软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2015,26(8):1871−1885 [doi: 10.13328/j.cnki.jos.004711]
©中国科学院软件研究所版权所有.
E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

基于事件处理函数的 GUI 测试用例集约简技术*

陈军成[1,2,3], 薛云志[1,2], 陶秋铭[1,2], 赵 琛[1,2]

[1](中国科学院 软件研究所 基础软件测评实验室,北京 100190)

[2](基础软件国家工程研究中心(中国科学院 软件研究所),北京 100190)

[3](中国科学院大学,北京 100049)

通讯作者: 陈军成, E-mail: juncheng@nfs.iscas.ac.cn

摘 要: GUI测试用例集约简是降低GUI软件测试成本的有效手段.GUI软件的消息循环机制以及事件驱动特性,导致传统的基于控制流和数据流的测试用例集约简技术难以直接应用于 GUI 测试用例集约简.如何在尽可能保持原有测试用例集缺陷发现能力的基础上,尽可能地降低 GUI 测试用例集规模,是 GUI 测试用例集约简的一个挑战.以事件处理函数为核心,结合控制流和数据流技术,根据事件处理函数代码结构特征以及事件处理函数之间的数据依赖关系定义测试冗余规则,制定并实现了3种测试用例集约简技术.实验结果表明:与已有技术相比,其中两种根据事件处理函数之间的数据依赖关系制定的测试用例集约简技术达到了较好的约简效果.

关键词: GUI 测试用例;测试用例集约简;事件处理函数;定义-引用;冗余测试用例

中图法分类号: TP311

中文引用格式: 陈军成,薛云志,陶秋铭,赵琛.基于事件处理函数的 GUI 测试用例集约简技术.软件学报,2015,26(8):1871−1885. http://www.jos.org.cn/1000-9825/4711.htm

英文引用格式: Chen JC, Xue YZ, Tao QM, Zhao C. GUI test suite reduction techniques based on event handler functions. Ruan Jian Xue Bao/Journal of Software, 2015,26(8):1871−1885 (in Chinese). http://www.jos.org.cn/1000-9825/4711.htm

GUI Test Suite Reduction Techniques Based on Event Handler Functions

CHEN Jun-Cheng[1,2,3], XUE Yun-Zhi[1,2], TAO Qiu-Ming[1,2], ZHAO Chen[1,2]

[1](Laboratory of Fundamental Software Testing and Evaluation, Institute of Software, The Chinese Academy of Sciences, Beijing 100190, China)

[2](National Engineering Research Center for Fundamental Software (Institute of Software, The Chinese Academy of Sciences), Beijing 100190, China)

[3](University of Chinese Academy of Sciences, Beijing 100049, China)

Abstract: GUI test suite reduction is an effective approach to reduce test cost. Due to the mechanics of message loop and the event-driven characteristic of GUI software, it is difficult to directly apply traditional test suite reduction techniques, such as control-flow based technique and data-flow based technique, to GUI test suite reduction. How to eliminate more redundant test cases without loss of the ability of finding errors is still a great challenge. Combining control flow technique and data flow technique, this paper proposes three test reduction techniques based on source code structure of event handler functions and the data dependencies among them. Experimental results show that two of the techniques that based on the data dependency among event handler functions achieve good results.

Key words: GUI test case; test suite reduction; event handler function; def-use; redundant test case

图形用户界面(graphic user interface)是现代软件的重要组成部分,GUI 功能正确性是影响用户体验的一个重要因素.GUI 测试是提高 GUI 软件质量的一种有效方式,它用于验证被测应用的 GUI 功能是否与规约一致[1].

* 基金项目: 国家自然科学基金(61100067, 61100070); 国家重大科技专项(2012ZX01039-004)
收稿时间: 2014-05-20; 修改时间: 2014-07-21; 定稿时间: 2014-08-24

© 中国科学院软件研究所 http://www.jos.org.cn

测试用例集规模直接决定测试成本,特别是在回归测试中,测试用例集规模对测试成本影响重大[2].当前,大部分 GUI 自动化测试研究围绕事件流图模型或其衍生模型展开[1,3−6],以事件流图模型为基础的测试覆盖准则根据相邻事件序列长度制定[7].理论上,随着测试覆盖准则中要求的事件序列长度的增加,测试用例规模呈指数级增长;在具体实践中,针对开源软件 FreeMind,GanttProject,jEdit 和 OmegaT 的仅包括种子测试用例的数量分别为 309 136,84 681,204 304,38 809,以 50 台机器并行执行这些种子测试用例的时间分别为 1 293.54 小时、294.68 小时、662.53 小时、143.39 小时[8].在回归测试中,这种规模的测试用例和执行时间会极大地增加测试成本.

测试用例集约简,是降低测试成本的一种有效手段.传统的测试用例集约简技术通过程序控制流图上的结构覆盖准则(如语句覆盖、分支覆盖、路径覆盖)或依赖关系覆盖准则(如定义-引用覆盖)等定义测试用例冗余规则,对测试用例集进行约简.GUI 软件基于消息循环,具有事件驱动特性.GUI 软件启动时,首先完成软件相关的初始化工作,然后进入消息循环,当用户操作 GUI 控件时产生 GUI 消息,触发底层事件处理函数执行,GUI 软件状态发生迁移,完成预期的功能,然后再次进入消息循环,等待下一个 GUI 消息,直至用户关闭 GUI 软件.GUI 软件的这种行为特征无法用程序控制流图直接进行描述,因此,传统的测试用例集约简技术无法直接应用于 GUI 测试用例集约简.

现有的专门针对 GUI 测试用例集约简的研究较少.为了降低 GUI 测试用例集的规模,Mc Master 等人提出了一种基于 GUI 运行时的调用线程栈约简技术[9].这种技术认为:如果两个测试用例执行时,对应的线程最大深度调用线程栈的执行序列一致,则认为两个测试用例等价,并以此规则进行测试用例集约简.实验结果表明:相对于其他约简技术,这种约简技术能够在很大程度上保持原有测试用例集的缺陷发现能力,但是测试用例规模的约简率较低[9].如何在尽可能保持原有 GUI 测试用例集的缺陷发现能力的基础上尽可能地降低 GUI 测试用例集规模,是 GUI 测试用例集约简的一个挑战.

对于 GUI 软件开发而言,绝大部分 GUI 开发框架都提供了完整的消息循环支持、事件处理函数注册机制以及功能丰富的 GUI 库.在多数情况下,开发者只需要根据 GUI 软件需求,重载或开发相应的事件处理函数.事件处理函数响应用户的 GUI 操作完成软件的预定义功能,根据需求准确实现事件处理函数是软件开发人员的核心工作.如图 1 所示,窗口 Form1 中的 3 个按钮“Func1”,“Func2”和“Reset”上的点击事件分别触发对应箭头所指的 3 个事件处理函数 $btnFunc_1$,$btnFunc_2$,*reset* 的执行.

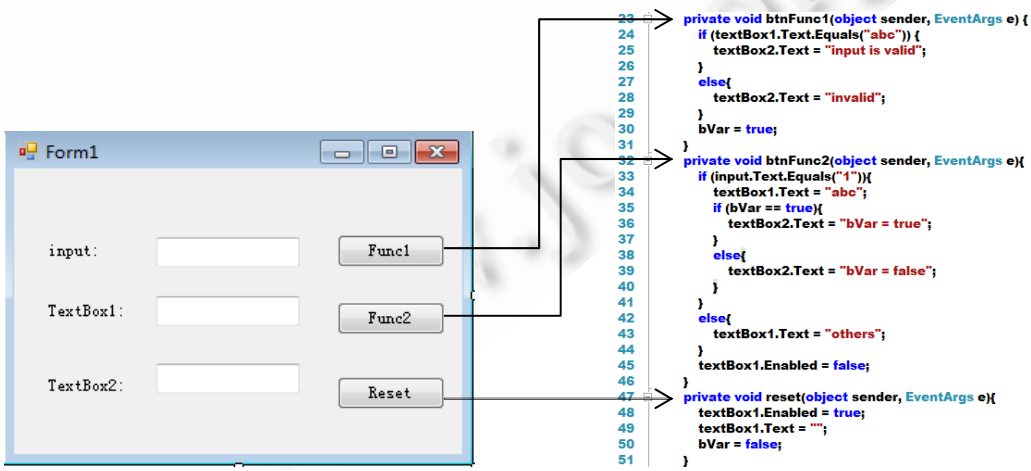

Fig.1 Example of GUI application

图 1 GUI 程序示例

GUI 框架通常复用已有的成熟框架,如 MFC,.Net 等,其出错的概率较低;而开发者重载或开发的事件处理函

© 中国科学院软件研究所 http://www.jos.org.cn

数比较容易出错,事件处理函数的正确性以及事件处理函数之间共享变量关系的正确性是 GUI 测试的重点[10],尤其是在测试资源有限的情况下,应当优先加以测试.

基于上述观察,本文以事件处理函数为核心,从事件处理函数的结构覆盖和事件处理函数间数据依赖关系覆盖两个角度定义测试用例冗余规则,并以此为基础进行测试用例集约简.

本文首先提出事件处理函数的执行路径、事件处理函数定义-引用对和事件处理函数执行路径定义-引用对等概念,然后以此为基础,根据事件处理函数的结构覆盖和事件处理函数之间的数据依赖关系覆盖,定义相应的 GUI 测试用例冗余规则,并设计和实现相应的 GUI 测试用例集约简算法.

以 GUI 测试用例集约简基准测试程序为基础,本文比较分析了已有的 GUI 测试用例集约简技术和本文提出的 GUI 测试用例集约简技术.实验结果表明:

- 与传统的测试用例集约简技术(基于语句行覆盖、基于函数覆盖等)相比,本文提出的 3 种技术具有较低的约简下降率和较低的错误检测下降率,去除的有效测试用例(指能发现错误的测试用例,下文同)也更少.
- 与基于调用线程栈约简技术相比,本文提出的两种基于数据依赖关系覆盖的技术中:一种在错误检测率略差的情况下,测试用例集规模约简下降率明显高于调用线程栈约简技术;另一种与调用线程栈约简技术具有一致的约简下降率和错误检测下降率,但是不会去除有效测试用例.

本文第 1 节根据事件处理函数特征定义基本概念.第 2 节介绍 GUI 测试用例冗余规则.第 3 节介绍测试用例集约简算法和测试用例集约简工具的执行流程.第 4 节介绍本文提出的测试用例集约简技术在已有 GUI 测试用例集约简基准测试程序上的实验效果以及与其他方法的比较分析.第 5 节介绍 GUI 测试用例集约简相关研究工作.最后,对本文进行总结并介绍下一步工作.

1 基本概念

本节首先介绍语句块、语句块的定义和引用;然后以此为基础,根据事件处理函数的控制流图特征定义事件处理函数的执行路径,根据事件处理函数之间的共享变量关系定义事件处理函数定义-引用对以及事件处理函数执行路径定义-引用对.

定义 1(语句块(statement block)). 语句块是一个具有单入口单出口的语句序列 $Sb=\langle sts_1,sts_2,\ldots,sts_n\rangle(n>0)$,其中,$Sb$ 标识语句块,$sts_i(1\leqslant i\leqslant n)$标识语句块中的简单语句[11].语句块中的语句执行时按照序列顺序执行,语句块中的语句 $sts_j(1<j<n)$具有唯一的前驱 sts_{j-1} 和唯一的后继 sts_{j+1}.

图 1 示例中的 3 个事件处理函数对应的 CFG(control flow graph,简称 CFG)用语句块表示,如图 2 所示.

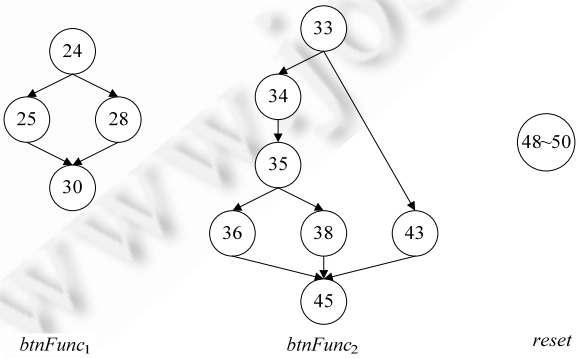

Fig.2 CFGs of event handler functions in the example

图 2 示例图中事件处理函数对应的 CFG

图中每个节点表示一个语句块,节点中的数字标识语句块中语句的行数(if 语句、while 语句、for 语句、

© 中国科学院软件研究所 http://www.jos.org.cn

switch 语句等中的条件表达式对应一个语句块).

定义 2(语句块的定义和引用(definition and use of statement block)). 语句块的定义是语句块中所有语句定义的变量的集合;语句块的引用是语句块中所有语句引用的变量的集合.若 $Sb=\langle sts_1,sts_2,\ldots,sts_n\rangle (n>0)$,则语句块的定义 $def(Sb)=\bigcup def(sts_i)$ $(1\leqslant i\leqslant n)$,语句块的引用 $use(Sb)=\bigcup use(sts_i)$ $(1\leqslant i\leqslant n)$,$def(sts_i)$和 $use(sts_i)$分别表示语句 sts_i 的定义和引用[12].

在 GUI 应用程序源代码中,通常存在两种类(class,面向对象程序设计中的一个概念):一种为 GUI 窗口类,另一种为非 GUI 窗口类.前者提供与用户交互的界面(包括输入文本框、按钮等控件元素)并重载特定的事件处理函数,以完成特定的功能需求;后者常用于表述内部对象,实现软件底层的复杂逻辑,被事件处理函数直接或间接调用.从 GUI 测试的角度看,GUI 测试主要关注单个事件处理函数以及事件处理函数之间的关系的实现是否与预期一致.

事件处理函数直接或间接调用其他函数.本文关注事件处理函数和被事件处理函数调用的 GUI 类的成员函数和非 GUI 类的成员函数.从事件处理函数代码结构覆盖的角度定义测试冗余规则,需要考虑事件处理函数的控制流图(CFG).CFG 是软件测试中常用的一种表述程序执行逻辑的测试模型,每个事件处理函数均存在一个对应的控制流图.为了进一步测试 GUI 事件处理函数对 GUI 类成员函数以及非 GUI 类成员函数的调用关系,使用过程间控制流图(inter-procedure control flow graph,简称 ICFG[13])表示事件处理函数的执行逻辑,在构建事件处理函数的 ICFG 过程中,以下调用可以当作过程间调用:

- 事件处理函数调用一个 GUI 类的成员函数;
- 事件处理函数调用一个非 GUI 类的成员函数;
- GUI 类中的成员函数调用 GUI 类的成员函数;
- GUI 类的成员函数调用非 GUI 类的成员函数.

根据定义 1,每个 ICFG 都包含一组语句块 $\{sb_{i1},sb_{i2},\ldots,sb_{im_i}\}$ $(i>0,m_i>0)$.每个 GUI 事件处理函数都有一个对应的 ICFG,因此,每个 GUI 事件处理函数的语句块集合可表示为 $func_{isb}=\{sb_{i1},sb_{i2},\ldots,sb_{im_i}\}$,其中,$func_{isb}$ 表示 GUI 事件处理函数的名称,$sb_{ij}(0<j\leqslant m_i)$表示事件处理函数中的语句块.

定义 3(事件处理函数的定义和引用(definition and use of event handler function)). 事件处理函数 ehf 的定义是事件处理函数中所有语句块定义的变量的集合,记为 $def(ehf)$;事件处理函数 ehf 的引用是事件处理函数中所有语句块引用的变量的集合,记为 $use(ehf)$.

定义 4(事件处理函数的执行路径(execution path of event handler function)). 事件处理函数执行路径是一个语句块有序序列,表示为 $ehf_{path_i}=\langle sb_{i1},sb_{i2},\ldots,sb_{im_i}\rangle$ $(i>0,m_i>0)$,其中,ehf 表示事件处理函数名称,$path_i$ 表示 ehf 的一条执行路径,sb_{ij} 表示执行路径 $path_i$ 上的语句块.本文用 $def(ehf_{path_i})$ 表示路径 $path_i$ 上的所有语句块的定义的变量的集合,用 $use(ehf_{path_i})$ 表示路径 $path_i$ 上的所有语句块引用的变量的集合.

定义 4 描述了测试用例执行时,事件处理函数内部的执行信息.

对于同一个事件处理函数 ehf,$def(ehf)\supseteq def(ehf_{path})$,$use(ehf)\supseteq use(ehf_{path})$,因为 $def(ehf)$,$use(ehf)$分别包含所有事件处理函数中的所有的语句块定义和引用的变量的集合,而 $def(ehf_{path})$,$use(ehf_{path})$则只分别包含事件处理函数某条特定的执行路径 $path$ 上的相关语句块的定义和引用的变量的集合.

若事件处理函数 ehf 存在两条执行路径 $ehf_{path_i}=\langle sb_{i1},sb_{i2},\ldots,sb_{im_i}\rangle$,$ehf_{path_j}=\langle sb_{j1},sb_{j2},\ldots,sb_{jm_j}\rangle$ $(i>0,m_i>0,m_j>0)$,且 $\exists k,sb_{i1}=sb_{jk}\wedge sb_{i2}=sb_{j(k+1)}\wedge\ldots\wedge sb_{im_i}=sb_{j(k+m_i-1)}$ $(0<k\leqslant m_j+1-m_i)$,则称 ehf_{path_i} 是 ehf_{path_j} 的一个子序列,记为 $ehf_{path_i}\subseteq ehf_{path_j}$.

在上述定义的基础上,如果考虑事件处理函数中的执行路径,则 GUI 测试用例可以表示为

$$\langle func_1:\langle sb_{11},sb_{12},\ldots,sb_{1m_1}\rangle,\ldots,func_n:\langle sb_{n1},sb_{n2},\ldots,sb_{nm_n}\rangle\rangle\ (n>0,m_1>0,m_n>0).$$

如果不考虑事件处理函数中的执行路径,则 GUI 测试用例可以表示为$\langle func_1,func_2,\ldots,func_n\rangle$.

事件处理函数之间存在共享变量关系,这种共享变量关系会导致事件处理函数之间存在数据依赖关系,如

© 中国科学院软件研究所 http://www.jos.org.cn

图1中的变量*bVar*使得事件处理函数$btnFunc_1$与$btnFunc_2$之间存在数据依赖关系.定义5和定义6分别从事件处理函数的角度和事件处理函数中的执行路径的角度对这种数据依赖关系进行描述.

定义5(事件处理函数定义-引用对(def-use pair of event handler function)). 若有测试用例$tc=\langle func_1, func_2,\ldots,func_n\rangle$,如果存在某个变量$var$,$var\in def(func_i)\wedge var\in use(func_j)$,且$var\supseteq def(func_k)(i<k<j)$,则称$(func_i,func_j)$为关于$var$的事件处理函数定义-引用对,记为$(func_i,func_j,var)$.

定义6(事件处理函数执行路径定义-引用对(path def-use pair of event handler function)). 若有测试用例$tc=\langle func_1:\langle sb_{11},sb_{12},\ldots\rangle,\ldots,func_i:\langle sb_{i1},sb_{i2},\ldots\rangle,\ldots,func_j:\langle sb_{j1},sb_{j2},\ldots\rangle,\ldots,func_n:\langle sb_{n1},sb_{n2},\ldots\rangle\rangle$,如果存在变量$var\in def(func_{ipath}),var\in use(func_{jpath})$,且$var\supseteq def(func_{kpath})(i<k<j)$,则称$(func_{ipath},func_{jpath})$为关于变量$var$的事件处理函数执行路径定义-引用对,记为$(func_{ipath},func_{jpath},var)$.

在定义6的基础上,如果存在两个事件处理函数执行路径定义-引用对:

$$pair_1=(func_{ipath_1},func_{jpath_1},var),pair_2=(func_{ipath_2},func_{jpath_2},var),且func_{ipath_1}\subseteq func_{ipath_2}\wedge func_{jpath_1}\subseteq func_{jpath_2},$$

则称$pair_1$是$pair_2$的子序列,记为$pair_1\supseteq pair_2$.

定义5和定义6描述了事件处理函数之间的依赖关系,前者从事件处理函数序列的角度考虑,后者则考虑事件处理函数中的执行路径.若有事件处理函数执行路径定义-引用对$pair_{path}=(func_{ipath_1},func_{jpath_1},var)$,则必然存在事件处理函数定义-引用对$pair_{func}=(func_i,func_j,var)$($func_{ipath_1},func_{jpath_1}$分别是$func_i,func_j$的一条执行路径).

2 GUI测试约简规则

冗余测试用例定义是削减测试用例的基础,在第1节定义的基础上,本节根据事件处理函数控制流图结构定义基于事件处理函数执行路径的冗余测试用例;根据事件处理函数之间的控制流和数据流定义基于事件处理函数定义-引用对的冗余测试用例以及基于事件处理函数执行路径的定义-引用对的冗余测试用例.

从事件处理函数的结构覆盖的角度来看,当两个GUI测试用例覆盖的事件处理函数集合相同,且每个事件处理函数中的执行路径一致时,则从事件处理函数结构覆盖的角度认为两个测试用例等价.事件处理函数之间还存在共享变量关系,这种共享变量关系使得事件处理函数之间产生数据依赖关系,当两个GUI测试用例覆盖的事件处理函数之间的这种数据依赖关系一致时,则从事件处理函数数据依赖关系覆盖的角度认为这两个测试用例等价.

事件处理函数的一条执行路径代表一个功能.如果测试用例集*TS*和其子集*TS′*覆盖的执行路径集合均为S_{path},即,*TS*和*TS′*对GUI软件的同一个功能集合完成了测试,则认为*TS′*是*TS*的一个关于事件处理函数执行路径的测试用例约简集.

根据事件处理函数对应的ICFG,基于事件处理函数执行路径的冗余测试用例定义如下:

定义7(基于事件处理函数执行路径的冗余测试用例(event handler function's path-based redundant test case)). 若有测试用例集*T*,其覆盖的事件处理函数执行路径集合为$R_{path}=\{r_1,r_2,\ldots,r_n\}$,GUI测试用例*tc*覆盖的事件处理函数执行路径为$R_{tc}=\{r_{1tc},r_{2tc},\ldots,r_{ktc'}\}(k>0)$,如果$\supseteq r\in R_{tc},\supseteq r'\in R_{path},r\supseteq r'$(见第1节子序列定义),则称*tc*是*T*基于事件处理函数执行路径的冗余测试用例,记为$Path(T,tc)$.

定义7从事件处理函数执行路径的集合角度进行测试用例冗余判定.除了事件处理函数的执行路径信息以外,事件处理函数之间还存在数据依赖关系,事件处理函数定义-引用对或者事件处理函数执行路径定义-引用对数据依赖关系进行描述,前者从函数的抽象层次描述事件处理函数之间的数据依赖关系,后者则从函数中具体的执行路径的层次描述事件处理函数之间的数据依赖关系.

如果测试用例集*TS*和其子集*TS′*覆盖的事件处理函数路径集合一致,并且覆盖的事件处理函数定义-引用对/事件处理函数执行路径定义-引用对集合均为R_{path},则可以认为*TS′*是*TS*的一个关于事件处理函数定义-引用对/事件处理函数执行路径定义-引用对的测试约简集.

根据事件处理函数之间的数据依赖关系,基于事件处理函数定义-引用对和事件处理函数执行路径定义-引用对的冗余测试用例定义如定义8和定义9所示.

© 中国科学院软件研究所 http://www.jos.org.cn

定义 8(基于事件处理函数定义-引用对的冗余测试用例(event handler function's def-use pair-based redundant test case)). 若测试用例集 T 中的所有测试用例覆盖的事件处理函数执行路径集合和事件处理函数定义-引用对集合分别为 $R_{path}=\{r_1,r_2,\ldots,r_n\}$, $R_{dufunc}=\{r'_1,r'_2,...,r'_m\}$,GUI 测试用例 tc 覆盖的事件处理函数执行路径集合和事件处理函数定义-引用对集合分别为 $R_{tc}=\{r_{1tc},r_{2tc},\ldots,r_{ktc}\}$, $R'_{tc}=\{r'_{1tc},r'_{2tc},...,r'_{ktc}\}$ (k>0),若$\forall r\in R_{tc},\exists r'\in R_{path}$, $r\subseteq r'$,并且$\forall r\in R'_{tc}, r\in R_{dufunc}$,则称 tc 是 T 的基于事件处理函数定义-引用对的冗余测试用例,记为 $duFunc(T,tc)$.

定义 9(基于事件处理函数执行路径定义-引用对的冗余测试用例(event handler function's path def-use pair-based redundant test case)). 若测试用例集 T 中的所有测试用例覆盖的事件处理函数执行路径集合和事件处理函数定义-引用对集合分别为 $R_{path}=\{r_1,r_2,\ldots,r_n\}$, $R_{dufunc}=\{r'_1,r'_2,...,r'_m\}$, GUI 测试用例 tc 覆盖的事件处理函数执行路径集合和事件处理函数定义-引用对集合分别为 $R_{tc}=\{r_{1tc},r_{2tc},\ldots,r_{ktc}\}$, $R'_{tc}=\{r'_{1tc},r'_{2tc},...,r'_{ktc}\}$, 如果$\supseteq r\in R_{tc}$, $\supseteq r'\in R_{path}, r\supseteq r'$,并且$\supseteq r\in R'_{tc}, \supseteq r'\in R_{dupath}, r\supseteq r'$,则称 tc 是 T 的基于事件处理函数执行路径定义-引用对的冗余测试用例,记为 $duPath(T,tc)$.

若有 GUI 测试用例集 T 和一个 GUI 测试用例 tc,如果测试执行按照同样的顺序执行,根据定义 7、定义 8 和定义 9,有如下结论:

结论 1. 如果 $duFunc(T,tc)$成立,则 $Path(T,tc)$必然成立.

结论 2. 如果 $duPath(T,tc)$成立,则 $Path(T,tc)$必然成立.

3 GUI 测试用例约简算法及实现方法

根据测试用例冗余规则定义,测试用例集约简算法如算法 1 所示.

算法 1. 测试用例集约简算法.

```
Input
TS: Original Test Suite(未约简测试用例集).
Procedure:
1. RTS=∅;
2. TR=∅;   //RTS 覆盖的测试需求集合
3. for all tc∈TS do
4.       If IsNotRedundant(RTS,TR,tc) then
5.            AddTc(RTS,tc);
              //获取 tc 覆盖的测试需求
6.            tcCovered=GetAllCovered(tc);
              //将 tc 覆盖的测试需求加入 TR 中
7.            AddCovered(TR,tcCovered);
8.       endif
9. endfor
Output
RTS: Reduced Test Suite(约简测试用例集).
```

算法 1 中,*TR* 用于收集约简测试用例集 *TS* 覆盖的测试需求集(根据定义 7 判断测试用例冗余时,测试需求是事件处理函数执行路径;根据定义 8 判断测试用例冗余时,测试需求是事件处理函数执行路径以及事件处理函数定义-引用对;根据定义 9 判断测试用例冗余时,测试需求是事件处理函数执行路径以及事件处理函数执行路径定义-引用对).第 3 行~第 9 行逐个对未约简测试用例集 *TS* 中的测试用例 *tc* 的执行信息进行检查,判断 *tc* 是否是冗余测试用例(第 4 行中的 *IsNotRedundant* 函数):如果不是,则将 *tc* 加入约简测试用例集 *RTS* 中,并将 *tc* 覆盖的测试需求加入 *TR* 中.

在算法 1 中,*IsNotRedundant* 判断一个测试用例 *tc* 是否冗余分别根据定义 7~定义 9,其对应的测试用例集

© 中国科学院软件研究所 http://www.jos.org.cn

约简技术分别称为 path-reduction,duFunc-reduction 和 duPath-reduction 技术.

算法 1 中,判断测试用例集中的测试用例冗余是关键部分,其描述如算法 2 所示.

算法 2. 测试用例冗余判断.

Input

RTS: Reduced Test Suite(约简测试用例集);

TR: *RTS* 中的测试需求;

tc:待判断是否冗余的测试用例.

Procedure:

//获取 *tc* 覆盖的测试需求

1. *tr*=*GetAllCovered*(*tc*)
2. for each *tr′* in *tr*
3. if *tr′*∉*TR*
4. return true
5. endif
6. endfor
7. return false

output

true:表示不冗余,false:表示冗余

算法 2 中的 for 循环判断 GUI 测试用例 *tc* 覆盖的测试需求是否已全部被 *TR* 覆盖:如果全部覆盖,则表示 GUI 测试用例 *tc* 是冗余的;否则,*tc* 不冗余.其中,第 3 行判断测试用例 *tc* 所覆盖的测试需求 *tr′*是否已经被 *TR* 覆盖,在 path-reduction 中,*tr* 与 *TR* 表示事件处理函数中的执行路径集合;在 duFunc-reduction 中,*tr* 与 *TR* 表示事件处理函数执行路径集合和事件处理函数定义-引用对集合;在 duPath-reduction 中,*tr* 与 *TR* 则表示事件处理函数执行路径集合和事件处理函数执行路径定义-引用对集合.

若上述 4 种技术产生的约简测试用例套规模(即,测试用例套中测试用例的个数)分别为 Num_{path},Num_{duFunc},Num_{duPath},且上述 4 种技术在算法 1 中遍历测试用例的顺序一致,根据第 2.2 节中的结论 1、结论 2,则必有:

- $Num_{path} \leqslant Num_{duPath}$;
- $Num_{path} \leqslant Num_{duFunc}$.

为了验证分析本文提出的 GUI 测试用例集约简技术的有效性,我们设计并实现了一个基于 JDT(https://projects.eclipse.org/projects/eclipse.jdt)的 GUI 测试用例集约简原型工具.此工具完成事件处理函数静态分析、事件处理函数执行路径上的定义与引用的获取、根据测试用例信息对测试用例集进行约简等,其测试约简流程如图 3 所示.

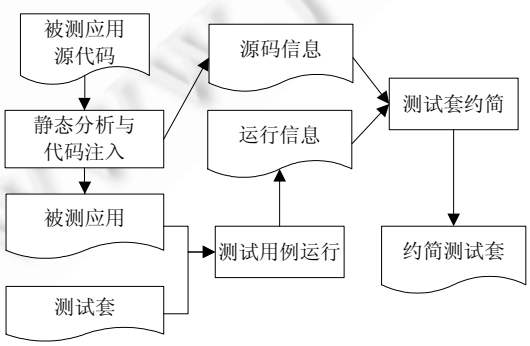

Fig.3 GUI test suite reduction process

图 3 GUI 测试用例集约简流程

© 中国科学院软件研究所 http://www.jos.org.cn

GUI 测试约简流程说明如下:

(1) 静态分析与代码注入.借助 Eclipse 的 JDT,对基准测试程序中的项目源码静态分析,根据事件处理函数特征,自动识别事件处理函数,为每个事件处理函数构建 ICFG,获取 ICFG 中每个语句块的定义和引用;为了跟踪 GUI 测试用例在事件处理函数中的运行信息,在关注的函数中的每个语句块的第 1 个语句之前注入监控语句(监控语句描述语句块的唯一标识),在每个事件处理函数的第 1 个语句块之前注入监控语句;根据事件处理函数的 ICFG 与语句块的关系和事件处理函数路径与语句块的关系,计算事件处理函数的定义和引用以及事件处理函数执行路径的定义和引用.此步骤产生一个注入监控代码的被测应用以及一个记录语句块的定义和引用信息的源码信息文件.

(2) 测试用例运行.在步骤(1)产生的注入监控代码的被测应用上运行测试套,产生包含每个测试用例运行记录信息的运行信息文件;在此文件中,每个测试用例运行的记录信息格式如下:

$$tcName:\langle func_1:sb_{11},sb_{12},\ldots,func_i:sb_{i1},sb_{i2},\ldots,func_n:sb_{n1},sb_{n2},...\rangle(i>0,n>0).$$

其中,$tcName$ 表示测试用例名称,$func_i$ 表示事件处理函数名称,sb_{ik} 表示 $func_i$ 执行的的第 $k(0<k<n)$个语句块.

(3) 测试约简.在步骤(1)产生的源码信息的基础上,根据定义 7~定义 9,利用算法 1 对运行信息进行计算,产生相应的约简测试用例集.

4 实验结果及分析

4.1 实验设计

本文从约简测试用例集规模的下降率、错误检测下降率以及约简测试用例集发现错误的数量这 3 个方面对本文提出的基于事件处理函数的 GUI 测试用例集约简技术和已有 GUI 测试用例集约简技术[3](包括基于调用线程栈、基于事件覆盖、基于事件交互覆盖、基于语句覆盖、基于函数覆盖以及随机约简的技术)进行比较评估.

本文选取由 Atif Memon 领导的 GUI 测试研究小组设计开发的 TerpOffice 中的 TerpSpreadSheet,TerpWord,TerpPaint 这 3 个开源应用作为被测对象,由 Atif Memon 的软件工程课程学生实现.这 3 个对象已被当做 GUI 测试用例集约简相关研究的基准测试程序(http://comet.unl.edu/benchmarks.php?q=group&g=umd.reduction.tse.2008),此基准测试程序具有如下特征:

(1) TerpSpreadSheet,TerpWord,TerpPaint 是微软 Office 工具套件的 Java 开源实现,实现了相应 Office 工具的大部分功能,从软件功能角度可以代表一般的办公软件;

(2) 对 TerpSpreadSheet,TerpWord,TerpPaint 产生的原始测试用例用例集中包含的测试用例个数分别为 1 000,1 000,1 500,根据事件流图中的测试覆盖准则生成;

(3) 根据常见的错误对 TerpSpreadSheet,TerpWord,TerpPaint 分别产生的错误版本个数分别为 234,295,263,由开发学生根据常见错误注入错误,每个错误版本中包含一个注入错误,用于对约简测试用例集发现错误的能力进行评估;

(4) TerpSpreadSheet,TerpWord,TerpPaint 本身所具有的事件[1]数量(#Events)、代码行数(#Lines)、类个数(#Classes)、函数个数(#Methods)、事件处理函数个数(#EHFs)、语句块个数(#Blocks)见表 1.

Table 1 Feature of GUI application under test

表 1 被测 GUI 应用属性

	TerpPaint (TP)	TerpWord (TW)	TerpSpreadSheet (TS)
#Events	181	219	110
#Lines	11 803	9 917	5 381
#Classes	330	197	135
#Methods	1 253	1 380	746
#EHFs	48	79	41
#Blocks	2 933	2 898	1 742

© 中国科学院软件研究所 http://www.jos.org.cn

为了便于与已有的 GUI 测试用例集约简技术[9](包括基于语句行、基于事件、基于事件交互、基于函数、以及基于调用线程栈的测试约简技术)比较分析,本文沿用文献[9]中对被测应用测试用例集的分组方法,即,每个被测应用的原有测试用例集利用随机方法分为 8 组测试用例集,分别代表测试用例个数为 50,100,150,200,250,300,350,400 的测试用例集,每组测试用例集中有 25 个测试用例集(每组测试用例集均覆盖所有的原有测试用例),其中,每个测试用例集中的原有测试用例个数与当前组对应的测试用例个数一致.

4.2 实验结果及分析

下述各图中,cs,L,funcpath,dufunc,dupath,M,E1,E2 分别表示基于调用线程栈覆盖、语句行、path-reduction、duFunc-reduction、duPath-reduction、函数、事件、事件交互的测试用例集约简技术;图中的 TP 代表 TerpPaint,TW 代表 TerpWord,TS 代表 TerpSpreadsheet.

图 4 描述了针对 3 个被测应用,每组测试用例集规模下降的比率为组中 25 个测试用例集规模下降的平均值.

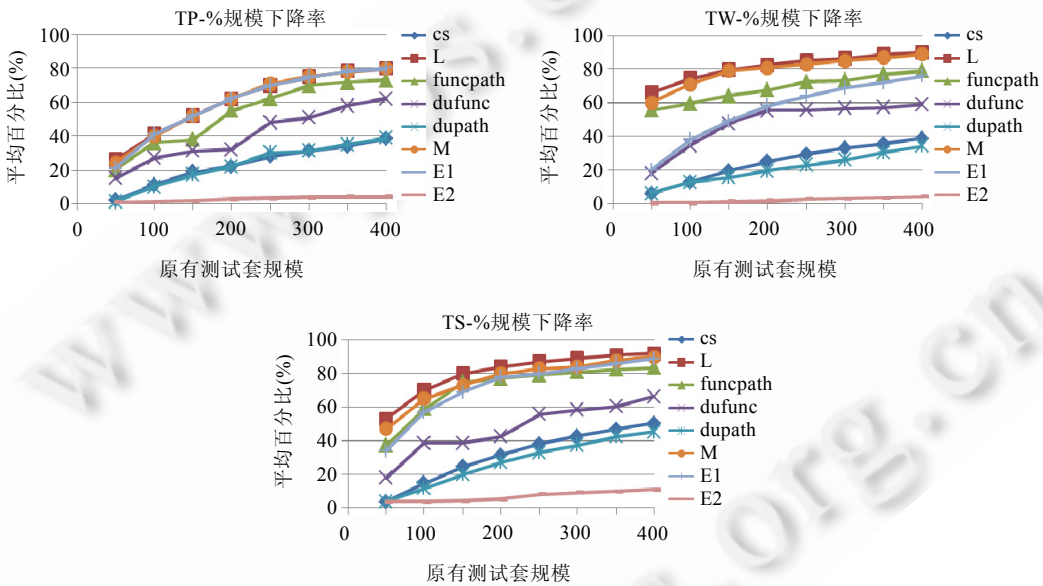

Fig.4 Size reduction of applications under test (AUTs)

图 4 测试用例集规模下降率

在 3 个应用中,随着测试用例规模的增加,cs,L,funcpath,dufunc,dupath,M,E1,E2 的测试用例集规模约简率(简称测试约简率,下文同)增加.对相同规模的原有测试用例集进行约简时,针对以上 3 个应用,具有如下特征:

- L(基于语句行的测试用例集约简技术)具有最高的测试约简率,E2(基于事件交互的测试用例集约简技术)具有最低的测试约简率.
- M 具有次高的测试约简率,与 L 非常接近.GUI 测试用例根据语句判断测试用例冗余时,根据语句所属的函数判断也必然冗余.
- 在 TP 和 TS 中,E1 与 M 非常接近;而在 TW 中则不明显,E1 的测试约简率明显要低于 M.
- funcpath 的测试约简率低于 M,其原因比较明显:funcpath 是以事件处理函数中的 ICFG 的路径为基础约简;M 只在函数抽象层次进行约简,未考虑其中的路径.当根据函数判断测试用例冗余时,根据 funcpath 判断则未必冗余;相反,如果根据 funcpath 判断冗余时,则根据函数判断必然冗余.因此,funcpath 的测试约简率低于 M.
- dufunc 的测试约简率低于 funcpath,其原因如定义 2 所示.

© 中国科学院软件研究所 http://www.jos.org.cn

- dupath 和 cs 的测试约简率非常接近,其原因在于,dupath 依据内部执行路径和对应的路径上的定义-引用关系的判断冗余,cs 则根据调用线程栈的最大深度进行判断.由于 cs 的判断依据的是线程的最大深度的中的执行序列,在此类办公软件中,GUI 线程中对应的是事件处理函数序列执行时具有最大深度的执行序列;而 dupath 根据事件处理函数内部执行路径和对应路径上的定义-引用关系,其对应的最大深度调用线程栈在绝大部分情况下与 cs 中的一致.因此,dupath 与 cs 的测试约简率较为接近,且两者的测试约简率均低于 dufunc.

约简后的测试用例集具有的发现错误的能力,是判断一个测试用例集约简技术是否有效的重要因素.为了进一步说明各种测试用例集约简技术的效果,特别是本文提出的 funcpath,dufunc,dupath 发现错误的能力,根据 funcpath,dufunc,dupath 约简后的测试用例集,针对每个应用,分别利用随机选择方法在每个原有测试套的基础上随机生成与 funcpath,dufunc,dupath 相同规模的测试用例集,分别用 RAND1,RAND2,RAND3 表示.图 5 描述了每组测试用例集发现缺陷下降的比率(组中 25 个测试用例集发现缺陷下降的比率的平均值),其特征如下:

- 若不考虑随机约简的测试用例集,在绝大部分情况下,E1 具有最高的错误检测下降率;在 TP 和 TS 中,M 具有次高的错误检测下降率;在 TW 中,L 具有次高的错误检测下降率;而 dufunc,dupath,cs 和 E2 较为接近,其错误检测下降率比其他技术都低.
- 随机约简的测试用例集,其错误检测下降率呈不规则变化.总体来看,约简后的测试用例集规模越大,错误检测下降率越低.

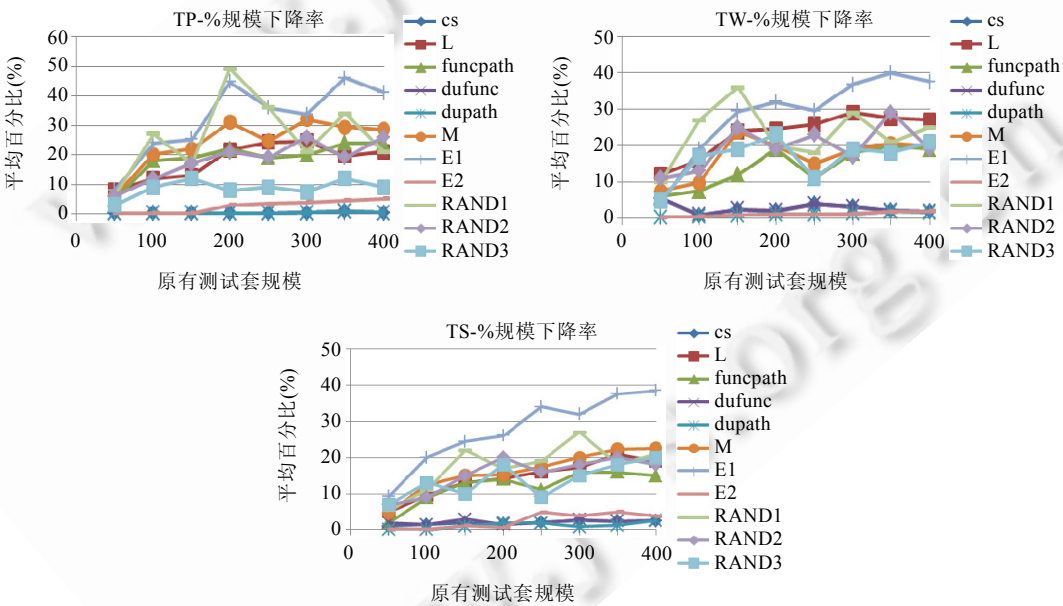

Fig.5 Fault detection reduction of AUTs

图 5 测试用例集错误检测下降率

错误检测下降率反映了约简后的测试用例集发现错误的能力:其值越高,表明约简后的测试效果越差;其值越低,则表明约简后的测试效果越好.

实际检测出的错误数目直接反映了约简测试集发现错误的能力,如图 6 所示.在约简过程中,一方面除了要对冗余测试用例进行约简,另一方面则要保证在约简过程中,尽可能地不去除有效测试用例(指能发现错误的测试用例,下文同).图 6 描述了约简后的测试集发现的各个应用的错误版本的个数(即发现错误的个数),其特征如下:

- 对于 TP,TW,TS 这 3 个被测应用的错误版本(由植入错误生成),在不约简的情况下,在每个分组中都能

© 中国科学院软件研究所 http://www.jos.org.cn

检出的错误数分别为 43,18 和 101.本文与文献[3]一致,都是分析比较同等条件下,不同测试用例集约简技术约简后的测试用例集检出错误版本的能力.虽然部分错误版本未被检出,但并不影响各种测试用例集约简技术的比较分析.

- 3 个被测应用中,虽然 E1 约简后的测试用例集规模最小,但其发现的错误数最少,即,约简了太多有效测试用例;L 和 M 次之,发现的错误数也比较少.
- 3 个应用中,dupath 发现错误的数量与未约简前发现的错误数量一致,即,dupath 的错误检测能力与原有测试套一致;dufunc 和 cs 发现错误的能力基本一致,略低于 dupath;E2 在 TW 中与 dufunc,cs 一致,在 TP 和 TS 中则低于 dufunc 和 cs.

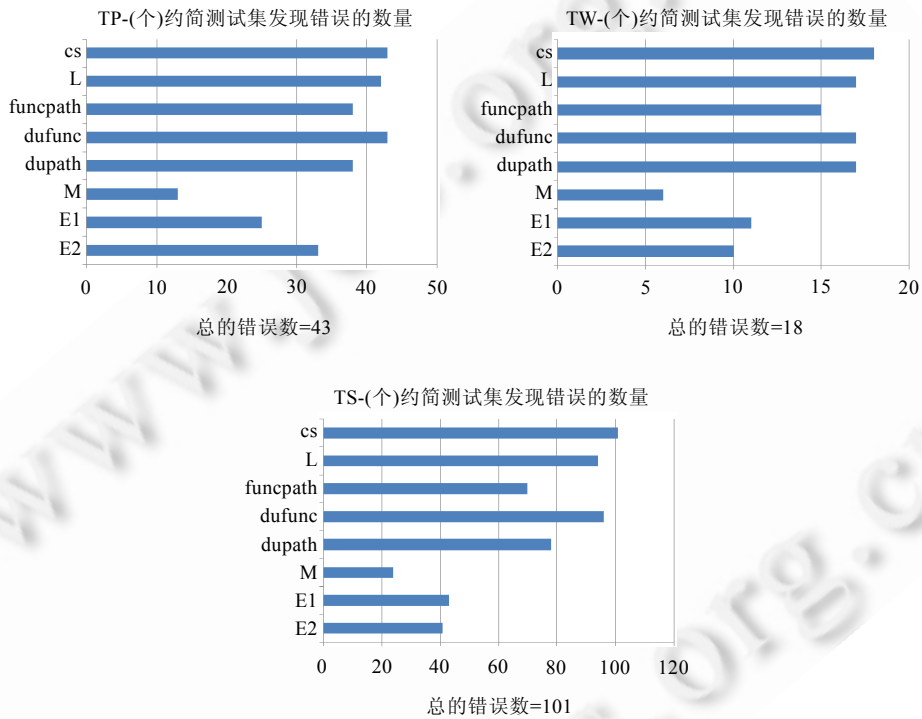

Fig.6 Faults found after reduction by techniques

图 6 约简后的测试用例集发现错误的个数

综合上述 3 组实验,针对 TerpOffice 中的 3 个被测应用,上述各种测试用例集约简技术具有如下特征:

- 传统的测试用例集约简技术 L 和 M 具有较高的测试约简率,但是去除有效测试用例规模较大,导致未发现的错误数也比较多.
- 基于事件的测试用例集约简技术 E1 具有较高的测试约简率,但是约简后的测试用例集发现错误的能力最差;而基于事件交互的测试约简技术 E2 在测试约简率比 dufunc 更低的情况下,其发现错误的能力还低于 dufunc.
- 本文提出的 3 种基于事件处理函数的约简技术中,dupath 与 cs 几乎具有一致的测试约简下降率,但是其未去除有效测试用例;dufunc 的错误检测下降率和发现的错误数量比 cs 略差,但是前者产生的测试用例规模明显低于后者(当原有测试用例规模大于 300 时,前者只有后者的 40%~68%).

为了进一步地分析 dufunc 和 cs 产生的测试用例集发现的错误之间的关系,若 cs 约简测试用例集发现的错误集合为 $E_{cs}=\{e_{cs1},e_{cs2},\ldots,e_{csn}\}$,dufunc 约简测试用例集发现的错误集合为 $E_{df}=\{e_{df1},e_{df2},\ldots,e_{dfm}\}$,则

$$R_{df/cs}=|E_{df}\cap E_{cs}|/|E_{cs}|\times 100\%$$

© 中国科学院软件研究所 http://www.jos.org.cn

表示 dufunc 约简测试用例集发现的错误占 cs 约简测试用例集发现错误的比率.$R_{df/cs}$ 值越高,则说明针对 E_{cs}, dufunc 约简技术与 cs 约简技术越接近,见表 1.

Table 2 Distribution of $R_{df/cs}$

表 2 $R_{df/cs}$ 分布表

原测试用例集规模(个)	$R_{df/cs}$ (%)		
	TS	TW	TP
50	66.67	91.67	88.89
100	80.96	72.41	88.24
150	73.53	78.38	88.89
200	80.56	84.78	87.80
250	80	84.78	91.84
300	91.11	91.3	91.38
350	91.11	92	96.77
400	89.58	92	91.3

表 2 说明:$R_{df/cs}$ 值在大部分情况下能达到 80%以上;特别是当原有测试用例集规模达到 300 后,$R_{df/cs}$ 值几乎都在 90%以上.即,在 dufunc 约简测试用例集规模明显小于 cs 约简测试用例集规模的情况下(前者只有后者的 40%~68%),依然能发现 cs 约简测试用例集检测发现的绝大部分错误.

综合上述分析,与 cs 约简技术比较,dufunc 约简技术发现错误的能力略差,但是能较大程度地降低测试用例规模,减少测试运行时间;dupath 则在与 cs 保持基本一致的测试用例规模的情况下,具有更好的发现错误的能力.

5 相关工作

测试用例集约简的通用办法是:依据测试目标,对冗余测试用例集进行约简.具体而言,存在两种方法:一种是根据测试用例与测试需求之间的对应关系,对满足特定测试需求的多个测试用例进行约简;另一种是对复杂的测试需求进行约简,得到简化的测试需求,根据简化的测试需求生成测试用例,以此达到约简测试用例的目的[14].本文提出的测试用例集约简技术采用的是第 1 种方法,以事件处理函数中的执行路径以及事件处理函数之间的数据依赖关系为测试需求元素,根据测试用例与测试需求覆盖的元素的对应关系进行 GUI 测试用例集约简.

根据不同的测试需求,测试用例集约简所采用的技术也不同.Gupta 等人根据软件需求规约,制定测试需求决策表(decision table),并根据测试用例与测试需求之间的关系进行测试用例集约简[15];Black 等人针对 def-use 依赖关系的不足,从过程间 all-use 的角度对测试用例集进行约简[2];结合模糊测试方法,Kumar 等人通过模糊聚类对测试用例进行分类,以此为基础判断测试用例是否冗余并进行约简[16];Chen 等人则从被测系统的有限状态机模型角度考虑,根据版本演进过程的模型变化对测试用例集进行约简[17];Schroeder 等人以输入与输出之间的对应关系为测试需求,对测试用例集进行约简[18];降低测试用例集规模可能降低测试套发现缺陷的能力,为了尽可能地降低这种影响,Jeffrey 等人提出了一种基于多种测试覆盖准则的测试用例集约简方法[19,20].

测试用例集约简研究主要集中在测试用例集最小化以及测试用例集约简、覆盖准则和发现错误的能力这三者之间的关系.测试用例集最小化是一个 NP 问题,为此,测试用例集约简根据不同的测试需求,采用启发式算法[21]、贪心算法[22]、基因遗传算法[23]进行最小化测试用例集约简,文献[24,25]对几种测试用例集约简算法进行了相关比较研究.

测试用例集约简、覆盖准则和发现错误能力三者之间的关系是测试人员关注的一个研究方向.文献[26–28]研究测试用例集约简、测试需求覆盖率、发现错误能力之间的关系,总体来看,测试用例集约简会降低错误发现的能力.文献[29]研究约简测试用例集与错误定位的关系,总体而言,根据不同的测试覆盖产生的约简测试用例集,定位错误的能力不同.本文从约简测试用例集规模和发现错误的能力的角度,对已有的 GUI 测试用例集约简技术以及本文提出的 3 种 GUI 测试用例集约简技术进行分析.

Mei 等人通过静态分析技术获取 JUnit 测试用例的函数调用图,以函数调用图为核心,在不执行 JUnit 测试用例的情况下,从函数覆盖层次或语句覆盖层次估算 JUnit 测试用例的覆盖率,以此对 JUnit 测试用例集进行排

© 中国科学院软件研究所 http://www.jos.org.cn

序[30].本文利用静态分析技术构造事件处理函数的 ICFG,根据测试用例执行时的运行信息,从执行路径的覆盖和执行路径上的数据依赖覆盖两个角度定义冗余测试用例,并对测试用例集进行约简.本文提出的测试用例集约简技术根据事件处理函数的 ICFG 展开,与文献[30]中的函数调用图类似,从这个意义上讲,两者有相通之处.但是,文献[30]在不执行测试用例的条件下,根据测试用例的函数调用图估算测试用例的覆盖率,指导测试用例集排序;本文则需要确切的运行时信息(执行路径的覆盖和执行路径上的数据依赖覆盖)对测试用例集进行约简.

Mc Master 等人从 GUI 测试用例执行时调用线程栈的角度提出了一种基于调用线程栈覆盖准则的 GUI 测试用例集约简技术,而且通过实验验证了此技术在基本不影响测试效果的情况下,降低 GUI 测试用例规模[3],该方法重点考察的是调用线程栈中的方法序列.本文提出的 GUI 测试测试用例集约简方法,借鉴传统测试用例集约简技术中的测试冗余规则,根据事件处理函数中的控制流结构(事件处理函数执行路径)和数据流信息(事件处理函数之间的数据依赖关系)特征制定测试用例冗余规则,重点考察事件处理函数中的控制流信息和数据流信息.

6 总结与未来工作

本文从事件处理函数的代码结构特征以及事件处理函数之间的关系角度,利用控制流和数据流技术对事件处理函数的执行信息以及事件处理函数之间共享变量进行分析,分别提出语句块、事件处理函数执行路径、事件处理函数定义-引用对和事件处理函数执行路径定义-引用对等基本概念,以此为基础定义测试用例冗余规则,制定并实现了 funcpath,dufunc 和 dupath 这 3 种测试用例集约简技术.针对 TerpOffice 中 3 个应用的实验结果表明:

- 与基于事件的测试用例集约简技术 E1 和传统的测试用例集约简技术 L,M 相比,三者具有较低的约简下降率和较低的错误检测下降率,去除的有效测试用例也更少.
- 与基于调用线程栈的技术 cs 相比,funcpath 具有更低的测试约简下降率和较高的错误检测下降率;dupath 与 cs 具有基本一致的测试约简下降率和错误检测下降率,且 dupath 没有去除有效测试用例;dufunc 的错误检测能力略差于 cs,但是其约简测试用例集规模明显低于 cs.

下一步工作将以现有工作为基础,从如下两个方面展开:

(1) 针对 GUI 测试用例集约简过程中的测试执行顺序进行研究,以期进一步降低测试用例规模,并进一步提高约简测试用例集发现 GUI 错误的效率;

(2) 将本文提出的测试冗余规则应用于一般的具有事件驱动特性的软件测试用例集约简中,研究评估其约简效果.

References:

[1] Memon AM. An event-flow model of GUI-based applications for testing. Software Testing, Verification & Reliability, 2007,17(3): 137−157. [doi: 10.1002/stvr.364]

[2] Black J, Melachrinoudis E, Kaeli D. Bi-Criteria models for all-uses test suite reduction. In: Proc. of the 26th Int'l Conf. on Software Engineering (ICSE 2004). New York: ACM Press, 2004. 106−115. [doi: 10.1109/ICSE.2004.1317433]

[3] Fraser G, Wotawa F. Redundancy based test-suite reduction. In: Proc. of the FASE 2007. LNCS 4422, Berlin, Heidelberg: Springer-Verlag, 2007. 291−305. [doi: 10.1007/978-3-540-71289-3_23]

[4] Memon AM, Soffa ML, Pollack ME. Hierarchical GUI test case generation using automated planning. IEEE Trans. on Software Engineering, 2001,27(2):144−155. [doi: 10.1109/32.908959]

[5] Brooks P, Memon AM. Automated GUI testing guided by usage profiles. In: Proc. of the 22nd Int'l Conf. on Automated Software Engineering. New York: ACM Press, 2007. 333−342. [doi: 10.1145/1321631.1321681]

[6] Yuan X, Memon AM. Iterative execution-feedback model-directed GUI testing. Information and Software Technology, 2010,52(5): 559−575. [doi: 10.1016/j.infsof.2009.11.009]

© 中国科学院软件研究所 http://www.jos.org.cn

[7] Memon AM, Soffa ML, Pollack ME. Coverage criteria for GUI testing. In: Proc. of the 8th European Software Engineering Conf. on Held Jointly with 9th ACM Sigsoft Int'l Symp. on Foundations of Software Engineering. New York, 2001. 256−267. [doi: 10.1145/503271.503244]

[8] Yuan X. Feedback-Directed model-based GUI test case generaton [Ph.D. Thesis]. Maryland: University of Maryland, 2008.

[9] Mc Master S, Memon AM. Call stack coverage for GUI test-suite reduction. IEEE Trans. on Software Engineering, 2008,34(1): 99−115. [doi: 10.1109/TSE.2007.70756]

[10] Chen JC, Xue YZ, Zhao C. An approach for GUI testing based on event handler function. Ruan Jian Xue Bao/Journal of Software, 2013,24(12):2830−2842 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/4399.htm [doi: 10.3724/SP.J.1001.2013.04399]

[11] Yu DQ, Peng X, Zhao WY. Automatic refactoring method of cloned code using abstract syntax tree and static analysis. Journal of Chinese Computer Systems, 2009,30(9):1752−1760 (in Chinese with English abstract).

[12] Ammann P, Offutt J. Introduction to Software Testing. Cambridge: Cambridge University Press, 2008. 33−42.

[13] Pande HD, Landi WA, Ryder BG. Interprocedural def-use associations for C systems with single level pointers. IEEE Trans. on Software Engineering, 1994,20(5):385−403. [doi: 10.1109/32.286418]

[14] Zhang XF, Xu BW, Nie CH, Shi L. An approach for optimizing test suite based on testing requirement reduction. Ruan Jian Xue Bao/Journal of Software, 2007,18(4):821−831 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/18/821.htm

[15] Gupta A, Mishra N, Kushwaha DS. Rule based test case reduction technique using decision table. In: Proc. of the 2104 IEEE Int'l Conf. on Advance Computing. Los Alamitos: IEEE Computer Society, 2014. 1398−1405. [doi: 10.1109/IAdCC.2014.6779531]

[16] Gaurav Kumar, Pradeep Kumar Bhatia. Software testing optimization through test suite reduction using fuzzy clustering. CSI Trans. on ICT, 2013,1(3):253−260. [doi: 10.1007/s40012-013-0023-3]

[17] Yanping Chen, Robert L. Probert, Hasan Ural. Regression Test Suite Reduction Using Extended Dependence Analysis. In: Proc. of the 4th Int'l Workshop on Software Quality Assurance. New York: ACM Press, 2007. 62−69. [doi: 10.1145/1295074.1295086]

[18] Schroeder PJ, Korel B. Black-Box test reduction using input-output analysis. In: Proc. of the 2000 ACM SIGSOFT Int'l Symp. on Software Testing and Analysis. New York: ACM Press, 2000. 173−177. [doi: 10.1145/347636.349042]

[19] Jeffrey D, Gupta N. Test suite reduction with selective redundancy. In: Proc. of the 21st IEEE Int'l Conf. on Software Mainetance. Los Alamitos: IEEE Computer Society, 2005. 549−558. [doi: 10.1109/ICSM.2005.88]

[20] Jeffrey D, Gupta N. Improving fault detection capability by selectively retaining test cases during test suite reduction. IEEE Trans. on Software Engineering, 2007,33(2):108−123. [doi: 10.1109/TSE.2007.18]

[21] Harrold MJ, Gupta R, Soffa ML. A methodology for controlling the size of a test suite. ACM Trans. on Software Engineering and Methodology, 1993,2(3):270−285. [doi: 10.1145/152388.152391]

[22] Tallam S, Gupta N. A concept analysis inspired greedy algorithm for test suite minimization. In: Proc. of the 6th ACM SIGPLAN-SIGSOFT Workshop on Program Analysis for Software Tools and Engineering. New York: ACM Press, 2005. 35−42. [doi: 10.1145/1108768.1108802]

[23] Ma XY, Sheng BK, He ZF, Ye CQ. A genetic algorithm for test-suite reduction. In: Proc. of the IEEE Int'l Conf. on Systems, Man and Cybernetics. Los Alamitos: IEEE Computer Society, 2005. 133−139. [doi: 10.1109/ICSMC.2005.1571134]

[24] Li Z, Harman M, Hierons R. Search algorithms for regression test case prioritization. IEEE Trans. on Software Engineering, 2007, 33(4):225−237. [doi: 10.1109/TSE.2007.38]

[25] Smith AM, Kapfhammer GM. An empirical study of incorporating cost into test suite reduction and prioritization. In: Proc. of the 2009 ACM Symp. on Applied Computing. New York: ACM Press, 2009. 461−467. [doi: 10.1145/1529282.1529382]

[26] Heimdahl M, George D. Test-Suite reduction for model based tests: Effects on test quality and implications for testing. In: Proc. of the 19th Int'l Conf. on Automated Software Engineering. Los Alamitos: IEEE Computer Society, 2004. 176−185. [doi: 10.1109/ASE.2004.1342735]

[27] McMaster S, Memon A. Fault detection probability analysis for coverage-based test suite reduction. In: Proc. of the IEEE Int'l Conf. on Software Maintenance (ICSM 2007). Los Alamitos: IEEE Computer Society, 2007. 335−344. [doi: 10.1109/ICSM.2007.4362646]

© 中国科学院软件研究所 http://www.jos.org.cn

[28] Namin AS, Andrews JH. The influence of size and coverage on test suite effectiveness. In: Proc. of the Int'l Symp. on Software Testing and Analysis. New York: ACM Press, 2009. 173−177. [doi: 10.1145/1572272.1572280]

[29] Yu YB, Jones JA, Harrold MJ. An empirical study of the effects of test-suite reduction on fault localization. In: Proc. of the 26th Int'l Conf. on Software Engineering (ICSE 2008). New York: ACM Press, 2008. 201−210. [doi: 10.1145/1368088.1368116]

[30] Mei H, Hao D, Zhang LM, Zhang L, Zhou J, Rothermel G. A static approach to prioritizing JUnit test cases. IEEE Trans. on Software Engineering, 2012,38(6):1258−1275. [doi: 10.1109/TSE.2011.106]

附中文参考文献:

[10] 陈军成,薛云志,赵琛.一种基于事件处理函数的 GUI 测试方法.软件学报,2013,24(12):2830−2842. http://www.jos.org.cn/1000-9825/4399.htm [doi: 10.3724/SP.J.1001.2013.04399]

[11] 于冬琦,彭鑫,赵文耘.使用抽象语法树和静态分析的克隆代码自动重构方法.小型微型计算机系统,2009,30(9):1752−1760.

[14] 章晓芳,徐宝文,聂长海,史亮.一种基于测试需求约简的测试用例集优化方法.软件学报,2007,18(4):821−831. http://www.jos.org.cn/1000-9825/18/821.htm

陈军成(1980－),男,湖北天门人,博士,主要研究领域为GUI测试,协议一致性测试.

陶秋铭(1979－),男,博士,副研究员,主要研究领域为形式化方法,编译优化,软件测试.

薛云志(1979－),男,博士,高级工程师,主要研究领域为软件测试,编译优化.

赵琛(1967－),男,博士,研究员,博士生导师,主要研究领域为编译优化,软件测试,形式化方法.

© 中国科学院软件研究所 http://www.jos.org.cn