面向程序自动修复的缺陷分类方法研究*

易 昕，毛晓光，纪 涛
（国防科学技术大学 计算机学院 计算机科学与技术系，长沙 410072）

摘 要：程序自动修复是近年软件工程的研究热点，但其发展难以满足工业应用的要求。分析了自动修复技术的局限性，引入了缺陷分类的思想，并与正交缺陷分类方法相结合，提出了面向软件自动修复的缺陷分类方法（APRDC）。基于 APRDC 提出了自动修复技术集成的思想。实验中将基于 APRDC 与基于随机的自动修复技术集成工具作了比较。基于 APRDC 的方法提高了成功率并缩短了修复时间。实验结果表明了 APRDC 方法的有效性。

关键词：软件缺陷；程序自动修复；缺陷分类；技术集成
中图分类号：TP311.53 **文献标志码**：A **文章编号**：1001-3695(2016)06-1748-04
doi:10.3969/j.issn.1001-3695.2016.06.033

Research on defect classification method of automated program repair

Yi Xin, Mao Xiaoguang, Ji Tao
(*College of Computer Science*, *National University of Defense Technology*, *Changsha* 410072, *China*)

Abstract: Automated program repair is a hot topic of software engineering in recent years, but it is far from the application in industry. This paper analyzed the limitation of automated repair technologies and put forward a defects classification method of automated program repair (APRDC). APRDC introduced the idea of defect classification, and combined with the orthogonal defect classification. Based on APRDC, it put forward the idea of multiple automated repair technologies integration. In the experiment, the integrated tool of multiple automated repair technologies which based on APRDC was compared with which based on random method. Consequently, based on APRDC, the integration of multiple automated repair technologies improves the effect of repair. It also shows that the validity of APRDC.

Key words: software defect; automated program repair; defect classification; technology integration

0 引言

软件缺陷不可避免，为保证软件的正常运行，软件开发人员需要花费大量的精力来查找并修复软件缺陷，导致软件的维护成本过高。在 2002 年，美国国家标准技术研究所（National Institute of Standards and Technology, NIST）的报告[1]中就指出软件故障（software fault）每年所导致的美国经济损失约 595 亿美元，大约为其 GDP 的 0.6%。软件自动修复技术拟将软件缺陷的修复过程交由计算机完成，软件自动修复技术的目的是减少修复软件缺陷的时空开销，把软件开发人员从耗时耗力的软件故障修复工作中解放出来，大大减少软件开发成本，并提高软件开发的效率和质量。

软件自动修复是近年来软件工程的前沿研究方向，由于缺陷修复的复杂性，自动修复技术主要局限于学术研究，难以满足工业应用的要求。现有软件自动修复技术对软件缺陷信息挖掘程度不够，自动修复技术趋向于修复简单缺陷，而对复杂缺陷修复能力较弱。自动修复技术所能得到的信息与软件维护人员所能得到的信息相同，同时修复技术又不具备软件维护人员对软件的充分了解，导致现有自动修复技术修复能力比不上具有丰富经验的软件维护人员。所以，提高自动修复技术的修复能力是目前软件自动修复领域的重要研究内容。

本文从软件自动修复的角度，结合正交缺陷分类方法（orthogonal defect classification, ODC[2]），研究面向自动修复的缺陷分类方法。本文主要工作包括：a）通过分析研究自动修复技术局限性，说明面向自动修复技术的缺陷分类的必要性；b）提出面向自动修复的缺陷分类方法（automated program repair defect classification, APRDC）；c）基于 APRDC 方法，提出自动修复技术集成思想，并通过实验分析表明了 APRDC 方法的有效性。

1 自动修复技术及其局限性分析

自动修复技术从修复流程上看，包括错误定位、补丁生成、补丁验证三个步骤，如图 1 所示。

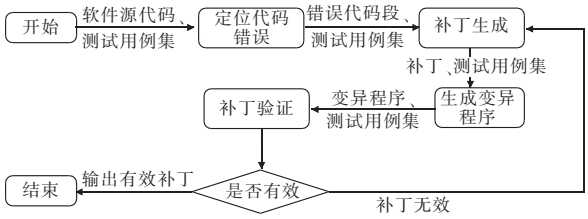

图 1 自动修复基本流程图

由于缺陷修复的正确性难以定义，通过回归测试的方式来

收稿日期：2015-06-18；**修回日期**：2015-07-30 **基金项目**：国家自然科学基金资助项目（61379054）

作者简介：易昕（1992-），男，硕士研究生，主要研究方向为高可信软件、软件自动修复（yixin_09@163.com）；毛晓光（1970-），男，教授，博士，主要研究方向为软件工程、高可信软件；纪涛（1991-），男，硕士研究生，主要研究方向为高可信软件、软件自动修复.

确认修复正确性是工业界较为认同的方式。自动修复的修复流程基本相同,但修复策略的不同导致了自动修复技术的多样性和局限性,下面从修复思想和局限性角度对自动修复的典型技术及工具进行简单分析。

1.1　基于搜索的自动修复技术

基于搜索的自动修复技术从指定的搜索空间中寻找解决方案,生成修复补丁。自动修复工具 Genprog 的搜索范围限于自身程序代码,使用遗传算法提高搜索效率。Genprog 针对现实程序中的 105 个缺陷进行修复,修复了其中的 55 个[3],最新研究表明[4],其中 37 个修复实际上是错误的(并不能通过实际的测试用例),只有 18 个修复能通过所有实际的测试用例。

最新针对回归错误的自动修复工具[5],修复了 35 个回归错误中的 23 个,而 Genprog 只修复了 5 个,新的修复思想从过去回归错误的修复中进行学习,得到了针对回归错误的修复操作,提高了对回归错误的修复能力。这表明基于搜索的自动修复技术需要挖掘更多的缺陷信息来辅助修复操作。

1.2　基于语义的自动修复技术

基于语义的自动修复技术,使用语义分析和程序综合的方法进行修复,代表性的工具是 SemFix[6]。SemFix 使用 Tarantula 技术进行缺陷定位,获得可疑语句排序,通过输入测试用例,用符号执行工具 KLEE 产生修复约束,利用程序综合产生变体,最终完成修复。实验从软件基础库 SIR[7] 中提取基准程序集,基于此实验集上获得了优于 Genprog 的修复效果(时间缩短和成功率提高)。

根据 SemFix 的实验设置和它自己的描述,基于语义的自动修复技术有以下局限性:a)程序规模限制;b)测试用例数量限制;c)修复复杂性限制。具体来说,基于语义的自动修复,由于使用约束求解、符号执行等技术,其适用的程序规模较小(SIR 的 C 程序集基本少于万行),测试用例数量受限,面对多行代码出现错误的情况,修复能力不足。

1.3　基于模板的自动修复技术

基于模板的自动修复技术,使用了修复模板的思想,对某些错误存在的固定修复模式定制修复模板,根据修复模板生成修复补丁。Pan 等人[8]从多个 Java 开源程序中分析提取了 27 个修复模板(fix pattern)。自动修复工具 PAR[9] 使用了 10 个修复模板,对通过缺陷定位获得的可疑语句进行匹配,对适合其修复模板的语句用模板进行替换,然后进行补丁验证。

PAR 使用的修复模板主要是针对单行语句进行修复,比如对象初始化、空指针检查等。目前提供的修复模板对软件缺陷的覆盖能力还很弱,而且不同编程语言和不同功能的软件,总结修复模板较为困难。

1.4　基于变异的自动修复技术

基于变异(mutation-based)的修复[8]来源于变异测试领域。对程序使用变异操作来产生程序变体。变异操作来源于变异测试,不同于 Genprog 和 PAR,所有程序变体通过一次变异得到,然后逐个验证有效性。变异修复关键在于其变异操作的定义,由于变异操作可能对程序状态影响较大,针对多行代码的变异操作较为困难。变异修复操作主要是针对单行的运算表达式和条件判断语句。

1.5　自动修复技术局限性分析

自动修复技术的局限性可以归纳为以下两点:a)修复技术主要针对单行语句进行修复,修复的错误类型较为简单;b)除基于模板的修复技术,其他修复技术对适用于修复哪些缺陷缺乏一个清晰的定义,导致对修复技术的修复能力的认知存在一定的模糊性。

正是由于自动修复技术存在的局限性,需要对自动修复技术所能修复的软件缺陷有清晰的定义,才能进一步认清自动修复技术的修复能力,更清楚地认识目前修复能力的不足之处。

2　APRDC 缺陷分类方法

由于现阶段软件缺陷自动修复缺乏系统的解决方案,所以,多种修复技术的有机整合是提高修复能力的一个有效途径。针对这一需求,本章提出了面向自动修复技术的缺陷分类方法:APRDC。下面介绍该方法及其实现细节。

2.1　方法概述

APRDC 借鉴公认的正交缺陷分类方法(ODC),ODC 使用的分类方法学经过详细设计和长久维护,保证收集的数据的必要性和非冲突性。APRDC 方法针对软件自动修复,提供缺陷属性的清晰定义,包含关键字和语义等相关信息,保证分类的独立性。

APRDC 主要完成以下三个度量目标:

a)修复约束。修复约束包括重现缺陷需要满足的条件和修复完成需要满足的条件。这是确定自动修复输入输出的关键。

b)缺陷环境。缺陷环境包括缺陷的位置信息和缺陷所处的软件环境。缺陷环境是保证自动修复技术定位缺陷,并正确运行目标程序的基础。

c)缺陷类型。对缺陷类型的清晰定义,建立自动修复技术与缺陷类型的映射关系,从缺陷类型的角度明确对自动修复技术的修复能力的认知。

2.2　实现细节

根据 APRDC 的三个度量目标,APRDC 主要从以下四个属性来对缺陷进行定义:a)缺陷触发(triggers);b)软件环境(software-context);c)缺陷类型(defect-type);d)缺陷严重程度(severity)。下面对各属性含义以及属性值进行清晰定义。

根据 ODC 中对 Triggers 的描述,一个缺陷触发表示产生缺陷所需要的条件。对应到自动修复中,trigger 用来表示缺陷复现的条件,同时 trigger 提供验证缺陷修复的条件。其定义如下:

定义 1　Triggers = {state, input, output}

其中:state 用来表示缺陷发生时,软件所处的状态信息,具体为软件的占用的内存、所处的系统、时间等;input 表示导致缺陷发生的输入值,但不仅仅针对数值,可以包括文本、图片、数值以及其他类型的输入;output 是软件的输出,与输入相对应,记录软件的输出,用于自动修复的补丁验证。

缺陷的修复离不开其所处的软件环境,同时需要获取缺陷的位置信息。其定义如下:

定义 2　Software-context = {code-line, defect-location, software-lang}

其中:code-line 是软件的代码量;defect-location 是缺陷所处的位置,其粒度可能有所不同,具体值有模块、文件、函数等;software-lang 表示软件语言,即 software language,不同语言虽然有

其共性,但对实际修复来说,从编译、代码变换等方面来看,需要考虑到不同编程语言的独特性。

缺陷类型的定义采用 ODC 的缺陷分类思想,同时根据缺陷修复的本质进行分类。从自动修复的角度来看,修复是一个语句结构变换的过程,但从人修复的角度来看,修复包含着语义的改变,所以 APRDC 根据缺陷的 ODC 分类和结构进行分类。其定义如下:

定义 3 Defect-type = {ODC-type, structure}

其具体值如表 1 所示。ODC-type 包括赋值、检查、接口、算法、功能等缺陷类型;structure 类型对应缺陷修复的本质,missing 表示缺失代码,wrong 表示代码错误,extraneous 表示多余代码,分别可以对应修复的添加、变异、删除操作。

表 1 缺陷类型定义

ODC-type	structure
assignment(赋值)	
check(检查)	missing
interface(接口)	wrong
algorithm(算法)	extraneous
function(功能)	

为进一步给出缺陷类型的精确分类,本文对 fbc、gmp、gzip、libtiff、lighttpd、php、python、wireshark 八个软件中的 105 个缺陷(http://dijkstra. cs. virginia. edu/genprog/resources/genprog-icse2012-benchmarks/)进行分类。基本信息如表 2 所示。

表 2 程序基本信息

程序	代码行数	缺陷数	基本描述
fbc	97 000	3	编译器
gmp	145 000	2	数值计算
gzip	491 000	5	数据压缩
libtiff	77 000	24	图像处理
lighttpd	62 000	9	Web 服务
php	1 046 000	44	Web 编程
python	407 000	11	编译器
wireshark	2 814 000	7	数据包分析
总数	5 139 000	105	

根据表 1 中给出的缺陷类型,对 105 个缺陷进行分类,其 ODC 类型分布如图 2 所示。

根据缺陷结构特征的分布如图 3 所示。

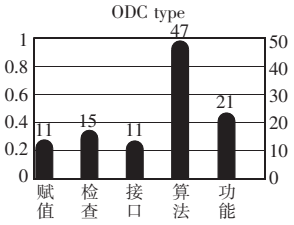

图 2 ODC 缺陷类型分布

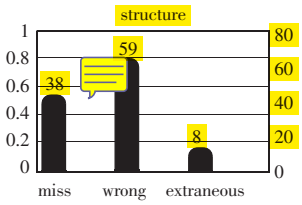

图 3 结构特征分布

进一步结合 ODC 缺陷类型和结构特征进行分类,分类数据如表 3 所示(第 2 栏表示 structure,使用了简写)。

从表 3 中可以看到,研究的 105 个缺陷都包含在定义的缺陷类型中,通过 ODC 类型和缺陷结构类型的组合定义,可以正确反映缺陷特征,给出缺陷的准确定位。

根据缺陷发生导致的软件状态的改变,评估缺陷的严重程度,由此给出定义 4:

定义 4 Severity = {normal, important, serious}

通过评估缺陷的严重程度,面对多个缺陷需要修复时,用于给修复技术一个缺陷优先级的参考。

综上所述,本节对 APRDC 方法的四个属性以及对应的属性值进行了定义,并对八个软件中的 105 个缺陷进行了分类。APRDC 方法较全面地给出了面向自动修复的缺陷分类方法,用于缺陷类型的精确定义。

3 基于 APRDC 的自动修复技术集成

根据自动修复技术的发展现状,考虑其存在的问题,APRDC 方法可以应用于现有自动修复技术的集成。

根据目前自动修复的研究现状,自动修复技术集成思想是本文首次提出,本文认为缺陷分类是多种自动修复方法集成的基础。自动修复技术集成的基本描述为:以缺陷分类为基础,建立缺陷类型到修复技术的映射关系,构建融合多种自动修复技术的修复平台,实现修复效果的叠加化。本文在实验中会对自动修复集成思想的可行性进行验证。

同时 APRDC 方法具有一定的前瞻性,可能适用于未来更强大的自动修复技术,更强的修复技术可能具有自我学习的功能[10],能够自我挖潜 APRDC 的四个属性信息,并实现自动分类,同时不断学习对应类型的修复操作,拥有自我提高的能力。

表 3 105 个缺陷分类表

语义	结构	举例说明	缺陷	百分比/%
赋值	miss	变量未初始化	2	1.90
	wrong	错误的赋值	6	5.71
	extra	不应该赋值的变量	3	2.86
检查	miss	if 结构丢失	8	7.62
	wrong	错误的逻辑表达式	4	3.81
	extra	多余的 if 结构	3	2.86
接口	miss	函数参数缺少	1	0.95
	wrong	传给函数错误的值	10	9.52
	extra	传给函数多余的值	0	0.00
算法	miss	算法不完整	27	25.71
	wrong	算法错误	18	17.14
	extra	算法存在多余步骤	2	1.90
功能	miss	需要新的程序模块	10	9.52
	wrong	大量代码更改	11	10.48
	extra	大量代码多余	0	0.00

4 实验及结果分析

本章拟通过实验对多自动修复技术集成思想的可行性进行验证,同时也对 APRDC 方法的有效性进行验证。实验主要回答以下问题:a)APRDC 方法能否有效应用于自动修复;b)基于 APRDC 方法的多自动修复技术集成的可行性?

4.1 实验方法

实验对象程序采用 SIR 中的 tcas 程序,tcas 是空中飞行器的冲撞检测和响应的应对系统,它主要通过监控雷达信息,检测附近侵入飞行器是否会产生冲撞。软件基础库 SIR 中包含 tcas 的 41 个错误版本,并提供了 1 608 个测试用例。

为验证基于 APRDC 方法的多自动修复技术集成的可行性,实验构造集成基于搜索(base_search)[11] 以及基于语义(base_semantic)(https://bitbucket. org/nguyenthanhvuh/ceti/)的自动修复工具原型。由于基于模板以及其他一些自动修复工具并不开源,为保证结果的可靠性,目前只采用基于搜索和基于语义的两种自动修复技术进行集成。为控制实验变量,实验统一采用 Tarantula[13] 方法进行缺陷定位,Tarantula 的公式为

$$\mathrm{susp}(s) = \frac{\mathrm{failed}(s)/\mathrm{totalfailed}}{\frac{\mathrm{passed}(s)}{\mathrm{totalpassed}} + \mathrm{failed}(s)/\mathrm{totalfailed}}$$

其中:s 表示程序中语句,测试用例包含正反两种测试用例,正测试用例保证程序功能,反测试用例反映程序缺陷;failed(s)表示执行路径经过 s 的反测试用例数;passed(s)表示执行路径经过 s 的正测试用例数;totalfailed 表示总共执行失败的测试用例数;totalpassed 表示总共执行(测试用例)成功的测试用例数。

实验采用对比分析的方法进行,对目标组首先构建映射表。通过对基于语义和基于搜索的自动修复的研究,同时研究 tcas 的 41 个错误版本,可以建立修复技术和缺陷类型的映射表,如表 4 所示。

表 4　修复技术与缺陷类型映射表

结构	缺陷类型	版本号	修复技术
错误	赋值算术表达式	v1,v25,v39	B_sema
	赋值算术操作	v2	B_sema
	赋值逻辑表达式	v3,v4,v9,v12	B_sema
	返回错误值	v6,v10,v11,v37	B_sema
	赋值常量错误	v7,v8,v13,v14,v16,v17,v18,v19,v36	B_sema
	If 条件错误	v28,v34	B_sema
缺失	赋值丢失逻辑表达	v5,v15,v26,v27,v41	B_sear
	函数调用缺失	v20,v21,v22,v23,v24,v31,v32,v40	B_sear
	缺失条件判断	v29,v30	B_sear
	数据类型错误	v33,v38	NA
	算法代码位置	v35	B_sear

表 4 第一行表示缺陷的结构特征;第二、三行表示缺陷更具体的类型,以及对应的版本号;最后一行表示可能对应的修复技术;B_sema 表示基于语义的修复技术;B_sear 表示基于搜索的修复技术;NA 表示基于语义和基于搜索的修复技术都不太适用于这种修复技术。

为体现修复效果,对比组采用随机分配的方法对 41 个错误版本的 tcas 程序进行修复,为获得较为可靠的数据,使用随机的方法进行了十次重复的修复。

4.2　实验结果及分析

实验主要从成功率、修复时间、单位时间修复缺陷数量三个方面来衡量修复结果。

表 5　修复结果比较

方法	修复缺陷数	时间/min	成功率/%	B/T
APRDC	27	34.71	65.8	0.78
Random	16.1	49.47	39.2	0.33
Base-search	11	82.48	26	0.12
Base-semantic	23	45.96	56.1	0.50

从表 5 中可以看出,相对于随机的方法,基于 APRDC 方法的修复有更高的成功率,同时修复时间更短,修复成功率提高了 26.6%,由于采用随机的方法作为对照,取修复时间以及修复缺陷数的平均值作为随机方法的基准值。同时可以看到,集成两种修复技术后,随机的方法修复效果接近两的平均值,而基于 APRDC 的方法在修复时间和成功率上比 Base-semantic 都有所提高,为更好定义修复能力,本文比较了每分钟修复缺陷的数量。其公式定义如下:

$$\text{每分钟修复缺陷数量} = \text{修复缺陷数}/\text{修复时间}$$

表 5 的 B/T 栏即表示每分钟修复缺陷的数量,进一步分析实验结果可以得到,基于 APRDC 方法的修复能力是基于随机的 2.36 倍,是基于搜索的 6.5 倍,是基于语义的 1.56 倍。

图 4 表示修复的缺陷数与时间的关系图,包括十次随机修复的结果和一次基于 APRDC 方法的修复结果,右下角的点代表基于 APRDC 方法的修复结果。可以看出,与其他十个点距离比较远,明显不能与其他点进行拟合,说明其修复效果与基于随机方法的修复效果具有显著差异性。

图 5 给出了 11 次修复中,每分钟修复的缺陷数。可以看到,存在一个异常值,明显偏离其余观测值。异常值代表基于 APRDC 方法的修复结果,说明了其修复效果的显著性。

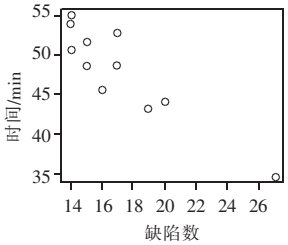

图 4　缺陷数与时间关系图

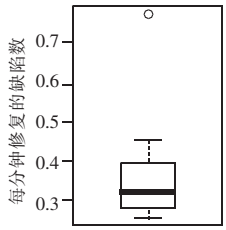

图 5　单位时间修复缺陷数

根据以上的实验结果可以回答:通过实验验证,相对于随机的情况,基于缺陷分类的、集成两种自动修复技术的工具,其修复效果有显著提升,在基于 APRDC 方法的多自动修复技术集成具有一定的可行性,同时也说明 APRDC 方法的有效性。

4.3　有效性威胁

实验实现了修复技术与缺陷类型的自动匹配,但对自动修复技术的修复类型分析来源于对工具对应的文章和提供的数据的了解,虽然实验效果表明分类的可行性,但可能结果还带有一定偏向性。另外,实验使用的缺陷定位技术可能影响实验结果,自动修复技术中缺陷定位技术的研究[13,14]指出,传统认为有效的定位技术可能不适用于自动修复技术。

实验只使用了单个程序的多个缺陷版本进行修复,没有考虑到缺陷位于不同程序的情况。同时,存在针对单一类型缺陷的自动修复技术,例如针对回归错误[5]、无限循环、数组溢出。这类自动修复技术融入了对所修复缺陷的信息,回归错误的修复技术从多个回归错误中学习到修复的模板,用于对新的回归错误的修复。针对单一缺陷的自动修复技术更具有针对性,对其针对类型的软件缺陷有较好的修复效果,这类自动修复技术仍然是从传统的缺陷类型的角度出发,没有从自动修复的角度来分析缺陷类型,这类修复技术的融合应用有待进一步研究。

实验进行了开拓性的工作,对多种自动修复集成的思想和 APRDC 方法的有效性验证,需要未来进一步实验。

5　结束语

软件自动修复技术发展有其必要性和挑战性,目前软件自动修复技术呈现方法多样性和修复能力的局限性,还未形成一个整体概念,本文提出面向自动修复的缺陷分类方法 APRDC,并基于此提出了自动修复技术集成的思想,并通过实验的对基于 APRDC 方法的多自动修复技术集成的可行性和 APRDC 方法的有效性进行验证,取得了较为显著的修复效果,开拓了自动修复发展的新思路。

参考文献:

[1] Tassey G. The economic impacts of inadequate infrastructure for software testing[R]. [S. l.]: National Institute of Standards and Technology, 2002.

[2] Chillarege R, Bhandari I S, Chaar J K, *et al*. Orthogonal defect classification-a concept for in-process measurements[J]. IEEE Trans on Software Engineering, 1992, 18(11): 943-956.

(下转第　　页)

(上接第　　页)

[3] Le Goues C, Dewey-Vogt M, Forrest S, *et al*. A systematic study of automated program repair: fixing 55 out of 105 bugs for MYM8 each [C]//Proc of the 34th International Conference on Software Engineering. [S. l.]: IEEE Press, 2012: 3-13.

[4] Qi Zichao, Long Fan, Achour S, *et al*. An analysis of patch plausibility and correctness for generate-and-validate patch generation systems, MIT-CSAIL-TR-2015-020 [R]. 2015.

[5] Tan S H, Roychoudhury A. Relifix: automated repair of software regressions[C]//Proc of International Conference on Software Engineering. 2015.

[6] Nguyen H D T, Qi Dawei, Roychoudhury A, *et al*. SemFix: program repair via semantic analysis[C]//Proc of the International Conference on Software Engineering. [S. l.]: IEEE Press, 2013: 772-781.

[7] Do H, Elbaum S, Rothermel G. Supporting controlled experimentation with testing techniques: an infrastructure and its potential impact[J]. Empirical Software Engineering, 2005, 10(4): 405-435.

[8] Pan Kai, Kim S, Whitehead Jr E J. Toward an understanding of bug fix patterns[J]. Empirical Software Engineering, 2009, 14(3): 286-315.

[9] Debroy V, Wong W E. Using mutation to automatically suggest fixes for faulty programs[C]//Proc of the 3rd International Conference on Software Testing, Verification and Validation. [S. l.]: IEEE Press, 2010: 65-74.

[10] Monperrus M. Software that learns from its own failures[J]. arXiv preprint arXiv: 1502.00821, 2015.

[11] Qi Yuhua, Mao Xiaoguang, Lei Yan, *et al*. The strength of random search on automated program repair[C]//Proc of the 36th International Conference on Software Engineering. [S. l.]: ACM Press, 2014: 254-265.

[12] Jones J A, Harrold M J. Empirical evaluation of the tarantula automatic fault-localization technique[C]//Proc of the 20th IEEE/ACM International Conference on Automated Software Engineering. [S. l.]: ACM Press, 2005: 273-282.

[13] Qi Yuhua, Mao Xiaoguang, Lei Yan, *et al*. Using automated program repair for evaluating the effectiveness of fault localization techniques [C]//Proc of the International Symposium on Software Testing and Analysis. [S. l.]: ACM Press, 2013: 191-201.

[14] 纪涛,齐玉华,毛晓光. 基于软件自动修复评估缺陷定位技术的工具:GenProg-FL[J]. 计算机科学, 2014, 41(9): 88-90, 124.