软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2018,29(6):1716−1738 [doi: 10.13328/j.cnki.jos.005541]

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

API 使用的关键问题研究*

李 正[1,2], 吴敬征[1], 李明树[1]

[1](中国科学院 软件研究所,北京 100190)

[2](中国科学院大学,北京 100190)

通讯作者: 李明树, E-mail: mingshu@admin.iscas.ac.cn

摘 要: API(application programming interface,应用程序编程接口)在现代软件开发过程中被广泛使用.开发人员通过调用 API 快速构建项目,节省了大量的时间.但由于 API 数量众多、文档不够完善、维护更新不及时等原因,开发人员在学习使用 API 的过程中面临着严峻的挑战.一旦 API 使用不正确,程序可能会出现缺陷甚至严重的安全问题.通过对 API 相关文献的深入调研,对近些年来国内外学者在该研究领域取得的成果进行了系统总结.首先,介绍了 API 的基本概念并分析出影响 API 使用的 3 个关键问题:API 文档质量不高、调用规约不完整以及 API 调用序列难以确定;接着,从 API 文档、调用规约和 API 推荐这 3 个主要方面对研究成果进行全面的分析;最后,对未来研究可能面临的挑战进行了展望.

关键词: API;调用规约;API 文档;API 推荐;API 使用

中图法分类号: TP311

中文引用格式: 李正,吴敬征,李明树.API 使用的关键问题研究.软件学报,2018,29(6):1716−1738. http://www.jos.org.cn/1000-9825/5541.htm

英文引用格式: Li Z, Wu JZ, Li MS. Study on key issues in API usage. Ruan Jian Xue Bao/Journal of Software, 2018,29(6): 1716−1738 (in Chinese). http://www.jos.org.cn/1000-9825/5541.htm

Study on Key Issues in API Usage

LI Zheng[1,2], WU Jing-Zheng[1], LI Ming-Shu[1]

[1](Institute of Software, The Chinese Academy of Sciences, Beijing 100190, China)

[2](University of Chinese Academy of Sciences, Beijing 100190, China)

Abstract: API (application programming interface) is widely used in modern software development process. Developers save a lot of time when they quickly build projects through invoking APIs. However, API is often difficult to use for many reasons, such as the presence of large number of interfaces, lack of perfect document, and no timely maintenance and updates. Further, API is often used incorrectly, resulting in bugs and sometimes significant security problems .This paper summarizes the recent domestic and overseas research results based on a thorough survey of the API related literatures. Firstly, it introduces the API concept and recognizes the three key issues that affect the API usage: poor API documentation, incomplete invocation specification and undetermined API call sequence. Next, it analyzes the latest advances from these three main aspects: API document, invocation specification and API recommendation. Finally, this paper outlines the challenges of the future research.

Key words: API; invocation specification; API documentation; API recommendation; API usage

API(application programming interface,应用程序编程接口)在现代软件开发过程中被广泛使用.开发人员通

* 基金项目: 国家科技重大专项(2014ZX01029101-002); 国家自然科学基金(61772507)

Foundation item: National Science and Technology Major Project (2014ZX01029101-002); National Natural Science Foundation of China (61772507)

收稿时间: 2017-09-18; 修改时间: 2017-11-09, 2017-11-29; 采用时间: 2017-12-26; jos 在线出版时间: 2018-02-08

CNKI 网络优先出版: 2018-02-08 13:25:17, http://kns.cnki.net/kcms/detail/11.2560.TP.20180208.1325.015.html

过调用 API 快速构建项目,节省了大量的时间.然而随着软件规模扩大,公开的 API 数量巨大,例如:GEF(graphical editing framework)作为一个常用的 Eclipse 的图形类库,就涉及 6 万多个 API 方法.开发人员在学习以及使用这些不熟悉的 API 时面临挑战,即使是编程专家,也需要花费很多时间学习这些 API.研究显示,开发人员学习 API 的时间占用了整个开发过程的 40%[1].

很多学者针对 API 的使用问题进行了深入分析[2−4].例如,2016 年,Nadi 等人[2]总结了开发人员在使用 JAVA 密码类 API 的时候所面临的问题:(1) 65%的开发人员认为 API 过于复杂,难以确定正确的 API 调用序列;(2) 文档不完善,开发人员需要基于例子/任务的说明文档;(3) 开发人员在加密算法选择、参数设置等方面存在困难.

同时,一旦 API 使用不正确,将会使程序出现缺陷甚至严重的安全问题[5−8].在这种形势下,近年来,关于 API 相关问题的研究大幅度增加.来自会议、期刊和网络的论文覆盖面很广,例如:API 文档研究如文献[3,9−33],API 调用规约研究如文献[34−76],API 推荐研究如文献[77−113]等.国内外还没有针对 API 使用的关键问题的系统性综述文章.

为了对该研究问题进行系统性的分析、总结和比较,通过在 IEEE,ACM,Springer 和 Elsevier 等论文数据库中进行检索,最终选择出与该研究问题直接相关的高质量论文共 138 篇(截止到 2017 年 8 月),从选择出的论文所发表的会议和期刊来看,其中将近一半的论文发表在软件工程领域的权威会议或期刊上,见表 1.

Tabel 1 Major references summary

表 1 主要参考文献汇总

刊物/会议名称	CORE2017 排名	CCF 级别	参考文献数目
Int'l Conf. on Software Engineering (ICSE)	A*	A	30
ACM SIGSOFT Symp. on the Foundation of Software Engineering/European Software Engineering Conf. (FSE/ESEC)	B	A	13
Int'l Conf. on Automated Software Engineering (ASE)	A	A	10
IEEE Trans. on Software Engineering (TSE)	A*	A	7
Int'l Symp. on Software Testing and Analysis (ISSTA)	A	B	3
Int'l Conf. on Software Maintenance and Evolution (ICSME)	A	B	2
ACM SIGPLAN Symp. on Programming Language Design & Implementation (PLDI)	A*	A	1
ACM Conf. on Object-Oriented Programming Systems, Languages,and Applications (OOPSLA)	A*	A	1

本文综述了 API 使用相关研究的历史、目前面临的主要问题以及今后的发展方向,试图为该研究方向勾画出一个较为全面和清晰的概貌,为 API 使用相关领域的研究人员提供有益参考.本文识别出影响 API 使用的 3 个关键问题:文档质量不高、调用规约不完整以及 API 调用序列难以确定,并从 API 文档、调用规约以及 API 推荐这 3 个主要方面对研究成果进行了全面的总结和分析.

1 API 基本概念

1.1 API定义

API 是软件库提供的一组可访问的接口,软件库通过 API 向外提供服务,开发人员通过使用 API 实现代码复用,提高生产效率[114].1968 年,在 NATO 软件工程会议上,McIlroy 首次提出了软件复用的概念[115],软件开发人员开始有意识地使用标准化组件设计并构建程序.1972 年,Parnas 的经典论文探索了使用模块化编程的标准[116].1992 年,Plauger 提出了关于 C 语言 15 个接口的完整定义和详细说明[117].随着面向对象技术以及互联网技术的快速发展,为软件复用提供了更全面的技术支持,API 技术快速发展,被视为解决软件危机、提高软件生产效率和质量的现实可行的途径.

API 被广泛使用有以下原因:(1) API 提供了一种代码复用机制,开发人员直接复用已有的程序,节省了大量的时间;(2) API 提供了一种信息隐藏的机制,用户不需要知道具体的实现细节,直接调用 API 完成相应的功能;(3) API 提供了访问某些资源的接口,用户只有通过设备驱动、操作系统 API 等接口才能够访问到某类资源.不同的组织或者开发人员使用了不同的术语来表示 API,并没有形成统一的标准:(1) 类库,如 C 语言的标准

库,Math 库等;(2) 框架,如.Net 框架,Eclipse 框架等;(3) 工具包,如 Google Web Toolkit,GIMP Toolkit 等;(4) API,如 Win32 APIs,Google Maps APIs 等.

1.2 API性质

(1) 可用性

可用性是衡量 API 质量的重要指标,通常包括 API 是否容易被开发人员使用、是否能够完成指定的功能等.文献[118−127]针对 API 的可用性进行了分析.例如,2016 年,Tsai 等人[118]针对 Ubuntu Linux 15.04 平台的 API 进行了系统性研究,评价了不同 API 的重要程度、复杂程度以及安全性.类似的,Qiu 等人[120]研究了 JAVA API 在 5000 多个开源的 JAVA 项目中的使用情况,发现很多 JAVA 核心 API 从未被使用等情况.

(2) 稳定性

随着软件库演化速度不断加快,API 存在失效或者消失的情况.API 的稳定性直接关系到系统的质量.例如,Android 平台平均一个月就会更新 115 个 API[128].APP 如果使用了这些失效的 API,就会影响系统的稳定性.2015 年,Linares 等人[129]调研了 APP 的质量与 API 演化之间的关系,指出使用稳定的 API 有助于提高 APP 质量.

(3) 安全性

API 的不规范使用会对系统带来安全风险.例如,Android 平台有很多未公开的 API,APP 使用这些 API 会造成隐私泄露等安全风险.2016 年,Li 等人[130]研究了 Android 平台未公开 API 的使用情况,发现大量的 APP 存在使用未公开 API 的情况,隐含着安全风险.与 Android 平台类似,IOS 系统也存在这种情况.研究[131−133]显示:IOS 攻击者使用这些未公开的 API 成功完成了攻击,同时躲避了检验过程.

1.3 影响API使用的关键问题

开发人员使用 API 完成指定任务时,通常会通过阅读文档或者搜索引擎选择合适的类库,了解如何正确地调用 API(如何设置参数、返回值的含义等),寻找针对特定应用场景 API 的调用序列或者示例程序,甚至需要了解 API 内部的设计原理.研究显示,开发人员在这一系列过程面临挑战[4].2011 年,Zibran 等人[134]研究了影响 API 使用的因素,手工分析了 5 个公开项目的 1 513 份缺陷报告,发现其中 37.4%与 API 误用有关,影响因素涉及 API 文档、API 调用规约等多个方面.通过分析开发人员使用 API 的整体过程以及调研相关文献,本文识别出影响 API 使用的 3 个关键问题,包括:

(1) API 文档质量不高

API 文档是软件复用的关键,也是开发人员学习使用 API 最为重要的材料.1994 年,Parnas[9]的研究指出:如果没有高质量的文档保证,软件复用就难以实现.然而,针对 API 文档的研究显示,现有文档的质量与文档的重要程度并不匹配.2010 年,Robillard 和 DeLine[3]采访了 440 位微软公司的工程师,研究他们使用 API 时面临困境的原因,发现开发人员在学习 API 的时候主要依赖 API 文档的使用,低质量的文档是造成 API 学习使用的最主要障碍.

(2) API 调用规约不完整

API 调用规约描述了开发人员调用 API 时应遵循的规则和约束条件.开发人员如果没有按照规约调用 API,程序将引入缺陷甚至严重的安全问题.2015 年,Saied 等人[10]研究了 JDK 7 中 4 种类型的调用规约在文档中的记录情况,如图 1 所示,结果显示:3 种类型的调用规约在文档中严重缺失,开发人员无法了解正确的 API 调用规约.同时,错误地调用 API 往往更会对应用程序和用户隐私数据的安全造成威胁[5−8].

(3) API 调用序列难以确定

开发人员完成任务的时候通常是以调用 API 序列为基础的,文献[15]总结了 JAVA API 的使用场景,见表 2.但很多复杂任务通常需要多个 API 的交互,例如:Java Ehcache API 用来布创建分式节点间的缓存,分布式节点间的交互需要 Java RMI(远程调用)API 的支持,开发人员面对复杂的情况难以确定正确的 API 调用序列, 严重影响了开发效率,开发人员需要更加完善的 API 推荐系统.

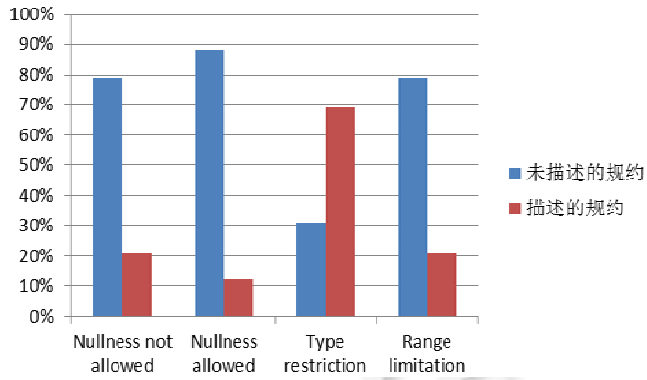

Fig.1 Documented vs non-documented specification

图 1 文档中描述的规约 vs 未描述的规约

Tabel 2 API usage scenario

表 2 API 使用场景

	使用场景	例子	API
类	创建类对象	*List⟨String⟩i=newArrayList⟨⟩*();	*java.util.List*
	变量声明	*String s*=“*abc*”	*java.lang.String*
	使用静态类	*intmax=Math*.max(1,2);	*java.lang.Math*
	继承	*public classmyThread extendsThread*{...}	*java.lang.Thread*
	返回类型	*publicDate getTime*(*String time*){...}	*java.util.Date*
	异常	*try*{...}*catch*(*IOException e*) {...}	*java.io.IOException*
方法	调用实例方法	*intlen=newString*(“*a*”).*length*();	*java.lang.String.length*()
	调用静态方法	*intmax=Math*.max(1,2);	*java.lang.Math*.max(*int*,*int*)
	调用构造函数	*String s=newString*();	*java.lang.String.String*()
属性	访问实例属性	*Point p=newPoint*(); *p.x*=1;	*java.awt.Point.x*
	访问静态属性	*System.out.println*(“Hello World!”);	*java.lang.System.out*

2 API 文档研究

开发人员遇到不熟悉的 API 时,通常需要阅读文档学习 API 的使用方法.Parnas[9]的研究指出:“文档是软件复用的关键,没有高质量的文档保证,软件复用难以实现”.API 文档是开发人员学习使用 API 的重要参考,帮助开发人员了解如何正确地使用 API.根据文档内容的不同,API 文档主要分为以下两类[11].

(1) API 参考文档(API reference documentation):软件生产商官方提供 API 参考文档,与软件库一起发行.API 参考文档通常以 API 元素作为索引,包括概念、指示信息、示例程序等内容,见表 3[11],例如.NET 参考文档、JDK 参考文档等;

(2) 其他文档:包括 API 教程(API tutorial)、FAQ、论坛讨论等多种形式的辅助资料,这些文档的内容通常以实例、问答等内容为基础,帮助开发人员解决使用 API 过程中的实际问题.

Tabel 3 Research on API documentation

表 3 API 文档研究

重要工作	年份	研究对象	研究问题	工作数量统计
Sacramento, *et al*.[12]	2006	.NET 参考文档	文档内容不完整,缺少示例程序、指示信息等	6
Rubio-González, *et al*.[13]	2010	Linux 手册		
Kim, *et al*.[14]	2013	JDK 参考文档		
Subramanian, *et al*.[15]	2014	Android API 文档等		
Chen, *et al*.[16]	2014	JDK 参考文档、Stack Overflow		
Treude, *et al*.[17]	2016	JDK 参考文档、Stack Overflow		

Tabel 3 Research on API documentation (Continued)

表 3 API 文档研究(续)

重要工作	年份	研究对象	研究问题	工作数量统计
Zhou, *et al*.[18]	2016	开源类库参考文档	内容失效	1
Zhou, *et al*.[19]	2017	JDK 参考文档	内容不一致	1
Zhong & Su[20]	2013	JDK 参考文档	内容不正确	1
Dekel & Herbsleb[21]	2009	JDK 参考文档	内容冗余, 无效信息多	3
Monperrus, *et al*.[22]	2012	JDK 参考文档、Jface 参考文档		
Maalej & Robillard[11]	2013	.NET 参考文档、JDK 参考文档		
Petrosyan, *et al*.[23].	2015	JodaTime 教程等	信息碎片化	2
Jiang, *et al*.[24]	2017	JodaTime 教程等		

研究[3,25]显示:低质量的文档是开发人员学习使用 API 的最主要障碍,包括内容不完整、表述不准确等多方面的问题.近年来,API 文档问题受到的关注越来越多,如何提高文档质量成为了 API 领域的研究热点.表 3 对重点工作进行了汇总.

2.1 API参考文档

API 参考文档是开发人员学习使用 API 的最权威指导,文献[12]汇总了 API 参考文档存在的 10 类问题,例如内容不完整、描述不准确、内容冗余、示例程序缺少解释、存在歧义等.研究人员进行了多方面的工作以提高 API 参考文档的质量,包括文档内容分类、文档内容增强、文档质量评估等.

2.1.1 文档内容分类

API 参考文档存在内容冗余、无效信息过多等问题[12].开发人员很难快速定位关键内容,严重影响了工作效率.文档内容分类的目标是明确文档中包含知识的类型,根据文档中内容类别进行归类,提高 API 文档的使用效率.

API 参考文档中的指示信息(directive)描述了开发人员使用 API 的正确方法以及需要注意的约束.2009 年,Dekel 和 Herbsleb[21]最先针对 API 参考文档中的指示信息进行分析,构建了一款 Eclipse 插件 eMoose,针对 JAVA 参考文档中的指示信息进行醒目的标识并推送给开发人员,以提高编程的准确性.

eMoose 基于 API 设计人员对指示信息的手工标记,并没有对 API 参考文档中的指示信息进行系统分析.2011 年,Monperrus 等人[22]针对 API 参考文档中的指示信息进行了经验研究,将指示信息分成了 23 类,包括参数、返回值、异常处理、字符串格式、数值范围等多个方面,为后续研究奠定了知识基础.

2013 年,Maalej 和 Robillard[11]扩大了研究范围,并不局限于指示信息,而是针对 JDK 以及.NET 的参考文档的全部内容进行了深入分析,17 名专业人员利用内容分析的技术对文档内容进行分类,最终文档内容被分成 12 类,见表 4.同时统计了每一类信息所占的比例,结果显示:JDK 参考文档中 43%的内容参考价值并不高;而在.NET 参考文档中,这样的内容占到了 51%.

Tabel 4 Knowledge types taxonomy of API reference documentation

表 4 API 参考文档内容分类

内容类型	描述
功能行为	描述 API 能够做什么
概念	解释 API 元素的含义
指示信息	描述如何正确调用 API 等信息
目的	解释提供该 API 元素的目的
质量	描述 API 的质量性质(性能等)
控制流程	描述 API 具体的控制流程(触发了哪些事件等)
组织结构	描述内部组织结构(继承等)
使用模式	描述如何使用 API 实现一个场景
代码示例	提供实现某一功能的代码示例
环境	描述 API 使用的环境(包括兼容性、版本等信息)
参考引用	指向外部文档的链接
其他	非相关信息

2.1.2 文档内容增强

维护全面完整的文档资料是一项艰巨的任务,即使微软、苹果等公司也很难完成.研究[12,13]显示,API 参考文档在指示信息、示例程序等多个方面存在不完整的情况.研究人员利用公开的示例程序、Stack Overflow 上的问答信息等多种资源对 API 参考文档进行补充.

API 设计人员通常会在参考文档中提供描述性文字来说明如何使用这些 API,但单纯的文字性描述有时会带来歧义,开发人员通常会通过示例程序来学习使用这些 API[26].微软及苹果等公司都在 API 参考文档中加入大量的示例程序.然而,手工编写高质量的 API 程序需要大量的劳动和时间[27,28],难以全部完成.例如,JDK5 的参考文档包含了 27 000 种方法,但只有大约 500 个方法包含了示例程序.针对这样的问题,文献[14,15,29,30]开始将公开的示例程序与 API 参考文档链接起来.2013 年,Kim 等人[14]构建了一个文档增强系统 eXoaDocs,将文档与公开的示例程序链接起来.eXoaDocs 首先通过搜索引擎创建一个示例程序候选库,然后抽取程序的语义特征,对示例程序进行评分,找到最具代表性的示例程序嵌入到 JDK 文档中,为 JDK 中 75%的方法增加了示例程序.

当开发人员使用 API 遇到困难时,经常向 Stack Overflow 等社区寻求帮助[31],这些社区资源覆盖了很多 API 相关信息,但 API 参考文档与这些资源缺乏联系.2016 年,Treude 等人[17]观察到 Stack Overflow 中包含了很多 API 相关的重要信息,比如调用方法等,而 API 参考文档却没有包括这些信息.Treude 等人研发了 SISE 系统,利用机器学习方法从 Stack Overflow 中抽取 API 相关的关键语句来补充文档内容,使用了包括语句自身特征(单词数、语句的位置、API 元素的位置等)、问题特征(问题中是否包括 API 元素、问题关注度等),答案特征(答案评分、回答者信息等)等多维特征来抽取关键语句补充文档内容.类似的,文献[16]将 Stack Overflow 中的信息以 FAQ 的形式整合到 API 参考文档中.

2.1.3 文档质量评估

软件库开发升级速度越来越快,撰写校对文档需要大量的时间和投入,很难及时维护.与此同时,撰写文档的人员很多时候并不参与实际的项目开发.在这种情况下,API 参考文档非常容易存在一些错误或信息不一致等情况,严重误导开发人员.但手工分析这些错误非常耗时,研究人员开始尝试自动化地发现这些问题.

2013 年,Zhong 和 Su[20]最先开始利用自然语言处理技术与代码分析技术针对 API 参考文档中存在的问题进行分析,这些问题包括:(1) API 参考文档描述了失效的 API;(2) API 参考文档存在语法错误[32];(3) API 参考文档中的示例程序使用了不存在的类或方法.

结果显示:针对 5 个常用的 JAVA 类库文档,该方法成功检测出了 1 000 多个文档错误.类似的,2016 年,Zhou 等人[18]对 API 失效问题进行了进一步分析,指出:API 文档中很多示例程序没有及时更新,经常处于过期状态.

2017 年,南京航空航天大学的周宇等人[19]研究发现,API 参考文档存在指示信息与示例程序不一致的情况.如,在关于 *java.awt.event.InputEvent* 类的文档中描述,若参数 *button* 小于 0 或大于某个数,方法 *getMaskForButton*(*int button*)会抛出 *IllegalArgumentException* 异常.但是在示例代码中,该异常在 *button*=0 时也会抛出,该文档对参数约束进行的描述存在不准确的情况.针对这种文档缺陷,研究人员提出了一种自动化的方案来检测 API 文档描述的缺陷,分别抽取文档中的指示信息与示例程序的约束条件进行逻辑验证.结果显示:针对 JDK1.8 参考文档,该方法检测出了 1 158 个文档缺陷,准确率达到 81.6%,召回率达到 82.0%.

2.2 其他文档

上节主要介绍了关于 API 参考文档的研究.近年来,研究人员也开始关注 API 教程[23,24]、开发人员博客[33]、论坛讨论[16,17,31]等包含 API 知识的其他文档.这些文档通常以实例、问答等内容为基础,帮助开发人员解决使用 API 过程中的实际问题.第 2.1.2 节介绍了利用论坛讨论等资源增强 API 参考文档的方法.

2015 年,加拿大麦吉尔大学的研究人员[23]研究发现,API 教程存在信息碎片化的问题.与 API 参考文档不同,API 教程的内容通常是基于任务进行组织的,关于 API(类、接口等)的信息被分散在很多章节,并不集中.开发人员很难找到某类 API 的信息.提出了一种基于语言和结构特征的监督模型,对 API 教程中的信息进行分类,提取出某类 API 的相关信息.基于监督模型的方法需要大量人工参与,同时,提取的特征依赖于训练集的语料库,在实际应用中并不准确.2017 年,大连理工大学的江贺等人[24]提出了一种无监督模型 FRAPT,去除 API 教程中的无

关信息,得到与 API 信息真正相关的段落.主要包括以下几个步骤.

(1) 段落解析器解析 API 教程中的段落信息、预处理文字信息,利用 API 名称或本体名称替换代词等;

(2) 段落过滤器过滤所有与 API 信息无关的段落,同时使用主题模型以及 PageRank 算法对段落与 API 的相关度进行评分.根据评分得到与 API 真正相关的段落.

综合以上研究,结合表 3 可以看出:API 文档问题受到越来越多的关注,解决好文档问题能够极大地帮助开发人员提高工作效率.从研究对象来看,研究主要集中在 API 参考文档上(12 篇).但近年来,研究人员也开始关注 API 教程、论坛讨论等文档资源.从研究问题来看,较多的研究(6 篇)针对 API 参考文档内容不完整的问题进行了深入研究.2013 年以后,研究人员开始关注 API 参考文档的内容错误、不一致等质量问题,更加全面地分析和解决 API 文档的质量问题.

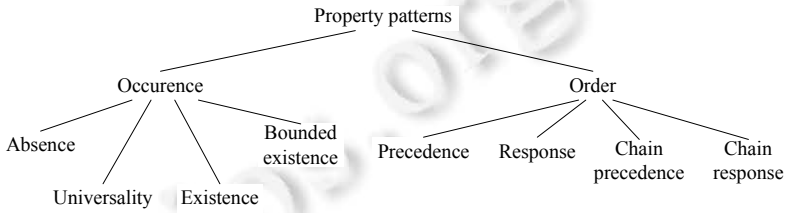

Fig.2 API specification pattern

图 2 API 调用规约模式

3 API 调用规约研究

API 调用规约描述了开发人员调用 API 时应遵循的规则和约束条件.如果没有按照规约调用 API,程序将引入缺陷甚至严重的安全问题.但是,编写维护 API 调用规约代价非常高,官方文档很少会提供完整的 API 调用规约.为此,研究人员提出了很多方法自动推断 API 调用规约,用于缺陷检测、程序验证、文档增强等领域.表 5 对重点研究进行了汇总.

Tabel 5 Tools of API specification inference

表 5 API 调用规约推断工具

重点工作	年份	工具名称	技术特点	规约模型	工作数量统计
Cook&Wolf[34]	1998		FSA	关联规则	14
Dallmeier, *et al*.[35]	2006	ADABU	FSA		
Pradel & Gross[36]	2009		PFSA		
Li & Zhou[37]	2005	PR-Miner	数据挖掘		
Livshits & Zimmermann[38]	2005	DynaMine	数据挖掘		
Yang, *et al*.[39]	2006	Perracotta	数据挖掘		
Thummalapenta & Xie[40]	2009	Alattin	数据挖掘		
Liang, *et al*.[41]	2016	AntMiner	数据挖掘		
Murali, *et al*.[42]	2017	Salento	贝叶斯模型		
Gabel & Su[43]	2008	Javert	$(ab^{+}c)^{*}$模板		
Gabel & Su[44]	2009		$(ab)^{*}$模板		
Gabel & Su[45]	2010	OCD	$(ab)^{*}$模板		
Zhong, *et al*.[46]	2010	Doc2Spec	NLP+模板		
Nguyen, *et al*.[47]	2009	GrouMiner	图模型		
Wasylkowski & Zeller[48]	2009	Tikanga	静态分析	Pre/Post 条件	3
Wei, *et al*.[49]	2012	AutoInfer	动态分析		
Nguyen, *et al*.[50]	2014		静态分析		
Henkel & Diwan[51]	2007	Chronicler	动态分析	行为模型	4
Ramanathan, *et al*.[52]	2007		静态分析		
Lorenzoli, *et al*.[53]	2008		GK-tail		
Krka, *et al*.[54]	2014		动态分析		

1999 年,Dwyer 等人[55]整理了 500 多份 API 调用规约,抽象出基于状态\事件的规约模式,如图 2 所示,例如

Existence(出现:状态\事件在一定范围内必须出现),Absence(不出现:状态\事件在一定范围内不能出现)等.随后,机器学习、数据挖掘等技术也广泛应用于推断 API 调用规约,取得了一系列研究成果.文献[56]针对 API 规约推断技术进行了系统性分析.按照 API 调用规约的描述方法,现有研究分成如下几类.

(1) 基于关联规则的调用规约:此类规约主要描述 API 间的调用关系.例如:调用函数 p 的同时必须要调用函数 q;调用函数 a 前必须要调用函数 b.这种调用规约模型并没有针对 API 的前置后置条件进行分析;

(2) 基于 Floyd-Hoare 逻辑的 pre/post 条件规约:pre/post 条件规约明确指出每一条 API 调用的前置以及后置条件,包括参数、返回值等约束条件;

(3) 基于行为模型的调用规约:基于行为模型的调用规约以更加正式的形式描述 API 的行为,强调 API 调用对于程序状态的影响,常见形式包括契约式规约(contract)、代数式规约(algebraic specification)等.

3.1 基于关联规则的调用规约

基于关联规则的调用规约主要描述 API 间的关联关系,例如:调用函数 p 的同时必须要调用函数 q;调用函数 a 前必须要调用函数 b.这种调用规约并没有针对 API 中具体的前置以及后置条件进行分析,同时,大部分的挖掘方法并没有深入分析代码的数据流以及控制流.

2001 年,斯坦福大学的 Engler 等人[57]提出了一种新的检测程序缺陷的思路,区别于以往的静态检测工具,新模型并不指定具体的规约条件,而是设计了一系列规约模式,避免了大量人工搜集规约条件的问题,例如:调用函数 a 前必须要调用函数 b 等,但并未具体指定 a 和 b 是什么函数.基于这样的假设:软件库中频繁出现的调用规约即认为是正确的.利用统计分析的方法从软件库程序中抽取具体的函数对作为调用规约,提高了静态分析工具的准确性.该项研究为后续工作提供了新的思路.目前,常见的方法有如下几类:

3.1.1 基于频繁项集的挖掘方法

研究人员将调用规约的提取问题转化为频繁子项的数据挖掘问题,利用数据挖掘技术推断 API 调用规约.基于频繁项集的数据挖掘算法在这些研究中起到关键作用.该类算法旨在从大量事务数据库中发现事务之间的关联关系,以揭示隐藏其中的行为模式,强调行为的共现性.通过计算频繁子项得到类似 $A\rightarrow C$ 的规则.其中,A 称为前件,C 称为后件.关联规则规定了如果事务中包含了前件 A,也极有可能会包含后件 C.

2005 年,Li 和 Zhou[37]设计研发了工具 PR-Miner,将函数映射成数据子项的形式存储在数据库中,利用 FPclose 算法[58]挖掘 API 调用规约,并利用这些规约在 Linux kernel,PostgreSQL 数据库等大型软件程序中发现了 23 个确认的 bug.DynaMine[38]与 PR-Miner 的整体框架类似,但 DynaMine 并不是针对软件库程序进行分析,而是从历史提交的代码更改中挖掘关联规则,DynaMine 采用了 Apriori 数据挖掘算法[59].

基于频繁项集的挖掘方法依赖于数据源的质量,如果数据源噪声过多,将严重影响结果的准确性.2016 年,中国人民大学的梁彬博士团队研发了 AntMiner[41],研究指出:大量与关键操作无关的程序语句给系统分析带来了噪声,影响了挖掘的准确性.AntMiner 系统采用程序分片技术抽取出关键操作,降低噪声的干扰,提高了挖掘算法的精度.研究人员定义了两类关键操作:(1) 函数在执行前参数需要进行校验的操作,如果校验失败,该函数无法执行;(2) 所有 return 语句.结果显示:AntMiner 报告了 52 个 Linux Kernel 中的 bug,并得到了官方确认.2017 年, Murali 等人[42]的工作更进一步,利用贝叶斯概率模型降低数据噪声对于结果的影响.

3.1.2 基于序列模式的挖掘方法

第 3.1.1 节介绍的基于频繁项集的挖掘方法强调子项的共现性,主要表现在一个子项出现的同时必然会伴随其他子项的出现,并未强调 API 调用的先后次序.最简单的 API 调用次序表示为(a,b),其中,a 必须保证在 b 之前执行.基于序列模式的挖掘方法主要用于推断 API 调用之间的时序关系,例如 Perracotta[39],Alattin[40]等工具.

2007 年,Kagdi 等人[60]比较了基于频繁序列的算法与基于频繁子项的算法的效果,显示前者产生的调用规约准确度较高.同时,Kagdi 等人在文献[61]中利用 SPADE 算法抽取基于调用位置的 API 调用规约模型,利用上下文信息降低原有规约在检测 bug 时候的误报率.

2009 年,谢涛等人[62]针对异常处理机制中的 API 关联关系进行了深入分析,提出了新的关联规则模型$(FC_{c1}\ldots FC_{cn})\wedge FC_a\Rightarrow(FC_{e1}\ldots FC_{en})$,主要用于挖掘异常处理方面的调用规约.例如:如果系统在多种上下文条件

(FC_{c1}…FC_{cn})下某些操作 FC_a 发生异常,将会出现回滚操作 FC_{e1}.最后,结合频序列模式挖掘算法得到新的调用规约.

3.1.3 基于自动机的挖掘方法

自动机模型能够描述 API 之间的调用关系.1998 年,Cook 和 Wolf[34]首次采用匿名自动机来表示 API 调用序列,匿名状态机采用边来表示函数调用,缺乏状态信息.2006 年,Dallmeier 等人[35]设计了工具 ADABU 分析对象的行为,同时为自动机添加了状态信息,如图 3(a)所示.例如:调用 *init*()方法创建了一个空的 Vector 后的状态标记为 *isEmpty*(),如果调用了 *add*()方法后的状态被标记为¬*isEmpty*().2009 年,Pradel 和 Gross[36]实现了基于概率的有限状态自动机(PFSA)模型,该方法使用状态来表示调用事件,如图 3(b)所示.同时,在边上标记了事件发生的概率.文献[63−66]也都采用了类似的思路挖掘 API 的调用规约.

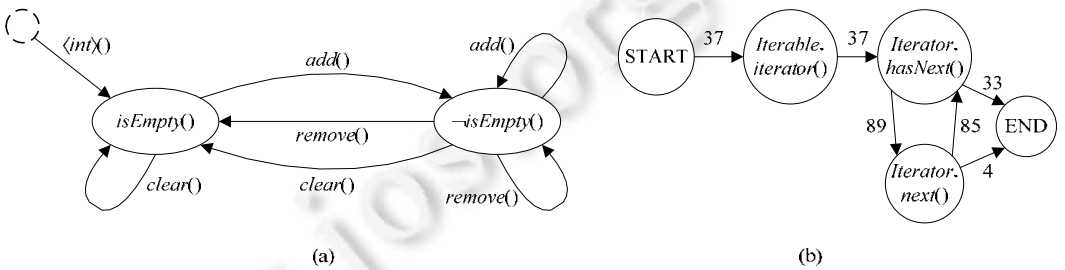

Fig.3 Automaton model

图 3 自动机模型

3.1.4 基于模板的挖掘方法

研究人员采用基于模板的挖掘方法寻找符合特定模板的 API 调用规约.但模式匹配算法的开销很大,大部分研究都是针对(*a*,*b*)的二元简单模式进行匹配.2008 年,Gabel 和 Su[43]对模板技术进行了扩展,使用模板表示资源使用过程为$(ab)^*$,如图 4 所示,包括资源请求、资源使用以及资源释放等 3 部分,在客户程序中搜索符合此模板的 API 调用规约.随后,文献[44,45]推断更加复杂的调用规约.例如,Gabel 和 Su[44]研发了 Javert,Javert 首先定义了两个简单的规则模板$(ab)^*$以及$(ab^+c)^*$,然后组合这两种模式为复杂的调用规约.

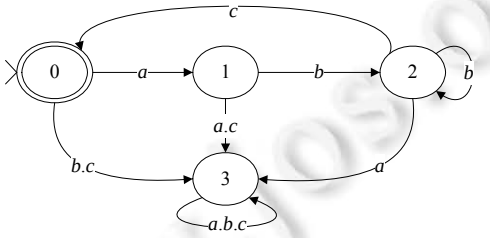

Fig.4 Template representing $(ab^+c)^*$

图 4 $(ab^+c)^*$模板

2009 年,Zhong 等人[46]首次提出从 API 文档推断资源调用规约的方法 Doc2Spec.Doc2Spec 定义资源使用模板如图 5 所示,该方法主要包括以下 3 个部分.

(1) 从 API 文档中抽取关于方法的描述;

(2) 利用 NLP 技术,从每一个方法的描述中构建(资源-行为)的对应关系;

(3) 最后,针对一种资源的所有操作分类到一个集合中,根据预先定义的模板推断出 API 调用规约.

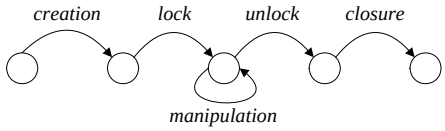

Fig.5 Template of resource usage

图 5 资源使用模板

3.1.5 基于图模型的挖掘方法

图模型相比较自动机等模型能够表达更加复杂的信息,图的边不仅可以表示控制依赖关系,还可以表示数据依赖关系.2007 年,Chang 等人[67]使用程序依赖图来表示 API 调用规约,如图 6(a)所示,虚线表示数据依赖关系,实线表示控制依赖关系.2008 年,钟浩博士[68]直接分析 API 源程序,提出 JRF 工具挖掘程序规则图,如图 6(b)所示,该图中的点表示函数,边则表示函数间的调用关系.例如:调用 *method*3 前需要调用 *method*1.2009 年,Nguyen 等人[47]提出了 GrouMiner,使用标记有向图来表示 API 调用的依赖关系.

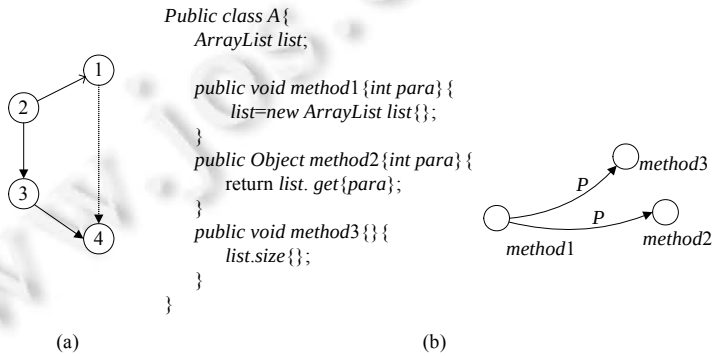

Fig.6 Graph models

图 6 图模型

3.2 基于Floyd-Hoare逻辑的pre/post条件规约

开发人员使用 API 完成软件功能时,需要遵守其前置及后置条件.例如:JDK 中的 String 类,在调用 *subString*(*beginIndex*,*endIndex*)前,必须确认参数 *beginIndex*⇐*endIndex* 的条件是否成立.这些前置和后置条件是 API 调用规约的重要组成部分,在程序验证、测试用例生成、缺陷检测等多个领域有着广泛的应用.

Wasylkowski 和 Zeller[48]研发了 Tikanga 工具,主要分析 API 的前置条件,得到的前置条件是方法或者函数的调用序列,并将这些模式定义为操作性前置条件.Ramanathan 等人[52]开发的 Chronicler 工具也具备类似的功能,同时对程序数据流进行了简单的分析,例如参数不能为空等.

2014 年,Nguyen 等人[50]将静态分析与数据挖掘技术相结合,从大量的客户程序中自动挖掘 API 的前置调用条件.基于这样的假设:在大量的客户程序中,正确的 API 的前置条件将会频繁出现.采用了 3 000 多个 JAVA 工程作为数据源,包含了超过 400 万个方法.该方法通过静态分析技术构建控制流图,确定 API 调用的前置条件.前置条件定义如下:

在控制流程图中,判断式 p 到方法调用 C 的所有路径都出自同一条边(TRUE 或者 FALSE),那么就定义判断式 p 为方法调用 C 的前置条件.如果判断式 p 的两条边(TRUE 与 FALSE)都能够到达方法调用 C,则判断式 p 不是其前置条件.整体的分析过程主要分为以下几个步骤.

(1) 对于每一个方法,利用静态分析技术构建控制依赖关系,分析出 API 调用的前置条件;

(2) 将这些前置条件进行规范化处理,对等价的关系进行合并;

(3) 对不经常使用的条件进行过滤,得到最终的前置条件.

结果显示:该方法达到了 82%的条件覆盖率,同时发现了 5 个官方文档缺失的前置条件.相比较于前置条件,开发人员更容易忽视后置条件的使用.Wei 等人[49]设计了自动推断工具 AutoInfer,采用动态分析技术推断 API 的后置条件.

3.3 基于行为模型的调用规约

基于行为模型的调用规约以更加正式的形式描述 API 的行为,强调 API 对于程序状态的影响.常见的形式包括契约式规约(contract)、代数式规约(algebraic specification)等.

契约式规约是程序验证的基础性技术[69],用于定义组件的行为,方便过滤无效输入以及定位错误等.契约式规约通常包括前置条件(方法调用前,必须保持为真的条件)、后置条件(方法调用成功返回后,必须保持为真的条件)以及不变式(程序运行过程中,始终保持的性质).例如,图 7 描述了 EiffelBase 中 *LINKED_LIST* 类的 *merge_right*(·)方法的规约.Require 字段描述了方法的前置条件,Ensure 字段描述了方法的后置条件.

```
Merge_right(other: LINKED_LIST[G])
require
    not after
    other≠Void
    other≠Current
ensure
    count=old count+old other.count
    index=old index
end
```

Fig.7 Contract of *merge_right*(·) in *LINKED_LIST*

图 7 *LINKED_LIST* 类的 *merge_right*(·)方法的规约

2000 年,Ernst 博士[70]率先采用动态分析技术推断程序不变式,研发了 Daikon 工具,通过监测程序执行过程中变量的数值推断出程序不变式,生成契约式调用规约,取得了良好的效果,系统整体框架如图 8 所示.

文献[71–73]针对更加复杂的不变式模型进行了研究.

2008 年,Lorenzoli 等人[53]利用不变式扩展了第 3.1.3 节介绍的自动机模型,设计了 GK-tail 算法生成扩展的自动机模型.在自动机的边上标记变量取值范围,将数值约束与交互行为联系起来.2014 年,Krka 等人[54]分析了自动机模型引入不变式在表达 API 调用规约时的具体效果,研究人员比较了动态推断自动机模型的 4 种策略:(1) 只从执行路径推断;(2) 只从不变式推断;(3) 从执行路径推断后利用不变式增强;(4) 从执行路径推断后利用不变式增强.结果显示:利用不变式使模型精度提高 4%,召回率提高 41%.

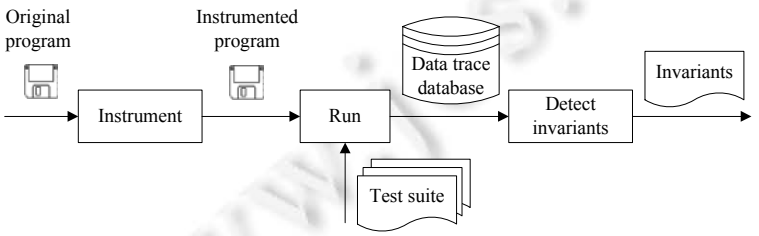

Fig.8 Daikon framework

图 8 Daikon 整体框架

除了契约式规约,代数式规约也用来描述 API 的行为.代数式规约更加强调 API 调用对于程序状态的影响以及 API 之间的关系.1978 年,Guttag 等人[74]就展开了针对代数式规约的研究,同时给出了代数式规约的定义,主要包含两个部分:(1) 代数式签名(algebraic signature),包括语法和类型信息;(2) 公理(axioms),定义操作之间的关系.例如:JAVA Intstack 类的代数式规约如图 9 所示.

2007 年,Henkel 和 Diwan[51]通过动态分析的方法推断 Java container 类的代数式规约,分析 API 之间的关系生成公理内容,用以补充文档内容.2014 年,Bagge 等人[75]比较了 API 的代数式规约与 Pre/Post 规约,指出代数式

规约更适合描述 API 调用规约.Pre/Post 规约侧重一条指令的输入以及输出的条件,而代数式规约更加注重描述不同操作之间的关系.

Type *IntStack*
FUNCTIONS
IntStack:→*IntStack*×*void*
push:*IntStack*′*int*→*IntStack*×*void*
pop:*IntStack*→*IntStack*×*int*
AXIOMS
∀*s*: *IntStack*,*i*:*int*
pop(*push*(*s*,*i*).*state*).*retval*=*i*
pop(*push*(*s*,*i*).*state*).*state*=*s*
pop(*IntStack*().*state*).*retval*~*ArrayIndexOutOfBoundsException*

Fig.9 Algebraic specification of *IntStack*

图 9 *IntStack* 类的代数式规约

综合以上研究,结合表 4 可以看出:自 1998 年 Cook 和 Wolf 首次使用自动机模型挖掘 API 调用规约模型以后,越来越多的挖掘方法应用于提取 API 调用规约.研究的数据源包括大型软件程序、基于搜索引擎的相似示例程序、代码库中的公开项目、软件文档、历史提交的代码更改等多元化数据.

现有研究主要针对 API 调用规约的某些模式进行了推断分析,例如:PRMiner[37]关注了的 API 之间的关联关系,Nguyen 等人[50]分析了 API 的前置条件等.这样得到的结果只体现了规约的一个方面,很难完整概括 API 调用规约,并没有形成统一的标准.

同时,API 调用规约的有效性需要进一步加强.2016 年,伊利诺伊大学的 Legunsen 等人[76]分析了前期研究提取的 JAVA API 调用规约的有效性,针对 200 个公开项目进行缺陷检测,得到了 652 条警告信息,其中只有 74 条确定为 bug,达到了 80%的误报率.

4 API 推荐研究

随着软件系统的规模和复杂性急剧增长,软件开发和维护的代价也在持续加大[135,136].如何高效重用现有的软件库和框架,推荐相关代码给开发人员,是软件工程领域的一个研究热点.日益兴起的 API 推荐技术能够针对大规模程序进行分析和规律挖掘,减少开发人员查找、理解、组合、调试代码的工作量,降低人工错误率,提高软件的质量.相关研究主要集中在类库推荐、方法推荐、参数推荐等多个方面,表 6 对重点工作进行了汇总.

Tabel 6 Tools of API Recommendation

表 6 API 推荐工具

重点工作	年份	工具名称	技术特点	推荐类型	工作数量统计
Teyton, *et al*.[78]	2012		关联规则	类库推荐	3
Thung, *et al*.[79]	2013		关联规则		
Chen & Xing[80]	2016	SimilarTech	NLP 技术		
Xie & Pei[81]	2006	MAPO	关联规则	方法推荐	8
Hindle, *et al*.[82]	2012		自然语言处理		
Wang, *et al*.[83]	2013	UP-Miner	关联规则		
Raychev, *et al*.[84]	2014	SLANG	自然语言处理+RNN		
Nguyen, *et al*.[85]	2015	GraLan	图模型		
Nguyen, *et al*.[86]	2016	SALAD	隐马尔科夫模型		
Fowkes & Sutton[87]	2016	PAM	概率模型		
White, *et al*.[88]	2016	DeepAPI	深度学习		
Zhang, *et al*.[89]	2012	Precise	KNN	参数推荐	2
Asaduzzaman, *et al*.[90]	2015	Parc	SimHash		
Xing, *et al*.[91]	2007	Diff-CatchUp	文本相似度	API 映射推荐	4
Dagenais & Robillard[92]	2008	SemDiff	API 上下文关系		
Zhong, *et al*.[93]	2010	MAM	文本相似度		
Nguyen[94]	2017	API2API	Word2Vec		

4.1 API类库推荐

文献[77]针对 GitHub 上面的 1 008 个项目统计显示:93.3%的项目使用了第三方类库,平均每个项目使用了 28 个第三方类库.开发人员选择合适的类库时面临挑战.2012 年,Teyton 等人[78]分析了第三方类库的演进过程,通过构造类库迁移图推荐更为合适的类库.例如:很多项目的类库从 Log4J 迁移成 SLF4J,系统便会构建类库迁移图,当开发人员使用 Log4J 时,系统就会推荐 SLF4J 为更好的选择.2013 年,Thung 等人[79]挖掘了类库之间的关联规则,发现了很多项目在使用类库的时候存在相似性,根据项目已经使用的类库推荐将会使用的第三方类库.2016 年,Chen 和 Xing[80]开发了 SimilarTech 支持跨语言推荐功能,SimilarTech 支持跨 6 种语言总共 6 715 个类库.SimilarTech 使用 NLP 技术解析 Stack Overflow 上关于类库的标签等信息,创建类库的知识基础,根据知识基础推荐相似功能的类库.

4.2 API方法推荐

Wang 和 Godfrey[95]研究发现:开发人员难以确定正确的 API 调用序列,在 stackoverflow 社区存在众多关于 API 调用序列的问题.例如,关于 UIScrollView 类的相关问题的浏览量达到了 8 422 次[95].为此,研究人员提出了多种方案挖掘 API 调用序列,并根据客户程序的上下文推荐合适的 API 方法.

4.2.1 基于数据挖掘的方法

2006 年,Xie 和 Pei[81]最先提出了挖掘 API 调用序列的经典算法 MAPO.MAPO 通过代码搜索引擎找到大量相似的代码片段,解析 JAVA 源文件并抽取 API 方法调用.然后,根据类名称、方法名称以及 API 名称计算 API 调用序列的相似度进行聚类.针对每一个子类,MAPO 使用 SPAM 算法挖掘出 API 的调用序列.

MAPO 算法返回的结果存在较大的冗余性,会出现很多类似的 API 调用序列.UP-Miner 扩展了 MAPO 算法试图降低结果的冗余性[83],挖掘出更加简明准确的 API 调用序列.UP-Miner 进行了 3 个方面的优化.

(1) 使用 BIDE 闭合频繁序列挖掘算法来挖掘 API 调用序列;

(2) 根据两个 API 调用序列的子项的重复性来度量相似性;

(3) 采用概率图模型来表示 API 调用序列,同时根据出现的频次进行排名.

序列模式挖掘算法在 API 调用序列挖掘过程中起着至关重要的作用,也一直是研究的热点问题.Agrawal 和 Srikant[96]在分析超市客户交易数据时,首次提出了序列模式挖掘的问题.之后,大量的研究集中于该领域,包括 GSP 算法[97]、PrefixSpan 算法[98]、SPADE 算法[99]以及 SPAM 算法[100].序列模式挖掘算法通常会伴随模式爆炸问题:如果设定的最小支持度太低,算法就会返回大量的序列模式结果,并且基于频次的挖掘方法得到的模式子项很多是无关的,是造成结果不准确的主要原因.为了解决模式爆炸问题,出现了很多解决方案,例如 BIDE 算法[101]、SQS-search 算法[102]以及 GoKrimp[103]算法.基于这些领先的算法,2016 年,Fowkes 和 Sutton[87]提出了基于概率分布的序列模式挖掘算法 PAM,结果显示:PAM 达到了最好的 69%的准确度,优于 MAPO, UPMiner 等经典工具.

4.2.2 基于自然语言处理的方法

自然语言处理技术广泛应用于机器翻译、语音识别、拼写纠错等领域,并且取得了良好的效果[104].2012 年,Hindle 等人[82]提出了一种观点:程序语言同样具有重复性和可预测性.同时,将自然语言处理技术(例如 *n*-gram 语言模型等)应用于 API 推荐等软件工程领域.2014 年,Raychev 等人[84]研究开发了 SLANG 系统,使用 *n*-gram 模型预测 API 的调用序列.使用 *n*-gram 模型基于以下两种假设.

(1) 假设序列中的子项是完全有序的;

(2) 假设一个子项的产生概率依赖于该子项前 *n* 项的出现情况.

但使用 *n*-gram 模型存在的主要问题是:API 调用序列并没有严格的顺序要求.如:很多对象初始化的操作可以调换位置,但 *n*-gram 模型有长度的限制,两个 API 调用位置可能离得很远,中间夹杂着很多无关的逻辑操作.

基于 *n*-gram 模型的局限,2015 年,Nguyen 等人[85]提出了一种基于图的统计语言模型 GraLan,该模型使用图结构来表示 API 调用序列,每一个节点用来表示 API 调用,每一条边代表两个节点之间存在控制关系或者数据

流联系.如果有新的节点和边加入,原图便会生成子图.GraLan 通过代码库学习 API 使用图结构,并计算原图产生新子图的概率,分析 API 的调用模式.结果显示:Gralan 的准确度比 *n*-gram 模型提高了 20%.虽然图模型可以更好地表现 API 的上下文关系,但子图的数量急剧增加,需要更多的时间来训练模型,同时需要更多地空间在存储子图结构.

2016 年,Nguyen 等人[86]针对移动平台的 API 调用序列进行了更为深入的研究,设计了 SALAD 系统,SALAD 使用隐马尔可夫模型学习 API 调用序列,图 10 显示了 FileReader 类以及 BufferedReader 类的隐马尔可夫模型.

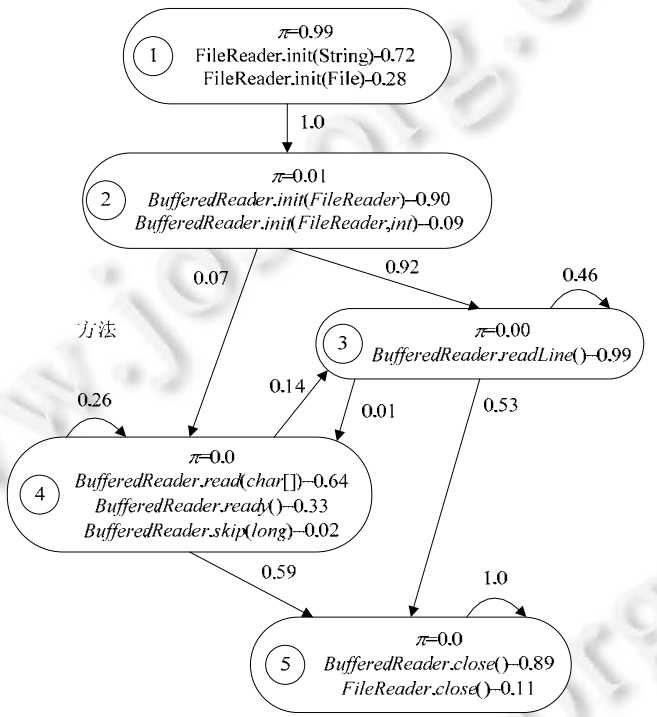

Fig.10 HAPI for *FileReader* and *BufferedReader* objects

图 10 *FileReader* 类以及 *BufferedReader* 类的隐马尔可夫模型

以上研究主要针对客户程序的分析,2016 年,Nguyen 等人[105]研究发现历史提交的代码更改同样具有重复性,研发了基于代码更改的 API 推荐工具 APIREC,选取了 50 个 GitHub 上面的开源项目作为训练数据,总共涉及 113 103 条更改.结果显示:针对前 5 的推荐结果,APIREC 达到了 77%的准确率.

4.2.3 基于机器学习的方法

神经网络模型有效地解决了自然语言处理中长距离依赖问题,被广泛应用于模式识别、自动问答等领域.2014 年,Raychev 等人[84]使用 RNN 模型学习 API 调用序列.RNN 模型[106]依次读取 API 方法,并预测下一步 API 方法出现的概率.在第 i 步,估计第 i+1 步 API 方法出现的概率 $p(y_{i+1}|y_1\ldots y_i)$,如图 11 所示.

(1) 首先,在输入层将当前的 API 方法 y_i 映射为向量 Y_i.$Y_i=input(y_i)$;

(2) 根据之前的隐藏层状态 h_{i-1} 和当前的输入 Y_i 生成新的隐藏层的状态 h_i.$h_i=f(h_{i-1},Y_i)$;

(3) 最后,根据隐藏层的状态计算 $p(y_{i+1}|y_1\ldots y_i)=g(h_i)$.

输出的结果是第 i+1 步 API 方法出现的概率,文献[107]给出了详细的参数训练过程.2016 年,Gu 等人[88]利用深度学习技术来学习 API 调用序列,采用 RNN Encoder-Decoder 模型[108]扩展了之前的神经网络模型,根据用户输入的文本查询,返回可用的 API 调用序列.

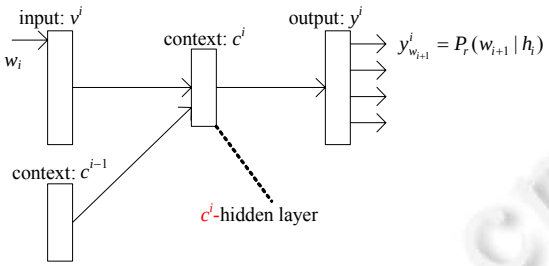

Fig.11 Model of a recurrent neural network (RNN)

图 11 RNN 模型

4.3 API参数推荐

研究[109,110]显示:代码完成过程中参数选择并不容易,一旦选择不正确的参数,程序有可能引入缺陷.2012年,上海交通大学的章程等人[89]提出了一种推荐 API 参数的方法 Precise.Precise 基于当前的上下文条件以及先前的程序样例进行参数推荐,整体过程主要分成 3 个部分.

(1) 首先,Precise 通过分析代码库抽取参数实例的特征创建参数基础库,特征包括参数抽象表示特征以及参数使用上下文特征;

(2) Precise 利用 *K* 近邻算法在参数数据库中进行查询,找到相似参数的候选集合;

(3) 最后,根据上下文的相似性以及使用频率对候选集合进行排序,帮助开发人员选择合适的参数.

Precise 针对布尔型参数,简单的实参推荐效果并不理想,例如 *frame.setVisible*(true)等.2015 年,Asaduzzaman 等人[90]研究发现,参数的使用存在局部性,例如:参数使用前一般会在临近的地方进行初始化.对 Precise 进行扩展研究发了 Parc,抽取参数前 *k* 行出现的关键词(方法名、类名、接口名称等)作为局部特征创建参数基础库,如图 12 所示.最后,Parc 利用 simhash 算法计算相似度推荐合适的参数.结果显示:Parc 给出的前 10 推荐结果的准确度达到了 72.06%.

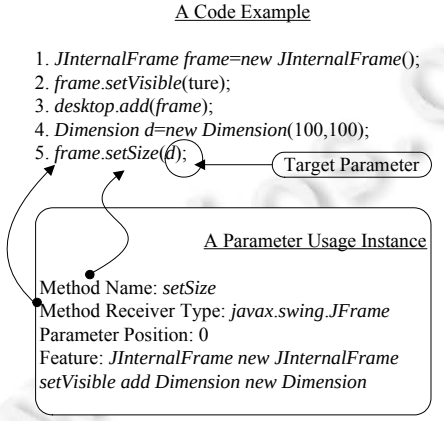

Fig.12 Parameter usage instance

图 12 参数使用实例

4.4 不同版本API推荐

软件类库版本发生演化或变迁均会造成原有推荐方案的失效.例如简单方法重构以及软件架构更新等.高频率的软件库修改和演化,必然会引发开发人员对已有失效推荐方案的检测和更新.开发人员需要挖掘不同软件库版本间的 API 映射迁移关系,利用映射关系对客户程序进行变更推荐.相关工作将迁移映射关系形式化为

规则“$I \rightarrow R$”.其中,I 表示旧版本的 API 元素集合,R 表示新版本中用于替换 I 的 API 元素集合,映射操作“→”给出具体的替换方法和步骤.按照推荐的策略的不同,API 映射模式的推荐工作可分为以下两类.

4.4.1 基于相似性度量的方法

API 版本演化过程中,85%的映射模式为简单重构[91],例如 API 元素重命名等.

Diff-CatchUp[111]以及 RefactoringCrawler[112]工具均能够挖掘这些简单重构模式.Diff-CatchUp 首先使用重构检测工具 UMLDiff 检测并记录新旧版本代码之间的差异.例如,旧版本中 *PlotFit.getFit*()方法被移除.然后,利用一种相似性度量的启发式方法识别出新版本中可替换 *PlotFit.getFit*()的方法集合,替换的依据是:如果相似类所含方法的返回值相同,就进行替换.针对一组候选集合,Diff-CatchUp 利用元素的名称、继承关系、引用关系以及关联关系这 4 种特征进行相似度度量.最后,根据评分推荐最为合适的元素.

2010 年,钟浩博士[93]设计的 MAM 工具能够挖掘不同语言编写的而具有相同功能的 API 映射关系,MAM 的分析目标是同时具有 Java 和 C#版本的项目,例如 Lucence 等.首先,MAM 利用类名称与方法名称的相似性对两个版本的程序进行对齐操作,例如 *IndexFile.java* 与 *IndexFiles.cs* 进行对应;然后,基于类成员名称的相似性形成类之间的映射关系;最后,通过构建 API 转换图实现方法的映射.结果显示,MAM 准确率达到了 80%.

4.4.2 基于 API 调用上下文关系的方法

基于文本相似度的方法对于方法名完全不一致的情况难以取得理想的结果.文献[92,113]通过分析 API 调用的上下文关系来获取 API 映射关系.2008,Dagenais 和 Robillard[92]研发了 SemDiff,SemDiff 利用 API 调用上下文关系来推断不同版本 API 间的映射关系,例如:旧版本中 *main* 函数调用了 *open* 和 *close*,新版本中 *main* 函数调用了 *open*2 和 *close*,*open* 与 *open*2 便存在“1 to 1”的映射关系.2010 年,Wu 等人[137]更进一步,对不同版本“1 to *n*”和“*n* to 1”类型映射关系模型进行了研究.

2017 年,爱荷华州立大学的的 Nguyen 等人[94]研发了 API2API,利用 Word2Vec 模型来刻画一个 API 方法.Word2Vec 模型先前用来捕获自然语言文本中的规律,使用一个单词周围的词汇来刻画这个单词,很多在类似语境下的单词被映射到一个向量空间中,通过向量操作来捕获这些单词的语义关系.API2API 利用 Word2Vec 使用 API 调用序列中周围 APIs 来表示一个 API 方法调用,其本质也是一种基于 API 调用上下文关系的方法.针对 JAVA 和 C#两种语言的 APIs 分别创建向量空间.最后,利用先前已有的 API 映射关系训练出向量空间转换模型,得到两种语言的 API 映射关系.结果显示:API2API 前 5 项推荐的准确率达到 77.9%,但 API2API 主要针对“1 to 1”的映射关系,并不能处理“1 to *n*”的映射情况.

综合以上研究,结合表 5 可以看到:自 2006 年,Xie 和 Pei[81]最先提出了挖掘 API 调用序列的算法 MAPO 以后,API 推荐便一直是研究的热点问题,绝大部分论文(8 篇)集中在方法推荐方面.2012 年,研究开始向类库推荐、参数推荐等方面扩展,以增强对开发人员的支持.

5 总结与展望

随着软件库不断发展,可使用的 API 数量急剧增长.伴随着 API 更新速度快、维护成本高等原因,形成了 API 文档质量低、调用规约不明确等问题,严重影响了工作效率,甚至会使程序出现严重的安全问题.API 相关领域问题成为了研究的热点问题,并日益得到了学术界和工业界的广泛关注.本文识别出影响 API 使用的 3 个关键问题:API 文档质量不高、调用规约不完整和 API 调用序列难以确定,并从 API 文档、调用规约以及 API 推荐这 3 个主要方面出发对研究成果进行了全面的总结和分析,仍有一些问题值得进一步深入研究.

5.1 API文档

软件生产商撰写校对文档需要大量的时间和投入,很难及时维护.与此同时,撰写文档的人员很多时候并不参与实际的项目开发,对很多细节并不真正了解.这样的矛盾使 API 文档质量难以保证.

(1) 多技术融合解决现有矛盾.如果能够从程序自动提取关于功能、规约的描述,将极大减少维护文档的工作量,人工智能技术在这个方面开始了初步尝试[138];

(2) API 参考文档没有形成统一的标准.例如 JDK 文档中,43%的内容对于开发人员并没有太高的使用价

值[11].如何自动化提取并推荐有效信息,值得进一步研究;

(3) API 教程、StackOverFlow 社区等资源发挥越来越重要的作用,但还没有研究评估这些资源的质量.

5.2 API调用规约

编写维护更新 API 调用规约代价非常高,官方很少会提供已编写好的 API 调用规约.现有研究虽然采用了很多方法推断 API 调用规约,但仍旧存在以下问题.

(1) 现有工作主要针对 API 调用规约的某些模式进行了推断,这样得到的结果只体现了规约的一个方面,很难完整概括 API 调用规约,并没有形成统一的标准;

(2) 目前,还没有方法能够客观评价调用规约的价值.挖掘到的调用规约的价值在于增加未了解的知识和经验,目前主要通过人工确认的方式,无法自动化评价这些结果;

(3) API 调用规约目前主要应用于缺陷检测领域,在增强文档、测试等领域应用还不广泛,这些应用对 API 调用规约的表示以及准确度等方面提出了更高的要求.

5.3 API推荐

API 推荐旨在帮助开发人员更加便捷地完成复杂的任务.如何优化 API 推荐值得继续关注.

(1) 现有 IDE 集成的 API 推荐工具并不完善.例如,输入 JDK 8 中的 *String* 类,现有推荐工具将会列出 67 个可能的方法[105],开发人员在选择合适的方法的时候仍然面临困难;

(2) 如果能够建立以社区为基础的 API 使用模式库,能够很好地实现 API 使用模式资源共享以及标准化,同时促进 API 相关领域的研究.

本文从 API 的定义、性质以及主要问题出发,对 API 文档、API 调用规约和 API 推荐等 3 个重点领域中经典的技术和方法进行了详细的综述,试图为该研究方向勾画出一个较为全面和清晰的概貌,为 API 领域的研究人员提供有益的参考.

References:

[1] Ko AJ, Myers BA, Coblenz MJ, Aung HH. An exploratory study of how developers seek, relate, and collect relevant information during software maintenance tasks. IEEE Trans. on Software Engineering, 2006,32(12):971−987. [doi: 10.1109/tse.2006.116]

[2] Nadi S, Kr S, Mezini M, Bodden E. Jumping through hoops: Why do Java developers struggle with cryptography APIs? In: Proc. of the 38th Int'l Conf. on Software Engineering. New York: ACM Press, 2016. 935−946. [doi: 10.1145/2884781.2884790]

[3] Robillard MP, Deline R. A field study of API learning obstacles. Empirical Software Engineering, 2011,16(6):703−732. [doi: 10.1007/ s10664-010-9150-8]

[4] Ko AJ, Myers BA, Aung HH. Six learning barriers in end-user programming systems. In: Proc. of the 2004 IEEE Symp. on Visual Languages—Human Centric Computing. Washington: IEEE Computer Society, 2004. 199−206. [doi: 10.1109/vlhcc.2004.47]

[5] Lazar D, Chen H, Wang X, Zeldovich N. Why does cryptographic software fail? A case study and open problems. In: Proc. of 5th Asia-Pacific Workshop on Systems. New York: ACM Press, 2014. 1−7. [doi: 10.1145/2637166.2637237]

[6] Egele M, Brumley D, Fratantonio Y, Kruegel C. An empirical study of cryptographic misuse in android applications. In: Proc. of the 2013 ACM SIGSAC Conf. on Computer & Communications Security. New York: ACM Press, 2013. 73−84. [doi: 10.1145/2508859.2516693]

[7] Georgiev M, Iyengar S, Jana S, Anubhai R, Boneh D, Shmatikov V. The most dangerous code in the world: Validating SSL certificates in non-browser software. In: Proc. of the 2012 ACM Conf. on Computer and Communications Security. New York: ACM Press, 2012. 38−49. [doi: 10.1145/2382196.2382204]

[8] Fahl S, Harbach M, Perl H, Koetter M, Smith M. Rethinking SSL development in an appified world. In: Proc. of the 2013 ACM SIGSAC Conf. on Computer & Communications Security. New York: ACM Press, 2013. 49−60. [doi: 10.1145/2508859.2516655]

[9] Parnas DL, Madey J, Iglewski M. Precise documentation of well-structured programs. IEEE Trans. on Software Engineering, 1994, 20(12): 948−976. [doi: 10.1109/32.368133]

[10] Saied MA, Sahraoui H, Dufour B. An observational study on API usage constraints and their documentation. In: Proc. of the 2015 IEEE 22nd Int'l Conf. on Software Analysis, Evolution and Reengineering (SANER). IEEE, 2015. 33−42. [doi: 10.1109/SANER.2015.7081813]

[11] Maalej W, Robillard MP. Patterns of knowledge in API reference documentation. IEEE Trans. on Software Engineering, 2013, 39(9):1264−1282. [doi: 10.1109/tse.2013.12]

[12] Sacramento P, Cabral B, Marques P. Unchecked exceptions: Can the programmer be trusted to document exceptions. In: Proc. of the 2nd Int'l Conf. on Innovative Views of .NET Technologies. 2006.

[13] Rubio-Gonz C, Liblit B. Expect the unexpected: Error code mismatches between documentation and the real world. In: Proc. of the 9th ACM SIGPLAN-SIGSOFT Workshop on Program Analysis for Software Tools and Engineering, Vol.1806687. ACM Press, 2010. 73−80. [doi: 10.1145/1806672.1806687]

[14] Kim J, Lee S, Hwang SW, Kim S. Enriching documents with examples: A corpus mining approach. ACM Trans. on Information Systems, 2013, 31(1):1−27. [doi: 10.1145/2414782.2414783]

[15] Subramanian S, Inozemtseva L, Holmes R. Live API documentation. In: Proc. of the 36th Int'l Conf. on Software Engineering. New York: ACM Press, 2014. 643−652. [doi: 10.1145/2568225.2568313]

[16] Treude C, Robillard MP. Augmenting API documentation with insights from stack overflow. In: Proc. of the 38th Int'l Conf. on Software Engineering. New York: ACM Press, 2016. 392−403. [doi: 10.1145/2884781.2884800]

[17] Chen C, Zhang K. Who asked what: Integrating crowdsourced FAQs into API documentation. In: Proc. of the 36th Int'l Conf. on Software Engineering. New York: ACM Press, 2014. 456−459. [doi: 10.1145/2591062.2591128]

[18] Zhou J, Walker RJ. API deprecation: A retrospective analysis and detection method for code examples on the Web. In: Proc. of the 2016 24th ACM SIGSOFT Int'l Symp. on Foundations of Software Engineering. New York: ACM Press, 2016. 266−277. [doi: 10.1145/2950290.2950298]

[19] Zhou Y, Gu R, Chen T, Huang Z, Panichella S, Gall H. Analyzing APIs documentation and code to detect directive defects. In: Proc. of the 39th Int'l Conf. on Software Engineering. Piscataway: IEEE Press, 2017. 27−37. [doi: 10.1109/icse.2017.11]

[20] Zhong H, Su Z. Detecting API documentation errors. ACM SIGPLAN Notices, 2013,48(10):803−816. [doi: 10.1145/2544173.2509523]

[21] Dekel U, Herbsleb JD. Improving API documentation usability with knowledge pushing. In: Proc. of the 31st Int'l Conf. on Software Engineering. Washington: IEEE Computer Society, 2009. 320−330. [doi: 10.1109/icse.2009.5070532]

[22] Monperrus M, Eichberg M, Tekes E, Mezini M. What should developers be aware of? An empirical study on the directives of API documentation. Empirical Software Engineering, 2012,17(6):703−737. [doi: 10.1007/s10664-011-9186-4]

[23] Petrosyan G, Robillard MP, Mori RD. Discovering information explaining API types using text classification. In: Proc. of the 37th Int'l Conf. on Software Engineering—Vol.1. Piscataway: IEEE Press, 2015. 869−879.

[24] Jiang H, Zhang J, Ren Z, Zhang T. An unsupervised approach for discovering relevant tutorial fragments for APIs. In: Proc. of the 39th Int'l Conf. on Software Engineering. Piscataway: IEEE Press, 2017. 38−48. [doi: 10.1109/icse.2017.12]

[25] Parnas DL. Precise documentation: The key to better software. In: Proc. of the Future of Software Engineering. Berlin: Springer-Verlag, 2011. 125−148.

[26] Bajracharya S, Lopes C. Mining search topics from a code search engine usage log. In: Proc. of the 2009 6th IEEE Int'l Working Conf. on Mining Software Repositories. Washington: IEEE Computer Society, 2009. 111−120. [doi: 10.1109/msr.2009.5069489]

[27] Dagenais B, Robillard MP. Creating and evolving developer documentation: Understanding the decisions of open source contributors. In: Proc. of the 18th ACM SIGSOFT Int'l Symp. on Foundations of Software Engineering. New York: ACM Press, 2010. 127−136. [doi: 10.1145/1882291.1882312]

[28] Dagenais B, Robillard MP. Recovering traceability links between an API and its learning resources. In: Proc. of the 34th Int'l Conf. on Software Engineering. Piscataway: IEEE Press, 2012. 47−57.

[29] Rigby PC, Robillard MP. Discovering essential code elements in informal documentation. In: Proc. of the 2013 Int'l Conf. on Software Engineering. Piscataway: IEEE Press, 2013. 832−841.

[30] Antoniol G, Canfora G, Casazza G, Lucia AD, Merlo E. Recovering traceability links between code and documentation. IEEE Trans. on Software Engineering, 2002,28(10):970−983. [doi: 10.1109/tse.2002.1041053]

[31] Duala-Ekoko E, Robillard MP. Asking and answering questions about unfamiliar APIs: An exploratory study. In: Proc. of the 34th Int'l Conf. on Software Engineering. Piscataway: IEEE Press, 2012. 266−276.

[32] Shi L, Zhong H, Xie T, Li M. An empirical study on evolution of API documentation. In: Proc. of the 14th Int'l Conf. on Fundamental Approaches to Software Engineering: Part of the Joint European Conf. on Theory and Practice of Software. Berlin: Springer-Verlag, 2011. 416−431.

[33] Parnin C, Treude C. Measuring API documentation on the Web. In: Proc. of the 2nd Int'l Workshop on Web 2.0 for Software Engineering. New York: ACM Press, 2011. 25−30. [doi: 10.1145/1984701.1984706]

[34] Cook JE, Wolf AL. Discovering models of software processes from event-based data. ACM Trans. on Software Engineering and Methodology, 1998,7(3):215−249. [doi: 10.1145/287000.287001]

[35] Dallmeier V, Lindig C, Wasylkowski A, Zeller A. Mining object behavior with ADABU. In: Proc. of the 2006 Int'l Workshop on Dynamic Systems Analysis. New York: ACM Press, 2006. 17−24. [doi: 10.1145/1138912.1138918]

[36] Pradel M, Gross TR. Automatic generation of object usage specifications from large method traces. In: Proc. of the 2009 IEEE/ACM Int'l Conf. on Automated Software Engineering. Washington: IEEE Computer Society, 2009. 371−382. [doi: 10.1109/ase.2009.60]

[37] Li Z, Zhou Y. PR-Miner: Automatically extracting implicit programming rules and detecting violations in large software code. SIGSOFT Software Engineering Notes, 2005,30(5):306−315. [doi: 10.1145/1095430.1081755]

[38] Livshits B, Zimmermann T. DynaMine: Finding common error patterns by mining software revision histories. SIGSOFT Software Engineering Notes, 2005,30(5):296−305. [doi: 10.1145/1095430.1081754]

[39] Yang J, Evans D, Bhardwaj D, Bhat T, Das M. Perracotta: Mining temporal API rules from imperfect traces. In: Proc. of the 28th Int'l Conf. on Software Engineering. New York: ACM Press, 2006. 282−291. [doi: 10.1145/1134285.1134325]

[40] Thummalapenta S, Xie T. Alattin: Mining alternative patterns for defect detection. Automated Software Engineering, 2011,18(3): 293−323. [doi: 10.1007/s10515-011-0086-z]

[41] Liang B, Bian P, Zhang Y, Shi W, You W, Cai Y. AntMiner: Mining more bugs by reducing noise interference. In: Proc. of the 38th Int'l Conf. on Software Engineering. New York: ACM Press, 2016. 333−344. [doi: 10.1145/2884781.2884870]

[42] Murali V, Chaudhuri S, Jermaine C. Bayesian specification learning for finding API usage errors. In: Proc. of the 2017 11th Joint Meeting on Foundations of Software Engineering. New York: ACM Press, 2017. 151−162. [doi: 10.1145/3106237.3106284]

[43] Gabel M, Su Z. Symbolic mining of temporal specifications. In: Proc. of the 30th Int'l Conf. on Software Engineering. New York: ACM Press, 2008. 51−60. [doi: 10.1145/1368088.1368096]

[44] Gabel M, Su Z. Javert: fully automatic mining of general temporal properties from dynamic traces. In: Proc. of the 16th ACM SIGSOFT Int'l Symp. on Foundations of Software Engineering. New York: ACM Press, 2008. 339−349. [doi: 10.1145/1453101.1453150]

[45] Gabel M, Su Z. Online inference and enforcement of temporal properties. In: Proc. of the 32nd ACM/IEEE Int'l Conf. on Software Engineering—Vol.1. New York: ACM Press, 2010. 15−24. [doi: 10.1145/1806799.1806806]

[46] Zhong H, Zhang L, Xie T, Mei H. Inferring resource specifications from natural language API documentation. In: Proc. of the 2009 IEEE/ACM Int'l Conf. on Automated Software Engineering. Washington: IEEE Computer Society, 2009. 307−318. [doi: 10.1109/ase.2009.94]

[47] Nguyen TT, Nguyen HA, Pham NH, Al-Kofahi JM, Nguyen TN. Graph-Based mining of multiple object usage patterns. In: Proc. of the the 7th Joint Meeting of the European Software Engineering Conf. and the ACM SIGSOFT Symp. on The Foundations of Software Engineering. New York: ACM Press, 2009. 383−392. [doi: 10.1145/1595696.1595767]

[48] Wasylkowski A, Zeller A. Mining temporal specifications from object usage. In: Proc. of the 2009 IEEE/ACM Int'l Conf. on Automated Software Engineering. Washington: IEEE Computer Society, 2009. 295−306. [doi: 10.1109/ase.2009.30]

[49] Wei Y, Furia CA, Kazmin N, Meyer B. Inferring better contracts. In: Proc. of the 33rd Int'l Conf. on Software Engineering. New York: ACM Press, 2011. 191−200. [doi: 10.1145/1985793.1985820]

[50] Nguyen HA, Dyer R, Nguyen TN, Rajan H. Mining preconditions of APIs in large-scale code corpus. In: Proc. of the 22nd ACM SIGSOFT Int'l Symp. on Foundations of Software Engineering. New York: ACM Press, 2014. 166−177. [doi: 10.1145/2635868.2635924]

[51] Henkel J, Reichenbach C, Diwan A. Discovering documentation for Java container classes. IEEE Trans. on Software Engineering, 2007,33(8):526−543. [doi: 10.1109/tse.2007.70705].

[52] Ramanathan MK, Grama A, Jagannathan S. Path-Sensitive inference of function precedence protocols. In: Proc. of the 29th Int'l Conf. on Software Engineering. Washington: IEEE Computer Society, 2007. 240−250. [doi: 10.1109/icse.2007.63]

[53] Lorenzoli D, Mariani L, Pezzè M. Automatic generation of software behavioral models. In: Proc. of the 30th Int'l Conf. on Software engineering. New York: ACM Press, 2008. 501−510. [doi: 10.1145/1368088.1368157]

[54] Krka I, Brun Y, Medvidovic N. Automatic mining of specifications from invocation traces and method invariants. In: Proc. of the 22nd ACM SIGSOFT Int'l Symp. on Foundations of Software Engineering. New York: ACM Press, 2014. 178−189. [doi: 10.1145/2635868.2635890]

[55] Dwyer MB, Avrunin GS, Corbett JC. Patterns in property specifications for finite-state verification. In: Proc. of the 21st Int'l Conf. on Software Engineering. New York: ACM Press, 1999. 411−420. [doi: 10.1145/302405.302672]

[56] Robillard MP, Bodden E, Kawrykow D, Mezini M, Ratchford T. Automated API property inference techniques. IEEE Trans. on Software Engineering, 2013,39(5):613−637. [doi: 10.1109/tse.2012.63]

[57] Engler D, Chen DY, Hallem S, Chou A, Chelf B. Bugs as deviant behavior: A general approach to inferring errors in systems code. SIGOPS Operating Systems Review, 2001,35(5):57−72. [doi: 10.1145/502059.502041]

[58] Qu JF, Liu M. Mining frequent itemsets using node-sets of a prefix-tree. In: Proc. of the 23rd Int'l Conf. on Database and Expert Systems Applications (DEXA 2012). Part I. Vienna, Berlin, Heidelberg: Springer-Verlag, 2012. 453−467. [doi: 10.1007/978-3-642-32600-4_34]

[59] Agrawal R, Srikant R. Fast algorithms for mining association rules in large databases. In: Proc. of the 20th Int'l Conf. on Very Large Data Bases. San Francisco: Morgan Kaufmann Publishers Inc., 1994. 487−499.

[60] Kagdi H, Collard ML, Maletic JI. Comparing approaches to mining source code for call-usage patterns. In: Proc. of the 4th Int'l Workshop on Mining Software Repositories. Washington: IEEE Computer Society, 2007. 20. [doi: 10.1109/msr.2007.3]

[61] Kagdi H, Collard ML, Maletic JI. An approach to mining call-usage patternswith syntactic context. In: Proc. of the 22nd IEEE/ACM Int'l Conf. on Automated Software Engineering. New York: ACM Press, 2007. 457−460. [doi: 10.1145/1321631.1321708]

[62] Thummalapenta S, Xie T. Mining exception-handling rules as sequence association rules. In: Proc. of the 31st Int'l Conf. on Software Engineering. Washington: IEEE Computer Society, 2009. 496−506. [doi: 10.1109/icse.2009.5070548]

[63] Lo D, Khoo SC. SMArTIC: Towards building an accurate, robust and scalable specification miner. In: Proc. of the 14th ACM SIGSOFT Int'l Symp. on Foundations of Software Engineering. New York: ACM Press, 2006. 265−275. [doi: 10.1145/1181775.1181808]

[64] Lorenzoli D, Mariani L, Pezz M, #232. Automatic generation of software behavioral models. In: Proc. of the 30th Int'l Conf. on Software Engineering. New York: ACM Press, 2008. 501−510. [doi: 10.1145/1368088.1368157]

[65] Walkinshaw N, Bogdanov K. Inferring finite-state models with temporal constraints. In: Proc. of the 2008 23rd IEEE/ACM Int'l Conf. on Automated Software Engineering. Washington: IEEE Computer Society, 2008. 248−257. [doi: 10.1109/ase.2008.35]

[66] Ammons G, Bod R, #237, Larus JR. Mining specifications. SIGPLAN Notices, 2002,37(1):4−16. [doi: 10.1145/565816.503275]

[67] Chang RY, Podgurski A, Yang J. Finding what's not there: A new approach to revealing neglected conditions in software. In: Proc. of the 2007 Int'l Symp. on Software Testing and Analysis. New York: ACM Press, 2007. 163−173. [doi: 10.1145/1273463.1273486]

[68] Zhong H, Zhang L, Mei H. Inferring specifications of object oriented APIs from API source code. In: Proc. of the 15th Asia-Pacific Software Engineering Conf. (APSEC 2008). IEEE, 2008. 221−228. [doi: 10.1109/APSEC.2008.54]

[69] Meyer B. Applying "Design by Contract". Computer, 1992,25(10):40−51. [doi: 10.1109/2.161279]

[70] Ernst MD, Cockrell J, Griswold WG, Notkin D. Dynamically discovering likely program invariants to support program evolution. In: Proc. of the 21st Int'l Conf. on Software Engineering. New York: ACM Press, 1999. 213−224. [doi: 10.1145/302405.302467]

[71] Csallner C, Tillmann N, Smaragdakis Y. DySy: Dynamic symbolic execution for invariant inference. In: Proc. of the 30th Int'l Conf. on Software Engineering. New York: ACM Press, 2008. 281−290. [doi: 10.1145/1368088.1368127]

[72] Ernst MD, Perkins JH, Guo PJ, McCamant S, Pacheco C, Tschantz MS, Xiao C. The Daikon system for dynamic detection of likely invariants. Science of Computer Programming, 2007,69(1-3):35−45.[doi: 10.1016/j.scico.2007.01.015]

[73] Flanagan C, Leino KRM. Houdini, an annotation assistant for ESC/Java. In: Proc. of the Int'l Symp. of Formal Methods Europe on Formal Methods for Increasing Software Productivity. London: Springer-Verlag, 2001. 500−517.

[74] Guttag JV, Horning JJ. The algebraic specification of abstract data types. Acta informatica, 1978,10(1):27−52. [doi: 10.1007/bf00260922]

[75] Bagge AH, Haveraaen M. Specification of generic APIs, or: Why algebraic may be better than pre/post. Ada Letters, 2014,34(3):71−80. [doi: 10.1145/2692956.2663183]

[76] Legunsen O, Hassan WU, Xu X, Roşu G, Marinov D. How good are the specs? A study of the bug-finding effectiveness of existing Java API specifications. In: Proc. of the 31st IEEE/ACM Int'l Conf. on Automated Software Engineering. New York: ACM Press, 2016. 602−613. [doi: 10.1145/2970276.2970356]

[77] Thung F. API recommendation system for software development. In: Proc. of the 31st IEEE/ACM Int'l Conf. on Automated Software Engineering. New York: ACM Press, 2016. 896−899. [doi: 10.1145/2970276.2975940]

[78] Teyton C, Falleri JR, Blanc X. Mining library migration graphs. In: Proc. of the 2012 19th Working Conf. on Reverse Engineering. Washington: IEEE Computer Society, 2012. 289−298. [doi: 10.1109/wcre.2012.38]

[79] Thung F, Lo D, Lawall J. Automated library recommendation. In: Proc. of the 2013 20th Working Conf. on Reverse Engineering (WCRE). 2013. 182−191.

[80] Chen C, Xing Z. SimilarTech: Automatically recommend analogical libraries across different programming languages. In: Proc. of the 31st IEEE/ACM Int'l Conf. on Automated Software Engineering. New York: ACM Press, 2016. 834−839. [doi: 10.1145/2970276.2970290]

[81] Xie T, Pei J. MAPO: Mining API usages from open source repositories. In: Proc. of the 2006 Int'l Workshop on Mining Software Repositories. New York: ACM Press, 2006. 54−57. [doi: 10.1145/1137983.1137997]

[82] Hindle A, Barr ET, Su Z, Gabel M, Devanbu P. On the naturalness of software. In: Proc. of the 34th Int'l Conf. on Software Engineering. Piscataway: IEEE Press, 2012. 837−847.

[83] Wang J, Dang Y, Zhang H, Chen K, Xie T, Zhang D. Mining succinct and high-coverage API usage patterns from source code. In: Proc. of the 10th Working Conf. on Mining Software Repositories. Piscataway: IEEE Press, 2013. 319−328.

[84] Raychev V, Vechev M, Yahav E. Code completion with statistical language models. SIGPLAN Notices, 2014,49(6):419−428. [doi: 10.1145/2666356.2594321]

[85] Nguyen AT, Nguyen TN. Graph-Based statistical language model for code. In: Proc. of the 37th Int'l Conf. on Software Engineering—Vol.1. Piscataway: IEEE Press, 2015. 858−868.

[86] Nguyen TT, Pham HV, Vu PM, Nguyen TT. Learning API usages from bytecode: A statistical approach. In: Proc. of the 38th Int'l Conf. on Software Engineering. New York: ACM Press, 2016. 416−427. [doi: 10.1145/2884781.2884873]

[87] Fowkes J, Sutton C. Parameter-Free probabilistic API mining across GitHub. In: Proc. of the 2016 24th ACM SIGSOFT Int'l Symp. on Foundations of Software Engineering. New York: ACM Press, 2016. 254−265. [doi: 10.1145/2950290.2950319]

[88] Gu X, Zhang H, Zhang D, Kim S. Deep API learning. In: Proc. of the 2016 24th ACM SIGSOFT Int'l Symp. on Foundations of Software Engineering. New York: ACM Press, 2016. 631−642. [doi: 10.1145/2950290.2950334]

[89] Zhang C, Yang J, Zhang Y, Fan J, Zhang X, Zhao J, Ou P. Automatic parameter recommendation for practical API usage. In: Proc. of the 34th Int'l Conf. on Software Engineering. Piscataway: IEEE Press, 2012. 826−836.

[90] Asaduzzaman M, Roy CK, Monir S, Schneider KA. Exploring API method parameter recommendations. In: Proc. of the 2015 IEEE Int'l Conf. on Software Maintenance and Evolution (ICSME). Washington: IEEE Computer Society, 2015. 271−280. [doi: 10.1109/icsm.2015.7332473]

[91] Dig D, Johnson R. How do APIs evolve? a story of refactoring: Research articles. Journal of Software Maintenance and Evolution, 2006,18(2): 83−107. [doi: 10.1002/smr.v18:2]

[92] Dagenais B, Robillard MP. Recommending adaptive changes for framework evolution. ACM Transactions on Software Engineering and Methodology, 2011,20(4):1−35. [doi: 10.1145/2000799.2000805]

[93] Zhong H, Thummalapenta S, Xie T, Zhang L, Wang Q. Mining API mapping for language migration. In: Proc. of the 32nd ACM/IEEE Int'l Conf. on Software Engineering—Vol.1. New York: ACM Press, 2010. 195−204. [doi: 10.1145/1806799.1806831]

[94] Nguyen TD, Nguyen AT, Phan HD, Nguyen TN. Exploring API embedding for API usages and applications. In: Proc. of the 39th Int'l Conf. on Software Engineering. Piscataway: IEEE Press, 2017. 438−449. [doi: 10.1109/icse.2017.47]

[95] Wang W, Godfrey MW. Detecting API usage obstacles: A study of iOS and Android developer questions. In: Proc. of the 10th Working Conf. on Mining Software Repositories. Piscataway: IEEE Press, 2013. 61−64.

[96] Agrawal R, Srikant R. Mining sequential patterns. In: Proc. of the 11th Int'l Conf. on Data Engineering. Washington: IEEE Computer Society, 1995. 3−14.

[97] Srikant R, Agrawal R. Mining sequential patterns: Generalizations and performance improvements. In: Proc. of the 5th Int'l Conf. on Extending Database Technology: Advances in Database Technology. London: Springer-Verlag, 1996. 3−17.

[98] Pei J, Han J, Mortazavi-Asl B, Pinto H, Chen Q, Dayal U, Hsu MC. PrefixSpan: Mining sequential patterns efficiently by prefix-projected pattern growth. In: Proc. of the 17th Int'l Conf. on Data Engineering. IEEE, 2001. 215−224.

[99] Zaki MJ. SPADE: An efficient algorithm for mining frequent sequences. Machine Learning, 2001,42(1-2):31−60. [doi: 10.1023/a:1007652502315]

[100] Ayres J, Flannick J, Gehrke J, Yiu T. Sequential PAttern mining using a bitmap representation. In: Proc. of the 8th ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining. New York: ACM Press, 2002. 429−435. [doi: 10.1145/775047.775109]

[101] Wang J, Han J. BIDE: Efficient mining of frequent closed sequences. In: Proc. of the 20th Int'l Conf. on Data Engineering. Washington: IEEE Computer Society, 2004. 79.

[102] Tatti N, Vreeken J. The long and the short of it: Summarising event sequences with serial episodes. In: Proc. of the 18th ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining. New York: ACM Press, 2012. 462−470. [doi: 10.1145/2339530.2339606]

[103] Lam HT, Mörchen F, Fradkin D, Calders T. Mining compressing sequential patterns. Statistical Analysis and Data Mining: The ASA Data Science Journal, 2014,7(1):34−52.

[104] Rosenfeld R. Two decades of statistical language modeling: Where do we go from here? Proc. of the IEEE, 2000,88(8):1270−1278.

[105] Nguyen AT, Hilton M, Codoban M, Nguyen HA, Mast L, Rademacher E, Nguyen TN, Dig D. API code recommendation using statistical learning from fine-grained changes. In: Proc. of the 2016 24th ACM SIGSOFT Int'l Symp. on Foundations of Software Engineering. New York: ACM Press, 2016. 511−522. [doi: 10.1145/2950290.2950333]

[106] Elman JL. Finding structure in time. Cognitive Science, 1990,14(2):179−211.

[107] Mikolov T, Karafiát M, Burget L, Cernocký J, Khudanpur S. Recurrent neural network based language model. In: Proc. of the INTERSPEECH 2010, Conf. of the Int'l Speech Communication Association. 2010. 1045−1048.

[108] Cho K,Van Merriënboer B, Gulcehre C, Bahdanau D, Bougares F, Schwenk H, Bengio Y. Learning phrase representations using RNN encoder-decoder for statistical machine translation. In: Proc. of the EMNLP. 2014. 1724−1734.

[109] Pradel M, Heiniger S, Gross TR. Static detection of brittle parameter typing. In: Proc. of the 2012 Int'l Symp. on Software Testing and Analysis. New York: ACM Press, 2012. 265−275. [doi: 10.1145/2338965.2336785]

[110] Pradel M, Gross TR. Detecting anomalies in the order of equally-typed method arguments. In: Proc. of the 2011 Int'l Symp. on Software Testing and Analysis. New York: ACM Press, 2011. 232−242. [doi: 10.1145/2001420.2001448]

[111] Xing Z, Stroulia E. API-Evolution support with diff-CatchUp. IEEE Trans. on Software Engineering, 2007,33(12):818−836. [doi: 10.1109/ tse.2007.70747]

[112] Dig D, Comertoglu C, Marinov D, Johnson R. Automated detection of refactorings in evolving components. In: Proc. of the 20th European Conf. on Object-Oriented Programming. Berlin: Springer-Verlag, 2006. 404−428. [doi: 10.1007/11785477_24]

[113] Schäfer T, Jonas J, Mezini M. Mining framework usage changes from instantiation code. In: Proc. of the 30th Int'l Conf. on Software Engineering. New York: ACM Press, 2008. 471−480. [doi: 10.1145/1368088.1368153]

[114] Roover CD, Lammel R, Pek E. Multi-Dimensional exploration of API usage. In: Proc. of the IEEE Int'l Conf. on Program Comprehension. IEEE, 2013. 152−161. [doi: 10.1109/ICPC.2013.6613843]

[115] McIlroy MD. Mass-Produced software components. In: Proc. of the 1st Int'l Conf. on Software Engineering. 1968. 88−98.

[116] Parnas DL. On the criteria to be used in decomposing systems into modules. Communications of the ACM, 1972,15(12):1053−1058. [doi: 10.1145/361598.361623]

[117] Plauger PJ. The Standard C Library. Prentice Hall PTR, 1991. 17−415.

[118] Tsai CC, Jain B, Abdul NA, Porter DE. A study of modern Linux API usage and compatibility: What to support when you're supporting. In: Proc. of the 11th European Conf. on Computer Systems. New York: ACM Press, 2016. 1−16. [doi: 10.1145/2901318.2901341]

[119] Atlidakis V, Andrus J, Geambasu R, Mitropoulos D, Nieh J. POSIX abstractions in modern operating systems: The old, the new, and the missing. In: Proc. of the 11th European Conf. on Computer Systems. New York: ACM Press, 2016. 1−17. [doi: 10.1145/2901318.2901350]

[120] Qiu D, Li B, Leung H. Understanding the API usage in Java. Information and Software Technology, 2016,73(C):81−100. [doi: 10.1016/j.infsof. 2016.01.011]

[121] Bissyandé TF, Réveillère L, Lawall JL, Muller G. Ahead of time static analysis for automatic generation of debugging interfaces to the Linux kernel. Automated Software Engineering, 2016,23(1):3−41. [doi: 10.1007/s10515-014-0152-4]

[122] Bissyandé TF, Réveillère L, Lawall JL, Muller G. Diagnosys: Automatic generation of a debugging interface to the Linux kernel. In: Proc. of the 27th IEEE/ACM Int'l Conf. on Automated Software Engineering. New York: ACM Press, 2012. 60−69. [doi: 10.1145/2351676.2351686]

[123] Jezek K, Dietrich J, Brada P. How Java APIs break—An empirical study. Information and Software Technology, 2015,65(C): 129−146. [doi: 10.1016/ j.infsof.2015.02.014]

[124] Lin Z, Zhong H, Chen Y, Zhao J. LockPeeker: Detecting latent locks in Java APIs. In: Proc. of the 31st IEEE/ACM Int'l Conf. on Automated Software Engineering. New York: ACM Press, 2016. 368−378. [doi: 10.1145/2970276.2970355]

[125] Businge J, Serebrenik A, van den Brand M. Analyzing the Eclipse API usage: Putting the developer in the loop. In: Proc. of the 2013 17th European Conf. on Software Maintenance and Reengineering (CSMR). IEEE, 2013. 37−46. [doi: 10.1109/CSMR.2013.14]

[126] Businge J, Serebrenik A, van den Brand MGJ. Eclipse API usage: The good and the bad. Software Quality Journal, 2015,23(1): 107−141. [doi: 10.1007/s11219-013-9221-3]

[127] Businge J, Brand MVD, Serebrenik A. Survival of Eclipse third-party plug-ins. In: Proc. of the 2012 IEEE Int'l Conf. on Software Maintenance (ICSM). Washington: IEEE Computer Society, 2012. 368−377. [doi: 10.1109/icsm.2012.6405295]

[128] McDonnell T, Ray B, Kim M. An empirical study of API stability and adoption in the android ecosystem. In: Proc. of the 2013 IEEE Int'l Conf. on Software Maintenance. Washington: IEEE Computer Society, 2013. 70−79. [doi: 10.1109/icsm.2013.18]

[129] Bavota G, Linares-Vasquez M, Bernal-Cardenas CE, Di Penta M, Oliveto R, Poshyvanyk D. The impact of API change-and fault-proneness on the user ratings of android apps. IEEE Trans. on Software Engineering, 2015,41(4):384−407. [doi: 10.1109/TSE.2014.2367027]

[130] Li L, Bissyandé TF, Le Traon Y, Klein J. Accessing inaccessible android APIs: An empirical study. In: Proc. of the 2016 IEEE Int'l Conf. on Software Maintenance and Evolution (ICSME). IEEE, 2016. 411−422.

[131] Han J, Kywe SM, Yan Q, Bao F, Deng R, Gao D, Li Y, Zhou J. Launching generic attacks on iOS with approved third-party applications. In: Proc. of the 11th Int'l Conf. on Applied Cryptography and Network Security. Berlin: Springer-Verlag, 2013. 272−289. [doi: 10.1007/978-3-642-38980-1_17]

[132] Wang T, Lu K, Lu L, Chung S, Lee W. Jekyll on iOS: When benign apps become evil. In: Proc. of the 22nd USENIX Conf. on Security. Berkeley: USENIX Association, 2013. 559−572.

[133] Deng Z, Saltaformaggio B, Zhang X, Xu D. iRiS: Vetting private API abuse in iOS applications. In: Proc. of the 22nd ACM SIGSAC Conf. on Computer and Communications Security. New York: ACM Press, 2015. 44−56. [doi: 10.1145/2810103.2813675]

[134] Zibran MF, Eishita FZ, Roy CK. Useful, but usable? Factors affecting the usability of APIs. In: Proc. of the 2011 18th Working Conf. on Reverse Engineering. Washington: IEEE Computer Society, 2011. 151−155. [doi: 10.1109/wcre.2011.26]

[135] Ellis B, Stylos J, Myers B. The factory pattern in API design: A usability evaluation. In: Proc. of the 29th Int'l Conf. on Software Engineering. Washington: IEEE Computer Society, 2007. 302−312. [doi: 10.1109/icse.2007.85]

[136] Stylos J, Myers BA. The implications of method placement on API learnability. In: Proc. of the 16th ACM SIGSOFT Int'l Symp. on Foundations of Software Engineering. New York: ACM Press, 2008. 105−112. [doi: 10.1145/1453101.1453117]

[137] Wu W, Guéhéneuc YG, Antoniol G, Kim M. AURA: A hybrid approach to identify framework evolution. In: Proc. of the 32nd ACM/IEEE Int'l Conf. on Software Engineering—Vol.1. New York: ACM Press, 2010. 325−334. [doi: 10.1145/1806799.1806848]

[138] Iyer S, Konstas I, Cheung A, Zettlemoyer L. Summarizing source code using a neural attention model. In: Proc. of the Meeting of the Association for Computational Linguistics. 2016. 2073−2083.

李正(1985－),男,天津人,博士生,主要研究领域为软件工程,系统安全.

吴敬征(1982－),男,博士,副研究员,主要研究领域为系统安全,漏洞挖掘,移动安全.

李明树(1965－),男,博士,研究员,博士生导师,CCF 会士,主要研究领域为操作系统深度设计,可信软件过程以及基础软硬件核心技术与应用.