·体系结构与软件技术· 文章编号：1000-3428(2017)02-0111-09 文献标志码：A 中图分类号：TP311.52

一种实时性缺陷定位方法及其可视化实现

李志敏[1]，殷蓓蓓[1]，张 萍[2]，王纪兵[2]，王 宾[1]，张金鹏[2,3]

（1. 北京航空航天大学 自动化科学与电气工程学院，北京 100191；2. 中国空空导弹研究院，河南 洛阳 471009；
3. 航空制导武器航空科技重点实验室，河南 洛阳 471009）

摘 要： 为满足飞行控制软件的强实时性要求，针对实时嵌入式软件提出一种实时性缺陷定位方法，并开发可视化工具提高其自动化程度。建立实时性缺陷定位模型，分别在软件模块及函数粒度上定义软件单元的可疑度。在计算可疑度时，利用模块实际执行时间与基准执行时间的差异、成功用例与失败用例中函数执行时间的差异以及模块与函数的调用关系，分别进行模块和函数级别的实时性缺陷定位。仿真结果显示，包含缺陷的模块或函数均具有较高的可疑度值，验证了该方法的有效性。

关键词： 飞行控制软件；实时性缺陷；执行时间；缺陷定位；可疑度；可视化工具

中文引用格式： 李志敏，殷蓓蓓，张 萍，等. 一种实时性缺陷定位方法及其可视化实现[J]. 计算机工程，2017，43(2)：111-119.

英文引用格式： Li Zhimin，Yin Beibei，Zhang Ping，et al. A Real-time Fault Localization Method and Its Visualization Implementation[J]. Computer Engineering，2017，43(2)：111-119.

A Real-time Fault Localization Method and Its Visualization Implementation

LI Zhimin[1]，YIN Beibei[1]，ZHANG Ping[2]，WANG Jibing[2]，WANG Bin[1]，ZHANG Jinpeng[2,3]

（1. School of Automation Science and Electrical Engineering，Beihang University，Beijing 100191，China；
2. China Airborne Missile Academy，Luoyang，Henan 471009，China；
3. Key Laboratory of Aviation Guided Weapon Aviation Science，Luoyang，Henan 471009，China）

【Abstract】 To meet strong real-time demand of flight control software，a real-time fault localization method for embedded software is proposed，and a visualization tool is developed to improve the automation degree of this method. A real-time fault localization model is established to define the suspiciousness on the two levels of software module and function respectively. While computing the suspiciousness，the difference between the actual execution time of modules and the criterion time，the difference between the function execution time in successful test cases and failure ones，and the call relations between modules and functions are used to give real-time fault localization algorithms on the module level and the function level. Simulation results indicate that modules and functions containing faults tend to have higher suspiciousness，which demonstrates the effectiveness of the proposed method.

【Key words】 flight control software；real-time fault；execution time；fault localization；suspiciousness；visualization tool

DOI： 10.3969/j.issn.1000-3428.2017.02.019

0 概述

作为飞控系统的控制核心，飞控软件是安全关键软件，它不仅对可靠性有着十分严格的要求，对实时性的要求同样很高。飞控软件完成主任务的周期一般严格限制在一定时间内，每个周期内必须完成输入信号的采集、控制律的计算、控制信号的输出以及各种任务的数据处理等[1]。然而由于各种实时性缺陷的存在，往往使得飞控软件不能满足这些实时性要求，从而导致系统失效乃至事故。因此，需要对实时性缺陷进行定位和剔除，以提高飞控软件的实时性。

软件缺陷定位是程序调试中一个非常昂贵、冗长且耗时的过程。自 20 世纪 70 年代以来，为了提

基金项目： 航空科学基金(20130151001)。

作者简介： 李志敏(1990—)，女，硕士研究生，主研方向为软件可靠性测试；殷蓓蓓，讲师；张 萍，高级工程师；王纪兵，工程师；王 宾，硕士研究生；张金鹏，博士研究生。

收稿日期： 2015-11-06 **修回日期：** 2016-01-06 **E-mail：** muxinmei@buaa.edu.cn

高测试人员定位缺陷的效率,出现了许多缺陷定位方法,主要包括基于程序切片的方法[2-3]、基于程序谱的方法[4-5]、基于程序状态的方法[6-7]、基于模型的方法[8-9]、基于数据挖掘的方法[10]等。相对于其他缺陷定位方法,基于程序谱的方法具有计算复杂度低及所需先验信息少的优点,因此成为了传统缺陷定位技术中的研究热点。该类方法所用到的频谱信息又分为可执行语句执行频谱、谓词次数频谱、方法调用顺序频谱以及其他信息的频谱,研究较多的是基于前两种频谱的方法,经典的算法有基于可执行语句的 Tarantula 算法[11]和基于谓词的 Sober 算法[12]等。

基于程序谱思想,40 多种可疑度计算公式陆续被提出,然而传统的缺陷定位技术大多都是针对软件功能性的缺陷,关于实时性缺陷定位的研究较少。目前也有一些针对于飞控软件实时性验证和测试方法的研究,如文献[13]在 Windows 操作系统下构建了一套高实时性的测试系统方案,实现了对无人机飞控软件的实时监测和测试。但这些研究多针对特定的飞控软件且侧重于对软件实时性的测试,对于如何定位实时性缺陷并没有进行深入研究。

本文借鉴基于程序谱的缺陷定位方法,分别设计模块和函数级别的实时性缺陷定位方法。模块级别的实时性缺陷定位是对比模块在失败用例中的实际平均执行时间与其基准执行时间的差异,差异越大,则其包含实时性缺陷的可疑度越大;函数级别的实时性缺陷定位则根据是否存在成功用例分为 2 种情况:若执行的测试用例中存在成功用例,则比较成功用例和失败用例中函数平均执行时间的差异,差异越大,函数可疑度越大;若全为失败用例,则首先计算出模块的可疑度,再根据模块与函数的调用关系,利用统计学方法计算出函数可疑度值。

为进一步提高软件实时性缺陷定位的自动化程度,本文开发一种实时性缺陷定位结果的可视化工具。该工具根据可疑度对可疑代码进行着色,同时可视化显示实时性缺陷定位分析结果,目的是以直观的形式显示可疑代码的位置以及可疑度计算结果,进一步提高测试人员调试程序的效率。

1 实时性缺陷定位模型

首先,本文由传统软件可靠性中的软件失效和缺陷的定义引出软件的实时性失效和实时性缺陷 2 个概念:实时性失效即程序未在限定的时间内完成任务;实时性缺陷即程序中导致其发生实时性失效的原因,如不合理的软件结构、错误的代码以及硬件故障等。为建立合理的实时性缺陷定位模型,本文从实时性缺陷种类、实时性失效标准以及实时性缺陷定位粒度 3 个方面对飞控软件的实时性缺陷问题进行简要分析。

1.1 实时性缺陷种类

对于嵌入式实时软件,常见的实时性失效模式是程序执行超时,即程序未在设定的周期时间内完成任务。造成程序执行超时的实时性缺陷可能有:主中断受到其他中断的影响,导致某输出信号未在规定时间内输出;程序中存在死循环,导致程序执行时间过长;程序中存在某些函数,如反三角函数,其执行时间过长;信息传输存在故障等。在仿真实验中,本文用延时函数来模拟这些实时性缺陷。

1.2 实时性失效标准

如图 1 所示,飞行器控制律程序在 t_1 时刻采集到输入信号飞行器姿态值,在 t_2 时刻输出计算结果控制指令,则控制律执行一个周期的时间为 $t = t_2 - t_1$。设定飞控程序的实时性要求为控制律执行时间不能超过 T,在实际情况中,往往可以由具体的任务需求得到 T 的值。

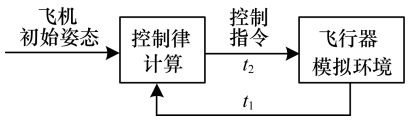

图 1 飞控软件结构

本文规定若控制律计算未在 T 时间内完成,则将这次计算称为失效周期。飞控程序完成一次任务往往需要循环执行多次控制律,因此,执行一个测试用例会包含 n 个控制律执行周期,若执行结果中没有失效周期,则称该测试用例为成功用例;否则称为失败用例。

1.3 实时性缺陷定位粒度

实时性缺陷定位粒度可以分为模块级别和函数级别 2 种。这里飞控软件中的函数是指为实现某一功能所定义的函数,这些函数往往用于计算所需变量的数值或期望值,如计算飞机与下滑线的距离的函数 *GetJuli*( )、计算期望升降舵指令的函数 *KeepHight*( )等。在计算函数执行时间时,本文指的是函数本体的执行时间,不包括它所调用函数的运行时间,因此,不需要考虑函数之间是否存在调用关系。本文把飞控软件中执行同一阶段任务所调用的函数及代码作为一个模块,如在自动着陆控制软件中,控制律程序则可分为盘旋模块、下滑模块、拉平模块、着陆模块等。每个模块会调用若干个函数,不同的模块也会调用相同的函数。

模块级别的实时性缺陷定位,是把每个模块看作一个软件单元,其目的是为程序中调用的模块计算可疑度值,从而帮助测试人员快速找到包含实时

性缺陷的模块。在工程实践中,对于模块执行时间的限定往往是已知的,称其上限为基准执行时间,因此,本文假定模块的基准执行时间是已知的。在仿真实验中,本文将覆盖所有路径且包含各类输入情况的测试用例集加载到被测程序中。运行一次程序会执行同一模块多次,因此,可以得到模块在该测试用例下执行一次的平均时间,称之为平均执行时间。将所有测试用例下平均执行时间的最大值作为其基准执行时间的估计值。

函数级别的实时性缺陷定位,是把每个函数看作一个软件单元,其目的是为每个函数计算可疑度值,从而帮助测试人员快速找到包含实时性缺陷的函数,其定位粒度比模块级别小,因此,可以更准确地定位到实时性缺陷。

2　实时性缺陷定位方法

借鉴基于谓词的软件缺陷定位算法的设计思想,本文针对模块和函数 2 种级别,分别提出相应的实时性缺陷定位方法。

2.1　基于谓词的缺陷定位方法

基于程序谱的缺陷定位方法,其基本设计思想是一个程序单元在运行成功与运行失败的执行信息的差异越大,其包含缺陷的可疑度越大。其中程序单元可以是语句、谓词(例如程序中 if, for, while, return 这样具有逻辑判断功能的语句)、分支或函数。程序谱则是指所用到的程序单元的执行信息,如执行次数、执行结果等。基于谓词的方法是通过比较谓词的成功执行谱和失败执行谱的差异来得到谓词可疑度,其中执行谱的信息是谓词的执行次数,主要方法有 CBI 算法[14]、Sober 算法等。

CBI 算法为了识别与缺陷相关的谓词,通过比较谓词在程序运行成功与失败时是否都被判断为“真”来确定谓词 P 的可疑度,具体计算公式如下:

$$Sus(P)=(F(P))/(S(P)+F(P)) \tag{1}$$

其中,$F(P)$和 $S(P)$分别为谓词 P 被判断为“真”时,运行失败和成功的测试用例的数目。

利用式(1)可以计算出程序中所有谓词的可疑度。CBI 算法的缺点是只适用于谓词最多在程序中执行一次的情况,但实际上,谓词在程序运行一次的过程中可能会被多次执行。

Sober 算法则在 CBI 算法的基础上,考虑了谓词的执行次数,并给出了“evaluation-bias”的定义,用来描述谓词 P 在一个测试用例中被判断为“真”的概率。通过比较每个谓词在成功的和失败的运行中被判定为“真”的实际概率之间的差异,来计算每个谓词的可疑度,差异越大,可疑度越大,计算公式如式(2)和式(3)所示。

$$\Phi(P)=n_t(P)/(n_t(P)+n_f(P)) \tag{2}$$

$$S(P)=-\lg(Sim(f_s(P),f_f(P))) \tag{3}$$

其中,$n_t(P)$和 $n_f(P)$分别为一次运行中谓词 P 被判定为“真”和“假”的次数;$\Phi_s(P)$和 $\Phi_f(P)$分别为运行成功和失败时,程序在执行一个测试用例时谓词 P 被判定为“真”的概率;$f_s(P)$和 $f_f(P)$分别为运行全部的测试用例后,由 $\Phi_s(P)$和 $\Phi_f(P)$得到的统计模型,如求均值;$Sim(f_s(P),f_f(P))$代表任意一种相似度函数,如计算平方差。

经实验验证,CBI 和 Sober 算法都可以在各自适用条件下取得较好的定位结果。借鉴这两种基于谓词的缺陷定位算法,本文考虑在实时性缺陷定位中,将程序单元(模块或函数)的执行时间信息作为时间谱,通过比较程序单元的成功时间谱和失败时间谱的差异,得到程序单元包含实时性缺陷的可疑度。因此,本文提出了以下实时性缺陷定位方法。

2.2　模块级别的实时性缺陷定位方法

模块级别的实时性缺陷定位方法设计思想如下:将模块 m_i 的基准执行时间记为 $t_{m_0}[i]$,然后对一个含缺陷版本的测试程序加载一组测试用例。若根据运行结果判断出其中有 n 个失败用例,则对于第 j 个失败用例,计算模块 m_i 的平均执行时间,记为 $t_m[i][j]$,对这 n 个平均执行时间求均值,得到$\overline{t_m[i]}$:

$$\overline{t_m[i]}=\sum_{j=1}^{n}t_m[i][j]/n \tag{4}$$

最后将$\overline{t_m[i]}$与 $t_{m_0}[i]$进行比较,得到模块 m_i 的实时性可疑度值 $s_m[i]$:

$$s_m[i]=\begin{cases}1, \overline{t_m[i]}>2t_{m_0}[i]\\ \dfrac{\overline{t_m[i]}-t_{m0}[i]}{t_{m_0}[i]}, t_{m_0}[i]<\overline{t_m[i]}\leqslant 2t_{m_0}[i]\\ 0, \overline{t_m[i]}\leqslant t_{m_0}[i]\end{cases} \tag{5}$$

根据工程经验,当$\overline{t_m[i]}>2t_{m_0}[i]$时,即实际的平均执行时间远大于基准执行时间,表示模块包含实时性缺陷的可能性非常大,因此,可疑度值记为 1;当$\overline{t_m[i]}\leqslant t_{m_0}[i]$时,表示模块的实时性性能满足要求,模块包含实时性缺陷的可能性比较小,因此,可疑度值记为 0;当 $t_{m_0}[i]<\overline{t_m[i]}\leqslant 2t_{m_0}[i]$时,$\overline{t_m[i]}$对于 $t_{m_0}[i]$的增量越大,则可疑度越大。

上述方法的设计思想比较简单,可以给出每个模块的实时性失效程度,对于强实时性软件,即使可疑度值很小,也要对该模块进行修改和反复调试,直至其稳定达到实时性要求。

2.3 函数级别的实时性缺陷定位方法

将函数级别的实时性缺陷定位方法分为存在成功用例和不存在成功用例 2 种情况进行介绍。

2.3.1 存在成功用例的情况

对一个含缺陷版本的测试程序加载一组测试用例,若其中有 n_1 个失败用例、n_2 个成功用例,则对于第 j 个失败用例,计算函数 f_i 的平均执行时间 $t_{\mathrm{ff}}[i][j]$,然后对 n_1 个平均执行时间求均值,得到 $\overline{t_{\mathrm{ff}}[i]}$:

$$\overline{t_{\mathrm{ff}}[i]}=\sum_{j=1}^{n_1}t_{\mathrm{ff}}[i][j]/n_1 \tag{6}$$

同样,对于第 j 个成功用例,计算函数 f_i 的平均执行时间 $t_{\mathrm{fs}}[i][j]$,然后对这 n_2 个平均执行时间求均值,得到 $\overline{t_{\mathrm{fs}}[i]}$:

$$\overline{t_{\mathrm{fs}}[i]}=\sum_{j=1}^{n_2}t_{\mathrm{fs}}[i][j]/n_2 \tag{7}$$

最后将 $\overline{t_{\mathrm{fs}}[i]}$ 与 $\overline{t_{\mathrm{ff}}[i]}$ 进行比较,得到函数 f_i 的实时性可疑度值 $s_{\mathrm{f}}[i]$:

$$s_{\mathrm{f}}[i]=\begin{cases}\dfrac{\overline{t_{\mathrm{ff}}[i]}-\overline{t_{\mathrm{fs}}[i]}}{\overline{t_{\mathrm{ff}}[i]}+\overline{t_{\mathrm{fs}}[i]}}, & \overline{t_{\mathrm{ff}}[i]}>\overline{t_{\mathrm{fs}}[i]}\\ 0, & \overline{t_{\mathrm{ff}}[i]}\leqslant\overline{t_{\mathrm{fs}}[i]}\end{cases} \tag{8}$$

当 $\overline{t_{\mathrm{ff}}[i]}\leqslant\overline{t_{\mathrm{fs}}[i]}$ 时,即函数在失败用例中的平均执行时间比在成功用例中的平均执行时间还要小,若该函数存在实时性缺陷,则执行到该缺陷的用例的执行时间会比正常情况下长,从而成为失败用例,因此,该函数很可能不包含实时性缺陷,从而将可疑度值记为 0;当 $\overline{t_{\mathrm{ff}}[i]}>\overline{t_{\mathrm{fs}}[i]}$ 时,即函数在失败用例中的平均执行时间比在成功用例中的要长,因此,该函数中可能存在实时性缺陷,而失败用例则是那些执行到该缺陷的用例,相反,成功用例则是未执行到该缺陷的用例。本文用两者的差值与其和的比值来表示两者的差异大小,比值越大,则函数的可疑度越大。

2.3.2 不存在成功用例的情况

当不存在成功用例时,则不能通过比较成功用例与失败用例的差异来计算函数的可疑度,在这种情况下,本文将模块级别的定位结果作为已知条件来讨论如何定位可能包含缺陷的函数。

若测试用例中只有失败用例,失败用例的个数为 n,则首先利用 2.2 节中介绍的模块级别的缺陷定位方法计算出模块的可疑度。分析可知,模块的可疑度越大,其调用函数的可疑度也越大。若同时有 l 个模块失效,则同时被这 l 个模块调用的函数的可疑度会更大。若失效模块调用的某个函数,同时也被未超时模块调用,则其可疑度要小于只被失效模块调用的函数的可疑度,只被未失效模块调用的函数的可疑度则比较小。由此可利用式(9)来计算函数可疑度:

$$s_{\mathrm{f}}[i]=\frac{\sum_j s_{\mathrm{m}}[j][i]}{\sum_{k=0}^{6}s_{\mathrm{m}}[k]}\times\frac{1}{2^{n_{\mathrm{t}}-n_{\mathrm{tf}}}} \tag{9}$$

其中,$s_{\mathrm{m}}[j][i]$ 表示调用了函数 f_i 的模块的可疑度;$\sum_j s_{\mathrm{m}}[j][i]$ 是所有调用了函数 f_i 的模块可疑度的和;$\sum_{k=0}^{6}s_{\mathrm{m}}[k]$ 是所有模块可疑度的和;n_{t} 表示调用了函数 f_i 的模块个数;n_{tf} 表示调用了函数 f_i 且失效的模块个数;$2^{n_{\mathrm{t}}-n_{\mathrm{tf}}}$ 的作用是区分调用某函数的模块都失效与只有一部分模块失效的情况的函数可疑度的不同,显然前种情况下函数的可疑度要大。

上述方法涵盖了对各种情况函数可疑度的分析,虽然不能精确地体现出函数的失效程度,但根据可疑度值对函数的可疑度排序是值得参考的。

3 仿真实验与结果分析

3.1 实验内容

本文仿真实验选用飞机着陆自动控制软件的控制律程序作为被测对象。实时性缺陷定位的实验过程如图 2 所示。

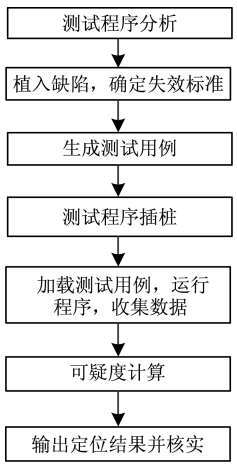

图 2 实时性缺陷定位实验过程

3.1.1 实验对象

对实验被测程序描述如下:

1)被测程序的组成

飞机着陆自动控制软件的控制律程序由 7 个模块组成,分别为盘旋模块、下滑模块、拉平模块、着陆模块、增稳模块、限幅模块和主函数本体模块,将其依次记为 $m_0\sim m_6$。控制律程序的 7 个模块共调用了 15 个函数(包括主函数本体),依次记为 $f_0\sim f_{14}$。模块与函数的调用关系如表 1 所示。

表 1　模块与函数的调用关系

模块	所调用的函数	模块	所调用的函数
m_0	f_1,f_2,f_3,f_6,f_{12}	m_4	f_9
m_1	f_1,f_2,f_5,f_8	m_5	f_{13}
m_2	f_0,f_1,f_2,f_4,f_{11}	m_6	f_{14}
m_3	f_7,f_{10}		

2)时间参数的设定

针对实验对象,根据大量测试用例的执行情况,设定控制律周期的最坏执行时间 T = 8 000 ns,即控制律执行一次的时间超过 8 000 ns,则该周期为失效周期。同样针对实验对象,设定 7 个模块的基准执行时间为:

$$t_{m_0} = \{3\ 500, 3\ 000, 2\ 000, 500, 60, 200, 1\ 300\}$$

实际上,代码的执行时间测量不能达到纳秒级的精度,由于实验研究中需要获取较短的代码执行时间,因此本文选用函数 *GetCpuCycle*( )作为时间戳。该函数中使用了可以获取当前 CPU 自上电以来的时间周期数的 RDTSC(Read Time Stamp Counter)指令。在一段代码的首尾分别调用该函数,得到的差值即为这段时间内 CPU 的周期数,其与 CPU 主频率的比值即为这段代码的执行时间。由于 CPU 的频率可以达到 GHz 级别,因此时间的精度可达到 ns,但缺点是其计算数值的浮动比较大。由于基准时间与实际时间都是利用该函数获得的,因此可以忽略浮动带来的影响。

下面分析插入时间戳对程序执行时间的影响。分别加载同一输入到各函数插入时间戳和未插入时间戳的控制律程序中,重复执行50 次得到平均执行时间,如表 2 所示。

表 2　时间戳对执行时间的影响

插桩情况	平均运行时间/ns	标准差
无时间戳	24 438	768.8
植入时间戳	24 558	918.5

由表 2 可以看出,时间戳使原程序的执行时间增加了 0.4% ,影响并不是很大,因此,该时间戳可以满足实验要求。

3)被测程序版本的缺陷设置

本文在原程序中通过人为设置延迟时间仿真实时性缺陷,使得程序出现实时性失效,以此作为包含缺陷的程序版本。延迟时间通过在某段代码内插入空语句的循环来产生,通过设置循环次数可以调整延迟时间的长短。

为充分地进行实时性缺陷定位方法的实验验证,根据实时性缺陷可能的分布情况,共设计 5 个实时性缺陷和 7 个程序版本。5 个缺陷所在的模块和函数如表 3 所示,7 个程序版本中所包含的缺陷及其所用级别如表 4 所示。

表 3　植入的实时性缺陷的分布情况

植入的缺陷	缺陷所在模块	缺陷所在函数
bug_1	m_0,m_1,m_2	f_2(分支内)
bug_2	m_1	f_8
bug_3	m_0,m_1,m_2	f_1
bug_4	m_5	f_{13}
bug_5	m_1	f_5

表 4　各程序版本介绍

版本	所包含缺陷	所用于的定位级别
v_1	bug_1	模块,函数
v_2	bug_2	模块
v_3	bug_3	模块,函数
v_4	bug_1, bug_2	模块,函数
v_5	bug_2, bug_4	模块
v_6	bug_4	函数
v_7	bug_5	函数

3.1.2　实验步骤

步骤 1　在被测程序的各版本中插入时间戳,即在模块和函数首尾分别插入函数 *GetCpuCycle*( ),用来获得软件模块和函数的执行时间。

步骤 2　设定失效标准,执行一个测试用例,需要 10 000 个 ~30 000 个执行周期。由于在实验过程中发现,每个测试用例的执行结果中总会有 1 个 ~5 个执行周期的执行时间远大于其他执行周期的执行时间,因此为排除该情况造成的延时,这里规定:失效周期数大于 10,则为失败用例;小于 10,则为成功用例,而理论上的失败/成功用例的定义仍如 1.2 节所述。

步骤 3　利用等价类划分和边界值分析等方法为被测程序设计 100 个测试用例,并使测试用例可以完全覆盖程序中的可执行代码和所有路径。

步骤 4　对于 7 个程序版本,依次加载这 100 个测试用例,得到运行结果,即每个周期的每个模块和函数的执行时间。

步骤 5　利用 2.2 节和 2.3 节提出的缺陷定位方法对运行结果进行处理,得到模块或函数的可疑度值及其排序。

步骤 6 对每个版本的缺陷定位结果分别进行分析,对提出的实时性缺陷定位方法进行评估。

3.2 实验结果和分析

根据 2.2 节和 2.3 节提出的 3 种不同情况下的缺陷定位方法,本文分别得到了仿真实验的缺陷定位结果。

3.2.1 模块级别的缺陷定位结果

根据实验得到的 v_1,v_2,v_3,v_4,v_5 这 5 个程序版本的模块的执行时间信息,利用式(4)和式(5)分别得到其模块可疑度值 s_m 和可疑度大小排序,如表 5 所示。表中加粗数值对应的函数为该版本程序中含有缺陷的函数。

表 5 模块可疑度

程序版本	比较内容	m_0	m_1	m_2	m_3	m_4	m_5	m_6
v_1	排序	1	2	3	3	3	3	3
	可疑度	**0.547**	**0.004**	**0.000**	0.000	0.000	0.000	0.000
v_2	排序	3	1	4	4	4	2	4
	可疑度	0.034	**1.000**	0.000	0.000	0.000	0.130	0.000
v_3	排序	1	1	1	2	2	2	2
	可疑度	**1.000**	**1.000**	**1.000**	0.000	0.000	0.000	0.000
v_4	排序	2	1	3	3	3	3	3
	可疑度	0.157	**1.000**	0.000	0.000	0.000	0.000	0.000
v_5	排序	2	1	2	2	2	1	2
	可疑度	0.000	**1.000**	0.000	0.000	0.000	**1.000**	0.000

由表 5 可知,v_1 版本中 m_0 的可疑度最大,为 0.547,m_1 的可疑度为 0.004,其余模块的可疑度皆为 0.000,已知 v_1 版本所包含的缺陷 bug_1 位于 m_0,m_1 和 m_2 中,根据结果,可以推测缺陷很可能位于 m_0 中,是符合实际的;v_2 版本中 m_1 的可疑度最大,为 1.000,而 v_2 版本所包含的缺陷 bug_2 恰好位于 m_1 中;v_3 版本中 m_0,m_1,m_2 的可疑度均为 1.000,其他模块可疑度均为 0.000,其所包含的缺陷 bug_3 恰好位于 m_0,m_1,m_2 中,因此定位效果很精确;v_4 版本中 m_1 的可疑度最大,为 1.000,而其所包含的缺陷 bug_1 和 bug_2 恰好共同位于 m_1 中;v_5 版本中 m_1 和 m_5 的可疑度均为 1.000,其他模块可疑度均为 0.000,而其所包含的缺陷 bug_4 和 bug_2 恰好分别位于 m_5 和 m_1 中,因此定位效果也很精确。

如果测试人员在检查模块时只需一次即可正确检查出是否包含缺陷,则根据模块可疑度列表,找到程序中的缺陷所需检查的模块的个数不超过 2 个。

综合以上分析,可疑度很大的模块均包含缺陷,定位精度较高,因此,模块级别的缺陷定位方法具有可行性。

3.2.2 存在成功用例的函数级别缺陷定位结果

本节分析存在成功用例情况下的函数级别的实时性缺陷定位结果,根据实验得到的 v_1 程序版本的函数的执行时间信息,利用式(6)~式(8)得到其函数可疑度值 s_f 和可疑度大小排序,如表 6 所示。

表 6 v_1 的函数可疑度

排序	函数	$\overline{t_{ff}}$	$\overline{t_{fs}}$	可疑度
1	f_2	2 161	918	**0.404**
2	f_1	3 376	2 649	0.121
3	f_{14}	1 802	1 659	0.041
4	f_7	174	167	0.021
5	f_6	188	186	0.005
6	f_0	125	127	0.000
6	f_3	1 168	1175	0.000
6	f_4	355	363	0.000
6	f_5	1 075	1129	0.000
6	f_8	591	617	0.000
6	f_9	55	56	0.000
6	f_{10}	723	1961	0.000
6	f_{11}	539	563	0.000
6	f_{12}	848	864	0.000
6	f_{13}	189	212	0.000

由表 6 可知,函数 f_2 的可疑度值最大,为 0.404,而 v_1 版本包含缺陷 bug_2 恰好位于 f_2 中。

根据 v_1 函数可疑度列表可知,找到缺陷所需检查的函数不超过 1 个。因此,存在成功用例情况下的缺陷定位方法定位效果比较理想。

3.2.3 不存在成功用例的函数级别缺陷定位结果

本节分析不存在成功用例情况下函数级别的实

时性缺陷定位结果,根据实验得到的 v_3,v_4,v_6,v_7 程序版本的模块的执行时间信息,结合模块与函数的调用关系,利用式(9)分别得到其函数可疑度值 s_f 和可疑度大小排序,如表 7 所示。

表 7　v_3,v_4,v_6 和 v_7 的函数可疑度

函数	v_3		v_4		v_6		v_7	
	排序	可疑度	排序	可疑度	排序	可疑度	排序	可疑度
f_0	2	0.333	4	0.000	3	0.000	3	0.000
f_1	1	**1.000**	2	0.579	3	0.000	2	0.250
f_2	1	1.000	2	**0.579**	3	0.000	2	0.250
f_3	2	0.333	3	0.136	3	0.000	3	0.000
f_4	2	0.333	4	0.000	3	0.000	3	0.000
f_5	2	0.333	1	0.864	3	0.000	1	1.000
f_6	2	0.333	3	0.136	3	0.000	3	0.000
f_7	3	0.000	4	0.000	2	0.033	3	0.000
f_8	2	0.333	1	**0.864**	3	0.000	1	**1.000**
f_9	3	0.000	4	0.000	3	0.000	3	0.000
f_{10}	3	0.000	4	0.000	2	0.033	3	0.000
f_{11}	2	0.333	4	0.000	3	0.000	3	0.000
f_{12}	2	0.333	3	0.136	3	0.000	3	0.000
f_{13}	3	0.000	4	0.000	1	**0.967**	3	0.000
f_{14}	3	0.000	4	0.000	3	0.000	3	0.000

由表 7 可知,v_3 版本中 f_1 和 f_2 的可疑度最大,为 1.000,其所包含的缺陷 bug_3 恰好位于 f_1 中;v_4 版本中 f_5 和 f_8 的可疑度最大,为 0.864,其所包含的缺陷 bug_1 和 bug_2 恰好分别位于 f_2 和 f_8 中;v_6 版本中 f_{13}的可疑度最大,为 0.967,其他模块可疑度均为 0.000,其所包含的缺陷 bug_4 恰好位于 f_{13}中;v_7 版本中 f_5 和 f_8 的可疑度最大,为 1.000,而其所包含的缺陷 bug_5 恰好位于 f_5 中。由此可知,找到缺陷所需要检查的函数不超过 2 个,因此定位精度较高。

综合上述分析,包含缺陷的函数的可疑度最大,因此定位效果较理想。该方法与存在成功用例情况下函数级别缺陷定位方法的区别是把函数划分为 3 个或 4 个可疑级别,共同被相同模块调用的函数可疑度往往相同,但包含缺陷的函数,其可疑度均比较大,在一定程度上,该方法可以帮助测试人员提高寻找缺陷的效率。

4　可视化工具开发

对于可视化工具的开发,本文首先进行着色技术研究,提出满足要求的着色函数,然后对可视化工具界面进行分析和设计,并运用 WPF(Windows Presentation Foundation)平台对其进行开发。

4.1　着色技术研究

文献[15]对测试信息的可视化进行了实验研究,用于辅助缺陷定位,其中使用了代码着色来显示可疑度。因此,本文使用的着色技术是指根据软件单元的可疑度值,对程序中可疑的代码、模块或函数进行着色。着色函数是指所着颜色与可疑度值之间的对应关系,即表示颜色的 *RGB* 值与可疑度值的对应关系。着色函数的选择决定了着色效果,直观地显示出代码的可疑度程度,起到指导作用越好。

在可视化工具所运行的 WPF 平台上,颜色的设置是调用 *Color. FromArgb*(*alpha*,*r*,*g*,*b*)方法,该方法是通过设定 *alpha*,*r*,*g*,*b* 这 4 个参数来设置颜色,其中,*alpha* 是颜色的透明度;*r* 是颜色的红色成分;*g* 是颜色的绿色成分;*b* 是颜色的蓝色成分,4 个参数均取值 0 ~ 255 之间的整数。本文选取红色(255,0,0)到绿色(0,255,0)之间的光谱颜色来对代码着色,越接近红色,可疑度越大;越接近绿色,可疑度则越小。得到的着色函数为:

$$\begin{cases} r=0,g=255,s<0 \\ r=(\text{byte})(510\times s+0.5),g=255,0\leqslant s\leqslant 0.5 \\ r=255,g=(\text{byte})(510\times(1-s)+0.5),0.5\leqslant s<1 \\ r=255,g=0,s\geqslant 1 \end{cases} \tag{10}$$

其中,添加的 0.5 是保证在进行强制转换成整数时,可以起到四舍五入的效果。此外,本文取 *alpha* = 180,*b* = 0。着色显示效果如图 3 所示。

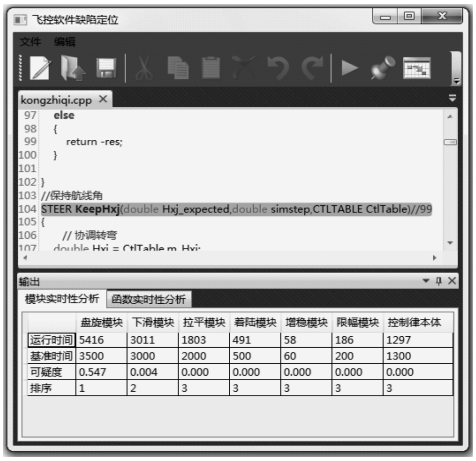

图 3 着色显示效果

图 3 显示了均匀分布在 1 ~0 之间的 14 个可疑度值所对应的颜色,为了便于比较,对相邻代码行依次进行着色,图 3 中显示的被着色代码从上到下,对应的可疑度值从 1 至 0 变化,即第 1 行代码的可疑度值为 1,最后一行的可疑度值为 0。

4.2 定位结果显示界面

缺陷定位的显示界面由被测程序的代码着色显示和可疑度结果的显示两部分组成,其中可疑度结果的输出窗口包括模块实时性分析和函数实时性分析两部分。

模块级别实时性分析的内容包括各模块运行平均时间、基准时间,可疑度以及可疑度排序。v_1 版本模块级别的实时性缺陷定位结果显示窗口如图 4 所示。

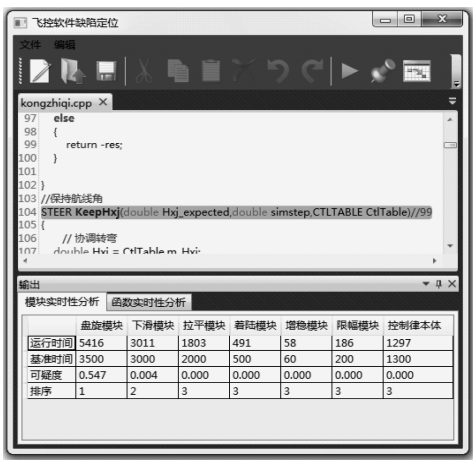

图 4 模块实时性分析显示界面

函数级别实时性分析的内容包括各函数可疑度、可疑度排序以及函数名所在的行数。函数实时性分析界面还实现了可疑度结果与程序代码之间的链接,即双击界面某函数所在的列,鼠标就可以调转到相应行数处的代码,即该函数的函数名在程序中的位置,为测试人员查找函数节约了时间。v_1 版本的函数级别的实时性缺陷定位结果显示窗口如图 5 所示,其中被着色代码即为被双击的可疑度最大的函数 f_2 的函数名。

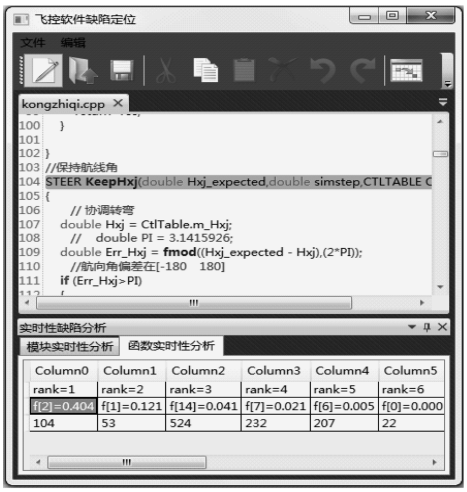

图 5 函数实时性分析显示界面

通过可视化工具可疑度结果的显示界面,测试人员可以清楚地看到执行时间、可疑度值及排序等信息。利用着色代码的显示界面以及可疑度结果与着色代码的链接,测试人员可以快速地找到可疑函数的位置,同时直观地看到函数的可疑程度。因此,可视化工具在一定程度上提高了软件缺陷定位的自动化程度。

5 结束语

本文首先根据飞行控制软件的运行特点,建立了实时性缺陷定位模型。然后基于成功用例与失败用例的执行时间差异越大、可疑度越大的思想,针对模块和函数 2 个级别的 3 种不同情况,分别提出了相应的实时性缺陷定位方法。实验结果表明,对于包含单个或多个缺陷的程序版本,3 种缺陷定位方法均可赋予包含实时性缺陷的模块或函数较大的可疑度值,帮助测试人员尽快检查到包含缺陷的软件单元,具有一定的可行性。最后开发了可视化工具,实现了代码着色和定位结果的显示功能,提高了缺陷定位方法的自动化程度。

本文提出的 3 种实时性缺陷定位方法有待于改善,可以利用数据挖掘、机器学习等更先进的方法,如对执行信息进行分析,得到更精确的定位结果。此外,在仿真实验中,程序中插入的时间戳会影响程序本身执行时间的测量精度。因此,可以进一步优化插桩策略,如利用动态插桩技术、基于动态链接库技术的代码插桩方法等,使其造成的影响尽可能小,从而减小实验误差。

参考文献

[1] 王　泉,张学宏,周敏刚,等. 无人机飞控软件测试方法研究[J]. 航空计算技术,2008,38(2):78-81.

[2] Korel B, Laski J. Dynamic Program Slicing[J]. Information Processing Letters, 1988, 29(3): 155-163.

[3] Wong E, Qi Yu. Effective Program Debugging Based on Execution Slices[J]. Journal of Systems and Software, 2006, 79(7): 891-903.

[4] Wong E, Debroy V, Choi B. A Family of Code Coverage-based Heuristics for Effective Fault Localization[J]. Journal of Systems and Software, 2010, 83(2): 188-208.

[5] Naish L, Lee H J, Ramamohanarao K. A Model for Spectra-based Software Diagnosis[J]. ACM Transactions on Software Engineering and Methodology, 2011, 20(3): 563-574.

[6] Jeffrey D, Gupta N, Gupta R. Fault Localization Using Value Replacement[C]//Proceedings of 2008 International Symposium on Software Testing and Analysis. New York, USA: ACM Press, 2008: 167-178.

[7] Zeller A. Isolating Cause-effect Chains from Computer Programs[J]. ACM SIGSOFT Software Engineering Notes, 2002, 27(6): 1-10.

[8] Wong E, Debroy V, Golden R, et al. Effective Software Fault Localization Using an RBF Neural Network[J]. IEEE Transactions on Reliability, 2012, 61(1): 149-169.

[9] Brun Y, Ernst M D. Finding Latent Code Errors via Machine Learning over Program Executions[C]//Proceedings of the 26th International Conference on Software Engineering. Washington D. C., USA: IEEE Computer Society, 2004: 480-490.

[10] Nessa S, Abedin M, Wong E, et al. Software Fault Localization Using N-gram Analysis[M]//Li Yingshu, Huynh D T, Das S K, et al. Wireless Algorithms, Systems, and Applications. Berlin, Germany: Springer, 2008: 548-559.

[11] Jones J A, Harrold M J. Empirical Evaluation of the Tarantula Automatic Fault-localization Technique[C]//Proceedings of the 20th IEEE/ACM International Conference on Automated Software Engineering. New York, USA: ACM Press, 2005: 273-282.

[12] Liu Chao, Fei Long, Yan Xifeng, et al. Statistical Debugging: A Hypothesis Testing-based Approach[J]. IEEE Transactions on Software Engineering, 2006, 32(10): 831-848.

[13] 周保宇,田　力. 高实时性的无人机飞控软件测试系统设计[J]. 计算机测量与控制,2012,20(9):2384-2385.

[14] Abreu R, Zoeteweij P, Golsteijn R, et al. A Practical Evaluation of Spectrum-based Fault Localization[J]. Journal of Systems and Software, 2009, 82(11): 1780-1792.

[15] Jones J A, Harrold M J, Stasko J. Visualization of Test Information to Assist Fault Localization[C]//Proceedings of the 24th International Conference on Software Engineering. New York, USA: ACM Press, 2002: 467-477.

编辑　金胡考

(上接第 110 页)

[8] Kogge P M. Final Report: Processor-in-Memory (PIM) Based Architectures for PetaFlops Potential Massively Parallel Processing[EB/OL]. (1996-07-15). http://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/19970001424.pdf.

[9] Elliott D G, Stumm M, Snelgrove W M, et al. Computational RAM: Implementing Processors in Memory[J]. IEEE Design & Test of Computers, 1999, 16(1): 32-41.

[10] Wang Yuhao, Zhang Chun, Yu Hao, et al. Design of Low Power 3D Hybrid Memory by Non-volatile CBRAM-crossbar with Block-level Data-retention[C]//Proceedings of ACM/IEEE International Symposium on Low Power Electronics and Dsign. New York, USA: ACM Press, 2012: 197-202.

[11] Wang Liyun, Zhang Chun, Chen Liguang, et al. A Novel Memristor-based rSRAM Structure for Multiple-bit Upsets Immunity[J]. IEICE Electronics Express, 2012, 9(9): 861-867.

[12] Morad A, Yavits L, Ginosar R. GP-SIMD Processing-in-Memory[J]. ACM Transactions on Architecture and Code Optimization, 2015, 11(4): 1-26.

[13] Paul S, Krishna A, Qian Wenchao, et al. MAHA: An Energy-efficient Malleable Hardware Accelerator for Data-intensive Applications[J]. IEEE Transactions on Very Large Scale Integration Systems, 2015, 23(6): 1005-1016.

[14] Lin Jie, Zhu Shikai, Yu Zhiyi, et al. A Scalable and Reconfigurable 2.5D Integrated Multicore Processor on Silicon Interposer[C]//Proceedings of 2015 IEEE Custom Integrated Circuits Conference. Washington D. C, USA: IEEE Press, 2015: 1-4.

[15] Ou Peng, Zhang Jiajie, Quan Heng, et al. A 65 nm 39GOPS/W 24-core Processor with 11Tb/s/W Packet Controlled Circuit-switched Double-layer Network-on-Chip and Heterogeneous Execution Array[C]//Proceedings of 2013 IEEE International Solid-State Circuits Conference Digest of Technical Papers. Washington D. C., USA: IEEE Press, 2013: 56-57.

[16] 林　杰. 2.5D 多核处理器关键技术研究——存储体系、片间接口及芯片实现[D]. 上海:复旦大学,2015.

[17] 毕厚杰,王　建. 新一代视频压缩编码标准——H.264/AVC[M]. 2 版. 北京:人民邮电出版社,2009.

[18] 欧　鹏. 面向通信和多媒体的多核处理器关键技术研究[D]. 上海:复旦大学,2013.

编辑　金胡考