文章编号:1671-4598(2022)03-0037-06 DOI:10.16526/j.cnki.11-4762/tp.2022.03.007 中图分类号:TP311 文献标识码:A

基于软件运行特征的故障检测方法研究

王正阳[1], 王雅文[1,2]
(1. 北京邮电大学 网络与交换技术国家重点实验室, 北京 100876;
2. 桂林电子科技大学 广西密码学与信息安全重点实验室, 广西 桂林 541004)

摘要: 针对软件源代码静态检测时故障报告中误报较多问题, 提出一种基于软件运行特征的故障检测方法, 通过引入动态分析的方式进行故障检测; 首先扩展了动态测试插装库, 设计了八种常见故障模式对应的探针函数, 然后在程序中搜索故障监控位置并进行故障监控探针的插装, 最后在软件执行过程中分析插装消息中的运行特征从而识别故障; 实验结果表明该方法能够有效检测程序故障且检测出的故障均为真实存在, 弥补了静态分析误报率高的问题。

关键词: 软件测试; 静态分析; 动态测试; 插装; 故障模式; 运行监控

Research on Fault Detection Method Based on Software Operational Characteristics

WANG Zhengyang[1], WANG Yawen[1,2]
(1. State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, Beijing 100876, China; 2. Guangxi Key Laboratory of Cryptography and Information Security, Guilin University of Electronic Technology, Guilin 541004, China)

Abstract: Aiming at the problem of many false positives in fault report during static detection of software source code, a method of fault detection based on software operation characteristics is proposed. Firstly, the dynamic test instrumentation library is extended from designing the probe functions corresponding to eight common fault modes. Then, the fault monitoring position is searched in the program and the fault monitoring probe is inserted. Finally, in the process of software execution, the operation characteristics in the instrumentation message are analyzed to identify the fault. The experimental results show that, the method can effectively detect the program faults and all detected faults are real existence. This method makes up for the problem of the high false positive rate in static analysis.

Keywords: software testing; static analysis; dynamic test; instrumentation; fault mode; operation monitoring

0 引言

软件测试[1]是发现软件故障、提高软件可靠性的重要手段, 源代码分析可以根据是否执行被测程序分为静态分析[2-4]和动态分析[5-7]两种方式。静态方法不需要执行被测程序, 而是扫描被分析程序的源代码, 通过使用符号执行技术进行静态路径分析、区间运算并根据设计的缺陷模式状态机检测代码中可能存在的非功能性故障。以静态分析工具 HPFortify[8]为例, 通过静态扫描得到的缺陷数量庞大, 如表1所示, 其中不可避免地包含了分析误报, 程序通过静态分析出现的误报需要专业的测试人员进行人工缺陷确认, 对于工程级别的代码, 所需要人工确认的缺陷数量可能是十分庞大的, 反而降低了静态分析的检测效率。

动态分析技术主要通过动态测试工具执行被测程序, 收集分析程序运行时信息进而发现或验证程序性质。一般是通过插装[10-11]的方式根据监测需求在程序中插入一些探针函数, 在不改变程序原有逻辑正确性以及完整性的基础上, 收集并分析程序运行时信息如变量值、程序运行路径以及程序运行时间等进而检验程序质量, 然而传统动态测试工具的故障检测能力十分有限。

表1 Fortify 扫描开源程序的结果[9]

程序	代码规模	总缺陷报告数量
Gzip-1.3.9	7 622	52
Libcgroup-0.37.1	16 104	351
cgminer-4.3.4	37 859	1 005
Sendmail-8.12.4	99 196	5 246
Git-1.7.2	117 607	4 621

针对上述问题, 本文提出一种基于软件运行特征的故

收稿日期:2021-12-22; **修回日期**:2022-02-08。
基金项目:广西密码学与信息安全重点实验室开放课题(GCIS202103)。
作者简介:王正阳(1997-),男,河南郑州人,硕士研究生,主要从事软件测试以及软件可靠性方向的研究。
王雅文(1983-),女,陕西宝鸡人,博士生导师,副教授,主要从事软件测试和程序分析方向的研究。
引用格式:王正阳,王雅文.基于软件运行特征的故障检测方法研究[J].计算机测量与控制,2022,30(3):37-42.

障检测方法。核心思想是以动态测试的方式，通过在软件的执行过程中监控的程序状态、系统状态及执行时间等软件运行信息对相应故障模式进行识别。首先通过对故障模式的分析构建故障监控探针插装库。再通过静态分析识别故障监控点，结合相应的故障条件确定监控内容，在不改变原有代码逻辑的情况下，生成监控探针插入到源程序中。以动态测试的方式在程序运行过程中自动识别故障并在源程序中定位。该方法适用于逻辑复杂且可靠性要求高的软件，是静态方法的重要补充。

1 故障检测的现状

由于静态分析中对于复杂的程序语句以及对于操作系统和库函数调用的符号推理不够精确导致静态分析无法同时满足可靠性和完备性，业界通常在静态分析工具的效率与精度之间做出取舍，因此出现漏报的同时会出现大量的误报。目前现有的静态分析工具漏报率为 9%～32% ，误报率为 35%～91%[12]。

对于静态分析出现的大量误报问题，有很多针对误报消除[13]技术的研究，文献［14］提出了一种基于缺陷检测系统的警报自动聚类方法；文献［15］在区间抽象域基础上引入符号化三值逻辑抽象域，以支持表达变量间的逻辑关联；文献［16］提出了一种程序语义缺陷警报关联的方法，通过挖掘警报间的深层次关联信息建立警报关联。这些技术有助于提升人工确认的效率，但是仍然无法避免需要人工方式进行确认，因此开发人员开发软件时使用静态分析工具的意愿并不高。

动态测试在运行时所发现的代码漏洞一定是真实存在的，不存在误报情况[17]，传统的动态测试工具主要用于进行覆盖分析[18]，通过达到较高的测试覆盖率保证软件可靠性，然而对程序出现的故障没有进行分析确认。

对于故障检测技术的研究，无论是静态方法还是动态方法都已经相对成熟，单纯对静态检测方法或动态检测方法进行改进，没有太大的提升空间。但是，动静结合的漏洞分析技术可以弥补互相的优缺点，已经成为目前研究的热点。文献［9］针对静态分析报告的目标程序中内存泄漏的静态警报，基于警报制导信息对目标程序进行混合执行测试，结合动态执行的方式进行故障确认，然而该方法仍是通过静态分析方式识别故障，对于逻辑复杂的软件仍然难以避免误报率较高的问题，因此监测软件运行时故障特征从而在动态执行时识别故障是一种新的思路，能够有效弥补静态分析的误报问题。

2 检测软件故障的插装库设计

2.1 故障模式

程序中出现的故障可能会导致程序崩溃或出现不可预料的错误结果，大大降低软件可靠性甚至会产生严重后果。故障模式通常是由专业程序设计者以及专业测试人员总结出来经常出现且具有一定模式的故障[19]，可以成为出现故障时故障原因的分析依据，也可用于软件开发时可靠性设计的依据。本文所研究的故障模式是在代码上所体现出的一种错误形式，当具有某些特征的代码被执行时，会出现运行时故障。因此故障模式可从出错的状态上划分为 3 类 8 种，如下。

1）导致错误的程序状态：在程序执行过程中，表达式出现了非法的取值：

①空指针引用模式——被解引用的指针表达式的取值等于 NULL；

②数组越界模式——数组下标表达式的取值超出上下界；

③非法计算模式——某些操作数或库函数参数出现非法取值；

④缓冲区溢出模式——对目标缓冲区的操作长度超出其界限等。

2）导致错误的系统状态：在程序执行过程中，分配到系统堆栈上的内存出现了不正常的状态：

⑤内存泄漏[20]模式——被分配到堆栈上的内存没有及时释放；

⑥资源泄漏模式——被分配到堆栈上的资源（文件、网络、数据库）没有及时释放。

3）执行时间异常：在程序执行过程中，某些比较复杂的循环或函数的执行时间超出预期：

⑦低效循环模式——循环结构的执行时间超出了预定的阈值；

⑧低效函数模式——函数调用的执行时间超出了预定的阈值。

能够体现故障模式出错时运行环境状态的代码特征信息称为故障特征，包括故障指令、故障对象和故障条件三部分。

1）故障指令：能够触发故障的程序指令，包括地址解引用操作、下标操作、部分算术运算操作符、以及部分内存或算术运算操作库函数。

2）故障对象：程序中直接与故障指令相关的对象，如一个变量或表达式。

3）故障条件：对故障对象的约束，如表达式的取值范围、内存地址上的成对操作、函数或循环的执行时间阈值。

2.2 扩展的插装库

程序插装技术最早是由文献［10］于 1979 年提出的，程序插装指的是在被测程序在保持原有代码逻辑不发生改变的情况下，在程序中插入一些探针函数，通过分析探针函数在程序运行时捕获的运行特征来获得程序的控制流以及数据流信息。程序插装的方式通常根据插装对象的不同分为源代码插装和目标代码插装。

源代码插装是最自然的插装方式，通过对程序源文件进行词法分析、语法分析后，根据探针函数的功能确定在源代码中的插装位置，目标代码插装技术不依赖程序源代

码和编程语言，通常是对目标代码进行必要的分析以确定插装位置，由于目标代码一般是可执行的二进制程序，因此人工插入代码是很难实现的，一般通过调用对应插装工具提供的 API 实现对目标代码进行插装。

源代码插装的缺点主要是依赖源程序的编译语言，且对程序源代码分析工作量较大。但是由于对源文件进行了较为完整的静态分析，能够获得更多的语义信息，插装点较为准确，插入的探针在编译时也会被当作源代码进行编译优化，因此本文采用源代码插装。

传统的动态测试工具大多只是简单追求高测试覆盖率，其插装过程一般是按照所选取的覆盖准则根据获得的控制流图在程序中插入覆盖监控探针，根据输入的测试用例进行动态执行。在程序动态运行时，通过覆盖监控探针返回的信息统计测试覆盖率或是进行代码执行频度分析。本文的基于软件运行特征的故障检测方法除了支持传统的覆盖分析，为支持故障的动态检测，研究设计了能够检测故障的探针函数。

故障触发的条件是程序执行到故障点时，故障对象的取值满足故障条件的约束，因此检测故障的探针首先需要插入到程序中可能触发故障的位置，通过动态测试执行到插装点时收集程序执行到故障点时的相关运行特征，判断收集到的运行特征是否满足对应的故障约束条件从而达到识别故障并定位的功能。软件运行特征对于描述软件行为具有重要作用[21]。本文所涉及的软件运行特征可根据研究的故障模式大致分为程序状态、系统状态以及执行时间三类。程序状态包括程序中可见变量的类型和取值情况、函数调用情况等，系统状态主要为程序执行过程中的系统资源使用情况，执行时间包括函数执行时间、循环执行时间以及语句序列执行时间等。

基于前文故障模式与软件运行特征的研究具体设计如下八种相应的故障监控探针：

1）空指针引用故障监控探针：探针收集指针表达式和位置信息，约束指针表达式取值为非 NULL，若该指针表达式出现等于 NULL 的状态时识别故障；

2）非法计算故障监控探针：探针收集敏感操作符或库函数名称、算术表达式和位置信息，针对传入敏感操作符或库函数名称，约束算术表达式的取值，若该算术表达式的取值与约束相悖则识别故障；

3）数组越界故障监控探针：探针收集被监控数组的数组下界、数组上界、数组下标的算术表达式和位置信息，约束数组下标算术表达式处于数组上下界之间，若数组下标算术表达式取值超出该范围则识别故障；

4）缓冲区溢出故障监控探针：探针收集目标缓冲区长度、源缓冲区地址和位置信息，约束源缓冲区地址长度不大于目标缓冲区长度，若超出该范围则识别故障；

5）内存泄漏故障监控探针：探针收集分配或释放操作标志、操作的内存地址和位置信息，在程序退出后，内存操作文件中对相同地址的操作应该配对，否则识别故障；

6）资源泄漏故障监控探针：探针收集分配或释放操作标志、操作的资源地址和位置信息，在程序退出后，内存操作文件中对相同地址的操作应该配对，否则识别故障；

7）低效循环故障监控探针：探针收集进入或退出循环标志、时间戳、限定时间和位置信息，在退出循环后，约束退出和进入循环的时间差应不大于限定时间，否则识别故障；

8）低效函数故障监控探针：探针收集进入或退出函数标志、时间戳、限定时间和位置信息，在退出函数后，约束退出和进入函数的时间差应不大于限定时间，否则识别故障。

3　基于故障特征的程序插装

故障监控探针需要通过程序插装技术插入在源程序中的相应监控位置，故障监控点的位置判定则首先需要通过对故障模式的分析构建故障模型库，包含相关故障指令、故障对象及故障条件等内容。其中，故障指令和故障对象可以用于定位监控点。之后对于识别到相应故障指令的监控点，需要结合对应故障对象以及故障条件，以插装库中的对应故障监控探针格式插入到源程序中。最后，故障监控探针在程序的执行过程中搜集程序状态、系统状态和执行时间等信息来识别故障，并定位于源程序中。

3.1　基于故障特征的程序插装设计

基于故障特征的程序插装主要分为如图 1 所示 3 个阶段：首先源程序进行静态分析，通过在静态分析过程中构建的如抽象语法树（Abstract Syntax Tree，AST）、控制流图以及符号表等程序抽象模型[22]中搜索故障触发指令，定位程序中所有与故障模式相关的位置。之后进行故障关联信息提取，从定位处提取故障模式相应关联对象。最后基于相应故障模式的触发条件，面向关联对象生成监控探针插入到源程序中。

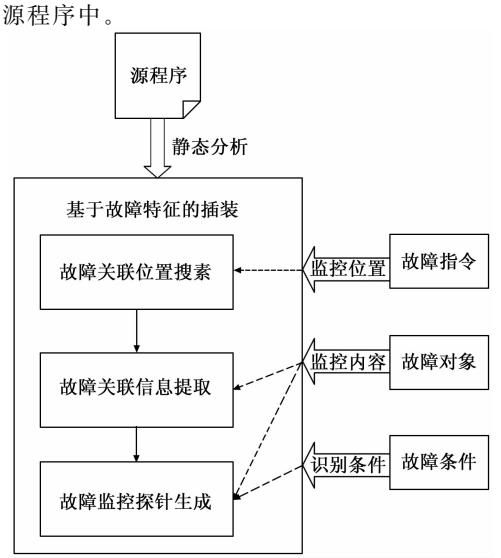

图 1　基于故障特征的程序插装流程

不同故障对应的故障监控探针插装设计如下：

1）空指针引用故障监控探针插装点位于空指针故障模式的触发指令之前，故障触发操作符包括对成员引用－＞、内容引用 * 、数组引用 []，故障触发库函数包括 fclose、fopen、strcat、strcpy、strcmp 等约束空参数的函数引用；

2）非法计算故障监控探针插装点位于非法计算故障模式的触发指令之前，故障触发操作符包括%、/、%＝、/＝等对右操作数有限制要求的，故障触发库函数包括 asin、acos、atan、div、fmod、ldiv、log、log10、sqrt 等对参数有限制要求的算术运算函数；

3）数组越界故障监控探针插装点位于数组成员引用操作之前，从检索到的故障触发指令对应的语句中提取故障关联对象，即数组下标表达式，同时从符号表中提取相关数组的上下界作为约束条件；

4）缓冲区溢出故障监控探针插装点位于 strcat、strcpy、strncpy、strcmp、strncmp 等对缓冲区进行操作的库函数之前，从检索到的故障触发指令对应的语句中提取故障关联对象，即源缓冲区地址，同时计算目标缓冲区的可用长度作为约束条件；

5）内存泄漏故障监控探针插装点位于 malloc、calloc、realloc 等具有内存分配特性的库函数之前，从检索到的故障触发指令对应的语句中提取故障关联对象，即被分配内存的地址标识符；其次，检索 free、delete 等具有内存释放特性的库函数并在这些函数调用前插入探针；

6）资源泄漏故障监控探针插装点位于 fopen、socket、accept 等资源分配函数之前，从检索到的故障触发指令对应的语句中提取故障关联对象，即被分配资源的地址标识符；其次，检索与分配资源对应的资源关闭函数并在这些函数调用前插入探针；

7）低效循环故障监控探针插装点位于 while、for、do 等循环结构开始之前以及循环结构结束位置之后；

8）低效函数故障监控探针插装点位于每个函数的入口处第一条语句之前和函数返回语句之前。

3.2 故障监控探针插装算法

针对本文研究的故障模式，根据对应故障触发指令定位其在程序中的插装点，主要通过遍历源程序静态分析生成的抽象语法树识别程序中的故障指令搜索故障监控探针插装点，并根据图 1 的插装流程提取需要监控的故障对象在源程序中生成探针函数得到插装后的程序，在程序执行过程中收集探针返回信息检测故障。

算法 1：描述了故障监控探针插装点定位算法，该算法将源程序通过静态分析生成的抽象语法树的根节点 root 以及初始化为空的插装点集合 instru _ set 作为输入，输出为确认后的故障监控探针插装点集合 instru _ set。

故障监控探针插装点定位算法中所用到的定义如下：

ASTNode：抽象语法树节点父类。

function_definition：函数体定义节点。

return_statement：return 语句节点。

iteration_statement：循环体节点。

simple_statement：简单语句节点。

expression：表达式节点。

getType(node)：获得节点 node 的节点类型。

getChildrenNodes(node)：获得 node 节点的所有子节点。

addLEFIn(instru_set，node)：将函数入口处的低效函数故障监控探针插装点加入插装点集合 instru_set 中。

addLEFOut(instru_set，node)：将返回语句前的低效函数故障监控探针插装点加入插装点集合 instru_set 中。

addLEC(instru_set，node)：将循环体之前以及之后的低效循环故障监控探针插装点加入插装点集合 instru_set 中。

check(node)：检查简单语句节点 node 是否包含 4 种程序状态出错故障触发指令、内存分配或释放指令以及资源分配释或放指令。

checkExpression(node)：检查表达式节点 node 是否包含 4 种程序状态出错故障触发指令、内存分配或释放指令以及资源分配释或放指令。

addFault(instru_set，node)：根据节点 node 中包含的故障指令，确定故障监控探针类型并将对应故障监控探针插装点加入插装点集合 instru_set 中。

算法 1：故障监控探针插装点定位算法

```
输入：root          //源程序的抽象语法树根节点
输出：instru_set    //故障监控探针插装点集合
1.    instru (ASTNode node, List instru_set) {
2.      if getType (node) == function_definition then
3.        addLEFIn (instru_set, node);
4.      if getType (node) == return_statement then
5.        addLEFOut (instru_set, node);
6.      if getType (node) == iteration_statement then
7.        addLEC (instru_set, node);
8.      if getType (node) == simple_statement then
9.        if (check (node)) then
10.           addFault (instru_set, node);
11.     if getType (node) == expression then
12.       if (checkExpression (node)) then
13.         addFault (instru_set, node);
14.     for each n getChildrenNodes (node)
15.       instru (n, instru_set);
16. }
```

算法主要思想是通过深度优先遍历源程序抽象语法树，识别对应的故障指令确认故障插装位置。算法 1～7 行通过判断语法树节点类型识别函数体和循环体，定位低效函数和低效循环故障监控探针插装点；8～13 行判断简单语句或者表达式中是否包含空指针引用、非法计算、数组越界、缓冲区溢出以及内存泄漏和资源泄漏的故障触发指令从而生成对应故障监控探针插装点。对于表达式节点需要拆分逻辑表达式，检查所有子表达式是否包含故障指令。14～15 行遍历当前节点的子节点，对子节点进行故障监控探针插装点定位。

4　实验结果与分析

前文已经介绍了基于故障特征的程序插装技术，为验证本文提出的基于软件运行特征的故障检测方法故障检测能力，将该技术应用于代码测试系统[23]（Code Testing System，CTS）中进行实验验证，实验环境为 Ubuntu 12.04 操作系统，配置 AMD Ryzen 9 5900HX（3.30 GHz）处理器，8 GB 内存以及 512 GB SSD 硬盘。

实验采用了 CTS 自带的 Norton 命令的 Unix 环境复制版开源项目 deco，项目包含的 10 个 C 文件如表 2 所示。为验证本文提出的方法，对被测程序进行了故障注入。其次，为了能够有效触发故障，通过 CTS 中人工输入测试用例的方式进行实验。

表 2　被测程序

程序	代码行数	函数数量
choice	373	10
draw	587	18
com	834	29
dir	691	17
view	400	10
edit	531	11
cmd	1116	46
key	153	7
run	189	12
help	380	6

为准确识别本文提及的两种执行时间异常故障，在程序中将进程挂起 5 秒。通过在程序执行后收集的故障监控探针函数返回信息可以得到故障检测结果，通常情况下，对于单次的测试用例生成是无法覆盖程序中所有路径，因此实验需要输入大量的测试用例从而达到覆盖所有触发故障的路径，但是可能会导致收集的信息包含了同一故障的多次记录。因此，在分析故障监控探针信息时，通过收集到的故障位置信息进行故障去重，将去重后的故障作为实验统计的最终结果。根据 3 种出错状态分别统计以本文方法检测出的故障，实验结果如表 3 所示。

表 3　故障检测结果

程序	程序状态出错				系统状态出错		执行时间异常	
	空指针引用	非法计算	数组越界	缓冲区溢出	内存泄漏	资源泄漏	低效循环	低效函数
choice	7	5	8	2	3	2	2	2
draw	10	7	7	3	5	3	3	3
com	15	8	10	5	8	3	5	4
dir	8	3	7	2	3	2	3	2
view	2	5	6	3	2	1	2	1
edit	10	5	2	3	11	2	5	3
cmd	22	12	16	12	12	8	7	12
key	5	3	2	2	2	1	2	3
run	2	3	8	2	2	1	3	3
help	5	4	2	3	4	2	1	2

实验结果表明本文所研究的八种故障模式均可通过该方法进行有效检测。根据实验所检测出的故障，在注入故障的程序中定位分析后发现所报故障均为真实存在且故障定位准确，不存在误报情况，该方法可有效弥补静态分析对于逻辑复杂软件进行故障检测时由于缺少运行时信息导致不精确的区间运算和符号执行产生的误报问题，大大减少了对于误报问题人工确认的成本。部分故障监控探针插装点位于程序中不可达路径上，无法通过生成测试用例执行覆盖到，已通过 CTS 自带的不可达路径[24]判断功能进行识别，实验符合预期。

5　结束语

本文针对静态分析出现大量误报的问题，结合静态分析和动态测试提出了一种基于软件运行特征的故障检测方法，通过研究按照出错状态划分为三类的八种故障模式，基于动态测试的插装技术，在传统用于覆盖分析的插装库中扩充了用于监控故障的探针函数。通过研究八种故障模式对应的故障特征，设计了相应的插装算法。该方法是对静态分析方法的重要补充，通过引入动态分析的方式在程序实际运行时收集程序路径上下文真实信息，避免了使用纯静态分析进行故障检测时完整路径上下文分析的组合爆炸[25]导致分析不准确的问题，有效弥补了静态分析高误报问题，可适用于逻辑较为复杂的程序。

该方法可通过收集触发故障对应的测试用例进行故障复现，帮助测试开发人员进行故障定位和修复。后续可以通过更精确的静态分析对故障监控探针插装点进行约简，但可能会导致插装性能下降，可以根据不同需求进行平衡。由于动态测试十分依赖输入的测试用例，不完备的测试用例会导致漏报[26]问题，因此触发故障导向的测试用例生成技术也将成为后续研究的方向。

参考文献：

[1] 宫云战. 软件测试 [M]. 北京：机械工业出版社，2008.

[2] 甘红星，金大海，宫云战. 基于源代码的内存泄漏静态分析方法 [J]. 内蒙古大学学报：自然科学版，2011，42（5）：515-520.

[3] 董玉坤. 基于属性可靠分析的空指针引用缺陷检测 [J]. 计算机工程与应用，2016，52（22）：7.

[4] 王雅文，姚欣洪，宫云战，等. 一种基于代码静态分析的缓冲区溢出检测算法 [J]. 计算机研究与发展，2012，49（4）：839-845.

[5] 梅　宏，王千祥，张　路，等. 软件分析技术进展 [J]. 计算机学报，2009（9）：1697-1710.

[6] 陈　厅. 动态程序分析技术在软件安全领域的研究 [D]. 成都：电子科技大学，2014.

[7] 周晓宇，黄文伟，史　亮，等. 基于源代码插桩的 C 程序内存使用错误动态检测 [J]. 舰船电子工程，2004，24（6）：70-73.

[8] HP Fortify [EB/OL]. [2021-12-22]. https://www.hp.com/us-en/services/managed-print-services/print-solutions.html.
[9] 李 筱，周 严，李孟宸，等. C/C++程序静态内存泄漏警报自动确认方法 [J]. 软件学报，2017，28 (4)：827-844.
[10] HUANG J C. Program instrumentation and software testing [J]. Computer，Apr，1987，11 (4)：25-32.
[11] 钟芳挺，刘 超，金茂忠. 程序动态分析系统中插装方式的改进 [J]. 计算机工程与设计，2007，28 (19)：4585-4588.
[12] HECKMAN S，WILLIAMS L. A systematic literature review of actionable alert identification techniques for automated static code analysis [J]. Information and Software Technology，2011，53：363-387.
[13] 赵云山，宫云战，周 傲，等. 静态缺陷检测中的误报消除技术研究 [J]. 计算机研究与发展，2012，49 (9)：1822-1831.
[14] 焦俊丽. 缺陷检测系统中警报自动聚类方法研究及实现 [D]. 北京：北京邮电大学，2018.
[15] ZHAO Y，WANG Y，GONG Y，et al. STVL：Improve the Precision of Static Defect Detection with Symbolic Three-Valued Logic [C] //Software Engineering Conference (APSEC)，2011 18th Asia Pacific. IEEE，2011：179-186.
[16] 王淑栋，刘 浩，董玉坤，等. 基于符号表达式的程序语义缺陷警报关联识别方法 [J]. 科学技术与工程，2020，20 (9)：3648-3655.
[17] 邵思豪，高 庆，马 森，等. 缓冲区溢出漏洞分析技术研究进展 [J]. 软件学报，2018，29 (5)：1179-1198.
[18] 孙华衿. C/C++单元自动化覆盖测试框架的研究与实现 [D]. 北京：北京邮电大学，2011.
[19] 肖 庆，杨朝红，毕学军. 一种基于故障模式状态机的测试方法 [J]. 北京化工大学学报：自然科学版，2007，34 (A01)：73-76.
[20] 匡海燕，张玉中，刘仁千，等. 基于 Qt 的软件内存泄漏静态检测技术研究 [J]. 计算机测量与控制，2019，27 (7)：36-39.
[21] 张贵民，李清宝，张 平，等. 基于运行特征监控的代码复用攻击防御 [J]. 软件学报，2019，30 (11)：3518-3534.
[22] 切 斯·韦斯特，董启雄，韩 平，等. 安全编程：代码静态分析：Secure programming with static analysis [M]. 北京：机械工业出版社，2008.
[23] 金凯峰，王雅文. 基于 Linux 平台的单元测试用例执行框架 [J]. 软件，2013，34 (12)：14-17.
[24] 陈 蕊，张广梅，李晓维. 程序中不可达路径的检测方法 [J]. 计算机工程，2006，32 (16)：86-88.
[25] 肖 庆，宫云战，杨朝红，等. 一种路径敏感的静态缺陷检测方法 [J]. 软件学报，2010 (2)：209-219.
[26] 李 雷，陈朝晖，李 轶，等. 软件故障定位技术研究综述 [J]. 计算机测量与控制，2019，27 (5)：1-4.

(上接第 36 页)

参考文献：

[1] 邓德平. 云计算下船舶网络入侵高精度检测系统设计 [J]. 舰船科学技术，2019，41 (20)：167-169.
[2] 庞学丰，王瑞文，张玉美，等. 基于机器视觉的航天器密封舱内结构装配精度检测系统设计 [J]. 计算机测量与控制，2020 (8)：53-57.
[3] 麻德明，刘焱雄，金永德，等. 面向对象的无人机遥感影像海岸线提取方法研究 [J]. 海洋科学，2020，44 (10)：46-51.
[4] 杨继文，刘欣岳，邓蜀江. 基于多时相遥感影像的海岸线变化监测研究 [J]. 测绘与空间地理信息，2020，43 (3)：115-116.
[5] 孙孟昊，蔡玉林，顾晓鹤，等. 基于潮汐规律修正的海岸线遥感监测 [J]. 遥感信息，2019，34 (6)：105-112.
[6] 王铁良，苏芳莉，董琳琳，等. 1985～2017 年辽河口滨海湿地海岸线变化特征 [J]. 沈阳农业大学学报，2020，51 (2)：129-136.
[7] 张 洁，倪小龙，刘 智，等. 高精度连续变倍率激光扩束系统设计 [J]. 中国光学，2019，12 (3)：693-700.
[8] 崔永俊，宋雪莹，刘 坤，等. 基于 FFT 的高精度相位测量系统设计 [J]. 电子器件，2019，42 (5)：114-118.
[9] 吴晓萍，徐涵秋. GF-2 PMS2 与 ZY-3 MUX 多光谱传感器数据的交互对比 [J]. 光谱学与光谱分析，2019，39 (1)：310-318.
[10] 张 禹，杨忠明，刘兆军，等. 大口径多光谱通道波前测量系统设计 [J]. 红外与激光工程，2020，49 (8)：155-161.
[11] 朱 琛，王小龙，崔 镭. 基于 DM6467 的图像伪装数字信号处理卡 [J]. 电子设计工程，2020，28 (12)：95-98.
[12] 顾振飞，袁小燕，张照锋，等. 基于透射图融合的红外图像传感器信号增强方法 [J]. 传感技术学报，2019，32 (7)：1070-1076.
[13] 董家臣，高钦和. 永磁直线同步电机电流环新型线性自抗扰控制 [J]. 电机与控制应用，2019，46 (1)：1-8.
[14] 任玲芝，余建立，许明坤. 基于动态电流注入技术的高线性混频器设计 [J]. 电子器件，2019，42 (6)：1358-1361.
[15] 李俊杰，傅俏燕. “高分七号”卫星遥感影像自动云检测 [J]. 航天返回与遥感，2020，41 (2)：112-119.
[16] 周 凯，任 怡，汪 哲，等. 基于主题模型的 Ubuntu 操作系统缺陷报告的分类及分析 [J]. 计算机科学，2020，47 (12)：35-41.
[17] 韩改宁，李永锋，高伊腾. 基于嵌入式 Qt 下的 MySQL 数据库设计与开发 [J]. 微型电脑应用，2020，36 (5)：29-31.
[18] 王桂霞，徐艳华. 一种高精度的激光传感器功耗智能检测系统 [J]. 激光杂志，2019，40 (11)：66-70.
[19] 荀 华. 基于生产管理信息系统的数据质量检查系统设计与应用 [J]. 内蒙古电力技术，2020，38 (5)：49-53.
[20] 黄凤荣，郭兰申，钱 法，等. 晶振误差对长航时高精度捷联惯性导航系统的影响 [J]. 科学技术与工程，2019，19 (5)：30-34.