知识难以继承的鸿沟.如何弥补调试过程中缺陷定位与缺陷修复研究之间的断层,是本文关注研究问题的大背景.下面以一个实例具体说明.

以图 1 所示的 WINE 兼容层中导致循环调用代码段为例,执行路径在 *CreteEditLabelT*(Label 控件创建函数)、*SendMessageW*(Windows 消息发送函数)、*call_window_proc*(Windows 消息发送函数)、*LISTVIEW_EditLabelT*(Label 控件编辑函数)中存在迭代调用循环,触发了同一控件不断闪烁的缺陷现象.利用基于完美缺陷理解假设下的频谱定位工具定位根因位置,根据可疑度计算原理,缺陷定位在 *LISTVIEW_EditLabelT* 函数中的路径分支谓词 *notify_dispinfoT*.但该函数负责判断是否接收到 *LVN_BEGINLABELEDITW* 编辑 Label 控件消息,该程序段对于调用 *CreateEditLabelT* 函数产生 Label 控件后仍然发送编辑消息的控件进行状态判断,并发送 *WM_CLOSE* 消息对控件进行关闭,缺陷的根因并不在谓词判断函数.单纯基于可疑度排名的单条语句定位结果既没有准确定位根因位置且不能解释缺陷位置与现象间的关系,程序员需要分析谓词上下文以及谓词函数自身逻辑明确缺陷来源,定位结果对修复工作起不到应有的帮助作用.

```
1   infoPtr→hwndEdit=CreateEditLabelT(infoPtr,dispInfo.item.pszText,isW);
2   if (!infoPtr→hwndEdit)
3       return 0;
4   if (notify_dispinfoT(infoPtr,LVN_BEGINLABELEDITW&dispInfo,isW))
5   {
6       if (!IsWindow(hwndSelf))
7           return 0;
8       SendMessageW(infoPtr→hwndEdit,WM_CLOSE,0,0);
9       infoPtr→hwndEdit=0;
10      return 0;
11  }
```

Fig.1 Code segment that generate circular calls

图 1 产生循环调用代码段

针对该缺陷问题,缺陷理解研究在缺陷特征分析的基础上构造修复建议.

- 一是通过缺陷特征分析辅助缺陷定位研究给出位置与缺陷现象之间的因果对应链,其中涉及到关键变量 *ret*(函数 *notify_dispinfoT* 返回值).*ret* 经过消息处理函数发生了周期性的变化,通过 *ret* 前向切片获

得影响变量的语句,结合另一测试用例包含 *LISTVIEW_EditLabelT* 函数的执行路径,即触发 FlashFTP 软件中无法进行右键重命名功能测试用例,将两者信息进行整合,提取路径关键消息处理函数 *Listview_WindowProc*,该函数调用 *LISTVIEW_GetnextItem* 函数及 *LISTVIEW_EraseBkgnd* 函数,完成 *LVN_BEGINLABELEDITW* 及 *LVN_ENDLABELEDITW* 消息成对分发,实现重命名功能.循环中导致 *ret* 变化的过程是该缺陷的因果对应链,通过定位结果对于关键变量进行前向追溯,是明确该缺陷产生原因的关键;

- 二是给出解决缺陷的修复依据.循环开始点与结束点之间的链接是辅助修复的重要信息,*notify_hdr* 函数对于 *ret* 变量的处理方式决定了路径分支的产生,修复位置定位在该函数针对 *ret* 变量处理语句.

该分析过程如图 2 所示,利用缺陷理解技术解决了缺陷定位结果难以应用的问题,弥补了定位与修复之间的鸿沟,缺陷理解研究搭建了缺陷定位与修复间知识互用的桥梁,对于人工调试过程中缺陷知识的复用、自动修复过程中补丁生成的可用性及调试过程的连贯性都起着十分重要的作用.

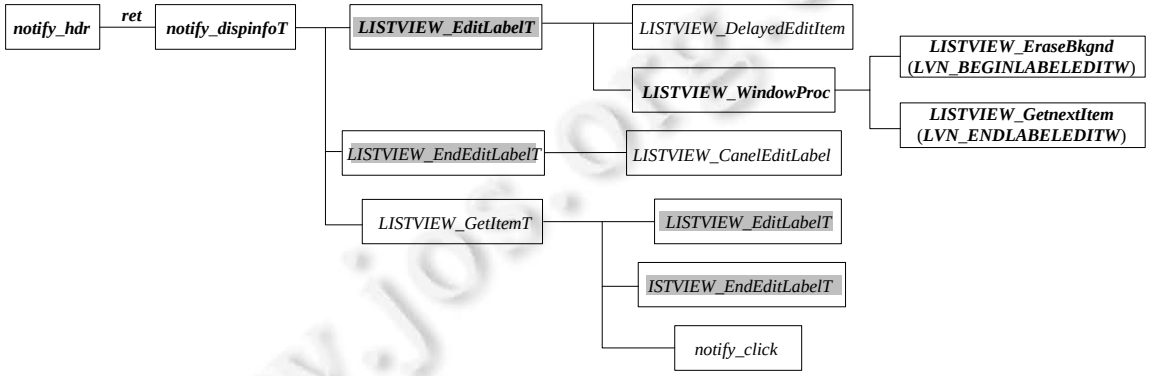


Fig.2 Key function call diagram for defect analysis

图 2 缺陷分析关键函数调用关系图

以上述例子中存在的问题为分析线索,通过工程实践与文献梳理,发现缺陷研究的目的不尽相同,但部分研究实质存在共性,利用或发现同一类缺陷特征属性实现缺陷生命周期中不同的研究工作,研究方法本质存在一定的相似性.因此,实现缺陷生命周期各过程间知识的互用、继承及累积,将极大地推进缺陷处理进程,缺陷理解研究正是贯穿在缺陷处理各过程间的桥梁技术,旨在研究缺陷本质特性,分析缺陷在不同场景不同环境下的独特性及不同研究阶段处理方法的共通性.

因此,本文深入分析缺陷理解问题,旨在为读者提供一种新的思路与角度看待缺陷研究,对缺陷分析这一领域产生新的思考.本文由狭义及广义角度解释缺陷理解问题,以近 10 年的文献分析展示缺陷理解问题的研究脉络,综合文献研究成果及应用实例分析梳理缺陷理解研究方向及方法,根据目前研究中存在的问题及缺陷研究领域发展趋势探讨未来可能的发展方向.

1 缺陷理解问题的界定与分析

本文由缺陷定位与缺陷修复研究之间的割裂关系出发,对缺陷研究领域工作进行全面分析,发现现有方法或局限于对缺陷语言(自然语言及代码语言)相似性的分析与应用,或偏重于完美缺陷理解假设下的缺陷分析.对完美缺陷理解假设不成立情况下软件缺陷的分析不够深入,缺少能够真正刻画软件缺陷本质属性特征的研究.在现有文献研究及实践工程问题中,缺陷理解研究绝大部分集中于缺陷定位与缺陷修复过程之间,少量研究涉及缺陷理解辅助缺陷发现、分类、预测及验证等工作[17,18].通过分析具体实例及研究成果,本文认为有以下 3 点原因:首先,缺陷理解是近年产生在软件调试领域的新概念[3,5],由调试过程中对 Top-*n* 形式可用性的质疑,研究方向产生了由给出缺陷可疑程序语句(或文件)到解释软件失效原因的转变;第二,软件调试过程对缺陷的解决

起到关键作用,问题性质涉及到分析缺陷共通性及差异性,例如,缺陷修复研究需要结合修复模式及环境特殊性分析;第三,软件调试过程具有明确的评价标准,例如,缺陷修复将测试用例通过且不影响其他已有测试用例运行结果作为修复的评价标准.但缺陷分类、预测及复现等领域评价准则较软件调试研究具有一定的模糊性,例如:缺陷分类研究分类标准本身具有差异性,缺陷检测中模糊测试领域多数研究将系统崩溃作为检测到缺陷的评价标准.尽管目前缺陷理解研究集中在软件调试过程中,但缺陷生命周期分析其他过程中研究数量也逐渐增多,研究人员逐渐认识到缺陷理解研究在缺陷分析领域的重要性及关键性.因此,本文不仅梳理了缺陷定位与修复之间的狭义缺陷理解研究,也分析了缺陷生命周期更广义角度的缺陷理解问题.

1.1 从狭义角度看缺陷理解问题

大量质疑完美缺陷理解假设下定位结果可用性文章的出现[7−13],引发了对调试过程间关系及缺陷知识研究的关注.由前面例子分析的知识割裂问题可见:缺陷定位与缺陷修复之间不存在一条直接可达的路径,调试过程中对缺陷的处理需要利用缺陷理解研究关联缺陷定位及缺陷修复过程,并在此基础上弥补两者之间结果难以应用的鸿沟.缺陷定位与缺陷修复之间经过缺陷特征分析过程,由定位技术结合信息挖掘方法构造修复建议完成缺陷修复.该过程需要解释的不仅是“什么(哪里)导致了程序的失效”,更需要解释的是“为什么程序会失效”.即:解释缺陷位置与程序失效之间的关联关系,分析缺陷现象与缺陷位置之间的因果关系,给出一种结合源代码及缺陷产物信息的缺陷表达形式,结合定位位置与理解信息加快调试进程,推进缺陷的解决.

结合图 1 所示缺陷分析实例,解释缺陷理解与缺陷定位及缺陷修复研究间的关系.

缺陷理解研究通过缺陷特征分析辅助缺陷定位研究.对于实例中 Label 控件循环闪烁现象,利用依赖关系、程序规约等方式查找原因链,在定位缺陷根因位置的基础上解释缺陷来源,一定程度上解决了缺陷定位领域精度及粒度研究的瓶颈问题[3,19].通过分析相似测试用例原因链重叠谓词 *notify_dispinfoT*,判断多缺陷等复杂缺陷存在的可能性,在定位的基础上更加全面地分析缺陷可能的衍生关系.该缺陷特征分析过程涉及到缺陷传播过程及缺陷间关系研究.由于调试的方向性,缺陷理解是依托缺陷定位过程进行研究,改善非完美缺陷理解下缺陷信息的完备性及可用性,明确不同场景及缺陷特征情况下研究的差异性,解释与缺陷位置相关联的问题,例如现象与位置间的关系、不同缺陷位置间的关系、位置与来源间的传播过程等.而传统缺陷定位在完美缺陷理解假设限制下,仅仅针对定位粒度及精度问题进行研究,将精准性作为度量定位过程分析结果的唯一方式,忽略了其可用性.而真实调试过程由于对象(具有构造策略差异性的工具或具有知识结构差异性的程序员)、应用目的(后续工作需求)、场景等因素使得该过程输出结果具有差异性,结果度量及信息构造方式具有多样性.缺陷理解研究不单独存在(理解需要具有背景及目的性),两者在研究内容上存在一定的相关性,但缺陷理解研究从研究背景、对象等多维度的角度思考缺陷定位在不同目的下研究的差异性,对缺陷定位研究起到支撑作用,关注该过程下信息的可解释性及可用性.

缺陷理解研究通过修复建议构造推进缺陷修复进程,并逆向推动细粒度缺陷修复信息在缺陷定位中的应用.对于实例中多函数循环调用问题,利用变量 *ret* 分析根因位置与关键谓词间的关系,根据根因位置对 *ret* 变量的处理方式,提出改动影响谓词赋值语句达到修复缺陷的目的,该过程涉及到缺陷关键信息的获取以及缺陷理解信息表达方式的研究.缺陷的独特性导致了缺陷修复的困难性,目前的缺陷修复研究忽略了缺陷的环境及场景,多数研究缺乏对缺陷的语义分析,利用遗传变异[20]、模式匹配[21]等方式构造补丁,导致结果的可用性较低.修复研究利用精确的缺陷位置作为输入,忽略了修复与调试过程间的关系,造成了目前缺陷修复技术难以应用在实际调试过程中的困境.不同场景下的缺陷特征差异性导致了缺陷修复的复杂性,缺陷理解研究依据特征分析构造修复建议,解决了信息不可用及准确率低的问题,对缺陷修复研究起到了辅助作用.并且,缺陷理解通过准确抓取缺陷特征,有效解决了基于历史信息缺陷分析中候选集数量爆炸问题,提高了缺陷分析的效率,即解决了缺陷生命周期研究各过程间知识复用的问题.

缺陷理解对缺陷定位与缺陷修复研究起到辅助支撑作用,对于两者间的过程研究起到桥梁作用.缺陷理解研究延伸应用到缺陷生命周期过程中的各个领域,本质上是解决缺陷知识的继承、复用与积累问题.

1.2 从广义角度看缺陷理解问题

从缺陷生命周期过程来看,缺陷理解具有更广的意义.理解是一种受人为因素影响较大的研究,表明主体利用已有知识背景及领域知识针对不同对象进行完善认知的过程.缺陷(bug,defect)是程序表达信息与软件需求之间的矛盾体现,区别于错误、失败与故障(硬件及电力电子领域)[22].

- 错误(error):是指系统中可能导致缺陷的一种情况[5],编码错误是导致软件缺陷产生的主要原因[23];
- 失败(failure):是指系统服务偏离其正确行为[5];
- 故障(fault):Wikipedia 中说明,故障(fault)一词常应用在硬件方面,是指硬件组件、设备或子系统产生异常情况(https://en.wikipedia.org/wiki/Fault).

缺陷理解结合“缺陷”及“理解”的定义,是一种将程序实现与用户需求冲突作为分析对象,针对缺陷本身及衍生产物建立完整认知,贯连缺陷检测、调试及验证过程的研究,是一种由表及里的缺陷信息挖掘、整合及学习的活动,涵盖了缺陷显性的可观测行为及隐性的内部信息,实现缺陷研究及缺陷间分析过程知识的积累与复用,提供了辅助缺陷知识库建立及应用,进而有效指导缺陷分析等工作开展的更为本质的研究思路.

缺陷理解问题与程序理解不同,程序理解关注软件性质,主要涉及的是软件及其使用和变更的活动[24],研究的目的是辅助程序员深入理解软件制品的含义,包括它所表达的软件功能及实现方式等,关注的重点是软件整体的行为及变更影响[24].首先,缺陷理解关注的对象与程序理解不同,关注缺陷代码及产物,对象更加明确,面向缺陷修复或缺陷检测等缺陷分析行为进行理解信息的提取,关注异常信息与产物关联关系,虽然部分研究方法与程序理解研究方法存在重叠,但程序理解现有方法很难达到缺陷理解所要求的程度,对于程序的异常没有进行深入的研究.其次,缺陷理解较程序理解具有更强的复杂性及目的性,缺陷是程序的副产物,缺陷的存在形态多样且触发环境复杂,具有很强的复杂性及隐蔽性,检测缺陷并排除缺陷是缺陷理解的最终目的.缺陷理解研究涉及到缺陷生命周期的多个过程,如缺陷的触发、传播、根因定位以及缺陷现象分析等.

缺陷理解与软件调试研究不同,软件调试是将异常行为转变为符合规约行为的过程,包括缺陷定位与缺陷修复等多个部分[25],调试是发现缺陷后解决缺陷过程的统称,主要辅助调试器(如 windbg 工具、SoftICE 工具)对缺陷的问题点实现快速发现[25,26].缺陷理解涉及到缺陷研究的各个过程,主要涉及到缺陷特征的提取以及关联产物的识别与信息的整合分析,理解作为心理过程不单独存在,缺陷理解存在于不同的软件缺陷生命周期中,目的是挖掘缺陷在不同场景、环境及需求下的关键信息,并发现缺陷生命周期各过程研究中本质的相似性.

从与已有方向的结合性角度来说,除了上文所述缺陷定位及修复研究外,缺陷理解还可以与缺陷检测、缺陷分类、缺陷预测及缺陷复现等缺陷生命周期各研究过程相结合.对于缺陷检测,目前大量的研究方法利用模糊测试(fuzz testing)[27,28]及污点分析(taint analysis)[29]技术对复杂缺陷进行检测,但仍缺少在有限时间内对大量二进制程序进行检测的有效方法[30].缺陷理解对于缺陷特征的分析以及产物信息的研究明确了缺陷检测过程中缺陷位置与测试用例间的关联关系,从本质上提高了缺陷检测的精确度.对于缺陷复现,由于闪退、崩溃及宕机等不具备特殊性及标志性的缺陷现象,测试人员难以给出有效的缺陷描述[31],导致该类缺陷难以复现进而导致软件调试难以进行.缺陷理解对复杂缺陷的分析从根本上提取缺陷的根因特征,由产生缺陷的本质问题出发,从语义层面上分析缺陷传播过程,弥补了缺陷复现及测试用例生成研究中自然语言相似性匹配导致的低精确性问题.缺陷的研究目的不同、实现手段不同,但缺陷理解分析缺陷特征、研究缺陷在不同场景下的共性及独特性以及缺陷的研究由于缺陷自身及环境的复杂性导致了不可一概而论的研究方式,研究方法可能与缺陷分析领域存在相似性,但缺陷理解以一种新的思路与角度分析缺陷问题,积累缺陷知识,是一项剖析缺陷分析本源的研究工作.

从研究对象及研究内容的角度来说,缺陷理解是对缺陷进行深度及广度剖析的过程,一种从多对象中获取知识的研究.缺陷通常被看作用程序语言表述的一种与预期需求产生冲突的对象,缺陷理解可以抽象地认为是一种从缺陷及产物中获取表达知识建立认知的过程,涉及到不同对象的整合,不仅对程序语义进行分析,同时对自然语言描述与程序表达内容进行融合.缺陷理解的关键不在于绝对的正确与错误之分,而是将违背程序预期状态的行为作为研究对象,实现缺陷本质分析研究,涉及到软件需求、测试过程、缺陷触发、修复效果、缺陷

与正确代码之间的关联性等相关问题,并且缺陷理解的重点不再是程序的表达信息,而是缺陷表达信息与预期产生效果的冲突分析.

从研究方法及实现技术的角度来说,缺陷理解以程序分析及自然语言处理技术作为基础,结合缺陷研究各过程不同产物及研究方法辅助缺陷分析.绝大多数缺陷理解技术集中于解决定位与修复间的知识割裂问题,明确定位结果的同时,需要对不同的定位技术进行研究,将成熟定位方法及关联规则(association rule)[32]、程序依赖(program dependency)[33]、代码逻辑分析(code logic analysis)[34,35]等研究技术作为基础,利用定位技术根因定位结果作为输入,构建缺陷理解技术的研究方法.另一方面,结合已有的缺陷修复技术及缺陷类型分析,利用特征挖掘(feature mining)[36,37]、缺陷签名(bug signature)[38,39]等研究获取更精细的缺陷理解信息,辅助自动修复工具,利用人工修复过程行为挖掘抓取修复缺陷的必要信息,辅助人工缺陷修复信息的生成.

从研究与工程应用的关系角度来说,工程应用中根据软件形态对缺陷的处理存在多种方式.在开发阶段,缺陷的更改涉及到算法实现思路或方法的改变,从根本上解决缺陷,目的是理解缺陷的来源.在应用阶段,缺陷的处理不涉及到方法层面的改动,尽量利用谓词判断、程序跳转将缺陷触发情况进行隔离处理,是在保证系统完整性的前提下,理解缺陷的传播过程以达到消除缺陷的目的.在维护阶段,缺陷的产生需要进行容错处理,在不影响主要功能的前提下,将缺陷的影响尽可能减小到最小,利用触发条件或环境对缺陷的产生进行控制[40],理解缺陷的共同点以保证系统的强健性与可靠性.

目前,软件系统逐渐趋于复杂,云环境、人机融合应用模式的出现导致了系统复杂性及不可靠性的提高,引发了大量复杂缺陷的产生,例如并发缺陷、多缺陷等.同时,高可信系统的研究对于缺陷的容错程度较低,面对信息产业的发达、用户隐私要求的提高以及关键领域系统的可信度要求,缺陷的处理需要更加快速准确,导致缺陷的研究需要更加深入与完善.了解软件缺陷的本质是软件工程的基础[23],对软件缺陷进行理解信息的提取,缺陷理解信息的可用性及准确性就成为了新的软件形态及软件要求下决定软件调试质量、效率的关键研究问题.

2 缺陷理解研究全景分析

为了对缺陷理解领域进行全面的认识及分析,本文采用自动检索及人工检索结合的方式对公开发表的文献进行广度及精度搜索,从而使论文检索结果达到最优.在 IEEE、Elsevier、ACM、Springer 及 CNKI 数据库中进行论文检索,将缺陷理解作为关键词,缺陷检索词为 bug、defect、fault,理解检索词为 comprehension、understanding,将两组词汇进行交叉组合,对 2008 年 1 月 1 日~2018 年 12 月 31 日间公开发表的文献进行检索.关键词检索共搜索到 316 600 篇文献,该检索方式仅是进行词汇的简单匹配,检索到的论文可能与本文讨论内容无关、质量较差或在不同检索词或数据库中发生重复,因此,对搜索结果进行以下几方面的筛选.

1) 论文题目筛查:通过论文题目分析研究对象为缺陷代码或产物,文章对软件缺陷进行分析研究;
2) 论文来源筛查:论文用英文发表,属于软件工程领域且不是发表于书,不属于灰色文献;
3) 论文长度筛查:论文属于期刊或会议的长文及部分人工筛选认定有价值和前瞻意义的短文;
4) 论文关键词及摘要筛查:论文研究内容属于缺陷理解领域,对缺陷实体、环境或产物进行分析;
5) 对于多次检索中出现的重复论文进行筛除.

利用人工检查方式,依据以上 5 个方面进行筛选,初步查出 177 篇论文.对初步筛查论文进行快速阅读,进一步筛查出属于缺陷理解领域文章 37 篇.第 2 阶段,通过关键论文引用检索在精度上查找缺陷理解研究文献,共查到 43 篇,为确保文章质量和强相关性,浏览文章全文并与合著作者讨论,最终确定 73 篇文章[7−13,17,31,36−39,41−100]作为研究对象,用作后续研究方向及方法分类重点分析对象.

对以上 73 篇文章从发表年代、发表期刊等级及研究主题这 3 个维度进行缺陷理解研究趋势分析,得到如图 3 所示的统计结果.

根据图 3 得出以下结论.

1) 自 2011 年起,研究人员逐渐认识到完美缺陷理解的局限性,对缺陷理解领域进行研究,并且关注度一直上升,在 2013 年逐渐达到研究峰值,并且热度一直持续至今;

2) 缺陷理解文章多数为高质量文章,40%发表在 CCF-A 类会议上,可见该研究领域被专家学者所认同,是目前缺陷研究领域的重要方向;

3) 缺陷执行动态信息分析是缺陷理解研究的重要问题,从 2008 年~2018 年,均有大量论文与缺陷动态执行信息相关;

4) 自 2011 年起,传统定位方法的实际应用性及有用性被大量质疑,导致在实际工程项目中对定位方法的实验验证及评估类文章大量涌现,促进了缺陷理解方向的快速发展.由于自动缺陷修复技术研究领域自 2014 年被逐渐重视,以及 2016 年前后深度学习(deep learning)及信息检索(information retrieval)研究的兴起,2016 年,缺陷理解研究呈明显上升趋势,并保持持续性研究至今.

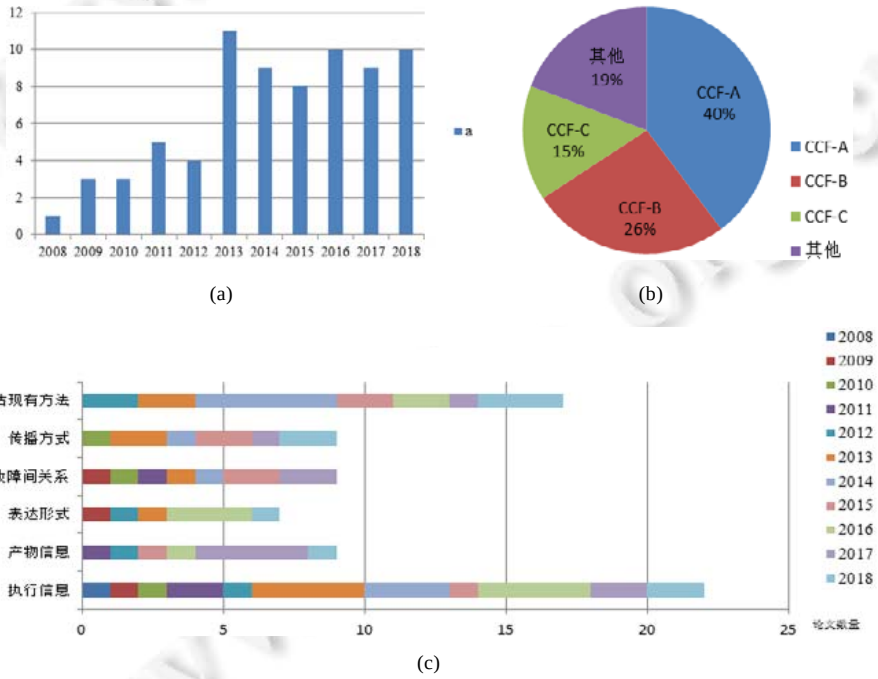
5) 文献研究主题覆盖了第 1 节所述的缺陷理解研究的 4 种基本形式及方向.

a) 对于缺陷关键信息进行检测及分析,包括执行信息及产物信息[31,37,48−50,58,64,67,80].随着深度学习及信息检索研究的兴起,缺陷关键信息的搜索及相似性分析研究数量突增,理解信息的精度及粒度有了很大提高;

b) 对于缺陷传播过程进行分析,例如,涉及缺陷组件可疑度分析[87,88]、缺陷间关系及动态执行信息分析[41,62,90],传统的定位方法忽略了调试过程中的不确定性,缺陷的传播方式分析及动态可疑度计算成为目前的研究热点;

c) 对缺陷间的关系进行研究,例如,缺陷根因变量的依赖关系分析[73]、多缺陷之间的关系研究[76,82]等,研究人员逐渐意识到缺陷产生环境对缺陷研究的重要性[69,85],对多缺陷、顺序执行缺陷等特殊缺陷的分析,一定程度上扩展了传统调试方法的应用范围,不同种类缺陷的特征分析研究得到了逐渐的重视,方法的可用性及适用性范围得到了扩展;

d) 对于缺陷的表达方式进行研究,例如动态传播图表示方法[57,77]、树型执行路径图表示方法[43],由传统的语句可疑度排名表示方法到可理解性模型的出现,对缺陷理解信息的表示形式由语法到语义层面进行了深入的分析研究.



(a) (b)

(c)

Fig.3 Trend map of defect understanding papers

图 3 缺陷理解论文趋势图

本文不仅关注论文来源的 CCF 分类,对于论文发表的会议类型也十分关注.如表 1 所示,缺陷理解论文来自 ICSE、ASE、FSE、ISSTA、ICPC 等会议,其中,ICSE 及 ICPC 论文数量较多.ICSE 及 ICPC 均属于软件工程领域会议,其中,ICPC 是将程序理解研究作为主题的国际会议,收录论文主题多与理解领域相关.

Table 1 Distribution list of research papers publishing conferences/periodicals

表 1 研究论文发表会议/期刊分布列表

类别	CCF 类别	会议/期刊名称缩写	数量	总数量
会议论文	A	{ICSE,ASE,FSE,ISSTA,PLDI}	{11,6,5,4,1}	27
	B	{ICPC,ESEM,其他}	{7,3,6}	16
	C	{MSR,ICST,其他}	{2,2,7}	11
期刊论文	A	{TSE}	{2}	2
	B	{ESE,JSS}	{2,1}	3
	C	无	{0}	0

由此可见:缺陷理解研究越来越得到研究学者的重视,并且研究点趋于广泛,研究内容趋于深入,是目前较为热点且较为关键的研究方向,作为调试过程中亟待解决的研究问题,逐渐得到了广泛的认同及认识.

3 缺陷理解研究方向及方法进展

3.1 缺陷理解研究方向

缺陷理解研究与缺陷特征及产物的信息类型相关,缺陷理解是辅助缺陷生命周期各过程代码理解性信息生成及缺陷产物信息关联过程的研究.因此,缺陷理解对于缺陷的研究集中在缺陷自身性质、存在环境及两者交互问题上,缺陷通常会在一个特定的上下文环境中显露出来,而明确这些上下文对于诊断和修正缺陷是必要的[19].根据以上分析及工程实例研究以及文献成果梳理,缺陷理解涉及缺陷关键信息、传播过程、缺陷间关系及缺陷表示方式研究这 4 个方向.

缺陷关键信息检测及分析,即对于缺陷研究各过程中起到决定作用的内容进行分析(例如执行路径差异点、关键变量前置条件及程序失效特征点集合等),静态分析方法通过结合自然语言与代码语言获取缺陷原因链,动态分析方法利用差异性研究获取执行路径中的关键信息.利用依赖关系及关联规则,分析关键信息与缺陷间的对应关系,构造缺陷理解信息.例如:Zuddas 等人[51]通过可疑度分析选取运行用例的关键路径点,获取关键路径点动态运行数据,比较正反测试用例中执行路径差异,对影响缺陷的关键点进行过滤,辅助缺陷定位技术,给出缺陷解释.

基于缺陷传播过程的研究,即对缺陷的执行过程及缺陷状态传播影响关系进行分析,考虑调试过程中的不确定性,构建缺陷状态传递模型,解释缺陷产生的原因.根据缺陷现象自然语言描述,获取触发缺陷测试用例输入的动态执行路径,分析路径中与缺陷现象相关的函数调用关系,将缺陷自然语言描述与源代码语言相对应,获取缺陷执行链,量化缺陷传播过程.例如:Jiang 和 Su[101]构建了程序缺陷相关的控制流路径解释定位结果,包括特征选择(准确地选择与缺陷相关的谓词)、聚类分析(将相关谓词分组)和控制流图遍历这 3 个阶段,利用控制流信息进行缺陷解释.

基于缺陷间关系的研究,即对缺陷触发环境进行分析,将多缺陷、顺序执行缺陷及并发缺陷等依赖软件环境且在缺陷之间产生关联关系的缺陷作为研究对象,打破了传统单缺陷的研究假设.整合产物信息与源代码信息,完善缺陷根因位置与缺陷现象间的关联关系.对于缺陷存在环境的研究有助于缺陷可追溯性的分析,是分析软件演化与缺陷演化之间关系的研究基础.

缺陷理解信息表示方式研究,即对缺陷理解信息的表达形式及辅助修复的准确性进行分析验证,解决缺陷语言,即代码语言与自然语言之间语义表示形式差异性,以及缺陷独特性及环境多样性导致理解信息表述方式构造的复杂性,提出一种公认的理解信息表示方法,完善理解信息评估框架.在此基础上,衡量表达形式的准确性,根据不同类型的缺陷特征(回归缺陷关注版本间代码差异、顺序执行缺陷关注缺陷触发逻辑)及信息获取方式(最小差异分析关注差异点间的关联、语义研究关注自然语言及代码语言的映射关系)提出对应的理解信息

表达形式.表达形式由缺陷特征、所处环境及关键信息获取方式等多方面因素决定.目前比较被认同的缺陷理解信息表示形式见表 2.较早的研究利用关键变量切片信息[101]、程序依赖图[88]对缺陷理解信息进行表示,该类研究存在信息冗余问题.Hsu 等人[79]利用缺陷签名方法对冗余问题加以解决,但缺陷签名存在测试用例选择导致的误差问题及可解释信息丢失问题.后续的正反测试用例差异化分析[42,43]有效地解决了测试用例选取导致的误差问题,但具有差异点不具关联性、难以理解的问题.Cheng 等人[77]通过判别图挖掘进行缺陷定位和相关上下文的识别,通过比较不同执行中的程序流,从中提取出区别度最大子图,提取的子图不仅定位了缺陷,还提供了用于理解和修复缺陷的上下文.Yi 等人[43]利用树型图解释回归测试场景下出现的缺陷,提出了版本升级过程中改动代码与缺陷现象之间的原因链概念,解释缺陷产生的原因.He 和 Gupta[102]利用基于路径的最弱前置条件自动地产生如何修改一个函数中出错语句的建议,以前置条件和后置条件的形式解释缺陷触发原因.

Table 2 Defect understanding information representation

表 2 缺陷理解信息表示方式

文献	表示形式	程序类型/缺陷类型	被分析程序
[43]	原因链树型图	C/C++ (回归缺陷)	seven widely used Linux applications such as find, bc, make, gawk and diff
[57]	概率争议模型(probabilistic dispute mode,简称 PDM)	C	Siemens benchmark
[77]	执行路径判别子图	C	the seven Siemens datasets
[68,79]	缺陷签名	C	Siemens programs and faults
[88]	概率程序依赖图(probabilistic program dependence graph,简称 PPDG)	C	Siemens suite
[91]	控制流切片信息	C	HR variants of the Siemens test suite, rhythmbox 0.6.4, RandomForests on rhythmbox
[93]	基于前置条件及后置条件的触发约束	C	Sum, Max, Binary Search, Array Copy, Quicksort
[94]	关键谓词状态原因链	C	Siemens test suite

3.2 缺陷理解研究方法及进展

从缺陷关键信息检测及分析的角度,缺陷理解研究是对缺陷特征进行提取及分析,涉及到最小差异性分析及特征提取算法.从缺陷传播过程的角度,缺陷理解是对缺陷的动态执行路径及静态程序关联关系进行可疑性分析,涉及到语句的关联规则构造及可疑度计算方法.从缺陷间关系的角度,缺陷理解是对缺陷产生环境进行分析,涉及到缺陷间的复杂逻辑关系构建及产物间相似性判断方法.从缺陷表达形式的角度,缺陷理解研究是对代码语言及自然语言描述进行分析,处理及融合缺陷及产物信息,完善缺陷知识表达形式,涉及到自然语言处理方法及信息融合技术.缺陷理解研究不应分割地看待缺陷生命周期的各个部分,缺陷理解研究方向与缺陷研究各个过程的关系如图 4 所示.

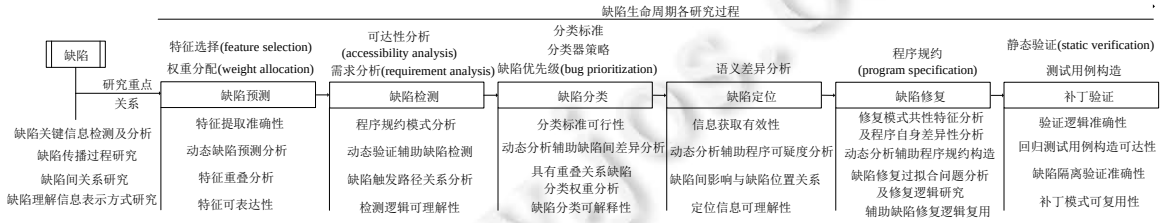


Fig.4 Relationship diagram of defect life research and defect understanding

图 4 缺陷生命周期各过程研究与缺陷理解关系图

缺陷生命周期各过程中缺陷理解研究逐渐被研究人员所重视.基于上文所述 4 个研究方向,下面我们分别梳理缺陷理解研究方法,讨论研究进展.

3.2.1 缺陷关键信息检测及分析

根据缺陷触发难度及获取输入类型不同,存在正反测试用例对比、单条缺陷路径难以理解或难以获得正确

执行路径等情况[103],分别利用缺陷动态执行信息正反测试用例对比分析方法及失败测试用例聚类缺陷特征信息提取方法对缺陷关键信息进行检测及分析.

(1) 正反测试用例对比分析

基于失败测试用例及成功测试用例(成功测试用例(passed test case)是指实际输出与预期输出一致的测试用例,否则称为失败测试用例(failed test case))的动态执行信息差异性分析,是目前缺陷理解较为普遍的研究方法,已有研究针对关键变量依赖关系差异性或正反测试用例执行路径差异性两个方向进行分析.

- 首先,从变量依赖关系出发,利用变量依赖图及变量切片等方式解释缺陷来源.Feng 等人[104]基于程序依赖图(program dependence graph)计算变量状态转换概率,利用预期程序节点状态及差异性分析变量状态演化过程及缺陷来源.该研究存在状态爆炸的隐患,对于依赖图规模较大且预期状态较多的复杂缺陷,无法判断与触发缺陷现象真正相关联的理解信息,结果存在冗余;
- 另外,从执行路径出发,Gao 等人[63]分析频谱缺陷定位技术中缺陷语句作为独立计算单元缺乏可疑度推理,导致缺陷来源不明确问题,利用序列模式挖掘技术获取执行谓词运行变量序列,分析关键谓词序列差异性,在定位的基础上明确缺陷相关上下文信息.但对于控制流复杂缺陷,序列差异性可能涉及与所关注缺陷无关的需求信息,导致结果存在冗余.

当输入类型为正反测试用例对比信息时,为了解决冗余问题,多数研究利用最小差异性分析(minimum difference analysis)[42,43,73,92,95]方法,通过动态执行路径的差异点或差异集合对缺陷进行解释.最小差异性分析基于动态执行,针对相同功能不同输入的测试用例,通过比较语句覆盖程度以及关键分歧点信息差异解释缺陷.由于每条失效执行路径覆盖程序语句的情况存在差异,分析覆盖语句差异化程度,减少冗余差异信息,利用根因定位语句与差异化语句间的可达路径达到解释缺陷的目的.

选取成功及失败的测试用例中相似性最大的执行路径,利用动态执行观察差异化路径分布,即包括公共执行点、测试用例执行路径差异点及关键谓词路径分叉点.S_n 表示程序语句单元,利用钩子函数等方式监控源代码运行路径,记录程序执行轨迹,记为$\{S_1,S_2,\ldots,S_n\}$,分析路径差异化,筛选缺陷对应失效功能差异点,减少冗余信息的过程称为最小差异化分析.

以上分析过程基于两个前提假设.

- 源程序具有同一功能的多输入性,保证了成功测试用例及失败测试用例的可执行性;
- 源程序具有测试用例差异可表示性,存在路径差异或关键点动态执行差异性.

Souza 等人[34]及 Licia 等人[35]提出差异性分析重点为减少信息冗余以及多点差异化数据的关联性分析,差异化数据可读性及关联可解释性严重影响程序员对于缺陷的理解程度,片段式的最小差异代码段缺少关联关系.如图 5 所示,图中,A_n 表示成功测试用例覆盖差异基本代码块,B_n 表示失败测试用例覆盖差异基本代码块,集合$\{A_1,A_2,A_3,B_1,B_2,\ldots,A_n,B_n\}$表示正反用例差异化分析结果,$C_n$ 表示差异代码间关联基本块,D_n 表示产生路径分歧的关键谓词.

差异点间的关系分析是目前最小差异化分析中的关键问题,关联度复杂缺陷需要对相似关联路径进行筛除,比较 C_n 与定位根因结果基本块及对应最小差异集的关联度(考虑距离、调用顺序及次数等特征).利用执行路径所覆盖基本块序列的相似性表示执行路径间的距离,基于距离度量的特征分析方法对关联性进行刻画,减少相似关联路径,增加关键路径分歧基本块 D_n 与缺陷根因位置间涵盖差异点的可达路径,达到信息补全的目的.解决最小差异基本块间的关联性问题,是提高最小差异性分析精确性及可理解性的重要研究方向.

(2) 失败测试用例聚类特征提取

针对单条缺陷路径难以理解或正确执行路径难以获得的情况,利用缺陷签名分析[79]进行缺陷理解信息生成.为了获取缺陷特征,采用缺陷签名方法收集缺陷执行信息,缺陷签名是一组突出显示缺陷原因或影响的程序元素,并提供上下文相关信息以供调试[79,105].缺陷签名应具有以下两个特征.

- 缺陷签名包括缺陷产生的原因及影响;
- 缺陷签名具有简洁的特征,减少冗余信息所带来的误差.

Hsu 等人[79]创造了术语缺陷签名,提出缺陷签名应包含提供缺陷上下文信息的多个元素,通过缺陷签名而不是通过自动缺陷隔离产生的单个可疑元素(语句或谓词)对缺陷进行后续研究,基于 Tarantula 定位技术[106]及序列挖掘算法将一组失败执行中的最长公共序列作为缺陷签名,利用公共执行语句表示缺陷理解信息.该方法对于顺序执行触发缺陷具有较好的效果,但该方法并没有对公共基本块之间的关联性加以表示,可理解性较差.随后,Cheng 等人[39]利用判别图挖掘方法识别缺陷签名,采用执行控制流图的形式挖掘失败执行中的缺陷特征.由于该方法仅考虑了控制流转换缺陷,无法对不产生控制流转换及差异的缺陷进行特征提取,应用面受到限制.为了增强缺陷签名的预测能力,Sun 和 Khoo[38]提出了谓词缺陷签名挖掘方法,利用关键谓词和控制流信息,设计了一种判别项集挖掘技术达到生成简洁谓词缺陷签名的目的.Zuo 等人[107]利用谓词修剪及增强技术减少缺陷签名谓词遍历工作量,利用粗粒度遍历指导细粒度检测,有效地减少了缺陷签名中冗余的理解信息.

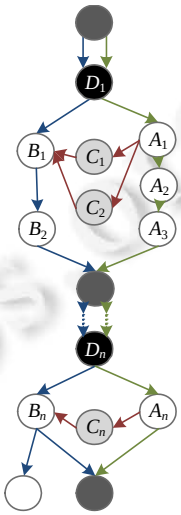


Fig.5 Path minimum difference point association diagram

图 5 路径最小差异点关联关系图

缺陷签名提供缺陷上下文,目的是解决单条语句难以理解或部分具有顺序执行特征的缺陷分析问题.缺陷签名基于根因缺陷定位技术,对不同路径分支进行可疑度计算,以签名的形式捕获缺陷上下文.该签名包含同一功能不同输入的所有错误执行路径分析,执行路径中包含与用户需求相对应的方法条目和分支决策,使用基于频谱的缺陷定位技术计算路径中语句基本块可疑度,通过可疑度进行聚类分析,挑选与缺陷相关的执行路径对其进行解释.

但缺陷关键信息不止存在于错误执行公共序列中,有可能存在于正确执行路径信息中,仅考虑错误执行路径可能导致理解信息的丢失;并且,输入数据选择过程中的模糊性引入无关测试用例执行信息,进而导致缺陷签名信息的冗余;同时,多数缺陷签名研究基于图挖掘方法,并没有考虑调试过程中的不确定性.以上 3 点导致缺陷签名方法生成理解信息准确性及精确性较低.结合输入数据预处理技术及过程可疑度计算方法对缺陷签名技术的准确性加以提高.程序运行路径图是有向图,根据有向图的定义,利用符号执行等工具表示程序执行路径.从缺陷本质出发,对于不同的背景环境,不仅需要考虑基本块(变量或函数)存在缺陷可疑度(节点可疑度),同时需要分析基本块调用关系及执行路径转换中缺陷传播可疑度(边可疑度),利用 Tarantula 等频谱缺陷定位方法计算语句基本块可疑度,并基于 Jaccard 等相似度距离公式计算执行路径覆盖缺陷传播可疑度,结合基本块及路径缺陷传播可疑度分析提高缺陷签名的精确性及准确性,如公式(1)、公式(2)所示.

- 边可疑度

$$susp_{s_{(edg)}} = \frac{n_{ef}(s_{(edg)})}{n_f + n_{ep}(s_{(edg)})} \tag{1}$$

- 节点可疑度

$$susp_s = \frac{n_{ef}(s)/n_f}{n_{ef}(s)/n_f + n_{ep}(s)/n_p} \tag{2}$$

其中,程序语句实体 s 及执行关系传递边 $s_{(edg)}$的交叉表见表 3.

Table 3 Cross table for statement s and execution relation delivery edge $s_{(edg)}$

表 3 语句 s 及执行关系传递边 $s_{(edg)}$构造交叉表

	语句 S 被测试用例集覆盖	语句 S 未被测试用例集覆盖	执行关系传递边 $S_{(edg)}$ 被测试用例集覆盖	执行关系传递边 $S_{(edg)}$ 未被测试用例集覆盖
成功测试用例集(n_p)	$n_{ep}(s)$	$n_{np}(s)$	$n_{ep}(s_{(edg)})$	$n_{np}(s_{(edg)})$
失败测试用例集(n_f)	$n_{ef}(s)$	$n_{nf}(s)$	$n_{ef}(s_{(edg)})$	$n_{nf}(s_{(edg)})$
$\Sigma(n)$	$n_{ef}(s)+n_{ep}(s)$	$n_{nf}(s)+n_{np}(s)$	$n_{ef}(s_{(edg)})+n_{ep}(s_{(edg)})$	$n_{nf}(s_{(edg)})+n_{np}(s_{(edg)})$

通过量化缺陷传播路径可疑度,提高缺陷签名输入数据及特征选取的准确性,结合可疑度分析及缺陷状态传播分析达到对缺陷签名信息去冗余及补全的目的,解决因输入数据选择导致的信息冗余问题及缺少成功测试用例执行信息导致的信息缺失问题,分析路径可疑度中节点可疑度与边可疑度的关系,是提高缺陷签名分析准确性的重要研究工作.

3.2.2 缺陷传播过程分析

缺陷传播过程分析主要依据程序静态分析方法研究缺陷传播途径、方式及影响,程序静态分析具有多种研究方法,如定理证明[108]、约束求解[109]、模型检验[110]等.面向缺陷的研究将需求与实现之间的矛盾关系作为研究对象,静态分析方法主要解决的是缺陷传播过程中语句与缺陷关联关系的分析问题.多数研究利用程序关联规则及最弱前置条件分析缺陷传播过程.

数据挖掘领域存在关联关系分析研究,利用关联度分析方法挖掘对象关联规则及频繁项集,判断关联程度.关联关系分析同样应用在缺陷分析领域,结合关联规则与程序语义分析缺陷传播过程.关联规则是指如语句 $X,Y,X\rightarrow Y$ 的蕴含表达式,X,Y 分别为关联规则的先导(antecedent 或 left-hand-side)和后继(consequent 或 right-hand- side)[32].在程序分析领域,是指语句 X,Y 的有向性可达规则[111],其中,X 和 Y 是不同语句,若两者间不存在同一控制流中的关联路径,即 $X\cap Y=\varnothing$.语句关联规则强度通过支持度(support)和置信度(confidence)度量[32].支持度确定规则用于表示给定数据集的可达集合频繁程度,而置信度确定 Y 在包含 X 的执行路径中出现的频繁程度.支持度 s 和置信度 c 的定义如公式(3)、公式(4)所示.

$$s(X \rightarrow Y) = \frac{\sigma(X \cup Y)}{N} \tag{3}$$

$$c(X \rightarrow Y) = \frac{\sigma(X \cup Y)}{\sigma(X)} \tag{4}$$

其中,N 为数据集总量,$\sigma(X)$为数据集中包含 X 的总量,$\sigma(X\cup Y)$为数据集中同时包含 X,Y 的总量.

利用关联规则及语义分析,改善传统的基于概率统计方法将语句单点作为独立分析个体进行研究而忽略缺陷传播过程分析导致的计算误差.Baah 等人[88]利用概率程序依赖图分析缺陷传播过程,其中,节点表示方法或基本块,边表示相应的调用、转移或返回.每个节点具有概率分布、子节点状态分布与父节点关联,利用状态图的形式分析并计算语句间的缺陷状态传播可疑率.Chen 等人[57]利用语句及程序运行信息建立概率争议图,计算语句在争议图中的可疑率,通过缺陷传播可达路径对缺陷的根因位置与表征位置之间的可疑度传播方式进行建模,分析两者的关联度,解释缺陷描述与缺陷根因之间的关联关系.南京大学徐宝文团队的 Xu 等人[87]将推理分析(调试过程中不确定性分析)与程序语义分析相结合,构建缺陷传播概率模型,计算执行覆盖的语句实例和变量可疑度概率分布,定位缺陷的根因位置并分析缺陷产生的原因.

对于难以获得动态信息的程序,利用最弱前置条件(weakest precondition)分析缺陷来源及传播方式,生成精

确缺陷理解信息.该方法对测试用例没有要求,通过静态分析即可获得原因链,最弱前置条件分析对单条执行路径基本块进行筛选,即发现一条违反规约路径即可进行研究.不同于语句关联性分析,最弱前置条件分析更多关注缺陷产生条件及来源.

最弱前置条件指的是当存在一条程序执行路径 $\pi^{1,n}=s_e^1,...,s_m^n$ 和初始命题变量 ϕ 时,就会触发语句 s_m^n 的执行并满足命题(后置条件)的最低前提[111,112].初始命题变量 ϕ 关于路径 $\pi^{1,n}$ 的最弱前置条件表示为 $WP(\pi^{1,n},\phi)$,具体变量解释如下.

1) 赋值语句 s:v:=e,$WP(s,\phi)=\phi(e/v)$,其中,e/v 表示将最弱前置条件中的变量 v 用表达式 e 进行替换;
2) 分支语句 s:*assume*(c),$WP(s,\phi)=\phi\wedge c$;
3) 序列 $s_1;s_2$,$WP(s_1;s_2,\phi)=WP(s_1,WP(s_2,\phi))$,如在计算整数最大值程序中存在语句 $s_m^{n_0}$ 为 min=a,初始条件 ϕ 为 min=*MIN*,反向执行语句 min=a 后,$WP(s_m^{n_0},\phi)$ 变为 a=*MIN*.

Yi等人[43]利用语义约束求解寻找缺陷最弱前置条件,查找原因链,形成树型结构图,在定位缺陷位置的同时对缺陷的产生过程进行分析及表示,有效地提高了缺陷定位结果的可信性及可用性.Qi 等人[113]基于正确执行与错误执行之间存在路径不一致的前提假设,针对程序演化过程,基于最弱前置条件构造结果差异化输入,利用执行路径差异对缺陷进行定位及解释.

但关联规则分析及最弱前置条件分析都存在状态爆炸问题[43],关联规则分析存在对象间传播过程较为复杂的情况,对于存在于控制流图复杂度较高程序中的缺陷,其路径条件的解空间与程序分支数量呈指数关系,即存在路径搜索空间爆炸问题.同样,最弱前置条件在计算过程中受循环变量影响,故可能存在死循环等问题.为了解决复杂路径搜索空间爆炸及死循环问题,对于关联规则及最弱前置条件中关键语句或组件的识别及分析(即结合第 3.2.1 节所述的关键信息研究)是未来重要的缺陷理解研究方向.

3.2.3 缺陷理解信息表达形式及缺陷间关系研究

缺陷理解信息表达形式及缺陷间关系研究均利用缺陷与产物的关联关系及缺陷间的融合信息进行分析,通过融合产物与源代码信息,结合自然语言分析及源代码信息处理技术,保证缺陷理解信息表达形式的可解释性及准确性.通过缺陷间触发关系、依赖关系及竞争关系分析,结合缺陷语言信息处理方法,对复杂缺陷间关系及缺陷表达形式进行研究.

(1) 缺陷与产物的关联分析

缺陷的研究一直伴随着对相关产物的分析,包括测试用例(test case)、提交(commit)信息、调试(log)信息、缺陷报告(bug report)等,该类产物包含了测试人员及缺陷检测人员对缺陷的理解信息,例如,d 缺陷表征现象的可理解性信息、缺陷源代码与表征现象之间关系信息、源代码根因位置(错误点)与表征位置(触发缺陷现象代码位置、失效点)之间的关联信息等.该类信息多利用自然语言描述,是程序员对缺陷现象与需求之间“矛盾冲突”观察最直观的语言表达,是缺陷理解分析信息的重要来源.

自 2016 年前后自然语言处理及深度学习技术的兴起,缺陷产物理解信息提取、分析类论文数量大为提升.通过对缺陷产物、需求说明(requirements document)以及源代码(即缺陷位置、传播路径、表征位置等代码语言)信息进行聚类分析及缺陷语言处理,提取产物关联信息,形成以缺陷为中心的理解信息集合,辅助缺陷生命周期各过程研究.具体研究方法及目的见表 4.

该研究具有两大研究重点.

1) 在方法层面上的研究重点是解决语义分歧问题,自然语言描述与源代码之间的语义差异是该类研究的首要难点[66].该问题不仅存在于缺陷领域,也是自然语言研究与代码分析领域结合学科的关键研究问题.多数解决方法依据程序编写规则具有可理解性的假设,利用信息检索方法对该问题进行分析,利用搜索的方式解决语义鸿沟[114−118],将自然语言与源代码分析信息进行融合;
2) 在问题层面上的研究重点是解决产物与源代码位置的关联问题,如缺陷位置与测试用例及表征现象之间的“一对多及多对一”问题.以缺陷位置为研究对象,关联多种缺陷产物,形成以代码语言为中心的缺陷社团模型,解释产物与代码之间的关联关系,涉及到复杂社团聚类问题、社团间产物的链接问题

及社团在软件版本中传递演化问题.通过传统的链接关联及追踪方法难以实现对社团的描述及演化分析,目前的多数研究局限于缺陷位置与单一关联产物信息的分析[97],将缺陷作为单独版本的独立个体进行研究,忽略了产物社团聚类信息分析及缺陷间关联性分析的重要性.

Table 4 Research method and purpose based on defect products

表 4 基于缺陷产物的研究方法及目的

产物类型	年份	研究方法	研究目的
缺陷报告	2012	语义词典(semantic dictionary)及自然语言处理技术	自动生成缺陷描述文本
	2014	vector space model	缺陷报告信息去冗余
	2014	vector space model	缺陷现象与缺陷报告相关性分析
	2014	EC and EMC classifiers	缺陷报告摘要自动生成
	2017	LDA (latent dirichlet allocation)	对于描述文字较少的缺陷报告进行自然语言自动补全
提交信息	2013	support vector machine	建立修复与提交间的链接,实现缺陷追溯
	2015	random forest	建立提交与缺陷报告间的链接
	2018	stepped auto-encoder network	缺陷报告自然语言总结
测试用例	2018	statistical causal inference theory	实现差异点精确测试,分析根因位置与测试用例间的链接关系

对于缺陷报告,DiGiuseppe 等人[67]对缺陷报告中自然语言描述的准确性进行分析,提出语义故障诊断(semantic fault diagnosis)方法,结合测试人员自然语言描述及程序源代码语义信息(包括类名、方法名、变量表达式、开发人员注释和关键字)描述缺陷,有效提升了人工自然语言描述的准确性.Ye 等人[49]利用功能标准对源代码文件进行分解,将修复提交信息、API 功能描述及更改历史等领域知识作为参照,利用自适应排序技术对源文件与缺陷报告间的关联度进行排名,结合产物领域知识,对源文件查找准确性及产物关联性进行了提高.Zhang 等人[31]利用缺陷文本语义相似性对文字量较少的缺陷报告自然语言描述加以丰富,评估了短报告(描述少于 100 个单词)对缺陷修复时间的影响,证实了缺陷报告自然语言描述长度提高可以有效缩短缺陷修复时间,分析了缺陷产物质量与缺陷修复效率间的关联关系.

对于测试用例、提交信息等产物,根因测试(causal testing)通过测试用例选择达到缩小差异执行范围的目的,结合测试用例信息定位缺陷根因位置,利用测试用例及源代码执行路径信息对缺陷进行解释.Le 等人[97]通过源代码上下文语义信息构造提交信息与缺陷报告间的关联链接,提高关联关系精度及准确性,结合程序语义分析方法生成高质量的缺陷产物理解信息.Thung 等人[119]通过遍历内核代码提交,挖掘历史提交数据中后向移植代码变化信息,构造驱动后向移植信息推荐系统.

(2) 缺陷类别及关联关系研究

由于缺陷自身的复杂性及缺陷与环境间的强交互性,导致了复杂缺陷类型的出现.在工程应用中,难以保证单缺陷隔离环境,缺陷间具有依赖关系、触发关系及竞争关系等复杂逻辑关系,存在对于同一缺陷位置涉及多条测试用例触发不同缺陷现象以及同一缺陷现象对应多个缺陷位置的复杂关系.缺陷理解信息间具有关联性,在缺陷类别分析的基础上研究缺陷间关系,辅助生成完备性缺陷理解信息,实现缺陷的完整修复与全面检测.

缺陷类别分析关注缺陷特殊性,即缺陷属性间差异性,缺陷对环境的依赖以及测试用例输入的要求导致了缺陷自身以及环境交互方式的独特性,引发了多种缺陷类型的出现,例如具有触发条件、环境交互、缺陷现象、缺陷影响等属性特殊性缺陷.本文对属性特殊性缺陷进行了研究方法的梳理.

对于触发条件特殊性缺陷,测试用例与缺陷间的关系是该类缺陷的研究重点,例如:测试用例顺序触发缺陷,关注缺陷之间的影响关系,缺陷的触发难度导致了顺序执行缺陷难以被测试套件发现,该类缺陷要求测试用例对谓词分支路径实现充分的覆盖,分析测试用例执行与缺陷间的关系.控制流信息的提取及测试套件的完善是解决该类缺陷的关键研究方法,明确顺序触发缺陷间的逻辑关系是理解该类缺陷的研究要点.Shin 等人[89]对于突变测试基本原理框架进行了研究,提出了程序之间差异行为的概念,解释突变测试在缺陷定位及修复领域中应用的差异性.该研究分析了测试套件应用的可解释性,明确测试用例与缺陷检测之间的关联关系.利用缺陷理解技术分析触发条件特殊性缺陷.提高测试用例选择的精确性及缺陷触发的准确性,是未来复杂缺陷理解的

重要研究问题.

对于环境交互特殊性缺陷,Saha 等人[17]研究长期存在缺陷的分类和修复过程,通过对 7 个开源项目的研究,分析长期存在缺陷的存在比例、分布情况及严重性等特征.并且,Saha 等人[65]对于缺陷报告中严重性判断准确性进行了实验评估,分析了缺陷严重性类别对于自动分类、缺陷分配及修复时间预测等缺陷分析工作的影响.

对于缺陷影响特殊性缺陷,面对竞争激烈的软件市场,对于性能缺陷的研究有助于软件应用性的提升.Jin 等人[36]及 Zaman 等人[37]通过缺陷报告挖掘性能缺陷特征,分析性能缺陷的影响.Baltes 等人[83]利用开源软件库对性能缺陷的修复实例进行特殊性分析,明确性能缺陷修复方法的差异性.

在缺陷类别分析的基础上,研究缺陷间关联关系.本文对错误点具有复杂逻辑关系的缺陷进行了研究方法的梳理.

1) 多缺陷(multiple bugs)

大型工业系统难以保证传统定位工具中要求的单缺陷应用环境[5],减少外界及自身软件组件对缺陷实验环境的影响.复杂大型系统中存在多缺陷的情况较为普遍,由于系统自身功能具有交互且各组件之间产生互用,导致了系统中多数缺陷间具有关联关系.由于多个缺陷间产生相互作用,控制流经过多个关键谓词导致了缺陷触发的复杂性,执行路径分支的不确定性导致难以对多缺陷关系建模,且多缺陷受到测试用例与缺陷现象及根因位置间关系不明确性的影响,多数单缺陷假设下的缺陷定位方法[5,120−122],尤其是基于切片的定位方法及基于程序频谱的定位方法,利用单一测试用例执行路径或实现相似功能测试用例语句覆盖分布定位缺陷位置或分析缺陷产生原因,忽略了缺陷间的复杂关系,难以应用在存在多缺陷的工程环境中.

利用缺陷理解信息对多缺陷进行精确定位,较传统的基于多缺陷相互独立假设的研究[123,124]及使用聚类算法将失败的测试用例分组到不同的缺陷聚类群集中,基于同一群集中失败测试用例与同一缺陷相关假设的研究[125],定位精度得到了较大提高.Zheng 等人[126]利用遗传算法及模拟退火算法定位多缺陷问题,提出了 FSMFL 多缺陷定位框架,利用搜索的方式检测多缺陷之间的依赖关系,但缺乏从语义层面上对多缺陷关联关系的深入分析.Gao 等人[127]通过修正 Kendall tau 距离衡量失败测试用例之间的关系,评估缺陷聚类数量,分析多缺陷间的关联关系,实现多缺陷的并行定位.该方法通过距离度量预测多缺陷的分布情况,提高了多缺陷的定位精度.

利用缺陷理解信息对多缺陷性质进行特征分析,Yan 等人[82]从缺陷理解的角度对多缺陷间的显性故障及隐性故障进行了实验分析,利用测试用例执行情况对多缺陷分布比例、交互影响等问题进行研究,解释多缺陷研究中有关独立缺陷比例、缺陷间影响关系、多缺陷分布情况及影响等关键问题.

2) 并发缺陷(concurrency bug)

并发缺陷在程序执行交互、任务等待、多任务同时处理等复杂系统环境下产生,分为死锁(deadlock)、数据竞争(data race)、原子性违背(atomicity violation)及顺序违背(order violation)这 4 类[128].并发缺陷经常触发严重的系统故障,导致宕机等严重危害.并发缺陷难以暴露、难以检测、难以调试及修复,目前,多数的解决方法是对系统进行并发缺陷容错,将缺陷特殊情况单独处理,利用触发条件等约束,降低并发缺陷带来的系统危害性.

Chord[129]及 RacerX[130]利用静态分析方法查找并发缺陷,但由于该方法需要对所有路径进行探索,因此将这些工具应用于大型复杂程序是不切实际的.另一方面,利用静态方法辅助动态信息分析能够降低并发缺陷检测误报率[59],但只能对特定执行中出现的并发缺陷进行检测,应用环境受到限制.

综上可见缺陷理解对并发缺陷研究的重要性.Cai 等人[131]分析并发缺陷的产生条件,选择执行预采集线程,保证两个采集线程之间不存在其他同步,达到对保持和等待这两种死锁产生必要条件的消除,解决了死锁的自动修复问题.Park 等人[61]利用传统缺陷定位工具附加并发解释信息,分析内存使用情况及分配模式,查找可疑内存使用方式.Liu 等人[52]分析缺陷修复改动代码上下文,基于 Petri 网建立无缺陷模型,检测代码改动是否会产生并发缺陷,即违背原子性导致的死锁,保证程序演化过程中不产生新的并发缺陷.

3) 回归缺陷(regression bug)

回归测试(regression testing)中发现的缺陷称为回归缺陷[132].回归缺陷与上版本源代码相关联,回归缺陷的来源是该类缺陷理解研究的重点,回归缺陷具有 3 种产生情况:新功能的引入、旧版本缺陷修复及测试套件的

改变.回归测试的失败不仅与代码改动相关,测试用例改变同样导致缺陷的产生,在版本演化过程中,回归缺陷不一定是独立版本的产物,回归缺陷中存在代码改动而导致的新生缺陷,与上版本缺陷无关,同时存在上版本缺陷演化导致的回归缺陷,不能通过简单的自然语言描述现象对比即认定回归缺陷与上版本缺陷间的关联性,对于缺陷间的逻辑关系挖掘是判定回归缺陷来源的重要研究.

版本间的改动代码(diff code)及其依赖关系是研究回归缺陷与原版本缺陷关系的重点,diff codes 与原版本间代码的关联性解释回归缺陷产生的原因.Yi 等人[43]利用最弱前置条件确定可疑改动代码,查找原因链,结合约束求解语义分析形成树型图,解释回归缺陷来源.Qi 等人[113]利用符号执行生成新的输入,通过稳定版本的输入执行与存在缺陷版本的执行信息进行对比,实现缺陷的理解信息提取并同时定位回归缺陷位置.

缺陷类别及关联关系研究涉及缺陷种类繁多,缺陷特征属性分析是缺陷间关系研究的前提,通过模型构造或依赖关系挖掘等方法实现缺陷间关系分析,关联缺陷产物信息,通过产物信息整合,即缺陷与产物的关联分析,完善缺陷表达形式,推进缺陷处理进程.

4 缺陷理解研究的特点

缺陷理解的研究重点在于缺陷生命周期各过程中对缺陷进行定性及定量的分析研究,它涉及到缺陷信息的提取、过滤、关联、整合及表示,是一种复杂的缺陷分析过程,具有范围广、涉及对象多、过程复杂等研究难点.目前,软件系统运行环境及自身代码都趋于复杂化,在部分大型系统中,缺陷发现后很难进行有效分析,例如内核程序、云端系统、深度学习系统等,在人工分析尚存在较大困难的情况下,很难利用自动化的方法对缺陷进行特征挖掘、行为分析等理解性相关研究,这是缺陷理解研究较为困难的大背景.

4.1 缺陷的复杂性及独特性问题

首先,缺陷具有很强的复杂性.缺陷的产生原因可能发生在需求实现的各个环节中,部分缺陷具有一定的隐蔽性,难以检测,并且程序自身的复杂性导致了缺陷的多样性,例如,并发缺陷、隐性缺陷及多缺陷等.同时,缺陷所处软件环境的复杂性,也加大了缺陷分析理解的难度.总的来说,缺陷自身及环境的复杂性共同导致了缺陷分析的困难性.

1) 调试环境复杂,缺陷触发条件繁多:多数缺陷环境属于闭源系统,传统调试工具难以深入分析,涉及到外部组件繁多,难以对外部闭源插件实现进行验证,例如用户自建插件、服务端数据库实现等.缺陷的产生过程涉及到测试用例输入处理过程,程序输入被多方、多过程处理,各个环节均有可能导致缺陷的产生;
2) 多级函数调用,数据结构复杂:部分缺陷具有循环函数调用等复杂关系,传统的调试工具难以对关键变量繁多、多线程调用等复杂缺陷进行研究.同时,复杂父子数据结构相关的指针传递和转换流程增加了分析复杂性,例如:通过子数据结构获取父数据结构、指针传递层级较深等情况,这类复杂数据及调用关系加大了缺陷理解研究的困难程度.

并且,缺陷具有很强的独特性,同类缺陷在不同场景和软件中表现出的现象可能不同,同一根因位置导致的缺陷由于底层交互方式及依赖库不同,在不同环境中输出结果不同,利用通用的控制流或数据流分析工具很难获取有用信息,环境配置文件难以分析,并且在复杂系统中缺陷的产生具有一定的随机性,测试用例具有偶然正确(coincidental correctness)的情况.源代码的错误是导致大多数缺陷产生的原因[23],但是软件环境、配置问题、操作问题均对缺陷分析产生影响,给缺陷理解研究带来很大的困难.

4.2 缺陷理解信息难以表达及整合

缺陷理解信息难以进行可解释性表达,程序员的思考及行为信息具有难以形式化的问题,在大量调试信息中查找与特定缺陷相关的信息并且在其中精确筛选出有用信息,是缺陷理解的关键问题.传统的缺陷表示方法利用自然语言进行现象表述,利用单点根因位置代码表示错误点信息,两者之间存在自然语言与代码语言表达上的语义鸿沟,对于涉及到场景差异及程序员心理过程的理解信息,难以提出一种通用的表示框架.

并且,源代码理解信息与产物中自然语言描述信息难以进行融合,需要对代码错误位置进行“一对多及多对一”问题的分析,同一语句对应多个触发,多个位置对应同一种现象,不同测试用例触发同一缺陷位置的执行路径不同,同一表征位置(例如 printf 函数、assert 函数等)对应的缺陷根因位置可能不同,导致缺陷现象相同或相似但缺陷位置不同,因此,报告单个语句的定位结果对于缺陷修复研究来说并不准确,并且缺乏产物与源代码关联信息,没有给出一种完善的缺陷表示方式,这是导致缺陷修复效率不高的主要原因[53].结合精准执行可达性分析[111]对触发同一语句的测试用例及动态执行信息进行整合,以及对同一现象不同位置缺陷进行隔离分析,解决缺陷理解信息缺失及冗余问题.如图 6 所示:WINE 兼容层中缺陷代码段,*Tooltips_calculatesize* 函数的修改需要结合两个触发该位置缺陷的测试用例理解信息,即 Eduboard 中 Tooltips 文字显示不全及 FoxitRead 中 Tooltips 组件 Icon 不显示用例,函数 *Tooltips_calculatesize* 中没有考虑 Icon 与文字同时显示时,Tooltips 大小计算问题(该情况较少,难以被测试套件触发),导致该情况下 Icon 无法显示;并且 *TITLE_TEXT_SPACING* 变量大小无法根据显示内容更改,导致 Tooltips 文字显示不完全,而两者可能存在同一控制流中导致隐性缺陷的产生,因此,*Tooltips_calculatesize* 函数单独根据一条测试用例信息难以进行完整的源代码修改.

```
1  hdc=GetDC(infoPtr→hwndSelf);
2  if (infoPtr→pszTitle)
3  {
4     RECT rcTitle={0,0,0,0};
5     TRACE ("title %s\n", debugstr_w(infoPtr→pszTitle));
6     if (infoPtr→hTitleIcon)
7     {
8        title.cx=ICON_WIDTH;
9        title.cy=ICON_HEIGHT;
10    }
11 if (title.cx!=0)
12 {
13    title.cx+=ICON_TITLE_SPACING;
14 }
15 hOldFont=SelectObject(hdc,infoPtr→hTitleFont);
16 DrawTextW(hdc,infoPtr→pszTitle,−1,
   &rcTitle,DT_SINGLELINE|DT_NOPREFIX|
17 DT_CALCRECT);
18 SelectObject(hdc,hOldFont);
   title.cy=max(title.cy,rcTitle.bottom−rcTitle.top)+
19 TITLE_TEXT_SPACING;
20 title.cx+=(rcTitle.right−rcTitle.left);
   }
```

Fig.6 Defect code segments corresponding to multiple test cases

图 6 对应多个测试用例缺陷代码段

Parnind 等人[12]发现:真实项目中,修复人员通过自然语言描述关键词,过滤与程序状态相关的产物,选取可用的调试信息.通过这类信息与修复人员自身项目背景知识相结合,整合多对象知识要点,形成缺陷修复方式,有效地对缺陷进行完整性修复.由于缺陷理解信息难以表达及整合,利用搜索技术对基于排名的缺陷定位技术进行调试信息完善分析及针对修复人员行为挖掘的研究,将成为缺陷理解领域未来分析的重点.

4.3 缺陷理解指标难以评价

从广义的角度来说,缺陷理解贯穿在缺陷生命周期的各研究过程中,缺陷理解的评价指标应与具体应用场景相对应,将引入缺陷理解后该应用场景的效果增量作为基础设计评价指标,并且,缺陷理解研究与程序员自身的知识结构具有主观的联系,缺陷理解服务对象(程序员、工具)差异引发了对人工调试过程中行为的挖掘及分析,产生了模拟人工分析思路的缺陷分析研究[87].缺陷理解的验证指标由于应用场景、需求及服务对象的差异性,构造策略具有一定的主观性,目前的验证方式分为增量验证及评价验证,增量验证方式多采用静态评价指标构造或动态实验效果差异性衡量理解信息有用性,评价验证方式利用人工评估进行验证,通过人工思考评估缺陷理解信息的可用性.因此,缺陷理解指标难以评价,特别是有用性的衡量及精确性的量化都比较困难.

增量验证方式中利用特定参数,通过静态评估策略或动态实验验证方式进行定量衡量,但却存在着特征难以选取、验证策略难以构造等问题,难以提出一种通用的定量衡量方式验证理解信息的准确性.同时,面临着不同场景下需求的差异性,对于增量的刻画存在多种方式.

在面向缺陷定位的缺陷理解研究中,利用错误定位指标反馈式地考察调试人员有效理解错误根源所需工作量,即引入缺陷理解研究后对缺陷定位工作的影响进行量化.Liu 等人[133]延用了 Yu 等人[134]验证测试套件差异性对频谱缺陷定位工作量影响的衡量策略,如公式(5)所示,*Expense* 表示有效定位缺陷所需工作量[134].但是,Liu 采用 *Expense_increase*(即采用方法前后的 *Expense* 差值)定义测试执行选择对错误根源理解产生的工作

量变化,衡量缺陷理解带来的效果增量.

$$Expense = \frac{\text{rank of fault statements}}{\text{number of executable statements}} \times 100\% \quad (5)$$

在面向缺陷修复的缺陷理解研究中,多数采用动态验证方式衡量理解信息的准确性及有用性.Vassallo 等人[135]利用修复时间差异及减少比例衡量持续集成中断修复建议的有效性;Lawall 等人[136]将驱动移植问题转化为代码演化过程中的补丁生成研究,利用 PQL(patch query language)查询历史数据获得驱动移植所需的帮助信息,通过衡量补丁正确数量的差异性说明理解信息对于驱动移植问题的有用性.

在增量验证方式中,评价指标自身的正确性难以证明,由于场景、需求、应用及服务对象等因素,策略构造具有一定的主观性,验证过程中难以保证策略的完备性及充分性.同时,实验过程难以消除人为知识结构等主观因素的影响.对于验证策略构造方面,目前仍然缺少针对缺陷理解信息的通用静态评估方式:首先,工作量的定义缺乏正确性证明,影响效果的参数数量繁多,对于参数间的关系缺乏研究;第二,静态验证方式难以衡量理解信息的准确性,即理解信息本身可能存在缺失与冗余,但目前尚未有研究证明增量间的差异与信息自身准确性间的关系.对于实验环境一致性方面,不仅要排除人为因素对实验的影响,同时需要对实验参数进行扩充,对修复时间、阅读时间、修复效果等多方因素进行全面的考量,并且分析不同参数的计算权重,研究参数间的影响关系,构造动态验证实验环境,给出一种较为合理的缺陷理解信息修复实验评估规则.

评价验证方式中大多采用人工打分机制,即主观评价方式对理解信息可用性进行评估.Vassallo 等人[135]在修复时间验证的基础上构建应用实验,人为地对信息的可理解性(understandability)、相关性(relevance of proposed solution)及适用性(applicability of proposed solution)进行评级,通过与原始信息间的比较,构造可用性评价标准,即 very low、below average 到 very high 这 5 种评价等级,并在此基础上,对于 very low 评级分析人工背景知识与评级间的关系,发现参与者自身的编程经验与评价级别具有较强的联系,对信息准确性的评价产生了较大影响.

评价验证方式具有难以控制实验变量的问题,人工背景知识及领域知识对实验结果产生直接的影响,难以实现控制单变量实验,评价结果与参与者自身的经验知识、项目认知具有极大的关联性,并且,该类实验缺乏严谨性,对于多种方法理解信息的比较、实验顺序对于实验结果的影响并没有进行明确的说明(信息会丰富人为认知,对后续信息验证产生影响),实验效果说服力较小.

另一方面,目前研究缺乏对理解信息的不完备性及冗余性的刻画,例如缺陷签名方法中切片信息涉及到非缺陷对应功能的冗余问题[79]及最小差异化分析导致差异点关联数据缺失问题[51].增量与精确性间的关系缺乏证明.增量间的差异能否量化信息准确性,是未来缺陷理解研究需要思考的重要问题.

因此,目前缺陷理解信息验证存在指标构造策略难以证明及精确性难以刻画的问题.指标的主观性难以消除,实验的一致性难以保证,并且缺少对于理解信息精确性的研究,缺陷理解信息验证与衡量在保证有用性分析的前提下趋于精准性量化研究.

5 总结与展望

随着软件系统内在规模和复杂度上的不断提高,缺陷理解成为缺陷研究领域的重要研究方向,缺陷理解研究是模拟程序员在缺陷分析处理过程中心理过程及行为模式的研究领域,对于缺陷生命周期分析以及缺陷检测、分类、定位、修复及验证都具有较为重要的作用,甚至对代码分析领域,缺陷理解研究都具有较深的影响.本文从工程研究中遇到的实际缺陷问题及学术论文研究应用局限性问题入手,从理论及应用两方面阐述了缺陷理解问题.目前,缺陷研究逐渐由缺陷精度粒度的研究趋向于挖掘缺陷本源研究,证明了缺陷理解研究具有丰富的理论价值及应用前景.但缺陷理解研究中仍然存在缺陷模糊性、验证主观性及多场景下应用差异性问题,并且缺陷理解研究给缺陷分析领域研究带来的新思考值得我们进一步加以关注.

从分析对象的角度,缺陷信息融合技术向实现信息完备性及精确性的方向发展.目前的研究集中解决缺陷社团成员间的关系,例如,位置与现象、缺陷与需求的对应,该研究已在传统意义缺陷分析的基础上进行了较深

入的思考.但缺陷的模糊性及理解的主观性在目前的缺陷理解研究中讨论的很少,部分复杂缺陷具有一定的不可判定性,现象描述具有较强的模糊性.证明分析输入自身的正确性及关联性是缺陷信息生成研究中需要考虑的重要问题,研究前提假设的合理性直接影响缺陷理解信息生成的效果.并且,缺陷的处理过程多处涉及到理解主观性的产物,基于历史数据的缺陷定位研究[14,49,55]考虑缺陷报告中的描述及总结信息,实现跨项目的缺陷定位,该类研究输入数据的正确性直接决定了结果的可用性.有专门的研究对缺陷报告的完备性及正确性间的关系进行分析[31],具有主观性的输入数据对于理解问题效果的影响需要考虑前提假设的合理性及完备性.从缺陷分析研究数据集的角度,传统的缺陷研究大多利用人工错误注入方式构建的数据集 Siemens suite、NanoXML 等,人工注入错误导致了缺陷的来源具有一定的不可解释性,真实项目中的缺陷由于修复错误、需求变更、内核升级等目的由改动代码导致了缺陷自身及缺陷间的关系随软件生命周期而产生变化,观察缺陷产生、演化到消亡的过程是理解缺陷到解决缺陷,甚至防御缺陷的必要工作.并且,目前的软件系统逐渐趋于复杂,对于注入缺陷数据的分析难以满足工程应用的需求,跨版本间的缺陷趋势分析逐渐被研究人员重视,并且云端系统及分布式大型系统的出现对于缺陷研究各领域工作都提出了更高的要求.缺陷的研究不应脱离真实的应用场景,将缺陷理解研究映射到具体系统环境中,分析缺陷与环境间的关系,利用缺陷理解研究使缺陷分析更加趋于实用性.

从应用场景的角度,由于缺陷理解概念的产生环境、缺陷研究各领域问题特性差异及发展情况等原因,多数缺陷理解研究集中在缺陷定位与缺陷修复间的过程,但缺陷理解同样应用于检测、分类、复现甚至补丁验证等缺陷分析相关领域.其中有两个问题值得关注.

- 第一,缺陷的本质属性是否存在跨越应用场景及服务对象的通用性,研究方法间的相似性表示了在不同缺陷研究领域中解决问题的出发点相似甚至相同.例如:对于缺陷错误点与修复信息(例如 commit)间的关系,基于信息检索的缺陷定位研究[137]利用已有修复文件与错误间的对应关系定位新的缺陷,考虑 commit 上下文结构与源代码间的相似性.定向灰盒模糊测试(directed grey-box fuzzing)[138,139]利用 commit 信息作为目标,通过基本块与目标块间的距离检测漏洞.实验结果证明:commit 上下文可以有效提高缺陷检测效率,缩小缺陷检测范围.两者均利用 commit 上下文实现缺陷分析的目的,缺陷研究方法的可扩展性分析是缺陷理解重要的研究趋势;
- 第二,应用场景的差异性是否导致了理解信息的变化,信息与场景间是否存在对应关系.缺陷定位多数研究关注缺陷与上下文代码间作用导致的违背软件正常运行的特征与现象间的映射关系,缺陷检测多数研究关注缺陷与源代码运行间的特征差异,缺陷分类更加关注缺陷特征间的差异性.

挖掘缺陷分析研究的本质是缺陷理解问题的初衷,应用场景差异与理解信息变化方式间的关系是缺陷理解未来重要的研究方向.

从缺陷理解与其他研究领域结合的角度,面对复杂的缺陷系统,利用上下文语义信息知识推理、深度学习技术及信息检索方法对关联信息进行提取,剔除简单的单一匹配关联方式,对源代码及产物之间的融合方式进行动态建模,利用自动化的方法对信息进行关联,深度学习及信息检索技术对于链接准确性及精确性的提高有较大的帮助.对于缺陷知识的累积,利用知识图谱及社团聚类方法,对多个数据源对象抽取知识进行融合,构造缺陷知识累积系统,知识图谱技术为缺陷知识推荐信息系统、知识推理、预测分析及缺陷可视化研究提供了一条有效途径.

从缺陷理解引发缺陷分析研究新问题的角度,缺陷分析作为较为成熟的研究领域,面临着应用面难以扩展及技术手段趋于饱和的研究瓶颈问题,如何从老的方向中发现新的问题,是目前缺陷分析研究的重要内容.辅助修复位置定位、分析缺陷可追溯性及构建缺陷生态系统,是打开缺陷分析研究的新窗口.缺陷知识的提取、表达及重用是缺陷分析乃至软件工程领域较为重要的研究方向,例如,目前缺陷仓库概念的提出[55]以及诸多技术问答社区的普及,缺陷理解研究引发了缺陷分析领域中新的研究问题与思考.

1) 传统缺陷定位工具简单地将缺陷根因位置视为修复位置,但面对复杂缺陷,缺陷存在环境及触发过程均对修复位置产生影响,由于人为判定或开发环境等因素导致修复位置并不是缺陷的根因位置,例如,涉及全局变

量的缺陷、缺少代码导致的缺陷及跨过程缺陷根因位置与修复位置间存在的复杂关系.

涉及全局变量的缺陷由于全局变量与其他变量间的强交互性,需要考虑关联缺陷的情况,难以根据单一测试用例对全局变量进行修改.缺失代码缺陷导致了变量传递及处理的异常行为,根因位置具有较强的隐蔽性,定位结果存在较大误差,导致修复位置往往与根因位置具有很大差别.跨过程缺陷经过多对象处理(例如,客户端与服务器的信息交互、外部插件的影响等)及复杂的父子结构数据传递过程(例如,指针传递及转换),修复位置具有多样性及可选择性,需要根据上下文场景决定,导致了错误点与修复位置间的差异性.

随着自动缺陷修复研究的兴起,缺陷修复位置定位问题亟待解决,结合定位结果与缺陷理解信息有效地推进软件调试自动化进程.辅助缺陷理解研究对于修复位置的精确定位是缺陷分析领域研究的新思考.

2) 缺陷的理解不止是对源代码的分析,缺陷产物的关联性以及缺陷在版本之间的演化过程分析也是缺陷理解研究的关键问题.缺陷理解研究辅助缺陷可追溯性分析,指的是缺陷与产物之间的关联关系以及版本间缺陷演化过程分析,主要解决缺陷社团各部分之间的追溯性问题,例如,测试用例、缺陷报告、提交信息等产物及缺陷根因位置及表征位置等源代码之间的关联关系研究.目前,多数可追溯性研究都是基于需求的分析[140−141],对于缺陷产物间关联性分析仅是对两者间的映射关系进行研究[47,81],并没有形成可追溯性研究系统,对于测试用例或缺陷报告等单对象产物的考虑导致分析结果可用性较低.并且,对于缺陷的研究均是将缺陷视为单版本的产物,忽视了缺陷跨版本的演化问题,回归缺陷也仅是在源代码级别上分析缺陷来源.但缺陷可能存在于跨版本中,缺陷可追溯性研究从缺陷演化本质上分析缺陷产生的根源,刻画缺陷的演化过程,根据工程实践经验,缺陷的演化过程具有多种形式.

- 从无到有、从有到无:即缺陷的产生与消亡;
- 扩张、缩减:利用触发缺陷的测试用例数量衡量缺陷的严重性,在保证测试套件不变性的假设下,缺陷经过版本间演化更易触发,即触发该位置缺陷的测试用例较上版本数量更多,认为缺陷进行了跨版本的扩张,缺陷较上版本难以触发,测试用例数量较上版本少,即认为该缺陷产生了缩减;
- 合并、分裂:具有依赖关系的多缺陷,如顺序执行缺陷,由于版本间代码变更,导致分裂为无触发关系及依赖关系缺陷,称缺陷产生了分裂.相反,上版本独立缺陷经过软件演化,形成了具有关联关系(触发关系、变量状态关联关系等)的显性缺陷及隐性缺陷,即认为缺陷产生了跨版本合并.

缺陷位置与现象之间的关联,一直是困扰缺陷研究领域的问题之一[5,142].缺陷位置不能完全解释缺陷的产生根因,缺陷可追溯性研究有助于缺陷的根本修复而不是简单地跳过或掩盖,有助于解决补丁生成过程中的过拟合问题.追溯缺陷产生的原因,刻画缺陷社团演化过程,从可追溯性的角度,以更广的范围分析缺陷问题.

3) 缺陷演化过程及可追溯性分析,引发了对缺陷生态系统的构建研究.缺陷生态系统的概念来自于软件生态系统[143],开源软件生态系统通过共享软件和服务进行无障碍的行为交互,实现了相似或关联开源软件间的知识复用[144].Nie 等人[75]将缺陷视为节点,缺陷之间的阻塞关系视为边,模拟系统中缺陷的增长过程,研究同一系统中缺陷的关联性及相似性.但目前的研究缺乏缺陷系统性概念,并没有对缺陷多对象信息及缺陷间关系进行深入的分析.通过缺陷理解研究挖掘缺陷生命周期各研究过程间的潜在关系,对缺陷及产物进行聚类分析及社团演化模型构建,形成缺陷知识网络,有效地解决缺陷研究中仅考虑单缺陷及单对象导致的误差问题.结合缺陷检测、触发、描述、定位、修复及验证过程,实现缺陷的完整性分析,对于缺陷系统性研究具有极为重要的作用;同时,有效解决了目前缺陷研究难以成体系、相互缺陷之间的信息无法进行互通及参考、大量缺陷分析知识没有进行体系性的累积及复用的问题.结合缺陷理解技术构建缺陷生态系统,是未来系统自主排错、自主升级及自动演化的重要研究内容.

深度学习、知识图谱及信息检索等基础性研究学科的发展,促进了缺陷理解领域新方法的出现.软件系统自身复杂性及可靠性需求的提高,促进了缺陷理解精确性及准确性的提升.区块链及隐私保护等新兴研究领域的出现,促进了缺陷理解应用面的扩展.另外,智能软件的大量涌现,促进了缺陷自处理问题的研究.未来软件系统趋于智能化、复杂化及自动化,要求系统自我识别、处理、消除缺陷,支持缺陷的自我认识及自动识别与修复.未来缺陷的自动理解及解决是人类智能生活的关键基石,缺陷理解技术是缺陷自处理的基础,缺陷理解研究

对未来人类进入全面无人机器自动化阶段有着重要的研究价值,人类进入全面智能化生活值得期待.

References:

[1] Jones JA. Semiautomatic fault location [Ph.D. Thesis]. Georgia Institute of Technology, 2008.

[2] Mei H, Wang QX, Zhang L, Wang J. Software analysis: A road map. Chinese Journal of Computers, 2009,32(9):1697−1710 (in Chinese with English abstract). [doi: 10.11897/SP.J.1016.2015.02203]

[3] Wang KC, Wang TT, Su XH, Ma PJ. Key scientific isues and state-art of automatic software fault localization. Chinese Journal of Computers, 2015,38(11):2262−2278 (in Chinese with English abstract). [doi: 10.11897/SP.J.1016.2015.02262]

[4] Xu X, Debroy V, Wong WE, Guo D. Ties within fault localization rankings: Exposing and addressing the problem. Int'l Journal of Software Engineering and Knowledge Engineering, 2011,21(6):803−827. [doi: 10.1142/S0218194011005505]

[5] Wong WE, Gao RZ, Li YH, Abreu R, Wotawa F. A survey on software fault localization. IEEE Trans. on Software Engineering, 2016,42(8):707−740. [doi: 10.1109/TSE.2016.2521368]

[6] Yoo S, Moon S, Kim Y, Shin Y. Ask the mutants: Mutating faulty programs for fault localization. In: Proc. of the 7th IEEE Int'l Conf. on Software Testing. IEEE, 2014. 153−162. [doi: 10.1109/ICST.2014.28]

[7] Lei Y, Mao X, Zhang M, Ren J, Jiang Y. Toward understanding information models of fault localization: Elaborate is not always better. In: Proc. of the Computer Software & Applications Conf. IEEE, 2017. 57−66. [doi: 10.1109/COMPSAC.2017.246]

[8] Le TDB, Thung F, Lo D. Will this localization tool be effective for this bug? Mitigating the impact of unreliability of information retrieval based bug localization tools. Empirical Software Engineering, 2017,22(4):2237−2279.

[9] Pearson S, Campos J, Just R, Fraser G, Abreu R, Ernst M, Pang D, Keller B. Evaluating and improving fault localization. In: Proc. of the 39th IEEE/ACM Int'l Conf. on Software Engineering (ICSE). IEEE Computer Society, 2017. 609−620. [doi: 10.1109/ICSE.2017.62]

[10] Le TDB, Lo D, Thung F. Should I follow this fault localization tool's output? Empirical Software Engineering, 2015,20(5): 1237−1274.

[11] Lucia L, Thung F, Lo D, Jiang L. Are faults localizable? In: Proc. of the Mining Software Repositories. IEEE, 2012. 74−77. [doi: 10.1109/MSR.2012.6224302]

[12] Parnin C, Orso A. Are automated debugging techniques actually helping programmers? In: Proc. of the Int'l Symp. on Software Testing & Analysis. ACM Press, 2011. 199−209. [doi: 10.1145/2001420.2001445]

[13] Motwani M, Sankaranarayanan S, Just R, Brun Y. Do automated program repair techniques repair hard and important bugs? Empirical Software Engineering, 2018,23:2901−2947.

[14] Lam AN, Nguyen AT, Nguyen HA, Nguyen TN. Combining deep learning with information retrieval to localize buggy files for bug reports. In: Proc. of the 30th IEEE/ACM Int'l Conf. on Automated Software Engineering. Lincoln, 2015. 476−481.

[15] Xiao Y, Jacky K, Kwabena B, Mi Q. Machine translation-based bug localization technique for bridging lexical gap. In: Proc. of the Information and Software Technology. 2018. 58−61.

[16] Huo X, Li M, Zhou ZH. Learning unified features from natural and programming languages for locating buggy source code. In: Proc. of the 25th Int'l Joint Conf. on Artificial Intelligence. New York, 2016. 1606−1612.

[17] Saha RK, Khurshid S, Perry DE. Understanding the triaging and fixing processes of long lived bugs. Information and Software Technology, 2015,65:114−128. [doi: 10.1016/j.infsof.2015.03.002]

[18] Zhou J, Zhang HY. Learning to rank duplicate bug reports. In: Proc. of the 21st ACM Int'l Conf. on Information and Knowledge Management (CIKM). New York: ACM Press, 2012. 852−861. [doi: 10.1145/2396761.2396869]

[19] Yu K, Lin MX. Advances in automatic fault localization techniques. Chinese Journal of Computers, 2011,34(8):1411−1422 (in Chinese with English abstract). [doi: 10.3724/SP.J.1016.2011.01411]

[20] Goues CL, Nguyen T, Forrest S, Weimer W. GenProg: A generic method for automatic software repair. IEEE Trans. on Software Engineering, 2012,38(1):54−72.

[21] Kim D, Nam J, Song J, Kim S. Automatic patch generation learned from human-written patches. In: Proc. of the 35th Int'l Conf. on Software Engineering. San Francisco, 2013. 802−811.

[22] Avizienis A, Laprie JC, Randell B, Landwehr CE. Basic concepts and taxonomy of dependable and secure computing. IEEE Trans. on Dependable and Secure Computing, 2004. 11−33. [doi: 10.1109/TDSC.2004.2]

[23] Binkley D. Source code analysis: A road map. In: Proc. of the 2007 Future of Software Engineering (FOSE). Washington: IEEE Computer Society, 2007. 104−119.

[24] Jin Z, Liu F, Li G. Program comprehension: Present and future. Ruan Jian Xue Bao/Journal of Software. 2019,30(1):110−126 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5643.htm [doi: 10.13328/j.cnki.jos.005643]

[25] Weiser M. Programmers use slices when debugging. Communications of the ACM, 1982,25(7):446−452.

[26] Li XY, Zhu SW, d'Amorim M, Orso A. Enlightened debugging. In: Proc. of the 40th Int'l Conf. on Software Engineering (ICSE). New York: ACM Press, 2018. 82−92.

[27] Woo M, Cha SK, Gottlieb S, Brumley D. Scheduling black-box mutational fuzzing. In: Proc. of the 2013 ACM SIGSAC Conf. on Computer & Communications Security. New York: ACM Press, 2013. 511−522.

[28] Wang J, Chen B, Wei L, Liu Y. Skyfire: Data-driven seed generation for fuzzing. In: Proc. of the IEEE Symp. on Security and Privacy (SP). San Jose, 2017. 579−594. [doi: 10.1109/SP.2017.23]

[29] Kang M, Mccamant S, Poosankam P, Song D. DTA++: Dynamic taint analysis with targeted control-flow propagation. In: Proc. of the Network and Distributed System Security Symp. (NDSS 2011). San Diego: DBLP, 2011. 1−14.

[30] Tian S, Liang HL. Survey of static analysis methods for binary code vulnerability. Computer Science, 2009,36(7):8−14.

[31] Zhang T, Chen J, Jiang H, Luo X, Xia X. Bug report enrichment with application of automated fixer recommendation. In: Proc. of the 25th IEEE/ACM Int'l Conf. on Program Comprehension (ICPC). IEEE, 2017. 230−240. [doi: 10.1109/ICPC.2017.28]

[32] Groce A. Error explanation with distance metrics. In: Jensen K, Podelski A, eds. Proc. of the Tools and Algorithms for the Construction and Analysis of Systems (TACAS). Berlin, 2004. 108−122.

[33] Zhang J, Zhang C, Xuan J, Xiong Y, Wang Q, Liang B, Li L, Dou W, Chen Z, Chen L, Cai Y. Recent progress in program analysis. Ruan Jian Xue Bao/Journal of Software, 2019,30(1):80−109 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5651.htm [doi: 10.13328/j.cnki.jos.005651]

[34] Souza HAD, Chaim ML. Adding context to fault localization with integration coverage. In: Proc. of the IEEE/ACM Int'l Conf. on Automated Software Engineering. IEEE Press, 2013. 628−633. [doi: 10.1109/ASE.2013.6693124]

[35] Lucia L, David L, Jiang L, Thung F, Budi A. Data from: Extended comprehensive study of association measures for fault localization. Journal of Software Maintenance & Evolution Research & Practice, 2014,26(2):172−219.

[36] Jin GL, Song LH, Shi XM, Scherpelz J, Lu S. Understanding and detecting real-world performance bugs. In: Proc. of the 33rd ACM SIGPLAN Conf. on Programming Language Design and Implementation (PLDI). New York: ACM Press, 2012. 77−88.

[37] Zaman S, Adams B, Hassan A. A qualitative study on performance bugs. Mining software repositories. In: Proc. of the 9th IEEE Working Conf. on Mining Software Repositories (MSR). Piscataway: IEEE Press, 2012. 199−208.

[38] Zuo ZQ, Khoo SC, Sun CN. Efficient predicated bug signature mining via hierarchical instrumentation. In: Proc. of the 2014 Int'l Symp. on Software Testing and Analysis (ISSTA). New York: ACM Press, 2014. 215−224.

[39] Lo D, Cheng H, Wang X. Bug signature minimization and fusion. In: Proc. of the 13th IEEE Int'l Symp. on High-assurance Systems Engineering. Boca Raton, 2011. 340−347. [doi: 10.1109/HASE.2011.36]

[40] Kong X, Zhang L, Wong WE, Li B. Experience report: How do techniques, programs, and tests impact automated program repair? In: Proc. of the 26th IEEE Int'l Symp. on Software Reliability Engineering. IEEE Computer Society, 2015. 194−204.

[41] Inozemtseva L. Understanding the software fault introduction process. In: Proc. of the IEEE/ACM Int'l Conf. on Software Engineering. IEEE, 2015. 843−846. [doi: 10.1109/ICSE.2015.274]

[42] Schwartz-Narbonne D, Oh C, Schäf M, Wies T. VERMEER: A tool for tracing and explaining faulty C programs. In: Proc. of the IEEE/ACM Int'l Conf. on Software Engineering. IEEE, 2015. 737−740. [doi: 10.1109/ICSE.2015.236]

[43] Yi QP, Yang ZJ, Liu J, *et al*. A synergistic analysis method for explaining failed regression tests. In: Proc. of the 37th IEEE/ACM Int'l Conf. on Software Engineering. Florence, 2015. 257−267. [doi: 10.1109/ICSE.2015.46]

[44] Rahman MM, Roy CK. Improving IR-based bug localization with context-aware query reformulation. In: Proc. of the 26th ACM Joint Meeting on European Software Engineering Conf. and Symp. on the Foundations of Software Engineering (ESEC/FSE). New York: ACM Press, 2018. 621−632.

[45] Davies S, Roper M. What's in a bug report? In: Proc. of the 8th ACM/IEEE Int'l Symp. on Empirical Software Engineering and Measurement (ESEM). New York: ACM Press, 2014. 10.

[46] Wang SW, Lo D. Version history, similar report, and structure: Putting them together for improved bug localization. In: Proc. of the 22nd Int'l Conf. on Program Comprehension (ICPC). New York: ACM Press, 2014. 53−63.

[47] Romo BA, Capiluppi A. Towards an automation of the traceability of bugs from development logs—A study based on open source software. In: Proc. of the 19th Int'l Conf. on Evaluation and Assessment in Software Engineering (EASE). New York: ACM Press, 2015.

[48] Rastkar S, Murphy GC, Murray G. Automatic summarization of bug reports. IEEE Trans. on Software Engineering, 2014,40(4): 366−380.

[49] Ye X, Bunescu R, Liu C. Learning to rank relevant files for bug reports using domain knowledge. In: Proc. of the 22nd ACM SIGSOFT Int'l Symp. on Foundations of Software Engineering (FSE). New York: ACM Press, 2014. 689−699.

[50] Kochhar PS, Tian Y, Lo D. Potential biases in bug localization: Do they matter? In: Proc. of the 29th ACM/IEEE Int'l Conf. on Automated Software Engineering (ASE). New York: ACM Press, 2014. 803−814.

[51] Zuddas D, Jin W, Pastore F, Leonardo M, Alessandro O. MIMIC: Locating and understanding bugs by analyzing mimicked executions. In: Proc. of the ACM/IEEE Int'l Conf. on Automated Software Engineering (ASE). ACM Press, 2014. 815−825.

[52] Liu P, Tripp O, Zhang CC. Grail: Context-aware fixing of concurrency bugs. In: Proc. of the 22nd ACM SIGSOFT Int'l Symp. on Foundations of Software Engineering (FSE). New York: ACM Press, 2014. 318−329.

[53] Perez A, Abreu R. Cues for scent intensification in debugging. In: Proc. of the 2013 IEEE Int'l Symp. on Software Reliability Engineering Workshops (ISSREW). Pasadena, 2013. 120−125. [doi: 10.1109/ISSREW.2013.6688890]

[54] Ma P, Wang Y, Su X, Wang T. A novel fault localization method with fault propagation context analysis. In: Proc. of the 3rd Int'l Conf. on Instrumentation, Measurement, Computer, Communication and Control. Shenyang, 2013. 1194−1199. [doi: 10.1109/IMCCC.2013.265]

[55] Sun CN. Software bug management [Ph. D. Thesis]. Singapore: National University of Singapore, 2013.

[56] Pastore F, Mariani L. AVA: Supporting debugging with failure interpretations. In: Proc. of the 6th IEEE Int'l Conf. on Software Testing, Verification and Validation. Luxembourg. 2013. 416−421. [doi: 10.1109/ICST.2013.58]

[57] Chen R, Liu Y, Jia Z, Gao J. Isolating and understanding program errors using probabilistic dispute model. In: Proc. of the IEEE 37th Annual Computer Software and Applications Conf. Kyoto, 2013. 633−638. [doi: 10.1109/COMPSAC.2013.102]

[58] Thung F, Lo D, Jiang L. Automatic recovery of root causes from bug-fixing changes. In: Proc. of the 20th Working Conf. on Reverse Engineering (WCRE). Koblenz, 2013. 92−101. [doi: 10.1109/WCRE.2013.6671284]

[59] Park S, Harrold MJ, Vuduc R. Griffin: Grouping suspicious memory-access patterns to improve understanding of concurrency bugs. In: Proc. of the 2013 Int'l Symp. on Software Testing and Analysis (ISSTA). New York: ACM Press, 2013. 134−144.

[60] Sukkerd R, Beschastnikh I, Wuttke J, Zhang S, Bruri Y. Understanding regression failures through test-passing and test-failing code changes. In: Proc. of the 35th Int'l Conf. on Software Engineering (ICSE). San Francisco, 2013. 1177−1180. [doi: 10.1109/ICSE.2013.6606672]

[61] Park S. Fault comprehension for concurrent programs. In: Proc. of the 2013 Int'l Conf. on Software Engineering (ICSE). Piscataway: IEEE Press, 2013. 1444−1446.

[62] Sumner WN. Automated failure explanation through execution comparison [Ph.D. Thesis]. West Lafayette: Purdue University, 2013.

[63] Gao Z, Chen Z, Feng Y, Luo B. Mining sequential patterns of predicates for fault localization and understanding. In: Proc. of the IEEE Int'l Conf. on Software Security & Reliability. Gaithersburg, 2013. 109−118. [doi: 10.1109/SERE.2013.33]

[64] Li XC, Jiang H, Liu D, Ren ZL, Li G. Unsupervised deep bug report summarization. In: Proc. of the 26th Conf. on Program Comprehension (ICPC). New York: ACM Press, 2018. 144−155.

[65] Saha RK, Lawall J, Khurshid S, Perry DE. Are these bugs really "normal"? In: Proc. of the 12th IEEE/ACM Working Conf. on Mining Software Repositories. Florence, 2015. 258−268. [doi: 10.1109/MSR.2015.31]

[66] Zhou J, Zhang H, David LO. Where should the bugs be fixed? More accurate information retrieval-based bug localization based on bug reports. In: Proc. of the 34th Int'l Conf. on Software Engineering (ICSE). Zurich, 2012. 14−24. [doi: 10.1109/ICSE.2012.6227210]

[67] Digiuseppe N, Jones JA. Semantic fault diagnosis: Automatic natural-language fault descriptions. In: Proc. of the ACM SIGSOFT the 20th Int'l Symp. on the Foundations of Software Engineering (FSE). New York: ACM Press, 2012.

[68] Wang Y, Zhong H. An empirical study of multi-entity changes in real bug fixes. In: Proc. of the 2018 IEEE Int'l Conf. on Software Maintenance and Evolution (ICSME). 2018. 287−298. [doi: 10.1109/ICSME.2018.00038]

[69] Zhang Y, Lo D, Xia X, Jiang J, Sun JL. Recommending frequently encountered bugs. In: Proc. of the 26th Conf. on Program Comprehension (ICPC). New York: ACM Press, 2018. 120−131.

[70] Digiuseppe N, Jones JA. Fault interaction and its repercussions. In: Proc. of the 27th IEEE Int'l Conf. on Software Maintenance (ICSM). Williamsburg, 2011. 3−12. [doi: 10.1109/ICSM.2011.6080767]

[71] Deng F, Jones JA. Inferred dependence coverage to support fault contextualization. In: Proc. of the 26th IEEE/ACM Int'l Conf. on Automated Software Engineering (ASE). Lawrence, 2011. 512−515. [doi: 10.1109/ASE.2011.6100112]

[72] Wang TT, Su XH, Ma PJ, *et al*. Comprehension oriented software fault location. In: Proc. of the 2011 Int'l Conf. on Computer Science and Network Technology. Harbin, 2011. 340−343. [doi: 10.1109/ICCSNT.2011.6181971]

[73] Sun J, Pham LH, Thi LT, Wang JY, Peng X. Learning likely invariants to explain why a program fails. In: Proc. of the 22nd Int'l Conf. on Engineering of Complex Computer Systems. 2017. 70−79. [doi: 10.1109/ICECCS.2017.12]

[74] Banerjee A, Roychoudhury A, Harlie JA, Liang Z. Golden implementation driven software debugging. In: Proc. of the 18th ACM SIGSOFT Int'l Symp. on Foundations of Software Engineering (FSE). Santa Fe, 2010. 177−186.

[75] Nie C, Zeng D, Zheng X, Wang F, Zhao H. Modeling open source software bugs with complex networks. In: Proc. of the 2010 IEEE Int'l Conf. on Service Operations and Logistics, and Informatics. Qingdao, 2010. 375−379. [doi: 10.1109/SOLI.2010.5551550]

[76] Debroy V, Wong WE. Insights on fault interference for programs with multiple bugs. In: Proc. of the 20th Int'l Symp. on Software Reliability Engineering. Mysuru, 2009. 165−174. [doi: 10.1109/ISSRE.2009.14]

[77] Cheng H, Lo D, Zhou Y, Wang XY, Yan XF. Identifying bug signatures using discriminative graph mining. In: Proc. of the 18th Int'l Symp. on Software Testing and Analysis (ISSTA). New York: ACM Press, 2009. 141−152.

[78] Leonardo Mariani FP. A toolset for automated failure analysis. In: Proc. of the 31st IEEE Int'l Conf. on Software Engineering. Vancouver, 2009. 563−566. [doi: 10.1109/ICSE.2009.5070556]

[79] Hsu HY, Jones JA, Orso A. Rapid: Identifying bug signatures to support debugging activities. In: Proc. of the 23rd IEEE/ACM Int'l Conf. on Automated Software Engineering (ASE). Washington: IEEE Computer Society, 2008. 439−442.

[80] Xiong YF, Liu XY, Zeng MH, Zhang L, Huang G. Identifying patch correctness in test-based program repair. In: Proc. of the 40th Int'l Conf. on Software Engineering. New York: ACM Press, 2018. 789−799.

[81] Ji T, Pan J, Chen L, Mao X. Identifying supplementary bug-fix commits. In: Proc. of the 42nd IEEE Annual Computer Software and Applications Conf. (COMPSAC). Tokyo, 2018. 184−193.

[82] Yan XB, Liu B, Li JX. The failure behaviors of multi-faults programs: An empirical study. In: Proc. of the 2017 IEEE Int'l Conf. on Software Quality, Reliability and Security Companion. IEEE, 2017. 320−330. [doi: 10.1109/QRS-C.2017.11]

[83] Baltes S, Moseler O, Beck F, Diehl S. Navigate, understand, communicate: How developers locate performance bugs. In: Proc. of the 2015 ACM/IEEE Int'l Symp. on Empirical Software Engineering and Measurement (ESEM). Beijing, 2015. 1−10. [doi: 10.1109/ESEM.2015.7321208]

[84] Dao T, Zhang L, Meng N. How does execution information help with information-retrieval based bug localization? In: Proc. of the 25th IEEE/ACM Int'l Conf. on Program Comprehension (ICPC). Buenos Aires, 2017. 241−250. [doi: 10.1109/ICPC.2017.29]

[85] Zhou C, Li B, Sun XB, Guo HJ. Recognizing software bug-specific named entity in software bug repository. In: Proc. of the 26th Conf. on Program Comprehension (ICPC). New York: ACM Press, 2018. 108−119.

[86] Ang A, Perez A, Deursen AV, Abreu R. Revisiting the practical use of automated software fault localization techniques. In: Proc. of the 2017 IEEE Int'l Symp. on Software Reliability Engineering Workshops (ISSREW). Toulouse, 2017. 175−182. [doi: 10.1109/ISSREW.2017.68]

[87] Xu ZG, Ma SQ, Zhang XY, Zhu SF, Xu BW. Debugging with intelligence via probabilistic inference. In: Proc. of the 40th Int'l Conf. on Software Engineering (ICSE). New York: ACM Press, 2018. 1171−1181.

[88] Baah GK, Podgurski A, Harrold MJ. The probabilistic program dependence graph and its application to fault diagnosis. IEEE Trans. on Software Engineering, 2010,36(4):528−545.

[89] Shin D, Bae DH. A theoretical framework for understanding mutation-based testing methods. In: Proc. of the IEEE Int'l Conf. on Software Testing. IEEE, 2016. 299−308. [doi: 10.1109/ICST.2016.22]

[90] Rodríguez-Pérez G, Zaidman A, Serebrenik A, Robles G, González-Barahona JM. What if a bug has a different origin? Making sense of bugs without an explicit bug introducing change. In: Proc. of the 12th ACM/IEEE Int'l Symp. on Empirical Software Engineering and Measurement. New York: ACM Press, 2018.

[91] Chen R, Chen S, Zhang N. Iterative path clustering for software fault localization. In: Proc. of the IEEE Int'l Conf. on Software Quality, Reliability and Security Companion. IEEE, 2016. 292−297. [doi: 10.1109/QRS-C.2016.85]

[92] Liu M, Liu P, Yang X, Zhao L. Fault localization guided execution comparison for failure comprehension. In: Proc. of the IEEE Int'l Conf. on Software Quality, Reliability and Security Companion. IEEE, 2016. 163−166. [doi: 10.1109/QRS-C.2016.87]

[93] Tu JX, Xie XY, Zhou YM, Xu BW, Chen L. A search based context-aware approach for understanding and localizing the fault via weighted call graph. In: Proc. of the 3rd Int'l Conf. on Trustworthy Systems and their Applications (TSA). Wuhan, 2016. 64−72. [doi: 10.1109/TSA.2016.20]

[94] Wang Y, Huang ZQ. Weighted control flow subgraph to support debugging activities. In: Proc. of the 2016 IEEE Int'l Conf. on Software Quality, Reliability and Security Companion. Vienna, 2016. 131−134. [doi: 10.1109/QRS-C.2016.21]

[95] Yi QP, Yang ZJ, Liu J, Zhao C, Wang C. Explaining software failures by cascade fault localization. ACM Trans. on Design Automation of Electronic Systems, 2015,20(3):1−28. [doi: 10.1145/2738038]

[96] Xie XY, Liu ZC, Song S, Chen ZY, Xuan JF, Xu BW. Revisit of automatic debugging via human focus-tracking analysis. In: Proc. of the 38th IEEE/ACM Int'l Conf. on Software Engineering (ICSE). Austin, 2016. 808−819.

[97] Le TDB, Linares-Vasquez M, Lo D, Poshyvanyk D. RCLinker: Automated linking of issue reports and commits leveraging rich contextual information. In: Proc. of the 23rd IEEE Int'l Conf. on Program Comprehension. IEEE, 2015. 36−47.

[98] Jonsson L, Broman D, Magnusson M, Sandahl K, Villani M, Eldh S. Automatic localization of bugs to faulty components in large scale software systems using bayesian classification. In: Proc. of the 2016 IEEE Int'l Conf. on Software Quality, Reliability and Security (QRS). IEEE, 2016. 423−430. [doi: 10.1109/QRS.2016.54]

[99] Wang XY, Liu YM. Fault localization using disparities of dynamic invariants. Journal of Systems & Software, 2016,122:144−154. [doi: 10.1016/j.jss.2016.09.014]

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

[101] Jiang L, Su Z. Context-aware statistical debugging: From bug predictors to faulty control flow paths. In: Proc. of the IEEE/ACM Int'l Conf. on Automated Software Engineering. New York: ACM Press, 2007. 184−193.

[102] He H, Gupta N. Automated debugging using path-based weakest preconditions. In: Wermelinger M., Margaria-Steffen T, eds. Proc. of the Fundamental Approaches to Software Engineering. LNCS 2984, Berlin, Heidelberg: Springer-Verlag, 2004. 267−280.

[103] Cleve H. Locating causes of program failures. In: Proc. of the 27th Int'l Conf. on Software Engineering (ICSE). New York: ACM Press, 2005. 342−351.

[104] Feng M, Gupta R. Learning universal probabilistic models for fault localization. In: Proc. of the ACM SIGPLAN-SIGSOFT Workshop on Program Analysis for Software Tools & Engineering. New York: ACM Press, 2010. 81−88.

[105] Parsa S, Naree SA, Koopaei NE. Software fault localization via mining execution graphs. In: Murgante B, Gervasi O, Iglesias A, Taniar D, Apduhan BO, eds. Proc. of the Computational Science and Its Applications (ICCSA 2011). LNCS 6783, Berlin, Heidelberg: Springer-Verlag, 2011. 610−623.

[106] Jones JA, Harrold MJ. Empirical evaluation of the tarantula automatic fault-localization technique. In: Proc. of the IEEE/ACM Int'l Conf. on Automated Software Engineering. New York: ACM Press, 2005. 273−282.

[107] Zuo Z. Efficient statistical debugging via hierarchical instrumentation. In: Proc. of the Int'l Symp. on Software Testing & Analysis. New York: ACM Press, 2014. 457−460.

[108] Nipkow T. Isabelle/Hol a Proof Assistant for Higher-Order Logic. Berlin, Heidelberg: Springer-Verlag, 2002. 53−64.

[109] Cruz J. Constraint reasoning for differential models. In: Cruz J, ed. Proc. of the 2005 Conf. on Constraint Reasoning for Differential Models. Amsterdam: IOS Press, 2005. 1−216.

[110] Clarke EM, Grumberg O, Long DE. Model checking and abstraction. In: Proc. of the NATO Advanced Study Institute on Deductive Program Design. Marktoberdorf, 1992.

[111] Yang K, He YP, Ma HT, Wang XF. Precise execution reachability analysis: Theory and application. Ruan Jian Xue Bao/Journal of Software, 2018,29(1):1−22 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5375.htm [doi: 10.13328/j.cnki.jos.005375]

[112] Dijkstra EW. A Discipline of Programming. Upper Saddle River: Prentice Hall PTR, 1997. 10−23.

[113] Qi D, Roychoudhury A, Liang Z, Vaswani K. Darwin: An approach for debugging evolving programs. ACM Trans. on Software Engineering and Methodology, 2012.

[114] Nan YH, Yang M, Yang ZM, *et al*. UIPicker: User-input privacy identification in mobile applications. In: Jung J, ed. Proc. of the 24th USENIX Conf. on Security Symp. (SEC). Berkeley: USENIX Association, 2015. 993−1008.

[115] Han X, Yu TT, Lo D. PerfLearner: Learning from bug reports to understand and generate performance test frames. In: Proc. of the 33rd ACM/IEEE Int'l Conf. on Automated Software Engineering. New York: ACM Press, 2018. 17−28.

[116] Jiang S, Armaly A, Mcmillan C. Automatically generating commit messages from diffs using neural machine translation. In: Proc. of the 32nd IEEE/ACM Int'l Conf. on Automated Software Engineering (ASE). ACM Press, 2017. 135−146.

[117] Palomba F, Panichella A, Zaidman A, Oliveto R, Lucia AD. The scent of a smell: An extensive comparison between textual and structural smells. IEEE Trans. on Software Engineering, 2017,44(10):977−1000. [doi: 10.1109/TSE.2017.2752171]

[118] Zhou Y, Gu RH, Cheny T, *et al*. Analyzing APIs documentation and code to detect directive defects. In: Proc. of the IEEE/ACM Int'l Conf. on Software Engineering. IEEE, 2017. 27−37.

[119] Thung F, Le XD, Lo D, Lawall J. Recommending code changes for automatic backporting of linux device drivers. In: Proc. of the 2016 IEEE Int'l Conf. on Software Maintenance and Evolution (ICSME). Raleigh, 2016. 222−232.

[120] Renieres M, Reiss SP. Fault localization with nearest neighbor queries. In: Proc. of the IEEE Int'l Conf. on Automated Software Engineering. IEEE, 2003. 30−39.

[121] Jones JA, Harrold MJ, Stasko J. Visualization of test information to assist fault localization. In: Proc. of the 24th Int'l Conf. on Software Engineering. ACM Press, 2002. 467−477.

[122] Wong WE, Wei TT, Qi Y, Zhao L. A crosstab-based statistical method for effective fault localization. In: Proc. of the Int'l Conf. on Software Testing. IEEE, 2008. 42−51.

[123] Kleer JD. Diagnosing multiple persistent and intermittent faults. In: Kitano H, ed. Proc. of the 21st Int'l Jont Conf. on Artifical Intelligence (IJCAI). San Francisco: Morgan Kaufmann Publishers Inc., 2009. 733−738.

[124] Abreu R, Cardoso N. A kernel density estimate-based approach to component goodness modeling. In: Proc. of the 27th AAAI Conf. on Artificial Intelligence. AAAI Press, 2013. 152−158.

[125] Jones JA, Bowring JF, Harrold MJ. Debugging in parallel. In: Proc. of the 2007 Int'l Symp. on Software Testing and Analysis (ISSTA). New York: ACM Press, 2007. 16−26.

[126] Zheng Y, Wang Z, Fan XY, Chen X, Yang ZJ. Localizing multiple software faults based on evolution algorithm. Journal of Systems and Software, 2018,(139):107−123.

[127] Gao R, Wong WE. Mseer—An advanced technique for locating multiple bugs in parallel. IEEE Trans. on Software Engineering, 2017. 1−21. [doi: 10.1109/TSE.2017.2776912]

[128] Shimomura T, Ikeda K. Two types of deadlock detection: Cyclic and acyclic. In: Chen L, Kapoor S, Bhatia R, eds. Proc. of the Intelligent Systems for Science and Information, Studies in Computational Intelligence, Vol.542. Cham: Springer-Verlag, 2014. 233−259.

[129] Naik M, Aiken A, Whaley J. Effective static race detection for Java. In: Proc. of the 27th ACM SIGPLAN Conf. on Programming Language Design and Implementation (PLDI). New York: ACM Press, 2006. 308−319.

[130] Engler D, Ashcraft K. RacerX: Effective, static detection of race conditions and deadlocks. ACM SIGOPS Operating Systems Review, 2003,37(5):237−252.

[131] Cai Y, Cao L. Fixing deadlocks via lock pre-acquisitions. In: Proc. of the 38th IEEE/ACM Int'l Conf. on Software Engineering (ICSE). Austin, 2016. 1109−1120.

[132] Nir D, Tyszberowicz S, Yehudai A. Locating regression bugs. In: Yorav K, ed. Proc. of the 3rd Int'l Haifa Verification Conf. on Hardware and Software: Verification and Testing (HVC). Berlin, Heidelberg: Springer-Verlag, 2007. 218−234.

[133] Liu ML, Yang XS, Zhao L, Wang LN. Discrete characteristic-based test execution selection for software fault localization and understanding. Computer Science, 2016,43(3):179−187 (in Chinese with English abstract). [doi: 10.11896/j.issn.1002-137X.2016.3.034]

[134] Yu Y, Jones J, Harrold MJ. An empirical study of the effects of test-suite reduction on fault localization. In: Proc. of the 30th ACM/ IEEE Int'l Conf. on Software Engineering. Leipzig, 2008. 201−210.

[135] Carmine V, Sebastian P, Timothy Z, Gall HC. Un-break my build: Assisting developers with build repair hints. In: Proc. of the 26th IEEE/ACM Int'l Conf. on Program Comprehension. 2018. 41−51.

[136] Lawall J, Palinski D, Gnirke L, Muller G. Fast and precise retrieval of forward and back porting information for Linux device drivers. In: Proc. of the Usenix Conf. on Usenix Technical Conf. USENIX Association, 2017. 15−26.

[137] Xiao Y, Keung J, Mi Q, Bennin KE. Bug localization with semantic and structural features using convolutional neural network and cascade forest. In: Proc. of the 22nd Int'l Conf. on Evaluation and Assessment in Software Engineering. New York: ACM Press, 2018. 101−111.

[138] Böhme M, Pham VT, Nguyen MD, Roychoudhury A. Directed greybox fuzzing. In: Proc. of the 2017 ACM SIGSAC Conf. on Computer and Communications Security. ACM Press, 2017. 2329−2344.

[139] Chen H, Xue Y, Li Y, Chen B, Xie X, Wu X, Liu Y. Hawkeye: Towards a desired directed grey-box fuzzer. In: Proc. of the 2018 ACM SIGSAC Conf. on Computer and Communications Security. ACM Press, 2018. 2095−2108.

[140] Mills C. Towards the automatic classification of traceability links. In: Proc. of the IEEE/ACM Int'l Conf. on Automated Software Engineering. Urbana, 2017. 1018−1021. [doi: 10.1109/ASE.2017.8115723]

[141] Rempel P, Mader P. Preventing defects: The impact of requirements traceability completeness on software quality. IEEE Trans. on Software Engineering, 2017,43(8):777−797. [doi: 10.1109/TSE.2016.2622264]

[142] Li B, He YP, Ma HT. Automatic program repair: Key problems and technologies. Ruan Jian Xue Bao/Journal of Software, 2019, 30(2):244−265 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5657.htm [doi: 10.13328/j.cnki.jos.005657]

[143] Mens T, Claes M, Grosjean P, Serebrenik A. Studying evolving software ecosystems based on ecological models. In: Mens T, Serebrenik A, Cleve A, eds. Proc. of the Evolving Software Systems. Berlin, Heidelberg: Springer-Verlag, 2014. 297−326.

[144] Hoving R, Slot G, Jansen S. Python: Characteristics identification of a free open source software ecosystem. In: Proc. of the IEEE Int'l Conf. on Digital Ecosystems & Technologies. Menlo Park, 2013. 13−18. [doi: 10.1109/DEST.2013.6611322]

附中文参考文献:

[2] 梅宏,王千祥,张路,王戟.软件分析技术进展.计算机学报,2009,32(9):1697−1710. [doi: 10.11897/SP.J.1016.2015.02203]

[3] 王克朝,王甜甜,苏小红,马培军.软件错误自动定位关键科学问题及研究进展.计算机学报,2015,38(11):2262−2278. [doi: 10.11897/SP.J.1016.2015.02262]

[19] 虞凯,林梦香.自动化软件错误定位技术研究进展.计算机学报,2011,34(8):1411−1422. [doi: 10.3724/SP.J.1016.2011.01411]

[24] 金芝,刘芳,李戈.程序理解:现状与未来.软件学报,2019,30(1):110−126. http://www.jos.org.cn/1000-9825/5643.htm [doi: 10.13328/j.cnki.jos.005643]

[30] 田硕,梁洪亮.二进制程序安全缺陷静态分析方法的研究综述.计算机科学,2009,36(7):8−14.

[33] 张健,张超,玄跻峰,熊英飞,王千祥,梁彬,李炼,窦文生,陈振邦,陈立前,蔡彦.程序分析研究进展.软件学报,2019,30(1):80−109. http://www.jos.org.cn/1000-9825/5651.htm [doi: 10.13328/j.cnki.jos.005651]

[111] 杨克,贺也平,马恒太,王雪飞.精准执行可达性分析:理论与应用.软件学报,2018,29(1):1−22. http://www.jos.org.cn/1000-9825/5375.htm [doi: 10.13328/j.cnki.jos.005375]

[133] 刘梦冷,杨小双,赵磊,王丽娜.面向软件错误定位与理解的测试执行离散特征筛选.计算机科学,2016,43(3):179−187. [doi: 10.11896/j.issn.1002-137X.2016.3.034]

[142] 李斌,贺也平,马恒太.程序自动修复:关键问题及技术.软件学报,2019,30(2):244−265. http://www.jos.org.cn/1000-9825/5657.htm [doi: 10.13328/j.cnki.jos.005657]

李晓卓(1992－),女,北京人,硕士,主要研究领域为程序分析,软件缺陷定位.

马恒太(1970－),男,博士,副研究员,主要研究领域为软件安全分析,操作系统安全.

贺也平(1962－),男,博士,研究员,博士生导师,主要研究领域为系统安全,隐私保护.