敏捷开发环境中的回归测试优化技术

王晓琳[1),2)] 曾红卫[1)] 林玮玮[1)]

[1)](上海大学计算机工程与科学学院 上海 200444)

[2)](上海市计算机软件评测重点实验室 上海 201112)

摘 要 版本频繁交付、功能不断新增或修改、测试用例不断增多是敏捷开发环境的特点.回归测试是软件测试的一个重要组成部分,它在敏捷开发环境中更应基于环境特点进行设计.但是,传统的回归测试优化技术(测试用例优先排序或回归测试选择等)各有其优缺点,且没有考虑敏捷开发环境对测试效率的影响.测试用例优先排序技术利用设计规则对所有测试用例进行排序,以提高错误检测率,但测试集基数大,花费时间长.回归测试选择技术选择部分测试用例执行,减少了测试执行时间,但存在不安全因素.为了解决这个问题,本文提出新的敏捷开发环境中的回归测试优化技术.首先,将回归测试拆分成两个过程,提出在这两个过程中的测试方法:敏捷测试用例优先排序和敏捷回归测试选择.敏捷测试用例优先排序方法基于历史排序的思想,将需求、错误反馈及历史信息三者结合,形成一个具有记忆的优先排序技术.敏捷回归测试选择方法结合错误信息和需求关联信息进行设计,选择以往版本中发现错误的测试用例及与新增测试有交互的测试用例作为测试子集,既检验曾经出错的功能是否正确,又检验新增功能加入是否影响已交付功能的稳定.其次,将敏捷排序和敏捷选择方法结合,设计回归测试优化模型,提出优化算法.为测试子集中的每一个测试用例设置一个失效标签以动态调整子集规模.最后,通过在不同规模的实验对象上进行实验,分析优化算法的有效性.实验结果表明,与现有的测试排序和测试选择及其混合方法相比,敏捷开发环境中的回归测试优化技术既可达到高错误检测率又可减少待执行测试用例数量同时保证较高的安全性.从统计分析 t 检验结果看,我们的方法在纠错速率上优于其它 5 种测试排序方法,因为 t 值均大于 0 且 p 值均小于 0.05;从纠错速率、效率成本百分比、运行时间缩减率及揭露错误百分比这 4 个方面的综合指数看,本文的方法最佳.

关键词 回归测试;测试用例优先排序;回归测试选择;敏捷开发;软件测试;软件工程

中图法分类号 TP311 **DOI 号** 10.11897/SP.J.1016.2019.02323

Techniques for Regression Testing in Agile Development Environment

WANG Xiao-Lin[1),2)] ZENG Hong-Wei[1)] LIN Wei-Wei[1)]

[1)](*School of Computer Engineering and Science, Shanghai University, Shanghai* 200444)

[2)](*Shanghai Key Laboratory of Computer Software Evaluating & Testing, Shanghai* 201112)

Abstract The frequent delivery of new versions, the frequent addition or modification of functions, and the increasing number of test cases are the characteristics of agile development environment. Regression testing, as an important part of software testing, should be designed based on these characteristics in agile development environment. However, most of traditional regression testing optimization techniques (test case prioritization or regression test selection, etc.) have their own advantages and disadvantages. And, they do not take into account the impact of environment on test efficiency. Test case prioritization sorts all of test cases by using specific rules to improve the rate of fault detection, but it takes a long time to perform because its test set has a large cardinality. Regression test selection selects part of available test cases which

收稿日期:2018-04-11;在线出版日期:2019-02-14.本课题得到国家自然科学基金(61572306)资助.**王晓琳**,博士研究生,主要研究领域为测试用例优先排序、回归测试和软件测试.E-mail: wangxiaolin@shu.edu.cn.**曾红卫**,博士,研究员,中国计算机学会(CCF)会员,主要研究领域为形式化方法、形式化验证和软件测试.**林玮玮**,博士研究生,主要研究领域为形式化方法和软件测试.

seems to be more effective in execution to reduce the test execution time, but it may be unsafe because some test cases that can detect faults may be missed. To address this problem, this paper proposes new regression testing techniques for agile development environment. Firstly, regression testing is split into two processes, and two approaches are put forward respectively: agile test case prioritization and agile regression test selection. Agile test case prioritization follows the idea of history-based test case prioritization and combines requirement, fault feedback, and test history to form a novel test case prioritization technique with memory. Agile regression test selection, designed by combining fault information and requirement correlation information, selects the test cases that detected faults in previous test versions and the test cases interacting with the new tests to form a test subset. It can not only verify whether the functions that have been wrong in the previous testing are correct, but also check whether the addition of new functions affects the stability of delivered functions. Secondly, these two methods are combined to design a regression testing optimization model and an agile regression testing optimization algorithm. The failure-tag is assigned to each test case of the test subset to dynamically adjust the number of test cases in the subset. Finally, the effectiveness of the agile regression testing optimization algorithm is analyzed by experiments on different-scale experimental objects. The representatives of test case prioritization techniques, regression test selection techniques, and their hybrid techniques are compared with our agile regression testing optimization algorithm. Experimental results show that the proposed agile regression testing optimization technique can achieve the higher fault detection rate in a shorter time with higher safety compared with existing test prioritization, test selection and hybrid methods. According to the statistical analysis results of t test, our agile regression testing optimization method is superior to the other five test case prioritization methods in terms of fault detection rate, because t values are all greater than 0 and p values are all less than 0.05. In addition, in terms of the composite index of fault detection rate, percentage of efficiency-cost, reduction ratio of running time, and percentage of detecting faults, our agile regression testing optimization method is superior to the others.

Keywords regression testing; test case prioritization; regression test selection; agile development; software testing; software engineering

1 引 言

敏捷开发是一种快速迭代的开发模式，是一个可持续集成开发过程. 在敏捷开发环境中，一次回归测试会话是指从上一版本发布开始，直至下一版本测试完成并发布的一个周期. 从软件需求功能的角度可将一次回归测试会话分割为旧版本已发布功能的测试和新版本新增功能的测试. 由于新功能的添加，新版本的测试会产生新的测试用例. 测试者需要利用全部新测试用例去测试软件新功能，以保证新功能的正确性、可靠性和稳定性. 本文将这类测试命名为“新测试”.

旧版本功能的测试是指对已交付的功能进行维护测试，旨在保证软件的稳定，不受新增功能的影响. 由于每个新版本都会产生新的测试用例，当版本更迭增长时，测试用例集规模逐渐庞大，测试者需要在有限时间内高效地利用这些测试用例，以期达到最大的测试效率. 这里称这类测试为“旧测试”.

在敏捷开发环境中，测试者既要保证新功能正确，又要保证已交付功能遇到新功能集成时可正常运行，所需开销提升. 管理者希望减少测试运行时间，降低测试成本，但要保证高质量测试效率，于是测试达到一个瓶颈. 有调研分析[1]表明，成熟的组织管理、合理的测试理念以及测试基建的高投资等会从策略上影响回归测试的效果；测试研究方法会在操作上影响回归测试的结果. 早期有学者提出一些优化方法，在不考虑新功能加入的情况下，对测试用例进行优先排序或者选择，以提高测试效率. 测试用例优先排序[2](Test Case Prioritization, TCP)是根

据某些预设规则将待执行的测试用例重新排序，旨在最快地揭露出更多错误. 回归测试选择[3](Regression Test Selection，RTS)是从测试用例集中挑选一部分作为子集去执行，确保软件修改没有引入新的错误. 这些排序或选择方法对于瀑布式开发模型的一次性交付软件具有较好的测试效率. 但对于频繁交付新版本的敏捷开发环境来说，传统 TCP 或 RTS 技术没有考虑处理新测试用例的排序和选择及新旧测试用例叠加的情况. 另外，Petschenik[4] 曾讨论过这样一个现象，要想有效地组织一个测试过程，所需资源必须达到 15%以上. Jussi 等人[5] 也发现，资源的可用性达到 60%～70%时，说明使用了测试用例优先排序技术. 也就是说，排序技术可提高资源的利用率，尤其在敏捷开发中时间和资源有限的情况下，测试用例优先排序技术是必要的.

本文主要贡献是：(1) 针对敏捷开发环境的“新测试”，提出敏捷测试用例优先排序方法. 该方法将需求、错误反馈及历史优先级结合，形成一种新型的基于历史的优先排序技术；(2) 针对“旧测试”，提出敏捷回归测试选择方法. 该方法选择已发布版本的失效测试用例和交互测试用例作为子集，并为每一个测试用例赋予一个失效标签以动态调整子集规模；(3) 将敏捷测试用例优先排序和敏捷回归测试选择结合，设计适用于敏捷开发环境的回归测试优化模型及优化算法；(4) 谷歌共享的测试结果数据集(Google Shared Dataset of Test Suite Results，GSDTSR)[6] 和一个实际工业项目 ChipTest 进行评估分析.

本文第 2 节将介绍回归测试优化领域的背景知识及相关研究；第 3 节重点描述我们提出的敏捷开发环境中的回归测试的优化技术；第 4 节设计具体实验并分析实验结果；第 5 节是结论与展望.

2　背景及相关研究

回归测试[3]. 已知 P 为一程序，P' 为 P 的修改版本，T 为 P 的一组测试用例集. 回归测试过程如下：

(1) 选择 $T' \subseteq T$，并用 T' 去测试修改版本 P' 以确保 P' 在 T' 上的正确性；

(2) 为 P' 的新功能创建新的测试用例集 T''，并用 T'' 测试 P' 以确保 P' 在 T'' 上的正确性；

(3) 从 T、T'、T'' 中为 P' 创建新的测试用例集 T''' 用于执行测试.

回归测试重复利用旧的测试用例集测试新版本 P'，旨在验证 P' 的正确性. 如果新功能迭代添加，测试集会随回归测试会话数的增加逐渐庞大，重复利用全部测试用例开销巨大. 据统计[7]，回归测试占整个测试过程的 80%，在软件维护过程中占到 50%[8]. 因此，一些学者研究如何对测试用例进行优先排序[9-12] 或选择部分测试用例[13-16] 进行测试，以提高测试效率. 近年来，软件演化中如何对测试用例集进行有效扩增[17-19] 也成为回归测试的研究热点.

2.1　测试用例优先排序

测试用例优先排序问题[2] 的定义为：

给定一组测试集 T、T 的全排列 PT 以及应用于每一个排列的从 PT 到实数的一个函数 f，f：$PT\text{-}R$. 找出 $T' \in PT$，使得 $(\forall T'')(T'' \in PT)(T'' \neq T')[f(T') \geq f(T'')]$.

TCP 寻找满足测试准则的最优排序，以提高错误检测率. 唯一不足的地方是它需利用全部的测试用例，测试集基数大，花费时间较长.

Rothermel 和 Elbaum 团队[20-21] 最早研究细粒度代码级 TCP 技术. 这类方法效果显著，但开销较大. 后来他们又分析程序实体粒度对排序的影响[9]，发现粒度越大，开销越小，但排序效果会降低. Just 等人[22] 从变异分析出发，研究程序代码的冗余变异对测试效率的影响. Zhang 等人[23] 结合测试约减与排序技术实现提高符号执行的效率. 此外，基于重构方法是利用一组重构故障模型重新排序测试序列[24]. 代码级 TCP 技术属白盒测试，执行时必须拿到源代码，开销昂贵. 黑盒测试不需要源代码及结构化信息，操作相对简单. 有数据[25] 显示，白盒测试和黑盒测试之间的错误揭露平均百分比值差异仅为 2%～4%，且仅在执行 56%～60%的测试用例时白盒测试效果较好. 近年来，研究者逐渐将研究重点转移到黑盒测试. Sihan 等人[26] 对比五种 TCP 算法，指出在重叠率适中的情况下，Additional 贪心算法和 2-优化对贪心算法都达到较好的效果. Srikanth 等人[11] 提出需求优先排序测试算法(Prioritization Of Requirements for Test，PORT)，以数值驱动的方式排序测试用例. Kim[10] 利用回归测试产生的历史数据对测试用例进行排序，但对有新增测试的情况效果不明显，因为赋予新增测试用例的执行概率与从未发现错误的测试用例执行概率相等. Garg 和 Datta[27] 在 Web 应用方面针对已修改功能提出新的 TCP 技术. Arafeen 和 Do[28] 提出一种基于需求聚类的 TCP 方法. 有学者从信息检索的角度出发提出一种名叫 REPIR 的回归测试有效排序方法[29]. Wang 等人[30] 从资源感知出发，通过修改适应度函

数,提出基于多目标优化的面向工业环境的 TCP 方法. Tyagi 等人[31]对于多目标的 TCP 问题采用多目标粒子群(MOPSO)算法进行测试选择及排序. 随后,他们又提出考虑故障检测率、故障检测百分比和风险识别能力三方面因素的多目标 TCP 方法[32]. 张娜等人[33]引入关注需求覆盖率、基于需求测试用例重要度和测试用例失效率这三个因素共同确定一个基于需求的 TCP 问题. Ray 等人[34]假定测试用例的优先级就是其覆盖组件的优先级的总和,使用遗传算法解决多目标 TCP 问题. Lu 等人[35]基于现有传统 TCP 技术进行对比实验,得出:新增测试将显著影响测试效率,使得传统技术效果甚微,仅源代码的更改对测试效果没什么影响;所有对比技术中,Additional 排序技术和基于搜索的排序技术效果最好. Spieker 等人[36]利用强化学习[37]设计适应于可执行集成环境的一种自动 TCP 技术.

2.2 回归测试选择

给定程序 P 及 P 的测试集 T,回归测试选择[3,38]为 P'(P 的修改版本)选择子集 T'($T'\subseteq T$),便其测试 P'.

RTS 技术重点选择能测试修改部分及与受修改影响的软件部分相关的测试用例. RTS 减少了测试执行时间,但该技术存在不安全因素,当选择策略漏选了可发现错误的测试用例时,其测试效果将下降. 这个缺点 TCP 技术可以弥补.

RTS 的研究相对于 TCP 较早. Rothermel 和 Harrold 在文献[3]中给出 RTS 的形式化定义,并详细分析几组 RTS 技术. Yoo 等人[13]对回归测试优化中的约减、选择、排序进行概况总结,详细地描述各个方法的区别及优缺点. Daniel 等人[39]在一个实际工业系统上进行实验,分析基于覆盖的回归测试选择、约减和排序的特点. 王克朝等人[40]提出面向有效错误定位的测试用例优选方法. 吴川等人[41]研究基于分支覆盖的回归测试路径选择问题. 基于文件依赖的测试选择技术[42]是选择新版本中修改文件所依赖的测试用例作为子集,虽比传统 RTS 选择更多测试用例,增加了执行时间,但降低了整体端到端时间,且提高了安全性. RTS 已存在多种方法适用于不同测试场景,但经验发现,目前难以存在一种 RTS 技术普遍优于其他技术[14,43].

2.3 排序与选择结合

一些学者将排序和选择结合提出新的混合方法. Bharti 和 Shweta[44]提出一种基于蚁群优化的测试用例选择与排序的方法. Ruchika 和 Arvinder 等人[45]提出一种回归测试选择与优化技术. 根据修改程序的覆盖率对测试集 T 进行优先排序,选择覆盖所有修改程序的测试用例作为子集 T',执行 T'. 郑锦勤和牟永敏[46]根据函数调用路径与测试用例的映射关系选出回归测试子集,根据绝对贡献度、缺陷检测能力及缺陷影响度对测试子集进行优先级排序,最后对优先排序结果再次选择,确定最小的测试集.

3 敏捷开发环境中回归测试优化技术

敏捷开发的一个回归测试过程既包括对新增功能的"新测试",又包括对已交付功能的"旧测试". 图 1 为敏捷开发环境回归优化的整体框架图.

图 1 中,首先,将回归测试依据前言的划分思想分为"新测试"和"旧测试",见上方矩形框所示. 其次,中间的 3 个圆角矩形为本节技术的设计重点. 对"新测试",结合敏捷开发中迭代更新多、历史数据多的特点,将需求信息、错误信息和历史信息三者结合设计敏捷测试用例优先排序技术(Agile Test Case Prioritization,A-TCP);对"旧测试",秉承精简执行测试集的思想,结合错误信息和需求关联信息设计敏捷回归测试选择技术(Agile Regression Test Selection,A-RTS);将 A-TCP 和 A-RTS 结合,设计回归测试优化模型(Regression Test Optimization Model,RTOM),最终输出待执行的测试序列,如图 1 下方的输出文档. 最后,执行有序的测试用例,本次优化完成.

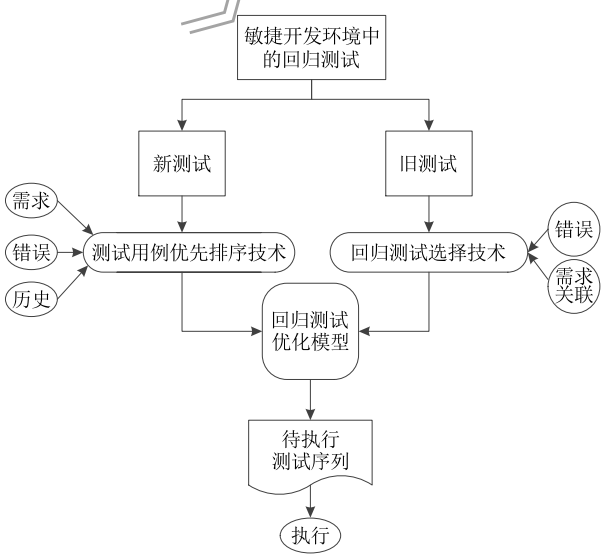


图 1 敏捷开发环境中的回归优化整体框架图

3.1 "新测试"中的测试用例优先排序

由于敏捷开发过程中,每一次版本开发都是细

分为不同阶段迭代式开发，所以测试也为逐步集成测试. 首先引入“阶段”[47]的概念.

定义 1.　测试阶段 ST. 假设软件版本交付之前分为 $n(n\geq 1)$ 个开发阶段，那么测试阶段相应的定义为 $ST_i(1\leq i\leq n)$. 每个测试阶段会产生测试用例集 $TS_i=\{t_1,t_2,\cdots,t_m\}$，$TS_i$ 对应 ST_i，为第 i 个测试阶段新增的测试用例集，$t_i(i=1,\cdots,m)$ 为测试用例. 所有测试阶段的 TS 组成全体测试用例集组 $T=\{TS_1,TS_2,\cdots,TS_n\}$，表示本次版本所有新增测试用例集的集合.

给出 A-TCP 规则. 对于有 n 个测试阶段的“新测试”，每进入一个新的测试阶段，其相应的当前阶段测试用例集 TS_i 整体放于待执行测试序列 T' 的首位. 也就是说，先对测试阶段排序，新阶段的测试用例集赋予最高优先级；然后再进行阶段内测试用例排序.

借助图 2 解释 A-TCP. 假设“新测试”过程共有 5 个测试阶段，前 4 个阶段已经测试完成，当前为第 5 阶段. 右上角的虚框矩形为第 5 测试阶段新产生的测试用例集. 根据 A-TCP 规则应将其优先级设为最高级 1 并置于待执行的测试序列首位，如图 2 斜线框矩形. 其余前几阶段产生的测试用例集赋次优先级 2，并排列在 1 级之后. 对于 1 级测试用例集，采用基于需求的测试用例初始化方法(*Initialization-SortTestCase*)对新产生的测试用例进行排序；对于 2 级测试用例集，采用基于历史的动态 TCP 方法(*HistorySortTestCase*)进行排序. 最后输出排好序的新执行序列 T'.

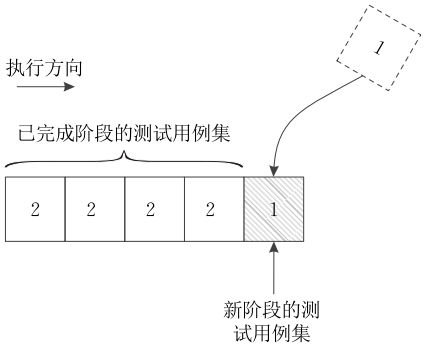


图 2　“新测试”中的测试用例执行序列

3.1.1　基于需求的测试用例初始化

基于需求的测试用例初始化具体思想如下.

(1) 对每一个需求进行重要性计算，用 IV(*Importance Value*)代表需求的优先级.

需求 r 重要性 $IV(r)$ 的计算等式为：

$$IV(r)=\omega_1\times CP\_IV(r)+\omega_2\times DP\_IV(r) \quad (1)$$

$IV(r)$ 由两部分组成：一个是客户分配的需求重要性值 $CP\_IV(r)$；另一个开发者分配的需求重要性值 $DP\_IV(r)$. 两者均为 5 个等级，可采用线性赋值，如 $n(n\in\{1,2,3,4,5\})$，也可采用非线性赋值，如 $a^n(a\in Z,n\in\{1,2,3,4,5\})$. 最终，两者加权平均，得到一个需求的优先级. $\omega(\omega_1,\omega_2)$ 为权值，总和为 1，可根据实际情况进行分配. 如未特别指定哪个因素的特定权值时，默认均等权值，即 $\omega_1=\omega_2=0.5$. 本文采取均等权值计算.

(2) 构造需求-测试用例关系矩阵. 假设 $R=\{r_1,r_2,\cdots,r_m\}$ 为需求集，$Tc=\{t_1,t_2,\cdots,t_n\}$ 为测试用例集. 对于每一个需求，至少有一个测试用例覆盖它；对于每一个测试用例，则有 1 或多个需求与之对应. 算法 1 为计算 R-T 关系矩阵. 矩阵的列为需求，行为测试用例. 第 i 行第 j 列表示 t_i 所覆盖的 r_j 的优先级，用 $IV(t_i,r_j)$ 表示，其值由 r_j 的需求优先级 $IV(r_j)$ 赋予. 当 $IV(t_i,r_j)$ 大于 0，表示 t_i 覆盖 r_j，即 t_i 与 r_j 有关联关系；为 0 表示 t_i 不覆盖 r_j，即 t_i 与 r_j 没有关系. R-T 矩阵构造完成后，可获得两点内容：一是测试用例覆盖需求的个数；二是测试用例覆盖需求的重要性.

算法 1.　计算关系矩阵 R-T.

```
1. FOR EACH t_i∈Tc, r_j∈R
2.    IF (t_i覆盖 r_j) THEN
3.       IV(t_i,r_j)=IV(r_j);
4.    ELSE IV(t_i, r_j)=0;
5.    ENDIF
6. ENDFOR
```

(3) 计算测试用例的优先级. 已知 t 覆盖的需求个数为 m，令 $RP(t)$ 为测试用例 t 的优先级：

$$RP(t)=\sum_{k=1}^{m}IV(t,r_k) \quad (2)$$

一个测试用例的优先级即为其覆盖的所有需求的优先级之和. 当每个需求的优先级均等，即不考虑需求的优先等级时，$IV(t,r_k)=1$，等式(2)简化为基于需求覆盖数的测试用例优先级赋值等式：

$$RP(t)=\begin{cases}\sum_{k=1}^{m}(1), & m>1;\\ 1, & m=1\end{cases} \quad (3)$$

其中 m 为 t 覆盖的需求个数.

根据 $RP(t)$ 大小排序即可得到初始状态下测试用例的优先排序.

3.1.2　基于历史的动态测试用例优先排序

基于历史的动态 TCP[48] 具体思想如下.

(1)首先根据每次回归测试的错误揭露情况动态调整需求优先级.

对于每一个测试用例每次揭露的错误,可根据R-T关系映射得到R-T-F三元关系,如图3.图3中,有5个需求(r_1-r_5),4个测试用例(t_1-t_4),6个错误(f_1-f_6).如果测试用例t与需求r是一对一关系,这时需求r与错误f的间接关系可以明确得到,如图3的t_2、t_3和t_4.如果t和r是一对多关系,这时需要深入分析错误并在三元关系中明确指明r和f的关系,如图3中f_1和f_2到r_1的虚线箭头.

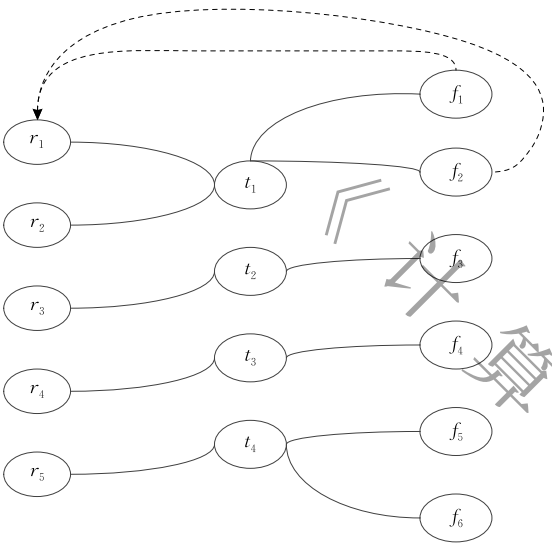


图3 需求、测试用例和错误的三元关系图

得到R-T-F三元关系后,根据相邻两次测试揭露错误个数的差值$Dvalue_fault$对每个需求进行优先级调整,等式如下.

$$\begin{cases} Dvalue\_fault = fault_{k-1}(t_i, r_j) - fault_{k-2}(t_i, r_j) \\ IV_k(t_i, r_j) = IV_{k-1}(t_i, r_j) + Dvalue\_fault \end{cases} \tag{4}$$

其中$fault_{k-1}(t_i, r_j)$为前一次回归测试中测试用例t_i覆盖的r_j对应的揭露错误的个数.$IV_k(t_i, r_j)$即为r_j在当前第k次回归测试中的优先级值.

例如,假设有5个需求,4个测试用例.第一轮回归测试的错误揭露结果如图3所示,第一轮需求初始化后的IV值如表1所示,调整后的需求IV值在表1中展示.

表1 调整需求优先级

	IV_1	$Dvalue\_fault$	IV_2
r_1	3	2	**5**
r_2	3	0	**3**
r_3	2	1	**3**
r_4	1	1	**2**
r_5	4	2	**6**

(2)根据调整后的需求优先级,利用等式(2)重新计算每个测试用例的RP.

根据表1的结果,可以计算出第二轮回归测试中每个测试用例的RP值,如表2.

表2 第2轮回归测试中测试用例的基于需求优先级以及基于历史优先概率($\sigma=0.8$)

	RP_1	RP_2	P_2
t_1	6	8	0.4291
t_2	2	3	0.1571
t_3	1	2	0.0996
t_4	4	6	0.3134

(3)利用RP和以往历史数据对所有测试用例重新排序.每个测试用例的优先执行概率由其当前基于需求的优先级和历史数据的长期影响计算获得,计算等式如下.

$$\begin{cases} P_1 = NRP_1 \\ P_k = \sigma NRP_k + (1-\sigma)P_{k-1}, \quad 0 \le \sigma \le 1,\ k \ge 1 \end{cases} \tag{5}$$

NRP为RP的标准化形式,每个t的NRP用其占所有t的RP值之和比率进行计算,即$NRP(t)=\frac{RP(t)}{\sum_{i=1}^{n} RP(t_i)}$,所得结果用小数表示,小数点后保留4位数字.$P_k$代表的是第$k$次回归测试中测试用例执行的优先概率;$\sigma$为历史数据的权重.当$\sigma$设定大于0.5时,$P_k$主要取决于最近两次回归测试会话的历史数据$NRP_k$;反之,$P_k$则主要取决于整个测试生命周期中除最近一次回归历史数据外其它全部回归测试会话的历史数据P_{k-1}.如果测试用例t的P_k值越高,被优先执行的概率越大.表2展示了4个测试用例经过计算后得到的第二轮回归测试的优先概率.

3.1.3 A-TCP

A-TCP算法的伪代码见算法2.假设当前测试阶段产生的新的测试用例集为TS_n.该算法输入为当前阶段的全体测试用例集组T,输出为新的测试序列T'.首先,搜索T中的所有TS,进行阶段排序.其次,对不同的阶段采用不同的方法进行阶段内排序.阶段排序和阶段内排序混合的好处是:优先级层次分明,减少冗余处理测试用例,提高测试效率.

算法2. A-TCP.

//假设当前测试阶段为ST_n

输入:T //输入当前阶段下已产生的全体测试用例集组.

输出:T' //输出新的测试序列.

1. $T'=\varnothing$;

2. FOR $j=n$ DOWNTO 1 DO

3. IF(TS_j是ST_n的测试用例集) THEN
4. $TS_j.Priority=1$; //将TS_j的优先级赋予最高级1
5. $TS'_j=InitializationSortTestCase(TS_j)$;
6. $T'=TS'_j+T'$; //将有序序列TS'_j置于T'队首位置
7. ELSE $TS_j.Priority=2$; //其余TS赋予次优先级2
8. $TS'_j=HistorySortTestCase(TS_j)$;
9. $T'=T'+TS'_j$; //将TS'_j置于输出队列的队尾位置
10. ENDIF;
11. ENDFOR;

沿用表1和表2的例子继续进行解释.假设当前测试阶段为第二轮回归测试ST_2,表2中的测试用例t_1—t_4为测试阶段ST_1的测试用例集TS_1.首先为ST_2产生新的测试用例集$TS_2=\{t_5,t_6,t_7,t_8,t_9,t_{10}\}$.根据初始化方法得到每个测试用例的初始化优先级为$RP(t_5)=2$,$RP(t_6)=7$,$RP(t_7)=4$,$RP(t_8)=3$,$RP(t_9)=1$,$RP(t_{10})=5$.其次,根据A-TCP将$TS_2$赋予优先级1,按照测试用例初始化方法进行排序,排序后置于输出队列的队首;将TS_1赋予优先级2,按照基于历史的方法进行排序,排序后置于输出队列队尾.最后,得到最终的T'为$t_6-t_{10}-t_7-t_8-t_5-t_9-t_1-t_4-t_2-t_3$.

3.2 “旧测试”中的回归测试选择

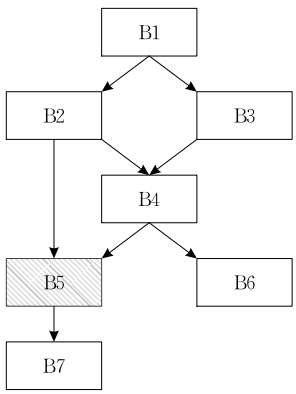
“旧测试”主要涉及两部分内容:一是检验曾经出错的功能是否还会发生错误,保证功能的正确性;二是检验新增功能加入是否会影响已交付功能出错,保证功能的稳定性.“重测所有”策略(Retest-All)[3,38]无疑是最能保证已交付功能正确性和稳定性的方法.但对于敏捷开发环境下逐渐庞大的测试集,执行所有测试用例显然效率低下.本节提出一种敏捷回归测试选择技术A-RTS.与A-TCP对应,这里给出“版本”的概念.

定义2. 测试版本VT.假设一个系统分为$k(k\geq 1)$个开发版本,那么测试版本相应的定义为$VT_i(1\leq i\leq k)$.一个测试版本中包含若干测试阶段.一个VT_i的测试用例即为其所有测试阶段全体测试用例集组T_i中的测试用例.所有测试版本的T形成的容器称为测试池$TP=\{T_1,T_2,\cdots,T_k\}$.

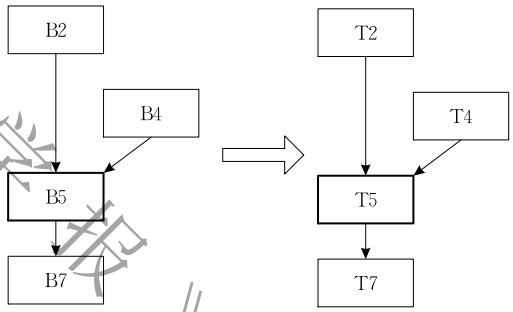
定义3. 失效测试用例ft.测试过程中发现错误导致软件失效的测试用例称为失效测试用例ft.

定义4. 交互测试用例nt.新版本中新增功能加入有时会涉及与旧版本功能有交互,测试这些存在旧版本已发布功能中被交互的功能的测试用例称为交互测试用例nt.交互测试用例可根据需求功能关联关系图[49-50]得到.

用图4解释如何获得交互测试用例.假设每个需求的粒度为模块,每个需求均有一个测试用例测试.图中有7个需求:B1～B7,则有7个测试用例(T1～T7)与之对应.图4(a)为需求功能关联关系图,展示了7个需求之间的关联关系.其中,矩形框为需求,有向边为两个需求的直接对应关系.假设B5这个功能为新增功能,抽取与B5有直接对应关系的关联子图如图4(b)左侧图示.根据3.1.1节算法1中得到的R-T关系矩阵得到测试B5的新增测试用例T5的交互测试用例为T2、T4和T7.



(a) 需求功能关联关系图



(b) 获得交互测试用例

图4　关于交互测试用例的例子

定义5. “旧测试”子集LT.在“旧测试”过程中通过测试选择技术得到的子集称为“旧测试”子集LT.它包含失效测试用例ft和交互测试用例nt.

A-RTS的伪代码见算法3.该算法的输入是测试池中现有的全部测试用例,且以T、TS为单位分组分层划分.输出为子集LT.搜索TP中所有的T,如果不是当前版本的T,遍历其内部所有阶段的测试用例集TS,将失效测试用例ft及交互测试用例nt加入到LT当中.重复上述过程直至搜索结束.

算法3. A-RTS.

//假设当前测试版本为VT_k
输入:$TP=\{T_1,T_2,\cdots,T_k\}$ //输入测试池中所有测试用例
输出:LT //输出通过选择技术筛选出来的“旧测试”子集

1. $LT=\varnothing$; //初始 LT 为空
2. FOR EACH $T_i \in TP$ //遍历测试池中每个 T
3. IF (T_i 不是 VT_k 的全体测试用例集组) THEN
//T_i 为已交付旧版本的 T
4. FOR EACH $t_m \in TS_j$, $TS_j \in T_i$
//遍历每个 T 中每个阶段的 TS 中的测试用例
5. IF (t_m 是 ft 或 nt) THEN
6. $LT=LT+\{t_m\}$;
7. ENDIF;
8. ENDFOR;
9. ENDIF;
10. ENDFOR;

借助图 5 解释 A-RTS 算法. 假设测试池 TP 中有 k 个全体测试用例集组 T, 如图 5 的圆角矩形. 当前版本的全体测试用例集组为 T_k, 且初始时"旧测试"子集 LT 为空集. 搜索所有的 T_i, 如果 T_i 不是 T_k, 则: 1) 找出 T_i 中曾经发现错误的测试用例, 作为失效测试用例 ft 加入 LT 中; 2) 找出 T_i 中与当前版本有功能交互的测试用例, 作为交互测试用例 nt 加入 LT 中. 搜索 T_1, T_1 不是 T_k, 遍历发现 T_1 中第 1 测试阶段的测试用例集 TS_1 中存在发现错误的测试用例(黑色五角星), 将其作为 ft_1 加入左下角的圆形 LT 中. 继续搜索, 当搜索到 T_3 时, 发现 T_3 中存在与 T_k 中某个阶段的测试用例有功能上交互的测试用例(黑色六角形), 将其作为 nt 加入 LT 中. 最终筛选得到"旧测试"子集 LT. 图 5 中, LT 中测试用例数量明显减少, 因此 A-RTS 算法可以有效减少测试时间, 节省开销.

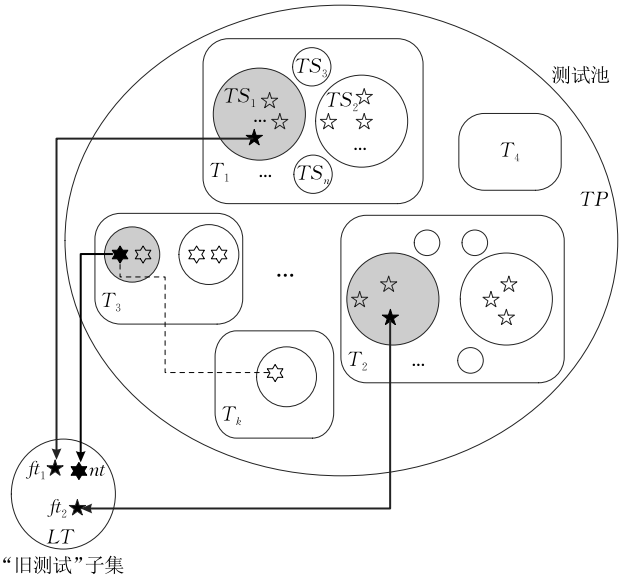


图 5 "旧测试"中待执行的测试用例集

3.3 回归测试优化

本节将前文提出的 A-TCP 和 A-RTS 有效融合, 建立回归测试优化模型 RTOM, 并设计适用于敏捷开发环境的回归测试优化算法(Agile Regression Test Optimization, A-RTO).

3.3.1 优化模型 RTOM

由于敏捷开发环境中的回归测试不断交付新功能, 对于"旧测试", 其 LT 也逐渐庞大, 为此设置一个失效标签以动态调整 LT 中的测试用例个数.

定义 6. 失效标签 Tag. 每个加入 LT 的测试用例 lt 被赋予一个失效标签 Tag, 初始值为 0. 每次测试执行后: 如果该 lt 没有失效, 即没有发现错误, Tag 值加 1; 如果中途某一次测试发现错误, Tag 值清 0. 当 $Tag>\theta$(θ 为阈值)时, 则将该 lt 从 LT 中剔除.

如图 6 所示, RTOM 的输入是当前版本当前测试阶段产生的新测试用例集 TS_i, 目的是通过测试用例优先排序和测试用例选择得到一个待执行的测试序列 T_i'', 最终执行 T_i''.

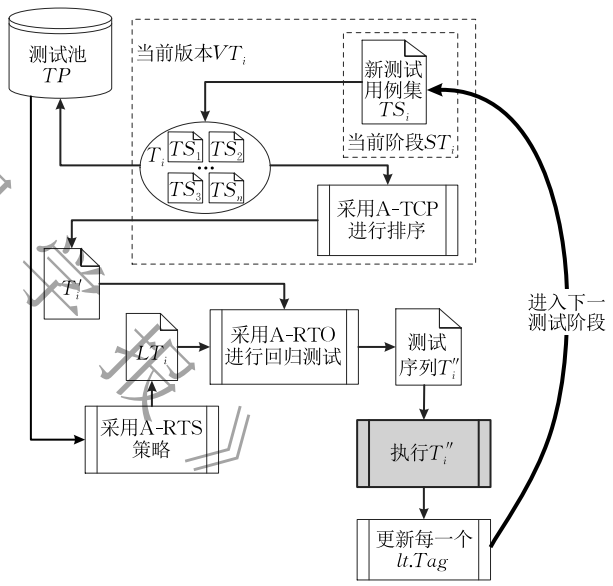


图 6 敏捷开发环境中的回归测试优化模型 RTOM

第 1 步. 产生当前版本当前阶段的新测试用例集 TS_i, 并加入到 T_i 中, 同时更新 T_i 在 TP 中的数据.

第 2 步. 采用 A-TCP 对当前版本的 T_i 进行排序, 得到当前版本所有测试用例的一个优化序列 T_i'.

第 3 步. 采用 A-RTS 搜索 TP 并将 ft 和 nt 筛选出来, 得到 LT_i.

第 4 步. 采用回归测试优化算法(A-RTO)将 LT_i 与 T_i' 融合成一个新的测试序列 T_i''.

第 5 步. 执行 T_i''.

第 6 步. 根据 T_i'' 的执行情况, 更新每一个 lt 在 LT_i 中的失效标签 $lt.Tag$ 值, 测试进入下一阶段.

3.3.2　优化算法 A-RTO

A-RTO 的过程见算法 4. 该算法从初始版本开始循环迭代：首先，为当前版本创建新的 T，并进行优先排序，得到“新测试”的测试优先排序队列 T'；其次，对上一版本的 T_{i-1} 进行测试选择，将 ft 或 nt 加入“旧测试”的待执行子集 LT，并为新加入的成员初始化 Tag；再次，对上一版本的测试执行进行失效标签修改与判断，从而动态调整 LT 的规模；最后，将 T' 置于输出 T'' 的队首位置，LT 按默认顺序置于 T'' 的队尾位置，形成一条优化 T'' 并执行.

算法 4.　A-RTO 过程.

输入：$TP=\{T_1\}$

//从初始版本时输入 TP，此时池中只有 T_1

1. $i=1$, $q=0$, $LT=\varnothing$;

//初始化全局变量 i, q, LT

2. WHILE(VT_i 不是最终版本 VT_k) DO
3. $T''=\varnothing$, $T'=\varnothing$;

//执行序列 T''、“新测试”排序 T' 置空

4. Create new T_i for VT_i; //为当前 VT_i 产生新的 T_i
5. $TP=TP+\{T_i\}$; //将新测试用例集组 T_i 加入 TP
6. $T'=A\_TCP(T_i)$;

//使用 A-TCP 得到新的优先排序 T'

7. FOR EACH $t_m \in TS_j$, $TS_j \in T_{i-1}$
8. IF (t_m 是 ft 或 nt) THEN
9. $lt_q=t_m$;　　//将 t_m 赋值给 lt_q
10. $LT=LT+\{lt_q\}$; //将 lt_q 加入“旧测试”子集中
11. $lt_q.Tag=0$;　　//将 lt_q 的失效标签初始化为 0
12. $q++$;
13. ENDIF;
14. ENDFOR;
15. FOR EACH $lt_j \in LT$

//遍历每一个 LT 中的测试用例 lt_j

16. IF (lt_j 在上一版本 VT_{i-1} 执行时未揭露错误) THEN
17. $lt_j.Tag++$; //该测试用例的失效标签值加 1
18. ELSE $lt_j.Tag=0$; //反之，其 Tag 值清零
19. ENDIF;
20. IF ($lt_j.Tag>\theta$) THEN

//如果 lt_j 的 Tag 值大于阈值 θ

21. $LT=LT-\{lt_j\}$; //将该测试用例从 LT 中剔除
22. ENDIF;
23. ENDFOR;
24. $T''=T'+T''$;　　//将 T' 置于 T'' 队首
25. $T''=T''+LT$;　　//将 LT 按默认顺序置于 T'' 队尾
26. Run T'';　　//执行测试序列 T''
27. Submit VT_i, i++, and enter VT_{i+1};

//VT_i 交付，进入 VT_{i+1}

28. ENDWHILE;

4　实验验证

为了验证本文提出的 A-RTO 的有效性，本节进行一系列实验，在不同规模的测试集中度量不同技术的差异. 由于 A-RTO 是排序和选择的混合技术，因此我们在实验过程中让 A-RTO 首先与现有的排序技术进行比较，其次与现有的选择技术进行比较，最后与现有的混合方法进行比较.

问题 1. 与测试排序技术相比，本文的 A-RTO 方法是否能更早地发现更多的错误？

问题 2. 与测试选择技术相比，本文的 A-RTO 方法是否具有高效率低成本？

问题 3. 与其他混合方法相比，本文的 A-RTO 方法是否效率更高？

4.1　实验对象

本文选取 2 个不同规模的系统(1 个开源程序，1 个实际工业系统)作为实验被测系统，使用它们原有测试用例及实际发生错误进行研究，不再额外设计测试用例及植入错误. 本实验中，采用需求功能关联关系图中的直接关联关系作为交互测试用例的计算内容，间接关联可在今后的研究中加入.

(1) GSDTSR

谷歌共享的测试集结果数据集 GSDTSR 是应用于谷歌产品样本的包含大于 350 万测试用例执行结果的数据集[35]，具体内容可在网址 https://code.google.com/archive/p/google-shared-dataset-of-test-suite-results/ 上下载(也可在 GitHub 上下载，https://github.com/nimrodbusany/Radiant4GoogleReps). 它包括测试名 ID、更改的需求、测试阶段、输出状态以及执行时间. 每个测试用例只有两个状态，通过或失败，即一个测试用例至多揭露一个错误. GSDTSR 的部分原始信息如表 3. 第 1 列为测试用例集 ID. 第 2 列为更改的需求数. 本文赋予该列新的意义，即更改需求的阶段. 例如表 3 中，第 0 个阶段改变的需求为 5 个(Change Request 为 0 的个数为 5)，第 13 个阶段改变的需求为 2 个. 第 6 列为测试用例集的分类，其分类依据是根据它们测试的类别规模. “Small”测试集测试类别是类，“Medium”测试集的目标是类和其邻类，“Large”测试集测试的类别最接近实际领域场景. 最后 1 列为输出状态，如果失败，即说明该测试用例发现错误. 其余几列可根据首行的名称知晓其意义.

表 3 GSDTSR 部分信息

Test Suite	Change Request	Stage	Launch Time	Execution Time/ms	Size	Shared number	Run number	Language	Status
324	0	post	1:00:04:58	52705	LARGE	6	0	JAVA	FAILED
226	0	post	30:23:59:18	64644	LARGE	0	0	WEB	FAILED
3501	13	post	1:01:15:02	58035	MEDIUM	0	8	SH	FAILED
1941	13	post	1:01:05:57	105524	LARGE	0	7	SH	FAILED
1431	36	post	1:01:08:27	269	MEDIUM	0	0	PY	FAILED
4080	0	post	30:23:59:00	140	MEDIUM	0	0	CC	PASSED
334	0	post	1:00:05:46	143	SMALL	0	0	GO	PASSED
4531	0	post	30:23:59:28	189	MEDIUM	0	0	SH	PASSED
4603	87	pres	3:02:40:53	661	SMALL	0	0	CC	PASSED

(2) ChipTest

ChipTest 是一个实际工业项目,用于测试硬件芯片.该系统包含大约 14 万行 JAVA 代码.这里取其版本 9～14 作为实验对象,每个版本包含 1～3 个阶段,且有新增功能加入.它的测试用例粒度较粗,每个测试用例对应一个较大的需求,所以每个测试用例可能包含几十甚至上百个测试脚本.详细信息如表 4.

表 4 ChipTest 各版本详细信息

版本号	新增功能	测试用例	错误
9.1	5	9	7
9.2	6	7	14
9.3	3	10	14
10.1	9	9	8
10.2	6	6	12
11.1	5	5	14
11.2	4	4	5
11.3	1	5	12
12.1	8	8	8
13.1	6	7	19
13.2	7	7	11
14.1	12	12	16
14.2	11	15	20

4.2 实验方法

本文的测试方法均采用黑盒测试,即不涉及代码结构化信息等内容.

(1) 与测试排序比较

首先,选取现有的测试用例排序方法作为基准方法,它们分别是:随机排序、Total 型基于覆盖的排序、Additional 型基于覆盖的排序、基于需求优先级的排序[11] 和基于历史的排序[10],记为 R、TC、AC、PORT 和 H.

其次,将基准方法与本文的 A-RTO 进行比较.其中,TC 和 AC 的覆盖粒度为需求覆盖.H 采用的历史信息为错误揭露信息[10],并与 A-RTO 方法中的 A-TCP 采用相同的历史平滑系数.本文假设"最近会话的历史数据比全局历史数据对当前会话的测试效果更有效".在该假设前提下,实验中对历史平滑系数设定为 $\sigma=0.8$.

将 6 种方法分别应用于 2 个实验对象系统上,不同的方法会生成不同的测试用例排序序列,评估这些序列的平均错误检测率.由于随机排序等方法存在不稳定性影响,因此每种方法独立运行 20 次并取其平均值作为实验结果.

A-TCP 方法中,测试用例初始化时的需求重要性使用线性赋值(等级分为 1,2,3,4,5).对于工业系统 ChipTest,可拿到其需求优先级;对于谷歌共享数据集 GSDTSR,根据数据集中的资料模拟给定需求优先级.

(2) 与测试选择比较

首先,选取现有的回归测试选择方法作为基准方法,它们分别是:重测所有、随机选择、基于修改的选择和基于文件依赖的选择[42],记为 RA、RS、MS 和 FDS.

其次,将基准方法与 A-RTO 进行比较.其中,MS 采用基本的基于程序变更及需求变更修改,不考虑其他变更相关影响.FDS 的依赖信息采集如下:对于 GSDTSR,将与某测试用例的测试编号相同的其他测试用例作为该测试用例的依赖测试用例;对于 ChipTest,根据测试用例名称依赖关系进行收集.

将 5 种方法分别应用于 2 个实验对象系统上,随着版本的逐步增加,得到不同的测试用例子集,评估这些子集的效率.其中,重测所有方法无疑是耗费时间最长的选择方法,但其安全性最高.可将该方法看作是边界方法与其他方法进行评估.

本实验中,失效标签阈值均取 $\theta=10$.

(3) 与其它混合方法比较

首先,选取现有的排序选择混合方法作为基准方法,它们分别是:基于修改程序的测试用例选择排序[45] 和面向函数调用路径的测试用例选择排序[46],记为 MC-ST 和 FCP-ST.其次,将基准方法与 A-RTO 进行比较.

将 3 种混合方法分别应用于 2 个实验对象系统上，并使用排序和选择评估指标进行评估.

4.3　评价指标

(1) 检测一个测试序列的有效性通常使用 $APFD$[9] 作为评价指标，评测该测试序列的平均错误检测百分比. $APFD$ 的计算等式如下所示：

$$APFD=1-\frac{TF_1+TF_2+\cdots+TF_m}{nm}+\frac{1}{2n} \quad (6)$$

其中 n 表示测试用例的个数，m 表示错误的个数，TF_m 代表首个检测出第 m 个错误的测试用例在该测试序列中所处的次序. $APFD$ 未考虑测试序列执行的时间及错误的严重程度.

(2) 针对测试选择技术，从揭露错误个数和测试集大小两方面评测效率，定义为“效率成本百分比”.

定义 7.　效率成本百分比 ECP. 假设测试用例集中的测试用例总数为 TN，揭露错误个数为 FN，该测试集的效率成本百分比为：

$$ECP=\frac{FN}{TN} \quad (7)$$

测试选择技术的使用使测试序列的长度变短，相应执行时间也会减少. 定义一个辅助评测指标-运行时间缩减率 (Reduction ratio of Running Time, RRT)，计算测试序列运行时间缩减的百分比. 假设每个测试用例执行时间为 1，RRT 的计算等式为：

$$RRT=\frac{N-n}{N} \quad (8)$$

其中，n 为测试序列的执行时间(即测试用例个数)，N 表示测试池中全体测试用例的执行时间.

定义一个揭露错误百分比(The percentage of Detecting Faults, DF)，监督测试选择技术的不安全性. DF 的计算等式如式(9).

$$DF=\frac{FN}{F} \quad (9)$$

其中 FN 为所执行的测试集揭露出的错误数，F 为测试池中所有测试用例所揭露出的错误总数. DF 的值越高，说明该选择技术的安全性越高.

4.4　实验结果分析

4.4.1　与测试排序方法比较

从表 5 可以看出，A-RTO 的 $APFD$ 值几乎均高于其它 5 种方法，说明 A-RTO 较其它几种 TCP 方法在纠错速率上具有更好的效果. 尤其在大数据量的情况下(GSDTSR)，$APFD$ 高达 95%以上.

在大部分版本中，考虑反馈信息的 H 和 A-RTO 比不考虑反馈信息 R\TC\AC\PORT 的 $APFD$ 值要高，说明考虑反馈信息的技术对于多版本的系统更适用. H 不考虑新增测试的优先排序，所以对于回归测试前几次版本的效果会更高，例如 GSDTSR 中前 3 个版本和 ChipTest 中 9.3 版本. 随着版本增加，测试用例集逐渐庞大，加之新增测试对测试结果的影响，A-RTO 技术均比 H 技术在纠错率上更胜一筹.

表 5　6 种技术的 *APFD* 值

系统	版本	*APFD*/%					
		R	TC	AC	PORT	H	A-RTO
GSDTSR (节选)	3	66.18	17.73	17.73	17.73	**99.81**	99.34
	10	39.42	21.83	21.83	21.83	85.21	**99.66**
	13	10.75	42.01	42.01	42.01	70.75	**99.28**
	14	45.66	20.15	20.15	20.15	64.84	**99.88**
	36	3.25	3.54	3.54	3.54	61.59	**99.89**
	44	28.77	0.42	0.42	0.42	50.03	**95.00**
	68	0.63	3.63	3.63	3.63	47.38	**99.90**
	88	0.12	1.86	1.86	1.86	46.20	**99.82**
	93	0.29	2.41	2.41	2.41	99.74	**99.86**
	95	10.64	4.29	4.29	4.29	99.72	**99.93**
ChipTest	9.2	41.9	20.95	20.95	36.67	40.48	**74.76**
	9.3	24.44	29.44	29.44	42.78	**72.78**	46.67
	10.1	21.3	17.13	17.13	9.72	29.35	**68.13**
	10.2	17.93	6.31	6.31	6.57	25.43	**84.94**
	11.1	13.46	6.98	6.98	14.27	24.13	**90.62**
	11.2	4.52	3.57	3.57	19.29	19.23	**92.76**
	11.3	2.91	27.13	27.13	49.22	52.25	**62.50**
	12.1	20.83	4.17	4.17	14.71	23.75	**85.14**
	13.1	24.7	7.71	7.71	19.34	26.82	**90.76**
	13.2	42.21	12.77	12.77	24.89	33.65	**87.17**
	14.1	33.33	33.33	33.33	35.67	23.34	**88.89**
	14.2	37.06	3.83	3.83	8.06	23.78	**84.86**

为验证所得结果的显著性差异，采用 t 检验对表 5 中所有结果进行评估. t 检验之前已使用单样本 K-S 检验验证六组样本数据均服从正态分布.

从表 6 的 t 检验结果数据上看，A-RTO 较其它 5 种方法均有显著差异. 例如在跟随机排序对比时，t 值大于 0($t=13.28$)且 p 值小于 0.05($p=0.000$)，说明 A-RTO 在纠错率上明显优于随机排序.

表 6　A-RTO 与其它 5 种排序方法的 t 检验

	t 标准值	p 值
A-RTO vs. R	13.28	0.000
A-RTO vs. TC	16.72	0.000
A-RTO vs. AC	16.72	0.000
A-RTO vs. PORT	12.86	0.000
A-RTO vs. H	6.47	0.000

图 7 展示了实验结果的箱形图. 其中，横轴表示 6 种优化方法，纵轴表示 $APFD$ 值. 每个矩形盒表示每种方法 $APFD$ 值的最大值、上/下四分位数、中位数和最小值. 用 $x(Median, Q1, Q3)$ 表示某个方法 $APFD$ 值的中值、上四分位数和下四分位数，6 种方法的结果分别为：$R(21.07, 4.20, 37.65)$，

TC(7.35, 3.62, 21.17), *AC*(7.35, 3.62, 21.17), *PORT*(16.22, 4.13, 27.59), *H*(46.79, 25.78, 71.26)和 *A-RTO*(91.76, 84.92, 99.7). 从图 7 和 x 数据可以清晰地看出,A-RTO 的所有表现均优于其它排序方法.

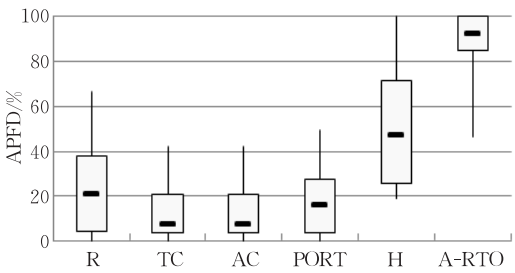


图 7 6 种方法的箱形图

4.4.2 与测试选择方法比较

表 7 列出了 A-RTO 与其它 4 种测试选择方法在两个实验对象系统的连续若干版本中进行实验得到的结果. 其中,GSDTSR 中只节选了前 100 个版本中出现错误的版本进行展示. 表中,黑体加粗的数据为最优结果. 由表 7 可以看出,大部分版本中 A-RTO 的 ECP 较高;运行时间最少的是 RS;在所有版本中,RA、FDS 和 A-RTO 这 3 种方法的 DF 均为 100%,说明它们的安全性最高.

对于大规模的 GSDTSR,其特点是版本迭代频繁,每个版本都有一定数量的新增功能,但错误数和测试用例数相差很大,从几千倍开始逐步递增. RA 的 ECP 和 RRT 均较低,不适用. RS 有较高的 RRT(从 36 版本开始 RRT 均达到 95%以上),但 ECP 基本最低(有超过三分之一的版本 ECP 为 0),安全性不稳定,且由于其随机性,使其测试效果无法预知,因此综合评估 RS 不适用. MS 和 A-RTO 的 RRT 与 RS 的差距仅有零点几的百分比,但 A-RTO 的 ECP 和 DF 比 MS 更胜一筹. FDS 的 DF 可达 100%,但它的 RRT 是除 RA 以外最低的,ECP 也比 A-RTO 低,总体性能不高. 所以,对于大数据量且错误数相对较少的系统,MS 和 A-RTO 均可选择,A-RTO 综合性能稍高. 对于中等规模的 ChipTest,其特点是版本迭代多,且每个版本新增功能较多,错误量高于测试用例数量. RS 在 ECP 和 DF 上均最低,不适用. RA 在 ECP 和 RRT 上均比 MS、FDS 和 A-RTO 差,DF 与 FDS、A-RTO 持平. 因此,对于这种中等规模的系统,MS、FDS 和 A-RTO 均可选择.

表 7 不同方法对 GSDTSR 和 ChipTest 系统测试的效率成本百分比、运行时间缩减率及揭露错误百分比 /%

系统	版本	ECP					RRT					DF				
		RA	RS	MS	FDS	A-RTO	RA	RS	MS	FDS	A-RTO	RA	RS	MS	FDS	A-RTO
GSDTSR	0	0.15	0.01	0.15	0.15	**0.15**	0	**0.33**	0	0	0	100	20.000	100.00	**100**	**100**
	3	0.24	0.07	0	1.01	**1.33**	0	**82.96**	82.15	76.29	82.04	100	5.000	0	**100**	**100**
	10	0.15	0.04	**0.69**	0.54	0.68	0	**79.97**	78.1	71.94	78.02	100	7.140	100.00	**100**	**100**
	13	0.61	**1.60**	1.44	1.32	1.44	0	**61.88**	57.7	53.66	57.60	100	25.640	100.00	**100**	**100**
	14	0.05	0.03	0.20	0.19	**0.25**	0	**89.86**	79.78	73.77	79.73	100	1.786	80.00	**100**	**100**
	36	0.01	0	**0.22**	0.09	0.21	0	**96.93**	96.46	90.99	96.36	100	0	100.00	**100**	**100**
	44	0.05	0.06	6.78	0.73	**8.22**	0	99.24	**99.55**	93.81	99.45	100	100.000	66.67	**100**	**100**
	68	0.01	0.01	0.19	0.08	**0.19**	0	**96.49**	96.37	90.97	96.26	100	5.000	100.00	**100**	**100**
	88	0.01	0	**0.36**	0.10	0.34	0	**98.65**	98.14	92.95	98.01	100	0	100.00	**100**	**100**
	93	0.01	0	**0**	0.09	**0.26**	0	**98.03**	97.58	92.52	97.45	100	0	0	**100**	**100**
	95	0.01	0	0	0.07	**0.14**	0	**97.49**	95.71	90.87	95.58	100	0	0	**100**	**100**
ChipTest	9	88.89	3.50	88.89	88.89	**88.89**	0	**44**	0	0	0	100	43.75	100	**100**	**100**
	10	60.61	6.00	90.91	62.50	**90.91**	0	**39**	33	3.00	33	100	90.00	100	**100**	**100**
	11	65.12	5.00	105.26	70.00	**107.69**	0	53	**56**	6.98	40	100	71.43	71.43	**100**	**100**
	12	15.69	4.00	**44.44**	16.67	23.53	0	61	**65**	5.88	33	100	100.00	100.00	**100**	**100**
	13	47.62	8.93	**175.00**	50.00	71.43	0	68	**75**	4.76	33	100	83.33	93.33	**100**	**100**
	14	40.00	5.00	**94.44**	41.86	54.55	0	**67**	60	4.44	27	100	83.33	94.44	**100**	**100**

4.4.3 与排序选择混合方法比较

MC-ST、FCP-ST 和 A-RTO 这 3 种混合方法在不同系统不同版本中实验时 4 个指标(APFD、ECP、RRT 和 DF)结果的表格与表 8 类似,这里不再展示. 图 8 和图 9 是根据 3 种混合方法的实验结果得到不同指标的柱状图. 其中,纯色图例表示 MC-ST,斜线图例表示 FCP-ST,波浪图例表示 A-RTO. 横轴按照两个不同的系统分布,分别是 GSDTSR 和 ChipTest. 纵轴为评测指标,值域范围为 0~100 百分比. 图 8(a)和(b)表示 APFD 和 ECP 的分布. 图 9(a)和(b)表示 RRT 和 DF 的分布.

从图 8(a)中可以看出,无论在哪种规模的系统中,A-RTO 的 APFD 最高,说明作为排序方法来看,A-RTO 所形成的有序序列能尽早地揭露出更多的错误. 从图 8(b)和图 9 可知,虽然在 ECP 和 RRT 上,A-RTO 均不如其它两种混合方法,但在 DF 上

却高于其它两种方法. 深入分析可知，MC-ST 和 FCP-ST 比 A-RTO 减少的测试用例数更多，得到的测试子集规模更小，但它们在漏掉揭露错误的测试用例的概率更高，因而安全性会降低. 从选择方法的角度看，3 种方法需要根据不同测试环境进行取舍.

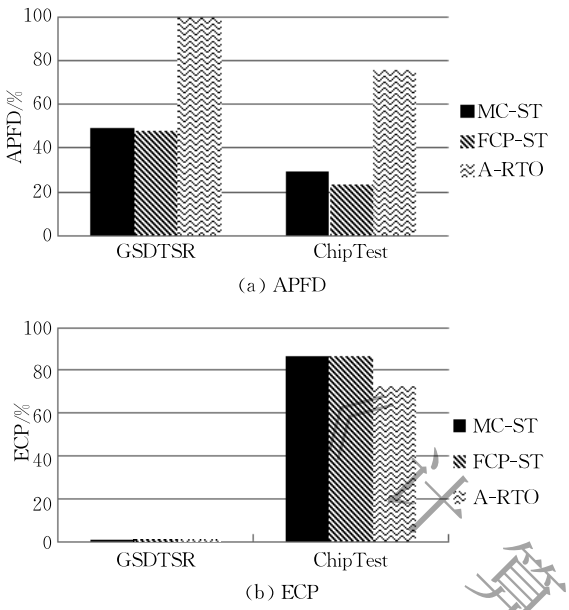


图 8　不同混合方法在不同系统下的 APFD 和 ECP 比较

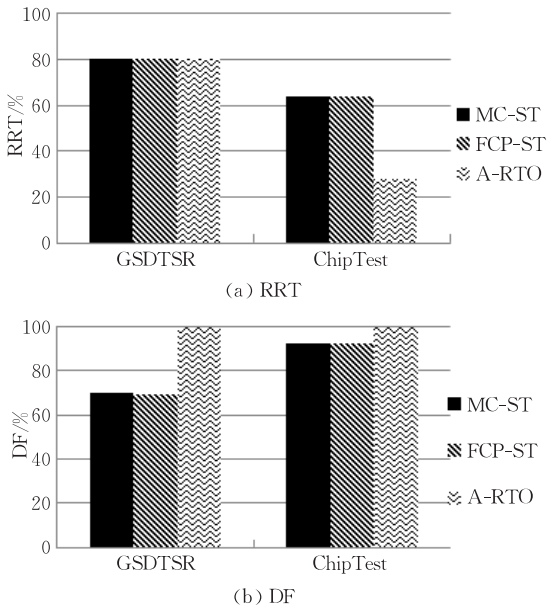


图 9　不同混合方法在不同系统下的 RRT 和 DF 比较

具体地，在大规模的 GSDTSR 中，A-RTO 的优势 APFD 和 DF 明显，但缺点 ECP 和 RRT 却与其它两者基本持平. 在 GSDTSR 中无论哪种方法 ECP 的数值最高只有 1.44%. 深入分析发现，这是因为 ECP 的计算是错误数比测试用例总数，由于 GSDTSR 的测试集庞大，错误数相比之下很少，因此使得无论使用哪种方法，ECP 差异并不明显. 同理，3 种方法的 RRT 数值出现基本持平也是由于 RRT 计算等式的分母为测试用例总数，在基数非常大时，分子的变化差异不明显会使得整个分数的变化不明显. 综合考虑，A-RTO 在 4 种指标对比中，两优两平，所以在 GSDTSR 中是最优选择. 对于中等规模的 ChipTest，由于测试用例数和错误数相对差距不大，且测试用例基数比 GSDTSR 小很多，4 种指标的差异均可明显显示出来. 如果时间相对充足但安全性要求较高，A-RTO 为最优选择，且高 APFD 使得即使时间有限无法执行全部测试序列，A-RTO 也可揭露出更多的错误.

将每个方法在每个指标下的所有实验结果数据进行平均值计算，结果作为平均指数. 把每个指标平均指数加权求和，得到该方法的综合指数 Z：

$$Z=\varepsilon_1 APFD+\varepsilon_2 ECP+\varepsilon_3 RRT+\varepsilon_4 DF \quad (10)$$

其中，$\varepsilon(\varepsilon_1-\varepsilon_4)$ 为加权系数，总和为 1，可根据实际情况进行分配. 当没有特别指定哪个指标需要特定权值时，默认均等系数，即 $\varepsilon_1=\varepsilon_2=\varepsilon_3=\varepsilon_4=0.25$. 本文采取均等权值系数计算.

图 10 表示 3 种混合方法的综合指数图. 从图中可看出，A-RTO 方法的综合指数最高，即综合各方面因素评价，A-RTO 为最佳方法.

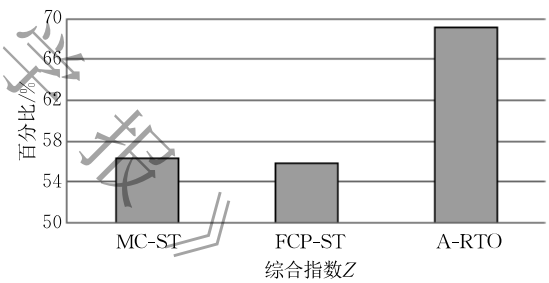


图 10　不同混合方法的综合指数比较

4.4.4　实验结果有效性分析

本文采用 APFD、ECP、RRT 和 DF 作为评价指标. 虽然 ECP、RRT 及 DF 均为新定义的指标，对评估结果可能会产生差异，但其内容是参考其它文献（如文献[39]）中相关等式进行定义，具有相对有效性. 另外，对于 APFD 指标，也有其它指标可替换，如 APFDc[51]. 由于本文假定错误无严重性区分且测试用例执行开销均等，所以选取 APFD 更为合适.

本文选取的实验对象，一个是谷歌共享的测试结果集，属于大数据量范畴，一个是实际工业界系统. 虽然所选对象仍有局限性，所得到的实验结果不能代表所有类型系统，但开源实验对象对实验公正性有较大帮助，工业界系统让所研究内容能在实际应用中实验，也有很强的说明性.

由于本文中一些特定的假设,如每个测试用例执行时间均等、每个错误严重等级程度均等,以及对一些参数给出经验设定,如计算需求重要性 IV 采取均等权值系数、计算优先概率 P_k 的历史平滑系数 σ 取 0.8、失效标签阈值 θ 取值为 10、综合指数采用均等权值系数等,这些设定均是根据经验结合具体实验对象系统给定,可能会对实验结果有效程度的范围和粒度有些许影响,但这些设定至少让所提出方法证实有效.如果进一步对这些设定进行细分研究,将会得到更加精准的结果.

5 结论与展望

本文针对敏捷开发环境的回归测试进行研究,将测试优先排序和测试选择结合,提出回归测试优化模型 RTOM 及其算法 A-RTO.其中,A-TCP 将需求、错误反馈及历史信息三者有效结合,提高了测试序列的纠错率;A-RTS 选择失效测试用例和交互测试用例作为子集,并引入一个失效标签动态调整子集规模,减少了测试序列个数.实验结果证明 A-RTO 优于其他方法,包括测试排序方法(R、TC、AC、PORT 和 H),测试选择方法(RA、RS、MS 和 FDS),排序选择混合方法(MC-ST 和 FCP-ST).尤其对于大数据量的系统对象,A-RTO 具有显著的效果.

在今后的工作中,一方面,进一步优化 RTOM 模型与算法,以适用更多场景;另一方面,考虑在大数据下如何优化测试,扩展测试研究的新方向.

参 考 文 献

[1] Parsons D, Susnjak T, Lange M. Influences on regression testing strategies in agile software development environments. Software Quality Journal, 2014, 22(4): 717-739

[2] Elbaum S, Malishevsky AG, Rothermel G. Prioritizing test cases for regression testing//Proceedings of the ACM SIGSOFT 2000 International Symposium on Software Testing and Analysis. Portland, USA, 2000: 102-112

[3] Rothermel G, Harrold M J. Analyzing regression test selection techniques. IEEE Transactions on Software Engineering, 1996, 22(8): 529-551

[4] Petschenik N H. Practical priorities in system testing. IEEE Software, 1985, 2(5): 18-23

[5] Kasurinen J, Taipale O, Smolander K. Test case selection and prioritization: Risk-based or design-based?//Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement. Bolzano-Bozen, Italy, 2010: 10

[6] Elbaum S, Rothermel G, Penix J. Techniques for improving regression testing in continuous integration development environments//Proceedings of the ACM SIGSOFT Symposium on the Foundations of Software Engineering. Hong Kong, China, 2014: 235-245

[7] Harrold M J. Reduce, reuse, recycle, recover: Techniques for improved regression testing//Proceedings of the IEEE International Conference on Software Maintenance. Edmonton, Canada, 2009: 5

[8] Chen Xiang, Chen Ji-Hong, Ju Xiao-Lin, et al. Survey of test case prioritization techniques for regression testing. Journal of Software, 2013, 24(8): 1695-1712(in Chinese)
(陈翔, 陈继红, 鞠小林等. 回归测试中的测试用例优先排序技术述评. 软件学报, 2013, 24(8): 1695-1712)

[9] Elbaum S, Malishevsky A G, Rothermel G. Test case prioritization: A family of empirical studies. IEEE Transactions on Software Engineering, 2002, 28(2): 159-182

[10] Kim J M, Porter A. A history-based test prioritization technique for regression testing in resource constrained environments//Proceedings of the 24th International Conference on Software Engineering. Orlando, USA, 2002: 119-129

[11] Srikanth H, Banerjee S. Improving test efficiency through system test prioritization. Journal of Systems and Software, 2012, 85(5): 1176-1187

[12] Huang Y C, Peng K L, Huang C Y. A history-based cost-cognizant test case prioritization technique in regression testing. Journal of Systems and Software, 2012, 85(3): 626-637

[13] Yoo S, Harman M. Regression testing minimization, selection and prioritization: A survey. Software Testing Verification and Reliability, 2012, 22(2): 67-120

[14] Engstrom E, Runeson P, Skoglund M. A systematic review on regression test selection techniques. Information and Software Technology, 2010, 52(1): 14-30

[15] Yoo S, Harman M. Pareto efficient multi-objective test case selection//Proceedings of the 2007 ACM International Symposium on Software Testing and Analysis. London, UK, 2007: 140-150

[16] Briand LC, Labiche Y, He S. Automating regression test selection based on UML designs. Information and Software Technology, 2009, 51(1): 16-30

[17] Santelices R, Chittimalli P K, Apiwattanapong T, et al. Test-suite augmentation for evolving software//Proceedings of the 23rd IEEE/ACM International Conference on Automated Software Engineering. L'Aquila, Italy, 2008: 218-227

[18] Xu Z, Kim Y, Kim M, et al. Directed test suite augmentation: techniques and tradeoffs//Proceedings of the ACM SIGSOFT Symposium on the Foundations of Software Engineering. Santa Fe, USA, 2010: 257-266

[19] Xu Z, Kim Y, Kim M, et al. Directed test suite augmentation: An empirical investigation. Software Testing Verification and Reliability, 2015, 25(2): 77-114

[20] Rothermel G, Untch R H, Chu C, et al. Prioritizing test cases for regression testing. IEEE Transactions on Software Engineering, 2001, 27(10): 929-948

[21] Rothermel G, Untch R H, Chu C, et al. Test case prioritization: An empirical study//Proceedings of the 1999 IEEE International Conference on Software Maintenance. Oxford, UK, 1999: 179-188

[22] Just R, Schweiggert F. Higher accuracy and lower run time: Efficient mutation analysis using non-redundant mutation operators. Software Testing Verification and Reliability, 2015, 25(5-7): 490-507

[23] Zhang C, Groce A, Alipour M A. Using test case reduction and prioritization to improve symbolic execution//Proceedings of the 23rd International Symposium on Software Testing and Analysis. San Jose, USA, 2014: 160-170

[24] Alves E L G, Machado P D L, Massoni T, et al. Prioritizing test cases for early detection of refactoring faults. Software Testing Verification and Reliability, 2016, 26(5): 402-426

[25] Henard C, Papadakis M, Harman M, et al. Comparing white-box and black-box test prioritization//Proceedings of the 2016 IEEE/ACM 38th IEEE International Conference on Software Engineering. Austin, USA, 2016: 523-534

[26] Li S, Bian N, Chen Z, You D, et al. A simulation study on some search algorithms for regression test case prioritization//Proceedings of the International Conference on Quality Software. Zhangjiajie, China, 2010: 72-81

[27] Garg D, Datta A, French T. A two-level prioritization approach for regression testing of web applications//Proceedings of the 19th Asia-Pacific Software Engineering Conference. Hong Kong, China, 2012: 150-153

[28] Arafeen M J, Do H. Test case prioritization using requirements-based clustering//Proceedings of the IEEE 6th International Conference on Software Testing, Verification and Validation. Luxembourg, 2013: 312-321

[29] Saha R K, Zhang L, Khurshid S, et al. An information retrieval approach for regression test prioritization based on program changes//Proceedings of the 37th IEEE/ACM International Conference on Software Engineering. Florence, Italy, 2015: 268-279

[30] Wang S, Ali S, Yue T, et al. Enhancing test case prioritization in an industrial setting with resource awareness and multi-objective search//Proceedings of the 2016 IEEE/ACM 38th International Conference on Software Engineering. Austin, USA, 2016: 182-191

[31] Tyagi M, Malhotra S. Test case prioritization using multi objective particle swarm optimizer//Proceedings of the 2014 International Conference on Signal Propagation and Computer Technology. Ajmer, India, 2014: 390-395

[32] Tyagi M, Malhotra S. An approach for test case prioritization based on three factors. International Journal of Information Technology and Computer Science, 2015, 7(4): 79-86

[33] Zhang Na, Yao Lan, Bao Xiao-An, et al. Multi-objective optimization based on-line adjustment strategy of test case prioritization. Journal of Software, 2015, 26(10): 2451-2464(in Chinese)
(张娜，姚澜，包晓安等. 多目标优化的测试用例优先级在线调整策略. 软件学报，2015，26(10)：2451-2464)

[34] Ray M, Mohapatra D P. Multi-objective test prioritization via a genetic algorithm. Innovations in Systems and Software Engineering, 2014, 10(4): 261-270

[35] Lu Y, Lou Y, Cheng S, et al. How does regression test prioritization perform in real-world software evolution?//Proceedings of the 2016 IEEE/ACM 38th International Conference on Software Engineering. Austin, USA, 2016: 535-546

[36] Spieker H, Gotlieb A, Marijan D, et al. Reinforcement learning for automatic test case prioritization and selection in continuous integration//Proceedings of the 26th ACM SIGSOFT International Symposium on Software Testing and Analysis. Santa Barbara, USA, 2017: 12-22

[37] Liu Quan, Zhai Jian-Wei, Zhang Zong-Zhang, et al. A survey on deep reinforcement learning. Chinese Journal of Computers, 2018, 41(1): 1-27(in Chinese)
(刘全，翟建伟，章宗长等. 深度强化学习综述. 计算机学报，2018，41(1)：1-27)

[38] Graves T L, Harrold M J, Kim J M, et al. An empirical study of regression test selection techniques. ACM Transactions on Software Engineering and Methodology, 2001, 10(2): 184-208

[39] Nardo D D, Alshahwan N, Briand L, et al. Coverage-based regression test case selection, minimization and prioritization: A case study on an industrial system. Software Testing Verification and Reliability, 2015, 25(4): 371-396

[40] Wang Ke-Chao, Wang Tian-Tian, Su Xiao-Hong, et al. Test case selection for improving the effectiveness of software fault localization. Journal of Computer Research and Development, 2014, 51(4): 865-873(in Chinese)
(王克朝，王甜甜，苏小红等. 面向有效错误定位的测试用例优选方法. 计算机研究与发展，2014，51(4)：865-873)

[41] Wu Chuan, Gong Dun-Wei, Yao Xiang-Juan. Selection of paths for regression testing based on branch coverage. Journal of Software, 2016, 27(4): 839-854(in Chinese)
(吴川，巩敦卫，姚香娟. 基于分支覆盖的回归测试路径选择. 软件学报，2016，27(4)：839-854)

[42] Gligoric M, Eloussi L, Marinov D. Practical regression test selection with dynamic file dependencies//Proceedings of the 2015 International Symposium on Software Testing and Analysis. Baltimore, USA, 2015: 211-222

[43] Zhang Zhi-Yi, Chen Zhen-Yu, Xu Bao-Wen, et al. Research progress on test case evolution. Journal of Software, 2013, 24(4): 663-674(in Chinese)
(张智轶，陈振宇，徐宝文等. 测试用例演化研究发展. 软件学报，2013，24(4)：663-674)

[44] Suri B, Singhal S. Implementing ant colony optimization for test case selection and prioritization. International Journal on Computer Science and Engineering, 2011, 3(5): 1924-1932

[45] Malhotra R, Kaur A, Singh Y. A regression test selection and prioritization technique. Journal of Information Processing Systems, 2010, 6(2): 235-252

[46] Zheng Jin-Qin, Mu Yong-Min. Regression test case selection and prioritization based on functions calling path. Application

Research of Computers, 2016, 33(7): 2063-2067(in Chinese)
(郑锦勤, 牟永敏. 基于函数调用路径的回归测试用例选择排序方法研究. 计算机应用研究, 2016, 33(7): 2063-2067)
[47] Wang Xiao-Lin, Shi You-Qun, Tang Cheng, et al. An improved RBAC authorization model based on time partition. Computer Science, 2013, 40(10): 155-158(in Chinese)
(王晓琳, 史有群, 唐成等. 基于时间分区的改进 RBAC 授权模型. 计算机科学, 2013, 40(10): 155-158)
[48] Wang X, Zeng H. History-based dynamic test case prioritization for requirement properties in regression testing//Proceedings of the International Workshop on Continuous Software Evolution and Delivery. Austin, USA, 2016: 41-47
[49] Sun Ying-Ying, Zhang Yi-Kun, Yang Kai-Feng, et al. Software test approach based on analyzing program association. Application Research of Computers. 2008, 25(12): 3650-3653(in Chinese)
(孙赢盈, 张毅坤, 杨凯峰等. 一种基于程序关联性分析的软件测试方法. 计算机应用研究, 2008, 25(12): 3650-3653)
[50] Zhang Zhi-Hua, Mu Yong-Min. Research of path coverage generation techniques based function call graph. Acta Electronica Sinica, 2010, 38(8): 1808-1811(in Chinese)
(张志华, 牟永敏. 基于函数调用的路径覆盖生成技术研究. 电子学报, 2010, 38(8): 1808-1811)
[51] Elbaum S, Malishevsky A, Rothermel G. Incorporating varying test costs and fault severities into test case prioritization// Proceedings of the 23rd International Conference on Software Engineering. Toronto, Canada, 2001: 329-338

WANG Xiao-Lin, Ph. D. candidate. Her current research interest is test case prioritization, regression testing, and software testing.

ZENG Hong-Wei, Ph. D., professor. His current research interests include formal method, formal verification, and software testing.

LIN Wei-Wei, Ph. D. candidate. Her current research interests include formal method and software testing.

Background

In an agile development environment, releases change quickly and the test suite becomes large, which requires both high-quality and low-cost testing. Regression testing in the agile development environment has its special characteristics. However, most of existing regression testing optimization techniques (TCP or RTS, etc.) do not consider the impact of test environment on test efficiency and effectiveness. TCP is to solve the problem of fault detection rate in regression testing. It sorts test cases according to certain rules, so that test cases can detect more faults as quickly as possible. But it has disadvantages that it needs to prioritize and execute all of the test cases of the test set, which is expensive. RTS is to solve the problem of test execution efficiency in regression testing. It selects part of available test cases from the test suite to perform them. But it has disadvantages that it may be unsafe, that is, it may miss out on those test cases that can detect faults. TCP can make up for the shortcomings of RTS, and vice versa. This paper, considering the combination of TCP and RTS, studies the regression testing optimization problem for a specific agile development environment and proposes a regression testing optimization model RTOM and its optimization algorithm A-RTO. It divides the testing process into two parts: 1) For the new version testing, this paper combines requirement, fault feedback, and test history to form an agile TCP technique (A-TCP for short) in order to address the fault detection rate issue; 2) For the old version testing, this paper proposes an agile RTS technique (A-RTS for short) which uses the test cases that detect faults in previous test versions and the test cases that interact with the new tests to form a test subset in order to address the test execution efficiency issue. And then, A-TCP and A-RTS are combined to design the algorithm A-RTO. It assigns a failure-tag to each test case in the test subset to dynamically adjust the number of test cases of the test subset. By comparing with different test case prioritization methods and regression test selection methods and their hybrid methods, A-RTO is better than the other methods in terms of the composite index of four aspects: fault detection rate, percentage of efficiency-cost, reduction ratio of running time, and percentage of detecting faults.

This research work is supported by National Natural Science Foundation of China (NSFC) under Grant No. 61572306.