 DOI: https://doi.org/10.12345/bdai.v6i5.32078

Automatic generation strategy of software test cases based on static analysis

Yongbin Wen

Guoxin Kegong (Beijing) Technology Industry Development Co., Ltd., Beijing, 100000, China

Abstract

This paper addresses the inefficiency and labor dependency of traditional test case generation methods in critical quality assurance processes, while leveraging static analysis's unique advantage of extracting program structure information without code execution to support automated testing. The study proposes a static analysis-based test case generation strategy that utilizes control flow analysis and data flow analysis to extract code paths and data dependencies, thereby automatically generating test cases with path coverage and data validation. A corresponding framework has been designed and implemented. Experimental evaluations demonstrate that this approach significantly enhances test coverage and defect detection capabilities while reducing manual costs, providing a viable solution for automated testing.

Keywords

static analysis; test case generation; path coverage; data flow testing; automated testing

基于静态分析的软件测试用例自动生成策略

温永彬

国信科工（北京）科技产业发展有限公司，中国·北京 100000

摘　要

本文结合保障质量关键环节的软件测试中传统用例生成方法效率低且依赖人工的情况，以及静态分析技术无需执行代码即可提取程序结构信息为自动化测试提供支持的特点，本文旨在研究基于静态分析的测试用例自动生成策略，通过利用控制流分析、数据流分析等技术提取代码路径与数据依赖关系，进而自动生成覆盖路径和数据的测试用例，还设计并实现了相应生成框架，且经实验评估表明该策略能有效提高测试覆盖率和缺陷检测能力、降低人工成本，为自动化测试提供一种可行方案。

关键词

静态分析；测试用例生成；路径覆盖；数据流测试；自动化测试

1 引言

在软件开发过程中具有至关重要作用且通过验证软件行为确保产品可靠性的软件测试，其传统测试用例生成方法主要依赖测试人员手工设计，存在效率低下、覆盖不全面且容易遗漏深层缺陷的问题。随着软件规模扩大和复杂度提升，传统方法难以满足现代软件开发对效率和质量的高要求。作为一种无需执行程序即可对源代码进行解析和检查并能够自动识别代码中的结构特征、潜在错误及执行路径信息的静态分析技术，为测试用例的自动化生成提供了重要基础，使其能够在开发早期介入，提升测试的系统性和自动化程度。基于此，旨在利用机器可读的代码信息构建高效、高覆盖的自动化测试方案以应对日益增长的软件测试挑战的基于静态分析的测试用例生成策略研究，成为软件工程领域的一个重要方向。

【作者简介】温永彬（1987–），男，中国天津人，硕士，从事软件测试研究。

2 静态分析技术基础

2.1 静态分析概念与原理

静态分析是一种通过解析程序源代码或中间表示来推导程序行为且无需实际执行程序的技术方法，该技术建立在对程序文本的系统化处理框架之上，包含四个核心层次，词法分析将源代码字符流转换为标记序列，识别标识符、关键字、运算符和常量等基本单元。语法分析依据编程语言的上下文无关文法构建抽象语法树，刻画程序的层次化结构关系。控制流分析构造控制流图形式化表示程序执行路径，其数学模型为有向图 G=(V,E)，其中 V 表示基本块集合，$E \subseteq V \times V$ 表示控制转移边的集合。数据流分析基于格理论建立数据流方程系统，通过迭代算法计算程序中各点的数据流属性，包括可达定义、可用表达式、活跃变量等。静态分析技

术通过这些方法能够精确识别代码结构特征，检测各类缺陷包括空指针解引用、数组越界、资源泄漏、并发竞争条件等，同时为路径覆盖测试提供完整的逻辑路径枚举。分析过程采用单调框架保证收敛性，通过转移函数和控制流边传递数据流信息，最终为测试用例生成提供结构覆盖目标和数据依赖关系。

2.2 常用静态分析工具

业界主流的静态分析工具采用各具特色的技术路线实现代码缺陷检测。PMD 直接分析 Java 源代码的抽象语法树，通过 Visitors 模式遍历 AST 节点匹配缺陷模式，其规则集覆盖空代码块、未使用变量、非优化代码结构等 240 多种问题类型，支持 XPath 规则自定义扩展。FindBugs 分析 Java 字节码指令序列，采用基于缺陷模式的检测方法，其检测器识别特定指令模式如空指针解引用对应 getfield 指令前缺少 null 检查，错误同步机制对应 monitorenter/monitorexit 指令不匹配，数学计算错误对应整数溢出模式识别 [1]。SonarQube 作为代码质量平台集成多种分析引擎，其静态分析核心基于符号执行和约束求解技术，对代码构建系统间调用图并进行过程间分析，检测跨方法的数据流问题如 SQL 注入路径、跨站脚本漏洞传递链。这些工具在测试流程中应用于持续集成阶段，通过 Maven/Gradle 插件形式自动执行扫描，输出结果包含缺陷位置、严重级别、修复建议三元组信息。测试团队依据这些输出定位高风险代码模块，针对严重级别为 Blocker/Critical 的缺陷设计验证用例，同时根据复杂度指标识别圈复杂度大于 10 的方法作为重点测试对象。

2.3 静态分析在测试中的应用

静态分析为测试用例生成提供多维度结构化输入，其应用主要体现在三个方面，控制流覆盖测试、数据流测试和基于缺陷模式的针对性测试。在控制流覆盖测试中，静态分析构建的控制流图生成所有可行路径集合，测试用例生成器基于路径约束采用符号执行技术提取路径条件，通过约束求解器如 Z3 求解输入值，确保覆盖每个分支和语句。例如对于条件语句 if(x>0 && y<10)，静态分析提取路径条件 x>0 ∧ y<10 和 ¬(x>0 ∧ y<10)，生成器据此产生满足条件的输入对。在数据流测试中，静态分析构建定义 - 使用链 DU 链，其中每个链元素为三元组，测试用例旨在覆盖所有 DU 链验证数据传递正确性，特别关注危险数据流如未初始化变量使用、敏感数据泄露路径。基于缺陷模式的针对性测试直接利用静态分析输出，如检测到空指针风险时生成对应参数的 null 值测试用例，发现数组越界时生成边界值测试数据 array.length 和 array.length-1 等输入。静态分析计算的代码复杂度指标如圈复杂度 CC 直接决定基本路径测试用例数量 N=CC+1，其识别的高复杂度模块提示需要增加测试覆盖的重点区域。此外静态分析提取的代码不变量和前置条件为基于规约的测试提供断言依据，使生成的测试用例不仅覆盖结构元素更验证设计契约。

3 基于静态分析的测试用例生成方法

3.1 路径覆盖测试用例生成

路径覆盖测试用例生成的核心是基于控制流图 CFG 的系统化路径枚举与约束求解，静态分析首先构建程序的 CFG 模型 G=(V,E)，其中 V={v1,v2,...,vn} 表示基本块节点集合，每个节点对应最大连续语句序列，$E \subseteq V \times V$ 表示控制转移边的集合。通过深度优先搜索遍历 CFG 识别所有从入口节点 ventry 到出口节点 vexit 的可行路径集合 $P=\{\pi 1, \pi 2,..., \pi m\}$。对每条路径 πi，采用符号执行技术收集路径条件 $\phi \pi i$，该条件是路径上各分支条件的逻辑合取式，例如对于路径包含分支条件 x>0 和 y<10，则 $\phi \pi i=(x>0) \wedge (y<10)$。测试生成器将路径条件转化为 SMT 约束系统，调用 Z3 或 CVC5 等约束求解器计算输入变量赋值。路径覆盖率的量化评估采用公式 $Cpath=Ntotal*Ncovered \times 100\%$，其中 Ncovered 表示已覆盖路径数量，Ntotal 表示总可行路径数量。实际工具实现中，Java 符号执行工具 Symbolic PathFinder 通过字节码插桩实现符号执行，为每条路径生成包含具体输入值的 JUnit 测试用例。针对路径爆炸问题，采用动态符号执行技术如 Concolic Testing，通过启发式策略优先覆盖高风险路径。对于循环结构，设置循环展开深度 k=3 限制路径数量，同时对不可行路径采用约束不可满足性证明进行消除。

3.2 数据流测试用例生成

数据流测试用例生成基于静态分析构建的定义 - 使用链 DU 链系统生成覆盖数据依赖关系的测试输入，数据流分析通过求解数据流方程计算各程序点的定义信息，其中到达定义分析的基本方程为 $IN[B]= \cup P \in pred(B)OUT[P]$ 和 $OUT[B]=GEN[B] \cup (IN[B] \setminus KILL[B])$，其中 IN[B] 表示基本块 B 入口处的定义集合，OUT[B] 表示 B 出口处的定义集合，pred(B) 是 B 的前驱基本块集合，GEN[B] 是 B 中生成的新定义集合，KILL[B] 是 B 中杀死的定义集合。基于 DU 链生成测试用例时，针对每个定义使用对 (d,u) 其中 d 是变量 v 的定义点，u 是 v 的使用点，生成确保存在执行路径从 d 到达 u 且变量值未被重定义的测试输入。测试生成器提取从 d 到 u 的路径条件 $\phi d,u$，构造约束系统调用求解器计算输入值 [2]。特别关注危险数据流模式，对于未初始化变量使用 (def= $\varnothing$, use) 生成触发 NullPointerException 的用例。对于敏感数据泄露路径如从密码字段到网络输出的数据流，生成验证数据加密的测试用例。数据流测试覆盖率采用定义覆盖准则 $Cdef=total(def-use-pairs)covered(def-use-pairs) \times 100\%$ 评估，工具实现如 Jtest 通过字节码插桩跟踪定义使用对覆盖情况。

3.3 组合测试用例生成

组合测试用例生成基于静态分析提取的程序输入参数及其约束关系，采用组合设计技术系统生成高覆盖率的测试输入集，本文使用的是针对于单元的测试用例。静态分析首先识别方法的参数类型、取值范围和依赖约束，通过数据流

分析追踪参数传播路径确定影响分支条件的参数子集。通过值集分析确定参数取值区间如 int 参数 x ∈ [0,100]。通过约束分析提取参数间约束关系如参数A>0时参数B不能为负。对包含 n 个参数的系统，采用配对测试 Pairwise Testing 生成覆盖所有参数值两两组合的测试用例集，其数学基础是覆盖数组CA(N; t, k, v)，其中 N 表示测试用例数，t=2 表示强度，k 表示参数个数，v 表示参数值个数。生成算法采用贪心算法如 IPO（In-Parameter-Order）算法，首先纵向扩展生成覆盖前两个参数所有值对的测试集，然后水平扩展逐步添加新参数，确保新增用例覆盖最多未覆盖参数对。对于参数取值存在约束的情况，将约束嵌入到组合测试生成过程，通过约束求解确保生成用例的可行性。组合测试显著减少测试用例数量同时保持较高缺陷检测能力，特别适合配置参数多的系统测试。测试生成工具如ACTS实现多种组合强度生成算法，输出符合 JUnit 格式的测试用例，支持约束条件描述语言指定参数间依赖关系。

4 自动生成策略的设计与评估

4.1 策略设计框架

基于静态分析的测试用例自动生成策略采用分层架构设计，整个框架由四个核心模块组成，输入处理模块、静态分析模块、测试生成模块和输出管理模块。输入处理模块接收源代码文件，支持 Java、C++、Python 等多种编程语言，通过语言特定的解析器将源代码转换为抽象语法树中间表示。静态分析模块包含控制流分析器、数据流分析器和缺陷模式检测器三个组件，控制流分析器构建 CFG 并识别所有可行路径 [3]。数据流分析器计算定义 - 使用链和程序依赖图。缺陷模式检测器基于预定义规则集识别空指针、数组越界等风险点。测试生成模块包含路径约束提取器、约束求解器和测试用例合成器，路径约束提取器从 CFG 路径中提取条件表达式。约束求解器调用 Z3 求解器计算满足条件的输入值。测试用例合成器将输入值组合为完整测试用例。

4.2 策略实现关键技术

策略实现采用 Java 语言开发，基于 Eclipse JDT 构建抽象语法树，使用 Soot 框架进行控制流和数据流分析。关键技术包括混合符号执行算法、测试用例约简算法和静态分析工具集成技术。混合符号执行算法结合具体执行和符号执行，首先通过随机测试生成初始测试集，然后对未覆盖路径进行符号执行，使用动态符号执行工具 JPF-Constraint 求解路径条件，有效处理复杂约束和外部调用 [4]。测试用例约简算法采用基于贪心算法的测试集优化，计算每个测试用例的独特路径覆盖数量，优先保留覆盖最多独特路径的用例，移除冗余用例，减少测试集规模 30%-50% 同时保持覆盖率不变。静态分析工具集成技术通过插件机制整合 PMD 和 FindBugs，在测试生成前调用 PMD 进行代码质量检查，识别高复杂度方法。调用 FindBugs 检测缺陷模式，优先为高风险代码生成测试用例。策略还实现增量生成机制，通过监控代码变更，仅对修改部分重新生成测试用例，减少生成时间 60% 以上。

4.3 实验评估与结果分析

实验评估选取 OpenSource 项目 Apache Commons Math 3.6.1 作为测试对象，包含 200 个数学计算类，总计 85,000 行代码。将本文策略与随机测试生成 RAND、基于搜索的测试生成 SBST 和动态符号执行 DSE 三种方法进行对比。评估指标包括分支覆盖率、缺陷检测数量和生成时间 [5]。实验环境为 Intel Xeon E5-2680 处理器，64GB 内存，JDK 1.8 运行环境。每个工具运行 10 次取平均值，测试时间限制为 60 分钟，实验结果如下表 1 所示。

表 1 实验结果

测试生成方法	分支覆盖率 (%)	缺陷检测数 (个)	生成时间 (秒)	测试用例数 (个)
RAND	62.3	15	1245	2850
SBST	75.8	23	1860	1520
DSE	83.4	31	3270	980
本文策略	91.2	38	2340	1260

实验结果表明本文策略在分支覆盖率方面比随机测试提高 28.9%，比搜索测试提高 15.4%，比动态符号执行提高 7.8%。缺陷检测数量较其他方法多出 7-23 个，主要因为静态分析优先覆盖高风险代码区域。生成时间介于搜索测试和动态符号执行之间，但测试用例数量比动态符号执行增加 28.6%，说明策略通过测试用例约简保持了较高效率。

5 结语

本文通过整合控制流分析、数据流分析和组合测试技术，基于静态分析的测试用例自动生成策略在本文中被系统阐述，进而构建出完整的自动化测试生成框架。此策略凭借静态分析提取代码结构特征与缺陷模式以指导测试用例生成过程，显著提升测试覆盖率和缺陷检测能力。研究成果为软件测试自动化提供切实可行的解决方案，能有效降低测试成本、提高软件开发质量，尤其适用于持续集成环境中的自动化测试场景，对提升软件可靠性与安全性具备重要工程应用价值。

参考文献

[1] 吉品,冯洋,吴朵,等.面向智能软件系统的测试用例生成方法综述[J/OL].软件学报,1-40[2025-09-17].

[2] 王毅,李可萌,曹国芳,等.复杂故障下静态安全分析的部分因子化修正改进方法[J/OL].电力系统自动化,1-21[2025-09-12].

[3] 我国牵头制定的自动驾驶测试场景评价与测试用例生成ISO国际标准正式发布[J].上海质量,2025,(07):86.

[4] 邹福泰,姜开达,占天越,等.基于前后端联合分析的JavaWeb漏洞挖掘方法[J/OL].计算机研究与发展,1-14[2025-09-12].

[5] 陈斌,谢晓刚,汤雨婷.大语言模型自适应安全测评框架设计与开发[C]//《信息安全研究》杂志社.2025网络安全创新发展大会论文集.华信咨询设计研究院有限公司;,2025:143-147.