



北京大学

## 硕士研究生学位论文

题目: 测试文档

姓 名: 某某

学 号: 0123456789

院 系: 某某学院

专 业: 某某专业

研究方向: 某某方向

导 师: 某某教授

某年某月



## 版权声明

任何收存和保管本论文各种版本的单位和个人，未经本论文作者同意，不得将本论文转借他人，亦不得随意复制、抄录、拍照或以任何方式传播。否则一旦引起有碍作者著作权之问题，将可能承担法律责任。



## 摘要

关键词：其一，其二



## **Test Document**

Test (Some Major)

Directed by Prof. Somebody

### **ABSTRACT**

Test of the English abstract.

**KEYWORDS:** First, Second





# 目录

序言	1
第一章 相关工作	3
1.1 缺陷定位技术	3
1.1.1 自动缺陷定位技术概述	3
1.1.2 基于切片的缺陷定位	4
1.1.3 基于频谱的缺陷定位	4
1.1.4 基于状态覆盖的缺陷定位	6
1.1.5 基于变异的缺陷定位	7
1.1.6 基于构造正确执行状态的缺陷定位	8
1.1.7 基于算法式调试的缺陷定位	8
1.1.8 基于差异化调试的缺陷定位	8
第二章 研究背景	11
2.1 缺陷定位例子	11
2.2 使用基于频谱的缺陷定位	11
2.3 使用基于状态覆盖的缺陷定位	11
结论	13
参考文献	15
附录 A 附件	19
致谢	21
北京大学学位论文原创性声明和使用授权说明	23



## 序言



# 第一章 相关工作

## 1.1 缺陷定位技术

随着软件的发展，生活中越来越多的方面都与软件有着紧密的关系。小到人们的日常出行、购物、餐饮等，大到航空航天、医药等领域，软件在人们的生活中扮演着重要的角色。随着软件的应用领域的扩大，软件的复杂性上升，提升了软件缺陷的可能性。软件缺陷可能会导致巨大的损失。一个著名的被广泛引用的例子，是在海湾战争时，一颗导弹由于导航软件的精度缺陷而偏离了目标，导致28人死亡和100人受伤{TODO:cite}。美国国家标准与技术研究院(NIST)2002年发表的一篇报告({TODO:cite})显示，软件缺陷每年会导致约595亿美元的经济损失。{TODO:每年这么多国内公司漏洞事件}发现并修复软件缺陷，保障软件的高质量成为一项重要的任务。

在发现软件缺陷之后，开发人员为了解决这个缺陷往往需要三步[29]。第一步，缺陷定位，需要找到程序中和这个缺陷有关的语句。第二步，理解缺陷，明白为什么会发生缺陷。第三步，修复缺陷，修改代码以让缺陷消失。这三个步骤合起来就是调试的过程。缺陷定位作为调试的第一步，其完成速度和准确性对后面的步骤有着很大的影响。在传统的开发环境当中，人们可以手动调试来定位缺陷，比如插入断点、打印日志信息等等。在1989年Collofello等人就指出尝试去减少软件中的错误会花费50%到80%的开发和维护的精力[8]。随着软件的复杂性的上升，手动地定位软件缺陷将会耗费更多开发者的时间和精力。为了提高定位缺陷的速度，研究人员对自动化的缺陷定位展开了研究，并取得了巨大的进展{TODO:cite}。然而在2011年，Partin和Osro的一篇调查[29]通过研究缺陷定位技术在实际应用场景下的效果，发现以往的评价指标并不能准确的反映缺陷定位技术在实际应用中的效果。以往的缺陷定位技术是基于一系列关于开发人员会如何调试的假设，而这些假设在实际场景的某些情况下会失效。自动化缺陷定位技术还有很大的发展空间。

### 1.1.1 自动缺陷定位技术概述

程序切片[33, 34]是自动调试最早的技术之一，但是程序切片之后可能出错的语句数量仍然比较庞大。为了解决程序切片调试方法的短板，一种通过观察错误程序的执行特征和正确程序的执行特征的调试技术被提出。这些技术通过收集程序执行信息，观察不同的某种特征，来定位缺陷。比如使用路径概要[31]，反例[5, 12]，语句覆盖[16]和谓词值[21, 23]等等。

本文根据北京大学熊英飞研究员对缺陷定位的分类[38]，将缺陷定位分为以下几类。

- 基于切片的缺陷定位
- 基于频谱的缺陷定位
- 基于状态覆盖的缺陷定位
- 基于变异的缺陷定位
- 基于构造正确执行状态的缺陷定位
- 基于算法式调试的缺陷定位
- 基于差异化调试的缺陷定位

本文的研究内容主要根据基于频谱的缺陷定位和基于状态覆盖的缺陷定位。

### 1.1.2 基于切片的缺陷定位

Weiser在1981年提出的程序切片[33, 34]是自动调试（特别是缺陷定位）最早的技术之一。给定一个程序 $P$ 和一个在 $P$ 的语句 $s$ 中使用的变量 $v$ ，程序切片会找到 $P$ 中所有可能会影响 $s$ 中 $v$ 的值的语句。如果 $s$ 中 $v$ 的值是错误的，那么导致这个错误的错误语句一定在这个切片当中。也就是说，不在这个切片当中语句可以在调试过程中被忽略。尽管程序切片已经减少了可能出错的语句的数量，但是切片中的语句的数量仍然比较大。为了解决这个问题，Korel和Laski在1988年提出了动态程序切片[19]。动态程序切片计算某一个特定执行的切片。后来又有很多的动态程序切片的变种被提出[10, 13, 43, 44]，用于解决调试问题，并且产生了大量研究工作[3, 4, 17, 18, 24, 25, 37]。

### 1.1.3 基于频谱的缺陷定位

基于频谱的缺陷定位是使用最广泛的自动化缺陷定位方法[38]。程序频谱(Program Spectrum)最早由Reps等人于1997年提出[31]，用于解决千年虫问题。Harrold等人在2002年[14]提出使用测试覆盖信息作为频谱信息的调试方法。Renieris等人在2003年提出使用通过的测试用例和失败的测试用例进行缺陷定位[30]，奠定了此后基于频谱的缺陷定位的基础。

考虑一种极端的情况。比如当某一个语句 $s$ 被执行的之后，测试用例就会失败。而通过的测试用例都不会执行语句 $s$ 。那么语句 $s$ 很有可能就是导致缺陷的语句。找出所有这样的语句 $s$ 就可以大幅减少需要排查错误的语句。但是，在实际的代码中这种极端的情况很少出现。对于一个出错的语句 $s$ ，它很可能既被失败的测试用例执行，也被通过的测试用例执行。因为一个语句在其不同的上下文作用下会产生不同的效果。简单地计算通过的测试用例覆盖的语句和失败的测试用例覆盖的语句的差集是无法准确找

$a_{ef}$	一个语句被失败的测试用例覆盖的次数
$a_{nf}$	一个语句未被失败的测试用例覆盖的次数
$a_{ep}$	一个语句被通过的测试用例覆盖的次数
$a_{np}$	一个语句未被通过的测试用例覆盖的次数
$a_f$	失败的测试用例的个数
$a_p$	通过的测试用例执行的次数

表 1.1 基于频谱的错误定位的数学符号及其意义

出错误语句的。利用通过的测试用例覆盖的语句的交集和并集，与失败的测试用例覆盖的语句取差集，是最早的一种基于频谱的缺陷定位方法[30]。这种方法也隐含着基于频谱的缺陷定位的假设：被失败的测试用例执行的语句，更有可能有错误。而被通过的测试用例执行的语句，更有可能是正确的。

为方便此后的表述，引入一些数学符号，见表1.1。表中的统计量就是程序频谱。

Jones等人提出的Tarantula[16]，直观地展示给开发者展示了每个语句在通过的测试用例和失败的测试用例下的参与情况。每条语句的参与情况，使用公式

$$\text{Tarantula}(s) = \frac{\frac{a_{ep}}{a_p}}{\frac{a_{ep}}{a_p} + \frac{a_{ef}}{a_f}}$$

计算。这个公式计算的值也被称为怀疑度。怀疑度更高的语句会在怀疑列表更靠前的位置。相比于交集并集差集的方法，Tarantula在Siemens数据集上可以将错误的语句放在怀疑列表更前面的位置[15]。

Tarantula之后，又有很多计算怀疑度的公式被提出。Ochiai由Abreu等人提出[1]。

$$\text{Ochiai}(s) = \frac{a_{ef}}{\sqrt{a_f \times (a_{ef} + a_{ep})}}$$

Ochiai由[26]提出用于计算基因的相似度。Abreu等人将其引入用于计算怀疑度，并与Jaccard[7]，Tarantula，AMPLE[9]比较，发现Ochiai计算的怀疑度使得定位效果更好[1, 2]。

Yoo等人在理论上证明了不存在单一最佳公式[39]。

除了直接提出用于计算的公式之外，研究人员也开始使用机器学习的方法去学习怀疑度的公式。Wong等人提出使用反向传播神经网络来定位缺陷[36]。使用的输入数据是频谱信息（语句覆盖信息）和对应的测试用例是通过还是失败。输入数据每一行对应一个测试用例。第*i*列为1表示的是该测试用例覆盖了第*i*个语句，为0则表示没有覆盖。预测的标签为1表示该测试用例失败了，为0表示通过了。为了减少需要分析的

$t_f$	一个谓词被观测为真的失败的测试用例的个数
$t_p$	一个谓词被观测为真的通过的测试用例的个数
$a_f$	一个谓词被观测的失败的测试用例的个数（谓词不一定为真）
$a_p$	一个谓词被观测的通过的测试用例的个数（谓词不一定为真）
$F$	失败的测试用例的个数
$P$	通过的测试用例执行的次数

表 1.2 基于状态覆盖的错误定位的数学符号及其意义

可能出错的语句的个数（每一行输入数据的维度），优先使用所有失败的测试用例覆盖的语句。此后Wong又提出了使用径向基核函数的神经网络来定位缺陷[35]。

### 1.1.4 基于状态覆盖的缺陷定位

在缺陷定位的时候，定位的程序元素的大小也会影响结果。程序元素可以是一条语句，一个方法，一个文件。程序元素的粒度越细，对测试信息的利用越精确。然而单个元素上覆盖的测试数量越少，统计显著性越低。如果把程序的每个执行状态作为程序元素，那么这会是一个比语句更加精细的粒度。定位结果也将更加精细，对测试的利用也会更加充分。但是，几乎不会有两个测试覆盖完全相同的状态，因为一个状态所包含的上下文信息往往十分复杂，很难完全一致。于是使用抽象状态代替具体状态。使用谓词将具体状态划分为抽象状态。谓词是形如 $a > 0$ 这样的条件式。

Liblit等人最早提出了预定义谓词来划分状态[21]，并提出了统计性调试。通过预定义在哪些代码结构中插入哪些谓词，统计性调试能够收集到许多抽象状态的覆盖情况。利用表1.2中的数学符号，统计性调试的公式可以表达为

$$\text{StatisticalDebugging}(s) = \frac{2}{\frac{1}{\frac{t_f}{t_f+t_p} - \frac{a_f}{a_f+a_p}} + \frac{\log(F)}{\log(t_f)}}$$

Liu等人改进了计算公式，提出了SOBER[22]。虽然Liblit的方法可以有效定位一些错误，但是Liblit的方法只考虑了一个谓词是否在一次执行中为真，而没有考虑为真的次数。SOBER提出新的计算公式。公式计算的是对一个谓词，在失败的测试用例下这个谓词为真的概率分布，和在通过的测试用例下这个谓词为真的概率分布是否相似。如果概率分布无论是在失败的测试用例中还是通过的测试用例中都一样，那么这个谓词对应的变量等和缺陷的关系就越小。如果两个概率分布相差很大，说明这个谓词对应的抽象状态很有可能就有缺陷状态。引入这个缺陷状态的语句很可能就是出错的语句。

除了预定义谓词以外，研究人员还提出各种从程序中获取谓词的方法。Le等人提



出Savant[20], 使用程序中的不变式的变化来划分状态。程序中的不变式使用Daikon[11]挖掘。Savant使用Learning-to-rank方法, 通过分析经典的怀疑度分数和在通过的测试用例和失败的测试用例上观察到的不变式, 来定位错误的方法。Savant基于三个出发点。一, 在失败的测试用例和通过的测试用例中表现出不同的不变式的程序元素, 被怀疑是有错误的。二, 如果这些程序元素拥有很高的经典的怀疑度分数, 那么它们更有可能是错误的。三, 有一些不变式比其他不变式更加可疑, 比如 `x == null`。而Savant的工作并没有引用Liblit[21]和Liu[22], 很可能是在不知道统计性调试的情况下完成的。

### 1.1.5 基于变异的缺陷定位

变异是对程序的任意随机修改, 由变异算子得到。变异分析是测试领域的一个概念, 被用于衡量一个测试集的好坏。变异分析在程序中插入变异, 得到很多变异体, 然后使用一组测试去执行变异体。如果一个测试集中任意测试在一个变异体上得到不同的结果, 那么这个变异体被这个测试杀死。能杀死越多变异体的测试集越好。

变异被引入缺陷定位, 用于定位缺陷。Papadakis等人提出Metallaxis[28], 一个基于变异的缺陷定位。Metallaxis基于两个假设:

- 当变异和错误在一个程序的同一条语句上时, 失败的测试用例输出发生变化的概率大于通过的测试用例输出发生变化的概率。
- 当变异和错误不在同一条语句上时, 通过测试用例输出发生变化的概率大于失败的测试用例输出发生变化的概率。

基于表1.3, Metallaxis的怀疑度计算公式为

$$\text{Metallaxis}(m) = \frac{m_f}{\sqrt{F \times (m_f + m_p)}}$$

与Ochiai类似。

Moon等人提出另一个基于变异的缺陷定位技术MUSE[27]。MUSE利用变异分析去捕捉单个语句和观察到的缺陷之间的关系。MUSE基于的两个假设是:

- 一个失败的测试用例, 比起在变异了正确语句的变异体上, 在变异了错误语句的变异体上更容易变成通过的。
- 一个通过的测试用例, 比如在变异了失败语句的变异体上, 在变异了正确语句的变异体上更容易变成失败的。

基于表1.3, MUSE的怀疑度计算公式为

$$\text{MUSE}(m) = m_{f2p} - m_{p2f} \times \frac{\sum_m m_{f2p}}{\sum_m m_{p2f}}$$

$m$	变异体
$m_f$	变异 $m$ 导致输出发生变化的失败的测试用例个数
$m_p$	变异 $m$ 导致输出发生变化的通过的测试用例个数
$m_{f2p}$	变异 $m$ 导致失败的测试用例变成通过的测试用例的个数
$m_{p2f}$	变异 $m$ 导致通过的测试用例变成失败的测试用例的个数
$F$	失败的测试用例的个数

表 1.3 基于变异的缺陷定位的数学符号及其意义

### 1.1.6 基于构造正确执行状态的缺陷定位

MUSE通过变异体，可以把失败的测试用例变成通过，通过的测试用例变成失败的。假如有一个变异体，它可以把失败的测试用例变成通过的，且不会影响通过的测试用例，那么这个变异体很可能就是缺陷的补丁。但是直接分析出这样的变异体是很困难的。

Zhang提出的谓词翻转[42]巧妙地避免了直接分析出正确的补丁，而是使用改变程序状态来达到相同的目的。假如出错的是一个布尔表达式，改变程序中一个布尔表达式的取值（把真变成假，或者把假变成真），强制改变执行的分支。假如谓词翻转后，失败的测试用例变成通过的，那么对应的布尔表达式很可能有错误。

谓词翻转是局限在布尔表达式，天使调试[6]则试图解决任意表达式的错误。天使调试要求同时具有天使性和灵活性。天使性是指，存在常量 $c$ （天使值）把表达式的求值结果替换成 $c$ ，失败的测试变得通过。灵活性是指，对于所有通过的测试中的每一次表达式求值，都可以把求值结果换成一个不同的值，并且测试仍然通过。利用符号执行约束求解计算得到天使值。也由于符号执行的开销，天使调试无法应用到大型程序上。

### 1.1.7 基于算法式调试的缺陷定位

Shapiro提出的算法式调试[32]，通过对子问题询问“是”或“否”来定位缺陷。算法式调试把复杂的计算步骤拆为小的子问题。算法是调试的一个问题是，子问题的正确结果可能是不知道的。如果是让人进行交互式地判断，那么人需要花费时间计算判断子问题的结果。

### 1.1.8 基于差异化调试的缺陷定位

差异化调试由Zeller等人提出[40, 41]。不同于以往的使用动态分析或静态分析的方法去关注源代码，差异化调试关注程序状态，特别地，差异化调试关注当程序没有出错时的程序状态和程序出错时的程序状态。差异化调试尝试找到一个最小的修改集

合，当把这个集合应用到没有出错时的程序状态后，程序出错了。



## 第二章 研究背景

### 2.1 缺陷定位例子

### 2.2 使用基于频谱的缺陷定位

### 2.3 使用基于状态覆盖的缺陷定位

比如，对于如下代码C代码，

```
1 a = abs(a);  
2 if (update_b) {  
3     b = sqrt(a);  
4 }
```

当a和b的类型都为int时，如果a的值为最小的int时（ $a = -2147483648$ ），则代码会在第3行出错（b的值为NAN）。这是因为当 $a = -2147483648$ 时，第1行的a会被赋值为一个负数，于是在第3行进行sqrt操作的时候，就被出错。在第1行的时候，考虑两个抽象的状态 $a \geq 0$ 和 $a < 0$ 。发现通过的测试只有 $a \geq 0$ 这个状态，而失败的测试只有 $a < 0$ 这个状态。所以可以认为 $a < 0$ 是缺陷状态，引入这个状态的第1行的语句很可能就是缺陷语句。



## 结论





## 参考文献

- [1] R Abreu, P Zoetewij and A. J. C Van Gemund. “An Evaluation of Similarity Coefficients for Software Fault Localization”. In: *Pacific Rim International Symposium on Dependable Computing*, **2006**: 39–46.
- [2] R Abreu, P Zoetewij and A. J. C Van Gemund. “On the Accuracy of Spectrum-based Fault Localization”. In: *Testing: Academic and Industrial Conference Practice and Research Techniques - Mutation*, **2007**: 89–98.
- [3] Hiralal Agrawal, Richard A. Demillo and Eugene H. Spafford. *Debugging with dynamic slicing and backtracking*, **1993**: 589–616.
- [4] Elton Alves, Milos Gligoric, Vilas Jagannath *et al.* “Fault-localization using dynamic slicing and change impact analysis”. In: *Ieee/acm International Conference on Automated Software Engineering*, **2011**: 520–523.
- [5] Thomas Ball, Mayur Naik and Sriram K Rajamani. “From symptom to cause: localizing errors in counterexample traces”. *Acm Sigplan Notices*, **2003**, 38(1): 97–105.
- [6] Satish Chandra, Emina Torlak, Shaon Barman *et al.* “Angelic debugging”. In: *International Conference on Software Engineering*, **2011**: 121–130.
- [7] Mike Y. Chen, Emre Kiciman, Eugene Fratkin *et al.* “Pinpoint: Problem Determination in Large, Dynamic Internet Services”. In: *Dependable Systems and Networks, 2002. DSN 2002. Proceedings. International Conference on*, **2002**: 595–604.
- [8] James S. Collofello and Scott N. Woodfield. *Evaluating the effectiveness of reliability-assurance techniques*. Elsevier Science Inc., **1989**: 191–195.
- [9] Valentin Dallmeier, Christian Lindig and Andreas Zeller. *Lightweight Defect Localization for Java*. Springer Berlin Heidelberg, **2005**: 528–550.
- [10] Richard A. Demillo, Hsin Pan and Eugene H. Spafford. “Critical slicing for software fault localization”. In: **1996**: 121–134.
- [11] Michael D. Ernst, Jeff H. Perkins, Philip J. Guo *et al.* “The Daikon system for dynamic detection of likely invariants”. *Science of Computer Programming*, **2007**, 69(1-3): 35–45.
- [12] Alex Groce, Daniel Kroening and Flavio Lerda. “Understanding Counterexamples with explain”. In *Computer-Aided Verification*, **2004**, 3114: 453–456.
- [13] Gyim, Tibor Thy, Besz *et al.* “An efficient relevant slicing method for debugging”. *Acm Sigsoft Software Engineering Notes*, **1999**, 24(6): 303–321.
- [14] Mary Jean Harrold, Gregg Roethermel, Kent Sayre *et al.* “An empirical investigation of the relationship between spectra differences and regression faults”. *Software Testing Verification & Reliability*, **2000**, 10(3): 171–194.

- [15] James A. Jones and Mary Jean Harrold. “*Empirical evaluation of the tarantula automatic fault-localization technique*”. In: *Ieee/acm International Conference on Automated Software Engineering*, **2005**: 273–282.
- [16] Jones, A James, Harrold *et al.* “*Visualization of test information to assist fault localization*”. *Bio-chemical Engineering Journal*, **2002**, 24(2): 115–123.
- [17] Xiaolin Ju, Shujuan Jiang, Xiang Chen *et al.* “*HSFal: Effective fault localization using hybrid spectrum of full slices and execution slices*”. *Journal of Systems & Software*, **2014**, 90(1): 3–17.
- [18] Z. A. Al-Khanjari, M. R. Woodward, Haider Ali Ramadhan *et al.* “*The Efficiency of Critical Slicing in Fault Localization*”. *Software Quality Journal*, **2005**, 13(2): 129–153.
- [19] Bogdan Korel and Janusz Laski. “*Dynamic program slicing ☆*”. *Information Processing Letters*, **1988**, 29(3): 155–163.
- [20] Tien Duy B. Le, David Lo, Claire Le Goues *et al.* “*A learning-to-rank based fault localization approach using likely invariants*”. In: *International Symposium on Software Testing and Analysis*, **2016**: 177–188.
- [21] Ben Liblit, Mayur Naik, Alice X. Zheng *et al.* “*Scalable statistical bug isolation*”. In: **2005**: 15–26.
- [22] Chao Liu, Long Fei, Xifeng Yan *et al.* “*Statistical Debugging: A Hypothesis Testing-Based Approach*”. *IEEE Transactions on Software Engineering*, **2006**, 32(10): 831–848.
- [23] Chao Liu, Xifeng Yan, Long Fei *et al.* “*SOBER: statistical model-based bug localization*”. In: *European Software Engineering Conference Held Jointly with ACM Sigsoft International Symposium on Foundations of Software Engineering*, **2005**: 286–295.
- [24] Chao Liu, Xiangyu Zhang, Jiawei Han *et al.* “*Indexing Noncrashing Failures: A Dynamic Program Slicing-Based Approach*”. In: *IEEE International Conference on Software Maintenance*, **2007**: 455–464.
- [25] Xiaoguang Mao, Yan Lei, Ziyang Dai *et al.* “*Slice-based statistical fault localization ☆*”. *Journal of Systems & Software*, **2014**, 89(1): 51–62.
- [26] Meyer, Andréia Da Silvagarcia, Antonio Augusto Francosouza *et al.* “*Comparison of similarity coefficients used for cluster analysis with dominant markers in maize (Zea mays L)*”. *Genetics & Molecular Biology*, **2004**, 27(1): 83–91.
- [27] Seokhyeon Moon, Yunho Kim, Moonzoo Kim *et al.* “*Ask the Mutants: Mutating Faulty Programs for Fault Localization*”. In: *IEEE Seventh International Conference on Software Testing, Verification and Validation*, **2014**: 153–162.
- [28] Mike Papadakis and Yves Le Traon. *Metallaxis-FL: mutation-based fault localization*. John Wiley and Sons Ltd., **2015**: 605–628.
- [29] Chris Parnin and Alessandro Orso. “*Are automated debugging techniques actually helping programmers?*” In: *International Symposium on Software Testing and Analysis*, **2011**: 199–209.
- [30] M Renieres and S. P Reiss. “*Fault localization with nearest neighbor queries*”. In: *IEEE International Conference on Automated Software Engineering*, 2003. *Proceedings*, **2003**: 30–39.

- 
- [31] Thomas Reps, Thomas Ball, Manuvir Das *et al.* “The use of program profiling for software maintenance with applications to the year 2000 problem”. *Acm Sigsoft Software Engineering Notes*, **1997**, 22(6): 432–449.
- [32] Ehud Y. Shapiro. “Algorithmic Program DeBugging”. **1982**.
- [33] Mark Weiser. “Program Slicing”. *IEEE Transactions on Software Engineering*, **1984**, SE-10(4): 352–357.
- [34] Mark Weiser. “Program slicing”. In: *International Conference on Software Engineering*, **1981**: 439–449.
- [35] W. Eric Wong, Vidroha Debroy, Richard Golden *et al.* “Effective Software Fault Localization Using an RBF Neural Network”. *IEEE Transactions on Reliability*, **2012**, 61(1): 149–169.
- [36] W. ERIC WONG and YU QI. “BP NEURAL NETWORK-BASED EFFECTIVE FAULT LOCALIZATION”. *International Journal of Software Engineering & Knowledge Engineering*, **2009**, 19(04): 573–597.
- [37] Franz Wotawa. “Fault Localization Based on Dynamic Slicing and Hitting-Set Computation.” In: *International Conference on Quality Software*, **2010**: 161–170.
- [38] Yingfei Xiong. *Fault Localization*, **2018**. [http://sei.pku.edu.cn/~xiongyf04/SA/2017/18\\_fault\\_localization.pdf](http://sei.pku.edu.cn/~xiongyf04/SA/2017/18_fault_localization.pdf), retrieved on 2018-04-07.
- [39] Shin Yoo, Xiaoyuan Xie, Fei-Ching Kuo *et al.* “No pot of gold at the end of program spectrum rainbow: Greatest risk evaluation formula does not exist”. *RN*, **2014**, 14(14): 14.
- [40] A. Zeller and R. Hildebrandt. “Simplifying and Isolating Failure-Inducing Input”. *Software Engineering IEEE Transactions on*, **2002**, 28(2): 183–200.
- [41] Andreas Zeller. “Isolating cause-effect chains from computer programs”. In: *ACM Sigsoft Symposium on Foundations of Software Engineering*, **2002**: 1–10.
- [42] Xiangyu Zhang, Neelam Gupta and Rajiv Gupta. “Locating faults through automated predicate switching”. **2006**, 2006: 272–281.
- [43] Xiangyu Zhang, Neelam Gupta and Rajiv Gupta. “Pruning dynamic slices with confidence”. *Acm Sigplan Notices*, **2006**, 41(6): 169–180.
- [44] Xiangyu Zhang, R Gupta and Youtao Zhang. “Precise dynamic slicing algorithms”. In: *International Conference on Software Engineering, 2003. Proceedings*, **2003**: 319–329.



## 附录 A 附件



## 致谢





## 北京大学学位论文原创性声明和使用授权说明

### 原创性声明

本人郑重声明：所呈交的学位论文，是本人在导师的指导下，独立进行研究工作所取得的成果。除文中已经注明引用的内容外，本论文不含任何其他个人或集体已经发表或撰写过的作品或成果。对本文的研究做出重要贡献的个人和集体，均已在文中以明确方式标明。本声明的法律结果由本人承担。

论文作者签名：                    日期：      年      月      日

### 学位论文使用授权说明

（必须装订在提交学校图书馆的印刷本）

本人完全了解北京大学关于收集、保存、使用学位论文的规定，即：

- 按照学校要求提交学位论文的印刷本和电子版本；
- 学校有权保存学位论文的印刷本和电子版，并提供目录检索与阅览服务，在校园网上提供服务；
- 学校可以采用影印、缩印、数字化或其它复制手段保存论文；
- 因某种特殊原因需要延迟发布学位论文电子版，授权学校在□一年/□两年/□三年以后在校园网上全文发布。

（保密论文在解密后遵守此规定）

论文作者签名：                    导师签名：                    日期：      年      月      日