软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2013,24(4):663−674 [doi: 10.3724/SP.J.1001.2013.04379]

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

测试用例演化研究进展*

张智轶[1,2], 陈振宇[1,2], 徐宝文[1], 杨 瑞[1]

[1](计算机软件新技术国家重点实验室(南京大学),江苏 南京 210093)

[2](南京大学 软件学院,江苏 南京 210093)

通讯作者: 陈振宇, E-mail: zychen@software.nju.edu.cn

摘 要: 由于功能增加、性能调优、错误修复等原因,软件常常动态演化.现有测试技术难以满足软件演化过程中变化的测试需求,因此需要系统的测试用例演化技术,以有效保障演化软件的质量.回顾测试用例演化技术的研究现状,分别对测试用例选择、测试用例修复和测试用例集扩增这 3 部分内容进行了详细的比较和分析.最后提出测试用例演化技术领域存在的挑战和未来的研究方向.

关键词: 软件演化;回归测试;测试用例选择;测试用例修复;测试用例集扩增

中图法分类号: TP311 **文献标识码**: A

中文引用格式: 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

英文引用格式: Zhang ZY, Chen ZY, Xu BW, Yang R. Research progress on test case evolution. Ruanjian Xuebao/Journal of Software, 2013,24(4):663−674 (in Chinese). http://www.jos.org.cn/1000-9825/4379.htm

Research Progress on Test Case Evolution

ZHANG Zhi-Yi[1,2], CHEN Zhen-Yu[1,2], XU Bao-Wen[1], YANG Rui[1]

[1](State Key Laboratory for Novel Software Technology (Nanjing University), Nanjing 210093, China)

[2](Software Institute, Nanjing University, Nanjing 210093, China)

Corresponding author: CHEN Zhen-Yu, E-mail: zychen@software.nju.edu.cn

Abstract: Software is always dynamically evolving due to additional functionalities, performance tuning, bug fixing, and others. The existing testing techniques are difficult to satisfy the changing testing requirements. Hence, systematic techniques are needed of test case evolution to ensure the quality of evolving software. This paper surveys the test case evolution and compares and analyzes these techniques in detail, including test case selection, test case repairing and test suite augmentation. Finally, the challenges and research directions of test case evolution are proposed.

Key words: software evolving; regression testing; test case selection; test case repairing; test suite augmentation

软件系统在社会各个领域的作用日益重要,人们对软件系统的要求也日益增高,软件质量首当其冲.尽管存在代码审查、形式化验证等辅助手段,软件测试依然是目前最主要的软件质量保障手段.过去几十年,软件测试得到了软件工程研究人员和从业人员的广泛关注,产生了一批优秀的软件测试方法和支撑工具,为软件质量提供了有效保障.另一方面,软件测试依然是一项非常复杂和耗时费力的工作.在实际应用中,软件测试的资源往往有限,同时又需要我们全面检测软件行为以满足更多的测试需求.因此,随着软件规模的增大和软件技术的更新,软件测试同时也面临着许多挑战.一种理想的软件测试方法需要同时具有高错误检测能力、低成本消耗和较为广泛的适用性等特点.因此,软件测试通常根据不同软件的特点进行方法优化和技术改进,并将不同方法和

* 基金项目: 国家自然科学基金(61003024, 61170067, 61170071)
收稿时间: 2012-07-01; 修改时间: 2012-09-29; 定稿时间: 2013-01-25

技术进行融合[1].

由于功能增加、性能调优、软件重构、错误修复等原因,软件通常处于动态演化.随着软件的演化,新的测试用例不断产生,往往积累大量冗余测试用例.测试用例执行、管理和维护的开销相当大,而测试资源有限,因此我们希望从中挑选部分代表性测试用例,称为测试用例选择(test case selection).测试用例选择应尽可能满足不同的测试需求,从而提高其错误检测能力[2].

在演化软件测试的实际应用中,仅采用测试用例选择并不能完全解决软件演化带来的挑战:(1) 软件演化可能导致部分测试用例不可用,直接丢弃这些测试用例将降低错误检测能力;(2) 软件演化引发模块的增加和修改,已有测试用例不能完全覆盖这些模块[3].为了满足软件演化带来的测试挑战,研究者们提出了测试用例修复(test case repairing)和测试用例集扩增(test suite augmentation)技术.测试用例修复是指对旧版本程序测试用例集中的不可用测试用例进行修复,使得修复后的测试用例能够在新版本程序上执行.测试用例集扩增技术是指根据新版本程序和已有测试用例来生成新的测试用例,促使新测试用例能够覆盖新版本程序的修改部分和新增部分.

测试用例选择、测试用例修复和测试用例集扩增构成了测试用例演化的三大主要部分.其中,测试用例选择技术已比较成熟.近几年,测试用例修复和测试用例集扩增引起了软件测试研究者的广泛兴趣并展开了研究.本文主要回顾并总结了测试用例演化(包括测试用例选择、测试用例修复和测试用例集扩增)的研究现状.最后提出了测试用例演化技术目前存在的挑战和未来的研究方向.

本文第 1 节介绍测试用例演化的基本概念.第 2 节~第 4 节分别对测试用例选择、测试用例修复和测试用例集扩增技术的研究现状进行详细介绍.最后一节总结测试用例演化目前存在的挑战和未来研究方向.

1 测试用例演化概述

软件演化使得原有测试用例集不足以满足变化的测试需求,因而需要演化已有测试用例集以确保新版本软件的质量,我们称其为测试用例演化.具体定义如下:

定义 1. 设程序 P,P'为 P 的新版本程序,T 为 P 的测试用例集,通过对 T 进行选择、修复和扩增等操作,构建一个新测试用例集 T',使得 T'尽可能地满足 P'的测试需求,我们称其为测试用例演化.

图 1 是测试用例演化的示意图.由于在软件演化过程中常常需要删除、添加和修改软件的某些对象,从而导致部分测试用例不能正常执行,这些测试用例称为不可用测试用例.在新版本程序中仍能正常执行的测试用例称为可用测试用例.测试用例演化首先要识别已有测试用例中的可用测试用例和不可用测试用例,进而分别加以处理.

在软件演化过程中,测试用例集也在增长,可能导致大量冗余测试用例存在.新版本程序重复执行过多冗余测试用例造成测试资源浪费,因而需要进行选择,使得选出的部分测试用例能够尽可能地满足新版本修改部分的测试需求.测试用例选择通常针对的是可用测试用例,即能够在新版本程序运行的测试用例.

仅对已有测试用例进行选择并不足以满足演化软件的测试需求,为了进一步确保演化软件的质量,需要利用已有信息生成新的测试用例,主要包括测试用例修复和测试用例集扩增两类.

测试用例修复是指对不可用的测试用例进行添加、删除或修改等操作,使得修改后的测试用例能够在新版本上正确执行.在软件演化过程中,不可用测试用例因软件演化导致而成,修复后的不可用测试用例往往更具有针对性,从而比生成新测试用例更为有效.因此,修复不可用测试用例对于满足演化软件测试新需求具有重要意义.不可用测试用例可分为两类:可修复测试用例和不可修复测试用例.不可用测试用例能够被修复使其能够在新版本的程序上执行,我们称其为可修复测试用例.理论上,所有测试用例均可修复,但需要考虑其修复成本.我们通常将修复成本过高的测试用例称为不可修复测试用例.不可修复测试用例的一个简单处理就是丢弃.

测试用例(集)扩增是指根据已有的测试用例信息和程序演化信息来生成新的测试用例集,该测试用例集与经过选择和修复的测试用例集合并共同满足新版本的测试需求.测试用例集扩增主要针对程序的修改和新增部分.与传统测试用例生成不同,测试用例集扩增强调利用已执行测试用例信息和程序演化信息辅助生成新测

试用例,以期提高生成效率并获得更具针对性的测试用例.

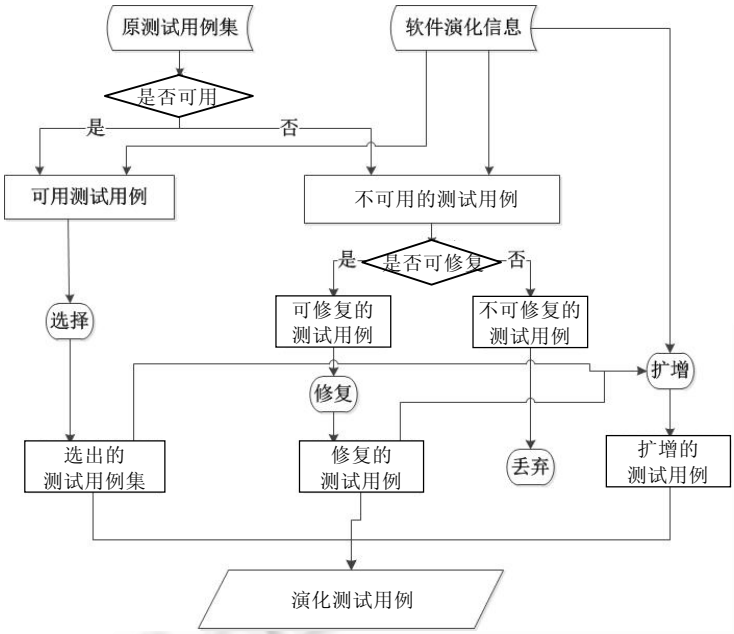

Fig.1 Test case evolution

图 1 测试用例演化

测试用例选择、测试用例修复和测试用例扩增之间相互依赖和补充.测试用例选择出的测试用例是对原有测试用例集的充分利用,是新版本测试用例集的重要组成部分.测试用例修复和测试用例扩增通过对原有测试用例集信息和程序演化信息的利用,生成满足新版本测试需求的测试用例.测试用例选择、测试用例修复和测试用例扩增通过不同形式对原有测试用例集信息进行充分利用,共同构成一个较为完善的测试用例集.

从应用场景和技术成熟度来看,三者有所不同.测试用例选择目前的技术较为成熟,可用于多种类型的程序和场景.测试用例修复对于 GUI 程序较为重要,可以解决 GUI 程序中因简单控件修改而导致大量测试用例不可用的问题.测试用例扩增的研究目前较少,已有的测试用例扩增技术主要用于控制台程序或是针对特定条件的扩增.从技术成本上看,测试用例集扩增成本高,测试用例修复次之,测试用例选择成本低.

我们重点分析测试用例选择、测试用例修复和测试用例集扩增这 3 个方面的研究现状和发展动态.

2 测试用例选择

测试用例选择是过去 20 年面向演化软件测试的研究热点之一[2,3].随着软件开发过程的不断迭代,测试用例集规模越来越大,从而导致大量冗余测试用例的存在.冗余测试用例的执行、管理和高维护成本将浪费大量资源,但测试资源往往有限,因此需要进行测试用例选择.测试用例选择的一个主要任务是从原有测试用例集中挑选部分测试用例并尽量满足新版本测试需求.根据不同的选择目标,测试用例选择分为 3 类:回归测试选择、测试用例集约简(test suite reduction)和测试用例优先级(test case prioritization)[2].其中,回归测试选择是测试用例演化技术中较为重要的一部分.

定义 2. 设程序 P,P'为 P 的新版本程序,T 为 P 的测试用例集,从 T 中选择子集 T',使得 T'尽可能地测试 P'中的修改部分,称为回归测试用例选择.

狭义回归测试选择主要确保软件修改没有引入新的故障.回归测试选择通过分析软件修改影响选择较小规模的 T',以达到较好的性价比.

若测试用例 t 能够检测 P'中的故障,我们称 t 为故障检测(fault-revealing)测试用例.回归测试中,通常假设 T

中所有测试用例在 P 中正确执行,则故障检测测试用例集简单定义为 $T_F=\{t\in T|P(t)\neq P'(t)\}$,其中,$P(t)$和 $P'(t)$分别是 t 在 P 和 P'的执行结果.直接求解 T_F 往往比较困难甚至不可行.Rothermel 等人提出一种安全的回归测试选择技术[4,5].首先引入一个新概念——修改经过(modification-traversing)的测试用例,即对于一个测试用例 t,我们称 t 为程序 P 和 P'修改经过测试用例当且仅当 t:(1) t 执行了 P'中的新代码或修改过的代码,或者(2) t 执行过包含在 P 中但在 P'被删除的代码.Rothermel 等人从 T 中选出一个修改经过子集 T_M.T_M 包含所有修改经过测试用例.需要注意的是,一个故障检测测试用例必然是修改经过的,反之不然.即,存在如下包含关系:$T_F\subseteq T_M\subseteq T$.另外,Rothermel 等人通过比较 P 和 P'的流图(如控制流图)并利用深度优先进行测试用例选择[6],同时,进一步将该方法扩展到基于程序间流图比较的测试用例选择[7].

在一些应用场景,选择的测试用例可能仍包含过多冗余测试用例,从而导致不能有效降低测试成本.为了进一步提高测试用例选择的精度,章宸等人将聚类分析技术引入回归测试选择中[8].他们首先收集所有测试用例的执行剖面(如函数调用序列),同时,基于执行剖面定义测试用例之间的距离(如欧氏距离),并对所有测试用例进行聚类.然后,针对每个类簇进行抽样(如 5%),若某一抽样测试用例为故障检测的,则选择该类簇中的所有测试用例.最终构成一个较小的回归测试用例集 T_C.虽然该聚类测试选择技术并非安全的测试用例选择技术,即它可能会丢失少量故障检测测试用例,但它能较大幅度地提高测试用例的选择精度(precision).他们的实验结果也验证了这一结论,并且显示,其同时还具有较高的召回率(recall).陈振宇等人采用语句块覆盖记录执行剖面进行测试用例的聚类选择[9].他们采用程序切片技术来大幅度地减少收集的语句块数目,从而有效降低测试用例的属性维度,提高了聚类效率.实验结果表明,该方法同时能够提高选择精度和故障检测测试用例的召回率.

Fischer 等人提出一种针对 FORTRAN 语言的测试用例选择方法[10,11].他们首先将被测程序分为多个程序块:每个程序块都只有一个输入接口和输出接口,且块之间是顺序执行的,然后将测试用例选择问题转换成整数规划问题.该方法无法处理控制流程图改变的程序.Harrold 等人提出了基于数据流分析的测试用例选择方法[12],在修改版本 P'中找出新的、修改的或删除的定义-使用对,然后选择执行过这些定义-使用对的测试用例.Yau 等人[13]通过静态分析程序代码及规格来对输入域进行划分,并保证每个输入域至少被测试用例执行 1 次,然后,通过符号执行对已有测试用例进行选择.Agrawal 等人提出了基于动态切片的选择方法[14]:记录测试用例执行的切片信息,包括动态切片信息和关联切片信息,并根据切片信息进行选择.Volkolos 等人提出了一种基于文本差异的测试用例选择方法[15,16].他们使用了一种名为 diff 的 Unix 工具来对当前版本 P 和修改版本 P'的源代码进行分析以确认 P'的修改部分.Bates 等人将程序依赖图和切片技术相结合,提出了一种基于程序依赖图切片技术的选择方法[17].Benedusi 等人引入路径分析概念,采用代数表达式建立路径模型进而选择测试用例[18].Chen 等人提出一种 TestTube 测试框架[19],并建立测试用例和程序实体之间的关联关系,然后选择经过修改程序实体的测试用例.Laski 等人[20]将控制流图中单入口和单出口的子图称为一个类簇,通过类簇的修改信息进行测试用例选择.

随着软件开发技术的发展,回归测试选择逐渐采用高层表示进行依赖分析.Leung 等人在软件集成的回归测试中引入防火墙(firewall)技术[21].他们将模块分为 3 类:NoChange:模块未被修改;OnlyCodeChange:模块代码修改但规格未修改;SpecChange:模块规格修改.对于模块 A 和 B,存在 9 种可能组合.这 9 种组合分为 3 类:(1) A 和 B 均为 NoChange;(2) A 和 B 均为 OnlyCodeChange 或者 SpecChange;(3) A 和 B 之一为 NoChange.其中,第 3 类作为修改依赖的边界(防火墙).该测试用例选择技术的基本思想是将修改模块的依赖传递隔离在防火墙内.该思想后来被广泛应用于面向对象程序和商业软件的回归测试中[22−24].Briand 等人提出一种基于 UML 设计模型的回归测试选择技术:首先,获取并维护一个代码级测试用例与设计模型之间的映射关系;然后,通过 UML 的形式化建模自动化地分析修改依赖关系,并据此选择代码级的测试用例[25−27].

综上所述,目前已存在多种回归测试选择技术,这些技术适用于不同测试环境和测试场景.经验发现,目前难以存在一种测试用例选择技术普遍优于其他选择技术[28].因此,测试人员常常需要根据不同测试需求选择合适的回归测试技术.限于篇幅,测试用例集约简和测试用例优先级在本文中不作介绍,可参阅文献[29,30].

3 测试用例修复

软件演化给测试用例维护带来很高的挑战[3].软件演化过程中,软件修改常常引起原有测试用例的失效.例如,软件中菜单、按钮或方法的增加、修改或删除都可能导致涉及该对象的测试用例不能正常运行[31−33].软件演化使得原有测试用例分为两类:可用测试用例与不可用测试用例.对于不可用测试用例,一种常用做法就是丢弃.然而,丢弃不可用测试用例存在两个风险:(1) 不可用测试用例所占比例可能很大[31−33],直接丢弃将浪费大量测试资源;(2) 不可用测试用例由软件演化引发产生,直接丢弃将导致漏测,从而降低测试用例集的错误检测能力.测试用例修复不但能够节省软件的测试成本,还能够获得更有针对性的测试用例,进而提高演化软件的测试效率.测试用例修复定义如下:

定义 3. 设程序 P,P'为 P 的新版本程序,T 为 P 的测试用例集,t 为 T 中的一个测试用例且 t 在 P'上无法正常执行,通过对 t 中的测试元素进行添加、修改、删除等修复操作得到一个新测试用例 t',使得 t'能够在 P'上正常执行,称为测试用例修复.

一般来说,不可用测试用例是由于该测试用例经过软件修改部分导致而成,修复这些测试用例往往比生成新的测试用例更为有效[31].测试用例修复首先需要对测试用例进行判定并分析不可用原因,然后根据软件演化分析演化信息及不可用原因并指导修复.测试用例修复的一个关键是需要界定修复范围.部分不可用测试用例由于修复成本过高而丢弃.

为方便读者对测试修复技术有一个大概了解,我们统计了目前已有的关于测试用例修复的文献.表 1 是关于这些文献的小结.表中第 1 列表示测试用例修复的文章名称,第 2 列表示该论文的发表年份,第 3 列代表该测试用例方法适用的程序类型,第 4 列代表该文章的关键字,即大概涉及的内容或技术.以“Repairing GUI Test Suites Using a Genetic Algorithm”文献为例,该文章发表于 2010 年,文中提出的修复技术适用于 GUI 程序,且与事件流图、遗传算法和用户图形界面等相关.在该表中,我们首先按照技术适用的程序类型将文章排序,对同一类型的则根据时间排序.下面,我们将按照测试用例修复技术适用的程序类型来分别详细介绍已有的测试用例修复技术.

Table 1 Summary of test case repairing artciles

表 1 测试用测修复文章总结

文章名称	发表年份	程序类型	关键字
Regression testing of GUIs[31]	2003	GUI程序	回归测试,测试用例修复,CUI测试, GUI控制流图,GUI调用图,调用树,事件分类
Automatically repairing event sequence-based GUI test suites for regression testing[34]	2008	GUI 程序	用户图形界面,回归测试,测试维护, 测试用例修复,测试用例管理
REST: A tool for reducing effort in script-based testing[35]	2008	GUI 程序	测试脚本,测试自动化,GUI 回归测试
Maintaining and evolving GUI-directed test scripts[36]	2009	GUI 程序	GUI 回归测试,测试用例修复,测试脚本, 测试用例保持和发展
Repairing GUI test suites using a genetic algorithm[37]	2010	GUI 程序	GUI 回归测试,遗传算法,测试用例修复, GUI 事件流图
Automated GUI refactoring and test script repair[38]	2011	GUI程序	用户图形界面,GUI重构, GUI自动化测试,GUI维护
Automated session data repair for Web application regression testing[39]	2008	Web 程序	回归测试,Web 应用
WATER Web application test repair[40]	2011	Web 程序	Web 应用,测试用例修复
Automatically repairing test cases for evolving method declarations[41]	2010	白盒测试	测试用例修复,测试自动化, 方法参数,数据流图
ReAssert: Suggesting repairs for broken unit tests[42]	2009	白盒测试	单元测试,测试用例修复,
ReAssert: A tool for repairing broken unit tests[43]	2011	白盒测试	单元测试,测试工具, 测试用例修复,ReAssert
On test repair using symbolic execution[44]	2010	白盒测试	测试用例修复,符号执行, ReAssert,Pex

3.1 GUI测试用例修复

测试用例修复技术最初主要应用于特定领域的测试,如 GUI 测试.Memon 等人在 2003 年提出 GUI 回归测试中不可用测试用例的修复问题[31].该文阐述了识别不可用测试用例的基本思想,并将 GUI 回归测试中测试用例不可用的原因归为两个:起始状态不正确或中间事件对的非法执行顺序.Memon 等人认为,起始状态不正确的测试用例不可修复,因此他们只对包含中间事件对非法执行顺序的不可用测试用例进行修复.他们首先界定了修复技术的限制条件:(1) 对象名称唯一;(2) 除非对象被修改,否则新版本中的对象名称必须与旧版本一致.他们运用 GUI 事件流图建立 GUI 回归测试框架.对于每个 GUI 回归测试框架,都存在一个顶点集合(代表 GUI 程序中的一个事件)和一个边集合(代表两个事件之间可以顺序执行).通过将旧版本 GUI 顶点集合与新版本 GUI 顶点集合相减,可以得到一个顶点集合(删除顶点集合),该集合中的顶点只存在于旧版本中.同理,也可以得到一个边集合(删除边集合).他们选择所有经过删除顶点集合或删除边集合的测试用例,并对这些测试用例进行修复.假设一个 GUI 测试用例$\{e_1,e_2,\ldots,e_n\}$(e_x代表 GUI 程序中的一个事件,e_x和 e_{x+1}顺序执行,代表边),对于选出的测试用例,有两种修复方法:(1) 从测试用例中被删除事件 e_i(或是被删除边连接的前事件)出发向后寻找,直至寻找到一个新事件 e_j,使得 e_j可以紧跟在 e_{i-1}后面执行.将事件$\{e_i,e_{i+1},\ldots,e_{j-1}\}$删除,即可得修复后的测试用例;(2) 如果存在一个事件 e_x,且$\{e_{i-1};e_x;e_j\}$可以顺序执行,则将事件$\{e_i,e_{i+1},\ldots,e_{j-1}\}$删除,并在 e_{i-1}事件后插入新事件 e_x以得到修复后的测试用例.由于此修复方法可能产生多个不同的修复结果,Memon 等人对修复后产生测试用例的个数进行了限制.实验结果表明,该 GUI 的测试用例修复技术是有效的[34].

Huang 等人采用另外一种修复策略对不可用测试用例集进行自动修复[37],他们将遗传算法引入了测试用例修复中.在该策略中,他们将每个测试用例视为一个染色体,通过交叉变异技术对不可用测试用例进行修复,然后计算其适应度.如果适应度达到一定值,则该测试用例被认为修复成功.重复这样的过程,直至没有可修复的不可用测试用例.通过实验,结果表明,使用遗传算法进行测试用例修复比随机算法更加有效.但该技术并非面向软件演化,而是为了解决由于事件流图(EFG)不精确导致的不可用测试用例问题.

由于人工进行 GUI 黑盒测试代价较大,Grechanik 等人重点研究了 GUI 待测软件和测试脚本之间的关系,并实现了一种开发工具 REST,用于指导测试人员对 GUI 程序和测试脚本进行分析[35].他们通过研究 GUI 对象变化及其对应的测试脚本关系,提出一种维护黑盒测试脚本的方法[36],即:确定 GUI 程序的修改对象,并利用此信息确认测试脚本中被修改部分影响的语句.由于一个测试脚本中的其他语句可能使用修改语句中计算的变量值,他们也对这些使用修改语句变量值的语句进行了分析.通过对 GUI 变化对象和受影响测试脚本语句的分析,该方法能够自动产生一些修复的提示信息,辅助有经验的测试人员修复不可用测试脚本,完成 GUI 测试脚本的维护和演化.该方法可辅助于 QTP 等目前业界常用的 GUI 测试工具.

与上面的黑盒修复方法不同,Daniel 等人[38]采用白盒方法进行 GUI 脚本的修复,他们将静态分析和动态分析相结合,用静态分析方法对 GUI 脚本进行分析并建立状态结果,然后用动态分析方法将 GUI 脚本和修改结合进行测试用例修复.该方法的优势是在修复时不需要执行所有的测试脚本,相反地,它能够直接获得只被 GUI 修改部分影响的测试用例集.

3.2 Web测试用例修复

Alshahwan 等人提出了一种完全自动化的 Web 测试用例修复技术[39].首先对修改的 Web 程序构造进行白盒测试以确认修改部分,然后对不可用测试用例进行 URL 参数和执行顺序的修复.如果 URL 中包含新的参数,赋予新参数有意义的初始值.如果对不可用测试用例进行执行顺序修复,则分为 3 种情况:边删除、点删除和其他情况.如果程序修改导致两个 Web 页面之间的链接(边删除)或一个 Web 页面被删除(点删除),则只需找到一个能够链接起始页面和目标页面的新路径,即可认为该测试用例修复成功.如果不存在一个能够完成修复的新路径(其他情况),则将该测试用例从被删除节点(边删除或点删除)断开,这样,原测试用例变成两个单独的测试用例.如果这两个测试用例都可以顺利执行,则对原测试用例完成修复.实验结果表明,该技术修复的测试用例不仅能够发现大部分用于失效网页中的数值,而且能够发现多个未被用户对话覆盖的变化.Choudhary 等人[40]进

一步对 Web 测试用例修复中可能存在的问题进行了研究.他们通过对 Web 网页进行分析,发现 4 种导致测试用例不可用的问题,包括未选择问题、错误选择问题、数据构建问题和过期内容问题.他们分别针对这 4 种情况进行测试用例修复.对于未选择问题和错误选择问题,将定位器进行更新,并根据新定位器对测试用例中的命令进行修改.如果没有发现定位器信息,则对过期内容问题进行修改,然后检查断言是否将实际输出值和预期输出值进行比较:如果有,则将预期值替代为实际值;否则,忽略此测试用例中的断言命令.对于数据构建问题,首先找到构建命令并确定新元素,然后对这些新元素赋予随机值.

3.3 白盒测试用例修复

白盒测试用例修复问题也是近年来的一个热门研究方向.在程序演化中,由于方法中参数和返回值的改变,部分原有测试用例集可能无法使用.Mirzaaghaei 等人研究了此类不可用测试用例的修复问题[41],通过整合 SVN, Jdiff,Soot 等工具,对源代码和数据流图进行分析,能够自动检测方法的参数变化情况,并通过数据流图对测试用例中的新参数进行初始化.如果方法中参数被修改或有新参数增加,则添加一个命令,定义一个新的参数,并对参数进行初始化.如果方法返回值类型发生改变,则从测试对象中获得返回值数据.该修复技术仅限于解决由于参数和返回值修改而导致的测试用例不可用问题.

Daniel 等人研发的辅助工具 ReAssert 主要用于单元测试的测试用例修复,能够自动修复不可用单元测试用例[42,43].ReAssert 主要针对由于程序修改而导致的断言失效,提出了 8 种断言自动修复策略进行不可用测试用例的修复.通过实验,结果表明,ReAssert 可以修复大部分断言引发的异常测试用例.但 ReAssert 仍然不够成熟,表现在:(1) ReAssert 仅能够修复部分测试用例;(2) ReAssert 的部分修复结果是次优的,即测试用例能够执行,但并不能很好地满足预定的测试需求.Daniel 等人进一步提出了基于符号执行的测试用例修复方法[44],将 ReAssert 和符号执行工具相结合,使用程序插桩跟踪确定断言失效的地点,通过符号执行和约束求解计算新的断言形式和预期结果.

4 测试用例集扩增

尽管选择和修复已有测试用例可以检测演化软件的部分行为,但对于修改模块尤其是新增模块,已有测试用例仍难以满足测试需求.为了全面测试演化软件,在回归测试中通常需要生成新的测试用例.回归测试中,测试用例生成与传统测试用例生成有所不同.经过测试用例选择和测试用例修复的检测后,演化软件形成一组特定的测试需求.对于这类需求,尽管已经存在一些自动化生成工具[45−47],测试用例生成仍然需要较高代价且针对性不强.因此,人们提出了面向软件演化的测试用例生成技术,通常称其为测试用例集扩增(test suite augmentation)或测试用例扩增.具体定义如下:

定义 4. 设程序 P,P'为 P 的新版本程序,T 为 P 的测试用例集,通过利用已执行测试用例信息和程序修改信息生成新测试用例集 T',使得 T'中的测试用例可以覆盖 P'的修改和新增部分,$T \cup T'$满足演化软件的测试需求,称为测试用例集扩增.

与传统测试用例生成不同,测试用例集扩增具有两个特点:(1) 新产生的测试用例只需覆盖演化软件的修改和新增部分;(2) 测试用例集扩增需要充分利用已执行测试信息并结合软件演化信息辅助新测试用例生成.

测试用例集扩增一词最早可能由 Harder 等人引入[48].该词的提出最初是为与 test suite minimization 相对应.在文献[48]中,Harder 重点阐述了如何进行测试用例集约简,但并未介绍如何进行测试用例集扩增.我们将测试用例集扩增的相关工作总结在表 2 中.表中第 1 列为测试用例扩增论文名称,第 2 列为论文的发表年份,第 3 列为论文的关键字,用于描述涉及内容和技术.以“Test-Suite Augmentation for Evolving Software”论文为例,该文发表于 2008 年,文中提出的扩增技术与控制依赖和数据依赖相关.

目前,已有的测试用例集的产生方法大体上分为两类:第 1 类是面向行为的测试用例扩增技术,即期望新测试用例显示演化软件的执行变化;第 2 类是面向覆盖的测试用例集扩增技术,即期望新测试用例覆盖演化软件的新增和修改部分.

Table 2 Summary of test suite augmentation artciles

表 2 测试用例扩增文章总结

文章名称	发表年份	关键字
Test-Suite augmentation for evolving software[49]	2008	测试用例扩增,软件演化,控制依赖,数据依赖
Automated scalable test-suite augmentation for evolving software[50]	2009	测试用例扩增,PIE 模型
Applying aggressive propagation-based strategies for testing changes[51]	2011	测试用例扩增,传播策略,谓词约束求解,MATRIX
Directed test suite augmentation[52]	2009	回归测试,测试用例扩增,concolic 测试
Factors affecting the use of genetic algorithms in test suite augmentation[53]	2010	回归测试,测试用例扩增,遗传算法,经验研究
Directed test suite augmentation: techniques and tradeoffs[54]	2010	回归测试,测试用例扩增,concolic 测试,遗传算法
Guided path exploration for regression test generation[55]	2009	测试用例扩增,动态符号执行,分支节点
Test generation to expose changes in evolving programs[56]	2010	软件演化,测试用例生成,符号执行

4.1 面向行为的测试用例集扩增技术

Santelices 等人提出仅采用控制流图或数据流图并不足以进行测试用例扩增,应结合符号执行技术对控制依赖和数据依赖进行分析并进行测试用例集扩增[49].同时,提出扩增的测试用例集应该满足两个需求:链需求和状态需求.若生成的测试用例集能够覆盖到所有以修改点为起始点的依赖链,则满足链需求.如果扩增的测试用例满足下面两个条件之一,则满足状态需求:(1) 测试用例在新版本软件能够到达依赖链结尾且结尾处活动变量有不同符号值;(2) 测试用例在新版本软件中执行时没有到达依赖链结尾但在原版本中能够到达依赖链结尾.链需求能够提高扩增测试用例集输出不同结果的可能性;状态需求能够保证修改部分不同输出结果可以传递到依赖链结尾并提高扩增测试用例的质量.若程序中存在多个缺陷,一个修改可能会影响另一个修改的状态,因此,在进行测试用例集扩增时只考虑链需求.同时,Santelices 等人明确提出了测试用例集扩增的两个步骤[50]:(1) 对演化版本的新行为进行评估以确认新行为是否能够被原版本测试用例集执行覆盖;(2) 将产生的新测试用例对剩余未被测试的新行为进行测试.

Santelices 等人进一步将传播技术引入到测试用例集扩增技术中.首先,将测试用例集扩增技术和 PIE 模型[57]相结合.PIE 模型描述了失效观测的 3 个条件:执行故障代码(E);产生错误状态(I);错误传播至输出(P).由于早期 MATRIX 工具[58]的限制,符号执行不能很好地扩展及多种测试要求不能满足.Santelices 等人引入传播分析和谓词约束求解进行测试用例集扩增[51].该方法在测试时并不需要预先计算需求条件,而是在程序执行期间动态地加以计算.程序执行时检查程序的依赖和状态区别以计算需求条件.满足执行需求条件的新测试用例添加至测试用例集中.该方法在保证 MATRIX 准确性的同时减少了符号执行的代价和进行预分析的代价.Santelices 等人开发的工具能够对演化软件的测试需求进行识别(识别出修改部分的分支覆盖、定义-使用覆盖等),然后,通过符号执行和约束求解获得测试输入.该工具同时记录已满足的测试需求,当所有测试需求满足时完成测试用例集的扩增.

Qi 等人提出面向程序修改的测试用例集扩增技术[56],即在路径导向过程中采用动态符号执行以期获得更好的结果.同时,Qi 等人通过修改影响传递树发现,如果修改变量在修改后未被执行或执行后对结果值没有影响,则修改不影响最终结果改变.该发现能够保证扩增的测试用例集都能够影响输出结果.Jiang 等人对协议软件的规模演变进行了回归测试分析[59],提出一种回归测试中的测试用例集扩增技术.即对程序建立起图模型后,如果一条边未被测试用例覆盖,则寻找一条链接初始节点和未覆盖边源节点的最短路径,将这些最短路径组合形成子图并分成新测试用例图,据此产生测试用例.

4.2 面向覆盖的测试用例集扩增技术

Xu 等人将 concolic testing 技术[60]引入到测试用例集扩增技术中,提出导向测试用例集扩增技术[52].通过对受影响测试用例的再执行,得到修改版本的一个分支集合,该集合中的分支未被测试用例覆盖.最后,结合实体

测试和符号测试产生新的测试用例.该技术可以保证扩增测试用例集对软件修改部分能够达到更高的覆盖率.Taneja 等人将无用路径分支去除,以减少测试用例集扩增代价[55].Xu 等人进一步引入遗传算法进行测试用例集扩增[53],并详细分析了遗传算法中各参数设置对测试用例集扩增结果的影响.这些参数包括确认受影响元素的算法、已存在测试用例的特性、受影响元素的考虑顺序和测试用例被执行的方法.实验结果表明,测试用例被执行方法的不同可能对测试用例集扩增有比较明显的影响,为使用者提供了一些参考依据.Rothermel 等人对遗传算法[61]和 concolic testing 在测试用例集扩增中的作用进行了详细的比较、分析[54],该实验使用分支覆盖作为检验准则.对于需要测试的分支,若存在一个没有被测试用例覆盖的点,则产生新测试用例,然后判断该点是否被扩增测试用例覆盖到,直到所有需要测试的点都被测试用例覆盖到为止.实验结果表明,使用 concolic testing 产生的测试用例更加有效,而使用遗传算法在产生测试用例时更加灵活.与面向行为的扩增技术不同,上述技术并未针对输出差异性,而是强调增加某种给定准则的覆盖率.

5 挑战与趋势

目前,尽管存在众多回归测试方面的研究成果,但关于测试用例演化研究尚不完善.仅进行测试用例选择难以保证演化软件质量,需要对演化软件进行测试用例修复和扩增.相对于测试用例选择,测试用例修复和测试用例集扩增的研究相对缺乏,在未来具有较好的研究价值.测试用例选择、修复和扩增的目标不同,目前这 3 方面的研究基本独立.但在软件演化过程中,这 3 方面紧密相关,均需利用原有测试用例信息和软件演化信息.因此,演化软件的测试需要一个多阶段融合的测试用例演化技术,进一步探索不同阶段测试信息的融合和互补,并形成一套完整的测试用例演化框架,以完整地满足演化软件测试的需求.综上所述,测试用例演化未来的研究可能集中在以下 3 个方面:

(1) 演化驱动的测试用例修复.测试用例修复是测试用例演化的重要组成部分.对不可用测试用例进行修复,不但能够节省测试成本,而且能够对演化软件进行更有效的测试.已有测试用例修复技术仍不完善:首先,对于测试用例是否可修复的判定,仍未有一个明确的定义,一种极端情况是,如果删除不可用测试用例中的所有元素并产生新的元素,则所有不可用测试用例都能够被修复;其次,修复技术应用领域仍较为狭隘.目前,测试用例修复主要针对 GUI 和 Web 测试,这些测试用例通常基于事件流,修复较为简单.其他类型的测试用例修复研究较少,缺乏自动修复测试用例的工具;

(2) 演化驱动的测试用例集扩增.选择和修复后的测试用例集能够检测演化软件的部分行为,但对于修改和新增部分,已有的测试用例集难以满足测试需求,因此需要测试用例集扩增.已有测试用例集扩增主要是利用已运行的测试用例集动态信息结合软件演化的静态信息产生新的测试用例,但这些测试用例集扩增技术主要依赖符号执行等传统生成技术,限制条件多且难以用于大规模程序.不同测试用例生成方法和测试用例集扩增效率仍缺乏一个系统的比较分析.缺乏测试用例集的自动扩增工具;

(3) 多阶段融合的测试用例演化框架.演化软件的测试主要集中在回归测试选择技术,而测试用例修复和测试用例集扩增的研究较少,同时也缺乏一个系统而完整的测试用例演化框架.测试用例演化框架是测试基础,负责协调测试用例选择、测试用例修复和测试用例集扩增这 3 部分.从成本上看,测试用例选择最小,修复次之,而扩增最大.演化软件测试的一个基本策略是先对原有测试用例集进行选择重新测试,然后对不可用测试用例进行修复,最后进行测试用例集扩增.测试用例演化过程需要一个整体测试需求作为终止准则,并将前期运行信息结合软件演化信息进行逐步引导,直至达到预定测试需求或耗尽指定测试资源.如何将测试用例选择、修复和扩增技术相结合形成一个多阶段融合框架,是未来测试用例演化的一个重要目标.

References:

[1] Bertolino A. Software testing research: Achievements, challenges, dreams. In: Briand LC, Wolf AL, eds. Proc. of the Future of Software Engineering (FOSE 2007). Minneapolis: ACM Press, 2007. 85−103. [doi: 10.1002/stvr.430]

[2] Yoo S, Harman M. Regression testing minimization, selection and prioritization: A survey. Journal of Software Testing, Verification and Reliability, 2012,22(2):67−120. [doi: 10.1002/stv.430]

[3] Harrold MJ, Orso A. Retesting software during development and maintenance. In: Müller HA, ed. Proc. of the Frontiers of Software Maintenance (FoSM 2008). Beijing: IEEE Computer Society Press, 2008. 99−108. [doi: 10.1109/FOSM.2008.4659253]

[4] Rothermel G, Harrold MJ. A safe, efficient regression test selection technique. ACM Trans. on Software Engineering and Methodology, 1997,6(2):173−210. [doi: 10.1145/248233.248262]

[5] Bible J, Rothermel G, Rosenblum DS. A comparative study of coarse- and fine-grained safe regression test-selection techniques. ACM TOSEM, 2001,10(2):149−183. [doi: 10.1145/367008.367015]

[6] Rothermel G, Harrold MJ. A safe, efficient algorithm for regression test selection. In: Proc. of the Int'l Conf. on Software Maintenance (ICSM 2003). Montreal: IEEE Computer Society Press, 1993. 358−367. [doi: 10.1109/ICSM.1993.366926]

[7] Rothermel G, Harrold MJ. Selecting tests and identifying test coverage requirements for modified software. In: Proc. of the Int'l Symp. on Software Testing and Analysis (ISSTA'94). Seattle: ACM Press, 1994. 169−184. [doi: 10.1145/186258.187171]

[8] Zhang C, Chen Z, Zhao Z, Yan S, Zhang J, Xu B. An improved regression test selection technique by clustering execution profiles. In: Proc. of the Int'l Conf. on Quality Software (QSIC 2010). Zhangjiajie: IEEE Reliability Society Press, 2010. 171−179. [doi: 10.1109/QSIC.2010.16]

[9] Chen Z, Duan Y, Zhao Z, Xu B, Qian J. Using program slicing to improve the efficiency and effectiveness of cluster test selection. Int'l Journal of Software Engineering and Knowledge Engineering, 2011,21(6):759−777. [doi: 10.1142/S0218194011005487]

[10] Fischer K. A test case selection method for the validation of software maintenance modifications. In: Proc. of the Int'l Computer Software and Applications Conf. (COMPSAC'77). Chicago: IEEE Computer Society Press, 1977. 421−426.

[11] Fischer K, Raji F, Chruscicki A. A methodology for retesting modified software. In: Proc. of the National Telecommunications Conf. New Orleans: IEEE Computer Society Press, 1981. 1−6.

[12] Harrold MJ, Soffa ML. Interprocedual data flow testing. In: Proc. of the 3rd ACM SIGSOFT Symp. on Software Testing, Analysis, and Verification (TAV3). ACM Press, 1989. 158−167. [doi: 10.1145/75308.75327]

[13] Yau SS, Kishimoto Z. A method for revalidating modified programs in the maintenance phase. In: Proc. of the Int'l Computer Software and Applications Conf. (COMPSAC'87). Tokyo: IEEE Computer Society Press, 1987. 272−277.

[14] Agrawal H, Horgan JR, Krauser EW, London SA. Incremental regression testing. In: Proc. of the Int'l Conf. on Software Maintenance (ICSM'93). Montréal: IEEE Computer Society Press, 1993. 348−357. [doi: 10.1109/ICSM.1993.366927]

[15] Vokolos F, Pythia FP. A regression test selection tool based on text differencing. In: Proc. of the Int'l Conf. on Reliability Quality and Safety of Software Intensive Systems. London: ROYAUME-UNI, 1997.

[16] Vokolos F, Frankl P. Empirical evaluation of the textual differencing regression testing technique. In: Proc. of the IEEE Int'l Conf. on Software Maintenance (ICSM'98). Bethesda: IEEE Computer Society Press, 1998. 44−53. [doi: 10.1109/ICSM.1998.738488]

[17] Bates S, Horwitz S. Incremental program testing using program dependence graphs. In: Proc. of the 20th ACM SIGPLAN-SIGACT Symp. on Principles of Programming Languages. Charleston: ACM Press, 1993. 384−396. [doi: 10.1145/158511.158694]

[18] Benedusi P, Cmitile A, De Carlini U. Post-Maintenance testing based on path change analysis. In: Proc. of the Int'l Conf. on Software Maintenance (ICSM'88). Santa Fe: IEEE Computer Society Press, 1988. 352−361. [doi: 10.1109/ICSM.1988.10187]

[19] Chen YF, Rosenblum D, Vo KP. Testtube: A system for selective regression testing. In: Proc. of the 16th Int'l Conf. on Software Engineering (ICSE'94). Sorrento: ACM Press, 1994. 211−220. [doi: 10.1109/ICSE.1994.296780]

[20] Laski J, SzermerW. Identification of program modifications and its applications in software maintenance. In: Proc. of the Int'l Conf. on Software Maintenance (ICSM'92). Chicago: IEEE Computer Society Press, 1992. 282−290. [doi: 10.1109/ICSM.1992.242533]

[21] Leung HKN, White L. A study of integration testing and software regression at the integration level. In: Proc. of the Int'l Conf. on Software Maintenance (ICSM'90). San Diego: IEEE Computer Society Press, 1990. 290−301. [doi: 10.1109/ICSM.1990.131377]

[22] White L, Robinson B. Industrial real-time regression testing and analysis using firewalls. In: Proc. of the Int'l Conf. on Software Maintenance (ICSM 2004). Chicago: IEEE Computer Society Press, 2004. 18−27. [doi: 10.1109/ICSM.2004.1357786]

[23] White L, Almezen H, Sastry S. Firewall regression testing of GUI sequences and their interactions. In: Proc. of the Int'l Conf. on Software Maintenance (ICSM 2003). Amsterdam: IEEE Computer Society Press, 2003. 398−409. [doi: 10.1109/ICSM.2003.1235450]

[24] Zheng J, Robinson B, Williams L, Smiley K. Applying regression test selection for COTS-based applications. In: Proc. of the Int'l Conf. on Software Engineering (ICSE 2006). Shanghai: ACM Press, 2006. 512−522. [doi: 10.1145/1134357]

[25] Briand LC, Labiche Y, Buist K, Soccar G. Automating impact analysis and regression test selection based on UML designs. In: Proc. of the Int'l Conf. on Software Maintenance (ICSM 2002). Montréal: IEEE Computer Society Press, 2002. 252−261. [Regression testing UML designs]

[26] Briand LC, Labiche Y, He S. Automating regression test selection based on UML designs. Information and Software Technology, 2009,51(1):16−30. [doi: 10.1016/j.infsof.2008.09.010]

[27] Pilskalns O, Uyan G, Andrews A. Regression testing UML designs. In: Proc. of the Int'l Conf. on Software Maintenance (ICSM 2006). Dublin: IEEE Computer Society Press, 2006. 254−264. [doi: 10.1109/ICSM.2006.53]

[28] Engstrom E, Runeson P, Skoglund M. A systematic review on regression test selection techniques. Information and Software Technology, 2010,52(1):14−30. [doi: 10.1016/j.infsof.2009.07.001]

[29] Zhang XF, Chen L, Xu BW, Nie CH. Survey of test suite reduction problem. Journal of Frontiers of Computer Science and Technology, 2008,2(3):235−247 (in Chinese with English abstract).

[30] Qu B, Nie CH, Xu BW. Survey of test case prioritization for regression testing. Journal of Frontiers of Computer Science and Technology, 2009,3(3):225−233 (in Chinese with English abstract).

[31] Memon A, Soffa ML. Regression testing of GUIs. In: Proc. of the 9th European Software Engineering Conf. on Held Jointly with the 11th ACM SIGSOFT Int'l Symp. on Foundations of Software Engineering (ESEC/SIGSOFT FSE 2003). New York: ACM Press, 2003. 118−127. [doi: 10.1145/940071.940088]

[32] Stack Overflow. Program evolution and broken tests. http://stackoverflow.com/questions/2054171/

[33] C2 Wiki. Deleting broken unit tests. http://c2.com/cgi-bin/wiki?DeletingBrokenUnitTests

[34] Memon A. Automatically repairing event sequence-based GUI test suites for regression testing. ACM Trans. on Software Engineering and Methodology, 2008,18(2):Article 4. [doi: 10.1145/1416563.1416564]

[35] Xie Q, Grechanik M, Fu C. Rest: A tool for reducing effort in script-based testing. In: Proc. of the Int'l Conf. on Software Maintenance (ICSM 2008). Beijing: IEEE Computer Society Press, 2008. 468−469. [doi: 10.1109/ICSM.2008.4658108]

[36] Grechanik M, Xie Q, Fu C. Maintaining and evolving GUI-directed test scripts. In: Proc. of the Int'l Conf. on Software Engineering (ICSE 2006). Vancouver: ACM Press, 2006. 408−418. [doi: 10.1109/ICSE.2009.5070540]

[37] Huang S, Cohen MB, Memon AM. Repairing GUI test suites using a genetic algorithm. In: Proc. of the 1st Int'l Conf. on Software Testing, Verification and Validation (ICST 2010). Paris: IEEE Computer Society Press, 2010. 245−254. [doi: 10.1109/ICST.2010.39]

[38] Daniel B, Luo Q, Mirzaaghaei M, Dig D, Marinov D, Pezzè M. Automated GUI refactoring and test script repair. In: Proc. of the 1st Int'l Workshop on End-to-End Test Script Engineering (ETSE 2011). Toronto: ACM Press, 2011. [doi: 10.1145/2002931.2002937]

[39] Harman M, Alshahwan N. Automated session data repair for Web application regression testing. In: Proc. of the 1st Int'l Conf. on Software Testing, Verification and Validation (ICST 2008). Lillehammer: IEEE Computer Society Press, 2008. 298−307. [doi: 10.1109/ICST.2008.56]

[40] Choudhary SR, Zhao D, Versee H, Orso A. WATER: Web application TEst repair. In: Proc. of the 1st Int'l Workshop on End-to-End Test Script Engineering (ETSE 2011). Toronto: ACM Press, 2011. [doi: 10.1145/2002931.2002935]

[41] Mirzaaghaei M, Pastore F, Pezzè M. Automatically repairing test cases for evolving method declarations. In: Proc. of the Int'l Conf. on Software Maintenance (ICSM 2010). Timişoara: IEEE Computer Society Press, 2010. 1−5. [doi: 10.1109/ICSM.2010.5609549]

[42] Daniel B, Jagannath V, Dig D, Marinov D. ReAssert: Suggesting repairs for broken unit tests. In: Proc. of the 24th IEEE/ACM Int'l Conf. on Automated Software Engineering (ASE 2009). Auckland: IEEE Computer Society Press, 2009. 433−444. [doi: 10.1109/ASE.2009.17]

[43] Daniel B, Dig D, Gvero T, Jagannath V, Jiaa J, Mitchell D, Nogiec J, Tan SH, Marinov D. ReAssert: A tool for repairing broken unit tests. In: Proc. of the Int'l Conf. on Software Engineering (ICSE 2011). Hawaii: ACM Press, 2011. 1010−1012. [doi: 10.1145/1985793.1985978]

[44] Daniel B, Gvero T, Marinov D. On test repair using symbolic execution. In: Proc. of the Int'l Symp. on Software Testing and Analysis (ISSTA 2010). Trento: ACM Press, 2010. 207−218. [doi: 10.1145/1831708.1831734]

[45] Visser W, Pasareanu C, Khurshid S. Test input generation with Java pathfinder. In: Proc. of the Int'l Symp. on Software Testing and Analysis (ISSTA 2004). Boston: ACM Press, 2004. 97−107. [doi: 10.1145/1007512.1007526]

[46] Sen K, Marinov D, and Agha G. CUTE: A concolic unit testing engine for C. In: Proc. of the 10th European Software Engineering Conf. on Held Jointly with the 13th ACM SIGSOFT Int'l Symp. on Foundations of Software Engineering (ESEC/FSE 2005). Lisbon: ACM Press, 2005. 263−272. [doi: 10.1145/1081706.1081750]

[47] Chen TY, Merkel R. Quasi-Random testing. IEEE Trans. on Reliability, 2007,56(3):562−568. [doi: 10.1109/TR.2007.903293]

[48] Harder M, Mellen J, Ernst MD. Improving test suites via operational abstraction. In: Proc. of the Int'l Conf. on Software Engineering (ICSE 2003). Hilton Portland: ACM Press, 2003. 60−71. [doi: 10.1109/ICSE.2003.1201188]

[49] Santelices RA, Chittimalli PK, Apiwattanapong T, Orso A, Harrold MJ. Test-Suite augmentation for evolving software. In: Proc. of the 23th IEEE/ACM Int'l Conf. on Automated Software Engineering (ASE 2008). L'Aquila: IEEE Computer Society Press, 2008. 218−227. [doi: 10.1109/ASE.2008.32]

[50] Santelices RA, Harrold MJ. Automated scalable test-suite augmentation for evolving software. In: Proc. of the Int'l Conf. on Software Engineering-Companion (ICSE Companion 2009). Vancouver: ACM Press, 2009. 379−382. [doi: 10.1109/ICSE-COMPANION.2009.5071026]

[51] Santelices RA, Harrold MJ. Applying aggressive propagation-based strategies for testing changes. In: Proc. of the Int'l Conf. of Software Testing, Verification and Validation (ICST 2011). Berlin: IEEE Computer Society Press, 2011. 11−20. [doi: 10.1109/ICST.2011.46]

[52] Xu Z, Rothermel G. Directed test suite augmentation. In: Proc. of the Asia-Pacific Software Engineering Conf. (APSEC 2009). Penang: ACM Press, 2009. 406−413.

[53] Xu ZH, 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). Portland: ACM Press, 2010. 1365−1372. [doi: 10.1145/1830483.1830734]

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

[55] Taneja K, Xie T, Tillmann N, Halleux J, Schulte W. Guided path exploration for regression test generation. In: Proc. of the Int'l Conf. on Software Engineering (ICSE 2009). Vancouver: ACM Press, 2009. 311−314. [doi: 10.1109/ICSE-COMPANION.2009.5071009]

[56] Qi D, Roychoudhury A, Liang Z. Test generation to expose changes in evolving programs. In: Proc. of the 25th IEEE/ACM Int'l Conf. on Automated Software Engineering (ASE 2010). UniMail: IEEE Computer Society Press, 2010. 397−406. [doi: 10.1145/1858996.1859083]

[57] Watt SK. Difficult dialogues, privilege, and social justice: Uses of the privileged identity exploration (PIE) model in student affairs practice. College Student Affairs Journal, 2007,26(2):114−126.

[58] Apiwattanapong T, Santelices R, Chittimalli PK, Orso A, Harrold MJ. Matrix: Maintenance-Oriented testing requirement identifier and examiner. In: Proc. of the Testing: Academic and Industrial Conf.—Practice and Research Techniques (TAIC-PART 2006). Windsor: Swell Press, 2006. 137−146.

[59] Jiang B, Tse TH, Grieskamp W, Kicillof N, Cao Y, Li X. Regression testing process improvement for specification evolution of real-world protocol software. In: Proc. of the Int'l Conf. on Quality Software (QSIC 2010). Zhangjiajie: IEEE Reliability Society Press, 2010. 62−71. [doi: 10.1109/QSIC.2010.55]

[60] Cadar C, Ganesh V, Pawlowski PM, Dill DL, Engler DR. Exe: Automatically enerating inputs of death. In: Proc. of the 13th ACM Conf. on Computer and Communications Security (CCS 2006). Alexandria: ACM Press, 2006. 322−335. [doi: 10.1145/1180405.1180445]

[61] McMinn P. Search-Based software test data generation: A survey. Software Testing, Verification and Reliability, 2004,14(2): 105−156. [doi: 10.1002/stvr.294]

附中文参考文献:

[29] 章晓芳,陈林,徐宝文,聂长海.测试用例集约简问题研究及其进展.计算机科学与探索,2008,2(3):235−247.

[30] 屈波,聂长海,徐宝文.回归测试中测试用例优先级技术研究综述.计算机科学与探索,2009,3(3):225−233.

张智轶(1987－),男,山东潍坊人,博士生,CCF 学生会员,主要研究领域为软件测试.
E-mail: xianlingzibiying@gmail.com

徐宝文(1962－),男,博士,教授,博士生导师,CCF 高级会员,主要研究领域为软件工程.
E-mail: bwxu@nju.edu.cn

陈振宇(1978－),男,博士,副教授,CCF 会员,主要研究领域为软件分析与测试.
E-mail: zychen@software.nju.edu.cn

杨瑞(1976－),男,博士生,CCF 学生会员,主要研究领域为软件测试.
E-mail: ruizi2000@gmail.com