

引用格式:张丽,戴丽丽,杜兰. 基于敏捷开发模式的回归测试用例排序技术[J]. 微电子学与计算机, 2020, 37(12): 48-52.
[ZHANG L, DAI L L, DU L. Test case prioritization for regression testing based on agile development mode[J]. Microelectron-
ics & Computer, 2020, 37(12): 48-52.]

基于敏捷开发模式的回归测试用例排序技术

张 丽,戴丽丽,杜 兰

(南京理工大学 紫金学院 计算机学院,江苏 南京 210046)

摘 要: 基于敏捷开发模式的回归测试,面临快速交付的问题.为此,提出一种动态回归测试排序算法:首先,将回归测试用例分为新功能测试用例和原功能测试用例;采用不同算法,分析需求与模块以及模块与测试用例之间映射关系,完成新功能测试用例排序;结合原有排序,分析新功能对原功能的影响,以及测试用例的历史执行信息,完成原功能测试用例排序.在测试序列执行过程中动态调整用例优先级.最后,使用 Google 公开数据集进行实验,验证算法的有效性,实验结果显示平均 APFD 值达到 98.08%.

关键词: 敏捷开发;回归测试;测试用例排序;软件测试

中图分类号: TP311

文献标识码: A

文章编号: 1000—7180(2020)12—0048—05

Test case prioritization for regression testing based on agile development mode

ZHANG Li, DAI Li-li, DU Lan

(Department of Computer Science, ZiJin College, Nanjing University of Science and Technology,
Nanjing 210046, China)

Abstract: Regression testing based on agile development mode is faced with the problem of fast delivery. So a new algorithm is proposed. Firstly, the regression test cases are divided into new function test cases and original function test cases, and different algorithms are adopted: analyzing the mapping relationship between requirements and modules as well as between modules and test cases, completing the sorting of new function test cases; combined with the original sorting, and analyzing the impact of new functions on the original functions, as well as the execution of test cases, completing the sorting of original function test cases. Then the priority of test cases is adjusted dynamically during the execution. Finally, through the experiments on the open data set of Google, the effectiveness of the algorithm is verified. The average APFD value is 98.08%.

Key words: agile development; regression testing; test case prioritization; software testing

1 引言

敏捷开发是一种新型软件开发模式,采用迭代、循序渐进的方法进行软件开发^[1],新版本的连续发布,使得回归测试频率很高.回归测试是重要的软件测试方法之一,在整个软件测试过程中占有很大的

比重,一般占软件产品测试预算的 80%以上,占软件维护预算的 50%以上^[2].

基于敏捷开发模式的回归测试用例集分为两种类型:新功能测试用例集和原功能测试用例集^[3].新版本的发布需要首先确保新功能正常,满

收稿日期: 2020—02—19; 修回日期: 2020—03—03

基金项目: 江苏省高等学校自然科学研究面上项目(19KJB520039);南京理工大学紫金学院校级科研项目(2019ZRKX0401009)

足用户需求. 原功能虽然在之前版本中已经过测试,也需要进行回归测试,验证新功能是否对原功能产生影响.

传统的回归测试用例优化技术没有考虑敏捷开发模式的特点. 基于敏捷开发模式的回归测试,新功能测试用例集和原功能测试用例集排序优化基于不同的因素,因此需要分别实现排序优化,并且确保新功能测试用例集优先测试.

本文结合新功能和原功能测试用例排序,实现基于敏捷开发模式的动态回归测试排序算法,并在执行过程中动态调整测试序列的优先级,验证了算法的有效性.

2 背景及相关研究

TCP 技术按照某种准则对测试用例进行排序,优先执行那些优先级高的测试用例,达到某种目标^[4]:故障发现速率、代码覆盖率等.

Elbaum 和 Rothermel 等人对 TCP 问题给出了一般性的定义^[5]:

给定:测试用例集 T , T 所有可能的排序集合为 PT ,以及 PT 到实数的目标函数 f ;

问题:寻找 $T' \in PT$,使得 $(\forall T'' \in PT)(T' \neq T'')[f(T') > f(T'')]$.

f 是测试用例集的排序准则,其值越大越好,因此 TCP 的目标是确定函数 f 以及具有最优排序的测试序列.

TCP 问题研究目前可以分为三类,分别是基于代码、基于模型和基于需求^[2].

基于需求的 TCP 技术是从需求角度出发,分析测试用例对于需求的覆盖情况,并且分析测试用例的历史执行信息,作为测试用例排序准则. 屈波等人^[6]基于测试用例的设计信息和执行信息,对测试用例优先级进行动态调整. 张娜等人^[7]从需求的角度对测试用例的优先级进行排序,实现了一个多目标的测试用例优化排序. 常龙辉等人^[8]分析测试用例历史执行信息,提出了一种动态自适应排序算法.

3 动态回归测试排序算法设计

3.1 新功能测试用例集排序算法 NT-TCP

新功能测试用例优先级计算过程分层次进行,首先计算需求重要度,再计算模块重要度,最后计算用例优先级.

3.1.1 需求重要度 RI 计算

Step1 需求优先级 RP 计算

需求优先级指的是用户,也即系统实际使用者分配的优先级. 系统功能中大约有 40% 的功能被用户经常使用,其他功能很少被用户使用^[9]. 对于用户经常使用的软件功能要优先进行测试,验证是否存在问题. 需求优先级可以进行线性赋值为 1-5 的数值.

Step2 需求关联值 RCV 计算

测试需求之间相互依赖,与其他需求存在关联关系的需求应该具有更高的测试优先级. 需求的关联值 RCV 的计算过程如下:

首先,根据新功能需求文档,分析需求之间的关系,建立需求功能关联关系^[10]. 然后,根据需求功能关联关系,计算需求关联值:

$$RCV_i = \sum_{i=1}^m 1 \quad (1)$$

m 表示与需求 R_i 有直接关联的需求总数.

Step3 需求重要度 RI 计算

在计算需求重要度时,考虑每个因素赋予不同的权重值 W ,计算需求 R_i 的重要度如下:

$$RI_i = W_1 * RP_i + W_2 * RCV_i \quad (2)$$

式中, $W_1 + W_2 = 1$, W_1 、 W_2 按实际情况进行取值.

3.1.2 模块重要度 MI 计算

Step1 关联需求重要度 MRI 计算

设 $R = \{R_1, R_2 \dots R_n\}$ 和 $M = \{M_1 \dots M_m\}$ 分别代表包含 n 个测试需求的需求集和 m 个模块的模块集. 测试需求和模块之间的映射关系用矩阵 D 来表示,矩阵大小为 $n \times m$,其中:

$$D_{ij} = \begin{cases} 1, \text{模块 } M_j \text{ 覆盖需求 } R_i \\ 0, \text{模块 } M_j \text{ 不覆盖需求 } R_i \end{cases} \quad (3)$$

计算模块 M_j 关联需求重要度:

$$MRI_j = \sum_{i=1}^n D_{ij} * RI_i \quad (4)$$

Step2 覆盖模块的测试用例数量 MCN 计算

模块与测试用例之间的映射关系可能是 1 对 1,也可能是 1 对多,设 $M = \{M_1, M_2 \dots M_m\}$ 和 $NT = \{T_1, T_2 \dots T_k\}$ 分别代表包含 m 个模块的新功能模块集和 k 个测试用例的新功能测试用例集. 建立新功能模块与新功能测试用例集关系矩阵 E ,大小为 $m \times k$,其中:

$$E_{ij} = \begin{cases} 1, \text{用例 } T_j \text{ 覆盖模块 } M_i \\ 0, \text{用例 } T_j \text{ 不覆盖模块 } M_i \end{cases} \quad (5)$$

覆盖模块 M_i 的测试用例数量 MCN 计算方式

如下:

$$MCN_i = \sum_{j=1}^K E_{ij} \quad (6)$$

Step3 模块重要度 MI 计算

在计算模块重要度时,考虑每个因素赋予不同的权重值 W ,计算最终模块 M_i 的重要度如下:

$$MI_i = W_1 * MRI_i + W_2 * MCN_i \quad (7)$$

式中, $W_1 + W_2 = 1$, W_1 、 W_2 按实际情况进行取值。

3.1.3 新功能测试用例排序

Step1 关联模块重要度 TMI 计算

根据模块与测试用例关系矩阵 E 和模块重要 MI ,计算用例 T_j 关联模块重要度 TMI 公式如下:

$$TMI_j = \sum_{i=1}^m E_{ij} * MI_i \quad (8)$$

Step2 用例属性值 TP 计算

测试用例属性包括测试用例类型和测试用例粒度。在测试设计时不仅需要考察功能的正常场景,还需要考虑各种异常场景,即正向用例和反向用例^[11],此为测试用例类型。正向用例需要优先测试,因此可以设置正向用例类型值比反向用例大,例如 2 和 1。根据测试用例所能覆盖的场景数量,测试用例粒度可以分为小型、中型和大型,具体标准可由测试人员根据实际经验设置。测试用例粒度越大,发现故障可能性相对越高,因此可以设置小型、中型和大型的用例粒度值分别为 1、2 和 3。

用例 T_j 属性值 TP_j 计算公式如下:

$$TP_j = TV_j + SV_j \quad (9)$$

式中, TV_j 指的是用例的类型值, SV_j 指的是用例的粒度值。

Step3 用例优先级 $NTCP$ 计算

用例 T_j 的优先级 $NTCP$ 计算公式如下:

$$NTCP_j = W_1 * TMI_j + W_2 * TP_j \quad (10)$$

式中, $W_1 + W_2 = 1$, W_1 、 W_2 按实际情况进行取值。

根据测试用例的 $NTCP$ 值,完成新功能测试用例优先级排序,得到初始的测试序列。

3.2 原功能测试用例集排序算法 OT-TCP

原功能测试用例集排序考虑如下因素:原有排序、受新功能影响以及历史执行信息^[8]。

3.2.1 原功能测试用例影响值 TIV 计算

Step1 原需求影响值 IV 计算

根据新功能需求文档和原功能需求功能关联关系,分析新需求与原需求关系,建立新的需求功能关联关系,原需求 R_i 影响值计算公式为:

$$IV_i = \sum_{k=1}^s 1 \quad (11)$$

s 为与需求 R_i 有关联关系的新需求总和。

Step2 模块影响值 MIV 计算

重新建立原需求与原功能模块关系矩阵 D ,

其中:

$$D_{ij} = \begin{cases} 1, \text{模块 } M_j \text{ 覆盖需求 } R_i \\ 0, \text{模块 } M_j \text{ 不覆盖需求 } R_i \end{cases} \quad (12)$$

根据 D 和需求影响值 IV ,计算原功能模块 M_j 的影响值 MIV :

$$MIV_j = \sum_{i=1}^n D_{ij} * IV_i \quad (13)$$

n 为需求个数。

Step3 测试用例影响值 TIV 计算

重新建立原功能模块与原功能测试用例关系矩阵 E ,其中:

$$E_{ij} = \begin{cases} 1, \text{用例 } T_j \text{ 覆盖模块 } M_i \\ 0, \text{用例 } T_j \text{ 不覆盖模块 } M_i \end{cases} \quad (14)$$

根据 E 和模块影响值 MIV ,计算原功能测试用例 T_j 的影响值 TIV :

$$TIV_j = \sum_{i=1}^m E_{ij} * MIV_i \quad (15)$$

m 为模块个数。

3.2.2 测试用例缺陷检测能力值 FDC 计算

软件缺陷严重性可以分为致命、严重、一般、

轻微等 4 个等级,根据严重性从高到低可以赋值为 4、3、2、1 或 15、10、5、3 等。分析历次回归测试版本中原功能测试用例发现的缺陷数量和等级,得出测试用例的历史缺陷检测能力。假设当前版本为 V_m ,回归测试历经版本集 $V = \{V_1, V_2 \dots V_{m-1}\}$,测试用例 T_j 的缺陷检测能力 FDC 如下计算:

$$FDC_j = \sum_{i=1}^{m-1} DC_{T_{i,j}} \quad (16)$$

$DC_{T_{i,j}}$ 表示测试用例 T_j 在版本 i 中发现的缺陷等级量化值,计算方法为其发现的每种等级缺陷数量乘严重性等级值,再求和。

3.2.3 原功能测试用例排序

计算原功能测试用例 T_j 优先级 $OTCP$ 公式如下:

$$OTCP_j = W_1 * TIV_j + W_2 * FDC_j \quad (17)$$

式中, $W_1 + W_2 = 1$, W_1 、 W_2 按实际情况进行取值。

根据 $OTCP$ 值完成原功能测试用例排序;对于 $OTCP$ 值为 0 的测试用例:首先根据用例归属版本号降序排序,然后每个版本中用例根据保存的 $NTCP$ 值二次排序。

3.3 回归测试算法设计

3.3.1 测试序列的动态调整

在测试用例执行过程中,若某个用例发现系统缺陷,其所在功能模块存在缺陷的风险较大,需要尽快测试该模块相关的测试用例. 因此,须合理地调整覆盖该模块尚未执行测试用例的优先级,以提高缺陷检测效率^[8].

本文采用了一种测试序列动态调整方法,具体步骤如下:

Step1 在测试序列执行过程中,记录测试失败用例 T_i .

Step2 根据 T_i 归属需求的版本号是否为新版本,遍历模块与测试用例关系矩阵 E 或者 E' , 获取 T_i 覆盖的模块列表 $\{M_1, M_2 \cdots M_m\}$.

Step3 遍历模块与测试用例关系矩阵 E 或者 E' , 获取与模块列表 $\{M_1, M_2 \cdots M_m\}$ 有覆盖关系的测试用例集 FT .

Step4 遍历 FT 中的用例,如果尚未执行并且其优先级低于 T_i ,则提高其优先级与 T_i 相同,重新排序未执行序列.

Step5 继续执行余下的测试序列,重复 Step1 至 Step4.

3.3.2 回归测试算法 AD-DTCP 模型

本算法的模型如图 1 所示. 模型依次处理全部版本的用例,输入是当前版本的新功能测试用例集和原功能测试用例集,最终得到一个待执行的已排序测试序列 T^* ,并执行测试序列 T^* .

算法中将 NT 排在 OT 前面生成测试序列 T^* . 待 T^* 执行结束后,将 NT 加入到原测试用例集,并进入下一个版本. 直到所有版本都完成上述过程.

4 实验验证

4.1 实验数据

为了验证本文提出的 AD-DTCP 的有效性,本文使用了谷歌共享的测试结果数据集 GSDTSR 作为实验对象. 部分原始信息如表 1 所示.

表 1 GSDTSR 的部分原始信息

Test Suite	Change Request	Launch Time	Execution Time	Size	Status
199	0	1;00;05;02	13818	MEDIUM	PASSED
209	0	1;00;05;03	17340	LARGE	PASSED
209	0	1;00;05;04	15577	SMALL	PASSED

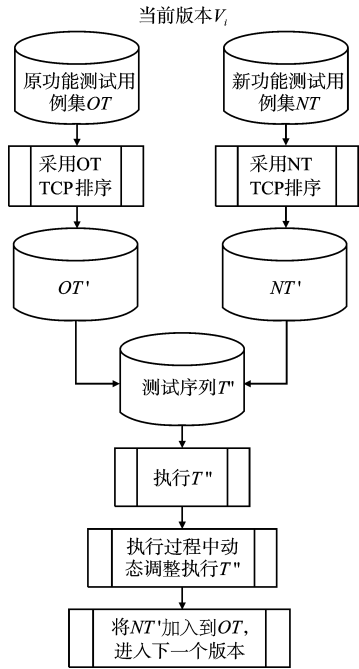


图 1 回归测试算法 AD-DTCP 模型图

其中第一列为测试集 ID, 本文将其处理为模块, 第二列为测试需求 ID, 第五列处理为测试用例粒度, 最后一列为测试用例状态, 也即执行结果, 如果执行结果为 Failed, 表示测试用例发现系统缺陷.

4.2 评价指标

本文采用故障检测平均百分比 $APFD$ 作为度量指标, 验证测试执行序列的有效性. 根据不同算法所生成的测试用例执行序列的 $APFD$ 值越大, 代表其故障检测速率越快.

假设当前版本中测试序列有 k 个测试用例, 共发现 s 个缺陷, 则 $APFD$ 的计算公式如下所示:

$$APFD = 1 - \frac{\sum_{i=1}^s T_i}{ks} + \frac{1}{2k} \quad (18)$$

式中, T_i 代表了首次检测出第 i 个错误的测试用例在该测试序列中的位置.

4.3 实验结果与分析

本文处理实验对象时, 按照数据集 Launch Time 字段划分版本, 采用了两种版本划分粒度: 天和小时, 以每天一个版本共有 17 个版本, 以每小时一个版本共有 347 个版本. 将本文提出的 AD-DTCP 算法与未优化随机排序 Random 算法分别在两种版本粒度上实验. 部分测试结果如表 2、表 3 所示:

表 2 部分版本的 APFD 值(版本粒度为天)

算法	AD-DTCP	Random
版本 1	80.16%	56.05%
版本 5	93.44%	70.46%
版本 8	86.62%	63.44%
版本 13	90.52%	56.28%
版本 16	99.57%	12.33%

表 3 部分版本的 APFD 值(版本粒度为小时)

算法	AD-DTCP	Random
版本 1	83.98%	26.32%
版本 50	97.98%	56.84%
版本 100	99.59%	49.86%
版本 150	99.96%	29.22%
版本 200	99.80%	47.68%
版本 250	98.13%	53.91%
版本 300	99.55%	70.56%
版本 347	97.67%	29.88%

从实验结果可以看出,AD-DTCP 算法相比 Random 算法可以显著提高 APFD 值,部分版本的 APFD 值已经超过了 99%,并且版本粒度为小时时,具有更高的 APFD 值,分析原因为更早使用原功能测试用例排序算法,具有更高价值.

分析 AD-DTCP 算法在版本粒度为小时的结果,剔除版本中没有发现缺陷的情况,平均 APFD 值为 98.08%,这个测试结果超出了王晓琳等^[3]大约 3 个百分点,而 Random 算法的平均 APFD 值仅为 44.33%;而版本粒度为天时,AD-DTCP 算法的评价 APFD 值为 88.47%.

分析以上结果可发现:同一个版本中,本文所提出的动态调整回归测试用例排序方法与 Random 算法相比,具有更高的故障检测平均百分比.本文所提出算法在版本粒度更精细的场景,具有更高的故障检测平均百分比,因此也更适用于版本快速迭代的敏捷开发模式.

5 结束语

本文针对基于敏捷开发模式的回归测试用例排序进行研究,提出动态回归测试排序算法 AD-DTCP,实现了回归测试用例集的排序.实验表明,AD-DTCP 算法比随机排序 Random 算法效果显著,适用于敏捷开发模式.

在实际的测试用例优先排序过程中,测试用例优先级的排序可能会受到较多因素的影响,如用户需求、用例属性等,因而可能需要借助于经验判断对测试用例进行优先级的初始排序.在今后的工作中,需要继续优化算法,并能够在其他的工业数据集中

验证有效性.

参考文献:

[1] LI Y L, WANG Q. Test set optimization in continuous integration: A systematic literature review [J]. Journal of Software, 2018, 29(10): 3021-3050.

[2] CHEN X, CHEN J H, JU X L, et al. Survey of test case prioritization techniques for regression testing [J]. Journal of Software, 2013, 24(8): 1695-1712.

[3] WANG X L, ZENG H W, LIN W W. Techniques for regression testing in agile development environment [J]. Chinese Journal of Computers, 2019, 42(10): 2323-2338.

[4] ROTHERMEL G, UNTCH R H, CHU C, et al. Prioritizing test cases for regression testing [J]. IEEE Transactions on Software Engineering, 2001, 27(10): 929-948.

[5] ELBAUM S, MALISHEVSKY A G, ROTHERMEL G. Prioritizing test cases for regression testing [C] // Proceedings of the 2000 International Symposium on Software Testing and Analysis. New York: ACM, 2000: 102-112.

[6] QU B, NIE C H, XU B W. Test Case Prioritization Based on Test Suite Design Information [J]. Chinese Journal of Computers, 2008, 31(3): 431-439.

[7] ZHANG N, YAO L, BAO X A, et al. Multi-Objective optimization based on-line adjustment strategy of test case prioritization [J]. Journal of Software, 2015, 26(10): 2451-2464.

[8] CHANG L H, MIAO H K, XIAO L. Self-adaptive test case prioritization based on history information [J]. Computer Science, 2015, 42(9): 154-158.

[9] YANG G H, BAO Y, LI D H, et al. Test case prioritization based on requirement [J]. Computer Engineering and Design, 2011, 32(8): 2724-2728.

[10] CHEN J, SHU Q, XIE H F. Priority ranking method of test cases based on fault location [J]. Computer Science, 2019, 46(8): 239-243.

[11] ZHU S M. Software testing [M]. (2nd Edition). Beijing: Posts & Telecom Press, 2016.

作者简介:

张丽 女, (1981), 硕士, 讲师. 研究方向为软件测试、测试用例优化. E-mail: 18913803106@163.com.

戴丽丽 女, (1986), 女, 硕士, 讲师. 研究方向为图像处理.

杜兰 女, (1986), 硕士, 高级工程师. 研究方向为人工智能、区块链.