自动程序修复方法研究述评

王 赞[1)] 郜 健[1)] 陈 翔[2)] 傅浩杰[1)] 樊向宇[1)]

[1)](天津大学软件学院 天津 300350)

[2)](南通大学计算机科学与技术学院 江苏 南通 226019)

摘 要 当开发人员面对大量缺陷报告无从入手的时候,自动程序修复(Automatic Program Repair,APR)可以成功完成其中一些缺陷的自动修复,从而有效减少开发人员的程序调试时间.因此自动程序修复逐渐成为当前软件维护领域中的一个研究热点并取得了一定的研究进展.该文通过对权威期刊和会议进行检索,搜集了100余篇相关论文,从现有自动程序修复的研究历程、该方法中的重要问题以及国内外相关研究团队及成果等几个方面对该领域进行了系统的总结.其中基于测试用例的自动程序修复方法是当前一种主流方法,其通过配套测试用例集来评估生成补丁的质量.论文将这类方法细分为三个阶段:软件缺陷定位阶段、生成补丁阶段和补丁评估阶段.其中软件缺陷定位阶段是自动程序修复的基础,其目标是尽可能精确识别出可能含有缺陷的语句.生成补丁阶段一般通过预先定义的修改操作对缺陷语句进行修改,代码修改操作在设定时可以考虑修复程序的自身代码、开源项目的代码或者问答网站中的知识等.补丁评价阶段则对生成的候选补丁进行评估,直到找到一个补丁可以使得所有测试用例均执行通过,并随后借助开发人员的人工分析进行最终确认.在上述三个阶段中,生成补丁阶段是自动程序修复方法的核心,论文将已有补丁生成方法细分为三类:基于搜索的方法、基于语义的方法和其他类型方法.其中基于搜索的方法在搜索空间内通过搜索生成补丁,并借助配套测试用例集对该补丁进行验证,经典的方法包括GenProg、PAR、AE、RSrepair等;基于语义的方法则借助语义信息来合成补丁,主要基于符号执行和约束求解,经典的方法包括SemFix、DirectFix、Angelix、Nopol等;而不属于上述两类方法的研究工作则被归于其他类型的方法,经典的工作包括基于问答网站和开源项目托管网站挖掘来生成补丁.而补丁评价阶段是自动程序修复方法研究的争议焦点,近两年研究人员更多关注正确补丁在搜索空间中的分布和补丁的正确性.特别是针对减少补丁的验证花销和提高补丁质量的问题做出了进一步的探索.随后论文针对特定领域的自动程序修复方法进行了总结,包括并发缺陷、数据库缺陷、空指针缺陷、数据结构缺陷、内存泄漏缺陷等的自动修复方法.其中重点分析了针对并发缺陷的自动修复方法,将已有研究工作分为数据竞争的自动修复、原子性违背的自动修复、顺序违背的自动修复和死锁的自动修复.然后论文总结了自动程序修复方法在有效性评估中经常使用的缺陷库,不难看出ManyBugs、IntroClass、Simens程序集和Defect4J是目前使用最多的缺陷库.为了方便研究人员更好的与自己提出的修复方法进行比较,论文搜集了目前已经共享的自动程序修复工具并给出了相关参考文献和具体下载地址.接着论文对国内外在程序自动修复领域比较活跃的研究小组进行了总结,并对每个研究组的主要贡献进行了总结,以方便国内外研究人员对他们的后续研究工作进行及时跟踪.最后总结全文,并依次从缺陷定位、补丁生成和评估、缺陷数量和类型、特定领域的缺陷修复以及缺陷修复在工业界中的应用这五个维度对未来的研究工作进行了展望.

关键词 自动程序修复;软件缺陷定位;基于搜索的软件工程;约束求解;并发程序缺陷修复

中图法分类号 TP311 **DOI号** 10.11897/SP.J.1016.2018.00588

Automatic Program Repair Techniques: A Survey

WANG Zan[1)] GAO Jian[1)] CHEN Xiang[2)] FU Hao-Jie[1)] FAN Xiang-Yu[1)]

[1)](*School of Computer Software, Tianjin University, Tianjin* 300350)

[2)](*School of Computer Science and Technology, Nantong University, Nantong, Jiangshu* 226019)

Abstract When developers do not know how to fix the defects in many bugreports, automatic program repair (APR) can successfully repair some bugs to reduce developer's debugging time

收稿日期:2016-06-12;在线出版日期:2017-07-28.本课题得到国家自然科学基金项目(61202030,61202006,71502125)部分资助.**王 赞**,男,1979年生,博士,副教授,中国计算机学会(CCF)会员,主要研究方向为基于搜索的软件工程、软件缺陷自动定位及缺陷自动修复.E-mail:wangzan@tju.edu.cn.**郜 健**,女,1992年生,硕士研究生,主要研究方向为软件测试优化、自动程序修复.**陈 翔**(通信作者),男,1980年生,博士,副教授,中国计算机学会(CCF)会员,主要研究方向为软件缺陷自动修复、软件缺陷预测、软件缺陷自动定位、回归测试.E-mail:xchencs@ntu.edu.cn.**傅浩杰**,女,1993年生,硕士研究生,主要研究方向为软件缺陷自动定位、软件缺陷自动修复.**樊向宇**,男,1992年生,硕士研究生,主要研究方向为软件缺陷自动定位、软件缺陷自动修复.

effectively. Therefore, APR has been an active research topic in software maintenance domain and has made some progress nowadays. After retrieving more than 100 papers from the authoritative journals and conferences, this survey offers a systematic overview of existing research work on APR to highlight the research progress, identify the important issues, and summarize the achievements of related research groups in APR. Among all approaches of APR, test suite based APR is the current mainstream, which utilizes the test suite to verify the generated patches and includes three main stages: Fault localization, patch generation, and patch evaluation. As the foundation stage of test suite based APR, fault localization aims to identify statements that may contain defects as precisely as possible. After localizing the buggy statements, some candidate patches are then generated by predefined modification operators. The design of modification operators is based on the program itself, the code from the open source projects, or the information in the Q&A sites. In the patch evaluation stage, the candidate patches are evaluated until a patch is found that allows all test cases to pass and subsequently be confirmed by developers manually. In these three stages, the patch generation phase is the emphasis and the existing patch generation methods can be classified into three categories: search based methods, semantic based methods and other types of methods. The search based methods, such as GenProg, PAR, AE, RSrepair etc., generate patches by searching in a search space, and use the test suite to verify the correctness of the patch; the semantic based methods, such as SemFix, DirectFix, Angelix, Nopol etc., exploit semantic information to synthesize patches with symbolic execution and constraint solving techniques; Those research work not belong to the above two categories can be attributed to other types of methods, some classical work of them includes mining Q&A sites and open source projects hosting site to generate patches. The patch evaluation stage is the focus of APR. In recent two years, researchers have paid more attention to the distribution of the correct patches in the search space and the correctness of the patch. Especially, further exploration has been made to pursue lower verification overhead and higher quality of patch. In addition, this survey summarizes APR methods in some specific domains, such as concurrent defects, database defects, null pointer defects, data structure defects, and memory leak defects. Among these specific areas, we specifically analyzed APR methods for concurrent defects, which can be classified into automatic repair of data race, atomicity violation, order violation and deadlock. Moreover, the paper summarizes the commonly used defect benchmarks in the empirical evaluation of APR. It is not hard to find that ManyBugs, IntroClass, Simens suites and Defect4J are the most popular benchmarks. To assist researchers in comparing other APR methods with their proposed methods, the survey also collects all the available APR tools, including the reference papers and corresponding download links. Next, some active research groups as well as their contributions are summarized to facilitate the timely follow-up of their future research work. Finally, the survey looks forward to the future research work from the five aspects, including fault localization, patch generation and evaluation, defect number and type, APR in specific domains and application of APR in the industry.

Keywords　automatic program repair; software fault localization; search based software engineering; constraint solving; concurrency bug repair

1　引　言

由于需求理解的偏差、开发过程的不合理或开发人员的经验不足等原因，均有可能产生软件缺陷(Software Defect)．软件中的缺陷会在一定条件下引发运行错误，产生异常的结果或行为，严重的情况下甚至会造成不可挽回的巨大损失．为了提高软件产品的质量，研究人员希望通过设计或生成高质量的测试用例集来尝试对被测软件产品进行充分测

试,以尽可能多的检测出内在缺陷,目前常用的自动测试用例生成技术包括动态符号执行、演化测试、组合测试等[1].但目前大部分待测项目的规模都较为庞大,一些研究人员借助软件缺陷预测技术[2],通过挖掘软件历史库,构建缺陷预测模型来预先识别出可能含有缺陷的程序模块,以优化测试资源的分配,进而提高缺陷检测的效率.

在上述缺陷检测的基础上,研究人员又进一步关注了一个更具挑战性的研究问题,即自动程序修复(Automatic Program Repair,APR).当开发人员面对大量缺陷报告无从入手的时候,如果 APR 方法能够成功修复部分缺陷,则可以有效减少开发人员的程序调试时间.因此针对 APR 问题的研究具有极高价值.但 APR 问题的研究也极具挑战性,例如在缺陷修复时,一般要求至少存在一个能够执行到缺陷代码并触发软件失效的测试用例.如果被测程序内部含有大量的选择和循环结构,即使设计一个可执行到缺陷代码的测试用例也可以被规约为可满足性问题[3],而该问题是一个典型的 NP-Complete 问题.

在过去的十年内,国内外研究人员对该问题展开了深入的研究.实践表明:若降低对该问题的求解预期,则仍有大量的研究工作可以去尝试.他们初步得到如下一些共识:

(1) APR 方法不能保证对任何类型的缺陷都可以成功修复.

(2) APR 方法离不开与需要修复程序所配套的高质量程序规约(Specification).目前常用的方法是借助配套的测试用例集来描述程序规约.其中一部分测试用例会执行到缺陷代码并产生失效,而另一部分测试用例则可以执行通过.在这种场景下,程序自动修复的目标就是通过生成相应的补丁以使得所有测试用例均能顺利执行通过.

(3) 当开发人员没有足够的时间完成所有缺陷的人工修复时,借助 APR 方法可以为一些缺陷程序自动生成临时补丁.随后开发人员可以参考这些临时补丁,借助人工方式来进一步提高补丁的质量.

APR 问题是当前软件维护领域中的一个研究热点,其中 Weimer 等人[4]的论文《Automatically finding patches using genetic programming》和 Kim 等人[5]的论文《Automatic patch generation learned from human-written patches》分别于 2009 年和 2013 年获得软件工程领域顶级会议 ICSE 的最佳论文.

为了对该研究问题进行系统的研究学习和分析对比,我们首先以(program ∨ fault ∨ software ∨ bug) ∧ (repair ∨ fix)为主要搜索关键词,在 Web of Science、ACM、IEEE、CNKI 等国内外论文数据库中检索出相关论文.随后我们人工筛选并移除了与该综述问题无关的论文,接着通过查阅剩余论文的相关工作和研究人员的已发表论文列表,来进一步添加之前未被检索到的论文,在人工筛选的过程中,我们保留了针对特定领域的缺陷修复文献,例如针对并发缺陷、数据库缺陷、空指针异常缺陷、数据结构缺陷和内存泄露缺陷等特殊缺陷类型修复的文献.最后,我们确定了从 2006 年到 2016 年 9 月期间发表的或已录用的与该研究问题直接相关的高质量论文,总计 77 篇.其中绝大部分选中的论文发表的会议和期刊都是软件工程领域及系统和程序语言领域内的权威会议或期刊,例如 ICSE 会议(20 篇)、ISSTA 会议(6 篇)、ESEC/FSE(FSE)会议(6 篇)、TSE 期刊(4 篇)、ASE 会议(4 篇)、PLDI 会议(4 篇)、ICSME 会议(2 篇)、ICST 会议(2 篇)、JSS 期刊(1 篇)、OSDI 会议(1 篇)、POPL 会议(1 篇)、SOSP 会议(1 篇)、软件学报(1 篇).

图 1 总结了 2008 年至 2016 年 9 月期间,与自动程序修复相关论文在不同年份中的分布,并针对期刊和会议进行了细分.不难看出从 2014 年起,每年发表的相关论文数都超过了 10 篇.

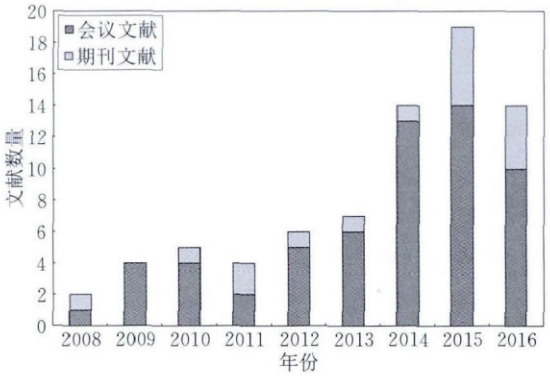

图 1 论文在不同年份的文献数量分布

由于自动程序修复问题是当前软件维护领域的一个研究热点,因此吸引了很多研究人员的关注,并且每年都有不少研究工作发表.目前 Le Goues 等人[6]和玄跻峰等人[7]分别于 2013 年和 2015 年对 APR 问题的已有研究成果进行了系统的总结.其中 Le Goues 等人[6]在 2013 年发表了一篇综述论文,对 GenProg 方法的优劣和当前自动程序修复研究面临的挑战进行了系统的总结.玄跻峰等人[7]在 2015 年从基于搜索、基于代码穷举和基于约束求解

这三个方面对已有的 APR 方法进行了梳理，并总结了学术界针对 APR 问题的两次争议. 基于这两篇综述的总结，我们进一步吸收了国内外该领域的最新研究成果(特别是 2015 年 8 月以来的最新研究成果). 与最新的综述[6-7]相比，首先，在传统缺陷修复问题上，我们补充了 2015 年 8 月以来的最新论文 15 篇，这些研究工作从补丁搜索空间的设定、正确补丁的搜索效率、基于语义的方法的可扩展性问题以及挖掘版本控制系统或问答网站来生成补丁等角度进行了研究，取得了新的成果并有效的推动了该领域的研究进展. 其次，我们对并发程序、Web 应用、数据库应用等特定领域的缺陷修复工作进行了总结，涉及参考文献 23 篇，而这部分内容在之前的两篇最新综述[6-7]中并未提及. 最后我们对自动程序修复领域经常使用的缺陷库和已共享的修复工具进行了总结，这部分内容可以为对 APR 领域感兴趣的研究人员迅速开展研究工作提供帮助.

论文的主要贡献可以总结如下：

(1) 系统分析了 APR 领域近些年来取得的研究成果，通过整理分析和对比总结搜集到的 77 篇国内外 APR 领域内发表的高质量相关论文，提出了 APR 研究框架，该框架主要包括三个阶段：软件缺陷定位阶段、软件生成补丁阶段和补丁评价阶段. 针对这三个阶段，我们分析了其中的研究动机、研究目的、主要问题和现有的解决方案.

(2) 系统总结了并发程序领域的缺陷修复的研究进展，与普通缺陷修复过程类似，我们也根据缺陷定位、生成补丁和补丁评价这三个阶段对已有研究成果进行分析.

(3) 总结了 APR 研究中经常使用的缺陷库和目前已经共享的 APR 工具.

(4) 总结了目前在 APR 领域比较活跃的研究组，并对他们的研究工作进行了总结.

本文第 2 节提出 APR 问题的研究框架，将 APR 问题细分为三个阶段：软件缺陷定位阶段、生成补丁阶段和补丁评价阶段；第 3 节对软件缺陷定位阶段的已有方法进行梳理分析；第 4 节对生成补丁阶段的已有方法进行总结；第 5 节对补丁评价阶段的常见评价准则进行整理；第 6 节对特定领域的自动修复方法特别是并发领域的程序修复进行总结梳理；第 7 节分析实证研究中经常使用的缺陷库和已经共享的 APR 工具；第 8 节列出在 APR 领域比较活跃的研究组，并对他们的研究工作进行了总结；最后总结全文并展望未来可能的研究方向.

2　自动程序修复的研究框架

根据对补丁评估方法的不同，已有自动程序修复方法可以简单划分为两类：基于测试用例的自动程序修复方法和其他类型的自动程序修复方法.

2.1　基于测试用例的自动程序修复方法

目前大部分自动程序修复方法可归为这类方法. 一般来说，这类方法的输入是含有缺陷的程序和与之配套的测试用例集，其中要求配套测试用例集至少包含 1 个可触发缺陷的失败测试用例. 若 APR 方法生成的候选补丁可以使得配套的所有测试用例均能执行通过，则返回该补丁. 本文将这类方法的研究框架细分为三个阶段：(1) 软件缺陷定位阶段；(2) 生成补丁阶段；(3) 补丁评估阶段. 其中软件缺陷定位阶段是自动程序修复的基础，其目标是识别出可能含有缺陷的语句. 目前软件缺陷定位问题仍是软件调试领域中的一个研究热点，相关研究进展可以参考综述论文[8-11]. 生成补丁阶段一般通过预先定义的修改操作对缺陷语句进行修改，代码修改操作在设定时可以考虑修复程序的自身代码、开源项目的代码或者问答网站中的知识等. 补丁评价阶段则对生成的候选补丁进行评估，直到找到一个补丁可以使得所有测试用例均执行通过，并随后借助开发人员的人工分析进行最终确认. 其具体研究框架如图 2 所示. 论文在第 2 节到第 4 节，将依次从这三个阶段对 APR 的已有研究工作进行分析.

2.2　其他类型的自动程序修复方法

上述研究工作借助测试用例来描述缺陷程序的预期正确行为. 也有研究人员提出的自动程序修复方法，并未通过测试用例的执行结果，而是借助基于前置条件/后置条件的契约或缺陷报告等信息来对候选补丁的正确性进行评估.

Dallmeier 等人[12]提出了一种基于对象行为模型(Object Behavior Model)的方法 PACHIKA，该方法通过跟踪程序的执行来获取信息，以一定的抽象粒度提取对象行为模型，对比行为模型的异同从而检测出对象异常行为. PACHIKA 从程序的正确执行中学习方法调用的前置条件，并检查错误执行中不满足前置条件的行为模型. 通过在方法调用前改变程序状态以满足前置条件或删除该方法调用来进行缺陷修复.

Perkins 等人[13]提出了 ClearView 方法. 该方法通过观察程序的正常执行行为，来学习与安全行

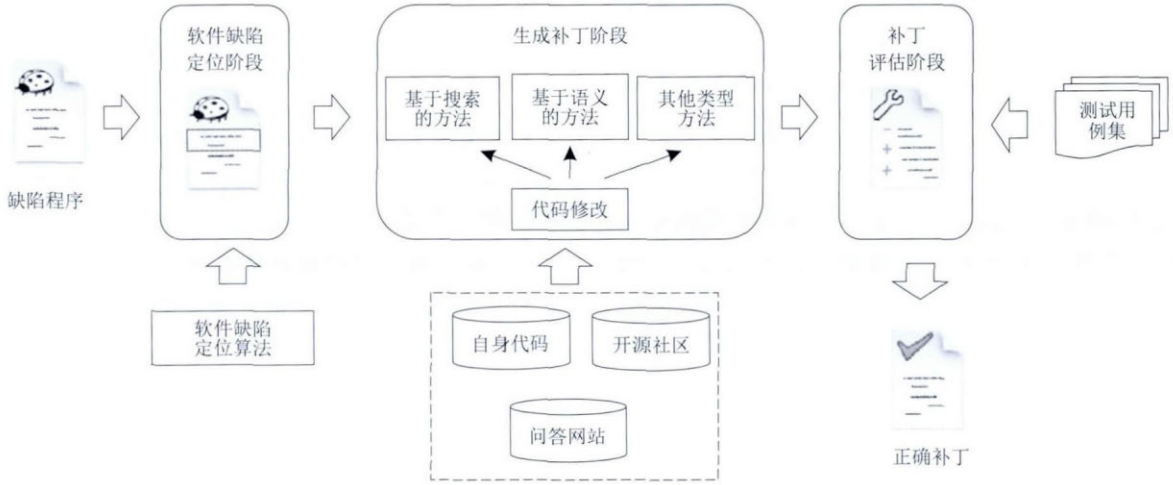

图 2 自动程序修复的研究框架图

为相关的不变式(Invariants).通过部署监控,可以检测出程序崩溃、非法控制转移和越界写缺陷,随后选择一个附加的不变式,来触发缺陷并尝试生成补丁.该方法的特点可总结如下:首先并未借助测试用例的执行信息来生成补丁;其次可以直接生成二进制补丁,因此不需要访问源代码,并且在补丁部署时不需要终止应用程序的运行;最后不需要知道触发该缺陷的测试用例的预期输出结果.

Pei 等人[14-18]提出一种基于契约(Contract)的自动程序修复方法 Autofix,并在基于 Eiffel 的软件项目中验证了该方法的有效性.契约可用于自动缺陷定位和自动程序修复,分析类中的布尔序列可形成基于前置条件、后置条件、中间断言和类不变式的契约,如果程序执行过程中违反任何一条断言,则认为该执行失败.Autofix 通过从程序中抽取执行序列来对程序行为特征进行抽象,并根据语法特点和程序执行信息来定位疑似缺陷语句.随后选择不同的修复模式,运用值替换、表达式替换等方式生成补丁代码.如果应用补丁后的程序可以使所有覆盖缺陷语句的测试用例均成功执行,则接受补丁,最后借助启发式算法生成补丁列表,用户可以自主选择补丁合成到源程序当中.

Liu 等人[19]尝试通过分析缺陷报告来自动生成程序补丁.他们通过深入分析缺陷报告,发现一些缺陷报告由于缺陷无法重现或描述信息不完整是无法修复的,而在剩下的可修复的缺陷报告中,一些缺陷报告由于修改较为复杂(例如需要重新设计算法或增加新的特征)也会导致难以自动生成补丁.因此他们重点关注了三类特殊的缺陷类型(即缓冲区溢出、空指针错误和内存泄露),随后深入分析了针对上述缺陷类型的常见代码修改模式,并提出了 R2Fix 方法.具体来说,首先借助缺陷分类器,对缺陷报告进行解析和分类,若预测该缺陷报告缺陷类型属于上述三种缺陷类型,则随后通过分析缺陷报告和相关源代码抽取出模式参数,例如指针名称、缓冲区长度等.最后借助上述抽取出的模式参数、针对特定缺陷类型的代码修改模式和源代码,自动生成补丁.基于三个大规模项目(Linux kernel、Mozilla 和 Apache),他们发现该方法共自动生成了 57 个正确补丁,其中有 5 个是针对尚未修复的缺陷报告生成的补丁,并且已经被开发人员接纳并提交到代码库内.

Le 等人[20]在评价补丁选择时,重点分析候选补丁与补丁历史库中的相似度,而不是基于测试用例的执行结果.他们的方法包括三个阶段:(1)缺陷修复历史抽取;(2)缺陷修复历史挖掘;(3)程序补丁生成.其中前两个阶段可预先完成.在第一个阶段中,从 GitHub 中的大量项目的版本控制系统中挖掘历史缺陷修复.随后尝试找到经常出现或常用的缺陷修复方式,并将之建模为修改图(Change Graph),这种建模方式具有一般性,并且可以捕获不同的修改类型以及所处的上下文.这些修改图和出现的频率可以构成第三个阶段的知识基础.在第三个阶段,该方法会不断生成候选补丁,根据上述知识基础来进行排序,并通过人工审查的方式判断补丁的正确性.在他们的实证研究中,该方法可以成功修复 23 个缺陷,而 GenProg 方法[21]和 PAR 方法[5]则仅能分别成功修复 1 个缺陷和 4 个缺陷.

3 缺陷定位阶段

软件缺陷定位是自动程序修复过程中的首要步骤,定位结果的精准程度将会直接影响到后续补丁

的生成效率.软件缺陷定位方法[8-11]仍然是当前软件调试领域中的一个研究热点,研究人员提出了大量的自动缺陷定位方法.根据是否需要执行测试用例,已有的缺陷定位方法可以被分为两类:静态缺陷定位方法和动态缺陷定位方法[9].其中静态缺陷定位方法主要通过程序分析等方法来获取被测程序内的控制依赖或者数据依赖关系,在无需执行测试用例的情况下确定缺陷语句在被测程序中可能出现的位置.而动态缺陷定位方法则通过执行测试用例得到其执行信息,通过分析其执行轨迹和运行结果,并基于特定模型来定位缺陷语句在被测程序中可能出现的位置.

基于程序频谱的缺陷定位方法是目前动态缺陷定位方法中的主流.近些年来,一些研究人员对基于程序频谱的缺陷定位方法在实际软件调试中的作用进行了实证研究[22-23].Reps 等人[24]在分析千年虫缺陷时,第一次提出并使用了程序频谱的概念.随后,程序频谱因为其可以有效描述程序的动态行为特征,而被研究人员广泛用于动态缺陷定位方法的研究中.近些年来,研究人员不断提出并使用了可以有效描述程序行为特征的程序频谱,例如:在执行测试用例时程序实体(例如语句、谓词或函数等)是否被覆盖过,覆盖的具体次数,程序实体执行前后的状态,程序实体的执行时间,程序实体间的调用序列等.已有的程序频谱构造方式可以根据程序频谱的构造开销分为三类:轻量级、重量级和其他程序频谱构造方式[9].轻量级程序频谱构造方式在构造程序频谱时,仅简单统计程序实体被测试用例覆盖的信息.重量级程序频谱构造方式通过使用复杂的程序分析方法(例如分析程序实体之间的控制或数据依赖关系)来构建程序频谱.不同于前两类构造方式,其他程序频谱构造方式往往利用其他有效的程序特征信息,比如在构造程序频谱时利用方法的调用序列或程序实体的执行时间等信息.在自动程序修复问题中,为了提高修复效率,研究人员一般采用的是轻量级程序频谱构造方法.

在轻量级程序频谱构造方法中,程序实体的统计数据可以用一个四元组进行表示:$N(s)=\langle n_{ep}(s), n_{ef}(s), n_{np}(s), n_{nf}(s)\rangle$.其中 $n_{ep}(s)$表示覆盖程序实体 s 的成功测试用例数,$n_{ef}(s)$表示覆盖程序实体 s 的失败测试用例数,$n_{np}(s)$表示未覆盖程序实体 s 成功测试用例数,$n_{nf}(s)$表示未覆盖程序实体 s 的失败测试用例数.根据以上信息,可以得出所有执行成功的测试用例数 $n_p=n_{ep}(s)+n_{np}(s)$,所有执行失败的测试用例数 $n_f=n_{ef}(s)+n_{nf}(s)$.程序实体的统计数据交叉表如表 1 所示.

表 1　程序实体的统计数据交叉表

	程序实体被覆盖	程序实体未被覆盖	合计
成功执行	$n_{ep}(s)$	$n_{np}(s)$	n_p
失败执行	$n_{ef}(s)$	$n_{nf}(s)$	n_f
合计	$n_{ep}(s)+n_{ef}(s)$	$n_{np}(s)+n_{nf}(s)$	

一般来说,可疑度(Suspiciousness)是指程序实体内部含有缺陷的可能性,其与该程序实体被成功测试用例覆盖的次数成反比,与该程序实体被失败测试用例覆盖的次数成正比,在程序自动修复问题中,程序实体的可疑度决定了在后续缺陷修复时其被选中的概率.Le Goues 等人[25]和 Kim 等人[5]考虑了一种简单的缺陷定位方法,具体来说:若程序实体 s 的 $n_{ep}(s)$和 $n_{ef}(s)$的取值均大于 0,则将 s 的可疑度设为 0.1;若仅 $n_{ef}(s)$的取值大于 0,则将 s 的可疑度设置为 1;其他情况则均设置为 0.

Qi 等人[26]在他们提出的 Kali 方法中,考虑了另一种简单的语句排序方法,给定程序实体 s_1 和 s_2,他们通过式(1)来计算两者之间的优先级,若返回值为 1,则表示 s_1 的可疑度高于 s_2.

$$prior(s_1,s_2)=\begin{cases}1, n_{ef}(s_1)>n_{ef}(s_2)\\ 1, n_{ef}(s_1)=n_{ef}(s_2), n_{np}(s_1)>n_{np}(s_2)\\ 1, n_{ef}(s_1)=n_{ef}(s_2), n_{np}(s_1)=n_{np}(s_2),\\ \quad n_f(s_1)>n_f(s_2)\\ 0, \text{其他}\end{cases} \tag{1}$$

随后研究人员进一步考虑了软件缺陷定位领域中的一些经典可疑度计算公式.例如文献[3,27-29]考虑了 Tarantula 公式[30],其可疑度计算公式为

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

文献[3,31-34]考虑了 Ochiai 公式[35-36],其计算公式为

$$\frac{n_{ef}(s)}{\sqrt{n_f\times(n_{ef}(s)+n_{ep}(s))}} \tag{3}$$

文献[37]考虑了 Jaccard 公式[35-36],其计算公式为

$$\frac{n_{ef}(s)}{n_f+n_{ep}(s)} \tag{4}$$

另外,在基于语义的缺陷修复方法中还常常借助 Angelic Debugging[38]的缺陷定位方法来辅助生成程序需要满足的约束值.而基于程序切片的定位方法或基于控制/数据依赖分析的定位方法,由于存

在计算开销大的不足，因此在自动修复中使用甚少，在文中不作论述.

大部分研究工作借助在找到真正缺陷语句之前需要审查的代码量来评估软件缺陷定位效果，但该指标并不能反映开发人员复杂的调试行为，Qi等人[39]提出了一个新的评测指标NCP(Number of Candidate Patches)，其返回的是找到有效补丁前需要评估的候选补丁数，他们在GenProg方法的基础上考虑了15种不同的缺陷定位方法[40-41]，结果表明，需要审查代码量较少的SFL方法，并不一定可以取得更小的NCP取值.除此之外，他们发现Jaccard方法跟其他方法相比可以取得更小的NCP取值.

4 生成补丁阶段

已有的补丁生成方法可以简单分为三类：基于搜索的方法(例如GenProg、PAR和SPR)、基于语义的方法(例如SemFix、Nopol和DirectFix)和其他类型的方法(例如SearchRepair、Relifix和Minthint).其中基于搜索的方法在搜索空间内通过搜索生成补丁，并借助配套测试用例集对该补丁进行验证；基于语义的方法则借助语义信息(通过符号执行和约束求解)来合成补丁，而不属于上述两类方法的研究工作被归于其他类型的方法.

4.1 基于搜索的补丁生成方法

这类方法又被称为“生成-确认”方法，其主要受到基于搜索的软件工程(Search Based Software Engineering，SBSE)思想的启发.SBSE[42]思想最早由Harman等人提出，并逐渐成为软件工程领域中的一个热点研究领域.该领域借助基于搜索的方法(即元启发式搜索算法)对软件工程中的组合优化问题进行求解.目前SBSE已经成功应用于软件项目开发的各个阶段，包括：需求分析、项目规划、项目维护和逆向工程等.SBSE日益得到越来越多的研究人员的关注，因为在搜索空间规模较大，且存在多个相互竞争/冲突目标的情况下，SBSE能够提供自动化/半自动化的解决方案.目前SBSE领域重点考虑的元启发式搜索算法包括：模拟退火、遗传算法、蚁群算法以及粒子群优化等.

Arcuri和Yao[29,43]最早在2008年提出了基于协同演化(Co-evolutionary)的方法来自动生成程序的补丁.该方法首先借助形式化规约生成一系列单元测试用例，随后将有缺陷程序与单元测试用例同时进行迭代.在单元测试用例的选择过程中采用基于搜索的软件测试技术，经过多轮迭代后，可以选出具有更强缺陷检测能力的测试用例和正确修复补丁.

Weimer等人[4]在2009年提出的GenProg方法，该方法是自动程序修复研究领域的开创性研究工作.GenProg方法在生成补丁阶段使用基于遗传规划(Genetic Programming)的方法，其首先把源程序转化成对应的抽象语法树(Abstract Syntax Tree，AST)，然后迭代使用交叉算子和变异算子对已有的抽象语法树中的节点进行随机删除、增加或替换，以生成新的AST并转换成对应的程序补丁.随后他们对GenProg方法进行了一系列优化[21,25,44-45].为了进一步评估GenProg方法的实际缺陷修复成本，他们[25]将GenProg部署到亚马逊的EC2云计算平台上，最终结果表明：平均修复一个缺陷仅需要8美元，其修复成本要显著低于开发人员人工修复的成本.该研究工作极大的推动了研究人员对APR问题的研究兴趣.

随后研究人员从语义等价补丁的识别、搜索方法的选择以及代码修改操作的设定等几个角度对GenProg方法进行了优化.

为了尽可能减少生成的候选补丁数，Weimer等人[46]提出了AE(Adaptive Equivalence)方法，该方法基于近似语义等价关系来识别出语义等价的候选补丁，从而可以有效减少需要评估的候选补丁数.而Long等人[47]提出的SPR方法首先对参数化补丁模式(其借助抽象取值来表示分支条件)进行选择.对于某一个模式，SPR方法会预先判断实例化该模式是否存在可以使得所有测试用例都执行通过的可能性.若存在，则通过对模式中的抽象取值进行实例化来生成候选补丁并进行验证.该方法可以有效减少需要生成的候选补丁数.在SPR方法可修复的19个缺陷中，有11个可以在评估第一个候选补丁时就找到正确补丁.Long等人[48]随后提出的Prophet方法，根据代码特征对候选补丁进行优化排序.该方法通过机器学习的模式对开源项目中正确代码的特征进行学习，为多种可用于识别补丁的特征赋以权值形成概率模型.针对搜索空间的候选补丁，分析补丁自身的特征和补丁与周围代码的交互，从而为每个补丁形成二进制特征向量.通过概率模型的计算得出候选补丁可能正确修复程序的概率值，最后根据概率值排序以优先验证正确补丁.

在搜索方法的选择上，Qi等人[49]认为GenProg方法和PAR方法中用到的遗传规划方法存在计算开销大且难以辨识有效补丁的问题，他们在候选补

丁搜索的时候考虑了简单搜索(Random Search)，并提出了 RSRepair. 并且实验结果表明：在大部分实验程序中(23/24)，RSRepair 比 GenProg 方法可以更快找到有效补丁，并且 RSRepair 与 GenProg 方法相比，需要执行的测试用例数更少.

Debroy 和 Wong[27] 提出了基于变异测试的程序修复方法，他们借助变异测试(Mutation Testing)中的变异算子(Mutation Operator)来尝试生成程序补丁. 具体来说，首先借助 Tarantula 缺陷定位方法来定位可疑语句，随后将变异算子应用到可疑语句中，基于 Simens 程序集和 Ant 程序上，初步验证了该方法的可行性.

代码修改操作的设定决定了自动程序修复方法的搜索空间. 目前研究人员在代码修改操作设定时主要考虑两种方式：基于复制或者删除语句的策略和挖掘开源项目的策略.

(1) 基于复制或者删除语句的策略

GenProg 方法在程序修复时基于冗余假设，即缺陷修复可以通过拷贝或者重排已有代码来完成. 该假设的核心是修复代码已经存在于程序的其他地方. Martinez 等人[50] 借助软件时序冗余性(Software Temporal Redundancy)对冗余假设进行了分析. 他们认为一个代码提交时具有时序冗余性是指这次提交的代码可以通过对之前提交的代码进行重新排列来获得. 通过分析六个开源项目，他们发现 31%～52% 的代码提交具有时序冗余性，即大部分时候，软件在封闭世界里进行演化(即没有产生全新的代码). 而 Barr 等人[51] 将该假设称为整形外科手术猜想(Plastic Surgery Hypothesis)，并通过分析来自 12 个大规模 Java 项目的 15723 个代码提交，进一步确认了 Martinez 等人的发现.

(2) 基于挖掘开源项目的策略

Kim 等人[5] 从开源项目 Eclipse JDT 中分析了 62656 个开发人员人工编写的补丁，总结出了常用的 10 种代码修改模板. 基于上述代码修改模板，他们提出了 PAR 方法. 基于他们搜集的 119 个实际 Java 缺陷，实证研究结果表明：PAR 方法可以成功修复其中 27 个缺陷，而 GenProg 方法仅可以成功修复 16 个，而这两种方法均可以成功修复的缺陷仅有 5 个. 除此之外，PAR 方法生成的程序补丁具有更好的可读性. 表 2 罗列了 PAR 中总结的 10 个模板.

表 2　PAR 方法使用的 10 个代码修改模板

模板名称	描述
参数替换	用相同类型的变量或表达式替换方法调用中的参数
方法替换	用有相同参数和返回值类型的方法替换另一个方法调用
参数增加或删除	增加或删除一个方法调用中的参数
表达式替换	用另一个表达式替换条件分支的谓词部分
表达式增加和删除	插入或删除一个条件分支中的谓词项
空指针检查	增加 if()语句，检查一个语句中引用的对象是否为空
对象初始化	在方法调用变量之前插入一个初始化语句
数组下标范围检查	增加 if()语句，检查一个语句中引用的数组下标是否越界
集合范围检查	增加 if()语句，检查一个索引变量是否超出已给集合的范围
类转换检查	插入 if()语句，借助 instanceof 函数来检测被转换的对象是否属于被转换的类型

随后 Soto 等人[52] 重点针对 Java 编程语言，基于对 Github 网站的代码修复信息的挖掘，他们重点分析了表 2 中的参数替换、方法替换、参数增加或删除、表达式增加或删除、对象初始化、空指针检查、数组下标范围检查和集合范围检查这八种修改模板的出现频率. 他们发现上述八种修改模板可以覆盖 14.78% 的实际补丁. 除此之外，他们从基于语句级别的代码修改角度进行了分析，他们发现大部分缺陷通过修改控制流就可完成缺陷修复. 因此他们的研究工作表明：直接将基于 C 的代码修改操作引入到 Java 的代码并不一定会成功. 因为这两种语言的构造并不相同，例如他们认为可能需要引入字段或方法的插入修改操作.

Tan 等人[53] 采取的措施是不提供修改模板来指导修复搜索，因为这会对搜索空间产生限制，并使得修复更加倾向于采用预先提供的修复模板. 因此他们建议使用反模式(Anti-patterns)来禁止一系列转换操作. 为避免验证疑似正确补丁，他们通过人工审查确认疑似补丁的共同特点，并基于遗传规划算法定义了一系列禁止的修改操作，这些操作大多处于控制流图的层面. 即便采用这种修改操作可使补丁通过测试用例集，也并不认为是正确的补丁.

研究人员对基于搜索的方法展开了大量实证研究，例如 Kong 等人[54] 对基于遗传规划的 GenProg 方法[21]、基于随机搜索的 RSRepair 方法[49]、基于自适应的 AE 方法[46] 和改进后的 GenProg 算法进行实验评估. 他们发现在小规模程序上具有很好修复效果的方法在中等规模程序上则变得代价高昂且修复效果不佳. 除此之外，他们发现大部分方法(除了 RSRepair 方法)满足二八原则，即修复方法仅评

估前 20%的候选补丁就可以找到 80%的正确补丁. Zhong 和 Su[55]设计并开发出 BUGSTAT 工具,对补丁进行提取和分析. 他们基于 6 个 Java 项目的 9000 个真实缺陷,通过将手工修复与自动修复进行比较,对缺陷定位的复杂度、程序修复的复杂度、程序修复中需要用到的算子、API(Application Program Interface)的重要性以及程序修复过程中对文件的修改程度等问题进行了实证研究. 其结果表明:超过 80%的缺陷在修复时,最多修改 4 个源文件. 在修复时,增加代码的操作要多于删除代码的操作. 大约 50%的缺陷在修复时不需要进行 API 修复操作,但代码复杂度越高,API 修复操作则越频繁.

基于搜索的自动程序修复方法可看作是从搜索空间中查找补丁再进行验证的过程. 搜索空间和搜索策略是这类方法的关键,研究者对搜索空间的设定和修复效率的关系进行了研究,搜索空间设定过小会导致无法找到最优解,而搜索空间设定过大会导致算法效率太低,同时会生成太多疑似正确的补丁,加大补丁评价的难度. 此外,如何排除疑似正确的补丁并选出更加符合人类认知的修复补丁也是该类方法的研究重点,学术界多次就此类方法的修复效果进行争论,其有效性有待进一步深入研究. 基于搜索的方法是最早提出的一类自动程序修复方法,研究人员已经取得较大进展,在自动程序修复中占有重要地位,同时该类方法也符合当今开源共享和大数据的潮流,值得更进一步深入地研究.

4.2 基于语义的补丁生成方法

除了基于搜索的补丁生成方法以外,也有研究人员深入研究了基于语义的补丁生成方法. 基于语义的补丁生成方法的总体流程如图 3 所示,首先使用缺陷定位方法根据可疑值对程序语句排序,之后生成能使程序通过测试用例的天使值[38](即针对某一语句的输入输出预期值),接着通过收集程序运行时信息提取出补丁需要满足的约束,最后将修复约束作为程序合成(Program Synthesis)的规约,借助约束求解器生成补丁. 程序合成得到的补丁满足所有约束,一般都能通过验证,成为正确的程序补丁.

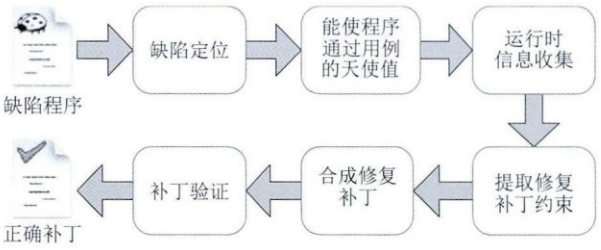

图 3 基于语义的修复方法流程

基于语义的方法根据程序必须满足的约束合成代码,不需要依赖冗余性假设,因此修复的效果普遍要优于基于搜索的修复方法. 同时基于语义的方法对程序本身有较多的分析,使得它的修复能够更有针对性,研究者们据此提出了多种针对特定缺陷的修复方法.

Nguyen 等人[3]提出了 SemFix 方法. 具体来说,该方法首先借助 Tarantula 缺陷定位方法[30]计算出每个语句含有缺陷的可能性,随后从含有缺陷可能性最高的语句开始尝试修复,直至找到可以使得所有测试用例执行通过的补丁为止. 在针对每个可疑语句修复时,他们的方法包括两个步骤,首先针对该语句生成正确的规约,他们借助了天使调试(Angelic Debugging)[38]的思想,针对该缺陷语句的输入,得到可以使得对应测试用例执行通过的输出. 最后基于上述规约,借助基于不同级别的组件(例如常量、加减运算符等)的代码合成方法[56],合成正确的表达式. 为了进一步提升修复性能,他们考虑了两种优化策略:(1)并不是基于所有测试用例来立刻生成约束,而是通过逐个分析测试用例来增量生成约束;(2)在可以合成的候选表达式中,他们将这些表达式按照复杂度从小到大进行排序,这种修复方法有助于快速找到简单修复补丁.

为了进一步提高生成补丁的可读性和可理解性,Mechtaev 等人[57]从简化补丁角度展开了研究. 他们发现生成的补丁是否更为简单与尝试修复缺陷的语句相关. 已有方法在缺陷修复的时候借助软件缺陷定位方法返回的语句可疑度,但可疑度高的语句可能会生成复杂的补丁. 因此他们尝试将软件缺陷定位和程序补丁生成融合成一步并提出了 DirectFix 方法,并将该问题转化成部分最大可满足性问题(pMaxSAT),并借助基于组件的程序合成技术[56]生成补丁.

为了使得基于语义的修复方法可以修复更大规模的缺陷程序,Mechtaev 等人[37]提出了 Angelix 方法. 其在代码合成的时候考虑了一种轻量级约束(即 Angelic Forest),该约束更为简单且与缺陷程序的规模无关,同时又可以包含足够的语义信息来合成程序补丁. 相比于 GenProg 方法和 SPR 方法,该方法可以修复更多的缺陷,除此之外,该方法还成功修复了 Heartbleed 安全漏洞.

Xuan 和 DeMarco 等人[33-34]重点关注了条件语句缺陷(Conditional Bugs)的修复并提出了 Nopol 方法. 其主要通过修改已有的 if 条件语句,或者在

语句(块)前添加前置条件(即 guard 语句)来完成缺陷的修复.具体来说,该方法首先借助天使修复定位(Angelic Fix Localization)来识别测试用例执行时的预期条件取值,随后通过搜集运行时轨迹,来搜集变量和其具体取值.最后借助搜集的数据将之转化为 SMT(Satisfiability Modulo Theory)问题,并通过求解出可行解来将之转换为代码补丁.其在条件合成(Synthesize Condition)的时候借助基于组件的程序合成方法来完成.Durieux 和 Monperrus[58] 进一步针对 Nopol 方法设计了新的基于动态探索的代码合成引擎 DynaMoth,其在程序信息收集阶段使用 Java 提供的调试信息接口得到当前环境中调用方法的返回值,并将它们带入到程序合成方法中生成更多的布尔表达式.因此,DynaMoth 引擎生成的 if 条件补丁中可以包含方法调用,进一步扩大了寻找正确补丁的搜索范围.

Marcote 和 Monperrus[59] 针对无穷循环(Infinite Loop)这一类型的缺陷进行修复,并提出了 Infinitel 方法.他们首先通过代码变换对循环代码插桩,随后设置一个循环次数的最大值来探测程序中可能存在的无穷循环代码,接着枚举得到能使程序通过测试用例的循环次数作为天使值.最后使用与 Nopol 同样的方法,基于收集的程序信息和程序合成方法生成潜在的循环修复补丁.

Gopinath 等人[60] 提出了一种基于规格说明和约束求解的半自动化程序修复方法.他们首先使用缺陷定位方法对程序中可能存在缺陷的位置排序,然后通过生成程序对应的参数化的控制流图(Control Flow Graph,CFG),基于 Alloy 语言[61] 的程序规格描述产生一系列的代码约束公式.最后将程序的前置状态、代码约束和后置条件代入到约束求解器中求解,得到满足所有约束条件的参数值.如果具体化后的 CFG 能够通过所有的输入并产生正确结果,则将该 CFG 转化成最终的补丁程序.该方法能够识别并解决绝大部分的赋值语句错误和分支条件错误,但是对语句缺失引起的错误的修复效果较差,并且修复过程中需要人工参与进行程序的规格描述.

基于语义修复的方法是自动程序修复领域近两年的一个研究热点,基于语义的修复方法综合应用了缺陷定位、天使调试、约束求解、程序合成等方法,这些都是软件工程领域内的重要研究问题.因此,基于语义的修复方法的高准确率与研究人员之前在这些领域内深入的研究工作是分不开的,同时自动程序修复方法的研究也对这些领域内的研究成果进一步向实用转化起到重要的促进作用.另一方面,我们也发现基于语义的修复方法对特定类型缺陷的修复有较好的效果,利用基于语义的方法尝试去修复更多类型的缺陷以及结合已有的缺陷修复方法形成综合修复系统是未来基于语义的修复方法的主要研究方向.

4.3　其他类型的方法

问答网站(例如 StackOverflow)具有覆盖的程序类型多且内容丰富等优点.许多在软件开发中反复出现的缺陷问题已经在问答网站中引起了讨论,且开发人员往往在缺陷修复时,会在第一时间去问答网站上寻求帮助,以得到一些技术解决方案,因此问答网站为软件缺陷修复提供了丰富的资源.Gao 等人[62] 首次利用问答网站来尝试进行缺陷修复.他们首先搜集缺陷程序执行时的崩溃踪迹(Crash Trace),随后他们利用崩溃踪迹信息去问答网站进行搜索并得到相关网页.通过分析从问题贴和答案贴中抽取出的错误代码和修复代码,来生成编辑脚本.最后生成程序补丁.实证研究表明该方法可以有效修复来自 Android 项目中的复发崩溃缺陷,其中很多缺陷借助基于搜索的方法是无法修复成功的.

Ke 等人[28] 提出了 SearchRepair 方法.他们认为可以借助语义代码搜索(Semantic Code Search)[63],从大量开源项目托管网站(例如 GitHub、BitBucket 或 SourceForge)上找到与缺陷代码在语义上具有一定相似度的代码段.具体来说:首先将开源项目中大量人工编写的代码段编码为基于输入输出行为的 SMT 约束,并存储到数据库中.其次借助软件缺陷定位方法定位到可能含有缺陷的代码,并分析出其预期的输入输出行为.接着借助最新的约束求解器,从数据库中搜索出可满足预期行为的代码段,并尝试将之嵌入到缺陷代码中(例如通过变量重新命名等),并生成补丁.最后通过测试用例的执行结果来验证补丁是否正确.基于 IntroClass 缺陷库的实证研究结果表明:该方法可以修复 778 个缺陷中的 150 个缺陷,其中有 20 个缺陷不能被 GenProg 方法[21]、RSRepair 方法[49] 和 AE 方法[46] 修复.

软件在开发和维护过程中,因移除缺陷或增添新的功能等原因会执行代码修改并引发软件演化.回归测试旨在有效保证修改后代码的正确性并避免代码修改对被测程序其他模块产生副作用.Tan 和 Roychoudhury[32] 重点对软件演化过程中产生的缺陷(简称回归缺陷)的自动修复问题进行了研究.与传统的缺陷修复问题不同,回归缺陷修复的挑战是:修

改补丁需要在修改后版本中尽可能多的保留修改前的功能,同时又能够重现在修改前版本上的回归测试行为.针对该问题,他们人工分析了来自 CoREBench 库[64]中的 73 个实际软件回归缺陷,并提取出了 14 种代码转换操作.随后他们提出了 relifix 方法,该方法首先借助 Ochiai 缺陷定位方法,将语句按可能含有缺陷的概率从大到小进行排序,随后依次在可疑语句上应用代码转换操作,直至生成的补丁可以在配套测试用例集上执行通过.实证研究结果表明该方法比 GenProg 方法可以修复更多的回归缺陷,且引入新的缺陷的概率要低于 GenProg 方法.

Kaleeswaran 等人[31]提出一种半自动化的修复方法 Minthint,该方法并未考虑直接生成程序补丁,而是尝试生成修改提示来协助开发人员人工生成补丁.具体来说,他们首先借助软件缺陷定位方法识别出含有缺陷可能性高的语句,并依次借助 Minthint 方法尝试得到修复提示.针对某个可疑语句,该方法首先借助动态符号执行得出一个状态转化,其本质是一个函数,输入是每个测试用例在执行到该语句前的程序状态,输出是执行完该语句后的预期正确程序状态.针对程序的修补空间,借助统计相关分析,计算出每个表达式可修复该语句的概率并进行排序.随后将概率高的表达式与缺陷代码进行模式匹配并生成代码修改提示.

4.4 补丁生成方法总结

该小节对已有的补丁生成方法进行总结.基于搜索的自动程序修复方法发展最早,主要尝试利用已有代码替换缺陷代码来进行修复,GenProg 属于基于搜索的自动程序修复方法的开创性工作,之后 PAR、AE、RSrepair 等方法不断涌现.此类方法在单缺陷修复上取得一定的成效,但由于搜索空间庞大、效率有限导致程序修复的比率不高,同时补丁的可读性和准确性上也有待进一步改进.另外,当程序中存在多缺陷时会让搜索空间呈指数级增长,因此,此类方法在多缺陷修复问题上的表现较差,有待进一步探索.不同于基于搜索的方法,基于语义的方法主要是通过约束条件来合成代码,因此可有效提高补丁的准确性和有效性,但所修复程序本身需要满足的条件也更多.目前这类方法已经对赋值语句、谓词、多缺陷程序等情况下的缺陷修复做出尝试,因此对缺陷分类后使用相应的修复方法能够进一步提高修复的成功率.随着该领域的发展,研究者们在自动程序修复的效果、缺陷类型、缺陷复杂度等方面均进行了相关的实证研究.表 3 对已有的补丁生成方法从方法名称、发表年份、方法类型、实验程序使用的编程语言、针对的缺陷类型以及主要特点等方面进行了总结.

5 补丁评价阶段

随着补丁生成方法的不断涌现,准确评价生成的补丁成为自动修复领域内的热点问题.补丁评价工作的进展有助于进一步比较已有补丁生成方法的效率并对今后的发展方向做出指导.近两年研究人员更关注正确补丁在搜索空间中的分布和补丁的正确性.特别是针对减少补丁的验证花销和提高补丁质量的问题做出了进一步的探索.

5.1 补丁评估的优化方法

补丁的评估是使用测试用例集验证候选补丁质量的过程,是自动程序修复中代价较为高昂的一个环节,其开销主要集中在测试用例执行阶段.对补丁评估阶段的优化可以尽早的验证正确的补丁,减少不正确补丁的验证开销,研究人员提出了很多优化方法来减少上述开销.

Qi 等人[49,65]借助测试用例优先排序技术来减少候选补丁的评估时间.他们提出了 NCP 指标,该指标返回的是自动程序修复方法在找到有效补丁前需要评估的候选补丁数.与 GenProg 方法相比,他们提出的 RSRepair 方法的 NCP 取值更小.

Long 和 Rinard[47] SPR 方法中使用了一系列借助人工编码的启发式排序方法.他们将生成的候选补丁按照如下方式进行排序:(1)仅修改分支条件的补丁;(2)在语句块内首个语句前插入 if 语句的补丁;(3)在语句附近插入 if-guard 的补丁;(4)在语句块内的首个语句之前执行替换语句等操作的补丁;(5)剩余所有补丁.而对于同一类的补丁,则基于语句缺陷定位效果进行排序.Long 等人[48]在随后提出 Prophet 方法考虑与 SPR 方法相同的搜索空间,但基于历史补丁库的挖掘,获得一个概率模型,该模型可以为搜索空间内的候选补丁设置一个概率值,并借助该概率值进行排序对候选补丁进行优化.实证研究表明:Prophet 的缺陷修复成功率相比已有的方法有显著的提升.Martinez 等人[66]深入挖掘了 14 个开源项目的软件仓库,他们基于抽象语法树级别对 89 993 个缺陷修复代码进行了分析,他们发现并非所有的修改操作都具有相同的概率,即一些修改操作能成功修复缺陷的概率更高.因此这些信息可以在修复空间搜索过程中被充分利用,从

而提高自动程序修复方法的补丁生成效率.实证研究结果表明:Prophet 的缺陷修复成功率相比已有的方法有显著的提升.

由于候选补丁的数量众多,在评估过程中时间花销较大,目前的候选补丁优化方法主要关注测试用例集的优化、候选补丁特征识别两方面.前者优先执行补丁判别能力较强的测试用例,削减错误补丁的评估开销.后者优先验证正确特征较高的候选补丁,从而更早地识别出正确的补丁.

表 3　已有补丁生成方法总结

修复方法	参考文献	发表年份	方法类型	实验程序	针对缺陷类型	特点说明
GenProg	[4,21,25,44-45]	2009	基于搜索	C 程序	通用	基于抽象语法树 使用遗传规划算法进行搜索
Debroy 和 Wong	[27]	2010	基于搜索	C 程序和 JAVA 程序	通用	使用变异方法生成补丁
Gopinath 等人 借助 SAT 修复	[60]	2011	基于语义	JAVA 程序	赋值语句错误 和分支条件错误	使用 Alloy 工具集来描述约束
JAFF	[29,43]	2011	基于搜索	JAVA 程序	通用	程序修复中运用搜索算子
PAR	[5]	2013	基于搜索	JAVA 程序	通用	借助代码修改模板来提高补丁的可读性
AE	[46]	2013	基于搜索	C 程序	通用	借助语义等价关系来减少生成的补丁数
SemFix	[3]	2013	基于语义	C 程序	通用	首个基于语义的修复方法
RSRepair	[49]	2014	基于搜索	JAVA 程序	通用	使用随机搜索策略代替遗传规划算法
Minthint	[31]	2014	其他类型	C 程序	通用	采用基于模式匹配的合成算法生成修复提示
Kali	[26]	2015	其他类型	C 程序	通用	通过移除代码进行程序修复
SPR	[47]	2015	基于语义	C 程序	通用	通过参数化补丁来减少生成的补丁数
Prophet	[48]	2015	基于搜索	C 程序	通用	采用机器学习方法来提高正确补丁的搜索效率
DirectFix	[57]	2015	基于语义	C 程序	通用	缺陷定位与补丁生成合为一步,尝试生成更简单的补丁
Relifix	[32]	2015	其他类型	C 程序	通用	首个针对回归测试错误进行修复的方法
SearchRepair	[28]	2015	其他类型	C 程序	通用	借助语义代码搜索生成补丁
QACrashFix	[62]	2015	其他类型	Andriod 应用	崩溃缺陷	基于问答网站来生成补丁
Infinitel	[59]	2015	基于语义	JAVA 程序	无穷循环	针对无穷循环的修复
Nopol	[33-34]	2016	基于语义	JAVA 程序	If 条件语句错误 及前置条件错误	针对条件语句缺陷的修复
DynaMoth	[58]	2016	基于语义	JAVA 程序	If 条件语句错误 及前置条件错误	基于 Nopol 使用了新的代码合成引擎
anti-patterns	[53]	2016	基于搜索	C 程序	通用	基于反模式来生成补丁
Angelix	[37]	2016	基于语义	C 程序	通用	可修复更大规模缺陷程序的基于语义的方法

5.2　补丁质量的评估指标

早期的研究工作假设若生成的程序补丁在所有测试用例上均能成功执行,则认为该补丁成功修复了缺陷.例如 Le Goues 等人[25]提出的 GenProg 方法可以成功修复 105 个缺陷中的 55 个.Qi 和 Mao 等人[49]提出的 RSRepair 方法可以成功修复全部 24 个缺陷(他们从 GenProg 方法可以成功修复的 55 个缺陷中选择了 24 个).Weimer 等人[46]提出的 AE 方法可以成功修复 105 个缺陷中的 54 个.但随后 Qi 等人[26]通过人工分析后,却发现 GenProg 方法仅成功修复了 2 个缺陷,RSRepair 方法仅成功修复了 2 个缺陷,AE 方法仅成功修复了 3 个缺陷.他们将能在所有测试用例上均执行成功的程序补丁称为疑似正确的补丁(Plausible Patch),但疑似正确的补丁并不一定是正确补丁,他们将造成疑似正确补丁并非正确补丁的原因进行了总结.首先实验脚本内存在的问题会造成这些补丁不是正确补丁.其次缺陷程序配套的测试用例集存在不完整性问题,即测试用例集并未对软件的所有功能进行覆盖.因此一些补丁通过删除部分代码,可以使得所有测试用例均能执行通过,但这些补丁并不是正确补丁.基于上述观察,他们提出了 Kali 方法,该方法仅考虑了功能删除,其主要操作包括:删除语句(块),将 if 语句中的条件用 true 或 false 进行替代,在函数体内插入简单的 return 语句.结果表明该方法可以成功修复的缺陷数与 GenProg 方法、RSRepair 方法和 AE 方法相当.Kali 方法的提出并不是真正用于修复缺陷,而是旨在检测配套测试用例集是否存在不完整性问题.

随后 Long 和 Rinard[67]基于 SPR 方法[47]和 Prophet 方法[48],对补丁的搜索空间特征进行了更为深入的分析.他们的分析结果表明:在搜索空间内,疑似正确的补丁较多,但正确补丁却非常稀缺.

例如在来自 8 个开源项目的 69 个缺陷中,仅有 19 个缺陷在 SPR 方法和 Prophet 方法的搜索空间内含有正确补丁.而在 GenProg 方法和 RSRepair

方法的搜索空间内,仅有 1 个缺陷有正确补丁,在 AE 方法和 Kali 方法中,仅有 2 个缺陷有正确补丁.虽然已有研究表明:若搜索空间内正确补丁太少,会影响到自动程序修复方法的修复效果.但通过考虑更多的可疑语句或更多的代码修改操作,虽然可以有效增加搜索空间的规模,并提高疑似正确补丁和正确补丁的数量,但也会降低自动程序修复方法从疑似正确补丁中隔离出正确补丁的能力.

随后 Martinez 等人[68]在基于 Java 的 Defect4J 缺陷库[69]上对已有缺陷修复方法的有效性进行了重新验证.他们考虑了 GenProg 方法[21]、Kali 方法[26]和 Nopol 方法[33-34],其中前两种方法采用了 Java 编程语言进行了重新实现.结果表明:在 Defect4J 的 224 个缺陷中,这三种缺陷修复方法可以针对 35 个缺陷生成疑似正确的补丁,其中 Nopol 方法、GenProg 方法以及 Kali 方法分别可以针对 35 个缺陷、27 个缺陷、22 个缺陷生成疑似正确的补丁.在生成的 84 个疑似正确补丁中,他们随后通过人工分析,发现其中 11 个是正确补丁,61 个是错误补丁,而剩余的 12 个补丁还需要借助领域专业知识进行确认.他们的发现与 Qi 等人的发现[26]保持一致.

Smith 等人[70]对自动程序修复方法中的过拟合问题进行了分析,他们将配套测试用例划分成不相交的两部分,一部分用于生成程序补丁,另一部分用于补丁评价.借助机器学习领域的概念,他们将前一部分测试用例称为训练集,将后一部分测试用例称为测试集.若生成的程序补丁可以在训练集上执行通过,而在测试集上不能执行通过,则称该补丁存在过拟合问题.他们基于 IntroClass 缺陷库,确认了 GenProg 方法[21]和 RSRepair 方法[49]中均存在过拟合问题.

Jiang 等人[71]提出了 MT-GenProg 方法,该方法结合了蜕变测试(Metamorphic Testing)[72]及 GenProg 方法[21],他们提出了一个框架将二者进行集成,并通过实验与 GenProg 进行对比,实验结果表明该工具可以有效修复程序中的缺陷.给定已有的测试用例,蜕变测试可以根据一定规则生成一组新的测试用例,随后执行这些测试用例,并检查这些测试用例的输出与已有测试用例的输出,通过判断是否满足一定性质,以确定被测软件是否存在缺陷.上述性质在蜕变测试研究中又被称为蜕变关系(Metamorphic Relation).现有的程序自动修复技术都是依赖一个测试用例预言(Test Oracle)来判断每个测试用例的执行结果是否成功,因此当测试用例预言不存在时会限制程序自动修复技术的执行.由于蜕变测试的整个执行过程不需要测试用例预言,因此应用蜕变测试可以有效缓解当前程序自动修复领域很多测试用例并不存在测试用例预言的问题.

补丁的非功能性属性主要包括可维护性、可读性以及是否有助于开发人员提高软件调试效率等.

Fry 等人[73]组织了 150 个参与者,针对 32 个实际缺陷和 40 个补丁进行了分析.他们发现自动生成的补丁在可维护性上要稍弱于人工编写的补丁,但如果在自动生成的补丁中添加一些具有可读性的文档(例如对补丁的效果和上下文进行描述)时,则会得到相反的结论.

Tao 等人[74]则分析了自动生成的补丁是否可以有效辅助开发人员进行软件调试.他们提供了三种辅助线索:缺陷位置、高质量补丁和低质量补丁.他们组织了 95 个参与人员做了一组对照试验,这些开发人员来自研究生和软件工程师等.结果表明:高质量的补丁可以显著提升软件调试的效率,缺陷难度越高,其提升程度越显著;相反低质量的补丁则会显著降低软件调试的效率.除此之外,他们发现辅助线索的类型对软件调试的时间影响不大,而开发人员的经验和缺陷的难度与软件调试的时间密切相关.

Tao 等人[75]从 Eclipse 和 Mozilla 项目中,提取出 300 个不合格的人工生成的补丁,并通过分析这些补丁总结出 12 类不合格的补丁类型,以及通过对开发人员进行在线问卷调查,总结出补丁更容易被驳回的原因,以此来帮助开发人员写出更容易被接受的补丁.该研究工作可以用于辅助评估 APR 方法自动生成的补丁质量.

Le 等人[76]对缺陷修复的时间进行预测,若发现 APR 方法需要修复缺陷的时间太长,他们建议还是由相关开发人员来完成该缺陷的修复.以 GenProg 方法作为分析对象,他们抽取了可能影响到修复效率的相关特征,结果表明该方法可以有效识别出 GenProg 方法难以修复的缺陷.

由于程序规约难以获取,补丁正确性的验证仍处于探索阶段.基于测试用例集的自动程序修复方法主要采用测试用例集作为补丁正确性验证的主要工具,通过判断实际输出与预期输出的一致性来统计测试用例的执行情况.但实际输出与预期输出的一致性判断仍需要进一步提高准确性.并且由于测试用例集的不完整性,使得通过验证的补丁可能在运行其他测试用例时执行失败,因此,完善测试用例集和寻找其他补丁判定指标是提高补丁正确性验证的重要思路.此外,近两年也有一些研究人员对补丁的功能性属性如可读性、可维护性等进行了实证研

究，基于现有的研究成果，未来可进一步关注修复过程中文档的生成.

6　特定领域的自动程序修复方法

6.1　并发领域的自动程序修复方法

随着计算机多核时代的来临，多线程程序变得越来越普遍，并发程序缺陷逐渐成为软件可靠性的重大威胁，并发程序缺陷的修复也越来越成为程序自动修复领域的研究热点. 由于导致并发程序缺陷产生的原因来自于不同线程之间的交错执行(Thread Interleaving)，具有不确定性，因此并发程序缺陷的修复极具挑战. 在正确修复软件中原有的缺陷的前提下，一个正确的补丁应该避免引入新的缺陷，且不应该对程序的性能造成不必要的影响或降低程序的可读性.

由于并发程序不同线程之间的执行顺序随机性较大，并发程序缺陷不一定会在程序每一次执行中都被触发，因此仅通过测试用例不能够充分地触发和检测到并发程序缺陷. 并发程序缺陷的修复需要首先通过控制并发程序的执行，尽最大可能使并发程序缺陷暴露；随后准确地定位到并发程序缺陷产生的位置并检测到其出错类型；最后针对不同类型的并发程序缺陷，在保证程序的正确性和执行效率的前提下对缺陷进行修复. 常见的并发程序缺陷修复方法主要分为通过静态分析添加同步设施和通过运行时措施控制线程调度，两类方法分别为针对不同类型的并发程序缺陷提出. 生成的补丁需要进行正确性测试，再将相关的补丁进行适当的合并，最终保留可读性强且不严重影响程序性能的补丁. 图 4 给出了并发程序缺陷自动修复框架，其大致描述了并发程序缺陷自动修复的主要步骤. 常见的并发程序缺陷主要有四大类，即数据竞争(Data Race)、原子性违背(Atomicity Violation)、顺序违背(Order Violation)以及死锁(Deadlock). 当然并发程序缺陷不仅仅局限于以上四类，且随着研究人员对并发程序缺陷的研究更加深入透彻，许多新的并发程序缺陷类型也被陆续提出.

图 4　并发程序缺陷自动修复框架

(1) 数据竞争的自动修复

数据竞争是并发程序缺陷中的一个主要类型，主要指的是至少两个不同线程针对同一个共享变量进行并发地访问，且其中至少一个为写访问，针对该问题的检测及修复工作开展地较早. 修复数据竞争的主要方法是使对某个全局共享变量的来自于不同线程的访问(其中至少一个为写)形成确定的访问顺序. CFix[77]是一个集成的并发程序缺陷修复系统，针对数据竞争的修复，CFix 提出了两种可能的修复策略：① 形成一种顺序关系，使得一个指令总是在另一个指令结束之后执行；② 为某一特定代码区域添加互斥锁.

(2) 原子性违背的自动修复

原子性违背是指对某一共享变量的一系列访问序列，被来自其他线程的访问打破原子性. 原子性违背的主要修复方法是通过添加同步设施，为某一可能发生原子性违背的代码区域添加锁保护或形成某种确定的顺序关系. AFix[78]是一个自动化修复单变量原子性违背并发程序缺陷的工具. AFix 利用已有的并发程序缺陷检测工具得出的检测报告，通过静态分析并辅助运行时动态代码监控，成功修复了 8 个真实原子性违背缺陷中的 6 个，并有效地降低了其余 2 个缺陷的失败概率. CFix 将 CTrigger[79]作为原子性违背检测工具，并根据 CTrigger 得出的缺陷检测报告，给出所有可能消除原子性违背的解决方案，即保证执行某种特定顺序关系或者为可能发生原子性违背的代码区域添加锁保护. HFix[80]通过对 77 个真实的人工生成的并发程序缺陷补丁进行了深入研究，发现人工补丁中对于原子性违背的修复大多通过添加锁，HFix 则利用程序中已经存在的同步设施操作来修复原子性违背.

(3) 顺序违背的自动修复

在真实的非死锁并发程序缺陷中，顺序违背大约占有三分之一. 各个线程之间交错执行的顺序的不确定性，导致不同线程对共享变量的访问顺序没有按照预期执行，导致顺序违背问题的发生. CFix 将 ConMem[81]作为顺序违背的检测工具，针对 ConMem 检测出的悬挂资源和未初始化的读问题这两种不同顺序违背的类型，CFix 分别提出执行 allA-B 顺序关系和 firstA-B 顺序关系来解决. 根据实证研究结果，HFix 通过增加线程合并操作，替代信号量来保证顺序关系，此外，HFix 也采用了利用程序中已经存在的同步设施操作来修复顺序违背. 经过实验验证，HFix 可修复大量不同类型的并发程序缺陷，且生

成的补丁更简单.

(4)死锁的自动修复

死锁是一种特殊的并发程序缺陷,它是指至少两个线程在执行过程中由于资源竞争造成彼此互相等待的问题.死锁由同步引起,很多和并发程序缺陷相关的方法都是把死锁和其他并发程序缺陷进行分开对待[82-85].例如:ConcBugAssist[82]从数据竞争、违反原子性和违反执行顺序这三个方面进行分析,将基于语义的方法应用到并发程序修复中.Lu 等人[83]对 105 个真实的并发程序缺陷进行了实证研究,并分别针对死锁缺陷和非死锁缺陷得出了不同的研究结论;Park[84]认为死锁和非死锁并发程序缺陷有着不同的性质,需要不同的方法来调试和修复,并只针对非死锁并发程序缺陷提出了新的调试技术;Surendran 等人[85]针对结构化并行语言中的数据竞争问题,提出了一个静态和动态结合的以测试驱动的修复技术.而 AFix、HFix 等方法则无法修复死锁.

Cai 等人[86]提出了针对死锁的自动修复方法 DFixer.区别于已有的并发程序缺陷修复方法,该方法仅仅选取一个线程而不是全部线程进行修复,采取锁的提前获取技术,来破坏形成死锁的必要条件.即持有并等待某个锁,从而可以达到修复死锁的目的,且保证不引入新的死锁.实验验证表明,DFixer 与之前的死锁修复方法对比,在修复成功率更高的情况下,只需引入很少的额外计算开销.

此外,还有一些修复方法与统计学方法相结合,取得了不错的效果.CCI[87]借鉴 CBI[88-89]的思想精髓,在程序运行中监测与程序交错相关的谓词的执行信息,通过采样保证较低的开销,利用统计学模型来对收集到的运行信息进行处理从而确定导致程序失败的根本原因.

并发程序缺陷的修复极具挑战,而多线程程序的普及也使得并发程序缺陷的修复变得尤为重要.并发程序缺陷的自动修复研究已经取得了丰硕的成果,但仍然有一些需要未来研究关注的重点,如开发可修复各种类型缺陷的综合性工具,可借助硬件的支持进行并发程序缺陷的修复,此外,一般缺陷的修复方法也可与并发程序缺陷的修复方法进行相互借鉴.

6.2 其他领域的自动程序修复方法

除了并发缺陷之外,数据库缺陷、空指针异常缺陷、数据结构缺陷和内存泄露缺陷等较为细致的程序缺陷同样得到了研究人员的广泛关注.

Durieux 等人[90]提出 BanditRepair 系统,可以系统探索和评估一系列运行时补丁.所谓运行时补丁是指在程序运行时通过对执行状态的修改来修复空指针解引用(Null Dereference)这一类缺陷,而不会使得整个系统崩溃的修复补丁.BanditRepair 借助在线机器学习算法进行运行时修复,并首次对空指针解引用的修复搜索空间的特征进行了描述,并系统的展开了实证研究.实验通过对 16 个空指针解引用领域的缺陷进行修复,共确定生成了 8460 个不同的有效运行时补丁.

Cornu 等人[91]提出针对 Java 中空指针异常的修复工具 NPEfix,该工具在异常发生之前检测到异常,并在运行时通过代码转换来实现针对空指针异常提出的 9 种修复策略,从而达到程序自动修复的目标.为了验证 NPEfix 的修复能力,他们实施了两组实验,分别针对 11 个真实的缺陷和在 3 个开源程序中植入的 519 个缺陷,实验结果表明 NPEfix 可以成功修复 11 个真实缺陷中的 10 个及 519 个植入缺陷中的 318 个.

Gao 等人[92]解决了许多极具挑战的问题,例如过程间泄漏、全局变量及来自于多分配的泄漏等,并将针对所有问题的解法集成到一个方法中.除此之外,他们针对 C 程序中与内存泄露相关的缺陷,提出了 LeakFix 方法.该方法能够成功修复大量的内存泄漏问题,并且可以被应用于大型程序中.相比于一般缺陷的自动修复,针对内存泄漏的自动修复问题的形式更简单,其主要修复方式是在程序的合适位置插入存储单元分配语句.

Long 等人[93]开发了能自动修复运行时错误的 RCV 系统,该系统能让因遭遇运行时错误终止的程序重新运行,目前主要针对除零错误和空指针引用错误这两类缺陷类型.

Gopinath 等人[94]提出一种全新的数据驱动方法,自动修复数据库语句中选择条件语句中的缺陷.针对给出的错误数据库程序和输入数据,该方法结合机器学习和组合搜索来确定引起错误的数据的正确行为,学习正确行为得到的决策树会给出一种合理的修复建议.

Ocariza[95]等人重点关注了 Web 应用中与 DOM(Document Object Model)有关的一类 JavaScript 代码缺陷,他们收集并分析了 190 个真实的缺陷修复报告,得到针对这类缺陷最常用的修复策略,并以此为基础提出了 VEJOVIS 工具,该工具可在 Web 应用调试阶段为开发人员提供修复建议,给出可替

换出错 DOM API 方法、属性中的参数或赋值的 DOM 元素. 对于 22 个真实的缺陷，VEJOVIS 可以准确给其中的 20 个提供修复建议，并有 13 个缺陷的正确修复在所有修复建议中排名首位.

Samirni 等人[96]提出了 PHPQuickFix 和 PHP-Repair 工具，这些工具借助字符串约束求解方法来自动修复 PHP 程序. 他们将 HTML 生成错误问题描述为字符串约束问题，PHPRepair 借助动态修复方法针对 HTML 网页中的常见错误进行修复，使得修复后的程序可以通过验证测试用例集.

Demsky 等人[97]根据程序中一系列数据结构一致性约束，提出了自动检测和修复数据结构中缺陷的系统，该系统可以自动地检测数据结构中的不一致性，通过将违背的约束转化为析取范式，自动地修复不一致的数据结构，使程序可以继续成功运行. 之后他们又提出了一个新的数据结构修复系统[98]，开发人员只需提供一个关于数据结构一致性约束的规格说明，系统即可将关键一致性约束转换为抽象模型中的集合和关系表达，并在程序执行过程中自动检测及修复对一致性约束的违背，从而使程序成功运行.

Sidiroglou-Douskos 等人[99]提出了 CodePhage，其通过从被修复应用中定位缺陷，并从其他应用迁移代码来移除缺陷的方法. 其可修复的缺陷类型包括整数溢出、缓冲区溢出等.

特定领域及特定类型缺陷的自动修复给程序自动修复研究提供了新的思路，并且取得了有效的修复成果. 专注于特定领域的自动修复工作需着重考虑特定领域缺陷的特征，给出更具针对性的修复建议，并避免引入新的缺陷，基于现有的研究思路，研究人员在未来可更多关注如何提高修复的准确度及综合性的修复方法的开发.

7　缺陷库和自动程序修复工具

7.1　缺陷库

对自动程序修复方法的效果的有效评估，离不开高质量的缺陷库. 高质量的缺陷库一般需要具备如下特征：(1) 缺陷需要来自实际项目(而不是人工植入)；(2) 缺陷所在的评测程序需要具备一定的规模；(3) 测试用例集(或其他形式的程序规约)能较好地覆盖程序的预期行为、有较高的代码覆盖率和正确的测试用例预言(Test Oracle).

论文对实证研究中经常采用的并已经共享的缺陷库进行了总结，最终结果如表 4 所示，该表总结了使用的编程语言、首次使用时间、累计使用次数以及相关文献等，随后我们将依次对这些缺陷库的特征进行简要介绍.

表 4　自动程序修复方法评测程序集统计

缺陷库名称	编程语言	总缺陷数	首次使用时间	累计使用次数	相关文献
ManyBugs	C	185	2009	8	[4,21,25,37,45,47,49,100]
IntroClass	C	778	2015	3	[28,70,100]
Simens 程序集	C	90	2013	3	[3,27,57]
Defects4J	JAVA	357	2015	2	[20,68]

SIR 库中的 Simens 程序集含有不同类型的应用程序，从空中交管系统(Tcas)、调度工具(Schedule 和 Schedule2)到字符串和文件操作程序(Replace 和 Grep). 每个应用程序都配套了大量测试用例和多个缺陷版本，每个缺陷版本均借助人工缺陷注入方式生成.

Le Goues 等人[100]整理并共享了两个基于 C 编程语言的缺陷库 ManyBugs 和 IntroClass①. 这些缺陷库可用于评估自动程序修复方法的效果. ManyBugs 和 IntroClass 的测试集包括黑盒测试集和白盒测试集，黑盒测试集有人工方式依据等价类划分执行. 白盒测试集实现测试预言的分支覆盖.

ManyBugs 缺陷库共搜集了来自 9 个大规模开源项目(fbc、gmp、gzip、libtiff、lighttpd、php、python、walgrind、wireshark)的 185 个缺陷. 其通过分析版本控制系统来挖掘缺陷，总的来说该缺陷库共含有 590 万行代码和 1 万多个测试用例.

IntroClass 缺陷库来自在一门本科生课程中布置的 6 个 C 代码题目，该缺陷共含有 200 个学生提交的 956 个缺陷，这些缺陷被分为两组，其中第一组含有 762 个缺陷，这些缺陷在导师配套的黑盒测试用例集上不能完全通过，第二组含有 810 个缺陷，这些缺陷在借助 KLEE 工具[101]自动生成的白盒测试用例集上不能完全通过.

Defects4J 缺陷库②由 Just 等人搜集并进行了共享[69]. 该缺陷库通过挖掘五个 Java 项目(即 Commons Lang、JFreeChart、Commons Math、Joda-Time 和 Closure Compiler)，共抽取出 357 个缺陷并经过了严格的同行评审，是目前规模最大的 Java 缺陷库. 该缺陷库中的每个缺陷均含有配套测试用例集，且其中至少存在一个可以触发缺陷的失败测

① http://repairbenchmarks.cs.umass.edu
② https://github.com/rjust/defects4j

试用例.

7.2 自动程序修复工具

为了更好的支持实证研究的重现,很多研究人员对自动程序修复工具进行了共享.论文对这些工具进行了搜集,具体结果如表5所示.表中罗列了工具的名称、相关参考文献和工具的下载地址.

表5 自动程序修复工具总结

工具	参考文献	下载地址
GenProg 工具	[21]	http://dijkstra.cs.virginia.edu/genprog/
AutoFix 工具	[17]	http://se.inf.ethz.ch/research/autofix
RSRepair 工具	[49]	http://qiyuhua.github.com/projects/rsrepair
SearchRepair 工具	[28]	http://github.com/ProgramRepair/SearchRepair
QACrashFix 工具	[62]	http://sei.pku.edu.cn/~gaoqing11/qacrashfix
Nopol 工具	[33]	http://github.com/SpoonLabs/nopol
ASTOR 工具	[102]	http://github.com/SpoonLabs/astor
Angelix 工具	[37]	http://www.comp.nus.edu.sg/~abhik/tools/angelix/
Historicalfix 工具	[20]	https://github.com/xuanbachle/bugfixes
Boa 工具	[52]	https://github.com/chupanw/BoaChallenge
Prophet 工具	[48]	http://groups.csail.mit.edu/pac/patchgen/
SPR 工具	[47]	http://groups.csail.mit.edu/pac/patchgen/
Kali 工具	[26]	http://groups.csail.mit.edu/pac/patchgen/
BanditRepair	[90]	https://github.com/Spirals-Team/bandit-repair
NEPfix	[91]	https://github.com/Spirals-Team/npefix
LeakFix	[92]	http://sei.pku.edu.cn/~gaoqing11/leakfix/home.htm
VEJOVIS	[95]	http://www.ece.ubc.ca/~frolino/projects/vejovis/

8 国内外研究组的总结

该节对国内外在程序自动修复领域比较活跃的研究组进行总结:

(1) Le Goues 和 Weimer 等人.

他们对基于搜索的缺陷自动修复方法进行了深入研究,例如提出了经典的 GenProg 方法[4,21,25]和 AE 方法[46]并进行了实证研究.除此之外,他们还为自动程序修复领域贡献了两个经典的缺陷库 ManyBugs 和 IntroClass[100].

(2) Rinard 等人.

他们通过手工分析了 GenProg 方法、RSRepair 方法和 AE 方法修复的补丁后,发现绝大部分补丁并不是正确补丁[26],他们对该问题进行了深入的分析.随后他们提出了 SPR 方法[47]和 Prophet 方法[48].

除此之外,他们还对很多特定缺陷类型的自动修复进行了深入研究[13,94,97-98].

(3) Roychoudhury 等人.

他们对基于语义的修复方法进行了一系列深入的研究,先后提出了 SemFix[3]、DirectFix[57]和 Angelix[37]等自动程序修复方法.除此之外,他们针对回归缺陷的自动修复也进行了研究并提出了 Relifix 方法[32].

(4) Lu 等人.

他们的主要贡献是并发程序缺陷的自动修复[77-81,83,87].对并发程序缺陷的特点进行了深入分析和研究,并针对数据竞争、原子性违背、顺序违背和死锁等不同类型的并发程序缺陷提出 CFix[77]和 Afix 修复工具[78].

(5) Monperrus 和 Xuan 等人.

他们的主要贡献是提出了 Nopol 方法[33-34],并针对 Defects4J 缺陷库上对多种自动缺陷修复方法的有效性进行了验证[68].除此之外,Monperrus[103]针对 PAR 方法[5]的评述论文也引发了软件缺陷预测领域研究人员对该问题的更为深入的思考,并集成了缺陷修复工具 ASTOR[102].

(6) Meyer 和 Pei 等人.

他们的主要贡献是基于 Eiffel 语言提出了 AutoFix方法[14-18].他们的研究主要是针对基于契约的 Eiffel 语言,提出了基于契约进行自动修复的 AutoFix 方法,并将该方法集成到 EiffelStudio 开发工具中.

(7) Kim 等人.

他们的主要贡献是在 GenProg 方法的基础上,引入了代码修改模板,并提出了 PAR 方法[5].除此之外,他们对自动生成的补丁是否有助于开发人员提高软件调试效率进行了实证研究[74].

(8) Qi 和 Mao 等人.

他们的主要贡献是提出了 RSRepair 方法[49],同时基于自动程序修复对常见的软件缺陷定位方法的效果进行了评估[39],此外,在候选补丁检验过程

中对测试用例集的排序进行了优化[49,65].

(9) Xiong 等人.

他们借助问答网站分析,提出了一种有效的缺陷修复方法 QACrashFix[62].并针对 C 程序中与内存泄露相关的缺陷,提出了 LeakFix 方法[92].

(10) Cai 等人.

他们重点关注的是并发程序的测试问题,并注重在大规模真实程序中对解决方案的有效性进行验证.他们提出针对死锁的并发程序缺陷自动修复方法 DFixer[86],该方法区别于现有的一般方法,仅仅选取一个线程而不是全部线程进行修复,可以保证修复给定死锁的同时不引入新的死锁.

9　总结和展望

随着软件项目复杂度的日益提高,程序自动修复问题因本身具有的丰富理论价值和应用前景,日益得到了学术界和工业界的广泛关注,通过该综述,我们发现虽然大部分开创新研究成果来自国外研究人员,但近些年来,国内研究机构(例如北京大学、国防科技大学、中国科学院软件研究所、武汉大学、上海交通大学)在该领域也做出了很多具有影响力的研究工作.该问题目前仍是软件维护领域中的一个研究热点且离实际应用还存在很大的距离,我们认为还存在一些值得国内研究人员进一步关注的研究问题.论文依次从缺陷定位、补丁生成和评估、缺陷数量和类型、特定领域的缺陷修复以及缺陷修复在工业界中的应用这五个维度对未来的研究工作进行展望.

(1) 针对缺陷定位维度的研究展望

需要进一步考虑更为精准的缺陷定位方法.虽然缺陷定位技术已经发展了很多年且取得了一定的效果[8-11].但从自动程序修复的角度来看,一方面需要深入比较不同的缺陷定位方法对缺陷修复效率的影响,另一方面仍需要研究出更为精准的缺陷定位技术,以提高自动程序修复的效率.

(2) 针对补丁生成和评估维度的研究展望

更有效的自动程序修复方法.在修复过程中,目前主流的两类研究思路是基于搜索的方法和基于语义的方法.当前,这两类方法的修复效果与缺陷类型、软件所处领域和软件规模等因素相关,因此通过分析上述因素,选出最为有效的方法是一个重要的研究问题.同时在程序修复时,将这两类方法进行融合也是一个重要的研究问题.此外,冗余假设的合理性仍需进行更为深入的分析和讨论,最后还需要进一步充分利用开源设计以及问答网站的知识,来提高补丁的质量.

提出有效方法可以从疑似正确补丁中识别出正确补丁.目前自动程序修复过程中存在测试用例集的不完整性的问题,使得通过验证阶段的补丁仍有可能不被开发人员接受,并且疑似正确的补丁会阻碍正确补丁的生成和验证.针对该问题,一方面需要额外设计新的测试用例,以避免缺陷程序配套的测试用例集存在 under-specification 问题.另一方面在补丁正确性评价的时候,不仅需要考虑其在配套测试用例集上的执行结果,也需要额外考虑其他信息,例如运行时的断言或不变量等.

(3)针对缺陷数量和类型维度的研究展望

针对多缺陷程序的自动修复方法.目前的研究大都关注单缺陷的修复,只有少量研究针对多缺陷的修复取得一定的进展(例如 Angelix 方法).但对三个大规模开源系统(Firefox、Mylyn 和 Evolution)的分析,发现修改的平均规模介于 24 行～62 行.如果研究人员能够将自动化修复方法可以处理的修改扩展到多行,则程序自动修复方法在实际场景中使用的可能性将会大大增强.由于一般情况下很难预先知道缺陷的个数,所以多缺陷的定位和修复问题将是未来该领域的重要研究问题.

对缺陷的类型进行更为深入的分析.Monperrus 等人[103]对缺陷类型(Defect Class)进行了分析.对缺陷进行分类时,处于同一类的缺陷具有相同的根本原因(Root Cause)(例如对规约的错误理解等)、相同的程序执行特征(例如异常)或者相同的修复类型(例如修改 if 语句的条件).当然根本原因和修复类型存在一定的相关性,例如空指针异常可以通过增加指针是否为空的判断来修复.探讨程序执行特征和修复类型比较复杂,不同的缺陷有时可以用相同的方式去解决,另一方面,相同的缺陷,在不同场景下可能有不同的修复方式.最后需要建立一个标准缺陷库,可以对自动程序修复方法的修复效果给出公正的比较.

(4)针对特定领域缺陷修复维度的研究展望

解决更多并发缺陷自动修复研究中的挑战.并发缺陷的修复不仅需要消耗较多的时间,而且容易引入新的缺陷,虽然该领域的研究已经取得了丰硕的成果,但还有一些亟待解决的问题和挑战.如研究

人员可致力于研究处理更复杂的并发缺陷,开发可修复各种类型缺陷的综合性工具,还可借助硬件的支持进行并发程序缺陷的修复,此外也可以关注一般缺陷的自动修复方法,以寻求此类修复方法与并发缺陷自动修复研究的结合点.

(5)针对缺陷修复在工业界中的应用维度的研究展望

将研究成果应用于工业界.将自动程序修复方法应用到实践中,需要工具能够自动阅读代码、变异代码和运行配套测试用例.但在成果转化中面临很多研究挑战,例如很多程序的代码是由多种编程语言混合完成的.一些代码因调用第三方库使得相关程序分析更为复杂.除此之外,一些程序只能在一些特定机器上编译和运行,或者不具有易用的测试接口等.因此需要进一步跟工业界合作,将科研的原型工具转化成实际可用的产品.

目前,对真实缺陷的自动修复仍存在巨大的研究挑战,因此自动程序修复方法并未成功应用到工业界的实践中,因此若能够对上述研究挑战提出有效的解决方案,将进一步提高该领域的应用价值,最终达到大幅度提高缺陷修复效率、缓解项目预算有限和软件产品高质量需求间的矛盾.

致 谢 感谢匿名审稿人和武汉大学玄跻峰老师对这篇综述提出的宝贵意见和建议!

参 考 文 献

[1] Anand S, Burke E K, Chen T Y, et al. An orchestrated survey of methodologies for automated software test case generation. Journal of Systems and Software, 2013, 86(8): 1978-2001

[2] Chen Xiang, Gu Qing, Liu Wang-Shu, et al. Survey of static software defect prediction. Journal of Software, 2016, 27(1): 1-25(in Chinese)
(陈翔, 顾庆, 刘望舒等. 静态软件缺陷预测方法研究. 软件学报, 2016, 27(1): 1-25)

[3] Nguyen H D T, Qi D W, Roychoudhury A, Chandra S. SemFix: Program repair via semantic analysis//Proceedings of the International Conference on Software Engineering. San Francisco, USA, 2013: 772-781

[4] Weimer W, Nguyen T, Le Goues C, Forrest S. Automatically finding patches using genetic programming//Proceedings of the International Conference on Software Engineering. Washington, USA, 2009: 364-374

[5] Kim D, Nam J, Song J, Kim S. Automatic patch generation learned from human-written patches//Proceedings of the International Conference on Software Engineering. San Francisco, USA, 2013: 802-811

[6] Le Goues C, Forrest S, Weimer W. Current challenges in automatic software repair. Software Quality Journal, 2013, 21(3): 421-443

[7] Xuan Ji-Feng, Ren Zhi-Lei, Wang Zi-Yuan, et al. Progress on approaches to automatic program repair. Journal of Software, 2016, 27(4): 771-784(in Chinese)
(玄跻峰, 任志磊, 王子元等. 自动程序修复方法研究进展. 软件学报, 2016, 27(4): 771-784)

[8] Yu Kai, Lin Meng-Xiang. Advances in automatic fault localization techniques. Chinese Journal of Computers, 2011, 34(8): 1411-1422(in Chinse)
(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)

[9] Chen Xiang, Ju Xiao-Lin, Wen Wan-Zhi, Gu Qing. Review of dynamic fault localization approaches based on program spectrum. Journal of Software, 2015, 26(2): 390-412(in Chinese)
(陈翔, 鞠小林, 文万志, 顾庆. 基于程序频谱的动态缺陷定位方法研究. 软件学报, 2015, 26(2): 390-412)

[10] Wang Ke-Chao, Wang Tian-Tian, Su Xiao-Hong, Ma Pei-Jun. Key scientific issues and state-art of automatic software fault localization. Chinese Journal of Computers, 2015, 38(11): 2262-2278(in Chinese)
(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)

[11] Wong W E, Gao R Z, Li Y H, et al. A survey on software fault localization. IEEE Transactions on Software Engineering, 2016, 42(8): 707-740

[12] Dallmeier V, Zeller A, Meyer B. Generating fixes from object behavior anomalies//Proceedings of the International Conference on Software Engineering. Vancouver, Canada, 2009: 550-554

[13] Perkins J H, Kim S, Larsen S, et al. Automatically patching errors in deployed software//Proceedings of the ACM SIGOPS Symposium on Operating Systems Principles. Montana, USA, 2009: 87-102

[14] Wei Y, Pei Y, Furia C A, Meyer B, et al. Automated fixing of programs with contracts//Proceedings of the International Symposium on Software Testing and Analysis. New York, USA, 2010: 61-72

[15] Pei Y, Wei Y, Furia C A, et al. Code-based automated program fixing//Proceedings of the International Conference on Automated Software Engineering. Washington, USA, 2011: 392-395

[16] Pei Y, Furia C A, Nordio M, Meyer B. Automatic program repair by fixing contracts//Proceedings of the International Conference on Fundamental Approaches to Software Engineering. Grenoble, France, 2014: 246-260

[17] Pei Y, Furia C A, Nordio M, et al. Automated fixing of programs with contracts. IEEE Transactions on Software Engineering, 2014, 40(5): 427-449

[18] Pei Y, Furia C A, Nordio M, Meyer B. Automated program repair in an integrated development environment//Proceedings of the International Conference on Software Engineering.

Firenze, Italy, 2015: 681-684

[19] Liu C, Yang J, Tan L, Hafiz M. R2Fix: Automatically generating bug fixes from bug reports//Proceedings of the International Conference on Software Testing, Verification, and Validation. Luxembourg, Luxembourg, 2013: 282-291

[20] Le D X B, Lo D, Le Goues C. History driven program repair//Proceedings of the International Conference on Software Analysis, Evolution, and Reengineering (SANER). Osaka, Japan, 2016: 213-224

[21] Le Goues C, ThanhVu N, Forrest S, Weimer W. GenProg: A generic method for automatic software repair. IEEE Transactions on Software Engineering, 2012, 38(1): 54-72

[22] Xie X Y, Liu Z C, Song S, et al. Revisit of automatic debugging via human focus-tracking analysis//Proceedings of the International Conference on Software Engineering. Austin, USA, 2016: 808-819

[23] Parnin C, Orso A. Are automated debugging techniques actually helping programmers?//Proceedings of the International Symposium on Software Testing and Analysis. Toronto, Ontario, Canada, 2011: 199-209

[24] Reps T, Ball T, Das M, Larus J. The use of program profiling for software maintenance with applications to the year 2000 problem. Software Engineering, 1997, 22(6): 432-449

[25] 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//Proceedings of the International Conference on Software Engineering. Zurich, Switzerland, 2012: 3-13

[26] Qi Z, Long F, Achour S, Rinard M. An analysis of patch plausibility and correctness for generate-and-validate patch generation systems//Proceedings of the 2015 International Symposium on Software Testing and Analysis. New York, USA, 2015: 24-36

[27] Debroy V, Wong W E. Using mutation to automatically suggest fixes for faulty programs//Proceedings of the International Conference on Software Testing, Verification and Validation. Paris, France, 2010: 65-74

[28] Ke Y, Stolee K T, Le Goues C, Brun Y. Repairing programs with semantic code search (T)//Proceedings of the International Conference on Automated Software Engineering. Lincoln, USA, 2015: 295-306

[29] Arcuri A. Evolutionary repair of faulty software. Applied Soft Computing, 2010, 11(4): 3494-3514

[30] Jones J A. Empirical evaluation of the tarantula automatic fault-localization technique//Proceedings of the International Conference on Automated Software Engineering. Long Beach, USA, 2005: 273-282

[31] Kaleeswaran S, Tulsian V, Kanade A, Orso A. Minthint: Automated synthesis of repair hints//Proceedings of the International Conference on Software Engineering. New York, USA, 2014: 266-276

[32] Tan S H, Roychoudhury A. Relifix: Automated repair of software regressions//Proceedings of the International Conference on Software Engineering. Firenze, Italy, 2015: 471-482

[33] Xuan J F, Martinez M, DeMarco F, et al. Nopol: Automatic repair of conditional statement bugs in Java programs. IEEE Transactions on Software Engineering, 2016, 2017, 43(1): 34-55

[34] DeMarco F, Xuan J F, Berre D L, Monperrus M. Automatic repair of buggy if conditions and missing preconditions with SMT //Proceedings of the International Workshop on Constraints in Software Testing, Verification, and Analysis. Hyderabad, India, 2014: 30-39

[35] Rui A, Zoeteweij P, Gemund A J C V. An evaluation of similarity coefficients for software fault localization//Proceedings of the Pacific Rim International Symposium on Dependable Computing. University of California. Riverside, USA, 2006: 39-46

[36] Rui A, Zoeteweij P, Gemund A J C V. On the accuracy of spectrum-based fault localization//Proceedings of the Testing: Academic and Industrial Conference Practice and Research Techniques. Windsor, UK, 2007: 89-98

[37] Mechtaev S, Yi J, Roychoudhury A. Angelix: Scalable multiline program patch synthesis via symbolic analysis// Proceedings of the International Conference on Software Engineering. Austin, USA, 2016: 691-701

[38] Chandra S, Torlak E, Barman S, Bodik R. Angelic debugging// Proceedings of the International Conference on Software Engineering. New York, USA, 2011: 121-130

[39] Qi Y H, Mao X G, Lei Y, Wang C S. Using automated program repair for evaluating the effectiveness of fault localization techniques//Proceedings of the International Symposium on Software Testing and Analysis. Lugano, Switzerland, 2013: 191-201

[40] Naish L, Lee H J, Ramamohanarao K. A model for spectra-based software diagnosis. ACM Transactions on Software Engineering and Methodology, 2011, 20(3): 1-32

[41] Xie X, Chen T Y, Kuo F C, Xu B. A theoretical analysis of the risk evaluation formulas for spectrum-based fault localization. ACM Transactions on Software Engineering & Methodology, 2013, 22(4): 402-418

[42] Harman M, Mansouri S A, Zhang Y Y. Search-based software engineering: Trends, techniques and applications. ACM Computing Surveys, 2012, 45(1): 1-61

[43] Arcuri A, Yao X. A novel co-evolutionary approach to automatic software bug fixing//Proceedings of the Evolutionary Computation. Hong Kong, China, 2008: 162-168

[44] Weimer W, Forrest S, Le Goues C, Nguyen T. Automatic program repair with evolutionary computation. Communications of the ACM, 2010, 53(5): 109-116

[45] Le Goues C, Westley W, Stephanie F. Representations and operators for improving evolutionary software repair// Proceedings of the International Conference on Genetic and

Evolutionary Computation Conference. Philadelphia, USA, 2012: 959-966

[46] Weimer W, Fry Z P, Forrest S. Leveraging program equivalence for adaptive program repair: Models and first results//Proceedings of the International Conference on Automated Software Engineering. Palo Alto, USA, 2013: 356-366

[47] Long F, Rinard M. Staged program repair with condition synthesis//Proceedings of the Joint Meeting on Foundations of Software Engineering. Bergamo, Italy, 2015: 166-178

[48] Long F, Rinard M. Automatic patch generation by learning correct code//Proceedings of the ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages. St. Petersburg, USA, 2016: 298-312

[49] Qi Y H, Mao X G, Lei Y, et al. The strength of random search on automated program repair//Proceedings of the International Conference on Software Engineering. Hyderabad, India, 2014: 254-265

[50] Martinez M, Weimer W, Monperrus M. Do the fix ingredients already exist? An empirical inquiry into the redundancy assumptions of program repair approaches//Proceedings of the International Conference on Software Engineering. Hyderabad, India, 2014: 492-495

[51] Barr E T, Brun Y, Devanbu P, et al. The plastic surgery hypothesis//Proceedings of the ACM SIGSOFT International Symposium on Foundations of Software Engineering. Hong Kong, China, 2014: 306-317

[52] Soto M, Thung F, Wong C P, et al. A deeper look into bug fixes: Patterns, replacements, deletions, and additions//Proceedings of the International Conference on Mining Software Repositories. Austin, USA, 2016: 512-515

[53] Tan S H, Yoshida H, Prasad M, Roychoudhury A. Anti-patterns in search-based program repair//Proceedings of the International Symposium on Foundations of Software Engineering. Seattle, USA, 2016: 727-738

[54] Kong X, Zhang L, Wong W E, et al. Experience report: How do techniques, programs, and tests impact automated program repair?//Proceedings of the International Symposium on Software Reliability Engineering, Baltimore, USA, 2015: 194-204

[55] Zhong H, Su Z. An empirical study on real bug fixes//Proceedings of the International Conference on Software Engineering. Piscataway, USA, 2015: 913-923

[56] Jha S, Gulwani S, Seshia S A, et al. Oracle-guided component-based program synthesis//Proceedings of the International Conference on Software Engineering. New York, USA, 2010: 215-224

[57] Mechtaev S, Yi J, Roychoudhury A. DirectFix: Looking for simple program repairs//Proceedings of the International Conference on Software Engineering. Firenze, Italy, 2015: 448-458

[58] Durieux T, Monperrus M. DynaMoth: Dynamic code synthesis for automatic program repair//Proceedings of the International Workshop on Automation of Software Test. New York, USA, 2016: 85-91

[59] Marcote S R L, Monperrus M. Automatic Repair of Infinite Loops. France: University of Lille, Arxiv: 1504.05078v1, 2015

[60] Gopinath D, Malik M Z, Khurshid S. Specification-based program repair using SAT//Proceedings of the Tools and Algorithms for the Construction and Analysis of Systems. Saarbrücken, Germany, 2011: 173-188

[61] Jackson D. Software Abstractions: Logic, Language, and Analysis. Massachusetts, USA: The MIT Press, 2006

[62] Gao Q, Zhang H, Wang J, Xiong Y, et al. Fixing recurring crash bugs via analyzing Q&A sites//Proceedings of the International Conference on Automated Software Engineering. Lincoln, USA, 2015: 307-318

[63] Stolee K T, Elbaum S, Dobos D. Solving the search for source code. ACM Transactions on Software Engineering & Methodology, 2014, 23(3): 1-45

[64] Hme M, Roychoudhury A. CoREBench: Studying complexity of regression errors//Proceedings of the International Symposium on Software Testing and Analysis. New York, USA, 2014: 105-115

[65] Qi Y H, Mao X G, Lei Y. Efficient automated program repair through fault-recorded testing prioritization//Proceedings of the IEEE International Conference on Software Maintenance and Evolution. San Francisco, USA, 2013: 180-189

[66] 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

[67] Long F, Rinard M. An analysis of the search spaces for generate and validate patch generation systems//Proceedings of the International Conference on Software Engineering. Austin, USA, 2016: 702-713

[68] Martinez M, Durieux T, Xuan J, et al. Automatic repair of real bugs in Java: A large-scale experiment on the Defects4J dataset. Switzerland: University of Lugano, HAL: hal-01387556, 2016

[69] Just R, Jalali D, Ernst M D. Defects4J: A database of existing faults to enable controlled testing studies for Java programs//Proceedings of the International Symposium on Software Testing and Analysis. San Jose, USA, 2014: 437-440

[70] Smith E K, Barr E T, Le Goues C, Brun Y. Is the cure worse than the disease? Overfitting in automated program repair//Proceedings of the Joint Meeting on Foundations of Software Engineering. Bergamo, Italy, 2015: 532-543

[71] Jiang Ming-Yue, Chen Tsong Yueh, Kuo Fei-Ching, et al. A metamorphic testing approach for supporting program repair without the need for a test oracle. The Journal of Systems & Software, 2017, 126(4): 127-140

[72] Segura S, Fraser G, Sanchez A, et al. A survey on metamorphic testing. IEEE Transactions on Software Engineering, 2016, 42(9): 805-824

[73] Fry Z P, Landau B, Weimer W. A human study of patch maintainability//Proceedings of the International Symposium on Software Testing and Analysis. Minneapolis, USA, 2012: 177-187

[74] Tao Y D, Kim J, Kim S, Xu C. Automatically generated patches as debugging aids: A human study//Proceedings of the ACM SIGSOFT International Symposium on Foundations of Software Engineering. Hong Kong, China, 2014: 64-74

[75] Tao Y, Han D, Kim S. Writing acceptable patches: An empirical study of open source project patches//Proceedings of the IEEE International Conference on Software Maintenance and Evolution. Victoria, Canada, 2014: 271-280

[76] Le X B D, Le T D B, Lo D. Should fixing these failures be delegated to automated program repair//Proceedings of the International Symposium on Software Reliability Engineering. Gaithersburg, USA, 2015: 427-437

[77] Jin G L, Zhang W, Deng D D, Lu S, et al. Automated concurrency-bug fixing//Proceedings of the USENIX Conference on Operating Systems Design and Implementation. Hollywood, USA, 2012: 221-236

[78] Jin G L, Song L H, Zhang W, et al. Automated atomicity-violation fixing//Proceedings of the Programming Language Design and Implementation. San Jose, USA, 2011: 389-400

[79] Park S, Lu S, Zhou Y. CTrigger: Exposing atomicity violation bugs from their hiding places. Computer Architecture News, 2009, 37(1): 25-36

[80] Liu H, Chen Y, Lu S. Understanding and generating high quality patches for concurrency bugs//Proceedings of the International Symposium on the Foundations of Software Engineering. Seattle, USA, 2016: 715-726

[81] Zhang W, Sun C, Lu S. ConMem: Detecting severe concurrency bugs through an effect-oriented approach. Transactions on Software Engineering, 2010, 38(1): 179-192

[82] Khoshnood S, Kusano M, Wang C. ConcBugAssist: Constraint solving for diagnosis and repair of concurrency bugs//Proceedings of the International Symposium on Software Testing and Analysis. Baltimore, Maryland, USA, 2015: 165-176

[83] Lu S, Park S, Seo E, Zhou Y Y. Learning from mistakes: A comprehensive study on real world concurrency bug characteristics. ACM SIGARCH Computer Architecture News, 2008, 43(3): 329-339

[84] Park S. Debugging non-deadlock concurrency bugs//Proceedings of the International Symposium on Software Testing and Analysis. Lugano, Switzerland, 2013: 358-361

[85] Surendran R, Raman R, Chaudhuri S, et al. Test-driven repair of data races in structured parallel programs//Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation. Edinburgh, UK, 2014: 15-25

[86] Cai Y, Cao L W. Fixing deadlocks via lock pre-acquisitions//Proceedings of the International Conference on Software Engineering. Austin, USA, 2016: 1109-1120

[87] Jin G, Thakur A, Liblit B, Lu S. Instrumentation and sampling strategies for cooperative concurrency bug isolation//Proceedings of the International Conference on Object Oriented Programming Systems Languages and Applications. Reno, USA, 2010: 241-255

[88] Liblit B, Aiken A, Zheng A X, et al. Bug isolation via remote program sampling//Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation. San Diego, USA, 2003: 141-154

[89] Liblit B, Naik M, Zheng A X, et al. Scalable statistical bug isolation//Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation. Chicago, USA, 2005: 15-26

[90] Durieux T, Hamadi Y, Monperrus M. BanditRepair: Speculative exploration of runtime patches. France: University of Lille, Arxiv: 1603. 07631v1, 2016

[91] Cornu B, Durieux T, Seinturier L, et al. NPEFix: Automatic runtime repair of null pointer exceptions in Java. France: University of Lille, Arxiv: 1512. 07423, 2015

[92] Gao Q, Xiong Y, Mi Y, Zhang L, et al. Safe memory-leak fixing for C programs//Proceedings of the International Conference on Software Engineering. Firenze, Italy, 2015: 459-470

[93] Long F, Sidiroglou-Douskos S, Rinard M. Automatic runtime error repair and containment via recovery shepherding//Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation. Edinburgh, UK, 2014: 227-238

[94] Gopinath D, Khurshid S, Saha D, et al. Data-guided repair of selection statements//Proceedings of the 36th International Conference on Software Engineering. Hyderabad, India, 2014: 243-253

[95] Ocariza F S, Pattabiraman K, Mesbah A. Vejovis: Suggesting fixes for JavaScript faults//Proceedings of the International Conference on Software Engineering. Hyderabad, India, 2014: 837-847

[96] Samirni H, Schafer M, Artzi S, et al. Automated repair of HTML generation errors in PHP applications using string constraint solving//Proceedings of the International Conference on Software Engineering. Piscataway, Zürich, Switzerland, 2012: 277-287

[97] Demsky B, Rinard M. Automatic detection and repair of errors in data structures. ACM SIGPLAN Notices, 2003, 38(11): 78-95

[98] Demsky B, Rinard M C. Goal-directed reasoning for specification-based data structure repair. IEEE Transactions on Software Engineering, 2006, 32(12): 931-951

[99] Sidiroglou-Douskos S, Lahtinen E, Long F, Rinard M. Automatic error elimination by horizontal code transfer across multiple applications//Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation. Portland, USA, 2015: 43-54

[100] Le Goues C, Holtschulte N, Smith E K, et al. The ManyBugs and IntroClass benchmarks for automated repair of C programs. IEEE Transactions on Software Engineering, 2015, 41(12): 1236-1256

[101] Cadar C, Dunbar D, Engler D. KLEE: Unassisted and automatic generation of high-coverage tests for complex systems programs//Proceedings of the USENIX Symposium on Operating Systems Design and Implementation. San Diego, USA, 2008: 209-224

[102] Matias M, Martin M. ASTOR: A program repair library for Java (demo)//Proceedings of the International Symposium on Software Testing and Analysis. Saarbrücken, Germany, 2016: 441-444

[103] 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// Proceedings of the International Conference on Software Engineering. Hyderabad, India, 2014: 234-242

WANG Zan, born in 1979, Ph. D, associate professor. His research interests include search based software engineering, software fault localization and automatic program repair, etc.

GAO Jian, born in 1992, M. S. candidate. Her research interests include optimization on software testing and automatic program repair.

CHEN Xiang, born in 1980, Ph. D., associate professor. His research interests include software defect prediction, regression testing, fault localization, and combinatorial testing, etc.

FU Hao-Jie, born in 1993, M. S. candidate. Her research interests include software fault localization and automatic program repair.

FAN Xiang-Yu, born in 1992, M. S. candidate. His research interests include software fault localization and automatic program repair.

Background

Automatic program repair (APR) is a hot research topic in the field of software engineering. To generate patches which can fix defects and achieve the most appropriate patches, APR approaches determine the possible locations of buggy statements by fault localization approaches firstly, then search and generate patches for buggy statements by some approaches. Many different kinds of methods have been proposed to solve this problem and the repair result is acceptable. This paper summarizes these methods into three categories: Search based, constraint-solving based and other approaches, then analyze some important issues in this area, such as patches search space, existing APR tools, program repair evaluation metrics and the future challenges.

This work is supported by the National Natural Science Foundation of China under Grant Nos. 61202030, 61202006 and 71502125. These projects mainly aim to study the methods which can ensure correctness and reliability of software. We have done some work on regression testing optimization and fault localization which can help the developers to localize faults in programs. Naturally, the next problem is how to fix the faults. This paper surveys the current research on automatic program repair technology which is one of the hottest research topic in software maintenance, and presents the challenges and future work in this area.