ISSN 1000-9825, CODEN RUXUEW
Journal of Software, Vol.20, No.6, June 2009, pp.1425−1443
doi: 10.3724/SP.J.1001.2009.03526

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

软件错误注入测试技术研究*

陈锦富+, 卢炎生, 谢晓东

(华中科技大学 计算机科学与技术学院,湖北 武汉 430074)

Research on Software Fault Injection Testing

CHEN Jin-Fu+, LU Yan-Sheng, XIE Xiao-Dong

(School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, China)

+ Corresponding author: E-mail: cjfnet@126.com

Chen JF, Lu YS, Xie XD. Research on software fault injection testing. *Journal of Software*, 2009,20(6): 1425−1443. http://www.jos.org.cn/1000-9825/3526.htm

Abstract: The software fault injection testing (SFIT) technique has been developed for thirty years. It is one of the most active parts in software testing research. As a non-traditional testing technique, it plays a very important role in enhancing software quality, eliminating software failures and improving the process of software development. A detailed review of the research on SFIT is presented based on the survey and classification of the current SFIT techniques. Then, some important testing frameworks and tools that are effective at present are also discussed. Meanwhile, a brief description of the testing system CSTS (Component Security Testing System) is provided as well. Based on the precise investigation on SFIT, the issues and challenges of SFIT are pointed out and the future development trend for SFIT is proposed.

Key words: software testing; fault injection; software mutation; robustness; fault injection analysis

摘 要: 软件错误注入测试(software fault injection testing,简称 SFIT)技术经过近 30 年的发展,一直是软件测试领域最活跃的研究内容之一.作为一种非传统的测试技术,在提高软件质量、减少软件危害及改进软件开发过程等方面起着重要作用.对软件错误注入测试的研究现状及动态进行了调研,对该领域相关技术进行了归类及介绍,并对当前较为有效的测试框架和原型工具进行了总结,同时介绍了正在研发的基于 SFIT 技术的构件安全性测试系统 CSTS.在认真分析现有技术的基础上,总结了当前软件错误注入测试存在的问题和面临的挑战,并指出了其未来发展的趋势.

关键词: 软件测试;错误注入;软件变异;鲁棒性;错误注入分析

中图法分类号: TP311 **文献标识码**: A

错误注入(fault injection)技术作为一种非传统的软件测试技术是指按照特定的故障模型,用人为的、有意识的方式产生故障,并施加特定故障于待测系统中,以加速该系统错误和失效的发生.错误注入技术最早应用于 20 世纪 70 年代的硬件测试,后来根据不同的测试对象发展有基于硬件的技术[1]、基于软件的技术[2,3]、基于模拟的技术[4,5]和离子辐射的技术.软件错误注入测试(software fault injection testing,简称SFIT)是指采用软件的方

* Supported by the National Research Project Foundation of China under Grant No.513150601 (国家部委项目)
Received 2008-06-22; Accepted 2008-11-17

法对软件系统进行测试的技术,最早始于 1978 年DeMillo提出的程序变异测试[6].SFIT可以提高软件质量,评估软件等级.在一些高可靠性软件及安全关键软件等领域应用广泛,如航空、航天软件的测试等.

相对于传统的软件测试技术,SFIT主要有以下一些优点:1) 加速错误的发生;2) 能够测试COTS(commercial- off-the-shelf)软件.COST软件源代码不可知及高度独立,而使用SFIT可以在一些公共方法及API等接口层注入错误,进而观察和评估待测试软件;3) 增加测试覆盖性.对于一些小概率错误及难以达到的路径具有较好的测试效果;4) 能够进行变异分析.通过测试用例及变异充分度推测错误隐藏性.一般有 3 个分析过程,即执行分析(execution analysis)、感染分析(infection analysis)和传播分析(propagation analysis);5) 对系统鲁棒性和可靠性测试具有较好的测试效果等.有两类问题将引起软件问题,一类是有缺陷的软件,另一类是有缺陷的软件环境.当前绝大部分的软件保证手段都是解决第 1 类问题,但它却误导了软件质量的检测范围.软件错误注入技术包含很多方法试图解决上面提到的两类问题[7,8].SFIT技术早期主要用于程序变异测试、软件可靠性及容错能力评估,近几年在各种测试领域研究广泛,主要领域有利用错误注入提高软件测试覆盖性[9]、单元测试、集成测试[10]、COTS软件测试[11]、可靠性分析与测试[12]、容错分析[13]、安全脆弱性评估[14]、系统测试与维护、鲁棒性测试、风险评估和安全分析[12]等.特别是对安全关键软件及高可靠性软件的测试和分析一直是学术界的研究热点.

本文对 SFIT 的相关理论和技术进行了研究,分析和总结了当前 SFIT 的理论、方法及典型原型工具.同时,对我们的研究项目(基于 SFIT 的构件安全性测试理论与技术)的研究成果及现状进行了阐述.另外,在分析 SFIT 技术的基础上,总结了其目前存在的问题及面临的挑战,并指出了其未来发展趋势.

1 SFIT 基础

1.1 几个基本概念

定义 1. 软件错误(software fault) F 是一个三维空间,定义 $F=\{T,P,L\}$,其中 T 为错误发生的时间或错误持续时间,P 为错误类型或值的类型,L 为错误发生的位置.

定义 2. 错误注入属性集[15]是一个四元组:$\{F,A,R,M\}$.其中F为错误集合;A为激活集,描述目标系统的运行轨迹;R为读回集,收集系统在错误下的行为反应;M为度量集,收集错误注入测试的实验结果.一次完整的错误注入测试过程就是确定错误注入的属性集合.

定义 3. 软件可测试性是指软件在测试期间其错误被检测出的能力,或代码揭露问题的能力,或测试用例测试错误的能力.

定义 4. 软件失效是指软件的输出不符合软件需求规格说明或软件异常崩溃.

软件失效的 3 种解决方法是错误避免、错误消除及错误容忍.其中错误避免是在软件需求、设计及编码阶段采用标准化的方法来实现;错误消除一般是在软件测试及维护过程实现;错误容忍是在设计时已经考虑软件对错误的承受能力.软件错误注入测试技术一般用在错误消除过程及评估软件的容错能力.

定义 5. 软件变异(software mutation)测试是一种面向缺陷的软件测试方法,根据变异算子对程序作一些修改产生大量的新程序,每个新程序称为一个变异体.然后根据已有的测试数据、运行变异体、比较变异体和原程序的运行结果进一步分析和测试被测程序.软件变异包括代码变异和数据状态变异.

定义 6. 变异算子是模拟程序员错误或环境错误的规则.

定义 7. 软件鲁棒性(robustness)是指软件在异常输入、异常操作及异常环境下程序不中断、不崩溃及不产生错误输出的能力.

Du认为软件系统由应用程序和运行环境组成[16].所有被认为不属于运行程序的代码就属于环境.相应的软件错误注入方法可以分为应用程序错误注入和环境错误注入.故错误可以被注入到软件的输入、输出、中间代码、数据状态及运行环境.程序错误注入也可以理解为模拟程序员的错误以观察软件的行为.而环境错误注入是指在软件运行中对软件和操作系统交互的环境实体进行错误注入,常见的环境实体有环境变量、内存,网络、磁盘文件系统、外部组件及注册表等.

1.2 错误注入层次

为了更好地理解SFIT的优缺点,我们把错误注入分成低层次错误注入和高层次错误注入.低层次错误注入是指模拟硬件错误以验证系统反应的测试方法[17,18],而高层次错误注入是指模拟软件相关错误进行注入的测试方法[2,3,19],如模拟代码、数据、变量、程序状态及程序环境等.表 1 对两个层次的方法特点进行了对比分析.其中,SFIT工具是低成本、轻量级的错误注入工具.基于软件的方法也可以注入错误到软件低层模拟硬件错误,即采用软件的方法直接注入错误到硬件存储设备中,如CPU寄存器值、内存位置等.

Table 1 Comparison of fault injection level

表 1 错误注入层次对比

Item	Based on hardware		Based on software	
	With contact	Without contact	Static injection	Dynamic injection
Characteristic	Direct physical contact such as the pin injection, changing voltage and current,dynamic probe and chip pin etc.	Without direct physical contact such as radiation, electronic interference etc. Difficulty for controlling injection position.	Program modification before loading and running program.Injecting faults into source codes or assembly codes for simulating hardware faults and software faults etc.	Injecting faults when program running. With fault trigger mechanism such as timeslice, exception traps and code insertion etc.
Injection faults	Faking current,too much power and reversing bit etc.		Modifying program variables,statements,storage data (register, memory, disk) and communication data etc.	
Cost	High	High	Low	Low
Perturbation	No	No	Low	High
Risk of damage	High	Low	No	No
Repeatability	High	Low	High	High
Controllability	High	Low	High	High
Prototype tools	Messaline[20], FIST[21]	FIST[21], MARS[22]	PiSCES[23]	Ferrari[3],Ftape[8], Doctor[24], Xception[2]

1.3 SFIT基本原理及方法

SFIT 的目的是根据一定的错误类型模拟注入相关错误,以便观察系统的输入与输出并评估系统的鲁棒性和可靠性,最终避免在实际应用中发生软件失效并造成损失.通常,可注入的错误类型有:1) 内存错误;2) 处理器错误;3) 通信错误;4) 进程错误,如死锁、活锁、进程循环、进程挂起及使用过多的系统资源等;5) 消息错误,如失去消息、已损坏的消息、无序的信息、信息复制和组件间等待消息时间超时;6) 网络错误;7) 操作程序错误;8) 程序代码错误,如语句、变量等.

由第 1.1 节软件错误的定义可知,软件错误是一个由类型、时间及位置组成的三维空间,理论上分析是一个无限空间.穷尽所有的错误组合是不可行的,易产生组合爆炸.故选择合适的错误测试用例生成算法非常重要.错误注入测试用例产生中经常使用随机生成算法,如随机选择变异算子、随机选择注入点(位置)以及随机选择错误数据值等.随机生成算法具有一定的测试效果,但却不可避免地存在盲目性.当前,错误注入测试用例生成算法研究集中在自动化生成算法、最小 N 因素组合覆盖算法、环境自适应算法及智能错误注入测试用例生成算法上等.

错误注入的一般步骤[8]是:1) 分析当前测试对象、过程及测试目的;2) 采用一定的方法生成错误用例;3) 模拟错误类型;4) 管理错误注入过程,如错误覆盖及恢复机制等;5) 运行目标系统实施错误注入;6) 分析与评估测试结果及测试性能.

主要的软件错误注入方法有以下 10 种:

(1) 程序变异[25,26].程序变异是一种基于源代码的SFIT测试常用方法.主要有代码变异和数据状态变异两类,其中代码变异是直接修改源代码,从而改变程序执行状态;数据变异是指程序运行时修改程序的内部状态,如内存、全局变量及时间等.

(2) 软件陷阱.错误注入指令可以放置在目标程序的任何地方,当程序运行到设置的trace bit时,程序被切换到错误注入进程,这时由错误注入进程执行系统调用完成错误注入.典型的实现工具是FERRARI[3].

(3) 基于反射机制的错误注入方法[19,27].利用某些高级语言的反射功能实现错误注入,如FIRE方法.当程序运行时,通过反射机制获取元对象信息并改变方法调用顺序,从而执行错误注入.

(4) 封装测试法[28].增加适当的错误注入机制于被测试对象并封装成一个封装对象,然后对该封装对象进行错误注入.主要用在COTS软件的测试上.例如,FST(fault simulation tool)[29]工具,把DLL组件封装并用伪DLL进行替换实施错误注入,用来评估Windows NT系统的鲁棒性.

(5) 增加扰动函数法[7].通过扰动函数强制修改变量的内部状态实施错误注入.扰动函数通常用于源代码中,为了不被侵扰通常被编译成字节码或独立的程序体.如flipBit函数,倒置二进制位,即使指定位从0到1或从1到0,也可用于数据变异测试.

(6) 接口变异测试[10].修改组件接口层语法单元实施错误注入,如修改函数调用、调用顺序、返回值、参数及共享变量等.

(7) 断言违背机制.代表程序当前状态的断言(布尔表达式)在程序执行前,执行中及运行后都应为真值.真值代表当前程序运行正常,假值代表程序运行有错误.断言违背就是故意使断言为假的错误注入测试.另外,断言违背还可以增加测试覆盖性.

(8) 环境错误注入法[16,30].对运行时待测对象和操作系统交互的环境进行扰动,如注入内存、网络、磁盘文件等错误;而对基于消息的测试对象可注入通信时的环境错误,如注入多通信方进行压力测试及损坏消息错误等.主要技术是采用系统API截获法.

(9) FUZZ测试法[31,32],也称随机测试法.根据不同的待测试对象采用不同的随机函数生成FUZZ测试用例,目前主要有对文件名、文件格式、文件内容、函数参数、程序变量及内存值的FUZZ测试方法.FUZZ法容易实现,现实应用较多,但测试效率不高.

(10) 规约变异[33].该方法对软件需求设计时的规约(合约)进行变异,通过定义有效的变异算子来减少规约变异体的数量,从而降低变异测试的计算代价,提高测试效率.进一步可发现由于规约被错误理解或实现所导致的软件错误.

1.4 SFIT度量准则

常用的SFIT度量准则有以下几项[7,10,34]:

(1) 错误覆盖率 *FC*:衡量错误注入的覆盖性,*FC*=触发的错误异常数/注入的错误总数.

(2) 代码覆盖率 *CC*:衡量错误注入的代码覆盖性,有利于衡量一些小概率的程序代码及异常处理代码的执行情况.*CC*=错误注入后执行代码数/总代码数.

(3) 测试充分性 *TA*:由错误覆盖率 *FC* 和代码覆盖率 *CC* 的二元组来度量,一般取 *FC* 和 *CC* 的总得分来衡量,即 *TA=FC+CC*.

(4) 程序变异测试中的变异体充分度 *MA*:衡量变异测试中产生的测试用例的充分程度,$MA=D/(M-E)$,其中 *D* 为死去的变异体个数,*M* 为变异体总数,*E* 为与原来程序等价的变异体个数.

(5) 接口变异测试中的充分性度量准则:衡量接口变异测试中组件接口测试的充分程度,包含方法覆盖准则 *MC* 和接口变异度量准则 *IM*.设一个接口 *I* 中包含 *M* 个方法,则方法覆盖准则 *MC* 定义为:*MC*=(至少执行过一次的方法的数目/该接口中方法的总数 *M*)×100%.而接口变异度量准则 *IM*=被杀死的变异体个数/非等价的变异体总数.*IM* 比值越大,说明测试越充分,*IM* 是比 *MC* 严格的充分性度量准则.两者之间的关系是充分性准则之间的包含关系,即 *IM* 包含 *MC*.

(6) 通过PIE模型[35]进行错误注入分析的相关准则.利用错误注入技术在源代码中分析哪些地方测试用例不能揭露错误及揭错率等.分析算法有传播(propagation)分析算法、感染(infection)分析算法、执行(execution)分析算法及扩展传播EPA(extended propagation analysis)分析算法.

(7) AVA算法相关度量准则.AVA(adaptive vulnerability analysis)[36]是基于源代码的自适应脆弱性分析方法,通过模拟未知和已知的错误攻击来分析程序的脆弱性,详见第5.3节.

1.5 SFIT主要应用领域

由于SFIT技术的诸多优点,目前在众多领域得到了广泛的应用,概括起来主要有以下几个方面:

(1) 错误或失效容忍测试[15].评估系统在发生错误时能否继续正常执行而不影响其正常操作行为.错误容忍可理解为:1) 虽然程序有逻辑错误,但仍然能计算出可接受的结果;2) 虽然程序在执行期间接受了经篡改的输入数据,但程序仍然能计算出可接受的结果;3) 系统在异常环境下仍能正常运行.

(2) 鲁棒性测试[37].主要用于测试操作系统、应用程序、COTS软件、构件及服务协议等软件和协议的可靠性及健壮性.在操作系统和安全关键软件等一些重要软件的测试上尤为重要.采用的主要方法有扰动系统环境和封装测试法.一般通过错误注入模拟系统API功能产生异常以观察系统反应;或采用随机或智能的方式产生输入测试用例作用于待测试系统,从而测试应用程序的鲁棒性.此外,在软件接口层注入错误,建立可靠性矩阵,进而可分析与评估构件软件可靠性.典型的支持工具如Ballista[38],FUZZ[31],RIDDLE[28]等.

(3) 安全性测试与评估[14].主要用于安全关键软件、Web应用程序安全性测试和基于错误注入的软件安全性评估.通过先分析待测试系统的安全属性,然后模拟恶意和非恶意的攻击注入安全相关异常,最后观察待测试系统是否被攻破及攻破程度来测试与评估系统的安全性.自适应脆弱性分析法AVA[36]是一种基于错误注入的安全评估方法.此外,也有学者研究基于错误传播分析的软件脆弱点识别方法[39].渗透性测试[40]也是一种安全评估方法.该方法采用在单元和系统层模拟攻击者攻击被测试软件的方法来评估软件的安全性.

(4) 提高测试的覆盖性[9].采用软件错误注入的方法既可以提高软件测试覆盖率,又能缩短测试时间并降低测试成本.对容错软件可以用较少的代价对故障恢复代码和异常处理程序进行测试,以解决这部分代码难以测试的问题.

(5) 源程序变异测试[25].通过使用变异算子产生变异体系统地模拟程序中的各种错误,然后构造能够杀死这些变异体的测试用例集.变异测试具有排错能力强、方便灵活、自动化程度高等优点,既可以用来揭示程序错误,又可以用来衡量测试用例集的揭错能力,评估测试的充分性.当前在单元测试、面向对象软件的测试和合约测试[33]等方面都得到了广泛应用.

(6) 集成测试.软件单元集成时通过变异全局变量、调整单元调用顺序或在单元间的交互层注入参数个数、参数顺序、参数类型及参数值错误等方法测试软件集成时引入的软件缺陷,有时也称接口测试[10].

(7) 错误分析[35].通过对源代码中某些位置故意植入某些错误,然后产生测试用例执行测试.进一步利用传播分析、感染分析和执行分析模型分析程序中错误隐藏和暴露情况,同时也可以预测哪些错误难以被发现及测试用例充分性.此外,也可利用EPA及AVA算法进行错误注入后软件容错性及安全性分析与评估.

(8) 其他应用领域,如与时间相关错误测试及软件维护测试与评估等.

1.6 SFIT分类

纵观软件错误注入测试技术近 30 年的发展,软件错误注入测试技术在大体上分为静态错误注入测试技术和动态错误注入测试技术,如图 1 所示.静态错误注入测试是指待测试对象在运行前对其源程序进行变异或者编译时插入额外的错误代码所进行的测试;而动态错误注入测试是指待测试对象处于运行状态时对其程序状态进行变异或者对其运行系统环境进行错误注入时所进行的测试.

2 静态错误注入测试

静态错误注入测试主要有传统的程序变异测试、接口错误注入测试以及编译时错误注入测试.基本的错误注入测试步骤有:软件结构与错误管理分析、错误注入用例选择、测试计划、错误注入、执行测试触发、观察测试结果、评估测试结果、测试覆盖分析和更新错误注入测试用例库.

2.1 软件变异测试

2.1.1 基本概念

软件变异测试[25,26]是SFIT的最早应用,是一种面向错误的软件测试方法,主要用于程序单元及较小的模块单元测试.给定一组程序P及一组测试用例集T,变异测试的基本原理是:1) 使用变异算子对P作较小的变动,例如:将数学运算符“+”用“–”来替换等,产生大量的变异体$\psi(P)$;2) 然后运行任一变异体$M(M\in\psi(P))$观察输出结

果,如果M和P的运行结果不同,就称M被杀死了,否则M是活的.导致M不能被杀死的原因有两个:1) 测试用例集还不够充分;2) M为等价变异体.测试用例集T是充分的当且仅当:$\forall Q\in\psi(P)$,Q等价于P或Q与P在至少一个测试用例($t\in T$)上输出不同.软件变异测试本质上是一种测试用例生成方法,这些测试用例能够最大可能地揭露软件隐藏的错误[25,41].

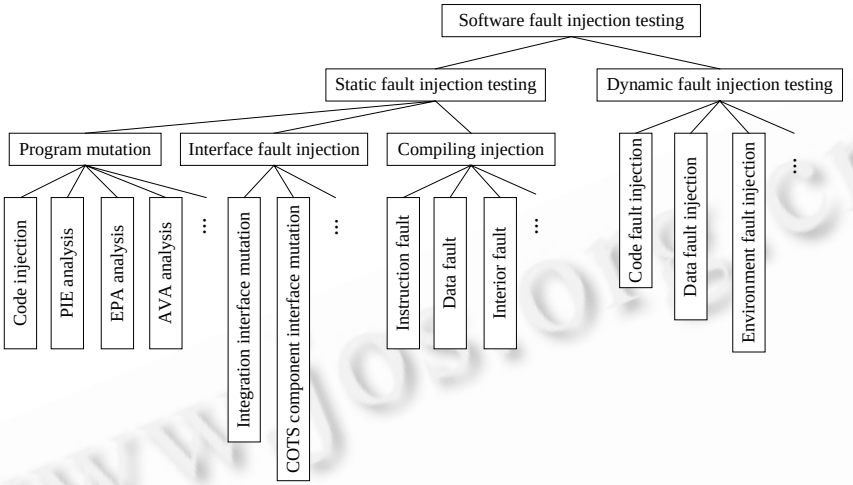

Fig.1 Classification of software fault injection testing technologies

图 1 软件错误注入测试技术分类

变异测试的一般步骤如下[34]:

(1) 根据变异算子产生被测程序 P 的一组变异体$\psi(P)$.

(2) 生成测试用例集.

(3) 对原来的程序和它的变异体都使用同一组测试数据进行测试,并记录它们在每一个输入值上的输出结果.如果一个变异体在某个输入上与原来的程序产生不同的输出值,则称该变异体被该输入数据杀死了.若一个变异体在所有的测试数据上都与原来的程序产生相同的输出,则称其为活的.

(4) 删除不能杀死至少 1 个变异体的测试用例.

(5) 分析剩下每个活变异体,或者生成一个测试用例杀死它,或判定该变异体是否等价于原来的程序.

(6) 对不等价于原来程序的活变异体进行进一步测试,直至充分性度量达到令人满意的程度.其中,变异体充分度=$D/(M-E)$,式中 D 为死去的变异体个数,M 为变异体总数,E 为与原来程序等价的变异体数.

最后两步需要消耗大量的计算机资源和人力资源,并且判断一个活的变异体是否等价于原程序是一个不可判定问题[42].所以一般采取减少变异体的数目,如弱变异方法[41,43].弱变异测试中变异体是在变异体执行以后与原始程序状态比较,而不是在整个程序执行以后,即在执行变异语句、块、表达式等变异体后即可与原始状态比较,所以其测试开销明显比(强)变异测试小却仍然有较好的测试效果.但弱变异测试仍然没有解决判定变异体等价的问题.

2.1.2 变异算子及变异测试用例生成

变异测试系统地模拟软件中程序员的各错误,并且寻找能够发现这些错误的测试用例集.实验结果表明,变异体与真实的错误非常相似[26].测试对象不同,变异算子也有差异,例如,有基于Ada语言、Fortran语言及C语言等语言的变异算子.常用的变异算子见表 2[44].变异测试代价昂贵,通常采取选择性变异方法可减少测试成本.根据变异程序的具体情况,可以从表 2 中的 22 种常用的变异算子选择若干变异算子进行测试.典型变异工具有Mothra[45].

有了变异算子,变异测试中至关重要的还有生成能够杀死变异体的测试用例集.手工生成测试用例集耗时费力,目前主要采取自动生成的方法.自动生成测试用例集的方法主要有基于约束的生成方法(constraint-based

test data generation,简称CBT)[46]、动态域削减方法(dynamic domain reduction test data generation,简称DDR)[47]和杀死多个同位变异体的方法[25].CBT采用控制流分析及符号执行建立约束关系,然后求解约束关系得到测试用例集.CBT具有较好的生成效果,但其难于分析处理依赖于输入变量的判断条件和表达式,且可能导致有解的约束关系无解.动态域削减DDR方法弥补了CBT 方法的不足,其动态特性改进了对判断条件和表达式等的处理,而且根据动态控制流图求解约束关系效率较高.虽然DDR方法改进了CBT方法,但其都生成针对杀死单个变异体的测试用例,测试效率不理想.为了进一步提高生成测试用例集的效果和效率,减少测试开销.文献[25]提出一种生成杀死多个同位变异体的测试用例方法,该方法根据同一位置多个变异体条件相近的特点,生成同时杀死该位置多个变异体的测试用例.该方法与面向路径的测试数据自动生成方法相结合,具有更好的测试效果.

Table 2 Mutation operators

表 2 常用变异算子

No.	Mutation operator	Description	No.	Mutation operator	Description
1	AAR	array reference for array reference replacement	12	GLR	GOTO label replacement
2	ABS	absolute value insertion	13	LCR	logical connector replacement
3	ACR	array reference for constant replacement	14	ROR	relational operator replacement
4	AOR	arithmetic operator replacement	15	RSR	RETURN statement replacement
5	ASR	array reference for scalar variable replacement	16	SAN	statement analysis
6	CAR	constant for array reference replacement	17	SAR	scalar variable for array reference replacement
7	CNR	comparable array name replacement	18	SCR	scalar for constant replacement
8	CRP	constant replacement	19	SDL	statement deletion
9	CSR	constant for scalar variable replacement	20	SRC	source constant replacement
10	DER	DO statement end replacement	21	SVR	scalar variable replacement
11	DSA	DATA statement alterations	22	UOI	unary operator insertion

2.2 接口错误注入测试

静态接口错误注入测试有时也叫接口变异测试[10].接口变异的概念最早由Delamaro等人[10]提出,并首次把程序变异用于集成测试.在这之前一般将程序变异的测试方法用于单元测试级别.目前发展成为面向系统集成的接口变异测试和面向COTS构件的接口变异测试.面向系统集成的接口变异测试基础是将变异体建立在模块或子系统集成时的接口上,即仅关注模块之间集成错误或接口错误,而不考虑模块的内部错误.相对于传统的程序变异测试,接口变异测试减少了变异开销和测试成本,具有更高的实用价值.

随着构件化软件工程的进一步发展,构件软件系统中经常采用 COTS 构件.COTS 构件通常是不提供源代码,且高度独立,传统的接口变异测试方法不适用于 COTS 构件系统的测试.面向 COTS 构件的接口变异测试通过扩展程序变异中的变异算子和增加构件接口层的变异算子在构件接口层进行错误注入,然后观察分析系统测试结果,找出构件的缺陷.

面向COTS构件的接口变异算子包括基于程序语言的变异算子、基于接口规约的变异算子和基于IDL语言的变异算子[10].常用的接口变异算子有参数属性替换、参数互换、参数增减、参数赋值及参数置空等.根据接口变异算子和接口变异充分性准则MC和IM,面向COTS构件的接口变异测试的一般步骤如下:

(1) 分析待测构件的所有接口得到接口集合$I=\{i_1,i_2,\ldots,i_n\}$及方法集合$M=\{m_1,m_2,\ldots,m_n\}$等信息;

(2) 为每一个$i_i\in I$,选择接口变异充分性准则$MC(IM)$;

(3) 为任意一个$m_i\in M$,根据变异算子生成变异体;

(4) 生成测试用例并执行构件;

(5) 如果发现错误则记下,否则转下一步;

(6) 若变异体是活的且 $MC(IM)$没有达到要求,则扩充测试用例以提高 $MC(IM)$;

(7) 调整接口变异充分性准则 $MC(IM)$,转第(3)步;

当前接口错误注入测试中还有封装器测试法[10],把构件接口封装后注入错误,运行构件并监测接口输入与输出.此外,接口传播分析也是接口错误注入测试中的一个研究热点,目的是分析一个构件错误注入后的失效是否会影响到另外一个构件的失效,最后导致整个构件系统的失效.

2.3 编译时错误注入测试

在被测试对象运行前的编译期间加入额外的错误代码,如JAVA语言程序的字节码错误注入测试[48]等.主要通过错误注入器修改汇编代码或目标代码进行错误注入,错误被硬编码到编译代码中,能模拟硬件、软件及短暂性错误等.编译时,错误注入器模拟的错误类型主要有:

(1) 指令错误:被注入到被编译的用户代码装载的处理器内存区域;

(2) 数据错误:被注入到当前低优先进程工作区;

(3) 内部错误:被注入到内存位置处的数据区.

3 动态错误注入测试

动态错误注入测试是指待测试对象处于运行状态时对其程序状态进行变异或者对其运行系统环境进行错误注入所进行的测试.程序状态变异一般采取修改、插入系统目标代码和断言违背的方法进行测试.在软件运行期间注入错误时,程序状态变异要求必须有触发错误注入的机制.触发错误注入的机制主要有:1) 定时机制:时间片到即可注入错误,适合模拟短暂性错误和间歇性错误;2) 异常 traps 机制:当一定的异常、条件或事件发生时触发错误注入;3) 代码插入或修改机制:运行时在一些代码前插入代码或修改代码.断言违背错误注入测试中,前置条件和后置条件蕴涵了程序期望的逻辑断言.前置条件是一个有关系统状态属性的断言,该断言必须返回真,以确保前一个函数调用为真时该函数将正确运行.后置条件是一个描述输入状态和正确函数输出状态之间的关系.实现错误注入的方法是在程序运行时动态地违反执行条件.当定时机制或异常 traps 机制触发错误注入时,动态错误注入通常是通过修改寄存器值或改变内存位模拟错误,改变的内存位置包含程序指令位和数据位.当然,动态错误注入可以注入更广泛的错误,如硬件错误中的总线错误、不正确的 CPU 指令等.

目前,广泛使用的是对被测系统进行动态环境错误注入测试[49].基本原理是软件运行时对其环境实体如环境变量、寄存器、内存、磁盘文件系统、网络、注册表及系统调用等进行错误注入测试.其中最常用的实现方法是对操作系统和运行软件之间的系统调用进行截获.截获后通过修改系统DLL的函数参数、返回值等来注入错误,然后返回给软件调用.截获的系统调用主要涉及文件操作、内存操作、网络操作以及外部DLL操作等.通常其截获方式有 3 种[49]:基于源代码的截获、路由截获、基于目标的截获.

最近几年,基于动态错误注入技术的构件鲁棒性测试研究如火如荼,详细内容将在第 4 节进行专题介绍.此外,典型的运行时错误注入工具主要有:Ferrari[3],Ftape[50],Doctor[24],Xception[2],Fiat[51],Holodeck[30,49]等.

4 基于错误注入的构件鲁棒性测试

4.1 研究现状

目前,越来越多的商业软件包括一些安全关键软件使用COTS构件,如军事、医疗、银行及金融等行业.COTS构件的鲁棒性决定了整个软件的可靠性与安全性,测试COTS构件成了重要一环,也是较为困难的一环.图 2 是COTS构件鲁棒性测试的一般步骤,图中虚线框为根据不同的测试方法采用的不同错误注入模块.当前的一些研究也主要采用了随机法(FUZZ)[31,32]、封装测试法[28]、API扰动法[38]和环境错误注入法[16,30],如FUZZ[31],FST[29],Ballista[38]等COTS构件测试工具.FUZZ,FST和Ballista都是较早用于测试系统构件鲁棒性的工具,且均具有一定的测试数据自动化生成能力.其中,Ballista通过给API函数注入非法的输入参数来测试COTS系统构件的鲁棒性,具有较好的测试效果.Nathan的技术报告[52]研究了对不同系统的测试结果,结果也表明基于API参数的错误注入具有较好的测试效果.此外,可靠软件技术公司的Ghosh等人[53]一直致力于基于Windows系统的COTS软件测试的研究.采用的方法是在COTS软件和OS之间开发一个封装器,用于对OS的系统调用进行封装.封装器的主要功能是模拟和处理系统异常,并分析COTS软件在异常下的反应以评估COTS软件的鲁棒性.开发的典型工具是FST.

近两年的研究集中在基于错误注入的COTS构件安全性测试技术[54,55]、基于错误注入技术的实时COTS软

件的鲁棒性验证与评估[11]和利用错误注入技术加固COTS软件的鲁棒性和安全性[56].在此仅对利用错误注入技术对COTS软件的鲁棒性和安全性进行加固技术作一介绍[56].其工作过程是:首先静态分析COTS构件的接口信息及运行环境,找出可能的错误注入点;然后动态注入一些假设的常用错误参数并观察构件反应;最后总结、分析输入参数和发生异常的情况,对COTS构件进行封装并增加对异常输入的处理功能以加固构件的鲁棒性和安全性.

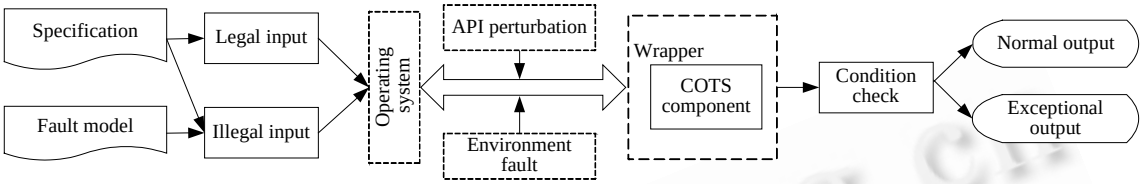

Fig.2 Testing process for COTS component robustness

图 2 COTS 构件鲁棒性测试过程

4.2 基于错误注入的构件安全性测试

4.2.1 错误注入模型及测试方法

随着构件化软件工程的发展,基于构件的软件开发技术进一步提高了软件开发效率及软件性能,但构件的安全性问题一直困扰着构件的开发方和使用方.构件安全测试是保证构件系统质量的一种有效手段,但目前构件安全性测试的研究成果并不多见.在国家项目的支持下,我们一直致力于基于错误注入的构件安全性测试研究,相继提出了构件安全测试错误注入模型(fault injection model,简称FIM)、基于错误注入的测试方法[55]、基于动态监测的测试方法[54]以及设计实现了一个构件安全测试平台[57].

设用 R 和 T 分别表示待测试构件 CUT(component under testing)的错误注入测试要求集和错误注入测试用例集,并假定两个集合都是非空有限集.对于错误注入要求集中的任一子集 $R'(R'\subseteq R)$,在测试用例集 T 中都存在子集 $T'(T'\subseteq T)$覆盖该测试要求集.

定义 8. 错误注入模型FIM是一个三元组$\{R,T,S\}$,其中R为错误注入测试要求集;T为错误注入测试用例集;S为二元关系$S(2^R,2^T)$,即$S(2^R,2^T)=\{(R',T')\in 2^R\times 2^T$,测试用例集$T'$覆盖测试要求集$R'\}$.

定义 9. $R=\{IP,M,DF,PRS,NET,REG\}$,其中 IP 为接口参数,M 为内存,DF 为磁盘文件系统,PRS 为外部构件或进程,NET 为网络,REG 为构件注册信息.规定了对 CUT 进行错误注入的 6 个方面.实际应用中,根据不同待测试对象的不同测试要求,R 可以进行修改和扩充.

图 3 是一个构件安全测试错误注入模型简略图.根据 COTS 构件的特点,我们提出的测试方法采用静态接口参数错误注入和动态环境错误注入相结合的方式来实施错误注入测试,多方面测试 COTS 构件的安全性.接口参数错误注入首先分析构件接口信息,然后对输入参数、参数类型、参数个数、参数顺序等进行错误注入.而环境错误注入是指在构件运行时对构件和操作系统交互环境进行错误注入.其错误注入层次如图 4 所示.即对运行时的构件,从内存、磁盘系统、进程、网络及注册信息等环境实体进行错误注入.内存错误注入主要是限制构件访问的内存大小、内存损坏等;磁盘文件错误注入主要是限制对文件的读写访问、文件损坏等;进程错误注入主要有进程资源不足、调用构件不能找到等;网络错误注入主要是在网络线路及网络构件上进行模拟错误注入,如网络断线、拒绝服务等;注册信息错误注入主要有恶意修改注册表信息、拒绝访问等.

对于给定的被测试 COTS 构件,基于 FIM 模型的错误注入算法如下:

(1) 如果构件接口有输入参数,则进行接口参数错误注入,否则执行步骤(2).同时,记录注入结果;

(2) 对被运行构件进行内存错误注入,同时记录注入结果;

(3) 分别对被测试构件进行文件系统、进程、注册信息及网络错误注入,并记录注入结果;

(4) 对被测试构件执行最小两因素组合覆盖错误注入,并记录注入结果;

(5) 对被测试构件执行最小三因素组合覆盖错误注入,并记录注入结果;

(6) 对上述记录的错误注入测试结果进行异常分析总结并生成测试报告.

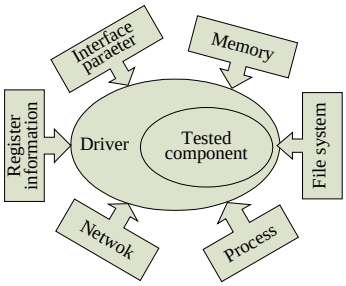

Fig.3 A brief figure of fault injection model
图 3 一个错误注入模型简略图

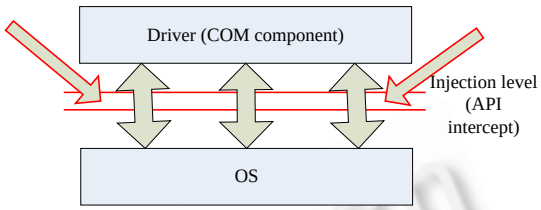

Fig.4 Framework of environment fault injection
图 4 环境错误注入框架

4.2.2 构件安全测试系统 CSTS

基于 FIM 模型及相关错误注入测试方法,我们设计并实现了一个构件安全测试系统:CSTS(component security testing system).CSTS 是在 WindowsXP 平台上用 VC6 和 Visual Studio .NET 2005 C#联合开发的一个针对微软 COM(component object model)构件安全性测试的集成测试系统原型,主要包含以下主要功能:1) 对 COM 构件进行接口分析,得到必要的接口信息,同时生成构件安全需求说明;2) 自动生成构件方法级测试程序,通过进一步生成接口方法测试用例实现接口参数错误注入测试;3) 编译测试程序,生成构件测试驱动;4) 基于 FIM 模型对被驱动运行的 COM 构件进行环境错误注入测试;5) 动态监测被测试 COM 构件的运行情况,记录构件安全异常,并写入安全日志文件;6) 根据构件安全漏洞检测算法及评估算法对 COM 构件进行安全评估与分析.各功能模块详细信息见文献[57].

5 错误注入测试分析

错误注入测试分析是一项基于错误注入测试的分析技术,主要包含 PIE 分析(也称敏感性分析)、扩展传播分析及自适应脆弱性分析.其主要功能分别是分析程序位置错误传播感染的概率及评估测试用例的测试能力、分析程序的容错能力及安全性和分析基于源代码的程序自适应脆弱性.错误注入分析技术是一项相对的分析技术,其分析结果与程序位置、测试用例数目及选择测试用例的次序等均有关.

5.1 PIE算法

PIE(propagation infection execution)[35]模型由传播、感染及执行 3 种算法组成,3 种算法基于软件的重复执行,并从系统中收集统计信息预测系统隐藏错误的概率[58].传播和感染算法使用了错误注入且准确性较高,此外,软件需要执行每个位置来寻找故障,具有一定的测试开销.主要功能有:(1) 通过PIE算法统计程序各个位置的相应参数得分,分析结果表明,应如何测试程序及生成什么类型的测试用例才能最大可能地揭露错误;(2) 帮助生成更好的测试用例;(3) 执行分析算法可以显示测试覆盖性;(4) 根据位置数、数据状态感染数、注入数据状态变异数来对测试用例进行优劣排序;(5) 通过PIE技术使形式化验证和测试更有机地结合,对于不适合测试的小规模或局部代码采用形式化验证方法.

根据 PIE 模型,一个错误要发生必须满足以下 3 个条件:(1) 执行一个错误输入;(2) 在执行后错误必须感染数据状态;(3) 感染的数据状态必须传播到输出.3 个条件如有一个未发生,则错误将被隐藏.表 3 归纳总结了原始 PIE 算法的内容、优缺点等.针对 PIE 算法的不足,有关学者陆续提出了修改的 PIE 算法.修改后的算法其原理基本一致,仅在测试用例选取及结果分析上作了优化.

5.2 扩展传播EPA分析

扩展传播分析(extended propagation analysis,简称EPA)[59]是一种基于源代码改变程序状态的错误注入测

试分析方法,而修改程序状态通常可以通过修改程序变量、构件间的通信以及内存、磁盘或数据库中的值来实现.一个程序状态,有时也称为数据状态,是一个所有变量和变量在软件执行时特定位置取值之间的映射集.EPA的目的是通过修改程序状态内容来观察程序的输出,进一步分析程序的容错能力及安全性.

Table 3 Comparison of PIE algorithm

表 3 PIE 算法对比分析

Item	Algorithm *P*	Algorithm *I*	Algorithm *E*
Main steps	01 Set **counter** to zero; 02 Randomly select a data state that occurs after location *l* according to *D* (*D* is the testing distribution); 03 Perturb the data state for some variable *a* and continue execution; also perform a different execution using that data state without perturbing it; 04 Compare resulting outputs; if different, increment *counter*; 05 Repeat Steps 2~4, *N*–1 additional times; 06 The propagation estimate for variable *a* at location *l* is **counter** divided by *N*; 07 Perform Steps 1~6 for different variables; 08 Perform step 7 for different locations in the code.	01 Set **counter** to zero; 02 Create a mutant *M* of location *l*; 03 Execute the location *l* and mutant *M* on a randomly selected data state that occurs before *l* according to *D*; 04 Compare resulting data states, if different, increment counter; 05 Repeat Steps 3 and 4 *N*–1 additional times; 06 The infection estimate for mutant *M* at location *l* is counter divided by *N*; 07 Perform Steps 1 through 6 for different mutants; 08 Perform Step 7 for different locations.	01 Set $\mathbf{counter}_l$ to zero; 02 Increment $\mathbf{counter}_l$ each time location *l* is executed; make sure that $\mathbf{counter}_l$ is incremented at most once per input, hence if *l* is repeatedly executed for the same input, counter is only incremented once; 03 Execute the code with *N* test cases selected according to *D* (*D* is the testing distribution); 04 The execution estimate for location *l* is $\mathbf{counter}_l$ divided by *N*; 05 Perform Steps 1-4 for all locations in the code.
Functions	Analyzing the probability of hiding (propagating) faults or the ability that test-cases reveal faults.	Analyzing the probability of infecting faults or the ability that test-cases reveal faults.	Analyzing the testing coverage of test-cases in program statements.
Evaluation parameters	$\boldsymbol{Counter_{la}/N}$:Propagation evaluation about *l* and *a*.	$\boldsymbol{Counter_{lm}/N}$:Infection evaluation about *l* and *m*.	$\boldsymbol{Counter_l/N}$: Testing coverage evaluation about *l* and test-cases.
strongpoints	High accuracy	High accuracy	Fast
shortcomings	High cost, hundreds of locations	High cost, lots of mutations.	Low accuracy
Prototype tools	Yes	Yes	Yes
Remarks	Because of lots of program statements and high cost, analysis locations will be reduced to improve the efficiency.		
	Three ways of disturbing data state: Flip one or more bit data;interfere with current value;assign new value regardless of original value. Typical disturbed functions are flipBit, allBitsHigh etc.	Generate mutation versions according to the mutation operators,selective mutation is commonly used.	E algorithm is a simple algorithm, it is often applied in the high-reliable large-scale systems.

为了唯一地识别语法结构,EPA定义了一些语句位置,如赋值语句、输入/输出语句、if语句及while语句等.设l是一个语句位置,C是一个程序,x是程序输入,Δ是程序C所有的合法输入,Q是Δ的概率分布,i是由x驱动的l的一次特定执行,A_{lCix}是程序C在输入x后位置l的第i次执行的程序状态,m_{xl}是输入x后l的执行次数,那么所有程序状态的集合是:$A_{lCx}=\{A_{lCix}|0\le i\le m_{xl}\}$,所有合法输入下程序状态的集合是:$\alpha_{lC\Delta}=\{A_{lCx}|x\in\Delta\}$,$A'_{lCix}$是错误注入以后的程序状态集合,$PRED_C$是异常输出的逻辑断言.修改程序状态的常用方法有随机修改状态值、输入异常值及输入边界值等方法.EPA算法[59]如图 5 所示.

01 For each *l* in *C*, perform Steps 2-7;
02 Set **count** to 0;
03 Randomly select an input *x* according to *Q*. If *C* halts on *x* in a fixed period of time, find the corresponding A_{lCix} in $\alpha_{lC\Delta}$. Set *Z* to A_{lCix};
04 Alter *Z*, and denote the new data state as *Z*′. Execute the succeeding code on *Z*′;
05 If the output satisfies $PRED_C$, increment **count**;
06 Repeat Steps 2~4 *n*–1 times;
07 Divide **count** by *n* yielding ψ_{alCQ}. The numerical value, $1-\psi_{alCQ}$, is a measure of fault tolerance, and the value is the proportion of times *C*'s output was of the undesirable type.

Fig.5 EPA algorithm

图 5 EPA 算法

算法的结果$\psi_{alCQ}=count/n$,ψ_{alCQ}是程序C失效的比率,而 $1-\psi_{alCQ}$代表程序C的容错能力.此外,EPA中修改程序状态的错误模拟方法还有:模拟组合错误、模拟程序计数器错误及模拟错误持续时间.

5.3 自适应脆弱性分析

AVA(adaptive vulnerability analysis)[36]是基于源代码的自适应脆弱性分析方法,通过模拟未知和已知的错误攻击来分析程序的脆弱性.当前AVA在基于源代码的缓冲区溢出漏洞检测上应用较广,主要通过缓冲区溢出扰动函数注入超长字符长串、异常数值及边界值等异常值来分析程序的缓冲区溢出漏洞.

AVA 的理论模型包含以下 3 点:(1) 通过错误注入模拟程序代码脆弱性;(2) 自动产生测试用例;(3) 定义一个检查脆弱性的断言 PRED.AVA 的基本框架如图 6 所示.

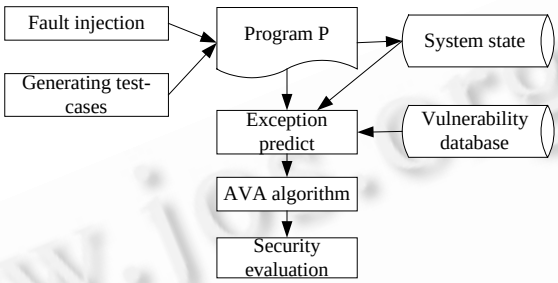

Fig.6 AVA framework

图 6 AVA 方法框架

设P是一个程序,l是P中一个语句位置,a是l中的一个变量,x是程序输入,T是脆弱性错误测试用例集,$T=\{t_1,t_2,t_3,\ldots,t_n\}$,$\Delta$是程序$P$所有输入集,$Q$是$\Delta$的概率分布,$Q'$是$\Delta$的逆概率分布,PRED是异常入侵的逻辑断言.AVA算法[36]如图 7 所示.

01 For each location l in P that is appropriate, perform Steps 2-7;
02 Set count to 0;
03 Randomly select an input x or input sequence from Q or Q',and if P halts on x in a fixed period of time, find the corresponding data state created by x immediately after the execution of l, Call this data state Z;
04 Alter the sampled value of variable a found in Z creating Z', and execute the succeeding code on Z'. The manner by which a is altered will be representative of the threat class from T that is desired;
05 If the output from P satisfies PRED, increment count;
06 Repeat Steps 3~5 n times, where n is the number of input test cases;
07 Divide count by n yielding ψ_{alPQ}, a vulnerability assessment, for each line l. This means that $1-\psi_{alPQ}$ is the security assessment that was observed, given P, Q and T.

Fig.7 AVA algorithm

图 7 AVA 算法

算法结果$\psi_{alPQ}=count/n$,ψ_{alPQ}是程序P在位置l处的脆弱性比率,而 $1-\psi_{alPQ}$代表程序P在位置l处的安全性.进一步设M为P的位置数,θ_{lPQ}为P在l处执行概率,ρ为单位时间P执行次数,则P的平均入侵时间MTTI及最短入侵时间 MinTTI 分别为 $MTTI=\left[\frac{\sum_{l=1}^{M}(\psi_{\text{alPQ}}\cdot\theta_{lPQ})\cdot\rho}{M}\right]^{-1}$ 及 $\text{Min}TTI=[\max_l(\psi_{\text{alPQ}}\cdot\theta_{lPQ})\cdot\rho]^{-1}$.

6 SFIT 原型工具

SFIT 工具(SFITts)是一类对被测试软件故意注入错误以检验其是否满足规定的需求或评价其失效情况的软件测试工具.SFITts 主要可以完成下列测试任务:1) 根据规约生成程序错误注入测试用例;2) 验证软件是否符合软件需求规格说明;3) 对软件运行环境进行错误扰动;4) 评估软件的容错性与安全性;5) 测试软件的鲁棒性;6) 动态监测软件的运行情况.当前存在的软件错误注入测试原型工具很多,表 4 仅给出了典型 SFIT 原型工具的概况.

Table 4 Some representative SFIT tools

表 4 典型 SFIT 原型工具

Tools	Injecting technology	Injecting fault type	Appli. scene and scopes	Strongpoints	Shortcomings	Static/ Dynamic	Remarks
FERRARI[3]	Exception traps mechanism	Register, memory, bus, etc.	COTS software	Simulate hardware faults; Two concurrent injection processes	Additional exception handling code	Dynamic	Traps are triggered by program counter or timer
Xception[2]	Debugger, inject codes, built-in fault trigger	CPU, register, memory, etc.	Appication, COTS software	Program to hardware in debugger; Don't modify goal program	Memory and register values are destroyed	Dynamic	Fault injection unit becomes exception manage unit
GOOFI[63]	Set breakpoint and built-in logic, etc.	Modify variable values, inject memory faults, etc.	Appication, COTS software	Record injecting process	Breakpoint affects efficiency	Dynamic	Provide support platform for new injecting tech.
DOCTOR[24]	Overtime,traps and code modification	CPU, memory, net communication, etc.	Distributed real-time system	Simulate permanent and temporary faults; Data collection and analysis	Additional exception handling codes	Dynamic	Initialize time to determine when faults injected
FST[29]	Faults injected into wrapper, exception call	Parameter, memory, interface, etc.	COTS component	Wrap binary code, such as dll, etc.	Inject faults on purpose, exception call	Static dynamic	Change function call by application import address
FTAPE[50]	Deal with binary bit to simulate faults	CPU, memory, disk I/O, etc.	Appli. fault tolerance, performance evaluation	Inject lots of faults	Embedded perturbation function of binary bit	Dynamic	Inject faults into CPU register, memory and disk
RIDDLE[28]	Randomly inject interface faults based on specification	Interface abnormal value, border and random value, etc.	COTS component robustness	Randomly and intelligently generate test cases	Need testing specification	Dynamic	Using black box testing technology for generating test cases
Holodeck[30]	Environment fault injection	Memory, disk, network, process, etc.	Application security testing	Don't modify goal program, powerful monitoring	Low efficiency, poor stability	Dynamic	Intercept operating system API
Ballista[38]	Inject faults into API function	Illegal input parameters, memory, file, etc.	System component robustness	Automatically generate test case	Pre-Analyze system API library	Dynamic	API interception
FINE[64]	Implant codes, probe tech.	Memory, bus, CPU, I/O, etc.	UNIX system reliability	Powerful monitoring function	Easily damage address space of application	Dynamic	Propose fault propagation model
ExhaustiF[65]	Trigger and interception technology	Modify program variable, CPU, memory, etc.	Distributed system reliability	Inject faults into hardware and software	High resources cost	Dynamic	Some application in commerce
FIAT[51]	Software simulates hardware faults	Modify program variable, CPU, memory ,etc.	Application reliability	Automatic testing environment; Integrate faults database	Complicated fault injection process	Dynamic	Software simulates hardware faults
JACA[19]	Program mutation, parameter fault injection	Modify program attributes, method parameters, return values, etc.	JAVA program	Good fault model	Depend on reflective mechanism	Dynamic	Apply in assembly code fault injection after minor modification
BOND[66]	Perturbation technology	Inject faults into code/data seg., register, etc.	Source code, COTS software	Small impact, don't modify target program	User need expertise	Dynamic	Inject faults into program different levels and locations
CSTS[57]	Interface param. injection, API interception	Interface param., memory, file, etc.	COTS Component security	Static and dynamic fault injection	No good evaluation mechanism	Static dynamic	Current version only for COM component
PiSCES[23]	Program mutation	statements, variables, data state, etc.	C/C++ program	Predict software testability	No judge mechanism of equ. mutation	Static	Program mutation tool based on PIE model
Mothra[45]	Program mutation	Code and data state mutation	Fortran program	User-Friendly interface	No good standard for gen. test cases	Static	Unit testing
WhiteBox SafetyNet[67]	Program mutation, data perturbation	Code and data state mutation	Source program	Graphic interface, fault tolerance analysis; Disturbance function is stored in other library	Fault injection tool is embedded in source codes	Static	Fault injection technology based on EPA
SIMPLE[68]	Fault trigger and program mutation	Code and data state mutation	JAVA program	Semi-Automatic; easy-to-use	Need config file; Low performance	Static Dynamic	State-Based testing, verify system robustness

另外,还有基于网络层的脚本驱动错误注入器Orchestra[60]、基于错误注入的网格与Web服务可靠性评估工具Grid-FIT与WS-FIT [61]及模拟真实软件错误评估软件可靠性和容忍性的G-SWFIT[62]工具等一些新兴SFIT原型工具,限于篇幅,在此不作讨论.

7 SFIT 存在的问题及其研究展望

在了解 SFIT 各项研究领域、研究现状以及相关技术的同时,必须充分意识到 SFIT 目前存在的问题和挑战.另外,也应看到 SFIT 的未来发展趋势.

7.1 存在的问题

SFIT 在为用户提高测试效率及保障软件质量的同时,还存在一些问题.既有技术上的因素,也有设计的原因,还有用户的需求影响.概括起来主要有以下几个方面:

(1) 错误注入措施改变了系统的运行状态并影响了被测试系统的性能

无论是静态错误注入还是动态错误注入,都将给被测系统带来额外的负荷,进一步易产生 Heisenbugs(间歇性、不确定性错误).静态错误注入在代码执行前注入相关错误,虽然在程序运行时无须控制或扰动即可完成错误注入.但因缺乏控制及交互,无法获知错误被激发时间及对被测试系统的影响,可能改变系统的运行状态.运行时,错误注入由外部错误注入机制对被测系统进行控制、监测和分析,这样将影响被测系统的行为和性能,甚至改变原始软件结构,增加系统的负荷.

(2) 不同环境不同系统不同类型的错误模拟成为难点

软件错误注入测试的本质是模拟各种不同的错误,包括硬件和软件错误.一些系统复杂而多样、代码量大,准确而全面模拟真实的错误很困难.即使能够模拟,穷尽测试也是不现实的,错误注入测试技术也不能满足其效率要求.此时需要精心设计错误注入测试用例,注入一些最大可能触发被测试系统失效的错误.但目前要设计不同平台、不同系统及不同环境下的错误注入用例往往很困难.另外,SFIT 模拟故障延时、故障繁殖及短时延错误很困难,且不能很好地监测并观察被测试系统反应,错误更不能被注入到软件不能访问的位置.

(3) 软件 oracle 难以确定及错误注入测试用例难以自动生成

软件 oracle 难以确定,给测试结果的判断提出了挑战.即使有些错误可以根据需求规约给出 oracle,但也要测试者手动给出,很难自动产生.另外,错误注入测试用例自动产生也很困难,特别是对一些复杂系统的环境动态错误注入,首先要分析系统运行的环境找出一些可能的错误注入点及注入方式,然后再设计可行的错误注入用例进行测试,最后评估错误注入的结果.这些过程本身是很依赖环境的,很难独立地自动运行.一旦环境发生变化,原有的错误注入测试用例生成步骤也必须作相应的变更.

(4) 依赖于其他的辅助机制才能完成错误注入测试

有时单凭错误注入技术很难发现程序隐匿故障、软件错误异常及错误类型,需要结合动态监测机制或断言违背机制进行识别.此外,错误注入测试用例的生成也依赖于程序分析机制、静态扫描机制及反射机制等其他软件工程方法.

(5) SFIT 的研究大多停留于理论,缺乏商用系统

当前学术界对 SFIT 技术的研究如火如荼,各种理论与技术不断提出,研发的原型系统也不断增加.但是,许多研究(如源代码变异测试、扩展传播分析及 AVA 分析等)仅停留于理论,且 SFIT 原型系统一般都是针对特定平台、特定对象的,大多数系统中都使用其自行研发的错误注入环境,各研究机构开发的系统因缺乏相应的公共工作平台或移植困难而无法运转,原型系统的安装与配置复杂且应用范围小.这些都极大地限制了 SFIT 原型系统的共享与普及.

(6) 其他一些问题.例如,没有很好地解决错误注入后的现场恢复问题;预测 SFIT 注入的错误传播到一些指定的程序位置成为技术难点;SFIT 度量标准不够完善;资源可用性的约束,如代码、时间及其他系统资源等;在错误注入测试下的软件异常反应并不能验证软件是否符合需求说明.因此,SFIT 测试只是传统软件测试的一种补充,而不是替代,更不是银子弹.

7.2 研究展望

硬件错误注入花费高及适应范围小,软件错误注入具有较好的应用前景.近几年的研究成果显示,SFIT 的检测效率在 50%~90%.选择 SFIT 何种方法取决于被测试系统的规模及测试时间,采用动态错误注入方法具有较好的性价比,如 FERRARI,DOCTOR 等.SFIT 技术的目的是解决测试的效果及效率,提高软件质量,从其过去的研究历程与当前的研究状况来看,SFIT 的未来发展趋势主要有:

(1) 错误模拟范围将更广、更全、更准确

为了测试软件系统各种复杂的错误,模拟各种错误将是亟待解决的问题.SFIT 技术将探索模拟各种需求、各种平台、各种环境、各种类型及各种语言错误,只有这样才能测试复杂软件系统的各种错误.如模拟规格说明错误将能测试出软件设计相关错误;对于一些嵌入式系统、实时系统、并发系统及分布式系统等含有时间逻辑的软件系统,模拟时间逻辑将能够有效地检测与时间相关的错误等.

(2) 基于错误注入的安全性测试度量与评估方法

AVA是一种基于源代码的软件脆弱性评估方法,但对一些待测COTS软件在给定的错误注入用例作用下,如何根据输出域及输入域评估待测试系统的安全性将是一个很有意义的研究课题.如Δ是错误注入域,Ω是输出域,其中$\Delta=\{\Delta_1\cup\Delta_2\cup\ldots\cup\Delta_n\}$,$\Omega=\{\Omega_1\cup\Omega_2\cup\ldots\cup\Omega_n\}$,$P_i$是选择$\Delta_i$的概率,$Q_i$是系统输出$\Omega_i$脆弱性的概率,那么如何根据$\Delta_i$与$\Omega_i$来确定$Q_i$是未来研究的点.或确定$\Delta_i$,$P_i$,$\Omega_i$后,则基于SFIT的软件安全性能否通过$\sum_1^y \Delta_i P_i \Omega_i$来度量等.此外,SFIT 技术也可与马尔可夫模型相结合用于软件安全性预测与评估中.

(3) 不确定环境下的软件错误注入测试研究

很多跨平台的软件系统其部署环境无法预先说明,这必然加大错误注入测试的难度.现在系统环境复杂多样,硬件环境有 Intel CPU,AMD CPU 及单片机处理器等;软件环境有基于 Unix,Windows 及 SunOS 等系统平台;语言环境有基于 JAVA,C/C++,Fortran 等语言.研究在不同环境下的错误注入点、错误注入方式及错误注入测试用例的生成将极大地提高 SFIT 系统的应用范围及实效性.

(4) SFIT 过程应逐步实现自动化与智能化

当前,SFIT 大多数都需要人工参与,远远没有做到测试的自动化与智能化.尽管人工操作可以避免测试的机械化,降低了一些因错误注入测试用例当机的可能,有效地防止测试中断等问题.但从 SFIT 的长远发展来看,测试智能化是一种趋势,必然会逐渐减少人工参与的步骤.此外,人工参与效率低下,出错的主观因素更不可控制.SFIT 测试的自动化与智能化应沿着测试用例生成的自动化与智能化、错误注入的自动化与智能化、错误注入反馈的智能化及错误注入结果分析与评估的智能化路线进行研究.

(5) SFIT 将更注重应用与实践

一项技术或理论的提出、发展及成熟,乃至最终被人们所接受,一定要应用于实际,给人们的工作和研究带来方便.因此,SFIT 的逐步发展(包括理论研究与原型系统开发),都将体现在实际测试应用与工具系统中.目前,SFIT 相关技术及原型系统比较多,但尚未见较有实效的商业系统.SFIT 在未来发展中应更多地开发设计一些具有通用接口的错误注入构件,研究开发一些商业化产品,这将更有利于 SFIT 的发展与成熟.

(6) SFIT 的其他技术及新的应用

基于SFIT的一些新技术研究有:更准确的错误注入现场恢复技术;基于SFIT的形式化验证与测试方法,如生成动态故障树、测试树、执行树等形式化方法等.SFIT的一些新的应用场景有:Web服务可靠性测试、网格服务测试及网构软件测试,DBMS中安全性与可靠性测试,高安全关键分布式软件系统的测试,基于agent模型的容错系统测试等.此外,SFIT与软件缺陷预测技术[69]相结合也将成为一个新的研究点.

8 结 语

SFIT 代表着一种新型的测试技术,为实现高质量的软件提供技术保障.目前,SFIT 已引起了软件工程领域学者们的足够重视,研究者针对该领域中的问题从不同的角度进行了研究,相关理论、系统与原型工具也不断面世.

本文回顾了 SFIT 的研究内容,对相关技术进行了归类总结,同时还介绍了我们的基于 SFIT 的构件安全性测试研究成果.包括有源代码的静态注入测试与无源代码的接口变异测试、编译时错误注入测试、运行时动态错误注入测试、构件鲁棒性测试、错误注入测试分析技术以及各种 SFIT 原型工具的分类等.同时,在回顾 SFIT 目前发展的基础上,对 SFIT 中存在的问题进行了总结,并指出了 SFIT 的未来发展趋势.在接下来的工作中,立足于 SFIT 的研究现状,从提高测试效果、测试效率及现实应用出发,对已有的测试方法与工具进行改进或开发新的测试工具;研究具有高实用性与效率、低测试开销的 SFIT 机制与系统,为实现高质量的软件产品提供了技术保障.

References:

[1] Jean A, Martine A, Louis A, Yves C, Jean-Charles F, Jean-Claude L, Eliane M, David P. Fault injection for dependability validation: A methodology and some applications. IEEE Trans. on Software Engineering, 1990,16(2):166−182.

[2] Carreira J, Madeira H, Silva JG. Xception: A technique for the experimental evaluation of dependability in modern computers. IEEE Trans. on Software Engineering, 1998,24(2):125−136.

[3] Ghani AK, Nasser AK, Jacob AA. FERRARI: A flexible software-based fault and error injection system. IEEE Trans. on Computers, 1995,44(2):248−260.

[4] Goswami KK. DEPEND: A simulation-based environment for system level dependability analysis. IEEE Trans. on Computers, 1997,46(1):60−74.

[5] Looker N, Munro M, Xu J. Simulating errors in Web services. Int'l Journal of Simulation Systems, Science, 2004,5(5):29−37.

[6] DeMillo RA, Lipton RJ, Sayward FG. Hints on test data selection: Help for the practicing programmer. IEEE Computer, 1978,11(4):34−41.

[7] Voas J. Fault injection for the masses. IEEE Computer, 1997,30:129−130.

[8] Hsueh MC, Tsai TK, Lyer RK. Fault injection techniques and tools. IEEE Computer, 1997,30(4):75−82.

[9] Bieman JM, Dreilinger D, Lijun L. Using fault injection to increase software test coverage. In: Proc. of the 7th Int'l Symp. on Software Reliability Engineering (ISSRE'96). Washington: IEEE Computer Society, 1996. 166−174.

[10] Delamaro ME, Maidonado JC, Mathur AP. Interface mutation: An approach for integration testing. IEEE Trans. on Software Engineering, 2001,27(3):228−247.

[11] Barbosa R, Silva N, Duraes J, Madeira H. Verification and validation of (real time) COTS products using fault injection techniques. In: Lewis G, Schuster S, Smith D, eds. Proc. of the 6th Int'l IEEE Conf. on Commercial-off-the-Shelf (COTS)-Based Software Systems (ICCBSS 2007). Los Alamitos: IEEE CS, 2007. 233−242.

[12] Regina LOM, Eliane M, Naaliel Vicente M. Fault injection approach based on dependence analysis. In: Proc. of the 29th Annual Int'l Computer Software and Applications Conf. (COMPSAC 2005). Washington: IEEE Computer Society, 2005. 181−188.

[13] Avresky D, Arlat J, Laprie JC, Crouzet Y. Fault injection for formal testing of fault tolerance. IEEE Trans. on Reliability, 1996,45(3):443−455.

[14] Huang YW, Huang SK, Lin TP, Tsai CH. Web application security assessment by fault injection and behavior monitoring. In: Chen YFR, Kovacs L, Lawrence S, eds. Proc. of the 12th Int'l Conf. on World Wide Web (WWW 2003). New York: ACM, 2003. 148−159.

[15] Wang JY, Sun JC, Yang XZ. An experimental strategy of fault injection for the whole validation of fault-tolerant computer systems. Journal of Computer Research & Development, 2001,38(1):61−67 (in Chinese with English abstract).

[16] Du W, P.Mathur A. Testing for software vulnerability using environment perturbation. Quality and Reliability Engineering Int'l, 2002,18(3):261−272.

[17] Jean A, Yves C, Johan K, Peter F, Emmerich FG, nther HL. Comparison of physical and software-implemented fault injection techniques. IEEE Trans. on Computers, 2003,52(9):1115−1133.

[18] Lopez-Ongil C, Entrena L, Garcia-Valderas M, Portela MAPM, Aguirre MAAAMA, Tombs JATJ, Baena VABV, Munoz FAMF. A unified environment for fault injection at any design level based on emulation. IEEE Trans. on Nuclear Science, 2007,54(4): 946−950.

[19] Eliane M, Cec, lia MFR, Nelson GML. Jaca: A reflective fault injection tool based on patterns. In: Proc. of the 2002 Int'l Conf. on Dependable Systems and Networks (DSN2002). Los Alamitos: IEEE Computer Society, 2002. 483−487.

[20] Arlat J, Crouzet Y, Laprie J. Fault injection for dependability validation of fault-tolerant computing systems. In: Proc. of the 25th Int'l Symp. on Fault-Tolerant Computing, Highlights from 25 Years. Los Alamitos: IEEE Computer Society, 1995. 400−407.

[21] Gunneflo U, Karlsson J, Torin J. Evaluation of error detection schemes using fault injection by heavy-ion radiation. In: Proc. of the 19th Int'l Symp. on Fault-Tolerant Computing (FTCS-19). Washington: IEEE Computer Society, 1989. 340−347.

[22] Karlsson J, Arlat J, Leber G. Application of three physical fault injection techniques to the experimental assessment of the MARS architecture. In: Proc. of the 5th Ann. IEEE Int'l Working Conf. on Dependable Computing for Critical Applications. Los Alamitos: IEEE CS, 1995. 150−161.

[23] Voas JM, Miller KW, Payne JE. PISCES: A tool for predicting software testability. In: Miller KW, ed. Proc. of the 2nd Symp. on Assessment of Quality Software Development Tools. Washington: IEEE CS, 1992. 297−309.

[24] Han S, Shin KG, Rosenberg HA. DOCTOR: An integrated software fault injection environment for distributed real-time systems. In: Proc. of the 1995 IEEE Int'l Computer Performance and Dependability Symp. (IPDS'95). Los Alamitos: IEEE CS, 1995. 204−213.

[25] Liu MH, Gao YF, Shan JH, Liu JH, Zhang L, Sun JS. An approach to test data generation for killing multiple mutants. In: You-Feng G, ed. Proc. of the 22nd IEEE Int'l Conf. on Software Maintenance (ICSM 2006). Los Alamitos: IEEE CS, 2006. 113−122.

[26] Andrews JH, Brand LC, Labiche Y. Is mutation an appropriate tool for testing experiments? In: Proc. of the 27th Int'l Conf. on Software Engineering (ICSE 2005). Los Alamitos: IEEE Computer Society Press, 2005. 402−411.

[27] Eliane M, Amanda CAR. A fault injection approach based on reflective programming. In: Proc. of the 2000 Int'l Conf. on Dependable Systems and Networks (DSN 2000). Los Alamitos: IEEE Computer Society, 2000. 407−416.

[28] Ghosh AK, Schmid M, Shah V. Testing the robustness of windows NT software. In: Proc. of the 9th Int'l Symp. on Software Reliability Engineering (ISSRE'98). Los Alamitos: IEEE CS, 1998. 231−235.

[29] Schmid M, Ghosh A, Hill F. Techniques for evaluating the robustness of Windows NT software. In: Proc. of the 2000 DARPA Information Survivability Conf. and Exposition (DISCEX 2000). Los Alamitos: IEEE CS, 2000. 347−360.

[30] Thompson HH, Whittaker JA, Mottay FE. Software security vulnerability testing in hostile environments. In: Haddad H, Papadopoulos G, eds. Proc. of the 2002 ACM Symp. on Applied computing. New York: ACM, 2002. 260−264.

[31] Barton PM, Louis F, Bryan S. An empirical study of the reliability of UNIX utilities. Communications of the ACM, 1990,33(12):32−44.

[32] Miller BP, Koski D, Lee CP, Maganty V, Murthy R, Natarajan A, Steidl J. Fuzz Revisited: A Re-Examination of the Reliability of UNIX Utilities and Services, Vol.1, Wisconsin: Computer Sciences Department, University of Wisconsin, 2000. 1−23.

[33] Jiang Y, Hou SS, Shan JH, Zhang L, Xie B. Contract-based mutation for testing components. In: Proc. of the 21st Int'l Conf. on Software Maintenance (ICSM 2005). Los Alamitos: IEEE Computer Society, 2005. 483−492.

[34] Jian-jun Y, Wei-dong C, Cheng-qing Y, Yun-he P. Interface mutation for component. Journal of Zhejiang University (Engineering Science), 2003,37(2):129−133 (in Chinese with English abstract).

[35] Voas JM. PIE: A dynamic failure-based technique. IEEE Trans. on Software Engineering, 1992,18(8):717−727.

[36] Voas J, Ghosh A, McGraw G, Charron FACF, Miller KAMK. Defining an adaptive software security metric from a dynamic software failure tolerance measure. In: Ghosh A, ed. Proc. of the 11th Annual Conf. on Computer Assurance, Systems Integrity. Software Safety. Process Security (COMPASS'96). Piscataway: IEEE, 1996. 250−263.

[37] Ghosh AK, Schmid M, Hill F. Wrapping windows NT software for robustness. In: Pomeranz I, Sanders WH, eds. Proc. of the 29th Annual Int'l Symp. on Fault-Tolerant Computing. Los Alamitos: IEEE Computer Society, 1999. 344−347.

[38] Koopman P, Sung J, Dingman C, Siewiorek D, Marz T. Comparing operating systems using robustness benchmarks. In: Proc. of the 16th Symp. on Reliable Distributed Systems. Los Alamitos: IEEE Computer Society, 1997. 72−79.

[39] Ai-Guo L, Bing-Rong H, Si W. An approach for identifying software vulnerabilities based on error propagation analysis. Chinese Journal of Computers, 2007,30(11):1910−1921 (in Chinese with English abstract).

[40] Arkin B, Stender S, McGraw G. Software Penetration Testing. IEEE Security & Privacy, 2005,532−535.

[41] Howden WE. Weak mutation testing and completeness of test sets. IEEE Trans. on Software Engineering, 1982,8(4):371−379.

[42] Offutt AJ, Jie P. Detecting equivalent mutants and the feasible path problem. In: Ghosh A, ed. Proc. of the 11th Annual Conf. on Computer Assurance 'Systems Integrity. Software Safety. Process Security'(COMPASS 1996). Piscataway: IEEE, 1996. 224−236.

[43] Offutt AJ, Lee S D. An empirical evaluation of weak mutation. IEEE Trans. on Software Engineering, 1994,20(5):337−344.

[44] Offutt J, Lee A, Rothermel G, Untch R, Zapf C. An experimental determination of sufficient mutation operators. ACM Trans. on Software Engineering and Methodology, 1996,5(2):99−118.

[45] DeMillo RA, Guindi DS, McCracken WM, Offutt AJ, King KN. An extended overview of the Mothra software testing environment. In: Proc. of the 2nd Workshop on Software Testing, Verification, and Analysis. Los Alamitos: IEEE Computer Society, 1988. 142−151.

[46] DeMilli RA, Offutt AJ. Constraint-Based automatic test data generation. IEEE Trans. on Software Engineering, 1991,17(9): 900−910.

[47] Offutt AJ, Zhenyi J, Jie P. The dynamic domain reduction procedure for test data generation. Software Practice and Experience, 1999,29(2):167−193.

[48] Ghosh S, Kelly JL. Bytecode fault injection for Java software. The Journal of Systems and Software, 2008,47(2):1−10.

[49] Whittaker JA, Mottay FE, EI-Far LK. Testing exception and error cases using runtime fault injection. Florida Computer Science Technology Laboratory, 2001. 1−9.

[50] Tsai TK, Iyer RK, Jewitt D. An approach towards benchmarking of fault-tolerant commercial systems. In: Proc. of the Annual Symp. on Fault Tolerant Computing. Los Alamitos: IEEE Computer Society, 1996. 314−323.

[51] Barton JH, Czeck EW, Segall ZZ, Siewiorek DP. Fault injection experiments using FIAT. IEEE Trans. on Computers, 1990,39(4): 575−582.

[52] Kropp NP. Automatic robustness testing of off-the-shelf software components. Pennsylvania: Institute for Complex Engineered Systems, Carnegie Mellon University, 1998. 1−34.

[53] Ghosh AK, Schmid M. An approach to testing COTS software for robustness to operating system exceptions and errors. In: Schmid M, ed. Proc. of the 10th Int'l Symp. on Software Reliability Engineering (ISSRE'99). Los Alamitos: IEEE Computer Society, 1999. 166−174.

[54] Chen J, Lu Y, Xie X, Zhang W. Testing approach of component security based on dynamic monitoring. In: J Ni, ed. Proc. of the 2nd Int'l Multi-Symp. on Computer and Computational Sciences (IMSCCS 2007). IEEE Computer Society, 2007. 381−386.

[55] Chen J, Lu Y, Xie X. Testing approach of component security based on fault injection. In: Niu XM, Xu XF, eds. Proc. of the 2007 Int'l Conf. on Computational Intelligence and Security (CIS 2007): IEEE Computer Society, 2007. 763−767.

[56] Martin S, Christof F. Robustness and Security Hardening of COTS Software Libraries. In: Anderson T, Kaâniche M, eds. Proc. of the 37th Annual IEEE/IFIP Int'l Conf. on Dependable Systems and Networks (DSN 2007). Los Alamitos: IEEE Computer Society, 2007. 61−71.

[57] Chen J, Lu Y, Xie X, You L, Wen X. Design and implementation of an automatic testing platform for component security. Computer Science, 2008,35(12):229−233 (in Chinese with English abstract).

[58] Martin H, Arshad J, Neeraj S. EPIC: Profiling the propagation and effect of data errors in software. IEEE Trans. on Computers, 2004,53(5):512−530.

[59] Voas J, Charron F, McGraw G, Miller K, Friedman M. Predicting how badly "Good" software can behave. IEEE Software, 1997 14(4):73−83.

[60] Dawson S, Jahanian F, Mitton T. ORCHESTRA: A probing and fault injection environment for testing protocol implementations. In: Proc. of the 2nd Int'l Computer Performance and Dependability Symp. (IPDS 1996). Los Alamitos: IEEE Computer Society, 1996. 56−56.

[61] Looker N, Munro M, Xu J. A comparison of network level fault injection with code insertion. In: Proc. of the 29th IEEE Int'l Computer Software and Applications Conf. Washington: IEEE Computer Society, 2005. 479−484.

[62] Duraes JA, Madeira HS. Emulation of Software Faults: A field data study and a practical approach. IEEE Trans. on Software Engineering, 2006,32(11):849−867.

[63] Aidemark J, Vinter J, Folkesson P, Karlsson JAKJ. GOOFI: Generic object-oriented fault injection tool. In: Vinter J, ed. Proc. of the Int'l Conf. on Dependable Systems and Networks. DSN 2001. Los Alamitos: IEEE Computer Society, 2001. 83−88.

[64] Wei-lun K, Ravishankar K I, Dong T. FINE: A fault injection and monitoring environment for tracing the UNIX system behavior under faults. IEEE Trans. on Software Engineering, 1993,19(11):1105−1118.

[65] Antonio D, Jos FM, Lourdes L, Ana BG, Luis R. Exhaustif: A fault injection tool for distributed heterogeneous embedded systems. In: Berqia A, ed. Proc. of the 2007 Euro American Conf. on Telematics and information systems. New York: ACM, 2007. 1−8.

[66] Baldini A, Benso A, Chiusano S, Prinetto P. “BOND”: An interposition agents based fault injectorfor Windows NT. In: Proc. of the 2000 IEEE Int’l Symp. on Defect and Fault Tolerance in VLSI Systems. Los Alamitos: IEEE CS, 2000. 387−395.

[67] Voas J, McGraw G. Software fault Injection: Inoculating Programs Against Errors. New York: John Wiley and Sons, 1997. 191−197.

[68] Acantilado NJP, Acantilado CP. SIMPLE: A prototype software fault-injection tool [MS. Thesis]. Monterey: Naval Postgraduate School, 2002.

[69] Qing W, Shu-Jian W, Ming-Shu L. Research on software defect prediction. Journal of Software, 2008,19(7):1565−1580 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/19/1565.htm

附中文参考文献:

[15] 王建莹,孙峻朝,扬孝宗.一种用于容错计算机系统整体验证的故障注入试验策略.计算机研究与发展,2001,38(1):61−67.

[34] 杨建军,陈卫东,叶澄清,潘云鹤.面向组件的接口变异测试方法.浙江大学学报(工学版),2003,37(2):129−133.

[39] 李爱国,洪炳镕,王司.基于错误传播分析的软件脆弱点识别方法研究.计算机学报,2007,30(11):1910−1921.

[57] 陈锦富,卢炎生,谢晓东,游亮,温贤馨.一个组件安全自动化测试平台的设计与实现.计算机科学,2008,35(12):229−233.

[69] 王青,伍书剑,李明树.软件缺陷预测技术研究.软件学报,2008,19(7):1565−1580. http://www.jos.org.cn/1000-9825/19/1565.htm

陈锦富(1978－),男,江西信丰人,博士生,CCF 学生会员,主要研究领域为软件测试,数据库系统.

谢晓东(1976－),男,博士,副教授,主要研究领域为软件测试,数据库系统.

卢炎生(1949－),男,教授,博士生导师,主要研究领域为软件工程,数据库系统,数据挖掘.