基于变异的软件错误定位方法研究综述

迟 洋，苏小红，王甜甜

（哈尔滨工业大学 计算机科学与技术学院，哈尔滨 150001）

摘 要：软件调试横跨整个软件开发周期，而错误定位是软件调试中最困难、最耗时的任务之一。针对软件自动化调试的需求和应用背景，本文介绍了变异分析以及错误定位相关的国内外研究现状，选择了具有较高的错误定位精度的基于变异分析的软件错误定位方法进行研究，并对已有的基于变异分析的软件错误定位方法做出了分析和比较。最后，对未来的研究方向进行了展望。

关键词：软件调试；变异分析；错误定位

中图分类号：TP311 文献标志码：A 文章编号：2095-2163(2017)05-0157-05

A survey on mutation-based fault localization

CHI Yang，SU Xiaohong，WANG Tiantian

(*School of Computer Science and Technology*，*Harbin Institute of Technology*，*Harbin* 150001，*China*)

Abstract: Software debugging is across the entire software development cycle，and fault-localization is one of the most difficult and time-consuming task in software debugging. For automatic software debugging requirements and application background，this paper analyses the research status of mutation analysis and fault location in China and abroad；after that，chooses to study the mutaiton-based fault localization as it has higher fault location accuracy. Then the paper makes the detailed analyses and comparisons on previous researches of mutaiton-based fault localization. At last，the paper looks into the research priorities in future.

Keywords: software debugging；mutation analysis；fault localization

0 引 言

错误定位是指有效识别导致程序测试执行失败的缺陷或问题，以定位软件中的错误代码为目的，可通过在待测程序上运行合适的测试用例，分析并定位错误语句。这一过程通常需要人为理解待测程序的复杂的内部逻辑，以及辨识通过和失败的测试之间的不同原因，因此错误定位技术是软件调试中最具挑战性与开拓性，且对操作人员要求达到高等技能水平的任务之一。由于人为进行错误定位开销巨大[1]，自动化、高效可靠、且精准可达的错误定位方法研究工作则具有重大的理论意义和实用价值。

然而，虽然软件测试是减少软件错误、提高软件质量的一种重要手段，但是因为程序中的某些语句或分支仅在一些极端情况或满足特定条件时才能执行，同时某些影响程序的环境因素又往往是不可预见的，所以完备的测试几乎是不可能的，也是不现实的。随着软件规模越来越大、逻辑越来越复杂，测试有时很难覆盖程序中所有可能的执行分支，以致几乎不可能在程序测试过程中发现所有的问题。另外，虽然目前已有很多自动化软件测试工具，软件调试却大多采用设置断点等人工分析的方法，人工定位错误不仅极其耗时，而要准确定位软件中的错误则更是需要对其现存各种困难的长足突破。因此，如何有效检测并快速准确地定位软件错误，即已成为目前亟需解决的一个重要的科学问题。

迄今为止，已有很多不同的软件错误定位方法，例如基于语句覆盖的方法、基于程序状态的方法和基于程序依赖关系的方法等等。其中，基于语句覆盖的错误定位方法[1]（Coverage-based Fault Location，CBFL）得到了广泛的研究，该方法通过在待测程序上运行一个测试套件，程序谱，也就是研究项目实体（程序语句[2-4]或程序分支[5]）的覆盖情况，以及记录测试用例执行结果（成功，或失败）。这种错误定位技术根据覆盖情况可为每个实体计算可疑度作为其可能是一个错误的衡定依据。实体可疑度从高到低排序形成一个列表从而提供给开发人员用于定位错误实体。基于程序谱（语句覆盖）的错误定位方法研究主要集中在设计新的可疑度计算公式[6-9]，进行公式最优化的理论分析和公式的层次结构的分析[10-12]等方面。然而，这里所指的程序谱，仅仅是待测程序控制流和测试用例执行结果的结合，由于高度依赖测试用例在程序语句上的覆盖信息，对其准确性也存在着一定质疑[13]。随着错误定位研究的不断发展，提高错误定位的精度吸引了广泛的关注。因此，如何在已有的错误定位方法的基础上，寻找一种新的精确而稳定的错误定位方法辅助开发人员寻找软件错误并对错误进行修复，不仅仅是一个值得研究的问题，同时更对于提高软件质量、降低软件维护的成本、缩减人为开销、实现软件的自动化调试

基金项目：国家自然科学基金（61173021）。

作者简介：迟 洋（1991-），女，硕士研究生，主要研究方向：软件工程、软件错误定位；苏小红（1966-），女，博士，教授，博士生导师，主要研究方向：程序分析、克隆管理、软件错误定位等；王甜甜（1980-），女，博士，副教授，主要研究方向：程序分析、计算机辅助教学。

收稿日期：2016-06-27

与维护，具有至关重要的科学意义和使用价值。

考虑到测试和错误定位，超过二十年的变异测试领域的研究和实验表明，通过在程序上使用变异算子模拟人为错误，相比于传统的测试选择标准（如基于代码覆盖）可以有效检测未知的、真实的错误，这种方法因此而被称为基于变异分析的错误定位方法[14]（Mutation-based Fault Location，MBFL）。该方法通过变异体模拟人为错误，并通过其执行结果利用公式揭示出来以达到错误定位的目的，能够更为准确地定位出程序的错误语句。这一方法也为错误定位领域提供了一个良好的开端方向，对错误定位领域的发展必将发挥预期理想的推动作用。

本文通过对已有文献成果的研究和分析，总结了变异分析以及错误定位相关的国内外研究现状，并对已有的基于变异分析的软件错误定位方法做出了分析和比较。最后，对未来的研究方向进行了展望。

1 软件错误定位方法的研究

由于自动化错误定位技术具有重要的科学意义和研究价值，人们在错误定位领域开展了大量的研究。主要的方法包括，基于程序状态修改的方法[15-16]、基于程序依赖分析的方法[17]、基于程序切片的方法[18]以及基于语句覆盖的方法[1]等。在此，针对各类方法，将逐一给出实现原理概述如下。

首先，基于程序状态修改的方法就是通过修改状态识别程序错误，定位效果较好，但由于其复杂度较高，对于较大规模的程序定位效率并不理想，且对程序状态进行修改，容易导致无法预估的后果，如程序崩溃或陷入死循环等，错误定位的效果并不稳定，因而该方法在近几年涌现的研究成果颇为少见。

其次，基于程序依赖分析的方法是使用程序实体间的依赖关系，给出可疑语句的集合并形成可供程序员理解的调试上下文。这类方法考虑了程序实体间的相互影响和依赖关系，在理论上更为全面、也更深入，定位效果堪称良好。但该方法计算复杂度结果居高，且开销巨大，同时调试信息通常又会过于繁琐，包含冗余，增加了定位的难度。

接着，基于程序切片的方法通过将程序划分简化成程序切片，将定位的范围缩小到切片级别，该方法定位较为准确，所需测试用例较少，但无法提供程序语句可疑程度的描述，且在大量的程序切片中观察程序行为的开销较大，时间和空间复杂度均会较高。

最后，基于语句覆盖的方法主要是分析程序执行时每条语句的覆盖信息，根据成功和失败测试用例对程序语句的覆盖情况利用公式计算可疑度，如 Jones 和 Harrold[2,19] 提出的 Tarantula 方法、Abreu[20] 等提出的 Ochiai 方法以及 Jaccard[21] 方法等。Santelices 等[22] 通过实验验证了 Ochiai 方法的有效性。Tarantula 方法认为错误语句应该被失败测试用例大量的执行，而被成功测试用例少量的执行。具体地，利用公式(1)来计算每条语句的可疑值，可疑值越大，相应的语句越可能包含错误。公式(1)的数学表述如下：

$$suspiciousness(s) = \frac{\frac{failed(s)}{totalfailed}}{\frac{failed(s)}{totalfailed} + \frac{passed(s)}{totalpassed}} \quad (1)$$

其中，$failed(s)$ 和 $passed(s)$ 分别表示语句 s 被多少个失败测试用例和成功测试用例采纳与执行。

Ochiai 方法与 Tarantula 方法类似，就是通过分析测试执行时程序语句的覆盖信息定位错误语句，利用 Ochiai 公式计算语句的可疑度，计算公式如下：

$$Suspiciousness(e) = \frac{failed(e)}{\sqrt{totfailed * (failed(e) + passed(e))}} \quad (2)$$

运行结果在 0-1 之间，越靠近 1，可疑度越高。

Jaccard[21] 方法仍然通过分析程序执行时的程序覆盖信息定位程序中的错误语句，该方法利用 Jaccard 系数来计算语句的可疑度，数学表述如下：

$$Jaccard(s) = \frac{failed(s)}{failed(s) + passed(s) + failed(\bar{s})} \quad (3)$$

其中，$failed(\bar{s})$ 指的是未执行语句 s 的失败测试用例个数，即 $totalfailed$ 与 $failed(s)$ 的差值。

此外，Yoo 等[23] 提出使用信息论帮助解决‘错误定位优先次序’问题。展示了 CBFL 公式的风险评估可以使用遗传编程自动演化。提出了一些演化的公式，并证明这些公式与已知最好的 CBFL 度量公式 Op2 等价。

基于语句覆盖的错误定位方法依赖待测程序控制流和测试用例的覆盖信息，易于实现且容易理解，往往能够得到理想的结果。但该方法未考虑到程序控制流对程序的影响，对于谓词类型的错误不能达到有效定位；同时，因其未能处理巧合正确性的测试用例，使得定位的有效性和稳定性必将会受到影响。此外，由于过分依赖测试用例在程序语句上的覆盖信息，所有在同一个基本块中的语句会共享相同的覆盖信息，相同的排名。这即会使得错误定位的精确度发生降低，程序员需要检查的语句数量也随之增多，因此其定位效果并不恒稳于理想状态。

为了解决上述问题，人们提出了将变异分析与错误定位相结合的基于变异分析的错误定位方法。

2 变异分析相关研究

变异分析产生于上世纪 70 年代[24]，是指对程序源代码执行变异操作生成变异体，简单实例如图 1 所示，然后将测试用例在变异体上执行，获得执行结果信息，最后结合程序结构进行分析。程序的变异可分为一阶变异和高阶变异，分别是指对程序设计一处和多处变异产生变异体。

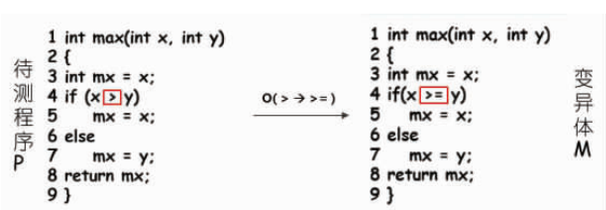

图1 程序变异举例

Fig. 1 An example of the mutation on program

变异分析最早用于变异测试，变异测试概念由 DeMillo 等[24] 在文章中首次提出，是一种通过修改程序源码模拟程序错误检验测试用例有效性的软件测试方法。该方法可以有效挖掘发现测试集的缺陷和不足，测试用例杀死变异体的数量

就将反映测试用例对程序变异的敏感程度，敏感程度越高，测试用例质量越高。变异测试的例子如图 2 所示。相关的方法还包括 Nica 等[25]的研究，在其研究中，变异体被用于产生测试用例，测试套件因此而会扩大，扩大后的测试套件即可用于执行基于模型的错误诊断。

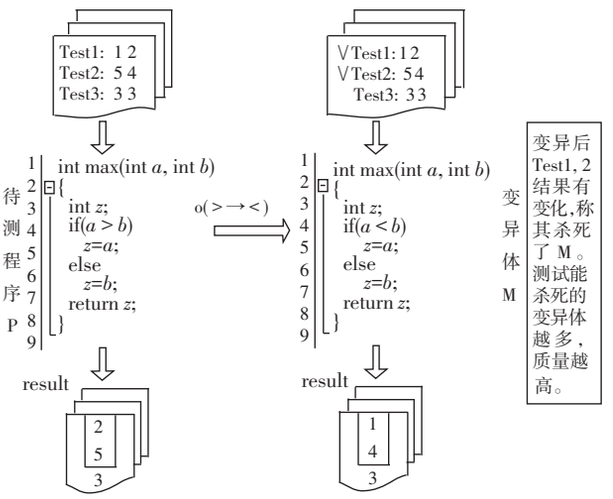

图 2　变异测试举例

Fig. 2　An example of mutation test

而后，变异分析也被用于错误修正，Debroy 和 Wong[26]提出了使用变异体来自动修复程序错误。该方法的基本思路是，利用基于语句覆盖的错误定位技术对程序错误进行定位，而后对拥有最高可疑度的错误语句引入变异，检查一个错误语句的变异体是否能够使所有测试用例都成功，即变异体是否能修复错误。假如所有测试均显现了成功，变异体就可称为是错误程序的一个可能的修复。

3　已有的基于变异分析的错误定位方法

使用变异体帮助软件错误定位过程的想法第一次在 Papadakis 和 Le-Traon[27]的错误定位研究中提出，称为基于变异分析的错误定位方法（Mutation-based Fault Location，MBFL），并开发出工具 Metallaxis-FL。该方法主要依赖变异体间的相似性来探测未知错误。其总体思路是，对待测程序进行变异处理生成变异体，执行测试用例，将执行结果与源程序测试结果构建特征比较，获得变异体可疑度，揭示与变异体在相同位置的程序语句的可疑度，判断错误语句的可能位置。

基于变异分析的错误定位方法举例如图 3 所示，图中“√”表示变异体被测试用例杀死，F-Cov 项中“X”表示被失败测试用例覆盖，“P，F”表示成功和失败测试用例，“#P，#F”分别表示杀死变异体的成功和失败测试用例数量，“M-Sup，S-Sup”表示变异体和语句的可疑度，“Rank”为语句可疑度排名。待测程序的错误为（Line7：m=x -> m=y），Mutation 列给出了各目标语句的不同变异体（一阶变异后的完整程序），这些变异体依次执行 6 个测试用例得出是否被杀死的结果，统计每个变异体被失败测试用例杀死的次数 *failed*(*e*)、被成功测试用例杀死的次数 *passed*(*e*) 和总的失败测试用例数量 *totfailed* 这 3 个数据带入 Ochiai 公式计算每个变异体的可疑度，语句的变异体中可疑度最高的值作为语句的可疑度，并排序。此时，根据 Ochiai 公式，Line7 的变异体可疑度是 0.71，排名最高，与实际错误语句位置吻合。

	Lines	F-Cov	Mutation	test1	test2	test3	test4	test5	test6	#P	#F	M-Sup	S-sup	Rank
				3,3,5	1,2,3	3,2,1	5,5,5	5,3,4	2,1,4					
int mid(int x,int y,int z){														
int m;	1	X												
m = z;	2	X	z -> ++z					√		1	0	0	0	13
			z -> --z					√		1	0	0		
if(y < z)	3	X	y -> ++y	√	√	√			√	3	1	0.5	0.5	3
			y -> --y	√	√	√			√	3	1	0.5		
			z -> ++z							0	0	0		
			z -> --z		√			√		2	0	0		
if(x < y)	4	X	x -> ++x							0	0	0	0.58	2
			x -> --x							0	0	0		
			y -> ++y	√	√				√	2	1	0.58		
			y -> --y	√	√				√	2	1	0.58		
m = y;	5													
else if(x < z)	6	X	x -> ++x							0	0	0	0	13
			x -> --x							0	0	0		
			z -> ++z							0	0	0		
			z -> --z							0	0	0		
m = y;	7	X	y -> ++y	√					√	1	1	0.71	0.71	1
			y -> --y	√					√	1	1	0.71		
else	8													13
if(x > y)	9													
m = y;	10													
else if(x > z)	11													
m = x;	12													
return m;	13	X	m -> ++m	√	√	√	√	√	√	5	1	0.41	0.41	4
			m -> --m	√	√	√	√	√	√	5	1	0.41		
}				P	P	P	P	P	F					

图 3　基于变异的错误定位举例（Line7：m=x→m=y）

Fig. 3　An example of mutation-based fault localization（Line7：m=x→m=y）

Papadakis 等[27]使用 Siemens 套件作为评测数据集，实验结果表明，Metallaxis-FL 能够精确有效地进行错误定位，且明显优于其他基于语句覆盖的错误定位工具。使用该方法实现错误定位不会导致有多条语句出现相同的排名，而且 Metallaxis-FL 拥有捕获数据流信息的能力，能区分属于同一基本块的程序语句，因此这一方法不依赖于程序结构。错误定位过程是在测试过程之后，使其能够充分利用测试过程的信息。此外，等价变异体也不会对错误定位造成影响。使用这一方法计算程序语句可疑度比以往的基于覆盖的错误定位方法更加精确，但缺点却是变异体数量过多，执行开销较大。

另一方面，类似地，Zhang 等[4]使用变异分析来鉴别程序员提交的一系列源代码库中的包含错误的代码：该方法是基于同一个位置的变异可能会导致相似的行为并产生相似的结果这一思想。这一思想为 MBFL 的发展提供了方向，具有重大意义。但程序变异会产生大量的变异体，这将使得变异体的执行开销巨大，错误定位效率低。

受程序自动修复技术的启发，Seikhyeon[28]等提出了一种基于变异分析的错误定位技术（MUtation - baSEd fault localization technique，MUSE），由于测试用例具有探测错误的能力，因此变异分析对错误定位也可提供积极作用，MUSE 通过变异体模拟不同的程序行为，并提出了揭示这些变异体之间不同的度量，根据这一度量计算语句包含错误的概率。此外，研究还设计了一种错误定位方法的评估策略 LIL，通过 LIL 分数更好地评估错误定位方法的有效性。评测时，使用的实验测试数据是 Unix 程序，包括 flex、grep、gzip、sed、space，结果表明，MUSE 的表现要胜过 Op2、Tarantula、Ochiai 等 CBFL 度量[10]。虽然 MUSE 与 MBFL 一样都是采用了变异算子来影响程序行为，但 MBFL 更多地主要是集中在通过测试套件执行变异体来展现类似的程序行为，因此 MBFL 与 MUSE 并不完全相同。由于变异算子具有显著的多样性，基于变异分析的方法如 MUSE 的缺点则是不能像 CBFL 那样自然地产生理论分析[11]。

近来，Gong P 等[29]提出使用一种动态变异执行策略来降低 MBFL 的执行开销，该方法首先变异程序语句生成有效变异体，然后在变异体上启动实现测试用例时，为了降低在大量变异体上执行测试用例的开销，采用了由变异体执行优化（Mutation Execution Optimization Strategy，MEO）和测试用例执行优化（Test Cases Execution Optimization Strategy，TEO）这 2 种优化构成的变异执行策略。MEO 的思想是基于语句所有变异体中的最大可疑度值被分配为该语句的可疑度值，因此只需要得到最高可疑度，尽量减少会得到较低可疑度的执行即可。TEO 的思想是可杀死语句 s 的一个变异体的测试用例可能也会杀死 s 的其他变异体。记录测试用例 t 杀死变异体的情况 history，优先执行能杀死更多变异体的测试用例，可以削减不必要的测试用例在变异体上的运转流程。相关的实验测试数据集为 Siemens 测试套件，结果表明，这一优化策略能有效降低开销且不损失精度。

4 研究展望

从目前已有的研究来看，虽然基于变异分析的错误定位方法定位错误的精度较高，但由于在计算语句可疑度之前，要执行大量的测试用例和变异体，因此变异分析成本是相当巨大的。为此预计在今后的一段时间内，如何有效降低这一开销是研究该错误定位方法的主要问题。降低变异分析成本可以通过以下 3 个方面得到控制与实现：

1）减少变异体数量。目前普遍使用的方法是变异体抽样，通过随机选择的方法选择变异体，这一方法简便快捷，但可能会降低错误定位精度；还可以采用选择变异，可通过启发式学习程序错误版本建立模型针对程序结构特征选择有效的变异体，或根据变异体的变异得分优选变异体；或者使用高阶变异，产生出融合多处变异的变异体以减少变异体数量。

2）去除冗余测试用例。由于 Siemens 套件等测试数据集中包含大量的测试用例，这些测试用例可能不会全部为错误定位做出贡献，因此分析针对不同的程序错误，选择不同的测试用例，去除那些不容易揭露错误的测试用例以减少测试用例的数量。

3）减少执行开销。设计算法优化执行以减少不必要的开销，或是采用弱编译的方法提高程序批量执行的速度以达到运行时间上的优化；或者使用变异体模式进行变异分析，使用程序图式编码将变异体划分成一元程序，其编译和运行速度将大大提高。

5 结束语

本文综述了近年来错误定位领域以及变异分析领域的主要研究方法，重点介绍了几个基于变异分析的错误定位方法的优缺点以及目前面临的主要问题，最后从 3 个方面展望了未来的研究重点。

参考文献：

[1] WONG W E，DEBROY V. A Survey of Software Fault Localization [R]. Dallas：University of Texas，2009.

[2] 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. Long Beach，California：ACM，2005：273-282.

[3] ABREU R，ZOETEWEIJ P，GEMUND A J C V. An evaluation of similarity coefficients for software fault localization [C] //Pacific Rim International Symposium on Dependable Computing. Melbourne，Victoria，Australia：IEEE，2007：39-46.

[4] ZHANG L，ZHANG L，KHURSHID S. Injecting mechanical faults to localize developer faults for evolving software [J]. Acm Sigplan Notices，2013，48(10)：765-784.

[5] MASRI W. Fault localization based on information flow coverage [J]. Software Testing Verification & Reliability，2010，20(2)：121-147.

[6] DALLMEIER V，LINDIG C，ZELLER A. Lightweight bug localization with AMPLE [C] // International Workshop on Automated Debugging，Aadebug 2005. Monterey，California，USA：ACM，2005：99-104.

[7] JONES J A，HARROLD M J，STASKO J T. Visualization for fault localization [C] // Proceedings of Icse Workshop on Software Visualization. Toronto，Ont.，Canada：IEEE，2001：71-75.

[8] WONG W E，QI Y，ZHAO L，et al. Effective fault localization using code coverage [C] // International Computer Software & Applications

Conference. Beijing, China: IEEE, 2007: 449-456.
[9] YOO S. Evolving human competitive spectra-based fault localisation techniques [C] // International Conference on Search Based Software Engineering. Riva Del Garda, Italy: Springer-Verlag, 2012: 244-258.
[10] NAISH L, HUA J L, RAMAMOHANARAO K. A model for spectra-based software diagnosis [J]. Acm Transactions on Software Engineering & Methodology, 2011, 20(3): 563-574.
[11] XIE X, CHEN T Y, KUO F C, et al. A theoretical analysis of the risk evaluation formulas for spectrum-based fault localization [J]. Acm Transactions on Software Engineering & Methodology, 2013, 22(4): 402-418.
[12] XIE X, KUO F C, CHEN T Y, et al. Provably Optimal and Human-Competitive Results in SBSE for Spectrum Based Fault Localisation [M] // RUHE G, ZHANG Yuanyuan. Search Based Software Engineering. Berlin/Heidelberg: Springer-Verlag, 2013: 224-238.
[13] PARNIN C, ORSO A. Are automated debugging techniques actually helping programmers? [C] // International Symposium on Software Testing and Analysis. London: ACM, 2011: 199-209.
[14] ANDREWS J H, BRIAND L C, LABICHE Y, et al. Using mutation analysis for assessing and comparing testing coverage criteria [J]. IEEE Transactions on Software Engineering, 2006, 32(8): 608-624.
[15] CLEVE H, ZELLER A. Locating causes of program failures [C] // Proceedings of the 27th international conference on Software engineering. St. Louis, MO, USA: ACM, 2005: 342-451.
[16] CHANG C H, PAIGE R. From regular expressions to DFA's using compressed NFA's [M] //APOSTOLICO A, CROCHEMORE M, GALIL Z, et al. Combinational Pattern Matching. Berlin/Heidelberg: Springer-Verlag, 1992: 90-110.
[17] BAAH G K, PODGURSKI A, HARROLD M J. The probabilistic program dependence graph and its application to fault diagnosis [J]. Software Engineering, IEEE Transactions on, 2010, 36(4): 528-545.
[18] STERLING C D, OLSSON R A. Automated bug isolation via program chipping [C] // International Symposium on Automated Analysis-Driven Debugging. California: ACM Press, 2005: 1061-1086.
[19] 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. Orlando: IEEE Computer Society, 2002: 467-477.
[20] ABREU R, ZOETEWEIJ P, GEMUND A J C V. On the accuracy of spectrum-based fault localization [C] // Testing: Academic and Industrial Conference Practice and Research Techniques-Mutation 2007. Windsor: TAICPART-Mutation, 2007: 89-98.
[21] REPS T, BALL T, DAS M, et al. The use of program profiling for software maintenance with applications to the year 2000 problem [M] // JAZAYERI M, SCHAUER H. Software Engineering-ESEC/FSE'97. Berlin/Heidelberg: Springer, 1997: 432-449.
[22] SANTELICES R, JONES J, YU Y, et al. Lightweight fault-localization using multiple coverage types [C] // International Conference on Software Engineering. Auckland: IEEE/ACM, 2009: 56-66.
[23] YOO S, HARMAN M, CLARK D. Fault localization prioritization: Comparing information-theoretic and coverage-based approaches [J]. Acm Transactions on Software Engineering & Methodology, 2013, 22(3): 96.
[24] DEMILLO R A, LIPTON R J, SAYWARD F G. Hints on test data selection: help for the practicing programmer [J]. Computer, 1978, 11(4): 34-41.
[25] NICA M, NICA S, WOTAWA F. On the use of mutations and testing for debugging [J]. Software Practice & Experience, 2013, 43(9): 1121-1142.
[26] DEBROY V, WONG W E. Using mutation to automatically suggest fixes for faulty programs [C] // International Conference on Software Testing, Verification and Validation. Paris, France: ICST, 2010: 65-74.
[27] PAPADAKIS M, TRAON Y L. Metallaxis-FL: mutation-based fault localization [J]. Software Testing Verification & Reliability, 2015, 25(5/7): 605-628.
[28] MOON S, KIM Y, KIM M, et al. Ask the mutants: mutating faulty programs for fault localization [C] // IEEE International Conference on Software Testing, Verification, and Validation. Cleveland: IEEE Computer Society, 2014: 153-162.
[29] GONG P, ZHAO R, LI Z. Faster mutation-based fault localization with a novel mutation execution strategy [C] // IEEE Eighth International Conference on Software Testing, Verification and Validation Workshops. Graz: IEEE, 2015: 1-10.
[30] MASRI W. Fault localization based on information flow coverage [J]. Software Testing Verification & Reliability, 2010, 20(2): 121-147.

(上接第 156 页)

4)便捷调整阶段。通过对大数据的安全状态检测数据安全维护过程实现的安全管理级别的评估,使其能够深入分析数据安全能力的现实需要,并针对当下数据需求,来科学研发有效改进,同时不断完善维护保障体系,使其能够在广泛背景基础上提高大数据安全维护管理能力。

4 结束语

本文首先论述了大数据所面临的安全威胁,针对存在的问题,进行了具体的探讨与分析,提出了合理解决对策,使其能够有效改变网络的安全环境。大数据环境下,涌现了很多的新兴产业与行业,并在一定程度上拓宽了营销渠道,带动了各行各业的经济效益。

综上所述,在大数据环境下,开展网络安全及维护方案研究,对当下社会的发展形势起到了一定的积极作用。在大数据环境下,只有不断完善网络环境,营造安全的网络环境,最大程度上保证人们的个人信息以及隐私、财产等不受侵害,这样才能使互联网获得更加长久稳定的良性发展。

参考文献:

[1] 王洪俭,王斌. 网络环境下的数据安全传输解决方案研究 [J]. 网络安全技术与应用, 2016(5): 35-36.
[2] 吕欣,韩晓露. 健全大数据安全保障体系研究 [J]. 信息安全研究, 2015,1(3): 211-216.
[3] 陈文芳. 网络环境下计算机信息安全与合理维护方案研究 [J]. 科技创新与应用, 2016(32): 108.
[4] 巫冬. 计算机网络安全防范在大数据时代的探讨 [J]. 科技展望, 2017(13): 17.