

基于测试用例设计信息的回归测试优先级算法

屈 波 聂长海 徐宝文

(东南大学计算机科学与工程学院 南京 210096)

(江苏省软件质量研究所 南京 210096)

摘 要 优先级技术是一种高效实用的回归测试技术. 文中针对现有优先级技术未能有效使用测试用例设计信息的不足, 提出了一组新的回归测试优先级动态调整算法. 与已有方法相比, 新算法充分考虑了测试用例的设计信息, 能够通过及时捕捉和利用测试执行信息对测试用例优先级进行动态调整, 具有时间复杂度低、检错效率高等优点. 将其应用于 Windows 平台下应用程序的回归测试结果表明, 新算法有益于在短时间内检测出更多的错误.

关键词 回归测试; 优先级技术; 算法; 软件工程

中图法分类号 TP311

Test Case Prioritization Based on Test Suite Design Information

QU Bo NIE Chang-Hai XU Bao-Wen

(School of Computer Science and Engineering, Southeast University, Nanjing 210096)

(Jiangsu Institute of Software Quality, Nanjing 210096)

Abstract Test case prioritization is an effective and practical technique of regression testing. This paper proposes a set of new prioritization algorithms based on test suite design information which has not been used to calculate selection probabilities of test cases in previous work. Compared with existing algorithms, the new algorithms can obtain and make use of run-time information to calculate and adjust test case prioritization in time, and are more efficient in detecting faults with low time complexity. The empirical studies performed under Windows platform show that new algorithms are helpful to detect more regression faults in shorter time.

Keywords regression testing; test case prioritization; algorithm; software engineering

1 引 言

优先级技术作为一种致力于提高测试用例使用效率的回归测试技术, 充分考虑了各种因素造成的测试用例重要程度的不同, 为每个测试用例赋予一个选择优先级, 按照优先级次序选择并执行测试用例, 从而达到较高的检错效率.

目前关于优先级技术的研究主要集中在根据测

试历史计算测试用例的选择优先级方面. 一开始, 人们通过测试用例的累计覆盖率来辅助解决回归测试的执行策略问题^[1], 但这种方法考虑的测试历史因素过于单一. 为此, 研究人员一方面以单次测试历史作为基础, 从语句覆盖、分支覆盖和检错能力等方面提出了多种优先级算法^[2-3], 另一方面指出应将回归测试视为有序的行为序列, 并据此提出了综合历次回归测试结果的优先级算法^[4]. 实验研究结果表明, 上述各种算法都能有效地提高回归测试效率^[5].

虽然已有算法对程序本身和测试历史进行了有效的分析和利用,但却忽略了设计目的等测试用例的设计信息,未能在测试过程中对其进行有效利用,从而影响了回归测试优先级算法的效率.为此,本文在提出通过关联矩阵来表示测试用例的设计信息的基础上,给出了一组测试用例优先级的动态调整算法 RTE、RTD 与 MIX,并严格证明了它们的一些优良特性,最后通过实验表明,这些算法比已有算法具有更高的效率.

2 研究背景

回归测试同普通测试的主要区别在于有无可复用的测试用例集,如何有效地复用测试用例因而成为回归测试主要的研究方向^[6-10]. Wong 等在深入研究了测试用例的选择和执行问题后指出,为了提高测试用例的使用效率,可以采用优先级技术,为每个被选中复用的测试用例赋予一个优先级,在回归测试过程中按照优先级顺序来选择执行相应的测试用例^[1].例如,当待测软件中所包含某类错误比较多时,优先运行可以检测这些错误的测试用例就有助于提高测试用例集的使用效率.为了达到这一目的,可以在测试过程中采用启发式方法来预测哪些测试用例应该被优先执行.

围绕回归测试的优先级问题已经有大量的研究工作. Wong 等最先提出了在回归测试选择技术基础上对测试用例集进行最小化或优先级处理,通过判定累计覆盖率等因素对测试用例进行排序,辅助选择使用测试用例^[1].在此基础上, Rothermel 等进行了针对性的实验研究,证实优先级技术能够有效地提高检错效率^[5]. Elbaum 和 Rothermel 等基于测试用例执行信息,将测试历史及动态反馈信息用于计算测试用例选择概率^[2-3]. Kim 等在已有回归测试研究的基础上指出应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^[4].此外, Srivastava 等提出了有效地分析二进制文件的回归测试优先级技术并实现了相关工具^[11]. Jones 等研究了面向 MC/DC 的回归测试优先级技术^[12]. Elbaum 等研究了如何针对不同场景选择有效的优先级技术^[13]. Srikanth 研究了基于需求的优先级技术^[14]. Walcott 等则研究了与时间因素相关优先级技术^[15].

现有的回归测试优先级算法均将测试历史和程序分析结果作为分析和计算基础,忽视了测试用例

设计信息对测试用例优先级带来的影响.另外,少量算法考虑动态反馈信息来及时调整测试用例的执行顺序,以使测试用例集更好地适应当前测试环境,但需要额外收集测试用例的执行信息,时间复杂度偏高.若将测试用例设计信息用于指导动态调整优先级,将能保留动态调整的优点,同时避免额外的开销.

例如,在图 1 中, (a) 和 (b) 分别是初始代码和在此基础上增加了求商的运算后的代码,针对它们所设计的测试用例 (c) 覆盖了代码中各个路径,其中测试用例 t_1 和 t_3 具有相同的设计目的,均用于测试变量 a 的边界值 ($a=0$). 按照历史覆盖率进行优先级赋值后,各个测试用例的执行优先级次序是 $t_1 > t_4 > t_3 > t_2$. 在测试过程中可以发现,选择具有最高优先级的 t_1 进行测试就可以检测到错误,此时若不进行调整,将会选择执行两个测试用例后才能再次检测到错误,若根据 t_1 的执行结果立即提高具有相同设计目的的 t_3 的优先级,剩余测试用例执行次序会被调整为 $t_3 > t_4 > t_2$, 其结果是 t_3 将被优先执行并检测到错误.可见,与固定优先级相比,这种调整有利于尽早发现和定位错误.此外,这种调整的依据来自测试用例设计时的信息,不需要使用诸如文献^[2-3]所述调整方法所需的语句执行信息,因此无须在运行时收集额外的信息,提高了算法效率.

<pre> if (a<=0){ a=a*2; } if (b<0){ b=-b; b=b+1; } c=b*a; </pre>	<pre> if (a<=0){ a=a*2; } if (b<0){ b=-b; b=b+1; } c=b*a; d=b/a; </pre>	<pre> t1: a=0, b=-1 t2: a=1, b=1 t3: a=0, b=1 t4: a=1, b=-1 </pre>
(a)	(b)	(c)

图 1 修改前后代码及测试用例

3 基于测试用例设计信息的优先级算法

测试用例设计信息对于测试用例优先级的计算具有重要的参考价值,因此我们提出了基于测试用例设计信息的回归测试优先级动态调整算法.本节首先基于测试用例设计信息定义了面向回归测试的测试用例相似关系和关联矩阵,然后在此基础上给出一组新的优先级算法,最后将对新算法的一些特性进行证明.

3.1 相似关系与关联矩阵

软件测试目标在测试过程开始前已经制定完

毕,测试人员可根据测试目标中不同的测试需求来设计测试用例^[16].测试用例中的设计信息中除了设计者、日期等基本内容外,还应包括预期对测试需求的覆盖情况,这部分内容通常被称作测试用例的设计目的.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.在回归测试过程中及时捕获并利用这些信息,有利于提高查找发现错误的效率,为后续故障定位等工作提供更多的支持.

不失一般性,假设每个测试用例只覆盖一个测试需求,且不同需求间没有交叉部分.不同测试用例若被设计为预期覆盖相同的测试需求,则称其具有相同的设计目的.

定义 1. 在测试用例集 $T = \{t_1, t_2, \dots, t_n\}$ 中,如果 $t_i, t_j \in T$ 且 t_i 和 t_j 具有相同的设计目的,则称 t_i 和 t_j 相似,记作 $t_i \sim t_j$.

相似关系描述了回归测试的测试用例集中不同测试用例在设计目的上的关联情况.可见,这种相似关系是等价关系,等价类集合 $\{[t] \mid t \in T\}$ 将测试用例集 T 按照不同的设计目的进行了一个划分,由于设计目的相同,处于同一个子集的不同测试用例往往会检测出相同类型的错误,因此一个测试用例的执行结果通常对同一子集的测试用例具有指导意义.为了方便在优先级算法中使用这种关系提供的信息,下面将在此基础上定义测试用例集的关联矩阵.

定义 2. 测试用例集 $T = \{t_1, t_2, \dots, t_n\}$ 的关联矩阵是一 n 阶的布尔方阵 $MRS = (k_{i,j})_{n \times n}$,其中,元素 $k_{i,j} = 1$ 当且仅当 $t_i \sim t_j$,否则 $k_{i,j} = 0$.

关联矩阵 MRS 的构造方法比较简单,只需针对每个测试用例遍历比较所有其它测试用例的设计目的,并设置相对应矩阵元素的取值.其构造算法的时间复杂度不超过 $O(n^2)$.

使用关联矩阵 MSR 的好处在于:首先,测试用例的设计目的容易获取,同时适用于源代码可见和源代码不可见的情况;其次,生成关联矩阵 MSR 的代价小,在测试用例设计完毕后用不超过 $O(n^2)$ 时间即可构造,且在后续回归测试过程中不需重复计算;最后,生成和使用关联矩阵 MSR 无需收集覆盖率等额外的执行信息,与程序规模无关.在下面的优先级算法中,将把关联矩阵 MSR 作为测试用例优先级计算和调整的依据和基础.

3.2 优先级动态调整算法

在以上定义基础上,就可以进一步讨论相应的测试算法.为提高检错效率,可以在回归测试一开始根据测试历史按照一定规则(如前期测试的检错率)为

每一测试用例赋予一初始优先级,并在测试过程中根据测试反馈信息和测试用例设计信息对其作动态调整,使测试用例的执行次序更适应当前的测试情况.

如所知,当某一测试用例检测到错误时,具有相同设计目的其它测试用例一般也有可能检测到相同或类似的错误.据此,就可以在用某个测试用例检测到错误时立即调整(提高)尚未使用的具有相同测试目的的测试用例的优先级,从而使这些测试用例很快得到使用.这样,在执行了一定量的测试用例并对有关测试用例的优先级作了相应调整后,可以检测出较多错误的测试用例就会被提升到较高的优先级,从而使得整体的检错效率得到提高.同时,调整后的测试用例集对后续的回归测试也有着重要的参考作用.

因此,我们可以按照这样的步骤来进行回归测试并在测试过程中动态调整测试用例:首先,根据测试历史对测试用例集进行初步的排序,这里可以采用各种方法,例如,可先将所有测试用例赋予相同的优先级,也可根据历史检错情况给各测试用例赋予相应的优先级.然后,选择并执行当前具有最高优先级的测试用例,当检测到输出与预期不符的情况(即检测出错误)时,顺序遍历未执行测试用例,动态地将具有相同设计目的的测试用例的优先级提高一级,再根据新优先级顺序选择执行测试用例,如此重复,直到所有测试用例都执行完毕.根据以上分析,可以得到如算法 1 所示的优先级动态提高算法 RTE (Run-Time Escalate).

算法 1. 优先级动态提高算法 RTE.

输入: T (测试用例集), R (测试用例的预期结果), MRS (近似关联矩阵)

输出: T' (检测出错误的测试用例集)

```
Sort(T);           //按照测试用例的历史检错情况
                    //进行初步排序
for i=1 to n       //按照优先级选择测试用例
     $R'_i = Run(t_i)$ ; //运行当前具有最高优先级的  $t_i$  并
                    //获得运行结果  $R'_i$ 
    if  $R_i \neq R'_i$  then //实际输出与预期不同,检测出错误
        Output( $t_i$ ); //输出检测出错误的测试用例
        for j=i+2 to n //  $t_{i+1}$  的优先级无需作提高处理
            if  $MRS[i, j] = 1$  and  $MRS[i, j-1] = 0$  then
                交换  $t_j$  与  $t_{j-1}$  的优先级;
            endif
        endfor
    endif
endfor              //顺序遍历剩余测试用例结束
endif
endif
endif
```

当使用一个测试用例时,既可能检测到错误也可能检测不到错误. 如果在使用一个按优先级排序的给定测试用例集时所能检测到错误的测试用例比较少,那么这时就不能采用 RTE 算法,因为这样会在较长时间内不对未执行测试用例优先级进行任何调整. 一般而言,当某一测试用例没有检测到错误时,具有相同设计目的的其它测试用例一般也有可能检测不到相同或类似的错误,因此,当运用一个测试用例检测不到错误时可以将具有相同设计目的的其它测试用例的优先级降低,以便使得其它测试用例优先得到执行. 这样就可以得到与 RTE 算法相对应的另一个算法——优先级动态降低算法 RTD (Run-Time De-escalate),如算法 2 所示. 其基本思想是,在执行当前具有最高优先级的测试用例时,如果输出与预期相符(即检测不到错误),那么就逆序遍历未执行测试用例,动态将具有相同设计目的测试用例的优先级降低一级,然后再根据新优先级顺序选择执行测试用例,如此重复,直到所有测试用例都执行完毕. 若大量测试用例执行时产生的输出与预期输出一致,这种方法会执行多次调整,因而 RTD 算法适合于在错误分布较为稀少的情况下快速提高测试用例集的使用效率.

算法 2. 优先级动态降低算法 RTD.

输入: T (测试用例集), R (测试用例的预期结果), MRS (近似关联矩阵)

输出: T' (检测出错误的测试用例集)

```
Sort( $T$ );           //按照测试用例的历史检错情况
                    //进行初步排序
for  $i=1$  to  $n$        //按照优先级选择测试用例
     $R'_i=Run(t_i)$ ;  //运行当前具有最高优先级的  $t_i$  并
                    //获得运行结果  $R'_i$ 
    if  $R_i \neq R'_i$  then //检测出错误
        Output( $t_i$ ); //输出检测出错误的测试用例
    else             //实际输出与预期输出相同
        for  $j=n-1$  to  $i+1$ 
            //  $t_n$  的优先级无需作降低处理
            if  $MRS[i, j]=1$  and  $MRS[i, j+1]=0$  then
                交换  $t_j$  与  $t_{j-1}$  的优先级;
            endif
        endfor      //逆序遍历所有测试用例结束
    endif
endfor
```

由以上分析可知,在使用一个已按优先级排序的测试用例集时,若检测到错误测试用例较多,算法 RTE 能获得较好的效果,反之,则使用算法 RTD 的效率较高. 在实际测试过程中,有时候不能确定一给

定测试用例集中,检测到错误的测试用例的大致比例. 这时采用前面两种算法理论上也能提高测试效率,但为了能及时根据测试结果调整未执行的测试用例,可以考虑将上述两种调整过程结合起来. 这样可以得到如算法 3 所示的测试用例优先级混合算法 MIX. 这种算法在每执行一个测试用例后都会对剩余测试用例进行调整.

算法 3. 优先级混合算法 MIX.

输入: T (测试用例集), R (测试用例的预期结果), MRS (近似关联矩阵)

输出: T' (检测出错误的测试用例集)

```
Sort( $T$ );           //按照测试用例的历史检错情况
                    //进行初步排序
for  $i=1$  to  $n$        //按照优先级选择测试用例
     $R'_i=Run(t_i)$ ;  //运行当前具有最高优先级的  $t_i$  并
                    //获得运行结果  $R'_i$ 
    if  $R_i \neq R'_i$  then //实际输出与预期输出不同,检测
                        //出错误
        Output( $t_i$ ); //输出检测出错误的测试用例
        for  $j=i+2$  to  $n$  //遍历剩余测试用例
            if  $MRS[i, j]=1$  and  $MRS[i, j-1]=0$  then
                交换  $t_j$  与  $t_{j-1}$  的优先级;
            endif
        endfor
    else             //实际输出与预期输出相同
        for  $j=n-1$  to  $i+1$  //逆序遍历剩余测试用例
            if  $MRS[i, j]=1$  and  $MRS[i, j+1]=0$  then
                交换  $t_j$  与  $t_{j-1}$  的优先级;
            endif
        endfor
    endif
endfor
```

3.3 算法性质分析

与已有算法相比,新算法具有一些优良的特性,例如与程序规模无关,不会改变具有相似关系的测试用例子集的有序性等. 下面是对这些特性的描述和证明.

研究表明,根据运行的动态反馈信息对剩余测试用例的执行顺序进行调整能有效提高检错效率,但已有的算法需要在测试用例运行时收集程序中每条语句的相关信息,算法复杂度较高且与程序规模直接相关. 新算法的优势在于它更多地考虑了反馈信息,有利于不断优化测试的进行,同原有的算法相比复杂度较低,且与待测程序的规模无关.

定理 1. 算法 RTE、RTD 和 MIX 复杂度与程序规模无关.

证明. 算法分为两个部分,第一部分是初步排序,第二部分是选择运行和动态调整.设程序规模为 m ,测试用例数为 n ,对于第一部分,采用各种排序算法的最坏时间复杂度不超过 $O(n^2)$.对于第二部分,算法 RTE、RTD 和 MIX 的外层循环选择执行测试用例,共进行 n 次,内层循环根据结果遍历调整剩余测试用例,复杂度为 $O(n)$,故第二部分总体时间复杂度为 $O(n^2)$.

综合对两个部分的分析,算法 RTE、RTD 和 MIX 的总体时间复杂度为 $O(n^2 + n^2)$,即 $O(n^2)$,与程序规模 m 无关. 证毕.

定理 1 意味着在测试过程中无需通过插桩等手段收集额外信息,算法也不会随着待测程序规模的增大而耗费更多的时间.在时间复杂度上,新算法明显优于采用了类似动态调整思想的其它算法,例如累计语句覆盖率(stmt-addtl,复杂度为 $O(mn^2)$)等^[3].

此外,算法 RTE、RTD 和 MIX 虽然在测试过程中不断调整了剩余测试用例的执行顺序,但不会破坏其中一些子序列的顺序结构.这将保证具有相同测试目的的测试用例子集中,优先级高的测试用例始终会优先执行.

引理 1. 算法 RTE、RTD 和 MIX 不会改变相关联的两个测试用例之间的顺序.

证明. 设有测试用例 t_i 和 t_j ,其中 $i < j$ 且 $t_i \sim t_j$.若运行的测试用例为 t_k ,当 $k \geq i$ 时, t_i 已运行,故 t_i 和 t_j 顺序不会改变,这里仅分情况讨论各种算法 $k < i$ 时, t_i 和 t_j 调整后的顺序.

(1) $R_k = R'_k$ 时的情形

对于算法 RTE, t_k 输出与预期相符时不对测试用例进行任何调整,因此 t_i 和 t_j 的顺序未改变.

对于算法 RTD:

若 $MSR[k,i]=1$,即 $t_k \sim t_i$,因为 $t_i \sim t_j$,所以 $t_k \sim t_j$,即 $MSR[k,j]=1$, t_i 和 t_j 未作任何调整.

若 $MSR[k,i]=0$,因为 $t_i \sim t_j$,所以有 $MSR[k,j]=0$. $i=j-1$ 时, t_j 不作任何调整; $i < j-1$ 时, t_j 至多被调整至 t_{i-1} ,调整后依然有 $t_i < t_j$.

对于算法 MIX, t_k 输出与预期不符时情况同算法 RTD.

(2) $R_k \neq R'_k$ 时的情形

对于算法 RTD, t_k 输出与预期不符时不对测试用例进行任何调整,因此 t_i 和 t_j 的顺序未改变.

对于算法 RTE:

若 $MSR[k,i]=0$,因为 $t_i \sim t_j$,所以有 $MSR[k,j]=0$, t_i 和 t_j 未作任何调整.

若 $MSR[k,i]=1$,即 $t_k \sim t_i$,因为 $t_i \sim t_j$,所以 $t_k \sim t_j$,即 $MSR[k,j]=1$. $i=j-1$ 时, t_i 不作任何调整; $i < j-1$ 时, t_i 至多被调整至 t_{j-1} ,调整后依然有 $t_i < t_j$.

对于算法 MIX, t_k 输出与预期不符时情况同算法 RTE.

综上,算法 RTE、RTD 和 MIX 不会改变相关联的两个测试用例之间的顺序. 证毕.

定理 2. 动态优先级算法 RTE、RTD 和 MIX 不会改变有关联测试用例子集的有序性.

证明. 设 $T' = \{t_{i1}, t_{i2}, t_{i3}, \dots, t_{in}\}$ 是一个相关联的测试用例子集.由引理 1 可知对于其中任意两个测试用例 t_{im} 和 t_{in} ,算法 RTE、RTD 和 MIX 均不会改变其顺序,因此在应用动态优先级算法后, T' 中测试用例顺序未变. 证毕.

4 实验分析

上面已经证明了算法 1~3 具有一些优良的特性,但同其它优先级算法一样,其本质上仍然是启发式方法,算法的有效性应该通过实验来验证.在研究过程中,我们开发了可以自动对 Office 系列软件进行测试和记录的工具 OAR,并在此基础上设计和进行了对比实验,用以检测三种新算法在实际应用中的性能表现.本节将对实验数据进行描述和分析.

4.1 实验设计

实验选用运行于 Windows 平台下的 PowerPoint 2003(11.8024.8036)作为测试对象,重点检测其对畸形文件格式的处理情况.原始测试用例集是在 PowerPoint 2000 生成的正常文件基础上,根据 6 种不同的测试用例设计目的(表 1),通过改变文件内容中对应字段的数值来生成的.我们在对 PowerPoint 2000、PowerPoint XP 进行初步测试后,筛选出原始测试用例集中检查到错误的部分作为待复用的测试用例集,除此以外还补充了一些新的测试用例用以检测软件新增的功能,并且设定新增测试用例历史检错个数为零.

最终的测试用例集如表 1 所示,其中不同的测试用例设计目的之间是互相独立的,因此不同测试目的对应的测试用例没有交叉,即一个测试用例仅对应一个测试目的.

表 1 测试用例设计目的及规模

测试用例来源	零地址引用	异常指针引用	缓冲区溢出	异常控件操作	异常分支处理	异常偏移值引用
复用测试用例	289	126	98	151	95	177
补充测试用例	120	120	120	120	120	120
总计	409	246	218	271	215	297

由于前文已经证明新算法时间复杂度优于其它动态调整算法,因此这里选用了具有相同时间复杂度的基于历史检错数的优先级算法(以下简称 HFB)^[4]作为对比.为了能够获得充分的实验数据,测试用例在同一台机器上按照不同的算法对目标软件进行了 4 次完整的测试,算法分别对应 HFB、RTE、RTD 和 MIX.调整算法所用到的关系矩阵根据表 1 所示测试用例设计目的生成.在实验数据基

础上我们比较了采用不同算法进行测试所耗费的时间,检测各种错误的比例以及检测错误的效率.

4.2 实验结果与分析

实验结果显示,在全部 1656 个测试用例中,共有 357 个测试用例检测出错误,约占总数的 21.56%.表 2 描述了检测出的各种类型错误个数,表 3 记录了 4 次测试过程所耗费的时间.

表 2 检出的错误类型与个数

测试用例来源	零地址引用	异常指针引用	缓冲区溢出	异常控件操作	异常分支处理	异常偏移值引用
复用测试用例	101	19	2	55	24	50
补充测试用例	41	15	7	15	16	12
总计	142	34	9	70	40	62

表 3 测试运行时间

测试策略	耗费时间/s
基于历史检错率	34310
动态提高算法(RTE)	34991
动态降低算法(RTD)	35277
混合算法(MIX)	36017

测试结果表明,未检测到错误的测试用例个数多于检测到错误的测试用例个数,相应地,算法 RTD 耗费的时间略多于算法 RTE 耗费的时间,而每次都进行调整的算法 MIX 耗费时间又多于算法 RTD 和 RTE 耗费的时间.与 HFB 算法相比较,采用优先级算法 RTE、RTD 和 MIX 的测试运行时间仅高出 1.98%,2.82%和 4.98%,这主要是因为动态调整算法与 HFB 算法具有相同的时间复杂度,同时测试用例运行时间远大于对优先级进行调整所耗费的时间,动态调整带来的额外开销所占比例较小.结果显示,采用动态调整算法 RTE、RTD 和 MIX 不会显著地提高测试耗费的时间.

以测试进行到总运行时间一半的时刻作为采样点,不同算法检测出各种错误情况如图 2 所示.图示数据依次是测试用例对应的设计目的、当前检测出错误测试用例个数以及所占当前检测出错误总数的比例.

从图 2 中可以看出, RTE、RTD 和 MIX 这 3 个算法检测出各种错误的比例非常接近,而同 HFB 算法相比,采用动态调整的方法基本保持了类似的检错比例.对特定类别的错误分析后可以发现,算法

RTE、RTD 和 MIX 检测出了更高比例的零地址引用错误,而缓冲区溢出和异常指针引用的检错比例则较低.在分析后我们了解到,造成这种结果的原因是算法进行动态调整的过程中,测试目的为零地址引用的测试用例被大量调整到高优先级,而测试缓冲区溢出和异常指针引用的测试用例被调整至相对较低的优先级.若以同类错误在全部错误中所占比例作为错误重要性的度量标准,采用算法 RTE、RTD 和 MIX 将有益于更多地检测出重要的错误,同时有利于及时补充相应的测试用例,对频繁出现的错误类型进行更完整的测试.

设定 1000 s 为取样时间间隔,以测试运行时间作为坐标横轴,以检测出错误的测试用例个数作为坐标纵轴,可以得到如图 3 所示的检测错误情况.

从整体上看,采用 RTE、RTD 和 MIX 算法的测试过程在检错情况上没有明显的差异,每次都进行调整的 MIX 算法与 RTE、RTD 具有类似的检错效率.将每次测试运行时间和检错情况进行比较可以看到 3 个不同的阶段:第一阶段从测试开始到第 10 个千秒,4 个不同测试过程性能表现大体相当;第二阶段从第 10 个千秒到第 33 个千秒,采用 RTE、RTD 和 MIX 算法的过程调整并逐步适应了测试环境,比 HFB 算法具有更好的检错效率;第三阶段从第 33 个千秒到结束,这个阶段剩余的错误较少,4 个不同测试过程先后结束了对测试用例的运行,检测错误数量近似.

为了更好地度量优先级方法,比较采用不同优先级算法的测试效率,Rothermel 在文献[3]中提出了错误检测加权平均百分比的度量标准 (APFD). 令执行测试用例运行情况作为横坐标,测试用例检错情况作为纵坐标,优先级方法的 APFD

值可通过实际检错曲线计算出来. 在图 3 基础上,可得到 RTE、RTD、MIX 算法的 APFD 近似值约为 76%,而 HFB 算法的 APFD 近似值约为 69%,采用算法 RTE、RTD 和 MIX 情况下,相同测试用例集的测试效率约有 10%的提高.

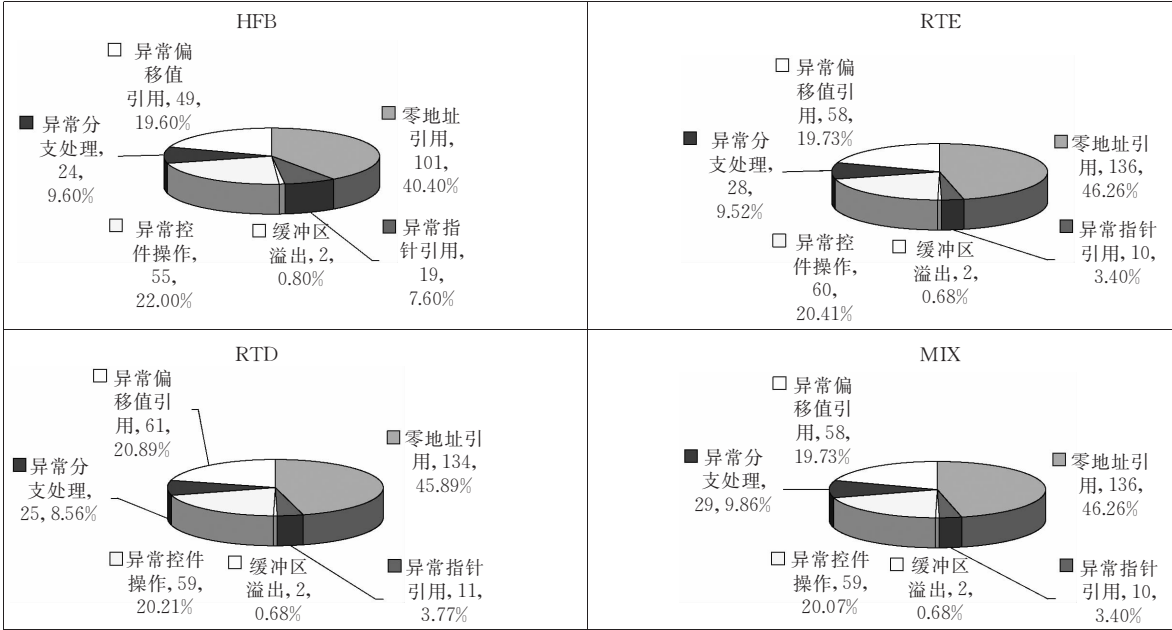


图 2 不同过程检测各类错误比例

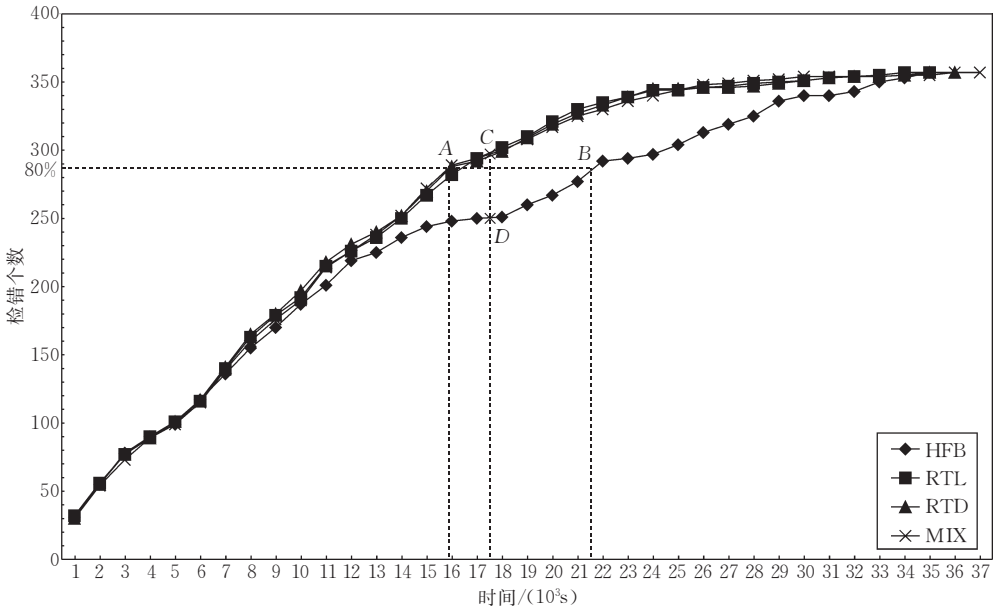


图 3 错误检测图

在实际进行回归测试时,为了保证测试的充分性,测试过程一般不会很快地结束,同时由于时间和资源的限制,测试又往往不能运行完所有的测试用例. 若实际测试过程结束于第二阶段,即实验中第 10 个千秒到第 33 个千秒时,采用动态调整的方法会得到比 HFB 算法更高的测试效率. 例如,以检测

出全部错误(357 个)的 80%(约 286 个)为标准, HFB 算法大约花费了 21000 s(B),而采用动态调整的方法仅需花费约 16000 s(A),节省了大约 27%的时间;以完全测试所需时间的 50%为标准, HFB 算法大约检测了 70%的错误(D),而采用动态调整的方法大约检测了 83%的错误(C),检错数量高出约 20%.

5 结束语

回归测试过程中,如何高效地复用测试用例是回归测试研究中的关键课题.近年来,我们在测试用例设计、自动生成和优化等方面进行了系统、深入的研究^[17-20],本文提出的基于测试用例设计信息的优先级动态调整算法,是对回归测试优先级技术一种新的探索.它弥补了已有技术未能有效利用测试用例设计信息的不足,在与程序规模无关的复杂度上,通过收集运行信息并不断调整测试用例的优先级,逐步优化并使其适应当前的测试环境,从而获得更好的检错效果.实验表明其检错效率与已有算法相比具有一定的优势.

本文给出了完整有效的算法并进行了相关实验比较,但围绕相关内容,仍有部分问题有待进一步的研究:

(1) 关联矩阵用布尔值反映测试用例间的相似关系,但当存在某些测试用例对应多个测试目的,或者测试目的之间存在交叉与重叠时,这种描述方式将会遇到一些困难.因此,后续工作的一个重要方向是研究关联矩阵改进的描述和生成方法.

(2) 在具体的测试过程中,基于测试用例设计信息的三种算法仅对测试用例使用顺序进行了调整,没有涉及具体优先级数值的运算.后续研究可以针对测试用例具体的选择概率,从更细的粒度上对优先级进行调整以获得更好的测试效果.

(3) 本文通过算法及相关实验研究了优先级技术在提高回归测试检错效率上的应用,如何通过类似算法在有限资源内达到更高的代码覆盖率,尚需进一步的理论和实验研究.

参 考 文 献

- [1] Wong W E, Horgan J R, London S, Agrawal H. A study of effective regression testing in practice//Proceedings of the 8th IEEE International Symposium on Software Reliability Engineering. Albuquerque, New Mexico, 1997: 264-274
- [2] Elbaum S, Malishevsky A G, Rothermel G. Prioritizing test cases for regression testing//Proceedings of the International Symposium on Software Testing and Analysis. Portland, Oregon, 2000: 102-112
- [3] Rothermel G, Untch R H, Chu Cheng-Yun, Harrold M J. Prioritizing test cases for regression testing. IEEE Transactions on Software Engineering, 2001, 27(10): 929-948
- [4] 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, Florida, 2002: 119-129
- [5] Rothermel G, Untch R H, Chu Chengyun, Harrold M J. Test case prioritization: An empirical study//Proceedings of the International Conference on Software Maintenance. Oxford, England, 1999: 179-188
- [6] Rothermel G, Harrold M J. Analyzing regression test selection techniques. IEEE Transactions on Software Engineering, 1996, 22(8): 529-551
- [7] Gupta R, Harrold M J, Soffa M L. An approach to regression testing using slicing//Proceedings of the Conference on Software Maintenance. Orlando, Florida, 1992: 299-308
- [8] Bates S, Horwitz S. Incremental program testing using program dependence graphs//Proceedings of the 20th ACM Symposium on Principles of Programming Languages. New York, 1993: 384-396
- [9] Blinkley D. Reducing the cost of regression testing by semantics guided test case selection//Proceedings of the Conference on Software Maintenance. Opio (Nice), France, 1995: 251-260
- [10] Rothermel G, Harrold M J. A safe, efficient regression test selection technique. ACM Transactions on Software Engineering and Methodology, 1997, 6(2): 173-210
- [11] Srivastava A, Thiagarajan J. Effectively prioritizing tests in development environment//Proceedings of the International Symposium on Software Testing and Analysis. Rome, Italy, 2002: 97-106
- [12] Jones J A, Harrold M J. Test-suite reduction and prioritization for modified condition/decision coverage//Proceedings of the International Conference on Software Maintenance. Florence, Italy, 2001: 92-101
- [13] 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
- [14] Srikanth H. Requirements-based test case prioritization//Proceedings of the Student Research Forum in 12th ACM SIGSOFT International Symposium on the Foundations of Software Engineering. Newport Beach, California, 2004
- [15] Walcott K R, Soffa M L, Kapfhammer G M, Roos R S. Time-aware test suite prioritization//Proceedings of the International Symposium on Software Testing and Analysis. Portland, Maine, 2006: 1-12
- [16] Myers G J. Software Reliability: Principles and Practice. Wiley: New York, 1976
- [17] Nie Chang-Hai, Xu Bao-Wen. A minimal test suite generation method. Chinese Journal of Computers, 2003, 26(12): 1690-1695(in Chinese)

(聂长海,徐宝文.一种最小测试用例集生成方法.计算机学报,2003,26(12):1690-1695)

[18]

Nie Chang-Hai and Xu Bao-Wen. An algorithm for automatically generating black-box test cases based interface parameters. Chinese Journal of Computers, 2004, 27(3): 382-388 (in Chinese)
(聂长海,徐宝文. 基于接口参数的黑箱测试用例自动生成算法. 计算机学报, 2004, 27(3): 382-388)

[19]

Xu Bao-Wen, Nie Chang-Hai, Shi Liang, Chen Huo-Wang. A software failure debugging method based on combinatorial design approach for testing. Chinese Journal of Computers, 2006, 29(1): 132-138(in Chinese)

[20]

Nie Chang-Hai, Xu Bao-Wen, Shi Liang. A new pairwise covering test data generation algorithm for the system with many 2-level factors. Chinese Journal of Computers, 2006, 29(6): 841-848(in Chinese)
(聂长海,徐宝文,史亮. 一种新的二水平多因素系统两两组合覆盖测试数据生成算法. 计算机学报, 2006, 29(6): 841-848)



QU Bo, born in 1981, Ph. D. candidate. His main research interests include software analysis and testing.

NIE Chang-Hai, born in 1971, Ph. D., associate professor. His research interests include software engineering and software testing.

XU Bao-Wen, born in 1961, Ph. D., professor, Ph. D. supervisor. His research interests include programming language, software engineering, parallel and network software.

Background

This work is supported in part by the National Natural Science Foundation of China (60425206, 60773104 and 60633010,60503033), Excellent Talent Foundation on Teaching and Research of Southeast University, Open Foundation of State Key Laboratory of Software Engineering in Wuhan University and Doctor Subject Fund of Education Ministry (20060286020). The research work reported in this paper focuses on test case prioritization for regression testing.

Test case prioritization is an effective and practical technique that helps to increase the effectiveness of test cases at meeting some performance goals when software evolves. It schedules test cases so that those with the highest priority, according to some criterion, are executed earlier in the regression testing process than lower priority test cases. Exist-

ing prioritization approaches mainly focus on sorting test cases based on their coverage information and execution history. And empirical studies indicate that test case prioritization techniques could increase the effectiveness of test cases especially in resource constrained testing environment. There are numerous techniques have been reported in literatures. However, none of these prioritization schemes explicitly consider the test suite design information. This paper presents a novel prioritization technique and corresponding algorithms. Comparing with existing techniques, the method could resort test cases according to the test suite design information and dynamic feed back. The experiments also shows that the new algorithms are helpful to detect more regression faults in shorter time.