中国科学: 信息科学 2014 年 第 44 卷 第 10 期: 1290–1304
www.scichina.com info.scichina.com
《中国科学》杂志社
SCIENCE CHINA PRESS
测试技术专刊 · 论文

基于函数调用路径的软件实现与设计一致性验证

牟永敏*, 杨志嘉

北京信息科技大学计算机开放系统实验室, 北京 100101
* 通信作者. E-mail: yongminmu@163.com

收稿日期: 2014–05–16; 接受日期: 2014–09–04
国家自然科学基金 (批准号: 61370129)、北京市属高等学校创新团队建设与教师职业发展计划 (批准号: IDHT20130519) 和北京市学科与研究生教育基金 (批准号: PXM2014, 014224, 000032) 资助项目

摘要 软件系统开发完成后, 验证其是否完成了软件设计说明书的所有功能并且与设计算法一致, 是软件测试的一项重要工作. 通过人工遍历分析源代码来完成实现与设计的一致性验证是复杂费力的, 并且需要测试人员具备丰富的编程经验和较强的算法分析能力. 论文提出了一种基于函数调用路径的软件实现自动验证方法. 从设计文档和源代码两个方面出发, 分别分析其函数调用关系, 提取函数调用路径, 生成功能簇模型. 其中文档方面通过人工理解设计文档, 确定函数调用关系, 然后自动生成标准功能簇模型; 源代码方面通过静态分析, 自动获取函数调用关系, 提取功能点特征, 利用这些特征提取功能点的具体实现算法, 自动生成软件的实际功能簇模型. 对比两个功能簇模型, 验证软件实现与设计的一致性. 实验结果表明: 算法能够准确获得软件系统的功能结构及实现算法特征, 对软件实现与设计的一致性做出有效判定, 为软件实现与设计的一致性自动化测试提出一种新的思路.

关键词 软件设计 软件实现 一致性 验证 函数调用路径 功能提取 程序理解

1 引言

在过去的几十年里, 软件数量激增, 同时软件的代码量和代码复杂度不断增加, 软件测试工作的复杂程度也随之不断提高 [1]. 大量的软件产品导致软件测试工作需要高效的完成, 自动化的测试工具已经成为软件测试迫切需求. 软件测试方法按照测试范围和阶段, 可以分为单元测试、模块测试、集成测试和系统测试 [2]. 不同的开发阶段对应不同的测试方法. 软件实现与设计一致性验证是指对已完成软件系统, 验证其功能实现是否按照软件设计说明书的要求完成, 是一种非常重要的测试工作. 黑盒测试只能保证软件完成了指定的功能, 而不能验证其完成的方式 [3]; 人工代码走查和代码审查的白盒测试需要测试人员有丰富开发经验, 充分理解源代码, 系统设计结构, 以及各个代码模块实现方法, 这对测试人员有着十分高的要求并且测试效率低, 易出错, 对大型软件的分析工作是一个十分耗时且昂贵的工作.

目前基于文档的测试大多采用的是文档指导测试的方法, 如 Baharom 等 [4~6] 提出的基于 MD-Test (module documentation-based testing) 的测试方法, 该方法从形式化的设计文档中提取软件设计信息, 指导测试工作, 自动生成测试用例, 执行测试用例, 计算测试结果, 但是没有针对软件具体实现

引用格式: 牟永敏, 杨志嘉. 基于函数调用路径的软件实现与设计一致性验证. 中国科学: 信息科学, 2014, 44: 1290–1304, doi: 10.1360/N112014-00141

方法进行进一步的验证. 本文所研究的软件实现与设计的一致性验证是一种基于文档和源代码的测试工作, 我们可以把该工作描述为: 给定一个程序设计文档 D 和该文档的软件实现 S, 通过使用方法 W 验证 D 与 S 的一致性, 其中方法 W 我们采取的是为 D 和 S 建立相同结构的模型, 分别为设计模型 MD 和实现模型 MS, 通过验证 MD 与 MS 的一致性完成设计文档与软件实现一致性验证的问题. 所以软件实现与设计的一致性验证问题可以分解为建立模型和模型对比两个子问题.

本文是基于面向函数调用路径 [7] 的方法建立模型. 面向函数调用路径的软件测试方法简化了面向基本路径的软件测试问题, 使得面向基本路径的测试工作量成指数级降低, 将面向路径的测试由不可能变成了可能 [8]. 面向函数调用路径变更影响域的分析策略和方法能够确定变更影响域, 进而确定变更影响域下的测试用例, 有效约简了测试用例数量, 显著提高了回归测试效率 [9]. 本文将面向函数调用路径的思想应用到软件功能划分上, 每一个函数代表了一个功能点, 每一条函数调用路径代表了一个系统任务. 据此建立功能簇模型为软件实现与设计的验证提供基础.

其中, 本文使用算法识别相关方法提取函数功能及其实现方式, 是建立 MS 的基础. 算法识别是随着程序分析理解技术的深入研究而发展起来的. 其主要研究目标是自动理解程序算法, 识别代码片段的功能, 为函数实现方式的验证提供了基础. 在文献 [10] 中, 鲁强等将算法识别技术分为使用传统人工智能方法来识别代码固有模式规则的知识表示法; 通过信息检索方法来完成代码与注释或文档中相关概念定位的基于信息检索方法; 使用数据挖掘, 机器学习方法来识别特定功能程序特征的基于程序特征的算法识别方法. 本文使用基于函数特征属性的方法来识别函数功能, 通过提取, 对比函数的特征来完成函数功能识别.

根据所建立的模型设计模型对比的方法. 由于模型的建立是基于函数的调用关系的, 对比过程也是依据这一关系. MD 和 MS 模型建立后, 逐个验证每个函数的功能描述和实现算法的一致性, 然后验证整条函数调用路径上的函数是否均一致. 根据完全匹配的函数调用路径的条数, 给出一致性验证的结果.

论文主要贡献归结如下:

(1) 提出了一种基于函数调用路径的软件实现与设计的一致性验证方法.

(2) 设计了一种基于函数特征的函数功能提取方法, 为软件实现与设计验证提供基础.

(3) 实验表明论文提出的方法是有效的, 算法能够准确获得系统功能结构, 并能进一步的对实现算法与设计算法的一致性做出判断.

2 方法概述

在软件的开发过程中, 文档作为软件开发人员在各个阶段中前期工作成果的体现和后阶段工作的依据, 起着非常重要的作用. 在开始编写软件代码前, 首先需要编写软件的概要设计说明书和详细设计说明书. 概要设计说明书是概要设计阶段的工作成果, 说明了功能分配、模块划分、程序的总体结构、输入输出以及接口设计、运行设计、数据结构设计和出错处理设计等, 为详细设计奠定基础. 详细设计说明书描述每一模块的实现方式, 包括实现算法、逻辑流程等. 描写详细的结构化设计说明书非常有益于测试工作 [11,12]. 论文中提出的软件实现与设计一致性验证方法可以分为三个主要过程: 建立设计功能簇模型过程、从源代码抽取实际功能簇模型过程、两个功能簇进行对比过程. 设计功能簇模型和实际功能簇模型均以以函数之间的调用关系为基础建立. 本文方法的详细框架如图 1 所示.

图 1 中左侧部分为建立标准模型过程. 首先人工分析设计文档 (design specification), 得到设计函数调用关系图 (design function calling graph), 然后根据函数调用图生成设计函数调用路径 (design

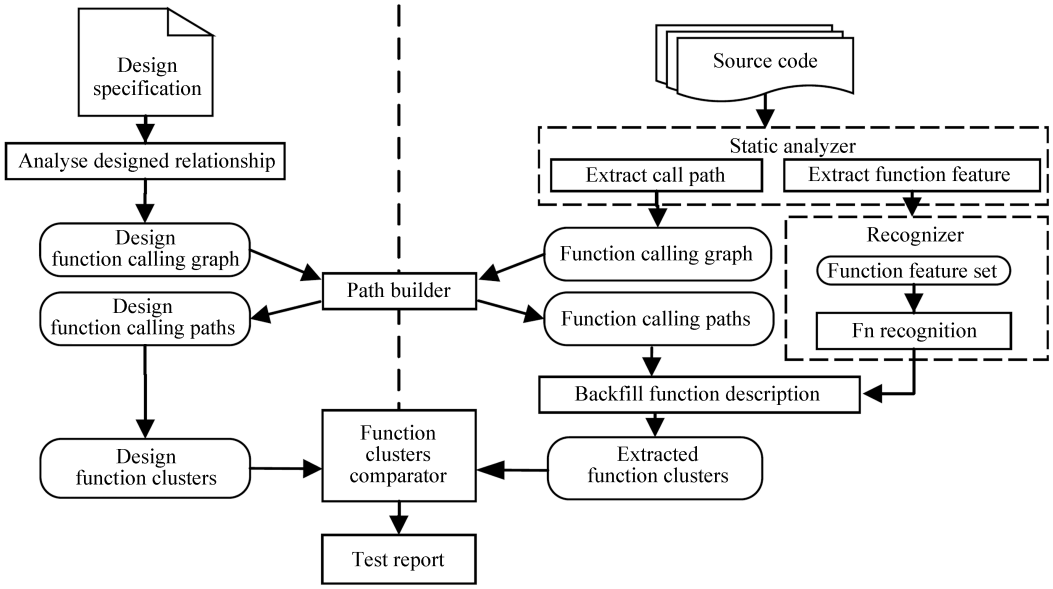


图 1　一致性验证框架

Figure 1　The framework of consistency validation

function calling paths), 生成被测系统的设计功能簇模型 (design function clusters).

图 1 中右侧部分为从源代码出发, 建立功能簇模型的过程. 首先静态分析源代码, 提取函数调用图 (extract call path) 进而生成函数调用路径 (function calling graph); 分析源代码中的每个函数, 获取函数的特征 (function feature), 然使用函数功能识别 (Fn recognition) 完成函数功能以及实现算法的提取. 将从源代码中提取函数功能等信息回填 (backfill function description) 到函数调用路径中, 按照每条路径是一个功能的基本思想, 得到系统的功能簇模型 (extracted function clusters).

最后对两个功能簇进行对比 (function clusters compare), 通过对实际功能簇模型与设计模型功能簇模型中的每一个功能点自顶向下分层对比, 验证软件系统功能实现是否达到设计说明书的要求, 确定软件实现了指定个数的功能, 并使用指定的方法实现. 完成对比后, 给出测试报告 (test report).

3　基于函数调用路径的软件功能簇模型建立与对比

3.1　函数调用路径

函数调用关系是以函数为基本单位, 通过分析源程序里基于控制流的函数之间的逻辑关系得来的. 它不同于函数包含关系, 函数包含关系仅仅分析源程序里某个函数包含了哪些函数, 而不考虑被包含函数之间的逻辑关系 [4]. 开源工具如 calltree[1)], codeviz[2)], gprof[3)] 等在绘制函数调用关系图 (call graph) 时, 其所绘制的是函数包含关系图, 并不包含函数之间的控制逻辑关系.

图 2 中举例说明了函数包含关系和函数调用关系的区别. 在图 2 的程序片段中, main 函数调用了 $f1, f2, f3$ 三个函数, 如果不考虑调用这些函数时的控制逻辑, 函数间的关系可以表示为函数包含

1) http://directory.fsf.org/wiki/Calltree.
2) http://www.csn.ul.ie/∼mel/projects/codeviz/.
3) http://www.cs.utah.edu/dept/old/texinfo/as/gprof.html.

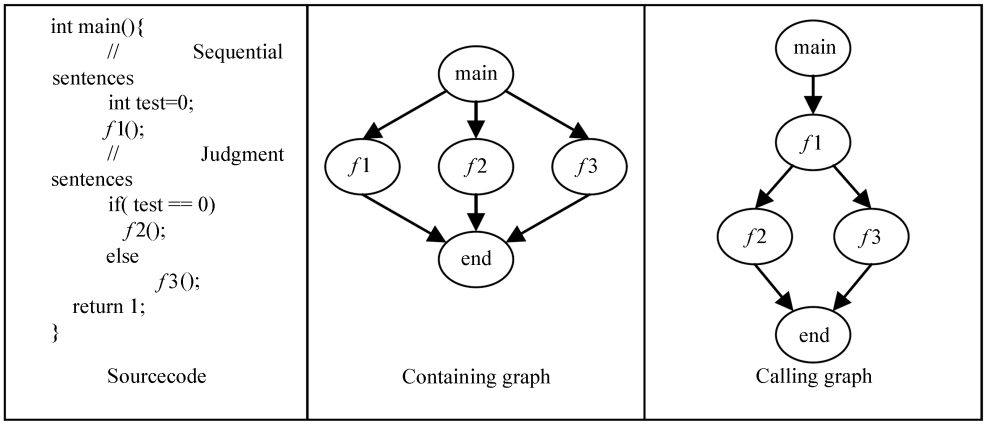


图 2 函数调用关系图

Figure 2 Function calling graph

关系图 (containing graph); 如果考虑由于 if 语句而产生的控制流信息, 使得函数 $f2, f3$ 不可能同时执行, 函数间的关系可以表示为函数调用关系图. 考虑函数包含关系和函数调用关系之间的区别是为了更准确的找到函数执行路径, 为后面的函数功能分簇提供基础. 在上例中, 函数包含关系图是不能正确反映函数执行路径, 或者会错误的认为有三条路径, 而函数调用关系图会准确的反映出函数可能的执行路径, 路径 1(main–> $f1$–> $f2$–>end), 路径 2 (main–> $f1$–> $f3$–>end). 文献 [9] 给出了函数调用关系图和函数调用路径的相关定义.

编程语言一般包括顺序、选择、循环三种形式的语句结构. 简单顺序语句是不会增加函数执行路径, 只有选择语句和循环语句才会产生不同的执行路径. 以 C 语言为例, 关键字 if, for, while, switch 将会产生多条执行语句. 所以在分析源代码建立函数调用关系图时, 需要关注这些可能产生分支的语句. 目前已经实现了软件测试工具 Regression Test For C/C++, 该工具可以静态分析程序源代码获得带有控制流的函数调用关系图和函数调用路径 [6~8].

3.2 功能簇模型

本文以函数调用关系为基础建立模型, 认为每个函数为一个功能点, 函数调用关系可以定义为系统功能关系图, 函数调用路径可以理解为系统功能路径, 其定义如定义 1 和定义 2.

定义 1 (系统功能关系图) 系统中每个函数执行了一定的功能. 函数之间的调用关系表示了系统中功能点的关系, 系统功能关系可以表示为一种有向图 $G = \langle V, E\rangle$. 其中 V 是结点集, 每一个结点称为一个功能点, 表示该结点函数的功能; $E = \{(x, y) \mid x, y \in V\}$ 是弧集, 表示功能点之间的依赖或者顺序关系.

定义 2 (系统功能路径) 以功能点为基本单位来描述系统的执行路径. 每条系统执行路径表示一个系统功能, 每条路径可以表示为系统功能关系图中的一个结点序列 $(V_i = V_{i0}, V_{i1}, \ldots, V_{im})$, V_i 为系统功能关系图中的结点.

软件系统可以抽象为一张系统功能关系图, 图中的结点为函数调用关系图中对应函数所实现的功能点. 一条函数执行路径代表系统完成一个独立的功能所需要的函数序列, 及系统功能路径. 每条系统功能路径对应了一个系统功能. 我们认为同一控制语句产生的不同分支其实现的功能类似. 因为相似的功能一般会划分在同一个功能模块下. 如 “文件” 模块下的 “打开”、“保存”、“新建”、“ 打印” 等

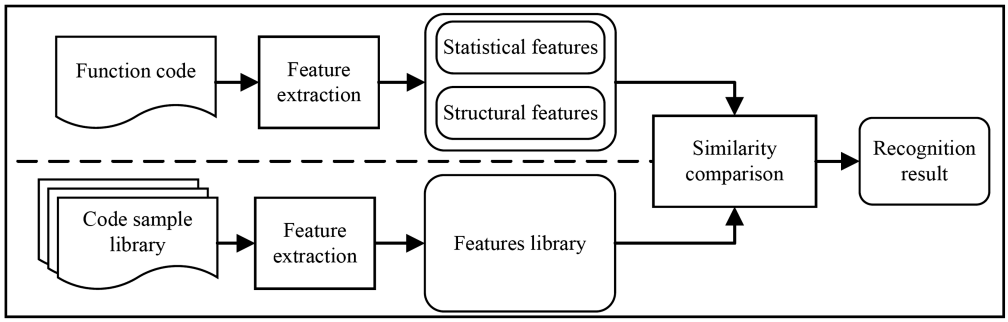


图 3　函数功能识别框架

Figure 3　Function identification framework

子功能均为对文件进行操作. 所以可以根据函数结点在函数调用图中的位置对系统功能进行分簇, 将相似的功能子模块分为一簇, 如定义 3 所述, 定义为系统功簇. 一个系统由不同层次的系统功能簇组成, 可表示为一个功能簇模型.

定义 3 (系统功能簇)　FC(function clusters) 系统功能关系图中由控制语句的分支结点 V_k 产生 2 条或者多条系统功能路径 Paths=$\{\text{path}_i(V_{i1}, V_{i2}, \ldots, V_{im})|\text{path}_i \in \text{SFP and } V_k \in \text{path}_i\}$. Paths 为分支结点 V_k 产生系统功能簇.

软件设计说明书描述了系统的整体设计和算法实现要求. 概要设计说明书说明了功能分配、模块划分、程序的总体结构等, 为详细设计奠定基础. 详细设计说明书描述每一模块的实现方式, 包括实现算法、逻辑流程等. 根据软件设计说明书可以获取函数的调用关系图, 以及每个函数的功能以及实现方式. 软件系统的设计功能簇模型是指通过人工分析设计说明书, 建立系统功能调用关系图, 然后自动生成功能调用路径, 建立功能簇模型. 另一方面, 通过静态分析源代码得到函数调用关系图以及函数调用路径, 然后通过函数功能识别得到函数的功能描述及其实现方式, 从而得到实际功能簇模型.

3.3　基于特征属性的函数功能识别

函数功能提取是软件系统实现与设计一致性验证的基础. 本文中的函数功能提取借鉴了算法识别的相关方法并且与代码相似度的一些方法比较类似. 以函数为单位, 根据函数调用图, 自底向上识别每个函数的功能. 函数的识别过程如图 3 所示, 首先从源代码中提取函数的特征, 然后与已知函数功能的模板集中的函数特征进行对比, 如果代码特征相似度达到一定阀值, 那么就认为被测函数的功能与已知函数的功能一致, 从而得到被测函数的功能. 对已知函数功能及其实现算法的函数, 抽取其函数特征, 即可得到函数特征模板集. 该模板集用于与被测函数的函数特征进行对比计算相似度.

在算法识别过程中, 有两个研究重点, 一是所提取特征的类型, 另外一个是针对该特征的相似度算法. 在算法识别 [13~15]、代码克隆 [16~18]、程序相似度 [19]、程序聚类 [20] 等各个领域都有相似的研究. 程序特征一般可以分为两类, 一种是基于统计的数字特征, 统计代码中可以量化的特征, 另一种是结构特征比如抽象语法树 (AST), 控制流图 (CFG) 等. 本文将同时提取两种特征, 并根据特征选用合适的相似度算法来计算函数的相似度.

预处理: 在提取函数特征之前, 首先需要对源代码进行预处理. 在对代码静态分析时, 按 token 作为基本处理单元进行处理分析, 源程序中的注释、空行、自定义字符串内容, 输入输出语句中的提示信息等都会对函数特征抽取结果产生影响, 所以需要删除这些干扰信息.

特征提取: Halstead[21] 提出是为了软件的度量方法, 并在程序理解中得到广泛应用. 在基于度量

表 1 数字特征

Table 1 Statistical features

Statistical features	Description
N_1	Total number of operators in an function
N_2	Total number of operands in an function
n_1	Number of unique operators in an function
n_2	Number of unique operands in an function
N	Program length $N = N_1 + N_2$
n	Program vocabulary $n = n_1 + n_2$
NoB	Number of blocks in an function
NoL	Number of loops in an function
LoC	Lines of code

值的代码特征提取中, 国内外学者首先采用 Halstead Metrics 作为基本特征. 在之后的研究中, 各国学者不断的扩充数字特征向量的维数, 将圈复杂度、代码行数等可以表征代码特征的可统计数字加入到特征向量中. 其中 Faidhi 等在文献 [22] 中使用了 24 维向量来表示程序特征. 本文中以 Halstead Metrics 为基础有选择的扩展特征向量. 比如表征代码结构的圈复杂度将不会选入特征向量, 因为本文在提取完数字特征后, 还会抽取结构特征, 这里不需要重复抽取. 另外一些如循环方向, 循环是升序还是降序将不考虑提取, 循环方向一般是可以互相转换的. 本文所采用的数字特征如表 1 所示.

数字特征的抽取我们主要借助于 flex[4)] 工具对源代码进行词法分析. flex 是一款自动化的词法分析工具, 是 20 世纪 70 年代贝尔 (Bell) 实验室开发的 lex 的开源实现. 通过 flex 我们只需要设计所需要的模式 (正则表达式与该正则表达式匹配时要执行的 C/C++ 代码) 即可完成对相应统计特征的提取与计数.

函数特征中的结构特征是指能够反映函数结构的特征. 本文将抽取表征语法特征的控制流图. 结构特征使用树或者图的形式来表示, 如图 4 所示为函数控制流图的提取过程. 使用 gcc 编译器调试信息来完成源代码的控制流图的抽取工作. gcc 是一款强大的 C 语言编译器, 通过 gcc 的 -fdump-tree-cfg-adress 选项编译源码即可得到图 4 所示的 CFG 形式的中间代码. 该中间代码主要由函数声明和函数体两部分组成. 声明部分有 gcc 内部的函数标号等信息, 函数体内是对源代码进行分块后的表示结果. 其中 ⟨bb*⟩ 模块表示基本代码块 basic block, goto 语句表示代码块之间的的执行顺序. 通过词法分析 (flex 工具) 该代码块, 分别匹配 ⟨bb*⟩, goto ⟨bb*⟩, 以及 ⟨L*⟩ (程序结束块), 即可得到源代码的控制流图. 图 4 右侧部分为通过词法分析生成 dot 语言的文件, 由 graphviz 自动画出的控制流图.

相似度计算: 每个函数的数字特征为一个 N 维向量. 特征的对比可以有多种方式, 如计算向量距离, 计算向量相似度, 使用机器学习的方法建立决策树 [13~15] 或者神经网络 [23] 等模型为被测函数分类. 计算向量距离会得到一个 0 到正无穷之间的数字, 不易把握阀值, 使用机器学习的方法将会耗费大量时间进行学习, 添加新的分类需要重新训练不易扩展, 其效果需要进一步的判断. 本文中采用计算向量相似度方法, 使用 Jaccard 系数来表示向量之间的相似度. 广泛使用的向量空间夹角余弦值注重考虑向量的方向之间的差异, 如果出现方向相似而度量值差距比较大的代码会法发生误判.

Jaccard 系数被广泛地应用到相似度计算中, 如数据库查询、网页匹配、广告分类等 [24,25]. Jaccard

4) http://flex.sourceforge.net.

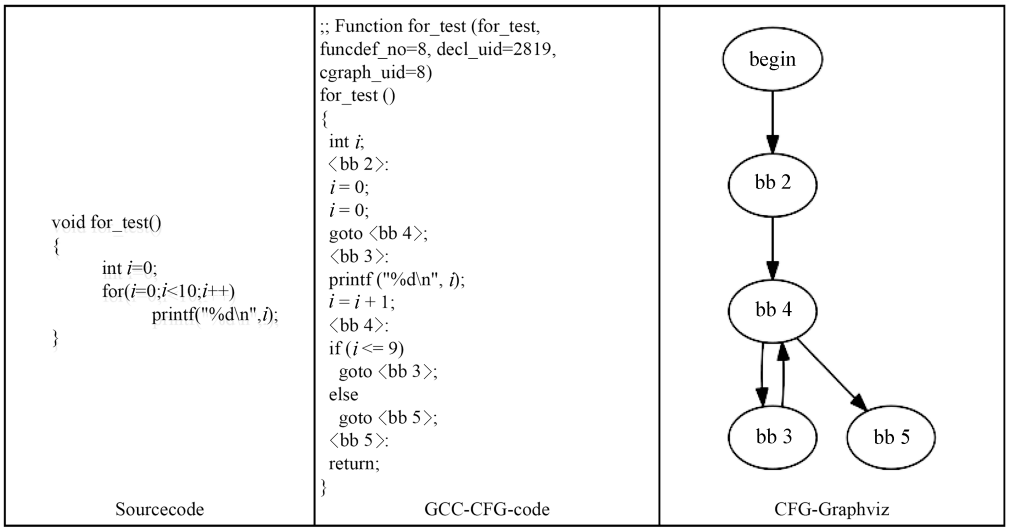


图 4　控制流图

Figure 4　Control flow graph

系数是一种基于集合的相似度函数, 其一般定义为给定两个集合 A 和 B, 这两个集合交的大小除以这两个集合并的大小. 两个集合的并等于两个集合的大小的和减去两个集合交的大小. 本文使用的是 Jaccard 系数的广义定义, 又称为 Tanimoto 系数, 其计算方法如公式 1 所示. Jaccard 系数的广义定义二元属性情况下归约为 Jaccard 系数.

$$\mathrm{NumSim}(V_p, V_q) = \frac{\sum_{i=1}^{n} x_i \cdot y_i}{\sum_{i=1}^{n} x_i^2 + \sum_{i=1}^{n} y_i^2 - \sum_{i=1}^{n} x_i \cdot y_i}. \tag{1}$$

代码结构特征采用图编辑距离来计算. 图编辑距离是指对于两张图 G_1, G_2, 经过 N 步最少次数的基本编辑操作使 G_1 转换为 G_2, 其中基本编辑是指对图中结点行进行增加、删除、编辑. 结构特征的相似度公式:

$$\mathrm{GrpSim}(G_1, G_2) = 1 - \frac{\mathrm{distance}(G_1, G_2)}{\max(|G_1|, |G_2|)}, \tag{2}$$

其中 $\mathrm{distance}(G_1, G_2)$ 是指 G_1 与 G_2 的编辑距离 $|G_1|$ 与 $|G_2|$ 为两张图的结点个数.

本文中计算图编辑距离是基于 Munkres 算法. Munkres 算法又称为 Hungarian 算法或者 Kuhn-Munkres 算法 [26], 为每个顶点分配一个标号, 将把求最大权匹配的问题转化为求完备匹配问题. Riesen 等 [27] 就是使用这种方法来计算图像之间的相似度的. 而我们需要的正是通过计算两张图的编辑距离来表示图的相似度. 另外 Munkres 算法需要给每条边赋值, 根据边的权重计算最大权匹配, 而我们将每条边的权重视为一样的, 因为控制流图中每条边所代表的含义是代码块之间的顺序执行关系, 所以边的权重均设置为 1.

功能识别过程: 数字特征相似度计算效率高, 结构特征相似度计算较复杂. 所以有序的计算数字特征和结构特征的相似度. 相似度计算过程可以概括为两步, 首先使用公式 1 计算被测函数的数字特征与模板特征集中的数字特征的相似度, 从中选取相似度高的模板作为进一步结构特征对比的模板集. 然后计算被测函数的结构特征与子模板集的结构特征的相似度, 将其中结构特征相似度最高的作为匹配结果, 最后该模板的功能及其实现方式标记到被测函数的功能描述中, 如图 5 所示, 实现如算法 1.

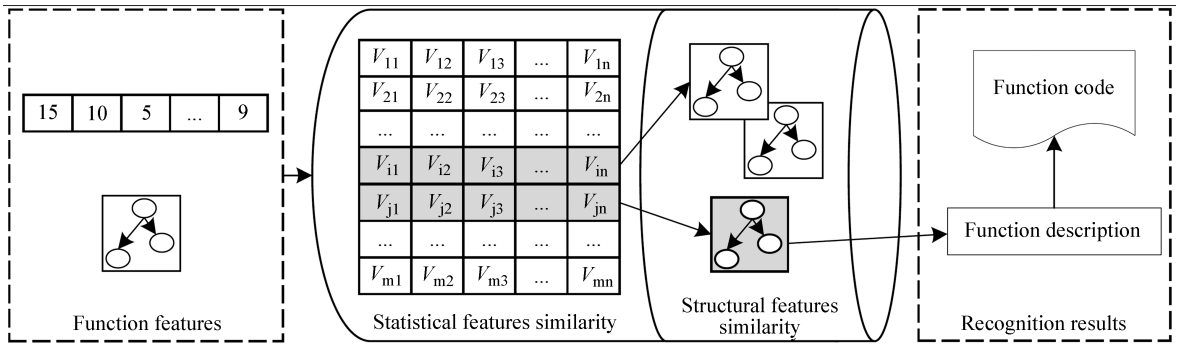


图 5 功能识别过程

Figure 5 Function recognition process

算法 1 Function recognition

Require: function statistical feature saf, function structural feature srf

Ensure: the function description

1: Read all template features set to SF;
2: **for all** Function feature sf in SF; **do**
3: Calculate the similarity of saf and sf's statistical feature;
4: **end for**
5: SF′ = several of sf which the statistical feature similarity is higher than others;
6: **for all** function feature sf in SF′; **do**
7: Calculate the similarity of srf and sf's structural feature;
8: **end for**
9: tt = sf which the structural feature similarity is highest;
10: **return** the function description of tt.

算法 1 中, 输入为函数的统计特征 saf 和函数的结构特征 srf, 输出为该函数的功能以及算法描述. 首先将模板特征集中的中的特征读入 SF. 然后依次对比 saf 与 SF 中的统计特征的相似度, 将与被测函数结构特征相似度高的模板作为子模板特征集 SF′. 之后对比 srf 与 SF′ 的结构特征相似度. 取相似度最高者作为匹配结果. 最后返回相似度最高的模板函数的功能以及实现方式.

3.4 功能簇对比

从图的角度去看两个功能簇模型对比顺序是一种自顶向下的, 从根结点开始, 以深度搜索的方式对比每一对具有相同函数名称的函数. 对比的内容包括函数功能以及实现方式. 如果函数功能描述一致, 则将这两个函数标记为真, 表示该功能点的实现与设计是一致的.

系统功能是由一系列的功能点组成的, 只有完全匹配的功能路径才能表示该系统功能是一致的. 完全匹配的功能路径是指两个模型中某两条函数调用路径中的每一个结点都是一一对应的, 并且功能描述匹配成功.

软件实现与设计如果存在不一致可能存在以下几种情况: 系统功能路径条数多于设计, 说明软件实现产生了多余的功能; 系统功能路径条数少于设计, 说明软件没有实现所有的功能; 功能路径中某些功能点描述与设计不一致, 说明某些功能点没有按照正确的方式实现. 以上是在理想的情况下, 即函数功能识别算法能够准确的提取函数功能并且功能对比正确. 然而我们的方法不能保证完全准确地识别函数功能, 在存在误判的情况下以上三种情况均会发生变化. 比如对一条功能路径判定为一致, 而

表 2 函数功能识别结果统计

Table 2 Function recognition results statistics

Function	Function name	Number	Correct number
Bubble sort	Bubblesort, bubble_sort, paixu1, ...	26	26
Insertion sort	Insertionsort, insert, insertion_sort, paixu2, ...	17	16
Quick sort	Quicksort, paixu3, ...	19	17
Heap sort	Heapsort, heapsort, paixu4, ...	17	15
Heap adjust	Head_adjust, Heapify	17	16
Merge sort	Mergesort, mergesort, paixu5, ...	18	18
Merge	Merge, merge_arrary, ...	18	18
Max num	Max, max, ...	17	17
Show nums	Output, shownums, print_array, ...	18	18
Other	Merge, heapsort, quicksort, print	4	0

该判定为误判的话, 实际软件很有可能为功能路径中某些功能点描述与设计不一致的情况. 在自动化完成软件实现与设计的一致性验证中, 很有可能导致误判, 所以在函数识别过程中, 被测函数与模板集中的特征相似度都不高时, 需要做出提示, 由人工判定.

4 实验与评测

为了证明本文方法的有效性, 能够完成对软件实现与设计的验证, 我们实现了上述方法的一个原形系统并且设计了实验展示基于函数调用路径的软件实现与设计验证方法过程.

4.1 实验配置与数据来源

主要使用与实现的工具有: 函数调用路径生成工具 —— 调用 Regression Test For C/C++ 实现; 统计特征提取器 —— 使用 flex 生成词法分析器, 对特定模式的代码进行匹配统计, 以 json 格式存储; 结构特征提取器 —— 使用 gcc/g++ 生成调试信息, 使用 flex 生成词法分析器、匹配函数名称、代码块、goto 语句、返回语句等, 设计算法生成 CFG, 以 json 格式存储; 模板集生成器 —— 读取提取的函数特征, 将其合并, 并加入原函数功能与实现方式描述, 以 json 格式存储; 函数识别器 —— 调用统计特征提取器和结构特征提取器提取被测函数特征, 然后与模板集比较计算相似度, 识别被测函数功能; 主程序 —— 调用函数识别器识别被测系统每一个函数, 统计完全匹配路径.

实验数据来自数据结构中常规算法的实现, 排序算法和查找算法作为主要的测试数据. 数据结构中的常规算法有着明确的功能说明以及实现方法要求, 并且同一问题有多种实现方法, 如内部排序可以有多种方法实现. 有利于函数功能提取以及实现算法的提取准确度验证. 实验中排序算法包括: 冒泡排序、插入排序、快速排序、堆排序、归并排序; 查找算法包括: 顺序查找、折半查找、二叉排序树、哈希查找等, 共 219696 个源代码文件. 选取 20 组代码作为为被测程序, 其余代码用于模板集生成.

4.2 函数功能识别

函数功能识别是功能簇对比的基础, 其识别率的高低会影响到功能簇对比的匹配率. 表 2 显示了其中排序算法的函数功能抽取结果.

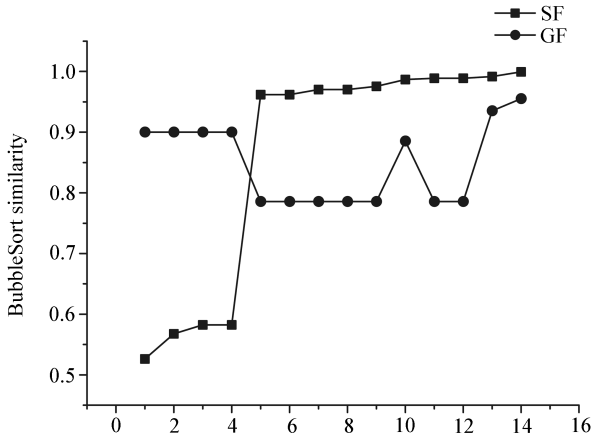


图 6 插入排序和冒泡排序相似度

Figure 6 The similarity of insertSort and bubbleSort

20 组实验数据中属于排序算法功能簇的函数共有 171 个. 从表 2 中可以看出, 167 个函数的功能是可以被识别算法提取, 4 个函数是不能识别的, 识别率达到了 97%. 通过人过分析验证, 函数识别正确的个数为 161 个, 功能提取的正确率为 96%. 经过分析无法识别的函数, 发现有函数使用了 STL 中的 qsort 函数来实现排序算法, 由于 qsort 是库函数, 在分析过程中以普通语句方式处理, 从而导致了该函数无法识别. 这也暴露出直接把库函数作为普通语句处理是不合理的, 在模板集中需要标记出库函数的功能. 另外一个函数的功能是使用 printf 函数输出数组内容, 其实现方式与模板集的所有代码的实现方式不同, 从而导致该函数没有正确识别. 所以丰富模板集是非常重要的.

虽然冒泡排序和插入排序的识别率很高, 经分析发现, 这两种算法实现的相似度是非常高的.

图 6 为同一冒泡排序与多个插入排序的实现相似度示意图. 图中横坐标为插入排序的编号, 纵坐标为相似度, 其中 SF 为统计特征相似度, GF 为结构图特征相似度. 从图中可以看出两种排序算法在统计和结构特征上均表现出比较相似. 其实两种排序算法的主要结构均为两层循环和一条判断语句组成, 由于判断语句的位置以及循环方式不同, 导致不同算法的不同实现方式在特征上表现出来的不同. 在丰富更多类似排序算法后, 可能会引起识别率的降低. 另外从统计特征看, 多种插入排序的实现方法可以明显的分为两种, 1∼4 统计特征类似, 5∼15 的统计特征类似. 对于相同实现方式, 统计特征有着轻微的变化. 从结构特征的角度看 1∼4 的结构特征是非常类似, 5∼9 以及 11 和 12 的结构特征非常类似, 并且与统计特征相比, 相同实现方法的插入排序, 结构特征一致, 基本不变. 这也说明了结构特征更加侧重表征算法实现的方法, 而统计特征能够体现出相同算法在实现的时候的细微差别. 并且, 从两种特征的变化可以看出统计特征相似度与结构特征相似度并没有表现出明显的相同的变化趋势, 说明两种特征的可能不存在一定的关联关系.

4.3 功能簇模型

选取的 20 组代码中, 每组代码均要求实现指定的排序和查找算法. 不同的算法保存在不同的文件中, 根据用户选择在主程序中调用不同的算法实现. 根据设计要求建立标准系统功能模型. 如图 7 所示.

完成被测程序的函数功能提取后, 将提取出的函数功能簇模型与设计的函数功能簇模型进行对

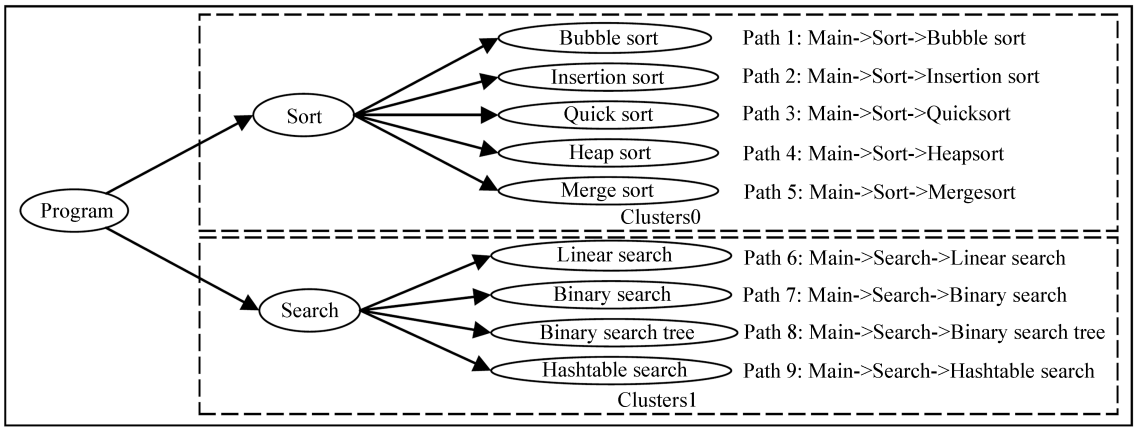


图 7　设计功能簇模型

Figure 7　Design function clusters

表 3　功能簇对比结果

Table 3　Function cluster comparison results

Code	Bubble sort	Insertion sort	Quick sort	Heap sort	Merge sort	Linear search	Binary search	Binarytree search	Hashtable search
Code1	1	1	1	1	1	1	1	1	1
Code2	1	1	1	1	1	1	1	0	1
Code3	5	0	0	0	0	1	1	0	1

比. 从表 3 中可以看到三组不同验证结果. Code1 是按照要求完成的, 每种算法均按照要求完成. 在对比 Code2 时发现没有二叉树查找算法, 该程序并没有实现该算法. 在 Code3 中发现了 5 个冒泡排序算法, 排序算法中只有一对节点是与标准模型相匹配的. 由这些数据可以看出 Code1 是按照要求完成的, Code2 实现了大部分算法, Code3 的排序算法只实现了一种, 并且拷贝在应该实现其他算法的函数中. 这说明了算法可以提取出被测程序的函数调用结构并完成对比工作.

在对比的过程中发现的第一个问题是设计功能簇模型不准确: 堆排序和合并排序一般情况下会实现一个子函数. 虽然实现程序的函数调用路径条数与设计功能簇模型相比是一致的, 但是该函数调用路径的深度不一致. 由于模板集中有类似的实现, 所以堆排序与合并排序都能正确识别, 功能簇对比在完成堆排序与合并排序功能点对比后停止运行. 设计功能簇模型不准确, 会导致验证误判.

另外多层次中间调用函数的识别是实验中的一个重点. 图 7 中 Sort 函数和 Search 都是中间调度函数, 没有实现排序算法, 只是根据需要调用相应的算法. 这样的函数有一个特点就是简单多分支. 本实验中为这样的函数设计了单独的抽象函数模板, 其特点分支多, 调用层次低. 然而这并不是多层次调用中, 上层函数功能识别通用方法. 在复杂的软件系统中, 函数的调用层次将会显著增加, 此时就需要根据特定的系统和领域丰富模板集, 为模板集合分类, 实现更多更全面的函数功能识别.

5　相关工作

论文提出的软件实现与设计一致性验证方法是一种软件实现与文档一致性的测试, 测试目的是验

证软件系统是否按照设计说明书中的要求实现. 测试方法主要采用了函数调用路径的思想对系统功能进行分析建模, 并借鉴了程序理解中的算法识别和代码形似度等相关方法来识别函数功能.

在相关的领域中 Baharom 等 [4~6] 提出了一种基于 MD-Test (module documentation-based testing) 的测试方法, 该方法从形式化设计文档中提取软件设计信息, 指导测试工作, 自动生成测试用例, 执行测试用例, 计算测试结果. 但是该方法是基于严格格式化的软件设计文档, 并不考虑软件代码实际的实现过程, 不能验证软件实现方法是否与设计一致性. Tamai 等 [28] 提出了一个集成格式化文档, 复审, 测试的框架来加强软件的可靠性并人认为说明文档中的每一个场景都需要实现代码中的若干条执行路径完成, 从而提出了一种种基于场景的分析方法来发现执行路径中的错误, 基于格式化的说明文档中自动生成测试用例检测执行路径的正确性. 但是该方法需要分析每一个场景下的每一条执行路径, 产生大量的测试用例. 另外, 在文档的自动化分析处理方面, 已经有了大量的相关研究, 如 Heitmeyer 等 [29] 提出形式化用需求模型来表示有限状态的软件系统, 自动检测需求文档中的内容, 检测类型错误 (type errors)、不确定语义 (nondeterminism)、循环定义 (circular definitions) 错误等; Ghosh 等 [30] 提出了一种介于自然语言描述需求说明书和使用精准的数学符号格式化设计需求文档的中间表达方式, 使用 ARSENAL 原形系统, 自动生成指定的格式化模型, 能够完成从自言语言描述的需求文档到形式化文档的转化; Harris 等 [31] 提出了从自然语言描述的文档中提取设计信息, 使用语法分析方法得到分析树, 从中抽取语义信息, 生成 Verilog. 这些方法都是从文档出发, 分析文档中的内容, 可以为基于文档软件自动化测试提供基础. 函数功能提取是程序理解的一个重要研究领域. Linger 等 [32,33] 使用 FX(function extraction) 技术计算了航空软件的行为, 根据函数可计算理论可以使用 CCA 的方式表示函数功能, 计算函数行为. 这种方式人为理解是相当困难的, 只适合计算机处理分析.

本文使用的基于函数调用路径的思想是在路径覆盖测试优化和回归测试中发展起来的. 我们认为每一个系统任务对应着一条函数调用路径, 一个系统看成是由有限条函数调用路径组成的, 如果对所有的函数调用路径都进行了充分测试并且无误的话, 那么这个系统是可靠的.

6 小结

随着软件维护成本的不断增加, 软件开发商迫切希望程序员是严格按照设计说明书的要求来实现的, 包括: 功能以及功能实现算法. 只有这样才能保证依据设计说明书进行的系统维护是有效的, 然而对于一个已经实现了的系统, 通过人工来验证软件实现与软件设计说明书的一致性是非常复杂, 繁琐, 而且费时费力的工作. 这需要测试人员一方面具有对设计文档充分的理解能力; 另一方面要求测试人员有深厚的程序功底和很强的算法基础和算法分析能力. 本文通过研究软件设计说明书以及软件实现之间的对应关系, 提出了一种基于函数调用路径的软件实现与设计一致性验证方法, 丰富了软件测试方法, 为设计文档的自动化测试提供了新的思路. 本方法能够帮助测试人员完成对源代码的分析, 自动提取实际系统的功能与实现方式, 提高了软件实现与设计的一致性验证效率.

本文的所提出方法存在一定的局限性, 需下一步工作继续完善. 一方面在建立软件系统的设计功能簇模型时, 需要人工分析软件设计说明书. 这限制了设计功能簇模型的建立效率, 下一步工作中, 可以对设计文档格式进行一定要求, 引入格式化文档方法, 从而实现提取自动化. 从文献 [2,4~6] 中可以看出, 格式化文档会给软件测试效率带来巨大提高. 另一方面, 本方法的实用性依赖于函数功能识别算法的准确率, 基于特征的函数功能识别仍然需要模板集. 模板集的规模影响着识别结果, 模板集过小影响识别范围, 模板集过大影响识别效率甚至识别结果. 在下一步工作中, 可以对模板集进行细化, 针对领域的做模板集, 对模板集进行分级分层管理. 甚至于找到不需要模板集的函数功能识别方法.

另外需要完善函数描述的对比方式. 假如设计要求为 “排序”, 没有明确指出排序方法, 而识别结果为“快速排序”, 快速排序是属于排序的, 但是文本对比无法得知这一事实. 所以可以采用一种语义级别的对比方式, 提高对比准确性.

参考文献

1 Kirkov R, Agre G. Source code analysis-an overview. Bulgarian Acad Sci, 2010, 10: 60–77

2 Baharom S, Shukur Z. The conceptual design of module documentation based testing tool. J Comput Sci, 2008, 4: 454–462

3 Liu S, Chen Y. A relation-based method combining functional and structural testing for test case generation. J Syst Software, 2008, 81: 234–248

4 Baharom S, Shukur Z. An experimental assessment of module documentation-based testing. Inform Software Tech, 2011, 53: 747–760

5 Baharom S, Shukur Z. Utilizing an abstraction relation document in grey-box testing approach. In: Proceedings of Electrical Engineering and Informatics, Selangor, 2009. 304–308

6 Baharom S, Shukur Z. Module documentation based testing using grey-box approach. In: Proceedings of Information Technology, Kuala Lumpur, 2008. 1–6

7 Zheng Y H, Mu Y M, Zhang Z H. Research on the static function call path generating automatically. In: Proceedings of Information Management and Engineering, Chengdu, 2010. 405–409

8 Mu Y M, Zheng Y H, Zhang Z H, et al. The algorithm of infeasible paths extraction oriented the function calling relationship. Chinese J Electron, 2012, 21: 236–240

9 Zhang Z H, Mu Y M. Research of path coverage generation techniques based function call graph. Acta Electron Sin, 2010, 138: 1808–1811 [张志华, 牟永敏. 基于函数调用的路径覆盖生成技术研究. 电子学报, 2010, 138: 1808–1811]

10 Lu Q, Li X L, Wang Z G. Survey on program algorithm recognition research. J Comput Appl, 2012, 32: 2863–2868 [鲁强, 李效恋, 王智广. 程序算法识别研究综述. 计算机应用, 2012, 32: 2863–2868]

11 Clermont M, Parnas D. Using information about functions in selecting test cases. In: Proceedings of the 1st International Workshop on Advances in Model-Based Testing, New York, 2005. 1–7

12 Feng X, Parnas D L, Tse T H. Fault propagation in tabular expression-based specifications. In: Proceedings of the 32nd Annual IEEE International Computer Software and Applications Conference, Turku, 2008. 180–183

13 Taherkhani A. Recognizing sorting algorithms with the C4.5 decision tree classifier. In: Proceedings of Program Comprehension, Braga, 2010. 72–75

14 Taherkhani A, Malmi L, Korhonen A. Using roles of variables in algorithm recognition. In: Proceedings of 9th Koli Calling International Conference on Computing Education Research, Sweden, 2010. 1–10

15 Taherkhani A, Korhonen A, Malmi L. Automatic recognition of students' sorting algorithm implementations in a data structures and algorithms course. In: Proceedings of the 12th Koli Calling International Conference on Computing Education Research, Tahko, 2012. 83–92

16 Roy C K, Cordy J R, Koschke R. Comparison and evaluation of code clone detection techniques and tools: a qualitative approach. Sci Comput Program, 2009, 74: 470–495

17 Rattan D, Bhatia R, Singh M. Software clone detection: a systematic review. Inform Software Tech, 2013, 55: 1165–1199

18 Bellon S, Koschke R, Antoniol G, et al. Comparison and evaluation of clone detection tools. Softw Eng, 2007, 33: 577–591

19 Cesare S, Xiang Y. Software Similarity and Classification. London: Springer, 2012. 1–6

20 Zhao W, Zhang L, Mei H, et al. A functional requirement based hierarchical agglomerative approach to program clustering. J Softw, 2006, 17: 1661–1668 [赵伟, 张路, 梅宏, 等. 一种基于功能需求层次凝聚的程序聚类方法. 软件学报, 2006, 17: 1661–1668]

21 Halstead M H. Elements of Software Science (Operating and Programming Systems Series). New York: Elsevier Science Inc., 1977. 3–8

22 Faidhi J A W, Robinson S K. An empirical approach for detecting program similarity and plagiarism within a university programming environment. Comput Educ, 1987, 11: 11–19

23 Zhu G, Zhu X. Autonomous mental development for algorithm recognition. In: Proceedings of Information Science and Technology (ICIST), Nanjing, 2011. 339–347
24 Lin X M, Wang W. Set and string similarity queries: a survey. Chinese J Comput, 2011, 34: 1853–1862 [林学民, 王炜. 集合和字符串的相似度查询. 计算机学报, 2011, 34: 1853–1862]
25 Bayardo R J, Ma Y, Srikant R. Scaling up all pairs similarity search. In: Proceedings of the 16th International Conference on World Wide Web, New York, 2007. 131–140
26 Dasgupta D, Hernandez G, Garrett D, et al. A comparison of multiobjective evolutionary algorithms with informed initialization and kuhn-munkres algorithm for the sailor assignment problem. In: Proceedings of the 2008 GECCO Conference Companion on Genetic and Evolutionary Computation, Atlanta, 2008. 2129–2134
27 Riesen K, Bunke H. Approximate graph edit distance computation by means of bipartite graph matching. Image Vision Comput, 2009, 27: 950–959
28 Liu S, Tamai T, Nakajima S. A framework for integrating formal specification, review, and testing to enhance software reliability. Int J Softw Eng Know, 2011, 21: 259–288
29 Heitmeyer C L, Jeffords R D, Labaw B G. Automated consistency checking of requirements specifications. ACM Trans Softw Eng Meth, 1996, 5: 231–261
30 Ghosh S, Elenius D, Li W, et al. Automatically extracting requirements specifications from natural language. arXiv preprint arXiv: 1403.3142, 2014
31 Harris I G. Extracting design information from natural language specifications. In: Proceedings of the 49th Annual Design Automation Conference, San Francisco, 2012. 1256–1257
32 Pleszkoch M G, Linger R C, Hevner A R. Introducing function extraction into software testing. ACM SIGMIS Dat, 2008, 39: 41–50
33 Linger R, Sayre K, Daly T, et al. Function extraction technology: computing the behavior of malware. In: Proceedings of System Sciences (HICSS), Hawaii, 2011. 1–9

Verify consistency of software implementation and design based on function call path

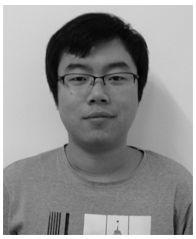
MU YongMin* & YANG ZhiJia

Open Computer System Laboratory, Computer School, Beijing Information Science & Technology University, Beijing 100101*, China*
*E-mail: yongminmu@163.com

Abstract Whether the developed software system completed all functions in the software design specification and consistent with the design algorithm is an important work in software testing. Analysis of the source code artificial and verify whether the software system is to meet the requirements of the design specification is laborious job. Testers need to have rich experience in programming and strong ability of algorithm analysis. This paper proposes a method to verify consistency of software implementation and design based on function call path. The method can be summarized as analyze the calling relationship of functions from design specifications and source code, extraction function calling path, generation function cluster model. In documents aspect, the function calling relationship can be got by manual analysis of the design specification, then the designed function cluster model can be generated automatically. We can get function calling relationship, function features by static analysis of source code and the specific implementations of function with those features. Finally, compare the two function cluster model to verify that the consistency of software implementation and design. The experimental results showed that this method can obtain the system function structure and algorithm features accurately, make effective judgment of the consistency of the design specification and the software implementation, provides a new way to verify the consistency of software implementation and design.

Keywords software design, software implementation, consistency, verification, function call path, function extraction, program comprehension

MU YongMin was born in 1961. Currently, he is a Professor of Computer Science in Beijing Information Science and Technology University. His research interests include automated software testing, automatic generation of application software and object-oriented technology. He is the director of Open Computer System Laboratory, a member of CCF, SUN Developers Alliance, senior member of Chinese Institute of Electronics.



YANG ZhiJia was born in in 1988. Currently, he is a postgraduate student of Computer Science in Beijing Information Science and Technology University. He is studying in Open Computer System Laboratory . His research interest includes automated software testing.