软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2015,26(10):2451−2464 [doi: 10.13328/j.cnki.jos.004745]

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

多目标优化的测试用例优先级在线调整策略*

张 娜[1], 姚 澜[2], 包晓安[1], 董 萌[1], 桂 宁[1,3]

[1](浙江理工大学 信息学院,浙江 杭州 310018)

[2](中国电子科技集团 第五十二研究所,浙江 杭州 310013)

[3](Distrinet Laboratory, University of Leuven, Belgium)

通讯作者: 包晓安, E-mail: baoxiaoan@zstu.edu.cn

摘 要: 从需求的角度对测试用例的优先级进行排序,定义了一个多目标的测试用例优化排序问题,引入关注需求覆盖率、测试用例重要度和测试用例失效率这3个测试用例优先级影响因子,分别定义权重因子α,β,γ用于权衡3个因子.设计了关注需求覆盖率和测试用例失效率的在线估计方法及算法,在此基础上,设计了一种基于多目标优化的测试用例优先级在线调整策略,该策略可利用测试过程中收集到的反馈信息,对测试用例优先级进行在线调整,实现在尽早达到测试覆盖率标准的同时,尽早覆盖重要的和具有较高失效率的测试需求,从而解决尽早检测到更多的、严重等级较高的软件缺陷这一多目标测试用例优化问题.实验结果表明:与随机测试、传统的单目标优先级排序方法和确定性排序方法相比,所提出的策略能够在更短的时间内完成同等质量的软件测试,从而提高了测试效率.

关键词: 测试用例优先级;关注需求覆盖;测试用例重要度;测试用例失效率;测试用例在线调整策略

中图法分类号: TP311

中文引用格式: 张娜,姚澜,包晓安,董萌,桂宁.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. http://www.jos.org.cn/1000-9825/4745.htm

英文引用格式: Zhang N, Yao L, Bao XA, Dong M, Gui N. Multi-Objective optimization based on-line adjustment strategy of test case prioritization. Ruan Jian Xue Bao/Journal of Software, 2015,26(10):2451−2464 (in Chinese). http://www.jos.org.cn/1000-9825/4745.htm

Multi-Objective Optimization Based On-Line Adjustment Strategy of Test Case Prioritization

ZHANG Na[1], YAO Lan[2], BAO Xiao-An[1], DONG Meng[1], GUI Ning[1,3]

[1](School of Informatics and Electronics, Zhejiang Sci-Tech University, Hangzhou 310018, China)

[2](CETC 52, Hangzhou 310013, China)

[3](Distrinet Laboratory, University of Leuven, Belgium)

Abstract: In order to properly rank the priority of test cases from the requirement's perspective, this paper introduces three impact factors: Concerned-Requirements coverage, test case importance degree and test case failure rate. Meanwhile, three weight factors α, β and γ are introduced to balance the three impact factors. This paper designs on-line estimating methods and algorithms based on the concerned-requirements coverage and the test case failure rate. Based on those metrics, a multi-objective optimization based test case prioritization on-line adjustment strategy is developed. The strategy is able to adjust the priorities of test cases dynamically using the feedback information collected in the test process, and thus can meet the coverage criteria earlier and cover test cases of importance and high failure rate. This strategy can also resolve the multi-object test case prioritization problem by detecting more severe bugs earlier. Experimental results show that, compared with random test, the traditional single-object test and the test with deterministic test case prioritization, the presented strategy can complete the test with equal quality by shorter time, thus improves the testing efficiency.

* 基金项目: 国家自然科学基金(61502430, 61379036); 浙江省自然科学基金(LY12F02041, Y13F020175); 浙江省人才计划(2013R10015, 2013R1029); 浙江理工大学 521 人才培养计划

收稿时间: 2014-04-21; 修改时间: 2014-06-20, 2014-09-29; 定稿时间: 2014-10-20

Key words: test case prioritization; concerned-requirement coverage; test case importance degree; test case failure rate; test case on-line adjustment

随着软件行业的迅速发展,软件产品升级换代的频率越来越高,使得回归测试的成本也越来越高[1,2].在大规模软件的测试中,完全运行测试用例进行集中的测试常常需要几天甚至几个月才能完成.在这种情况下,测试人员就希望对测试用例的优先级按某种标准进行排序,使得优先级高的用例能够尽早地被执行,从而提高测试效率.测试用例优先级技术正是为了解决这一问题而提出来的[2],该技术指出:不同测试用例对于测试目标的完成有着不同的贡献程度,为了能够更快地达成测试目标,有必要依据测试的历史信息将不同的测试用例进行比较和排序,从而优先执行相对重要的测试用例.

围绕测试用例的优先级问题已经有大量的研究:Wong 等人[3]最先在回归测试中运用优先级技术,将测试用例集按照其对代码的覆盖能力进行排序,并以此标准依次选择执行测试用例;Rothermel 等人[4]通过一系列有针对性的实验研究,证实了优先级技术在提高检错率方面的有效性;Elbaum[5]和 Arpad 等人[6]从语句覆盖、分支覆盖和检错能力等方面提出了多种基于动态反馈信息的优先级算法;屈波等人[7]讨论了基于测试用例设计信息的回归测试优先级设定问题,利用测试用例的设计和执行信息对测试用例优先级进行动态调整;Walcott 等人[8]研究了与时间因素相关的优先级方法,提出了按照测试用例的历史执行时间对其进行优先级排序的方法;Tonella等人[9]提出了一种机器学习的方法,利用测试者的经验对测试用例重要程度进行评估和排序;Srikanth等人[10]在系统级别上对基于需求的测试用例优先级排序问题进行了研究,根据测试用例满足需求的情况以及需求的权重设定其优先级;Kavitha 等人[11]在 Srikanth 研究的基础上,从需求变更、需求实现复杂度等方面出发,提出基于需求覆盖权重的优先级排序技术;Yoo 等人[12]则在测试用例选择中引入 Pareto 效率方法,以此结合多目标优化来设置测试用例优先级;Kim 等人[13]将回归测试视为一系列有序的行为序列,提出了综合考虑各种测试历史的优先级技术.另外,针对现有测试用例优先级技术缺乏明确的优化目标这一问题,我们提出了一种基于受控马尔可夫链的测试用例优先级在线调整方法[14,15].

针对现有的测试用例优先级算法很少关注测试用例静态的需求覆盖情况等设计信息和测试人员需要在线收集程序中每条语句的相关信息、对代码信息的依赖度较高这些情况,本文提出了一种基于需求的多目标优化测试用例优先级在线调整策略:首先,基于测试用例设计信息定义测试用例需求覆盖率矩阵;然后,在此基础上引入了关注需求覆盖率、基于需求的测试用例重要度和失效率这 3 个测试用例优先级影响因子,并根据测试过程中已覆盖的需求情况和测试用例的执行结果对测试用例的关注需求覆盖率和测试用例失效率进行在线估计;最后,实现测试用例优先级的在线调整.该方法的优点在于:在不依赖代码信息的条件下保留了优先级在线调整的优点,避免了额外的开销.同时,用 3 个因子共同确定测试用例优先级,避免了确定因素过于单一的缺点.

1 基于需求的测试用例优先级计算方法

1.1 优先级影响因素引入

在软件测试工程实践中,测试工程师最理想的目标是:尽可能地用最短的时间检测到尽可能多严重等级较高的软件缺陷.为此,首先要解决的问题是确定测试停止标准(即,软件测试充分性标准).在实际测试过程中,测试人员通常将覆盖率作为测试停止标准,实践证明:达到一定覆盖率标准的软件,通常其测试都是较为充分的,其质量或可靠性也较高[16].人们常用的覆盖率标准有代码覆盖率、功能覆盖率等,但实际上,功能是由一段段代码组成的,因此无论是功能点还是代码块,都可以被抽象成广义需求[17],基于广义需求的需求覆盖率标准可以广泛应用于各种类型的白盒测试和黑盒测试中.鉴于此,本文将需求覆盖率作为测试停止标准.

在测试过程开始之前,通常测试用例已设计完毕,测试人员可根据测试用例中的设计信息预期测试需求的覆盖情况,从而得到测试用例的需求覆盖率矩阵[18].为了用最短的时间达到需求覆盖率,要求每次都选取对整个的累积覆盖率贡献较大的测试用例,而不是单独考虑各测试用例固有的需求覆盖率.为此,本文引入关注需求集的概念,该概念是 Chen 等人[19]在介绍他们研究的 Test-Tube 工具时所提出来的,他们的设计思路是:在修改软件

时就将与之相关的测试确定下来,并据此确定相应的测试覆盖单元集合(即,关注需求集),从而寻求合适的测试用例.借鉴该思想,本文将测试过程中尚未被覆盖的测试需求称为关注需求,将测试用例的关注需求覆盖率作为决定测试用例优先级的一个重要因素,并根据测试用例执行情况对其进行在线估计和调整.由于关注需求引发软件失效的风险较大,因此优先执行关注需求覆盖率较大的测试用例,一方面有助于快速提高需求覆盖率、缩短开发周期,另一方面能够快速提高软件可靠性和软件测试效率.

由于测试用例的失效最终是由测试需求的失效引起的,而一个测试用例通常可覆盖多个测试需求,因此可以用测试用例覆盖的所有需求的失效率来衡量测试用例失效率.研究表明,覆盖相同或相似测试需求的测试用例往往会检测出相同或相似的错误[20].为了进一步确认问题或检测到类似问题,测试人员会对与之相关的测试用例都执行一遍,而这种测试用例的相关性通常表现在覆盖了相同的需求,在测试过程中及时捕获并利用这些信息,有利于提高发现错误的效率,为后续故障定位等工作提供更多的支持.因此,本文利用这种相关性,引入基于测试结果的测试用例失效率估计方法,在测试过程中,根据测试用例是否检测到缺陷对相关需求的失效率进行在线调整,进而实现测试用例失效率的在线调整,从而达到优先执行失效率较大的测试用例的目的,进而提高测试用例集的缺陷检测效率.同时,为了尽早检测到严重等级较高的软件缺陷,可以通过优先执行覆盖重要等级高的需求的测试用例来实现.实践结果表明,用户经常使用的软件功能只有 40%左右[17].一般来说,对用户更加重要的需求进行验证和测试,可以提高测试效益,因此,本文引入由需求重要度衡量的测试用例重要度因子,而需求重要度则根据测试人员的经验和用户对软件的使用情况来确定.

综上,本文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 3 个因素共同确定一个基于需求的测试用例优先级问题,并针对 3 个因素引入权重因子,保证工程人员可以根据具体的测试对象和环境对 3 个因素的重要程度进行调整,从而实现提高测试效率的目的.

1.2 变量定义

为了更好地描述基于需求的测试用例优先级问题,下面对文中涉及的基本概念进行如下定义.

定义 1(初始测试需求集 R_{t_0}). $R_{t_0}=\{r_1,r_2,...,r_m\}$ 是待测软件系统要求覆盖的测试需求的集合.这里所指的需求可以是通常所说的功能需求,而功能是由某一段代码集合实现的.可见,一个需求在本质上对应着一个代码集合,因此需求也可以是特定类、语句块或语句.

定义 2(初始测试用例集 T_{t_0}). $T_{t_0}=\{c_1,c_2,...,c_n\}$ 是针对目标软件系统及其测试需求集设计的一组测试用例的集合.

定义 3(测试覆盖矩阵 $\Delta(R,T)$). $\Delta(R,T)$是一个$|R|\times|T|$的二进制矩阵,从测试用例设计信息中提取而来,定义测试用例集 T_{t_0} 到需求集 R_{t_0} 的覆盖关系,矩阵元素如下式定义:

$$\delta(r_i,c_j)=\begin{cases}1, & c_j\text{覆盖了}r_i\\ 0, & c_j\text{未覆盖}r_i\end{cases} \tag{1}$$

定义 4(关注需求集 R_{t_i}). 表示在测试过程中,t_i 时刻尚未被覆盖的测试需求的集合.

定义 5(关注测试用例集 T_{t_i}). 表示在测试过程中,t_i 时刻尚未被执行的测试用例的集合.

定义 6(关注需求覆盖率 $C_{R_{c_j|t_i}}$). 表示测试用例 c_j 覆盖的关注需求数与总的关注需求数的比值,即:

$$C_{R_{c_j|t_i}}=\frac{count(R_{c_j|t_i})}{count(R_{t_i})} \tag{2}$$

其中, $R_{c_j|t_i}$ 代表测试用例 c_j 在 t_i 时刻覆盖的关注需求的集合.

定义 7(测试需求重要度 I_{r_i}). 测试需求的重要度一般由测试人员根据特定的标准确定,本文根据用户对软件系统的使用情况将需求的重要度赋值,对用户经常使用的需求赋予较高的优先级.

定义 8(测试需求失效率 $FR_{r_i|t_i}$). $FR_{r_i|t_i}$ 表示 t_i 时刻需求 r_i 失效(或引发软件失效)的概率.

定义 9(失效需求集 $R_{c_j|FR_{r_i|t_i}\neq 0}$). 表示测试用例 c_j 所覆盖的失效率 $FR_{r_i|t_i}$ 不为 0 的测试需求的集合,用 m_{c_j} 表

示该集合元素(即测试需求)个数.

定义 10(公共失效需求集 $R_{(c_j,c_k)|FR_{r_i|t_i}\neq 0}$). 表示测试用例 c_j 的失效需求集 $R_{c_j|FR_{r_i|t_i}\neq 0}$ 与 c_k 的失效需求集 $R_{c_k|FR_{r_i|t_i}\neq 0}$ 的交集.

1.3 优先级计算方法设计

根据第 1.2 节中给出的定义,可将基于需求的测试用例排序问题形式化为一个多目标测试用例集约简问题,表述如下.

给定备选测试用例集 T_{t_0}、测试需求集 R_{t_0}、测试覆盖矩阵$\Delta(R,T)$和测试停止标准,寻找测试用例集的最优代表集,且满足尽早达到测试覆盖率标准、尽快覆盖重要的测试需求以验证重要的软件功能和覆盖具有较高失效率的测试需求以尽早检测到更多的软件缺陷的 3 个测试目标.

为达到尽早达到测试覆盖率标准,要求优先执行关注需求覆盖率大的测试用例;为达到尽快覆盖重要的测试需求,要求优先执行覆盖了具有较高重要度需求的测试用例,事实上,测试需求重要度决定了测试用例的重要程度,但是由于一个测试用例通常覆盖了多个测试需求,本文用测试用例所覆盖的各个测试需求的重要度 I_{r_i} 的平均值来表示测试用例重要度 I_{c_j},如公式(3)所示;为了达到覆盖具有较高失效率的测试需求,要求优先执行覆盖率具有较高失效率的测试需求的测试用例,与测试需求重要度相同,测试需求失效率也决定了测试用例引发软件失效的概率,本文用测试用例所覆盖的各个可能引发软件失效的测试需求的失效率平均值表示测试用例失效率 $FR_{c_j|t_i}$,如公式(4)所示.

$$I_{c_j}=\frac{\sum_{i=1}^{n} I_{r_i}\delta(r_i,c_j)}{\sum_{i=1}^{n}\delta(r_i,c_j)} \tag{3}$$

$$FR_{c_j|t_i}=\frac{\sum_{i=1}^{n} FR_{r_i|t_i}\delta(r_i,c_j)}{m_{c_j}} \tag{4}$$

那么,测试用例 c_j 的优先级 P_{c_j} 由测试用例的关注需求覆盖率、测试用例重要度和测试用例失效率这 3 个因子确定,用三者的加权平均值表示,如下所示:

$$P_{c_j|t_i}=\alpha\cdot C_{R_{c_j|t_i}}+\beta\cdot I_{c_j}+\gamma\cdot FR_{c_j|t_i} \tag{5}$$

其中,α,β,γ分别表示关注需求覆盖率、测试用例重要度和测试用例失效率的权重因子,其取值可以根据测试对象及环境进行调整(满足$\alpha+\beta+\gamma=1$ 即可),从而调整对 3 个目标的偏重.

将公式(3)、公式(4)带入公式(5),可得公式(6):

$$P_{c_j|t_i}=\alpha\cdot C_{R_{c_j|t_i}}+\beta\cdot\frac{\sum_{i=1}^{n} I_{r_i}\delta(r_i,c_j)}{\sum_{i=1}^{n}\delta(r_i,c_j)}+\gamma\cdot\frac{\sum_{i=1}^{n} FR_{r_i|t_i}\delta(r_i,c_j)}{m_{c_j}} \tag{6}$$

由公式(6)可知, $P_{c_j|t_i}$ 是一个关于需求的函数.即:测试用例的优先级完全由其覆盖的需求决定,决定因子分别是关注需求覆盖率 $C_{R_{c_j|t_i}}$、测试需求重要度 I_{r_i} 以及测试需求失效率 $FR_{r_i|t_i}$.

2 多目标优化的测试用例优先级在线调整策略

多目标优化算法被广泛地应用于多目标问题的求解[21],直接计算 Pareto 最优解集并取得了一定的效果,但其仍然存在一定的局限性,如未能较好地解释早熟问题和欺骗问题、缺少完整的收敛性证明、对于解决约束优化问题缺乏行之有效的手段[22].与多目标优化算法相比,传统方法具有更低的计算复杂度,收敛速度较快,设计简单易懂,易于数学建模且理论支撑较为充分,更有利于在实际的软件测试中推广应用.因此,本文采用传统方法中的加权求和法求解多目标测试用例优化问题.该方法分别对 3 个测试目标赋予权值,并将其转化为一个单目标优化问题,以评价测试用例优先级,与直接计算 Pareto 最优解集相比,具有更低的计算复杂度.

测试开始之前,根据初始的关注需求覆盖率、测试需求重要度和测试需求失效率为每一个测试用例赋予一

个初始的优先级.在测试过程中,根据测试结果及其相关反馈信息对测试策略进行实时在线调整,即:实现测试用例优先级的在线调整,使得每次执行的测试用例最符合当前的测试情况,从而尽早地检测到尽可能多的软件缺陷.同时,在保证软件质量的前提下尽早地完成测试,提高软件测试效率.

2.1 参数在线估计方法设计

由公式(6)可知,测试用例的优先级由关注需求覆盖率 $C_{R_{c_j|t_i}}$、测试需求重要度 I_{r_i} 以及测试需求失效率 $FR_{r_i|t_i}$ 这 3 个因素所决定.测试开始前,我们需要根据软件的实际情况求得关注需求覆盖率、测试需求重要度和测试需求失效率的初始值,从而为每一个测试用例赋予一个初始的优先级;测试人员根据定义 7 确定测试需求重要度,t_i 时刻的关注需求覆盖率 $C_{R_{c_j|t_i}}$ 和测试需求失效率 $FR_{r_i|t_i}$ (即,测试用例失效率)为未知参数,需在每次执行完测试用例后,根据测试结果及相关反馈信息对二者进行实时的在线估计.

在实际测试中,测试自动化的应用极大地减少了测试过程中人力的参与度;但在测试初期,仍然需要测试人员执行相关初始化工作,如:从需求文档中提取测试需求、指定测试目标、参数设定等[18].因此,本文采用类似的方法,通过人工审查的方式从相关评测程序测试开发文档中分析获取测试需求、测试需求重要度、需求初始失效率等实验参数的初始值.其中,测试需求重要度由测试人员根据软件系统的使用情况对需求的重要度赋值[10,11],将必用、常用、很少用、几乎不用的需求重要度分别赋值为 1,0.7,0.4,0.1.相应的需求若引发缺陷,则其对应的严重等级分别为致命的、严重的、一般的以及较轻的.

2.1.1 关注需求覆盖率估计方法

在测试过程中,关注需求集和各个测试用例所覆盖的关注需求数是一个不断发生变化的过程,导致测试用例的关注需求覆盖率也不断发生变化.为了保证测试能够尽快达到覆盖率标准,优先执行关注需求覆盖率较大的测试用例,即,最大限度地覆盖未被覆盖的需求.为此,本文设计了一种关注需求覆盖率在线估计算法 RCE (requirement coverage estimation,见算法 2.1.1),用于在线估计各时刻每个关注测试用例的关注需求覆盖率 $C_{R_{c_j|t_i}}$.在 $t_{i-1}(i\geqslant 1)$时刻所选取的测试用例 $A_{t_{i-1}}$ 后,该算法利用 t_{i-1} 时刻的关注需求集 $R_{t_{i-1}}$、关注测试用例集 $T_{t_{i-1}}$ 及各关注测试用例所覆盖的关注需求 $R_{c_j|t_{i-1}}$,得到 t_i 时刻的关注需求集 R_{t_i}、关注测试用例集 T_{t_i} 及 t_i 时刻各关注测试用例所覆盖的关注需求 $R_{c_j|t_i}$,进而得到各关注测试用例的关注需求覆盖率 $C_{R_{c_j|t_i}}$,并为下一次求解关注需求覆盖率做好数据准备工作.

其中,t_i 时刻的关注需求集 R_{t_i} 由 t_{i-1} 时刻的关注需求集 $R_{t_{i-1}}$ 与 t_{i-1} 时刻执行的测试用例 $A_{t_{i-1}}$ 所覆盖的关注需求集的差集表示,如公式(7)所示.

$$R_{t_i}=R_{t_{i-1}}-R_{A_{t_{i-1}}|t_{i-1}} \tag{7}$$

t_i 时刻的关注测试用例集 T_{t_i} 通过将 $A_{t_{i-1}}$ 从 t_{i-1} 时刻的关注测试用例集 $T_{t_{i-1}}$ 中移除得到,如公式(8)所示.

$$T_{t_i}=T_{t_{i-1}}-A_{t_{i-1}} \tag{8}$$

每执行完一个测试用例后,都需要对当前的关注测试用例的关注需求覆盖情况进行更新,以进一步计算出关注需求覆盖率.若 t_{i-1} 时刻测试用例 c_j 的关注需求集 $R_{c_j|t_{i-1}}$ 与 $A_{t_{i-1}}$ 所覆盖的关注需求集 $R_{A_{t_{i-1}}|t_{i-1}}$ 中存在相同的关注需求,则 t_i 时刻测试用例 c_j 的关注需求集 $R_{c_j|t_i}$ 由 t_{i-1} 时刻该用例的关注需求集 $R_{c_j|t_{i-1}}$ 与 $A_{t_{i-1}}$ 所覆盖的关注需求 $R_{A_{t_{i-1}}|t_{i-1}}$ 的差集表示,如公式(9)所示,此时,各关注测试用例的关注需求覆盖率则如公式(10)所示.

$$R_{c_j|t_i}=R_{c_j|t_{i-1}}-(R_{c_j|t_{i-1}}\cap R_{A_{t_{i-1}}|t_{i-1}}),c_j\in T_{t_i} \tag{9}$$

$$C_{R_{c_j|t_i}}=\frac{count(R_{c_j|t_i})}{count(R_{t_i})},c_j\in T_{t_i} \tag{10}$$

估计 t_i 时刻各测试用例的关注需求覆盖率 $C_{R_{c_j|t_i}}$ 的具体算法如下所示.

算法 2.1.1. 关注需求覆盖率估计算法 RCE.

Estim($C_{R_{c_j|t_i}}$) //估计 $C_{R_{c_j|t_i}}$,$i\geqslant 1$

输入: $R_{t_{i-1}}, T_{t_{i-1}}, R_{c_j|t_{i-1}}$ //输入 t_{i-1} 时刻的关注需求集、关注测试用例集及各关注测试用例的关注需求;

输出: $R_{t_i}, T_{t_i}, R_{c_j|t_i}, C_{R_{c_j|t_i}}$ //输出 t_i 时刻的关注需求集、关注测试用例集及各关注测试用例的关注需求,关注需求覆盖率.

BEGIN

Updata(R_{t_i}) //由公式(7)更新关注需求集

$R_{t_i} = R_{t_{i-1}} - R_{A_{t_{i-1}}|t_{i-1}}$;

Updata(T_{t_i}) //由公式(8)更新关注测试用例集

$T_{t_i} = T_{t_{i-1}} - A_{t_{i-1}}$;

FOR j=1 to l //遍历所有关注用例

IF $c_j \in T_{t_i}$ AND $R_{c_j|t_{i-1}} \cap R_{A_{t_{i-1}}|t_{i-1}} \neq \varnothing$ THEN //判断 t_{i-1} 时刻关注需求集 $R_{A_{t_{i-1}}|t_{i-1}}$ 是否存在相同关注需求

Updata($R_{c_j|t_i}$) //由公式(9)更新关注用例覆盖的关注需求

$R_{c_j|t_i} = R_{c_j|t_{i-1}} - (R_{c_j|t_{i-1}} \cap R_{A_{t_{i-1}}|t_{i-1}})$;

Calculate($C_{R_{c_j|t_i}}$) //由公式(10)求 c_j 的关注需求覆盖率

$$C_{R_{c_j|t_i}} = \frac{count(R_{c_j|t_i})}{count(R_{t_i})};$$

END

2.1.2 测试用例失效率估计方法

由于覆盖相同测试需求的测试用例往往会检测出相同的错误,本文利用这种相关性,在测试过程中根据测试执行结果对各个需求的失效率进行实时的调整,进而实现测试用例失效率的在线调整.为此,本文设计了测试用例失效率在线估计算法 FRE(failure rate estimation,见算法 2.1.2),用于在线估计各时刻各测试用例的失效率 $FR_{c_j|t_i}$.

若 t_{i-1} 时刻执行的测试用例 $A_{t_{i-1}}$ 检测到缺陷,由于 $A_{t_{i-1}}$ 覆盖了多个测试需求,因此并不能马上定位是哪个需求引发了软件失效,该测试用例覆盖的任一需求的失效都可能是引发该用例失效的原因,所以相关需求引发软件失效的可能性都有所提高.为此,将该用例所覆盖的需求的失效率都增加Δc,未被该用例所覆盖的需求的失效率保持不变,则 t_i 时刻各个需求的失效率 $FR_{r_i|t_i}$ 可用公式(11)表示.

$$FR_{r_i|t_i} = \begin{cases} FR_{r_i|t_{i-1}} + \Delta c, & r_i \in R_{A_{t_{i-1}}} \\ FR_{r_i|t_{i-1}}, & \text{other} \end{cases} \tag{11}$$

此时,若测试用例 $c_j (c_j \in T_{t_i})$ 与 $A_{t_{i-1}}$ 的公共失效需求集 $R_{(c_j, A_{t_{i-1}})|FR_{r_i|t_i} \neq 0} \neq \varnothing$ (用 n 表示其元素个数),则测试用例 c_j 的失效率 $FR_{c_j|t_i}$ 将增加,可用公式(12)表示.

$$FR_{c_j|t_i} = (FR_{c_j|t_{i-1}} \times m_{c_j} + \Delta c \times n)/m_{c_j} \tag{12}$$

若 t_{i-1} 时刻执行的测试用例 $A_{t_{i-1}}$ 未检测到缺陷,则该测试用例所覆盖的需求的失效率都变为 0,即,认为这些需求都将不会引发软件失效,则这些需求将不再影响相关测试用例的失效率,未被该用例所覆盖的需求的失效率保持不变,则 $FR_{r_i|t_i}$ 时刻各个需求的失效率可用公式(13)表示.

$$FR_{r_i|t_i} = \begin{cases} 0, & r_i \in R_{A_{t_{i-1}}} \\ FR_{r_i|t_{i-1}}, & \text{other} \end{cases} \tag{13}$$

此时,若测试用例 $c_j (c_j \in T_{t_i})$ 与 $A_{t_{i-1}}$ 的公共失效需求集 $R_{(c_j, A_{t_{i-1}})|FR_{r_i|t_i} \neq 0} \neq \varnothing$, 则需要更新 c_j 的失效需求集 $R_{c_j|FR_{r_i|t_i} \neq 0}$, 将公共失效需求集中的测试需求从 c_j 的关注需求集中移除(如公式(14)所示);而测试用例 c_j 的失效率则可用公式(15)表示.

$$R_{c_j|FR_{r_i|t_i} \neq 0} = R_{c_j|FR_{r_i|t_i} \neq 0} - R_{(c_j, A_{t_{i-1}})|FR_{r_i|t_i} \neq 0} \tag{14}$$

$$FR_{c_j|t_i}=\begin{cases}0, & R_{(c_j,A_{t_{i-1}})|FR_{r_i|t_i}\neq 0}\neq\varnothing\\ \dfrac{\sum_{i=1}^{m}p\cdot FR_{r_i|t_i}}{m_{c_j}}, & \text{other}\end{cases} \tag{15}$$

公式(15)中,m 表示初始测试需求数,m_{c_j} 表示失效需求集 $R_{c_j|FR_{r_i|t_i}\neq 0}$ 的测试需求个数,$\sum_{i=1}^{m}p\cdot FR_{r_i|t_i}$ 表示 c_j 所覆盖的关注需求的失效率之和,其中,当 $r_i\in R_{(c_j,A_{t_{i-1}})|FR_{r_i|t_i}\neq 0}$ 时,p=1;否则,p=1.

估计 t_i 时刻各测试用例的失效率 $FR_{c_j|t_i}$ 的具体算法如下所示.

算法 2.1.2. 测试用例失效率估计算法 FRE.

Estim($FR_{c_j|t_i}$); //估计 $FR_{c_j|t_i}$,$i\geqslant 1$

输入: $FR_{r_i|t_{i-1}}$,$R_{c_j|t_{i-1}}$,m_{c_j} //t_{i-1} 时刻的需求失效率、关注需求及测试用例失效需求集中测试需求个数;

输出: $FR_{r_i|t_i}$,$R_{c_j|t_i}$,m_{c_j} //t_i 时刻需求失效率、覆盖的关注需求及测试用例失效需求集中测试需求个数.

```
BEGIN
IF ( A_{t_{i-1}} 检测到缺陷)
  IF  r_i ∈ R_{A_{t_{i-1}}}  THEN  FR_{r_i|t_i} = FR_{r_i|t_{i-1}} + Δc   //由公式(11)更新 t_i 时刻各个需求的失效率
  ELSE  FR_{r_i|t_i} = FR_{r_i|t_{i-1}};
  FOR j=1 to l   //遍历所有关注测试用例
    IF  c_j ∈ T_{t_i}  AND  R_{(c_j,A_{t_{i-1}})|FR_{r_i|t_i}≠0} ≠ ∅  THEN   //判断公共失效需求集 R_{(c_j,A_{t_{i-1}})|FR_{r_i|t_i}≠0} 是否为空集
      n = count(R_{(c_j,A_{t_{i-1}})|FR_{r_i|t_i}≠0});   //计算公共失效需求集 R_{(c_j,A_{t_{i-1}})|FR_{r_i|t_i}≠0} 的测试需求数
      FR_{c_j|t_i} = (FR_{c_j|t_{i-1}} × m_{c_j} + Δc × n) / m_{c_j};
IF ( A_{t_{i-1}} 未检测到缺陷)
  IF  r_i ∈ R_{A_{t_{i-1}}}  THEN  FR_{r_i|t_i} = 0   //由公式(13)更新 t_i 时刻各个需求的失效率
  ELSE  FR_{r_i|t_i} = FR_{r_i|t_{i-1}};
  FOR j=1 to l   //遍历所有关注测试用例
    IF  c_j ∈ T_{t_i}  AND  R_{(c_j,A_{t_{i-1}})|FR_{r_i|t_i}≠0} ≠ ∅  THEN
    R_{c_j|FR_{r_i|t_i}≠0} = R_{c_j|FR_{r_i|t_i}≠0} − R_{(c_j,A_{t_{i-1}})|FR_{r_i|t_i}≠0};   //由公式(14)更新测试用例 c_j 的失效需求集
      IF  R_{(c_j,A_{t_{i-1}})|FR_{r_i|t_i}≠0} ≠ ∅  THEN  FR_{c_j|t_i} = 0
      ELSE THEN
        m_{c_j} = count(R_{c_j|FR_{r_i|t_i}≠0});   //计算测试用例 c_j 的失效需求集 R_{c_j|FR_{r_i|t_i}≠0} 的测试需求数
        FR=0;
        FOR i=1 to m   //m 为初始测试需求数
          IF  r_i ∈ R_{(c_j,A_{t_{i-1}})|FR_{r_i|t_i}≠0}  THEN
            FR = FR + FR_{r_i|t_i};
        FR_{c_j|t_i} = FR / m_{c_j};
END
```

2.2 多目标优先级在线调整策略设计

多目标测试用例优先级动态排序问题是一个 NP 完全问题[19],解决这一问题可采用启发式贪婪搜索算法[20],该算法的核心思想是:一次选取一个局部最优解,循环至达到停止标准,所有的局部最优解组合成问题的整体最优解.在启发式贪婪搜索框架内,本文设计了一种基于需求的测试用例优先级在线调整算法 POLA (priority on-line adjustment,见算法 2.2),用于在线调整各时刻每个测试用例的优先级.

本文采用的启发式贪婪搜索框架内自适应调整测试用例优先级的技术是:根据前一时刻的关注需求覆盖率、测试用例重要度和测试用例失效率,对当前时刻备选测试用例的优先级进行计算,选取并执行当前状态下优先级最高的测试用例,利用算法 RCE,FRE 自适应调整关注测试用例集、关注测试需求集及各关注测试用例的关注需求覆盖率和失效率,进一步重新调整相关的测试用例的优先级,直至满足需求覆盖率标准.在启发式贪婪搜索框架内自适应调整测试用例优先级的策略是:根据初始的关注需求覆盖率、测试用例重要度和测试用例失效率对备选测试用例的优先级进行初始化计算,从关注测试用例集中选择并执行当前状态下优先级最高的测试用例;将所选用例从关注测试用例中移除,同时,从关注测试需求中移除该测试用例所覆盖的需求;再根据测试用例执行结果和被执行测试用例的关注测试需求覆盖情况,自适应调整各关注测试用例的关注需求覆盖率和失效率,从而调整相关测试用例的优先级;最后,更新关注需求覆盖率,直至达到需求覆盖率标准.

下面结合关注需求覆盖率估计算法和关注测试用例失效率估计算法,给出基于需求的测试用例优先级在线调整算法 POLA.

算法 2.2. 优先级在线调整算法 POLA.

输入:备选测试用例集 T_{t_0};

输出:优化选取的测试用例.

BEGIN

Initialize $R_{t_0}, T_{t_0}, \Delta(R,T), I_{r_i}, FR_{r_i|t_0}, m_{r_i|t_0}, \alpha,\beta,\gamma,\Delta c, C$ //数据初始化

l=*count*(T_{t_0}); l=*count*(R_{t_0}); //计算初始测试用例数、初始测试需求数

$C_{t_i}(i=0)=0$; //t_0 时刻需求覆盖率为 0

FOR j=0 to l THEN Calculate $P_{c_j|t_0}$; //计算测试用例优先级

GET A_{t_0} IF $P_{c_j|t_0}(c_j=A_{t_0})=\max\{P_{c_j|t_0}, j=1 \text{ to } l\}$; //选取优先级最高的测试用例

Run(A_{t_0}); //运行测试用例 A_{t_0}

i++; l–; //每执行一个测试用例,消耗 1 单位时间,关注测试用例数减 1

Update C_{t_i}; //更新测试需求覆盖率

While $C_{t_i}<C$; //达到测试覆盖率标准则停止测试,C 表示实验设定的覆盖率标准

Estim($C_{R_{c_j|t_i}}$); //调用关注需求覆盖率估计算法

Estim($FR_{R_{c_j|t_i}}$); //调用关注测试用例失效率估计算法

FOR i=0 to l THEN Calculate $P_{c_j|t_i}$;

GET A_{t_i} IF $P_{c_j|t_i}(c_j=A_{t_i})=\max\{P_{c_j|t_i}, j=1 \text{ to } l\}$;

Run(A_{t_i}); //运行测试用例 A_{t_i}

i++; l–;

Update C_{t_i}; //更新测试需求覆盖率

END

2.3 算法复杂度分析

与已有的大多数测试用例优先级技术相比,本文所提方法不依赖于代码覆盖率技术,无需在测试过程中收集代码信息,很大程度上降低了技术实现的复杂度.同时,在时间复杂度方面,基于需求的测试用例优先级在线调整算法的时间开销主要源于关注需求覆盖率估计算法 RCE、测试用例失效率估计算法 FRE 和测试用例优先级在线调整算法 POLA 中的 for 循环.不难看出,这些 for 循环体的时间复杂度处于同一数量级.假设备选用例的数量为 m,最坏情况下,所有的测试用例都执行 1 次,则一个 for 循环最多执行 $m(m+1)/2$ 次,即,时间复杂度为 $O(m^2)$.在实际应用中,对于 10^2 级的测试用例集,在线调整一次测试用例优先级所需执行时间一般不超过 10s,这相对于长达数小时甚至数天的回归测试完全可忽略不计.因此,利用本文所提出的技术和算法对测试用例集优

先级排序进行处理后,所获得的效率上的提高远大于其开销.

3 实验验证

本文所提方法和策略主要有两个方面:(1) 通过引入关注需求覆盖率、测试用例重要度和测试用例失效率这3个影响因子,提出一种多目标的测试用例优先级计算方法;(2) 设计了关注需求覆盖率和测试用例失效率的在线估计方法及算法(RCE 算法和 FRE 算法),从而设计出一种基于需求的测试用例优先级在线调整策略.为了验证本文所提方法和策略的有效性,我们设计了实验,以解答以下两个问题:

- 问题 1. 关注需求覆盖率、测试用例重要度和测试用例失效率如何影响测试用例优先级的设定,这 3 个因子的引入,能否实现尽早地检测到尽可能多的、严重等级较高的软件缺陷这一多目标测试用例优先级问题?
- 问题 2. 本文所提出的基于需求的测试用例优先级在线调整策略的效率和缺陷检测能力如何?

3.1 实验设计

在实证研究中,研究人员常常采用开源软件的评测程序来验证研究内容的有效性,其中,SIR 库和 SourceForge 包含了绝大部分评测程序[23,24].因此,为了验证本文设计的测试用例优先级调整策略在实际程序中的有效性,我们选取了 MET,CZT 及 Bash 这 3 个中等规模的开源软件程序.选择其最新版本和相应的用例集作为实验对象,具体相关信息见表 3.1.

Table 3.1 The related information of the three benchmark programs

表 3.1 3 个评测程序的相关信息

名称	代码行数	函数个数	缺陷个数	测试用例数
MET	10 674	270	11	80
CZT	27 246	756	27	548
BASH	59 800	1 051	15	1 061

在本文的实验中,需求覆盖矩阵、需求初始失效率及测试需求重要度等信息由相关评测程序测试文档分析获取.其中,3 个开源软件程序的测试需求重要度见表 3.2,测试开始时各个需求的初始失效率见表 3.3.如果某个测试用例覆盖的需求引发失效,则该用例所覆盖的需求的失效率都增加Δc,本文取Δc=0.1.由于需求覆盖矩阵规模过大,文中并未给出.针对以上系统,设计实验 1 和实验 2 来回答问题 1 和问题 2.

Table 3.2 Test case importance degree

表 3.2 测试需求重要度

r_i	r_1	r_2	r_3	r_4	r_5	r_6	r_7	r_8	r_9	r_{10}	r_{11}	r_{12}	r_{13}	r_{14}	r_{15}	r_{16}	r_{17}	r_{18}	r_{19}	r_{20}
I_{r_i}	1	0.1	0.7	0.7	1	0.4	0.7	0.7	0.4	0.7	0.4	0.4	0.4	0.1	0.7	0.4	0.1	1	0.4	0.7

Table 3.3 Initial failure rate of test case

表 3.3 测试需求初始失效率

r_i	r_1	r_2	r_3	r_4	r_5	r_6	r_7	r_8	r_9	r_{10}	r_{11}	r_{12}	r_{13}	r_{14}	r_{15}	r_{16}	r_{17}	r_{18}	r_{19}	r_{20}
$FR_{r_i\|t_0}$	0.01	1	0.01	1	1	0.01	0.01	1	0.01	0.01	1	1	0.01	0.01	1	0.01	0.01	0.01	0.01	1

3.2 实验1——优先级影响因子及多目标优先级方法的有效性

为了验证关注需求覆盖率、测试用例重要度和测试用例失效率这 3 个因素在确定测试用例优先级中所做的贡献,本文针对评测程序 MET 设计了 3 组实验,其中,(α,β,γ)的取值分别为(1,0,0),(0,1,0),(0,0,1).这 3 组实验分别对应了 3 种常见的单目标测试用例排序方法.实验结果分别见表 3.4~表 3.6,其中,t_i 表示选取测试用例的各时刻, A_{t_i} 表示各时刻选取的测试用例, C_{t_i} 表示各时刻执行完测试用例后的需求覆盖率, B_{t_i} 表示各时刻所执行的用例中覆盖的含有缺陷的需求(×表 t 时刻所执行的测试用例未检测到缺陷).

Table 3.4 The test results when (α,β,γ) is set to (1,0,0)

表 3.4 (α,β,γ)为(1,0,0)时测试用例执行情况

t_i	t_1	t_2	t_3	t_4	t_5	t_6	t_7
A_{t_i}	c_3	c_{24}	c_{12}	c_{21}	c_{13}	c_7	c_{11}
C_{t_i}	0.25	0.52	0.63	0.78	0.89	0.95	1.00
B_{t_i}	r_8, r_{11}, r_{16}	r_{13}, r_{15}	r_{17}	r_1, r_5	r_4, r_{20}	×	×

Table 3.5 The test results when (α,β,γ) is set to (0,1,0)

表 3.5 (α,β,γ)为(0,1,0)时测试用例执行情况

t_i	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}	t_{11}	t_{12}
A_{t_i}	c_1	c_{23}	c_{27}	c_9	c_{13}	c_{24}	c_5	c_{11}	c_{26}	c_{12}	c_2	c_3
C_{t_i}	0.15	0.19	0.25	0.42	0.51	0.65	0.71	0.76	0.80	0.89	0.97	1.00
B_{t_i}	r_{12}, r_5	r_8	r_{20}	r_{15}	r_4, r_{11}	r_{13}	r_{16}	×	×	r_{17}	r_2	×

Table 3.6 The test results when (α,β,γ) is set to (0,0,1)

表 3.6 (α,β,γ)为(0,0,1)时测试用例执行情况

t_i	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}	t_{11}	t_{12}	t_{13}	t_{14}
A_{t_i}	c_{11}	c_{13}	c_{19}	c_{27}	c_{25}	c_{21}	c_{22}	c_{28}	c_2	c_5	c_9	c_{24}	c_{30}	c_{12}
C_{t_i}	0.09	0.23	0.25	0.25	0.33	0.41	0.52	0.59	0.67	0.81	0.87	0.94	0.94	1.00
B_{t_i}	r_8	r_4, r_{11}	r_{12}	r_2, r_5	r_{20}	r_{15}	r_{13}	r_3	r_{16}	r_{18}	×	×	×	×

- 当(α,β,γ)的取值为(1,0,0)时,表示以关注需求覆盖率为测试用例优先级排序.由表 3.4 可知:每个决策时刻选取的都是关注需求覆盖率最大的测试用例,能够以最快的速度提高测试覆盖率,所以能够达到以最短的时间完成测试的目的.由表3.4可知:执行1个测试用例后,虽然需求覆盖率得到了快速提升且检测到缺陷,但检测到的缺陷严重程度低,而且严重的缺陷被检测到的时间较晚.当覆盖率达到 100%时,不一定能检测出所有严重的缺陷;
- 当(α,β,γ)的取值为(0,1,0)时,表示以测试用例重要度为测试用例优先级排序.由表 3.5 可知:每次选取的都是重要度最高的用例,重要的需求被优先覆盖到,因此能够优先检测到严重等级较高的缺陷,但其完成整个测试花费的测试用例较多;
- 当(α,β,γ)的取值为(0,0,1)时,表示以测试用例失效率为测试用例优先级排序.由表 3.6 可知:每次选取的都是失效率最高的测试用例,使得缺陷检测能力高的测试用例被优先执行,从而能够以最少的测试用例数检测到最多的缺陷;但相对于(α,β,γ)为(1,0,0)的情况,其完成整个测试花费的测试用例数仍然较多.

由以上分析可知,关注需求覆盖率、测试用例重要度和测试用例失效率这 3 个因素在决定测试用例优先级的问题上都能做出不同的有效贡献.在(α,β,γ)为(1,0,0),(0,1,0),(1,0,0)3 种不同的情况下,相当于 3 个单目标测试用例优先级排序方法.为了最大限度地提高测试效率,达到尽早检测到尽可能多的严重等级较高的软件缺陷这一多目标测试目的,测试人员需根据软件实际情况选取合适的(α,β,γ)的值,使 3 个因素达到一个平衡点.在此仿真实验中,取(α,β,γ)为(0.36,0.41,0.23)便能得到一个较为理想的结果,实验结果见表 3.7.

Table 3.7 The test results when (α,β,γ) is set to (0.36,0.41,0.23)

表 3.7 (α,β,γ)为(0.36,0.41,0.23)时测试用例执行情况

t_i	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8
A_{t_i}	c_{11}	c_{13}	c_{19}	c_{27}	c_{25}	c_{21}	c_{22}	c_{28}
C_{t_i}	0.15	0.32	0.41	0.55	0.69	0.86	0.93	1.00
B_{t_i}	r_8, r_{12}	r_4, r_{11}	r_5	r_2, r_{20}	r_{15}	r_{13}, r_6	r_{14}	×

这里给出的权重系数(α,β,γ)是通过在实验中不断尝试、不断调整而得到的能够产生较好测试效果的一组系数值,而在实际测试中,可利用专门的权重系数选择方法来确定,目前较为成熟的方法包括 Delphi 法、AHP 法、专家评分法等.

由表 3.7 可知:当取(α,β,γ)为(0.36,0.41,0.23)时,完成整个测试所花费的测试用例数是最少的.同时,重要的需求被尽早地覆盖,大多数严重等级较高的软件缺陷也被尽早地检测到.

综上可得:本文所引入的关注需求覆盖率、测试用例重要度和测试用例失效率这 3 个测试用例优先级影响因子是有效的,有助于实现尽早地检测到尽可能多的、严重等级较高的软件缺陷这一多目标测试用例优先级目标.

3.3 实验2——缺陷检测能力及测试效率

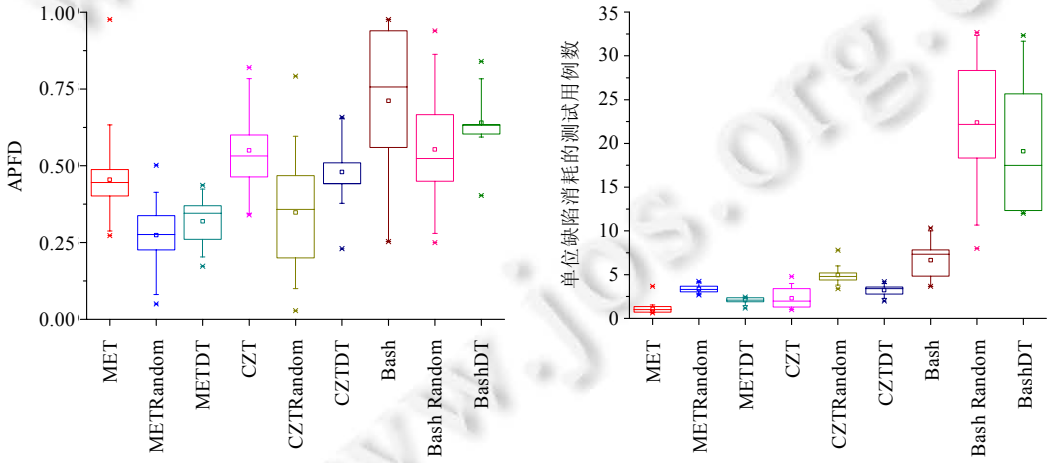
为了进一步验证本文所提出的基于多目标优化的测试用例优先级在线调整策略的测试效率和缺陷检测能力,我们通过实验将其与随机排序方法和确定性排序方法进行比较.在测试用例排序的度量标准研究中,为了计算执行过优先排序的测试用例集的检错能力,衡量测试用例集与其相应测试用例检测出错误的关系,评价测试用例排序的优化程度,研究人员常采用 APFD 度量标准来验证在线排序方法的有效性和优越性[23].

假设参与测试的测试用例集中共有 n 个测试用例,测试完成后检测出的错误共有 m 个,TF_i 代表测出第 1 个错误的测试用例在原测试用例集中的顺序.APFD 的计算公式如下所示:

$$APFD = 1 - \frac{TF_1 + TF_2 + \ldots + TF_m}{nm} + \frac{1}{2n} \tag{16}$$

其中,APFD 的取值在 0 和 100%之间,并且 APFD 的值越大,代表该测试排序的检错效率越高.

我们分别用随机排序方法、确定性排序方法和本文所提出的在线排序方法对评测程序 MET,CZT,Bash 进行多轮模拟测试,测试停止标准均为需求覆盖率达到 100%.在随机排序中,通过均匀概率分布从测试用例集中随机地选择测试用例,每个测试用例被选取的概率相等;在确定性排序中,采用本文所提出的优先级计算方法(即,由公式(6)所确定的优先级计算方法)计算各测试用例的优先级值,而不对相关参数进行在线调整,从而得到一个确定性的测试用例执行序列;在确定性排序方法和在线排序方法的前 50 轮测试中,均分别在(α,β,γ)的不同取值下进行,且(α,β,γ)的不同取值服从均匀概率分布.图 3.1(a)给出了实验中 3 种排序方法下 3 个评测程序的 APFD 值,图 3.1(b)给出了前 50 轮实验中 3 种排序方法在各轮测试中单位缺陷消耗的测试用例数.



(a) 3 种排序方法在评测程序中的 APFD 值 (b) 3 种排序方法单位缺陷消耗的测试用例数

Fig.3.1 Comparison of the APFD value and the number of test case used per unit defect of the three sorting methods

图 3.1 3 种排序方法的 APFD 值与单位缺陷消耗的测试用例数的比较

由箱线图 3.1(a)可知:在评测程序 MET 中,在线排序方法的 APFD 的最大值比确定性排序方法高 18.1%,确定性排序方法比随机性排序方法高 9.8%,在线排序方法的上下四分位数的值、平均值和中位数均高于确定性排序方法,远高于随机排序方法;对于评测程序CZT,在线排序方法的APFD的最大值比确定性排序方法高1.3%,确定性排序方法比随机性排序方法高 36.9%,在线排序方法的上四分位数的值均略高于确定性排序方法,均值和中位数略低于确定性排序方法,但远高于随机排序方法;评测程序 Bash 的情况与实验程序 CZT 相似,在线排序方法的 APFD 的最大值比确定性排序方法高 10.2%,确定性排序方法比随机性排序方法高 2.3%,在线排序方法的上四分位数的值和均值高于确定性排序方法,中位数略低于确定性排序方法,各项数值均远高于随机排序方法.从图 3.1(b)可以看出:在绝大多数情况下,在线排序方法的上下四分位数的值、平均值和中位线的值均不如确定性排序方法和随机性排序方法.这说明在线排序方法能够以较少的测试用例数检测到缺陷,测试代价相对较小.此外,本实验的 50 轮测试中,α,β,γ的值是随机选取的,用以模拟实际测试中各种不同权重的选取情况.实际应用中,工程人员将根据具体的被测对象及测试目标设置合理的α,β,γ.与确定性排序方法和随机性排序方法相比,在线排序方法将消耗更少的测试用例.

综上可知:本文所提出的基于多目标优化的测试用例优先级在线调整策略能够以更少的测试用例(即,能在更短的测试时间)检测到更多的、严重等级较高的软件缺陷,具有更高的缺陷检测能力和测试效率.

4 总结及未来工作

工程实践中,在对不同的软件版本进行测试时,对变更部分进行有选择的、有针对性的测试是最理想的选择,本文将这些变更的部分(功能点或者代码块)称为测试需求,从需求的角度对测试用例的优先级进行排序,并定义了一个多目标的测试用例优化排序问题,以解决测试用例优先级在线调整的目标问题.

为此,本文引入了 3 个影响测试用例优先级的因子,即:关注需求覆盖率、测试用例重要度和测试用例失效率,同时引入了α,β,γ这 3 个权重因子,使得工程人员可以根据测试对象的实际情况对 3 个优先级影响因子进行合理的调整,从而权衡这 3 方面的目标.其中,测试用例重要度是由测试人员根据用户对软件的实际使用情况预先确定的测试需求重要度计算而来,而关注需求覆盖率和测试用例失效率分别由各时刻测试用例的关注需求覆盖情况和测试用例执行结果估算而来.为此,本文分别设计了关注需求覆盖率估计方法和测试用例失效率估计方法及相应的算法,并在此基础上设计了一种基于多目标优化的测试用例优先级在线调整策略.该策略利用测试过程中反馈的实时信息对测试用例优先级进行在线调整,使其逐步适应当前测试环境,从而更好地提升测试效率.实验结果表明:与随机测试、单目标优先级排序方法和确定性排序方法相比,本文提出的基于测试需求的测试用例优先级在线调整策略总体上能够更早地检测到更多的、严重等级较高的软件缺陷,能够在更短的时间内完成同等质量的软件测试,从而缩短测试周期,降低测试成本,提高测试效率.

今后将进一步从以下两方面进行研究:一方面,进一步寻求与需求相关的测试用例优先级影响因素并对各因素的权重分配进行深入研究;另一方面,通过更多的实际的软件系统对基于测试需求的测试用例优先级在线调整策略的有效性进行验证,以进一步推动其在实践中的应用.

References:

[1] Rothermel G, Harrold MJ. Analyzing regression test selection techniques. IEEE Trans. on Software Engineering, 1996,22(8): 529−551. [doi: 10.1109/32.536955]

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

[3] Wong W, Horgan J, London S, Agrawal H. A study of effective regression testing in practice. In: Everet B, ed. Proc. of the Int'l Sypm. on Software Reliability Engineering. Albuquerque: IEEE Press, 1997. 264−274. [doi: 10.1109/ISSRE.1997.630875]

[4] Rothermel G, Unteh RH, Chu C, Harrold MJ. Prioritizing test cases for regression testing. IEEE Trans. on Software Engineering, 2001,27(10):929−948. [doi: 10.1109/32.962562]

[5] Elbaum S, Rothermel G, Kanduri S, Malishevsky AG. Selecting a cost-effective test case prioritization technique. Software Quality Journal, 2004,12(3):185−210. [doi: 10.1023/B:SQJO.0000034708.84524.22]

[6] Beszédes A, Gergely T, Schrettner L, Jász J, Langó L, Gyimóthy T. Code coverage-based regression test selection and prioritization in WebKit. In: Proc. of the 28th IEEE Int'l Conf. on Software Maintenance (ICSM). Washington: IEEE Press, 2012. 46−55. [doi: 10.1109/ICSM.2012.6405252]

[7] Qu B, Nie CH, Xu BW. Test case prioritization based on test suite design information. Chinese Journal of Computers, 2008,31(3): 431−439 (in Chinese with English abstract). [doi: 10.3321/j.issn:0254-4164.2008.03.008]

[8] Walcott KR, Soffa ML, Kapfhammer GM, Roos RS. Time-Aware test suite prioritization. In: Pollock L, ed. Proc. of the Int'l Symp. on Software Testing and Analysis. Portland: ACM Press, 2006. 1−12. [doi: 10.1145/1146238.1146240]

[9] Tonella P, Avesani P, Susi A. Using the case-based ranking methodology for test case prioritization. In: Mancoridis S, ed. Proc. of the 22nd IEEE Int'l Conf. on Software Maintenance. Los Alamitos: IEEE Press, 2006. 123−133. [doi: 10.1109/ICSM.2006.74]

[10] Srikanth H, Williams L. On the economics of requirements-based test case prioritization. In: Sullivan K, ed. Proc. of the Int'l Workshop on Economics-Driven Software Engineering Research. New York: ACM Press, 2005. 1−3. [doi: 10.1145/1083091.1083100]

[11] Kavitha R, Kavitha VR. Requirement based test case prioritization. In: Moses J, ed. Proc. of the Communication Control and Computing Technologies. Ramanathapuram: IEEE Press, 2010. 826−829. [doi: 10.1109/ICCCCT.2010.5670728]

[12] Yoo S, Harman M. Pareto efficient multi-objective test case selection. In: Rosenblum DS, ed. Proc. of the 2007 Int'l Symp. on Software Testing and Analysis. New York: ACM Press, 2007. 140−150. [doi: 10.1145/1273463.1273483]

[13] Kim JM, Porter A. A history-based test prioritization technique for regression testing in resource constrained environments. In: Feinberg D, ed. Proc. of the Int'l Conf. on Software Engineering. Orlando: IEEE Press, 2002. 119−129. [doi: 10.1145/581339.581357]

[14] Bao XA, Yao L, Zhang XW, Cao JW. Optimal testing for software defects based on controlled Markov chain. Computer Science, 2012,39 (5):117−119,136 (in Chinese with English abstract). [doi: 10.3969/j.issn.1002-137X.2012.05.027]

[15] Bao XA, Yao L, Zhang N, Song JY. Adaptive software testing based on controlled Markov chain. Journal of Computer Research and Development, 2012,49(6):1332−1338 (in Chinese with English abstract).

[16] Beizer B. Software Testing Techniques. 2nd., New York: Van Nostr and Reinhold, 1990. 168−203.

[17] Zhang XF, Nie CH, Xu BW, Qu B. Test case prioritization based on varying testing requirement priorities and test case costs. In: Mathur A, EricWong W, Lau MF, eds. Proc. of the Int'l Conf. on Quality Software. Portland: IEEE Press, 2007. 15−24. [doi: 10.1109/QSIC.2007.4385476]

[18] Gu Q, Tang B, Chen DX. A test suite reduction technique for partial coverage of test requirements. Chinese Journal of Computers, 2011,34(5):879−888 (in Chinese with English abstract). [doi: 10.3724/SP.J.1016.2011.00879]

[19] Chen YF, Rosenblum DS, Vo KP. TestTube: A system for selective regression testing. In: Fadini B, Osterweil L, Lamsweerde A, eds. Proc. of the 16th Int'l Conf. on Software Engineering (ICSE-16). Los Alamitos: IEEE Press, 1994. 211−220. [doi: 10.1109/ICSE.1994.296780]

[20] Chen TY, Huang DH, Zhou ZQ. On adaptive random testing through iterative partitioning. Journal of Information Science and Engineering, 2011,4(27):1449−1472.

[21] Xie T, Chen HW, Kang LS. Evolutionary algorithms of multi-objective problems. Chinese Journal of Computers, 2003,26(8): 997−1003 (in Chinese with English abstract). [doi: 10.3321/j.issn:0254-4164.2003.08.015]

[22] Gong MG, Jiao LC, Yang DD, Ma WP. Research on evolutionary multi-objective optimization algorithms. Ruan Jian Xue Bao/Journal of Software, 2009,20(2):271−289 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/3483.htm

[23] Cai KY, Chen TY, Li YC, Ning WY, Yu YT. Adaptive testing of software components. In: Haddad HM, ed. Proc. of the 20th ACM Symp. on Applied Computing. New York: ACM Press, 2005. 1463−1469. [doi: 10.1145/1066677.1067011]

[24] Chen X, Chen JH, Ju XL, Gu Q. Survey of test case prioritization techniques for regression testing. Ruan Jian Xue Bao/Journal of Software, 2013,24(8):1695−1712 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/4420.htm [doi: 10.3724/SP.J.1001.2013.04420]

附中文参考文献:

[7] 屈波,聂长海,徐宝文.基于测试用例设计信息的回归测试优先级算法.计算机学报,2008,31(3):431−439. [doi: 10.3321/j.issn:0254-4164.2008.03.008]

[14] 包晓安,姚澜,张晓文,曹建文.基于受控马尔可夫链的软件缺陷测试策略.计算机科学,2012,39(5):117−136. [doi: 10.3969/j.issn.1002-137X.2012.05.027]

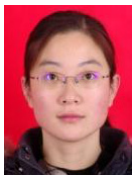
[15] 包晓安,姚澜,张娜,宋瑾钰.基于受控 Markov 链的软件自适应测试策略.计算机研究与发展,2012,49(6):1332−1338.

[18] 顾庆,唐宝,陈道蓄.一种面向测试需求部分覆盖的测试用例集简约技术.计算机学报,2011(5):879−888. [doi: 10.3724/SP.J.1016.2011.00879]

[21] 谢涛,陈火旺,康立山.多目标优化的演化算法.计算机学报,2003,26(8):997−1003. [doi: 10.3321/j.issn:0254-4164.2003.08.015]

[22] 公茂果,焦李成,杨咚咚,马文萍.进化多目标优化算法研究.软件学报,2009,20(2):271−289. http://www.jos.org.cn/1000-9825/3483.htm

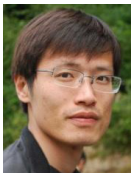
[24] 陈翔,陈继红,鞠小林,顾庆.回归测试中的测试用例优先排序技术评述.软件学报,2013,24(8):1695−1712. http://www.jos.org.cn/1000-9825/4420.htm [doi: 10.3724/SP.J.1001.2013.04420]



张娜(1977－),女,浙江奉化人,副教授,主要研究领域为软件工程,软件测试.

董萌(1989－),男,硕士生,主要研究领域为软件工程,软件测试.

姚澜(1988－),女,工程师,主要研究领域为软件工程,软件测试.



桂宁(1977－),男,博士,讲师,主要研究领域为软件工程,自适应软件.

包晓安(1973－),男,教授,主要研究领域为自适应软件,软件测试,智能信息处理.