软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2016,27(4):771−784 [doi: 10.13328/j.cnki.jos.004972]

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

自动程序修复方法研究进展*

玄跻峰[1,2], 任志磊[3], 王子元[4], 谢晓园[1,2], 江 贺[3]

[1](软件工程国家重点实验室(武汉大学),湖北 武汉 430072)

[2](武汉大学 计算机学院,湖北 武汉 430072)

[3](大连理工大学 软件学院,辽宁 大连 116621)

[4](南京邮电大学 计算机学院,江苏 南京 210006)

通讯作者: 玄跻峰, E-mail: jxuan@whu.edu.cn

摘 要: 自动程序修复帮助开发者降低人工修复 bug 的成本.基于测试集的修复方法旨在生成能够通过测试集的代码补丁,以使程序正常运行.回顾了基于测试集的程序修复的现有文献,按照自动修复方法和实证基础两个方面陈述了研究进展.首先,将已有的自动修复方法划分为 3 类,分别是基于搜索的、基于代码穷举的和基于约束求解的补丁生成方法;其次,细致地描述了程序修复的实证研究基础以及该研究领域中的争议;然后,简要介绍了程序修复的相关技术作为修复方法的补充;最后做出总结,描述了面临的机遇和挑战.

关键词: 自动修复;遗传规划;基于搜索的软件工程;测试集;实证基础

中图法分类号: TP311

中文引用格式: 玄跻峰,任志磊,王子元,谢晓园,江贺.自动程序修复方法研究进展.软件学报,2016,27(4):771−784. http://www.jos.org.cn/1000-9825/4972.htm

英文引用格式: Xuan JF, Ren ZL, Wang ZY, Xie XY, Jiang H. Progress on approaches to automatic program repair. Ruan Jian Xue Bao/Journal of Software, 2016,27(4):771−784 (in Chinese). http://www.jos.org.cn/1000-9825/4972.htm

Progress on Approaches to Automatic Program Repair

XUAN Ji-Feng[1,2], REN Zhi-Lei[3], WANG Zi-Yuan[4], XIE Xiao-Yuan[1,2], JIANG He[3]

[1](State Key Laboratory of Software Engineering (Wuhan University), Wuhan 430072, China)

[2](Computer School, Wuhan University, Wuhan 430072, China)

[3](School of Software, Dalian University of Technology, Dalian 116621, China)

[4](School of Computer Science and Technology, Nanjing University of Posts and Telecommunications, Nanjing 210006, China)

Abstract: Automatic program repair helps developers reduce the cost of manual bug fixing. Approaches to test-suite based repair aim to generate code patches to pass the test suite as well as maintain the program execution. This paper reviews available literature on test-suite based repair and report the progress in two directions: Approaches to automatic repair and empirical foundations. First, existing approaches to automatic repair are described in three categories: Search based, exhaustion based, and constraint-solving based patch generation. Second, empirical foundations on repair are detailed, including the argumentation in the research field. Related techniques are then briefly introduced as the supplementation of program repair. Finally, opportunities and challenges are presented to summarize this review.

Key words: automatic repair; genetic programming; search based software engineering; test suite; empirical foundation

程序 bug 是软件开发中不可避免的产物,其产生原因可以追溯到软件开发的各个阶段.历史数据表明,超过

* 基金项目: 国家自然科学基金(61502345, 61403057, 61370144, 61202032)
Foundation item: National Natural Science Foundation of China (61502345, 61403057, 61370144, 61202032)
收稿时间: 2015-09-02; 修改时间: 2015-10-15; 采用时间: 2015-11-20; jos 在线出版时间: 2016-01-13
CNKI 网络优先出版: 2016-01-14 13:15:55, http://www.cnki.net/kcms/detail/11.2560.TP.20160114.1315.001.html

45%的现代软件开发成本消耗于定位和修复 bug 过程中[1].无论工业生产还是学术研究领域,定位和修复程序 bug 都是软件工程的核心问题.随着软件测试和调试技术的提高,自动化的程序 bug 定位技术已经得到了一定的研究和发展[2,3],然而自动化的程序 bug 修复方法仍处在初级阶段.随着程序 bug 数量的日益积累,自动修复技术的深入研究及应用已迫在眉睫.据已有数据显示:在 2001 年~2010 年这 10 年间,开源软件项目 Eclipse 共接受了 333 371 个提交的 bug,平均每天新增 76 个 bug[4].为了提高软件质量并改进软件产品,开发者不得不耗费大量的人力和资源修复软件中的问题.

为了降低修复过程中的时间和人力成本,自动程序修复(automatic program repair)方法应运而生.该方法依据给定的程序问题,自动生成程序补丁(patch),进而修复程序中的错误.修复中产生的程序补丁既可以自动添加到程序中,也可以用于指导开发者继续改进代码.与现代软件工业中广泛出现的敏捷开发(agile development)和持续集成(continuous integration)相结合,自动程序修复方法将有效地降低程序修复的成本.从技术角度来看,研究者将程序修复看作从潜在的搜索空间(search space)中搜索补丁的过程,而优秀的修复技术可以大幅度提高补丁搜索的准确率,并减少用于搜索的时间消耗[5,6].该研究思路与基于搜索的软件工程契合[7].例如,自动程序修复中的先驱方法 GenProg 就是一种基于遗传规划(genetic programming)的 C 程序修复算法[8].

由于程序自身的复杂性,自动程序修复的研究面临着一系列的挑战.

- 一方面,研究者尝试设计算法为程序自动生成补丁,而所生成的补丁与真实人工添加的补丁尚存较大差别;
- 另一方面,研究者正在探索自动修复的实证基础(empirical foundation),然而该研究并非一帆风顺,曾在 2014 年[9]和 2015 年[10,11]掀起了两次领域内的学术争论.

为避免歧义,若无特别说明,本文的研究主题(即自动程序修复)专指基于测试集的修复(test-suite based repair)方法[9].在基于测试集的修复中,测试集(test suite)用于描述并限定程序的正确行为(program behavior).任何自动生成的补丁应至少保障测试集正确运行(passing the test suite),以确认未违反程序的需求说明.自动程序修复研究中的绝大多数成果都属于基于测试集的修复领域.

截至本文成文之际(2015 年 8 月 31 日),自动程序修复方法尚未有工业应用.本文回顾了该领域的研究历史,从修复方法和实证研究基础两个主要方面详述程序修复的现有研究进展以及所面临的机遇和挑战.本文选取的文献试图覆盖自动程序修复领域自 2008 年以来累计 8 年的主流工作.需要提及的是:虽然该领域的国内研究相对于国际同行较为滞后,但来自国防科学技术大学[12−16]和上海交通大学[17]的研究者们已发表了不少优秀成果,将在后文加以介绍.

本文首先概述自动程序修复的研究背景;其次,通过检索和筛选相关文献简要地回顾研究历史;再次,分两个方面介绍本文的主要部分:自动修复方法的研究进展和程序修复的实证研究基础;然后,作为补充,简要介绍与程序修复直接相关的技术;最后,分析自动程序修复研究面临的机遇和挑战.

1 概 述

1.1 基于测试集的程序修复

基于测试集的程序修复,简称为程序修复,旨在自动生成能够通过测试集中全部测试用例(test case)的程序补丁.该类方法主要以白盒(white-box)形式呈现,即,被修复程序的源代码和测试用例对自动修复方法完全可见.一个程序修复方法的输入通常是带有 bug 的程序(buggy program)及其当前的测试集,并且至少有一个测试用例因为 bug 而无法通过(fail).修复方法的输出为零个或一个程序补丁(有的方法可以输出多个补丁).生成零个补丁表示该方法无法自动修复程序.

处理程序并非易事:程序的需求信息(requirements)通常以自然语言描述,如项目文档或程序接口文档,这些信息无法被直接翻译为程序输入.另一方面,程序形式化的规格说明(specifications)也难以充分获取.因此,源代码结构和测试集中的测试用例成为修复程序的主要输入来源.获得可修复 bug 的代码补丁,实际上就是生成在程序特定位置的代码段.而这个特定位置和代码段的内容都是未知的.直观来说,程序修复包含两个部分,即如

何找到 bug 的位置和如何为 bug 生成代码段.为了解决程序修复问题,研究者们将这个带有 bug 的程序及其某个潜在位置的代码变动(例如代码的添加、删除或修改)看作一个个体,并将所有这样的个体看作一个巨大的搜索空间[5].因此,程序修复的过程可以看作如何从搜索空间中找寻可以修复 bug 的个体的过程.

1.2 修复算法的流程

图 1 展示了基于测试集的程序修复方法的流程.程序修复方法首先通过故障定位(fault localization)方法将给定带有 bug 的程序的所有语句按某种规则排序,得到可疑语句的排列;然后将这些语句作为疑似 bug 的位置,逐个传输给补丁生成算法(patch generation);补丁生成算法会检测当前 bug 的位置,决定是否能够输出补丁:如果能够,则输出补丁给测试集进行验证,通过验证(即是否能够通过测试集)的补丁将作为结果输出;若该疑似 bug 的位置无法输出补丁,则从语句排列中获取下一个可疑语句,重新运行补丁生成算法以继续寻找潜在的补丁.

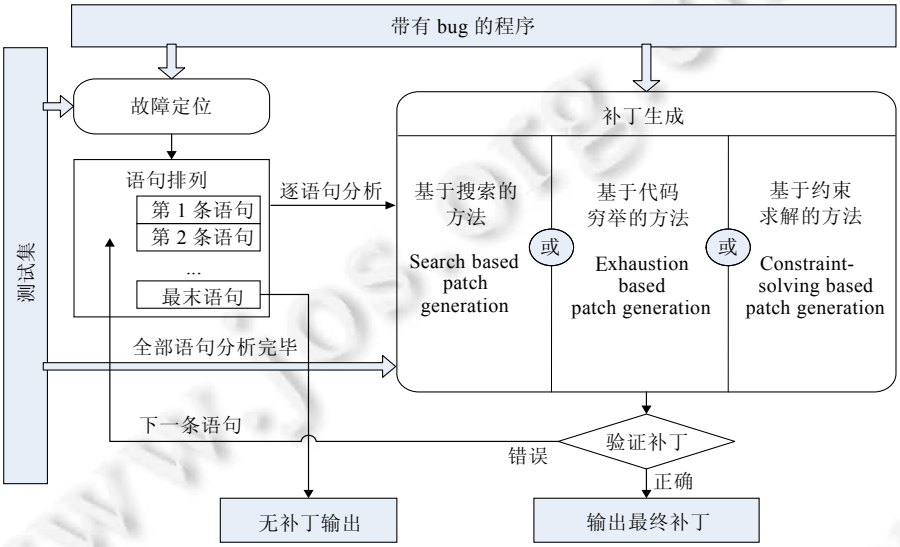


Fig.1 Overview of test-suite based repair

图 1 基于测试集的程序修复方法的流程概览

如图 1 所示,本文将所有的相关补丁生成方法粗略地划分为 3 类,即基于搜索(search based patch generation)的方法、基于代码穷举(exhaustion based patch generation)的方法和基于约束求解(constraint-solving based patch generation)的方法.基于搜索的方法使用源自基于搜索的软件工程[7,18,19]中“搜索”的定义,即,采用元启发式(meta-heuristics)方法、演化算法(evolutionary computation)等优化方法进行补丁的搜索.基于搜索的方法是程序修复中的主要部分,尤其在领域创始之初更占有重要地位,代表算法包括基于遗传规划的抽象语法树修复算法 GenProg[5]、基于程序等价性的遗传修复算法 AE[20]、基于随机搜索的修复算法 RSRepair[16]等.基于代码穷举的方法无差别地变异全部可疑语句,同时,有策略地穷举可能出现的代码修改,追求补丁的有效性而忽略算法效率.这类算法不多,代表算法包括程序变异算法[21]、代码消除算法 Kali[10].基于约束求解的方法,顾名思义,将补丁生成转换为约束求解问题,应用求解器直接计算可行解(feasible solution),进而转换为最终补丁,代表算法包括程序语义修复 SemFix[22]、补丁简化算法 DirectFix[23]、条件 bug 修复 Nopol[24,25]等.

需要注意的是:其一,基于代码穷举的方法和基于约束求解的方法也可以看作某种程度的搜索,但本文遵从基于搜索的软件工程领域的约定,只将第 1 类方法称为基于搜索的补丁生成方法;其二,图 1 的流程是经过总结已有方法而得到的大概流程[24,26].在一些修复算法中,某些步骤可能被简化.例如:在基于代码穷举的方法 Kali[10]中,所有的程序语句没有经过故障定位,而按照自然顺序直接排列;而在基于约束求解的方法 SemFix[22]中,在补丁生成后无需进行补丁的验证.第 3 节详细介绍修复方法.

1.3 测试集在程序修复中的作用

测试集是自动修复算法的重要输入.事实上,测试集是在实践上程序需求说明的代码表达.由于需求说明往往不可见,测试集中的测试用例可以制约程序的实际行为.在自动修复算法中,测试集指导补丁生成,同时验证生成后的补丁是否通过测试集.

度量基于测试集的修复算法,其基本要求是添加补丁的程序能够通过测试集中的全部测试用例.该度量标准被称为可修复性(fixability).现代软件开发中,测试集可以通过测试框架(如 Java 语言的 JUnit)自动运行.因此,可修复性能否达到是可以自动完成的.

能够通过测试集的补丁并不一定正确.正确性(correctness)指的是修复后的程序达到了预期的行为,即,程序的输出能够满足潜在的测试预言(test oracle).例如,一个修复划分三角形类别程序的补丁正确,指的是程序可以无误地划分任何潜在的三角形类别,而不是仅仅满足有限数量的测试用例的通过.正确性目前还不能通过自动技术完成,只能手动验证.近期的一些工作采用了正确性作为评价标准[10,25,26].

2 研究历史回顾

2.1 文献检索与筛选

本文的文献囊括了基于测试集的程序修复领域 8 年来的研究进展,同时包含了其他程序修复相关技术的主要文献.以下是本文检索和筛选文献的步骤.

(1) 文献检索的主要途径是谷歌学术搜索引擎(Google scholar)以及 ACM(ACM digital library)、IEEE (IEEE xplore digital library)和 Springer(Springer link)的论文数据库.检索过程中倾向于软件工程的主流会议期刊(如 ICSE,FSE 等),并包括系统和程序语言相关会议期刊(如 PLDI,OOPSLA 等);

(2) 检索关键词包括自动程序修复的关键词的组合,可简要概括如下(⊥表示空字符串):
(repair∨fix)∧(software∨program∨code)∧(failure∨bug∨test case∨test suite∨⊥);

(3) 检索时间为 2001 年至今的全部文献.由于基于测试集的修复领域始于 2008 年,因此,该检索将覆盖基于测试集的程序修复方法相关的全部文献;同时,我们也列出了其他程序修复方法的主要相关文献;

(4) 收入本文的文献类型包括期刊的长文和短文、会议的 research track,experience track,industry track, new idea and emerging result track 和 short paper track 以及技术报告(technical report).技术报告是该领域文献的重要组成部分,由于该领域发展活跃,部分论文尚未进入发表阶段,常以技术报告的形式呈现.但文献不包含会议的 poster track,doctoral symposium,student competition,原因是这些部分的正文较短,不足以了解文章内容;同时也不包括博士论文(Ph.D. dissertation),因为所涉及的博士论文的内容大部分已经以会议或期刊论文的形式发表;

(5) 基于上面检索到的论文,我们进行了人工筛选,最后获得第 2.2 节列出的文献.

2.2 历史文献回顾

经过文献检索与筛选,获得了基于测试集的程序修复领域自 2008 年~2015 年的学术论文 37 篇,其中会议长文 25 篇、会议短文 2 篇、期刊 6 篇、技术报告 4 篇.需要指出的是:这里仅包含本文主题,即,基于测试集的程序修复,这 37 篇论文将在第 3 节和第 4 节进行详细介绍,而与程序修复相关的其他论文将在第 5 节介绍.

表 1 列出了基于测试集的程序修复的相关文献的出处,其中包括软件工程顶级会议 ICSE 论文 8 篇、FSE 论文 4 篇、顶级期刊 IEEE Trans. on Software Engineering(TSE)论文 2 篇.

图 2 展示了自 2008 年创始以来该领域相关文献的分布,可以看到:文献数量缓步上升,而最近两年的文献数量最多,2014 年 8 篇,而 2015 年已有 11 篇.这一趋势说明,该领域正日趋活跃.另外,2011 年没有任何文献出现,暂时没有发现特殊原因.

Table 1 List of related literature on test-suite based repair

表 1 基于测试集的程序修复的相关文献的列表

类型	出版物名称	缩写	文章数	文章列表
国际会议	Int'l Conf. on Software Engineering	ICSE	8	[6,8,9,16,17,23,42,47]
	ACM SIGSOFT Int'l Symp. on the Foundations of Software Engineering	FSE	4	[11,40,45,48]
	Int'l Symp. on Software Testing and Analysis	ISSTA	3	[10,15,43]
	IEEE/ACM Int'l Conf. on Automated Software Engineering	ASE	3	[20,32,41]
	IEEE Int'l Conf. on Software Maintenance (and Evolution)	ICSM	3	[12,14,46]
	Int'l Conference on Software Testing, Verification and Validation	ICST	1	[21]
	Genetic and Evolutionary Computation Conf.	GECCO	3	[28,30,31]
	Int'l Workshop on Constraints in Software Testing, Verification, and Analysis	CSTVA	1	[24]
	IEEE Congress on Evolutionary Computation	CEC	1	[27]
国际期刊	IEEE Trans. on Software Engineering	TSE	2	[5,49]
	Communications of the ACM	CACM	1	[29]
	Empirical Software Engineering	ESE	1	[33]
	Software Quality Journal	SQJ	1	[44]
	Science China Information Sciences	SCIS	1	[13]
技术报告	Technical Report	TR	4	[25,26,34,50]
合计			37	

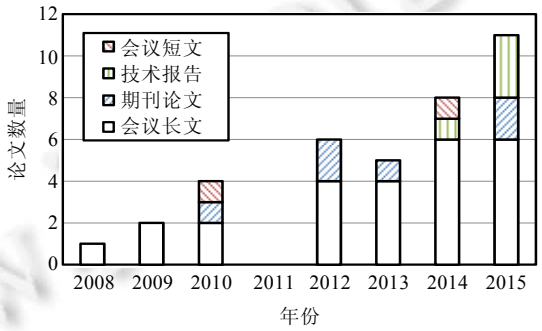


Fig.2 Illustration of literature by year

图 2 相关文献的年代分布

3 基于测试集的自动程序修复方法

正如第 1.2 节所说,本文按照获得补丁的过程将基于测试集的程序修复方法分为 3 类,即,基于搜索的方法、基于代码穷举的方法和基于约束求解的方法.其中,多数已有的修复算法可以认为是基于搜索的方法.

3.1 基于搜索的方法

在基于搜索的方法中,生成程序补丁的过程被看作是从补丁的搜索空间中寻找最优可行解的过程.对于被修复程序来说,潜在补丁的搜索空间是未知的,研究者将更改被测程序而产生的新程序看作潜在的补丁,进而获得补丁空间.基于搜索方法的核心之一,是使用基于搜索的软件工程中的启发式方法、演化算法等方法寻找补丁.

2008 年,Arcuri 和 Yao[27]最早提出了基于测试集的程序修复思路,完成了该领域开创性的工作.他们将被修复程序的补丁看作算法的解,应用协同演化(co-evolutionary)算法自动生成代码补丁.

2009 年,Weimer 和 Le Goues 等人[5,8,28,29]作为该领域的先驱,设计了 GenProg 算法.GenProg 将 C 程序看作抽象语法树(abstract syntax tree),将代码段看作子树;基于遗传规划算法,GenProg 将被测程序的抽象语法树通过插入、删除或替换生成新的程序,并应用测试用例集验证是否被修复.在该方法中,补丁的位置由一种内嵌的故障定位算法决定.遗传规划在 GenProg 中的作用是通过变换语法树获得新的语法树.

2010 年,Weimer 和 Le Goues 等人的后续工作,如 2010 年的 Fast 等人[30]和 2012 年的 Le Goues 等人[31],进一步从遗传规划的优化角度提升获得补丁的效率;而 2010 年,Schulte 等人[32]则应用了同样的方法修复汇编语

言的 bug.

2012 年,Qi 等人[12,13]提出通过弱重编译(weak recompilation)的方法优化 GenProg 的修复效率,能够较早地找到补丁.其后的 2013 年,他们[14]应用故障记录的测试用例排序(fault-recorded test prioritization)方法进一步提高了修复效率.此外,他们[15]还通过程序修复的结果来评估故障定位方法.

2013 年,Kim 等人[6]从人工书写的补丁中学习代码模式,并将模式与 GenProg 相融合生成补丁.该方法生成的补丁可以避免出现通过优化算法而得到的无意义的补丁.Weimer 等人[20]考虑增强发现潜在补丁的能力,提出了基于程序等价性(program equivalence)的遗传修复算法 AE.

2014 年,Qi 等人[16]通过研究发现,GenProg 算法中的遗传规划算法对于高效生成补丁并不奏效.他们用随机搜索(random search)替换了遗传算法,并设计了 RSRepair 用于程序修复.实验结果表明:相对于 GenProg 方法,RSRepair 能够减少生成补丁的时间消耗.

2015 年,Martinez 和 Monperrus[33]通过挖掘程序修复的历史数据,建立了概率模型预测新 bug 的修复.超过 6 万个 Java 语言的代码变更提交(code commit)被转换为抽象语法树,用于修复概率模型的建立.同是 2015 年,Long 和 Rinard[34]提出了 Prophet,一种学习现有补丁以指导未来补丁排序的算法.它通过最大似然(maximum likelihood)模型识别最可能成功的补丁的概率,其算法本质是补丁排序的过程.该方法可以修复 69 个真实 bug 中的 15 个,具有一定的准确度.

3.2 基于代码穷举的方法

基于代码穷举的方法通过策略穷举可能的代码修改,进而获得大量的潜在代码补丁.该类算法侧重于深入研究算法对代码的修复能力,追求补丁的有效性而忽略执行效率.相关算法数量不多.

2010 年,Debroy 和 Wong[21]提出了基于程序变异的修复算法.程序变异(program mutation)[35]源自变异测试(mutation testing),将程序进行一个小修改,如更改操作符或数值增加 1 等,进而得到修改后的程序.该方法应用变异获得可能的程序,将其作为修复后的程序并检验补丁的修复效果.

2015 年,Qi 等人[10]提出 Kali,一种代码消除算法.该方法仅通过简单的删除语句、修改条件真值和提前返回程序等步骤,获得消除一定代码后的程序版本,用于检验补丁的修复效果.该方法的设计并非为了修复 bug,而是为了研究其他修复算法的潜在风险.同年,Long 和 Rinard[36]提出了阶段条件合成算法 SPR,该算法定义了分阶段出现的修复参数,能够在早期阶段丢弃不能通过测试用例的补丁,获得较小的搜索空间.

3.3 基于约束求解的方法

基于约束求解的方法在寻找补丁的过程中将补丁生成转换为约束求解(constraint solving)问题,应用求解器获得可行解并转换为最终补丁.基于约束求解的特性,这一类算法往往能够获得精确的结果,但也会带来较大的算法执行时间.

2012 年,该类算法的先驱,Nguyen 等人提出 SemFix[22],一种 C 程序的基于约束的语义修复算法.该算法将测试用例转换为约束,并应用 SMT(satisfiability modulo theories)求解器求解,最终转换为补丁并输出.生成补丁的过程源自基于组件的程序合成算法(component based program synthesis)[37],该算法将备选语句作为组件,提取预期的输入输出,并建立约束求解模型.另外,SemFix 采用 Tarantula 算法[38]定位潜在的 bug 位置.与基于搜索的方法,如 GenProg 不同,SemFix 不需要为生成的补丁调用测试用例验证修复性;约束机制已将测试用例作为输入,并进而转化为解输出.

2014 年,DeMarco 和 Xuan 等人[24,25]设计了 Nopol,一种面向 Java 条件语句 bug 的基于约束求解的方法.该方法针对错误条件或条件语句缺失这两种常见 bug 进行修复.过程中,采用天使定位(angelic fix localization)算法[39]确定潜在修复的位置.与 SemFix 不同,Nopol 中的天使定位算法只针对条件值(即布尔值),其搜索空间小,计算成本低,可以应用于大规模程序.该算法中采用利于面向对象故障定位的 Ochiai 算法[40]获取被修复语句的排序.实验中,Nopol 可以修复 22 个大规模 Java 程序的真实 bug 中的 17 个,具有较高的准确率.

2015 年,Mechtaev 等人提出了补丁简化算法 DirectFix[23].该方法将补丁中潜在的程序组件转换为最大可满

足性问题(maximum satisfiability,即 MaxSat),求解后转换为具体的简化后的补丁.

Ke 等人[41]设计了 SearchRepair,一种基于语义代码搜索(semantic code search)的修复方法.该方法通过将数据库中的代码段编码为基于输入输出的 SMT 模型中的约束,进而求解补丁.相对于已有算法 GenProg、AE 和 RSRepair,算法 SearchRepair 可以获得更多的补丁,且补丁质量较高.

3.4 小 结

本节将基于测试集的程序修复方法分为基于搜索的方法、基于代码穷举的方法和基于约束求解的方法这 3 类,具体介绍了基于测试集的程序修复方法.下面简述回顾文献过程中的发现:

(1) 在算法数量上,基于搜索的方法占有相当的优势,而基于代码穷举的方法最少,只有两个;

(2) 在算法时间分布上,最近两年,即 2014 年和 2015 年,新方法的数量有明显上升,该领域正日趋活跃;

(3) 在算法多样性上,新的算法设计使该领域的研究更加多样.其中,2015 年,领域内新出现的基于约束,求解的方法较多.基于约束求解方法的数量上升这一现象表明:随着计算能力的提高,约束求解技术可以在一定程度上弥补基于搜索的方法的不足,为领域增添新的血液.

4 自动修复的实证研究基础

如前文所述,基于测试集的自动程序修复的研究始于 2008 年.与一些科研领域不同,该领域自始致力于修复真实 bug 的研究.尽管程序修复研究尚处于初步阶段,但领域内一直在探讨当前修复方法在实际 bug 修复中的应用.实证研究的基础是程序修复的重要研究环节.

4.1 程序修复基础

程序修复存在一系列的基础问题有待探索,例如何种 bug 容易被修复、算法准确程度和效率以及当前算法的实战能力等.

2012 年,Le Goues 等人[42]发表了题为《每个 8 美元,修复 105 个 bug 中的 55 个》(fixing 55 out of 105 bugs for $8 each)的文章,首次以系统研究的形式展示了早期算法 GenProg 修复 C 程序 bug 的能力.该工作表明,GenProg 可以自动修复超过半数的 bug 并消耗非常小的成本(相对于开发者人工修复的成本).随后的 2013 年,Fry 等人[43]通过人员研究(human study)的形式探索补丁的可维护性(patch maintainability).该方法表明,自动生成的补丁仍具有良好的可读性.同是 2013 年,Le Goues 等人[44]发表了展望文章,描述了自动程序修复面临的挑战.该文很好地总结了 GenProg 相关算法的优劣和当时程序修复的困难.

2014 年,Tao 等人[45]通过人员研究的形式讨论自动生成的补丁在帮助人工修复 bug 时的能力.他们通过自动算法生成程序的补丁,并以此补丁作为指导,改进学生修复 bug 的过程.实验结果表明:尽管自动生成的 bug 存在一定的问题,但可以作为辅助有效地提高人工修复 bug 的效率.该实验在一定程度上证实了自动程序修复的实战可能性.Tao 等人[46]通过对开源软件项目的补丁的实证研究(empirical study),探索人工设计补丁的可接受性(acceptance),即,什么样的补丁是在开源项目中可被接受的.

2015 年,Zhong 和 Su[17]通过实证研究,检验了超过 9 000 个真实补丁并得到 15 个新的发现.这些发现关注程序修复的两个重要方面:故障定位和代码补丁.该工作为故障定位和补丁生成的实验基础研究提供了详细的结果和细致的指导.

4.2 修复问题划定和算法评价

在 2014 年以前,程序修复研究以设计新的方法为主.这期间涌现了许多优秀的修复算法,例如 GenProg[5],AE[20],RSRepair[16]等.其中,2013 年的方法 Par[6],即通过学习人工补丁模式指导补丁生成的方法,获得软件工程顶级会议 ICSE 的最佳论文奖(distinguished paper award).当时,该方法首次融合了自动算法 GenProg 和人工补丁模式来生成最终的补丁.

然而,在 2014 年,Monperrus[9]在 ICSE 上发表了题为《严重审视“从人工补丁学习自动化代码生成”》(a critical review of automatic patch generation learned from human-written patches)的文章.该文章针对 Par[6]方法,

批评 Par 所属论文的算法和实验中的不足,进而引申到如何划定程序修复问题和评价算法的探讨中.在后续的讨论中,Monperrus 申明了基于测试集的程序修复研究的内容(即,修复算法核心为:在给定测试集的情况下获得高质量补丁,而低质量的测试集并非算法缺陷)和算法评价标准(即,算法比较应基于同样的数据集和同样的缺陷类型)等.尽管研究者对此文评价不一,但该文开启了基于测试集的程序修复研究自2008年以来持续6年研究后的第1次大规模学术争论.研究者们开始关注非算法部分,并深入探索问题的本质和修复实例的基础.

同是2014年,Martinez等人[47]和Barr等人[48]在互不知情的情况下,同时发表了两篇关于程序修复基础的文章.Martinez等人[47]发表了题为《代码修复的原料存在吗?》(do the fix ingredients already exist?)的文章,以实证调查(empirical inquiry)的形式探索了冗余假设(redundancy assumption)的存在性.所谓的"冗余假设"是指自动生成的补丁一定存在于程序的其他某处,而修复算法是将其他位置的代码重用(reuse)或组合(mix)而形成补丁的.早期算法,如 GenProg,AE,RSRepair 等,都基于该假设.文献[47]表明:冗余假设确实一定程度地存在着,但并非完全存在.几乎同一时间,Barr等人[48]发表了题为《整形外科手术猜想》(the plastic surgery hypothesis)的文章,探索程序修复的基础.所谓"整形外科手术猜想"与"冗余假设"类似,是指补丁中的代码能够从程序的其他位置(像植皮技术一样)移动到修复 bug 的位置.该文设计了复杂的实验,深入地讨论了猜想的存在性.

4.3 修复真实bug的能力

修复真实 bug 一直是程序修复研究的目标.以此为基础,程序修复进而才可能完成工业应用.然而,这一研究进程出现了波折.如第 4.1 节所述,Le Goues 等人[42]首次组织了 105 个真实的 C 程序 bug 的数据集,并应用 GenProg 算法评估修复能力.此后,多个研究者以此数据集作为实验平台,研究自动修复的相关问题.

直至2015年,Qi等人[10]发表了题为《貌似可信和正确的补丁的分析》(an analysis of patch plausibility and correctness)的文章,手工检查了早期算法在105个真实的C程序bug的数据集上的修复结果.结果发现:由于实验设置错误,在GenProg文章[42]报道的55个可修复的bug中,有37个为修复后的程序是无法通过全部测试用例的;而在AE文章[20]报道的54个可修复的bug中,有27个为修复后的程序是无法通过全部测试用例的.此外,基于人工验证:在全部105个bug中,GenProg的修复结果只有2个是语义正确的,而AE的结果只有3个是语义正确的;其他的修复均无法满足原有程序的需求说明.文献[20]的发表在领域内引起轩然大波,相关研究者不得不重新检验相关结果的正确性.在2014年的第1次学术争论之后,该文引发了领域内的第2次集体争论,进而导致研究者深度关注对于修复真实bug的可行性的研究.

同年,Smith等人[11]发表了题为《适得其反?》(is the cure worse than the disease?)的文章,研究修复算法的过拟合(overfitting).该文设计了可控实验,通过新开发者引入的bug和补丁,分析影响GenProg和RSRepair算法效果的因素,并指出修复算法对于测试用例的过拟合行为.同年,Le Goues等人[49]介绍了两个C语言的被修复程序集合 ManyBugs 和 IntroClass,共包含 1 141 个缺陷.基于该数据集,他们设计了量化实验,评估已有方法的修复效果.

仍是2015年,Durieux等人[26]选取了3种经典算法的Java版本,在Java语言的真实bug上进行修复实验.这3种算法是GenProg,Kali(Java版本[50])和Nopol[24,25];而数据集是包含222个bug的Defects4J[51].实验结果表明:在面向对象的程序中,Nopol能够修复35个bug,而GenProg和Kali分别修复27个和22个;而从语义正确的角度来看,Nopol可以正确地修复5个,而GenProg和Kali分别正确地修复了5个和1个.该结果表明:修复算法的当前研究仍处于初级阶段,与实际应用尚存一定距离.

4.4 小 结

修复真实 bug 是自动程序修复方法通向真实应用的必经之路.自该领域研究初期开始,如何精确而高效地修复真实 bug 一直是研究核心之一.由于程序自身的复杂性,实证研究是发现和检验修复真实 bug 能力的主要方法.然而,正如本节所介绍,经历了两次领域内的集体学术争论之后,自动修复的实证研究在波折之中继续前进.由本节内容可以得到如下发现:

(1) 随着对自动修复的基础的探索,更多的细节被深入挖掘,实证研究可在一定程度上为程序修复提供事

实依据;

(2) 作为自动修复的核心内容之一,修复真实的 bug 仍未曾被研究者攻克.更进一步地,生成符合程序语义的补丁十分困难;

(3) 由于不同算法源自不同的研究者,建立公平的算法比较还存在一定困难.其中之一就是从真实项目提取的被修复 bug 数量有限,不足以满足现实 bug 的自然分布.

5 程序修复相关技术

5.1 非基于测试集的修复方法的近期工作简介

除了基于测试集的修复方法,研究者们也关注其他形式的修复方法,例如形式化修复[52]和动态程序状态修复[53]等.下面对近期的一些工作进行简要的介绍.

Dallmeier 等人[54]提出了 Pachika,一种检测程序对象非正常行为(object behavior anomaly detection)的修复方法.该方法识别两个对象间的不同,进而增加或者删除方法调用.Carzaniga 等人[55,56]开发了一种用于避免 Web 应用错误的自动方法.该方法旨在找到可运行的程序变种(program variant),作为自动的临时变通版本(automatic workaround).Pei 等人[57]设计了 AutoFix,一个基于契约的修复(contract based repair)系统.该方法需要程序契约中的规约信息作为输入,例如函数的前置和后置条件等,可用于 Effiel 语言的 bug 修复.Tan 和 Roychoudhury[58]的近期工作 Relifix 针对程序回归中的问题进行修复.

Sidiroglou-Douskos 等人[59]提出了 CodePhage,一种通过从被修复应用中定位缺陷,并从其他应用迁移代码(code transfer)来消除缺陷的算法.该算法依赖于作为输入的应用中能消除缺陷且成熟的代码,可修复指定缺陷,如整数溢出、缓冲区溢出及零除数等.Raychev 等人[60]提出了 JSNice,一种面向 JavaScript 语言的自动程序美化工具.该工具通过从海量代码库学习代码特性,为程序预测变量名或生成一定的注释.该工具的特点是自动化,且生成的代码及注释具有一定程度的语义信息.另外一组相关工作是测试用例修复(test case repair)[61−64],旨在自动修复由于代码更新造成的测试用例无法正常运行的问题[65].

5.2 程序修复的相关技术的近期工作简介

定位 bug 和获得更好的测试集,是自动程序修复算法中两个相关步骤.下面简单介绍这两方面的近期相关工作.

故障定位旨在通过收集测试用例运行信息,推断潜在的 bug 所处的位置.自动修复算法中借助已有的故障定位技术,根据疑似故障的风险给出语句的排列,进而逐个尝试修复.基于程序谱的故障定位(spectrum based fault localization)方法是其中一类常见的方法.故障定位的成果浩如烟海,这里只列出部分近期工作.

Zhang 等人[66]设计了只依赖于失败测试用例的故障定位方法,有效地减少了测试用例的运行成本.Xie 等人[3]从理论上分析了基于程序谱的故障定位方法中风险函数(risk formula)之间的关系;Lucia 等人[67]从实证研究的角度给出了风险函数间的关联.Masri 和 Assi[68]针对故障定位中的巧合正确性(coincidental correctness)问题,分析了缓解影响定位准确率因素的措施.Xuan 和 Monperrus[69]通过机器学习技术,组合已有的风险函数,进一步提升故障定位的准确率.另外的一些近期工作包括 Jiang 等人[70]、Debroy 和 Wong[71]、Xu 等人[72]、Xuan 等人[73]的工作等.

测试用例增强(test case augmentation)是近期涌现的一系列工作.尽管测试用例生成技术的研究历史悠久,但受限于测试路径组合爆炸及面向对象程序的复杂状态等问题,测试用例生成技术尚不能满足所有应用的需求.测试用例增强旨在基于给定的测试用例,获得更好的测试效果,如生成可以弥补当前测试用例不足的新测试用例或更好地使用当前测试用例.

Xu 等人[74,75]提出了早期的直接测试用例增强(directed test suite augmentation)技术,该方法在当前测试用例集的基础上生成新的测试用例,进而提高测试用例的覆盖率等指标.Yoo 和 Harman[76]设计了测试重生成(test data regeneration)算法.该方法对于一个测试集,基于遗传算法生成全新的测试用例,用以弥补已有测试用例的

不足.Xuan 和 Monperrus[77]提出了测试用例提纯(test case purification)方法.该方法能够分离执行失败的测试用例,并切分为多个子测试用例,优化测试用例的执行,进而增强其识别故障的能力.

6 机遇与挑战

本文回顾了基于测试集的自动程序修复的研究进展.自动程序修复技术分析测试用例,搜索潜在的程序补丁,以通过全部测试用例集.在充分调研相关文献之后,从自动程序修复方法和实证研究基础两个方面详细介绍了该领域的近期成果.在自动修复方法方面,分 3 个类别依年份介绍了算法的发展历程;在实证研究方面,详述了实证研究的成果和潜在问题.基于测试集的自动程序修复的研究仅有 8 年,领域历史较短但成果丰富.

前文详述了领域的研究进展,下面简要分析自动程序修复面临的机遇和挑战:

(1) 修复算法的指引:在搜索程序补丁的过程中,自动程序修复算法往往并没有得到足够的指引.例如,早期的 GenProg 等算法实际上是借助于启发式规则寻找可能存在的补丁;后续的 SemFix 等算法虽然能够从测试用例提取出约束以缩减搜索空间,但仍遗留大量的潜在的不正确的补丁.因此,如何利用被修复程序代码结构和已知的测试用例指引自动算法搜索补丁,是该领域仍存在的重要问题;

(2) 多点 bug 修复:目前的自动修复算法均以单点 bug(single-point bug)为修复对象,即,程序中的 bug 只包含于一条语之中.尽管单点 bug 的修复研究尚不成熟,修复多点 bug(multi-point bug,即 bug 存在于多条语句之中)仍具有重要的科研价值.多点 bug 的修复并非简单地组合单点 bug 的修复算法:仅仅更新一条语句不能通过全部测试用例,因此,修复算法无法知晓当前操作的正确性.已有的多故障定位技术为多点 bug 修复提供了一定的思路和技术支持;

(3) 补丁定位:为了修复 bug,自动算法首先通过故障定位技术将潜在的 bug 位置按照疑似包含 bug 的可能性排序.尽管故障定位的研究已经发展了 15 年,但精确定位 bug 仍然面临着一系列难题.而自动程序修复的研究既要面对故障定位领域原有的困难,又为故障定位方法的改进提供了机遇;

(4) 测试集的利用:测试集是自动修复中的重要输入.如何基于当前的测试集获得更多的测试用例以及如何更好地利用当前测试用例,是利用测试集改进程序修复的途径之一.程序修复将为测试用例生成的相关技术提供应用场景和发展机遇;而测试用例生成也将为改进程序修复方法提供支撑.

References:

[1] Pressman RS. Software Engineering: A Practitioner's Approach. 7th ed., New York: McGraw-Hill, 2010. 437−443.

[2] Xie X, Chen TY, Kuo FC, Xu B. A theoretical analysis of the risk evaluation formulas for spectrum-based fault localization. ACM Trans. on Software Engineering and Methodology, 2013,22(4):31:1−31:40. [doi: 10.1145/2522920.2522924]

[3] Wen WZ, Li BX, Sun XB, Liu CC. Technique of software fault localization based on hierarchical slicing spectrum. Ruan Jian Xue Bao/Journal of Software, 2013,24(5):977−992 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/4342.htm [doi: 10.3724/SP.J.1001.2013.04342]

[4] Xuan J, Jiang H, Ren Z, Zou W. Developer prioritization in bug repositories. In: Proc. of the 34th Int'l Conf. on Software Engineering (ICSE). 2012. 25−35. [doi: 10.1109/ICSE.2012.6227209]

[5] Le Goues C, Nguyen T, Forrest S, Weimer W. GenProg: A generic method for automatic software repair. IEEE Trans. on Software Engineering, 2012,38:54−72. [doi: 10.1109/TSE.2011.104]

[6] Kim D, Nam J, Song J, Kim S. Automatic patch generation learned from human-written patches. In: Proc. of the ACM/IEEE Int'l Conf. on Software Engineering (ICSE). 2013. 802−811. [doi: 10.1109/ICSE.2013.6606626]

[7] Harman M, Mansouri A, Zhang Y. Search based software engineering: Trends, techniques and applications. ACM Computing Surveys, 2012,45(1):11:1−11:61. [doi: 10.1145/2379776.2379787]

[8] Weimer W, Nguyen T, Le Goues C, Forrest S. Automatically finding patches using genetic programming. In: Proc. of the Int'l Conf. on Software Engineering (ICSE). 2009. 364−367. [doi: 10.1109/ICSE.2009.5070536]

[9] Monperrus M. A critical review of automatic patch generation learned from human-written patches: Essay on the problem statement and the evaluation of automatic software repair. In: Proc. of the 36th Int'l Conf. on Software Engineering (ICSE). 2014. 234−242. [doi: 10.1145/2568225.2568324]

[10] Qi Z, Long F, Achour S, Rinard M. An analysis of patch plausibility and correctness for generate-and-validate patch generation systems. In: Proc. of the Int'l Symp. on Software Testing and Analysis (ISSTA). 2015. 24−36. [doi: 10.1145/2771783.2771791]

[11] Smith EK, Barr E, Le Goues C, Brun Y. Is the cure worse than the disease? Overfitting in automated program repair. In: Proc. of the European Software Engineering Conf. and ACM SIGSOFT Int'l Symp. on Foundations of Software Engineering (ESEC/FSE). 2015. 532−543. [doi: 10.1145/2786805.2786825]

[12] Qi Y, Mao X, Lei Y. Making automatic repair for large-scale programs more efficient using weak recompilation. In: Proc. of the IEEE Int'l Conf. on Software Maintenance (ICSM). 2012. 254−263. [doi: 10.1109/ICSM.2012.6405280]

[13] Qi Y, Mao X, Wen Y, Dai Z, Gu B. More efficient automatic repair of large-scale programs using weak recompilation. Science China Information Sciences, 2012,55(12):2785−2799. [doi: 10.1007/s11432-012-4741-1]

[14] Qi Y, Mao X, Lei Y. Efficient automated program repair through fault-recorded testing prioritization. In: Proc. of the IEEE Int'l Conf. on Software Maintenance (ICSM). 2013. 180−189. [doi: 10.1109/ICSM.2013.29]

[15] Qi Y, Mao X, Lei Y, Wang C. Using automated program repair for evaluating the effectiveness of fault localization techniques. In: Proc. of the Int'l Symp. on Software Testing and Analysis (ISSTA). 2013. 191−201. [doi: 10.1145/2483760.2483785]

[16] Qi Y, Mao X, Lei Y, Dai Z, Wang C. The strength of random search on automated program repair. In: Proc. of the 36th Int'l Conf. on Software Engineering (ICSE). 2014. 254−265. [doi: 10.1145/2568225.2568254]

[17] Zhong H, Su Z. An empirical study on real bug fixes. In: Proc. of the 37th Int'l Conf. on Software Engineering (ICSE). 2015. 913−923. [doi: 10.1109/ICSE.2015.101]

[18] Harman M, Jones B. Search based software engineering. Journal of Information and Software Technology, 2011,43(14):833−839.

[19] Jiang H, Xuan J, Ren Z. Approximate backbone based multilevel algorithm for next release problem. In: Proc. of the 12th Annual Conf. on Genetic and Evolutionary Computation (GECCO). 2010. 1333−1340. [doi: 10.1145/1830483.1830730]

[20] Weimer W, Fry ZP, Forrest S. Leveraging program equivalence for adaptive program repair: Models and first results. In: Proc. of the IEEE/ACM Int'l Conf. on Automated Software Engineering (ASE). 2013. 356−366. [doi: 10.1109/ASE.2013.6693094]

[21] Debroy V, Wong WE. Using mutation to automatically suggest fixes for faulty programs. In: Proc. of the 3rd Int'l Conf. on Software Testing, Verification and Validation (ICST). 2010. 65−74. [doi: 10.1109/ICST.2010.66]

[22] Nguyen HDT, Qi D, Roychoudhury A, Chandra S. SemFix: Program repair via semantic analysis. In: Proc. of the Int'l Conf. on Software Engineering (ICSE). 2013. 772−781. [doi: 10.1109/ICSE.2013.6606623]

[23] Mechtaev S, Yi J, Roychoudhury A. DirectFix: Looking for simple program repairs. In: Proc. of the 37th Int'l Conf. on Software Engineering (ICSE). 2015. 448−458. [doi: 10.1109/ICSE.2015.63]

[24] DeMarco F, Xuan J, Le Berre D, Monperrus M. Automatic repair of buggy if conditions and missing preconditions with SMT. In: Proc. of the 6th Int'l Workshop on Constraints in Software Testing, Verification, and Analysis (CSTVA). 2014. 30−39. [doi: 10.1145/2593735.2593740]

[25] Xuan JF, Martinez M, DeMarco F, Clément M, Lamelas-Marcote S, Durieux T, Le Berre D, Monperrus M. Nopol: Automatic repair of conditional statement bugs in Java programs. Technical Report, INRIA, 2015. 1−22.

[26] Martinez T, Durieux T, Xuan JF, Monperrus M. Automatic repair of real bugs in Java: A large-scale experiment on the Defects4J dataset. Technical Report, arXiv:1505.07002, ArXiv, 2015. 1−11.

[27] Arcuri A, Yao X. A novel co-evolutionary approach to automatic software bug fixing. In: Proc. of the IEEE Congress on Evolutionary Computation (CEC). 2008. 162−168. [doi: 10.1109/CEC.2008.4630793]

[28] Forrest S, Weimer W, Nguyen T, Le Goues C. A genetic programming approach to automated software repair. In: Proc. of the Genetic and Evolutionary Computing Conf. (GECCO). 2009. 947−954. [doi: 10.1145/1569901.1570031]

[29] Weimer W, Forrest S, Le Goues C, Nguyen T. Automatic program repair with evolutionary computation. Communications of the ACM, 2010,53(5):109−116. [doi: 10.1145/1735223.1735249]

[30] Fast E, Le Goues C, Forrest S, Weimer W. Designing better fitness functions for automated program repair. In: Proc. of the Genetic and Evolutionary Computing Conf. (GECCO). 2010. 965−972. [doi: 10.1145/1830483.1830654]

[31] Le Goues C, Weimer W, Forrest S. Representations and operators for improving evolutionary software repair. In: Proc. of the Genetic and Evolutionary Computation Conf. (GECCO). 2012. 959−966. [doi: 10.1145/2330163.2330296]

[32] Schulte E, Forrest S, Weimer W. Automated program repair through the evolution of assembly code. In: Proc. of the ACM/IEEE Int'l Conf. on Automated Software Engineering (ASE). 2010. 313−316. [doi: 10.1145/1858996.1859059]

[33] Martinez M, Monperrus M. Mining software repair models for reasoning on the search space of automated program fixing. Empirical Software Engineering, 2015,20(1):176−205. [doi: 10.1007/s10664-013-9282-8]

[34] Long F, Rinard M. Prophet: Automatic patch generation via learning from successful patches. Technical Report, MIT-CSAIL-TR-2015-027, CSAIL MIT, 2015. 1−11. http://hdl.handle.net/1721.1/97735

[35] Jia Y, Harman M. An analysis and survey of the development of mutation testing. IEEE Trans. on Software Engineering, 2011, 37(5):649−678. [doi: 10.1109/TSE.2010.62]

[36] Long F, Rinard M. Staged program repair with condition synthesis. In: Proc. of the 10th Joint Meeting of the European Software Engineering Conf. and the ACM SIGSOFT Symp. on the Foundations of Software Engineering (ESEC/FSE). 2015. 166−178. [doi: 10.1145/2786805.2786811]

[37] Jha S, Gulwani S, Seshia SA, Tiwari A. Oracle guided component-based program synthesis. In: Proc. of the ACM/IEEE Int'l Conf. on Software Engineering (ICSE). 2010. 215−224. [doi: 10.1145/1806799.1806833]

[38] Jones JA, Harrold MJ. Empirical evaluation of the tarantula automatic fault-localization technique. In: Proc. of the 20th IEEE/ACM Int'l Conf. on Automated Software Engineering (ASE). 2005. 273−282. [doi: 10.1145/1101908.1101949]

[39] Zhang X, Gupta N, Gupta R. Locating faults through automated predicate switching. In: Proc. of the 28th Int'l Conf. on Software Engineering. 2006. 272−281. [doi: 10.1145/1134285.1134324]

[40] Abreu R, Zoeteweij P, Gemund AGV. On the accuracy of spectrum-based fault localization. In: Proc. of the Testing: Academic and Industrial Conf. on Practice and Research Techniques (Mutation). 2007. 89−98. [doi: 10.1109/TAIC.PART.2007.13]

[41] Ke Y, Stolee KT, Le Goues C, Brun Y. Repairing programs with semantic code search. In: Proc. of the IEEE/ACM Conf. on Automated Software Engineering (ASE). 2015. 295−306. [doi: 10.1109/ASE.2015.60]

[42] Le Goues C, Dewey-Vogt M, Forrest S, Weimer W. A systematic study of automated program repair: Fixing 55 out of 105 bugs for $8 each. In: Proc. of the ACM/IEEE Int'l Conf. on Software Engineering (ICSE). 2012. 3−13. [doi: 10.1109/ICSE.2012.6227211]

[43] Fry ZP, Landau B, Weimer W. A human study of patch maintainability. In: Proc. of the Int'l Symp. on Software Testing and Analysis (ISSTA). 2012. 177−187. [doi: 10.1145/2338965.2336775]

[44] Le Goues C, Forrest S, Weimer W. Current challenges in automatic software repair. Software Quality Journal, 2013,21(3):421−443. [doi: 10.1007/s11219-013-9208-0]

[45] Tao Y, Kim J, Kim S, Xu C. Automatically generated patches as debugging aids: A human study. In: Proc. of the 22nd ACM SIGSOFT Int'l Symp. on the Foundations of Software Engineering (FSE). 2014. 64−74. [doi: 10.1145/2635868.2635873]

[46] Tao Y, Han D, Kim S. Writing acceptable patches: An empirical study of open source project patches. In: Proc. of the 30th Int'l Conf. on Software Maintenance and Evolution (ICSME). 2014. 271−280. [doi: 10.1109/ICSME.2014.49]

[47] Martinez M, Weimer W, Monperrus M. Do the fix ingredients already exist? An empirical inquiry into the redundancy assumptions of program repair approaches. In: Proc. of the 36th Int'l Conf. on Software Engineering (ICSE). 2014. 492−495. [doi: 10.1145/2591062.2591114]

[48] Barr ET, Brun Y, Devanbu PT, Harman M, Sarro F. The plastic surgery hypothesis. In: Proc. of the 22nd ACM SIGSOFT Int'l Symp. on Foundations of Software Engineering (FSE). 2014. 306−317. [doi: 10.1145/2635868.2635898]

[49] Le Goues C, Holtschulte N, Smith EK, Brun Y, Devanbu P, Forrest S, Weimer W. The ManyBugs and IntroClass benchmarks for automated repair of C programs. IEEE Trans. on Software Engineering, 2015. [doi: 10.1109/TSE.2015.2454513]

[50] Martinez M, Monperrus M. Astor: Evolutionary automatic software repair for Java. Technical Report, arXiv:1410.6651, ArXiv, 2015. 1−6.

[51] Just R, Jalali D, Ernst MD. Defects4J: A database of existing faults to enable controlled testing studies for Java programs. In: Proc. of the Int'l Symp. on Software Testing and Analysis (ISSTA). 2014. 437−440. [doi: 10.1145/2610384.2628055]

[52] Jobstmann B, Griesmayer A, Bloem R. Program repair as a game. In: Proc. of the Computer Aided Verification (CAV). 2005. 226−238. [doi: 10.1007/11513988_23]

[53] Perkins JH, Kim S, Larsen S, Amarasinghe S, Bachrach J, Carbin M, Pacheco C, Sherwood F, Sidiroglou-Douskos S, Sullivan G, Wong WF, Zibin Y, Ernst MD, Rinard M. Automatically patching errors in deployed software. In: Proc. of the ACM Symp. on Operating Systems Principles (SOSP). 2009. 87−102. [doi: 10.1145/1629575.1629585]

[54] Dallmeier V, Zeller A, Meyer B. Generating fixes from object behavior anomalies. In: Proc. of the IEEE/ACM Int'l Conf. on Automated Software Engineering (ASE). 2009. 550−554. [doi: 10.1109/ASE.2009.15]

[55] Carzaniga A, Gorla A, Mattavelli A, Perino N, Pezzè M. Automatic recovery from runtime failures. In: Proc. of the ACM/IEEE Int'l Conf. on Software Engineering (ICSE). 2013. 782−791. [doi: 10.1109/ICSE.2013.6606624]

[56] Carzaniga A, Gorla A, Mattavelli A, Perino N, Pezzè M. Automatic workarounds for Web applications. In: Proc. of the ACM SIGSOFT Int'l Symp. on Foundations of Software Engineering (FSE). 2010. 237−246. [doi: 10.1145/1882291.1882327]

[57] Pei Y, Furia CA, Nordio M, Wei Y, Meyer B, Zeller A. Automated fixing of programs with contracts. IEEE Trans. on Software Engineering, 2014,40(5):427−449. [doi: 10.1109/TSE.2014.2312918]

[58] Tan SH, Roychoudhury A. Relifix: Automated repair of software regressions. In: Proc. of the Int'l Conf. on Software Engineering (ICSE). 2015. 471−482. [doi: 10.1109/ICSE.2015.65]

[59] Sidiroglou-Douskos S, Lahitnen E, Long F, Rinard M. Automatic error elimination by horizontal code transfer across multiple applications. In: Proc. of the 36th ACM SIGPLAN Conf. on Programming Language Design and Implementation (PLDI). 2015. 43−54. [doi: 10.1145/2737924.2737988]

[60] Raychev V, Vechev M, Krause A. Predicting program properties from "big code". In: Proc. of the 42nd Annual ACM SIGPLAN/SIGACT Symp. on Principles of Programming Languages (POPL). 2015. 111−124. [doi: 10.1145/2676726.2677009]

[61] Daniel B, Jagannath V, Dig D, Marinov D. Reassert: Suggesting repairs for broken unit tests. In: Proc. of the Int'l Conf. on Automated Software Engineering (ASE). 2009. 433−444. [doi: 10.1109/ASE.2009.17]

[62] Mirzaaghaei M, Pastore F, Pezzè M. Supporting test suite evolution through test case adaptation. In: Proc. of the 5th Int'l Conf. on Software Testing, Verification and Validation (ICST). 2012. 231−240. [doi: 10.1109/ICST.2012.103]

[63] Mirzaaghaei M, Pastore F, Pezzè M. Automatic test case evolution. Software Testing, Verification and Reliability, 2014,24(5):386−411. [doi: 10.1002/stvr.1527]

[64] Gao Z, Chen Z, Zou Y, Memon A. Sitar: GUI test script repair. IEEE Trans. on Software Engineering, 2016,42(2):170−186. [doi: 10.1109/TSE.2015.2454510]

[65] Zhang ZY, Chen ZY, Xu BW, Yang R. Research progress on test case evolution. Ruan Jian Xue Bao/Journal of Software, 2013, 24(4):663−674 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/4379.htm [doi: 10.3724/SP.J.1001.2013.04379]

[66] Zhang Z, Chan WK, Tse TH. Fault localization based only on failed runs. IEEE Computer, 2012,45(6):64−71. [doi: 10.1109/MC.2012.185]

[67] Lucia L, Lo D, Jiang L, Thung F, Budi A. Extended comprehensive study of association measures for fault localization. Journal of Software: Evolution and Process, 2014,26(2):172−219. [doi: 10.1002/smr.1616]

[68] Masri W, Assi RA. Prevalence of coincidental correctness and mitigation of its impact on fault localization. ACM Trans. on Software Engineering Methodology, 2014,23(1):8:1−8:28. [doi: 10.1145/2559932]

[69] Xuan JF, Monperrus M. Learning to combine multiple ranking metrics for fault localization. In: Proc. of the 30th Int'l Conf. on Software Maintenance and Evolution (ICSME). 2014. 191−200. [doi: 10.1109/ICSME.2014.41]

[70] Jiang B, Zhai K, Chan WK, Tse TH, Zhang Z. On the adoption of MC/DC and control-flow adequacy for a tight integration of program testing and statistical fault localization. Journal of Information and Software Technology, 2013,55:897−917. [doi: 10.1016/j.infsof.2012.10.001]

[71] Debroy V, Wong WE. Combining mutation and fault localization for automated program debugging. Journal of Systems and Software, 2014,90:45−60. [doi: 10.1016/j.jss.2013.10.042]

[72] Xu J, Zhang Z, Chan WK, Tse TH, Li S. A general noise-reduction framework for fault localization of Java programs. Journal of Information and Software Technology, 2013,55:880−896. [doi: 10.1016/j.infsof.2012.08.006]

[73] Xuan JF, Xie X, Monperrus M. Crash reproduction via test case mutation: Let existing test cases help. In: Proc. of the 10th Joint Meeting on Foundations of Software Engineering (ESEC/FSE). 2015. 910−913. [doi: 10.1145/2786805.2803206]

[74] Xu Z, Kim Y, Kim M, Rothermel G, Cohen MB. Directed test suite augmentation: Techniques and tradeoffs. In: Proc. of the 18th ACM SIGSOFT Int'l Symp. on Foundations of Software Engineering (FSE). 2010. 257−266. [doi: 10.1145/1882291.1882330]

[75] Xu Z, Cohen MB, Rothermel G. Factors affecting the use of genetic algorithms in test suite augmentation. In: Proc. of the 12th Annual Conf. on Genetic and Evolutionary Computation (GECCO). 2010. 1365−1372. [doi: 10.1145/1830483.1830734]

[76] Yoo S, Harman M. Test data regeneration: Generating new test data from existing test data. Journal of Software Testing, Verification and Reliability, 2012,22(3):171−201. [doi: 10.1002/stvr.435]

[77] Xuan JF, Monperrus M. Test case purification for improving fault localization. In: Proc. of the 22nd ACM SIGSOFT Int'l Symp. on the Foundations of Software Engineering (FSE). 2014. 52−63. [doi: 10.1145/2635868.2635906]

附中文参考文献:

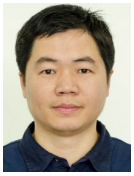
[3] 文万志,李必信,孙小兵,刘翠翠.一种基于层次切片谱的软件错误定位技术.软件学报,2013,24(5):977−992. http://www.jos.org.cn/1000-9825/4342.htm [doi: 10.3724/SP.J.1001.2013.04342]

[65] 张智轶,陈振宇,徐宝文,杨瑞.测试用例演化研究进展.软件学报,2013,24(4):663−674. http://www.jos.org.cn/1000-9825/4379.htm [doi: 10.3724/SP.J.1001.2013.04379]

玄跻峰(1984－),男,黑龙江哈尔滨人,博士,研究员,CCF 会员,主要研究领域为软件分析与测试,软件数据分析,基于搜索的软件工程.

谢晓园(1983－),女,博士,教授,CCF 会员,主要研究领域为软件测试调试,程序分析,基于搜索的软件工程.

任志磊(1984－),男,博士,讲师,CCF 会员,主要研究领域为演化计算,基于搜索的软件工程.



江贺(1980－),男,博士,教授,博士生导师,CCF 高级会员,主要研究领域为基于搜索的软件工程,软件仓库挖掘.

王子元(1982－),男,博士,副教授,CCF 会员,主要研究领域为软件测试.