

文章编号:1001-9081(2010)10-2749-05

基于运行监测的图形用户界面自动化测试模型

张博刚¹, 张 威¹, 陈月宁², 廖飞雄¹

(1. 装甲兵工程学院 信息工程系, 北京 100072; 2. 第二炮兵指挥学院 通信系, 武汉 430012)

(04281147@bjtu.edu.cn)

摘 要:为提高 GUI 自动化测试的覆盖率、故障定位的速率和精度,以及检测由于时空变化导致的空间错误引起的故障,建立基于运行监测的 GUI 自动化测试模型。模型将 GUI 分为窗口框架层、界面元素层、功能结构层和运行记录层四层。窗口框架层描述 GUI 所有窗口,界面元素层描述用户输入,功能结构层提出功能覆盖准则,运行记录层通过插桩记录代码动态监测软件每一次执行时路径和各个窗口的运行状态,从而提高测试的覆盖率,并根据运行记录中窗口的总执行次数和正确的执行次数为可靠性计算提供依据。由于监测代码的运行情况,因此故障能够定位到代码级,提高故障定位的精度和速率。最后以记事本程序为例验证了模型的有效性。

关键词:窗口导航;有向图;运行监测;GUI 测试;运行记录;循环展开

中图分类号: TP311.5 **文献标志码:** A

Graph user interface automatic testing model based on software monitoring

ZHANG Bo-gang¹, ZHANG Wei¹, CHEN Yue-ning², LIAO Fei-xiong¹

(1. Department of Information Engineering, Academy of Armored Forces Engineering, Beijing 100072, China;

2. Department of Communications, The Second Artillery Command College, Wuhan Hubei 430012, China)

Abstract: In order to improve the coverage and efficiency of locating faults and defect the faults in running repetitiously in Graph User Interface (GUI) automatic testing, a GUI automated testing model based on software monitoring was proposed. The model divided GUI into four-layers: windows framework layer, interface element layer, functional structure layer and running record layer. The windows framework layer described all windows and controls in GUI, the interface element layer described all input events, the functional structure layer described the rule of functional coverage, and the running record layer monitored the software dynamically in each running trace and window through instrumentation codes, so as to increase the test coverage, provide basis for reliability calculation according to the total and correct operation in running record window. Finally, notepad was taken as the example to verify the model's efficiency.

Key words: window navigation; directed graph; software monitoring; GUI testing; running record; loop unwinding

0 引言

图形用户界面(Graph User Interface, GUI)测试是软件测试的一个重要环节。由于 GUI 反映了窗口、会话、功能菜单等一系列对象之间的相互组合^[1], GUI 自动化测试成为提高测试效率的最有效途径。目前 GUI 自动化测试方法包括录制回放模式、脚本编辑回放模式和数据驱动模式^[2-3];录制回放模式通过人工编写和录制测试用例,然后对被测软件进行回放的方式获取测试数据;脚本编辑回放模式通过运行测试员利用系统接口写好的交互脚本达到 GUI 自动化控制的目的,通常还要在程序中加入期望的检查点和测试结果;数据驱动模式将需要执行的测试操作信息传递给回放系统,是脚本编辑回放模式的扩展。三种模式都存在以下问题。

1) 没有直接对 GUI 的覆盖程度进行控制,因此可能会遗漏交互产生的重要事件序列,使得测试不够完全,不能够保证测试覆盖率。

2) 现有检测状态和验证内容生成方法的不足,每个事件序列对应的检测点和验证点也可能会有所缺失,导致软件运行失效时难以定位代码的出错位置;并且三种模式都是采用

回放测试用例执行软件的方式检测软件的故障,因此,故障只能定位到某一项功能,无法定位到具体出错的代码,定位的精度不够高。

3) 软件失效可能由自身的逻辑错误引起,也可能由时空变化导致的空间错误引起,尤其是现代的多程序设计技术,使得系统中存在很多并发进程或线程,这种并行性使得程序的运行具有动态性,导致某些异常难以再现。因此通过测试不能检测所有缺陷,也就不能为软件可靠性的评估提供依据。

为了解决上述问题,在研究软件运行监测的基础上,建立基于运行监测的 GUI 自动化测试模型。目前的软件运行监测技术大多集中于对软件功能的监测,例如 Csail 等人^[4]提出的软件监测框架 EndoScope 和 AspectJ 的监测方法^[5]等。但都与软件的内部结构联系不够紧密,由于软件内部结构的复杂性,使得基于功能的监测方法都有局限性,其主要原因是软件中的一项功能往往牵涉到很多的程序段,一旦某一项功能出现问题,难以检测出导致该功能失效的程序段,也就不能及时进行故障定位,导致程序修复时间延长。因此,通过插桩记录代码获取软件运行记录的方法进行软件内部结构监测,通过分析执行路径解决覆盖率的问题,通过分析各个模块中代

收稿日期:2010-04-12;修回日期:2010-05-18。

作者简介:张博刚(1986-),男,陕西西安人,硕士研究生,主要研究方向:软件测试; 张威(1968-),男,教授,博士,主要研究方向:软件工程、软件测试、军事信息系统; 陈月宁(1983-),女,河北衡水人,硕士研究生,主要研究方向:软件工程; 廖飞雄(1985-),男,湖南湘乡人,硕士研究生,主要研究方向:软件测试。

码的执行状态解决快速故障定位和定位精度的问题,通过监测每一次执行中各个窗口的状态解决由于时空变化导致的空间错误引起的故障以及难以再现的问题。

1 GUI 自动化测试模型建立

GUI 运行过程中窗口界面的运行,实际是以用户操作触发 GUI 界面中的功能控件,从而驱动窗口界面的改变与运转并实现特定的功能,其中控件触发事件就是用户输入的按键字符。因此,为了分析方便,便于 GUI 测试、生成运行记录和故障定位,将用户操作、窗口控件等分为四层,依次为窗口框架层、界面元素层、功能结构层和运行记录层。GUI 自动化测试模型如图 1 所示。

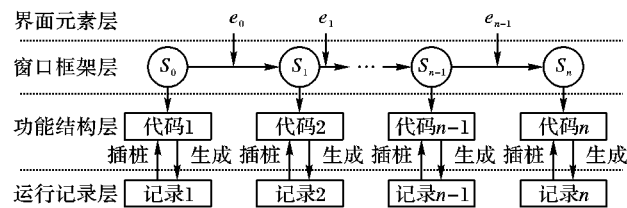


图1 GUI 自动化测试模型结构示意图

窗口框架层描述 GUI 的窗口界面和各种窗口控件以及为了实现某一功能各个窗口的顺序组合。窗口控件通过响应输入事件实现功能,完成控件之间状态的转移。

功能结构层为测试用例的生成提供依据,测试用例的生成按照功能寻找测试路径,最后通过测试用例序列执行测试路径完成 GUI 测试。功能结构层结合 GUI 软件的功能需求实现,同时也是保证 GUI 软件测试覆盖率的基础。

界面元素层表示各个控件接受的用户输入,是实现功能的测试用例,是组成测试用例序列的基本元素。

运行记录层是在窗口功能实现的代码处插入记录代码,程序运行时记录运行信息,动态监测 GUI 软件每一次的运行情况。

1.1 窗口框架层

窗口框架层描述 GUI 的窗口界面、控件以及控件间的组合,并能够体现用户输入对控件状态的改变,是测试路径生成的基础。因此,采用窗口导航有向图 (Directed Graph for Window Navigation, DGWN) 描述 GUI 窗口框架,为了描述方便,将 GUI 中窗口界面和所有控件统称为窗口。

定义 1 窗口导航有向图。窗口导航有向图 G 是有向连通多重图 $G = \langle S, E, \psi \rangle$, S 表示窗口集合, E 表示弧的集合, $\psi: E \rightarrow S \times S$ 是关联函数。每个顶点 $s \in S(G)$ 代表 GUI 中的一个窗口, 每条弧 $e \in E$ 代表一个用户输入, $\psi(e) = \langle u, v \rangle$, u 是执行用户输入前处于激活状态的顶层窗口, v 是执行用户输入状态后处于激活状态的顶层窗口。

在窗口导航有向图中,窗口只能通过用户输入转移焦点,所以用户输入只能与窗口有直接关系。在集合 S 中不参与变化的窗口表现为孤立的窗口,不引起窗口焦点转移的用户输入表现为孤立的输入。软件运行时,孤立窗口为不可达窗口,孤立输入是没有意义的输入,因此进行 GUI 测试时,不考虑孤立路径和孤立输入的情况。

定义 2 测试路径。一条起点为主窗口到终点为功能实现窗口的窗口导航有向图路径称为 GUI 的测试路径。即执行完一条路径实现一项功能。窗口导航有向图中任何一条路径都是窗口和用户输入的交替序列,表示为 $s_0 e_1 \dots s_k e_k \dots$

$e_{n-1} s_n$ 。

定义 3 前置窗口。指能够接受用户输入,并完成焦点向当前窗口转移的窗口。

定义 4 后置窗口。指当前窗口在当前用户输入的触发下所能到达的窗口。

定义 5 最底层窗口。为了实现某一个功能,测试路径中的最后一个窗口称为最底层窗口。最底层窗口的后置窗口为空。

测试路径从主窗口到达最底层窗口,测试路径中用户输入最终要转换为按键字符,生成测试用例序列以驱动 GUI 的运行。

窗口导航有向图的生成是通过静态分析 GUI 程序和界面生成所有窗口,结合功能需求和设计文档,对所有窗口进行顺序组合实现的。

1.2 功能结构层

功能结构层是测试用例序列生成的基础,是 GUI 的覆盖准则。在自动化功能测试用例设计和生成的过程中,覆盖准则的设计是为了防止软件输入模型状态空间的爆炸,给测试用例的生成空间限定范围和制定标准,同时也是为了保证所有功能都被覆盖到。目前的覆盖准则包括基于模型的覆盖准则^[6]、基于事件的覆盖准则^[7]以及 n 长度用户输入序列覆盖准则^[8]等。

图形化用户界面 GUI 反映了窗口及其组合在用户不同输入下的各种不同状态,不同的输入在不同窗口下有着不同的功能,因此其状态数是巨大的。但是从窗口导航有向图的角度看,仅仅需要关注哪一个事件导致了窗口的变化;从用户的角度看,只需要关注是否能够通过打开一个特定的窗口或点击一个下拉菜单,从而实现期望的功能。

因此,本文中测试用例的生成空间是用户需要的功能和软件能够实现的功能。覆盖原则是按照功能模型通过遍历窗口导航有向图覆盖每一条路径,即覆盖每一个功能。测试用例生成之前首先根据 GUI 软件需求、总体设计说明书、详细设计说明书分析出 GUI 的所有功能,通过静态分析,依次遍历所有功能,每个功能依据黑盒测试理论,划分测试用例。最后查询没有遍历到的窗口,再生成测试用例,覆盖到每一个窗口。

窗口导航有向图中一条测试路径就是一个完整的功能,因此,遍历所有功能就是遍历窗口导航有向图的所有路径。只要窗口导航有向图中所有路径被执行过,GUI 中所有功能也将被覆盖。

1.3 界面元素层

界面元素层是用户的输入,是两个窗口之间状态转移的测试用例。由于用户的输入 $\psi(e) = \langle u, v \rangle$, 是在用户输入前处于激活状态窗口的基础上接受输入,从而转向另一个窗口。因此,窗口导航有向图中的用户输入使用操作符表示。操作符指定用户输入的前置条件和后置条件。操作符表示成三元组 $\langle e, Pre, Eff \rangle$, 其中 e 是用户输入的名称及参数; Pre 是用户输入的前提,即前置窗口; Eff 是用户输入的结果,即后置窗口。根据窗口导航有向图,可以得到 GUI 在接受用户输入后的状态变化情况。三元组 $\langle e, Pre, Eff \rangle$ 描述如下:

- 1) $Pre(e) = \{pre_1(e), pre_2(e), \dots, pre_m(e)\};$
- 2) $Eff(e) = \{eff_1(e), eff_2(e), \dots, eff_m(e)\};$
- 3) $Eff_j(e) = map(Pre_j(e)); j = [1 \dots m].$

事件 e 有一组前置窗口和一组后置窗口,前置窗口组中的每个前置窗口都在后置窗口组中有一个后置窗口和它对应。同一输入的前置窗口和后置窗口都以对象属性的形式表示。

根据窗口导航有向图可以得到 GUI 在接受用户输入 e 后的状态变化情况,通过分析测试路径,可以得到为实现某一功能所需要的所有用户输入及其执行顺序,构成测试用例序列。

定义 6 测试用例序列。在窗口导航有向图中,任何一条路径都是窗口和用户输入组成的序列 $s_m e_m \cdots s_k e_k \cdots e_{n-1} s_n$, 其中:可达状态 $s_m = s_0 e_0 e_1 \cdots e_{m-1}$, s_0 是 GUI 软件刚启动时的主窗口, $e_m e_{m+1} \cdots e_{n-1}$ 是在窗口 s_m 下可执行的用户输入序列;期望状态 $s_k = s_{k-1} e_{k-1} (m+1 \leq k \leq n)$ 。将 $s_e = s_0 e_0 e_1 \cdots e_{n-1}$ 中 $e_0 e_1 \cdots e_{n-1}$ 定义为一个完整的测试用例序列,其中 s_e 表示最底层窗口。

在窗口导航有向图基础上一个测试用例序列执行一条路径,实现一个功能。因此,通过深度优先搜索算法 (Depth First Search, DFS) 生成从初始状态到各个可达状态的测试用例序列,驱动各个路径的执行。

由于窗口导航有向图中有循环的存在,应用 DFS 时需要将循环进行展开。目前的方法包括 GenPath 算法^[9],根据给定的循环 loop 和限制循环最大执行次数 maxTimes,生成所有可行的检验路径,并返回检验路径的集合,以及通过生成基本路径集^[10]覆盖所有状态。窗口导航有向图中,每一个功能就是一条完成的路径,因此,循环被执行一次就可以保证功能的实现,参考 GenPath 算法,将循环最大执行次数设为 1,同时将循环展开,由于循环中两个状态转移的用户输入并不相同,因此循环展开后将循环的开始窗口作为最底层窗口。

1.4 运行记录层

运行记录层通过在实现窗口功能的程序中插桩记录代码,在程序运行时获得运行记录信息。测试完成后通过分析记录信息,实现 GUI 的故障定位和覆盖率分析。在窗口导航有向图中,正常情况下,运行记录中的各条路径是完整的,即测试路径中所有窗口都被执行,运行记录中包含每一条路径从主窗口到最底层窗口所有窗口在测试用例驱动下依次执行的信息。如果软件运行到某一条路径中的某个窗口时出现故障,运行记录中就会有不完整的路径信息,通过和窗口导航有向图进行对比,可将故障定位到某条路径中的某个窗口,通过运行记录分析实现该窗口功能代码的运行状态,可将故障定位到具体代码,提高故障的速率和精度。

在测试覆盖率 100% 的情况下,窗口框架层中的所有路径都被执行。通过分析运行记录,可以得到软件运行的执行路径,通过将生成的执行路径和窗口框架层中的路径进行对比,可以发现还没有执行过的路径,计算出路径覆盖率。同时通过设计测试用例序列执行没有执行过的路径,以提高测试的覆盖率。

由于监测软件每一次的运行情况,因此可以检测到软件多次运行之后由于时空变化出现的故障,并且这些故障难以再现。这样可以提供多次执行之后,软件的正确执行次数和出错次数,进而为软件可靠性的评估提供依据。

通过以上四层可以看出,窗口框架层提供测试路径,功能结构层提供测试覆盖准则,界面元素层提供测试输入,最终三者构成了测试用例,并自动转化为测试用例序列,驱动被测 GUI 软件的自动化运行;运行记录层提供插桩代码,测试时生成软件运行记录信息,动态监测 GUI 软件的执行情况。

2 软件运行监测

软件运行监测通过插桩记录代码,获取软件的运行记录,从而动态监测软件的执行情况。考虑软件中语句的复杂性,记录每一条语句的运行信息将会非常复杂,因此插桩记录每一个函数的运行信息,在实现窗口功能的所有函数的入口和出口处分别插桩,记录该函数的入口和出口信息。

RunningRecord	
PK	NodeID
PK	Type
	WinID
	InID
	PreID
	Times

图 2 运行记录格式

2.1 运行记录生成

为了记录和分析运行信息,设计运行记录格式如图 2 所示。

NodeID: 当前函数的 ID。

WinID: 所属窗口的 ID。

InID: 触发该窗口的输入名称。

PreID: 执行本函数的前一个函数。

数。

Type: 本条记录是函数的入口或者出口, input 表示入口信息, output 表示出口信息。

Times: 该窗口执行的次数, 初时状态为 0。

运行记录是通过在源程序中插桩记录代码,程序运行时生成的。记录生成的关键是插桩点的选择和自动插桩算法的实现。首先对插桩点的选择,由于是基于功能的 GUI 测试,因此在窗口导航有向图的基础上,可以通过静态分析查找源程序中实现窗口功能对应的代码。

程序运行时,按照设计的记录格式将记录信息输出到文件或者数据库中。自动插桩算法目前已经研究的比较成熟,既可以借助 Lex 和 Yacc^[11]实现,也可以通过静态分析,在代码的入口和出口处插桩^[12]。本文采用静态分析法,在实现窗口功能的所有函数的入口和出口处分别插入记录代码。插桩代码如下。

1) 在程序入口处插桩,建立连接数据库的代码。

```
RunLog. Open;
RunLog. Append; //用 Access 存储运行记录
信息存储以函数 ID 为主键,记录每一个函数的执行信息。
```

2) 在各个函数的入口和出口处插入记录代码。

```
RunLog. FieldByName( 'NodeID' ). Value = sNodeID;
RunLog. FieldByName( 'InID' ). AsString = sInID;
RunLog. FieldByName( 'WinID' ). Value = sWinID;
RunLog. FieldByName( 'PreID' ). AsString = sPreID;
RunLog. FieldByName( 'Type' ). Value = sType;
RunLog. FieldByName( 'Times' ). AsString =
    inttostr( strtoint( times ) + 1 );
RunLog. Post;
RunLog. Close;
```

通过插入以上记录代码,在 GUI 软件运行时,可将运行信息写入运行记录数据库,为软件运行结果的分析提供依据。

2.2 故障定位

在正常情况下,运行记录中的各条路径是完整的,即每个窗口都有相应的入口和出口信息,如果软件运行到某一条路径中的某个窗口时出现故障,则运行记录中就会有不完整的路径信息,与窗口导航有向图进行对比,即可将故障定位到该窗口。然后通过分析运行记录,查找实现该窗口功能的所有函数的记录信息,如果某个函数只有入口信息,没有出口信息,则可以将故障定位到该函数,相对于现有的软件测试方法和 GUI 自动化测试模型来说,即可以将故障定位到功能级,

还可以将故障定为到代码级,提高故障定位的精确度。

由于运行记录中记录了函数的执行次数以及每次执行的状态信息,多次执行之后,通过分析运行记录可以得到在执行到第几次时出现故障以及出现故障时的状态信息,为软件的修复提供依据。同时通过计算入口信息中 Times 和出口信息中 Times 之差得到该函数一共出现故障的次数,此时入口信息中 Times 表示为该函数总共执行的次数,出口信息中 Times 表示该函数正确执行的次数,从而得到该函数的平均无故障时间 (Mean Time to Failure, MTTF), 计算可靠性为 $\text{Output (times)} / \text{Input (times)} \times 100\%$, 进而为计算该窗口甚至一条路径和系统的可靠性提供依据。这是其他 GUI 自动化测试模型所不能实现的。

2.3 覆盖率分析

覆盖率是通过将执行的路径与窗口导航有向图进行比较,执行路径数除以总的路径数得到的。其中最主要的是根据软件运行记录生成执行路径。首先搜索组成窗口的所有函数,从记录格式可知每一个函数的运行记录中都存储了前一个函数的 ID,因此,生成执行路径时首先查找到主窗口 s_0 , 然后通过 PreID 依次查询该函数的执行后的下一个函数,直到结果为空,表示搜索到该窗口的所有函数。然后通过运行记录中的输入 InID, 依次查询输入的前置窗口和后置窗口,从而搜索到一条完整路径。流程如图 3 所示。

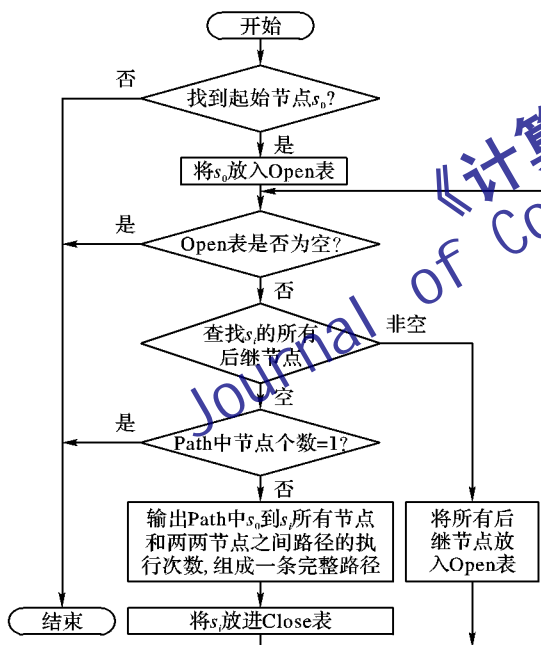


图3 执行路径生成流程

3 实例分析

以编写程序模拟记事本中文件的新建、打开和保存功能为例,窗口导航有向图如图 4 所示。图中窗口和用户输入描述如下:

- s_0 表示记事本模拟程序的主窗口,即为初始窗口。
- e_0 表示按下 ALT + F 键。
- s_1 表示弹出“文件”的下拉菜单后的窗口。
- e_1 表示按下 ESC 键。
- e_3 表示按下 O 或者 CTRL + O 键。
- e_4 表示按下 N 或者 CTRL + N 键。
- e_5 表示按下 S 或者 CTRL + S 键。
- s_2 表示弹出文件“打开”的窗口。

s_3 表示返回到新建文件的主窗口。

s_4 表示弹出文件“保存”的窗口。

e_2 表示按下 ESC 键。

e_7 表示选择文件,输入文件名称,按下 ALT + O 键。

e_6 表示按下 ESC 键。

e_8 表示选择保存位置,输入文件名称,按下 ALT + S 键。

s_5 表示返回到打开文件的主窗口。

s_6 表示返回到保存文件的主窗口。

3.1 测试用例序列生成和执行

将图 4 中的循环展开后结果如图 5 所示。以循环 $s_0 e_0 s_1 e_1 s_0$ 为例,展开后 s_0 为最底层窗口。循环展开之后,通过 DFS 遍历窗口导航有向图生成测试用例序列。以实现文件保存功能为例,在图 5 中,通过 DFS 生成的测试用例序列是 (e_0, e_5, e_8) , 用户的输入分别是 $(\text{ALT} + \text{F}, \text{CTRL} + \text{S}, 111, \text{ALT} + \text{S})$, 其中 111 是在 e_8 操作时,输入的文件名称。测试进行时通过测试用例序列驱动被测 GUI 软件自动运行,实现自动化测试。如果窗口 s_0 执行用例 e_0 之后,并没有到达窗口 s_1 , 该测试用例是否继续执行,即测试用例序列执行时出现异常情况怎么处理,因此需要对每一个用例执行的开始条件和终止条件进行设定。

首先判断是不是所有测试用例序列 SeqID_i 都被执行,如果执行完,结束整个过程;如果没有执行完,开始执行 SeqID_i 中的用例 e_i 。如果该用例的后置窗口 $\text{Eff}(e_i)$ 不能够正常到达,表示该用例不能正常执行,该测试用例序列终止,转而执行下一个测试用例序列。最后判断该用例的后置窗口是否是最低层窗口,如果是,执行另一个测试用例序列;如果不是,执行测试用例序列中的下一个用例。通过以上过程实现 GUI 测试的自动化执行。

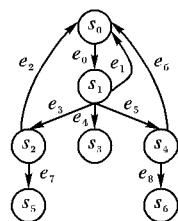


图4 窗口导航有向图

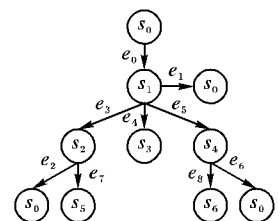


图5 循环展开后的窗口有向图

3.2 覆盖率分析和故障定位

通过图 3 的执行路径生成算法分析软件运行记录,得到的执行路径包括 $s_0 s_1 s_2 s_0$, $s_0 s_1 s_2 s_5$, $s_0 s_1 s_0$ 和 $s_0 s_1 s_4 s_6$, 对照窗口导航有向图,路径 $s_0 s_1 s_3$ 和 $s_0 s_1 s_4 s_5$ 没有被执行,如图 6 所示。图中节点表示软件执行的窗口,粗线表示运行覆盖的路径及窗口,因此覆盖率为 $4/6 = 66.7\%$, 为了提高覆盖率,通过测试用例序列 (e_0, e_4) 和 (e_0, e_5, e_6) 执行没有覆盖到的路径,从而提高覆盖率。

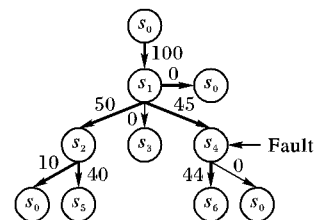


图6 窗口导航有向图路径分析

同时,通过统计发现窗口 s_4 执行了 45 次, s_6 执行了 44 次, 即路径不完整, s_4 窗口有一次没有出口信息,通过分析运行记录中窗口 s_4 的实现函数 41、42、43、44 的运行记录信息,在执

行到第 45 次时,函数 41、42、43 出口和入口信息完整,函数 44 没有出口信息,运行记录如表 1 所示,因此可以将故障定位到函数 44。

Nodeid	Type	*InID	InID	PreID	Times
11	input	1	0	01	100
11	output	1	0	01	100
41	input	4	5	11	45
41	output	4	5	11	45
42	input	4	5	41	45
42	output	4	5	41	45
43	input	4	5	42	45
43	output	4	5	42	45
44	input	4	5	43	45
					0

图 7 部分运行记录

通过监测软件运行 100 次可以发现,窗口 s_4 中的函数 44 总共执行了 45 次,在执行第 45 次时发生故障,因此可以计算其可靠性为 $44/45 = 97.8\%$,进而为计算窗口 s_4 的可靠性以及文件保存功能的可靠性和记事本模拟程序的可靠性计算提供依据。相对于其他 GUI 自动化测试模型来说,很难检测到软件多次运行之后出现的故障,并且这些故障难以再现,因此,无法对软件可靠性的评估提供数据支持,并且对出现的故障无法定位到代码中的具体函数。

4 结语

基于运行监测的 GUI 自动化测试模型将 GUI 分为窗口框架层、界面元素层、功能结构层和运行记录层。窗口框架层采用窗口导航有向图描述,反映为了实现 GUI 功能在用户输入下所有的控件的顺序组合,在功能结构层提出的覆盖准则的要求下,结合界面元素层中的用户输入,通过对窗口导航有向图中的环进行处理,采用 DFS 方法自动生成测试用例序列。测试开始时,通过测试用例序列驱动被测软件自动运行。运行记录层通过插桩记录每一条路径的执行情况和实现窗口功能的所有函数的执行状态,进行动态监测和故障定位,为提高故障定位的速率和精度以及覆盖率提供依据。由于监测软件代码中每一个函数的执行情况,因此在多次执行后,可以检测出由于时空变化导致的空间错误引起的故障,进而得到每一个函数在多次执行后的失效次数,为计算可靠性提供数据支持。

(上接第 2748 页)

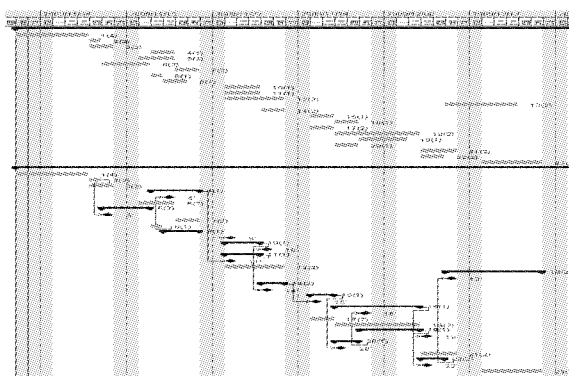


图 6 甘特图

参考文献:

- [1] Software Engineering Institute (SEI). CMMI for Development, Version 1.2 [S/OL]. Carnegie Mellon: Software Engineering Institute, 2006 [2009-12-15]. <http://www.sei.cmu.edu/reports/06tr008.pdf>.
- [2] Project Management Institute (PMI). A guide to the project management body of knowledge [S]. 3rd ed. Pennsylvania: Project Management Institute, 2004.

参考文献:

- [1] BROOKS P A, MEMOEN A M. Automated GUI testing guided by usage profiles [C]// Proceedings of 22th IEEE/ACM International Conference on Automated Software Engineering. Washington, DC: IEEE, 2007: 333-342.
- [2] 李翔, 高建华. 一种基于事件关系的图形用户界面程序测试方法[J]. 小型微型计算机系统, 2005, 26(4): 671-675.
- [3] 吴恒山, 王金红. 基于界面状态有效性的 GUI 自动测试模型[J]. 华中科技大学学报: 自然科学版, 2004, 32(12): 34-37.
- [4] CSAIL A C. Performance profiling with endoscope, an acquisitional software monitoring framework [J]. Proceeding of the VLDB Endowment, 2008, 1(1): 42-53.
- [5] ALLAN C, AVGUSTINOV P, CHRISTENSEN A S, et al. Adding trace matching with free variables to AspectJ [C]// Proceedings of the 20th Annual ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications. New York: ACM, 2005: 345-364.
- [6] 叶茂, 高海昌, 冯博琴, 等. 基于窗口导航有向图的 GUI 测试覆盖准则[J]. 西南交通大学学报, 2006, 41(4): 476-480.
- [7] OSTRAND T, ANODIDE A, FOSTER H, et al. A visual test development environment for GUI systems [C]// Proceedings of the ACM SIGSOFT International Symposium on Software Testing and Analysis. New York: ACM, 1998: 82-92.
- [8] MEMON A M, SOFFA M L, POILACK M E. Coverage criteria for GUI testing [C]// Proceedings of the 8th European Software Engineering Conference Held Jointly with 9th ACM SIGSOFT International Symposium on Foundations of Software Engineering. New York: ACM, 2001: 256-267.
- [9] 阮辉, 严俊, 张健. 基于路径分析的死循环检测[J]. 计算机学报, 2009, 32(9): 1750-1758.
- [10] 张广梅, 李晓维, 韩丛英. 路径测试中基本路径集的自动生成[J]. 计算机工程, 2007, 33(22): 195-197.
- [11] 晏华, 袁海东, 尹立孟. 代码自动插装技术的研究与实现[J]. 电子科技大学学报, 2002, 31(1): 62-66.
- [12] 王斌, 张伟, 段见飞, 等. 基于软件运行记录的测试方法[C]//第三届全国软件测试会议与移动计算、栅格智能化高级论坛论文. 山西: 电脑开发与应用, 2009: 79-82.

- [3] 武占春, 王青, 李明树. 一种基于 PDCA 的软件过程控制与改进模型[J]. 软件学报, 2006, 17(8): 1669-1680.
- [4] KIM K, de la GARZA J M. Evaluation of the resource-constrained critical path method algorithms [J]. Journal of Construction Engineering and Management, 2005, 131(5): 522-532.
- [5] LU MING, LI HENG. Resource-activity critical-path method for construction planning [J]. Journal of Construction Engineering and Management, 2003, 129(4): 412-420.
- [6] 万静, 何月娇, 易军凯. 面向任务的软件过程控制模型[J]. 计算机过程与应用, 2009, 45(22): 56-58.
- [7] EASA S M. Resource leveling in construction by optimization [J]. Journal of Construction Engineering and Management, 1989, 115(2): 302-316.
- [8] CHAN W-T, CHUA D K H, KANNAN G. Construction resource scheduling with genetic algorithms [J]. Journal of Construction Engineering and Management, 1996, 122(2): 125-132.
- [9] 朱颢东, 钟勇. 一种改进的模拟退火算法[J]. 计算机技术与发展, 2009, 19(6): 32-35.
- [10] RUSSELL S, NORVING P. 人工智能——一种现代方法[M]. 2版. 北京: 人民邮电出版社, 2004.
- [11] WIEST J D. Some properties of schedules for large projects with limited resources [J]. Operations Research, 1964, 12(3): 395-418.