软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2023,34(8):3605−3636 [doi: 10.13328/j.cnki.jos.006558]

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

用户特征请求分析与处理研究综述*

牛菲菲[1,2], 李传艺[1,2], 葛季栋[1,2], 骆 斌[1,2]

[1](计算机软件新技术国家重点实验室 (南京大学), 江苏 南京 210093)

[2](南京大学 软件学院, 江苏 南京 210093)

通信作者: 李传艺, E-mail: lcy@nju.edu.cn

摘 要: 特征请求是软件产品的真实用户在开放平台上提出的对现有特征的改进或者对新特征的请求. 特征请求在一定程度上反映了用户的真实意愿, 代表了用户的需求. 高效、准确地分析和处理用户特征请求对于提升用户满意度、提高产品竞争力起着至关重要的作用. 用户的广泛参与, 使得特征请求成为越来越重要的需求来源. 然而, 特征请求在其来源、内容以及形式等方面均与传统的软件需求不同. 进而将其充分应用于软件开发过程所采用的具体方法, 也有别于传统的需求工程. 目前已经有许多将特征请求应用于软件开发过程中的相关研究, 比如特征请求的获取、分类、排序、质量评估、为特征请求推荐开发者, 以及定位相关代码等. 随着相关工作的不断增加, 形成一个针对特征请求分析与处理研究综述的必要性日益增强. 因此, 调研 121 篇关于在软件开发过程中分析和处理特征请求的国内外学术研究论文, 从将特征请求应用于软件开发过程的角度对现有成果进行系统地梳理. 总结现有针对特征请求的研究主题, 提出将特征请求应用于软件开发过程的处理流程, 并与传统的需求工程过程进行对比. 此外, 深入分析在各个需求工程活动中使用的具体方法及方法之间的差别. 最后, 对特征请求的未来研究方向进行展望, 以期为同行研究人员提供参考.

关键词: 特征请求; 需求工程; 软件开发; 分析与处理

中图法分类号: TP311

中文引用格式: 牛菲菲, 李传艺, 葛季栋, 骆斌. 用户特征请求分析与处理研究综述. 软件学报, 2023, 34(8): 3605–3636. http://www.jos.org.cn/1000-9825/6558.htm

英文引用格式: Niu FF, Li CY, Ge JD, Luo B. Survey on User Feature Requests Analysis and Processing. Ruan Jian Xue Bao/Journal of Software, 2023, 34(8): 3605–3636 (in Chinese). http://www.jos.org.cn/1000-9825/6558.htm

Survey on User Feature Requests Analysis and Processing

NIU Fei-Fei[1,2], LI Chuan-Yi[1,2], GE Ji-Dong[1,2], LUO Bin[1,2]

[1](State Key Laboratory for Novel Software Technology (Nanjing University), Nanjing 210093, China)

[2](Software Institute, Nanjing University, Nanjing 210093, China)

Abstract: Feature requests refer to suggestions to perfect existing features or requests for new features proposed by software users on open platforms, and they can reflect users' wishes and needs. In addition, efficient and accurate analysis and processing of feature requests play a vital role in improving user satisfaction and product competitiveness. With users' active participation, feature requests have become an important source of software requirements. However, feature requests are different from traditional requirements in terms of source, content, and form. Therefore, methods of applying feature requests to software development must differ from that of traditional requirements. At present, massive research focuses on applying feature requests to software development, e.g., feature requests' acquisition, classification, prioritization, quality management, developer recommendation, and location of relevant codes. As related research emerges constantly, it is increasingly necessary to review user feature request analysis and processing. This study analyzes 121 global academic research papers on how to analyze and process feature requests in the software development process and systematically sorts existing

* 基金项目: 国家自然科学基金 (61802167); 南京大学计算机软件新技术国家重点实验室海外开放课题 (KFKT2020A05)
本文由"领域软件工程"专题特约编辑汤恩义副教授、江贺教授、陈俊洁副教授、李必信教授以及唐滨副教授推荐.
收稿时间: 2021-08-10; 修改时间: 2021-10-09; 采用时间: 2022-01-10; jos 在线出版时间: 2022-01-28
CNKI 网络首发时间: 2023-01-19

research results from the perspective of applying feature requests to software development. In addition, the study summarizes research topics on feature requests, suggests that feature requests be applied to software development, and makes a comparison with traditional requirements engineering processes. Furthermore, it analyzes existing research methods of different requirement engineering and points out the difference. Finally, the research direction of feature requests is discussed to provide guidance for future researchers.

Key words: feature request; requirements engineering; software development; analysis and processing

在开源软件仓库的公开论坛上, 用户可能以用户评论 (user review) 或问题报告 (issue report) 的形式, 针对软件提出缺陷报告 (bug report)、特征请求 (feature request) 等. 其中特征请求一般是请求新的特征或者对现有特征的改进. 近年来互联网迅速发展, 不仅各种类型软件的数量不断激增, 同时, 在应用商店或者开放平台 (如问题跟踪系统) 上, 也不断有用户针对已有软件提出新的请求. 例如, Apple AppStore 和 Google Play 自推出起, 两个应用商店平台已经积累了超过 100 万个可供下载和评论的应用程序[1], 伴随着这些应用程序, 也产生了大量的用户评论. 一方面, 这些用户请求体现了用户的真实意愿, 开发者满足用户的意见对于提升用户满意度、提高软件竞争力起着至关重要的作用. 另一方面, 特征请求与传统的软件需求在内容、形式等方面都有很大的不同, 传统的需求工程处理方法不一定适用于特征请求. 因此, 出现了许多针对如何将特征请求应用于软件开发和维护过程中的研究, 比如特征请求分类、优先级排序及管理等. 这些研究从不同的角度提出了不同的解决方法, 其目标都是将特征请求应用于软件开发和维护过程中, 以满足用户的请求进而提升用户满意度和产品竞争力.

目前, 已有学者发表了一些与特征请求相关研究综述. 例如, Cavalcanti 等人[2]在 2013 年对开源仓库 (如 Bugzilla、Mantis、Redmine 等) 的管理进行调研, 将变更请求 (change request) 的管理活动映射到不同的研究主题和领域, 并分别从机遇和挑战两个角度进行探讨. Tavakoli 等人[3]对从移动应用评论中提取软件开发的有用信息做了综述研究. 他们一共调研了 34 篇文章, 发现现有研究中主要采用监督学习、自然语言处理以及特征抽取等技术从用户评论中挖掘有用信息. 在移动应用开发中最经常使用的用户评论的主题包括缺陷报告和特征请求等. Wang 等人[4]对 44 篇将用户反馈应用于需求工程的研究做了综述研究. 他们将用户反馈的研究映射到需求工程领域阶段, 对显示反馈和隐式反馈均做了调研. Bakar 等人[5]对从自然语言需求描述中抽取软件特征的研究做了综述调研. 特征请求可能来源于用户反馈, 是软件特征的具体描述. 然而, 目前缺乏将特征请求应用于软件工程的系统调研, 尤其是从需求工程的视角, 包括其获取、分析、管理等, 以便于将其更好地应用于软件开发. 为系统地了解现有研究中是如何处理利用特征请求的, 本文对现有针对特征请求的研究进行梳理, 从特征请求的获取、分析、管理到开发等活动, 分类总结了相关研究工作. 并深入探究每一主题的研究进展, 展望未来研究方向, 以期为相关领域的研究人员提供参考.

传统的需求工程过程如图 1 所示, 包括需求获取、需求分析、需求规格化与验证, 以及需求管理等活动. 本文总结了现有针对特征请求的研究主题, 并与传统需求工程活动对应. 如图 2 所示, 这些研究主题主要包括: 特征请求获取、软件特征抽取、分析理解、优先级排序、特征请求评审、特征请求跟踪以及变更管理. 由于针对特征请求的规格化与验证相关研究较少, 因此本文主要讨论需求获取、需求分析与需求管理相关的研究活动.

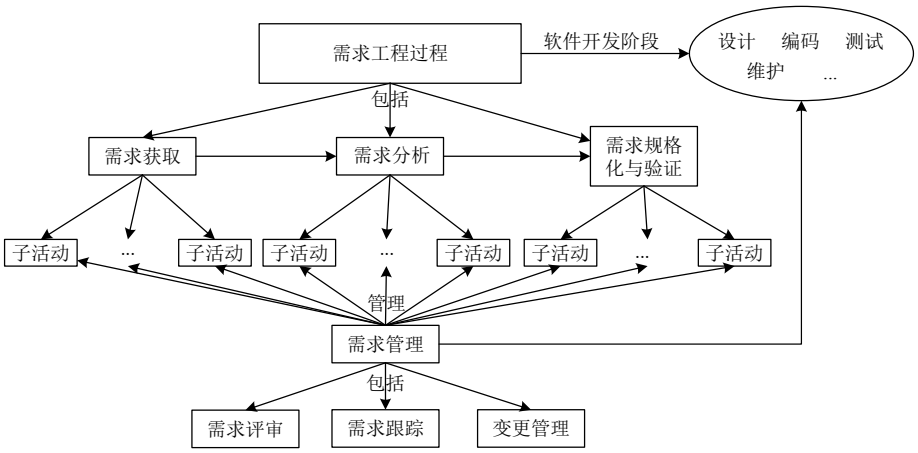

图 1 传统需求工程过程

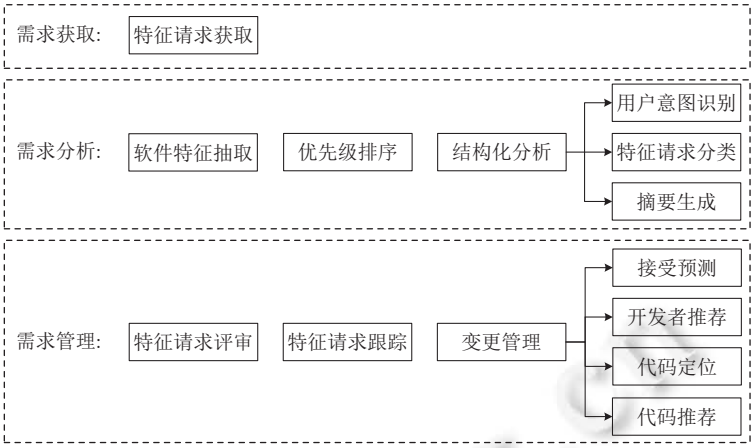

图 2 特征请求研究主题

本文第 1 节详细介绍文献检索与选择的过程. 第 2 节对特征请求相关的概念进行分析、总结. 第 3–5 节分别介绍在传统需求工程活动框架下, 特征请求相应的研究进展. 具体而言, 各节分别从需求获取、需求分析及需求管理的角度对现有针对特征请求的研究进行梳理与总结. 第 6 节汇总现有研究中关于特征请求的公开的工具以及数据集, 以共享研究资源. 第 7 节总结已有研究, 并探究特征请求的研究挑战与机遇. 最后总结全文, 并展望未来研究方向.

1 文献检索与选择

为了对已有关于特征请求的研究进行系统的分析, 本研究采用自动检索和滚雪球[6]相结合的方式检索相关文献. 具体检索步骤如下.

步骤 1. 以“Feature Request”和“特征请求”为关键词, 在与软件工程社区最相关的、广泛使用的文献检索库中检索相关论文: IEEE Xplore、ACM Digital Library、Science Direct、Springer Link、Wiley InterScience、Elsevier、Google Scholar 以及中国知网. 资源库的链接如表 1 所示.

表 1 文献资源库列表

资源库名称	链接
IEEE Xplore	http://www.ieee.org/web/publications/xplore/
ACM Digital Library	http://portal.acm.org
Science Direct	http://www.sciencedirect.com/
Springer Link	https://link.springer.com/
Wiley InterScience	https://onlinelibrary.wiley.com/
Elsevier	http://www.elsevier.com
Google Scholar	https://scholar.google.com
中国知网	https://www.cnki.net/

步骤 2. 根据表 2 所示筛选标准筛选步骤 1 得到的结果, 保留符合本综述研究主题的论文. 最终得到 23 篇初始文献.

步骤 3. 采用滚雪球的方法, 分别向前检索初始文献的引用文献, 向后检索引用初始文献的文献, 根据表 2 所示筛选标准保留符合特征请求主题的文献.

步骤 4. 针对步骤 3 中获取的文献, 重复步骤 3, 直到文献集合收敛, 不再有新的论文加入.

经过上述 4 个步骤, 最终的研究论文集合包括 121 篇文章 (截至 2021 年 5 月), 其中包括 15 篇工具类论文.

图 3 统计了不同年份论文的发表数量. 自 2010 年起针对特征请求的研究逐年增加. 图 4 与图 5 分别统计了论文收录的期刊、会议分布情况以及高频的会议、期刊. 本文调研文献有 82 篇收录于会议, 39 篇收录于期刊. 其中 CCF-A 类期刊与会议一共 18 篇, CCF-B 类期刊与会议一共 49 篇, CCF-C 类期刊与会议一共 17 篇. 大部分收录于软件工程领域的权威会议或期刊, 包括 RE 会议 (18 篇)、ICSE 会议 (10 篇)、REFSQ 会议 (7 篇)、ASE 会议 (6 篇)、JSS 期刊 (6 篇)、JSEP 期刊 (5 篇)、RE 期刊 (4 篇)、EASE 会议 (4 篇)、ESE 期刊 (4 篇) 等. 从统计结果可以看出关于特征请求的研究主要发表于软件工程主题相关的会议和期刊, 尤其是需求工程. 图 6 统计了主要的研究主题分布. 现有的针对特征请求的研究主要集中于需求获取与需求管理, 分别占所有研究的 41% 和 36%. 此外, 需求分析占 20%.

表 2 文献过滤标准

过滤类型	过滤标准
不包含	文献语言为中文和英文之外的其他语言
	主要研究对象为缺陷报告
	博士生或者硕士生毕业论文
	不是完整的研究, 研究方法、研究结果、结论等必备因素缺失
	网上无法下载完整的文章内容
包含	文章标题或者内容包含“特征请求”“feature request”关键词
	文章已发表, 发表在国内外的会议或者期刊上
	文章的主要研究对象是特征请求或者包含特征请求的概念, 如用户反馈等
	文章的主要研究目标是面向软件工程的特征请求研究

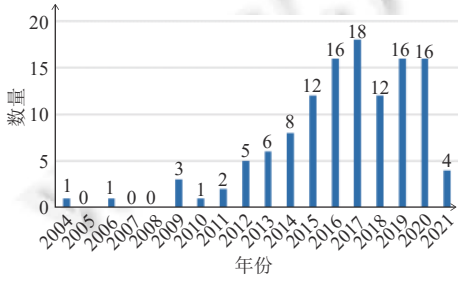

图 3 论文在不同年份的文献数量分布

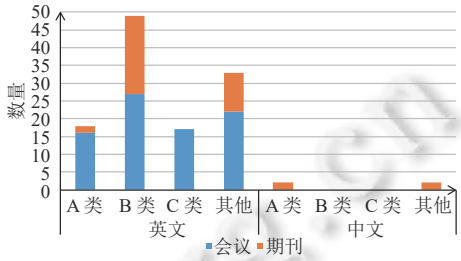

图 4 论文收录会议、期刊等级分布情况

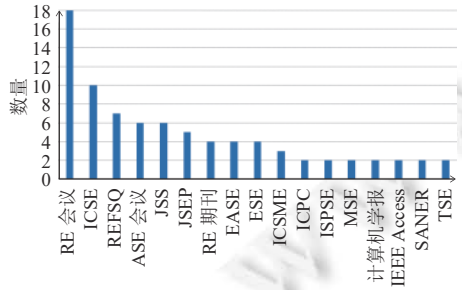

图 5 论文收录高频会议和期刊

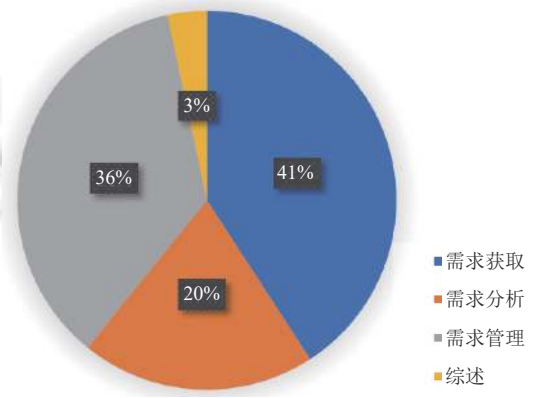

图 6 研究主题分布

2 特征请求相关概念

为系统地了解特征请求及其相关概念, 通过总结相关研究论文, 汇总了与特征请求相关的术语, 不同术语之间的关系如图 7 所示.

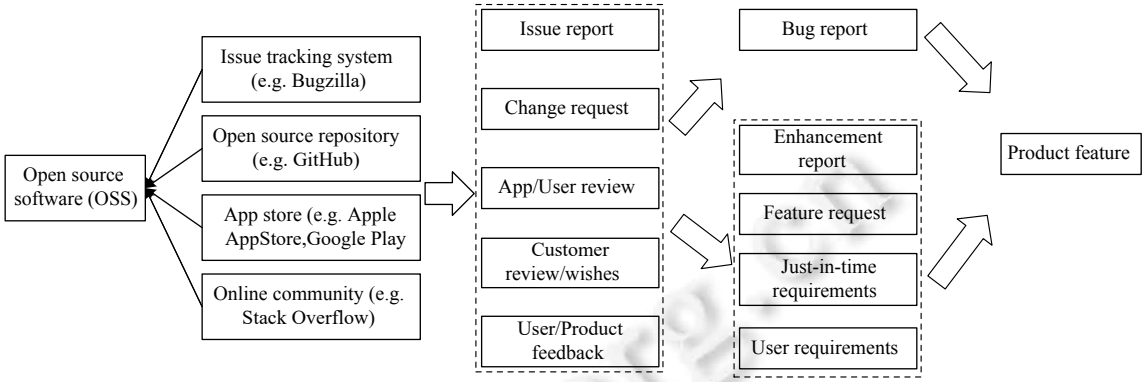

图 7 特征请求相关概念

Herzig 等人[7]将特征请求定义为: 特征或特征请求是一种应用于软件维护的, 实现了某种新功能的结构化请求, 其包括标题, 描述等许多属性. 特征请求一般是开源软件 (open source software) 的真实用户在开源用户平台上提出的新的特征请求或对已有特征的增强. 开源用户平台可能包括问题追踪系统 (issue tracking system, ITS), 开源仓库 (open source repository), 应用商店 (App store), 在线论坛 (online community) 等. 在这些开源用户平台上, 用户可以提交他们的问题报告、变更请求、用户评论、客户愿望 (customer wishes) 以及用户反馈 (user feedback) 等. 这些报告或者反馈, 可能是缺陷报告, 或特征请求, 也有可能是其他非信息性的评论 (如称赞). 目前大多数的研究中的特征请求主要有两种来源: (1) 开源用户平台直接产生, 比如 SourceForge.net 为设置专门的模块供用户提出特征请求或者提交缺陷报告; (2) 从用户反馈中提取, 用户反馈的内容一般被分为信息性和非信息性, 其中信息性内容一般包括特征请求和缺陷报告. 因此一些研究中使用分类等方法从用户评论中提取特征请求. 通过调研相关研究论文, 在一些研究中也会将用户评论中提取的特征请求称为增强报告 (enhancement report)、即时性需求 (just-in-time requirements) 或用户需求 (user requirements) 等. 在本文的研究中, 统一称之为特征请求. 为理解特征请求或者缺陷报告所对应的软件特征, 一些研究会从特征请求或者缺陷报告中提取产品特征 (product feature). 广义的产品特征指的是产品的属性或者功能[8]. 特征请求不同于缺陷报告, 缺陷会造成系统瘫痪, 无法使用; 而特征请求一般是提出新的功能请求或对现有功能的增强[9], 因此缺陷报告的优先级往往高于特征请求. 本研究仅针对特征请求进行调研.

3 需求获取

传统的需求获取方法包括面谈、原型和文档审查等. 不同于传统的需求获取活动, 特征请求主要来源于终端用户等. 本节主要总结现有研究中所提出的特征请求的来源、获取方法, 以及不同平台获取的特征请求的形式、特点等.

软件需求是指: 1) 为解决用户某一问题或达到某一目标所需的软件功能; 2) 系统或系统构建为了满足合同、规约、标准或其他正式实行的文档而必须满足或具备的软件功能. 在软件维护过程中, 软件需求文档和计划中蕴含的信息通常是过时的. 而用户反馈等记录了已经实现或部署, 以及有待实现的功能[10]. 实现用户反馈中的用户请求, 有助于提升用户的满意度. 在软件维护过程中, 如果新产品的特性不能引起用户的共鸣, 会造成开发工作的浪费、产品上市时间的延迟、产品的复杂性和运营成本的增加等[11]. 因此, 应用程序开发人员合理利用用户反馈中的用户请求, 有助于他们在竞争激烈的环境下更加高效地维护应用[12].

3.1 特征请求来源

特征请求的来源有很多, 常见的来源包括问题跟踪系统 (例如 Bugzilla), 开源软件仓库 (例如 GitHub), 应用商店 (例如 Apple AppStore, Google Play), 在线社区 (例如 Stack Overflow) 等. 问题跟踪系统是专门用于收集软件问题的平台, 用户以缺陷报告、特征请求、增强报告等的形式对软件问题进行反馈. 开源软件仓库是开源软件开发者进行开发管理的集中式场所, 用户可以提交问题报告或者工单. 其内容可能包括缺陷报告、特征请求、询问等. 应用商店提供不同的应用并允许用户进行评论, 用户的评论内容可能涉及对已有功能的用户体验或者对新功能的请求. 图 8 总结了现有研究中主要研究对象的形式. 从图中可以看出, 现有研究中特征请求来源多种多样, 可能的存在形式包括: 应用评论、问题报告、用户反馈、推特、开发邮件、顾客愿望、聊天信息等. 或者直接以特征请求、变更请求、增强报告的形式存在. 其中, 应用评论以及问题报告是最常见的特征请求来源. 其次, 也有研究从推特、开发邮件以及聊天信息等获取特征请求. 尽管不同平台上产生的用户反馈形式不尽相同, 其都一定程度以直接或者间接的方式蕴含了特征请求. Iacob 和 Harrison 通过调研发现, 大约有 23.3% 的应用评论是特征请求, 包括用户建议开发新的特征或者是对现有特征的改进[13]. Williams 和 Mahmoud 发现推特上用户评论中约 51% 是有用信息 (27% 是缺陷报告, 24% 是用户需求)[14]. 除此之外, 用户反馈中还可能包含缺陷报告, 用户使用体验等内容. 准确地识别出用户反馈中的特征请求和缺陷报告等有用信息, 对于软件维护和提升也起着非常重要的作用.

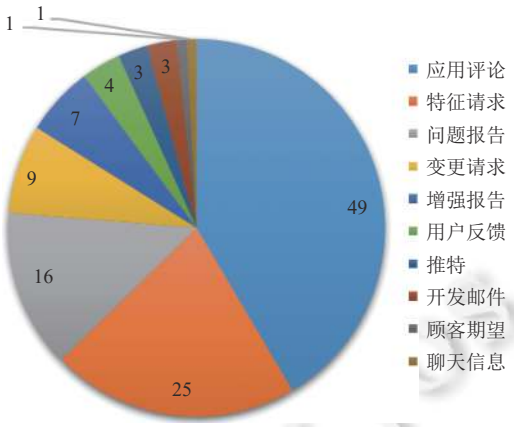

图 8 特征请求来源统计

3.2 特征请求获取

如何从各种形式的数据中提取对软件新特征请求的描述, 称为特征请求获取. 常见的从用户反馈中获取特征请求的方法有基于无监督学习方法、基于监督学习方法、基于半监督学习方法以及组合方法. 现有研究中采用的获取方法如表 3 所示[12−53], 最常用的方法是基于监督学习方法. 通常情况下, 用户反馈的数量巨大. 截至 2021 年 6 月, App store 上有超过 180 万应用. 每天产生的评论数量则是天文数字. 完全依赖人工从用户反馈中抽取特征请求的方法, 虽然可以确保较高的准确性, 但是是一件繁琐的任务, 费时费力, 在应用中不切实际.

表 3 常用的特征请求获取技术

学习方式		文献
无监督学习	规则匹配	[13,15,16]
监督学习	文本分类	[12,14,17–38]
	句子分类	[39–49]
	相似度	[50]
半监督学习	—	[51,52]
组合	—	[40,53]

3.2.1 基于无监督学习方法

早期的从用户反馈中获取特征请求的方法主要依赖基于无监督学习的规则匹配方法. 常见的方法是人工基于规则提取, 符合规则的描述会被抽取出来. Iacob 等人设计并实现了基于规则从用户评论中挖掘特征请求的原型工具. 定义了 237 条语义规则, 然后通过自动化的规则匹配算法提取出用户评论中的特征请求[13]. Carreno 和 Winbladh 基于信息检索技术, 采用 aspect and sentiment unification model (ASUM) 从用户评论中生成有建设意义的用户反馈. 抽取出的用户反馈可以作为软件下一版本开发的新需求或者改进需求[15]. 吕宏玉等人[16]提出基于句

式匹配和情感分析的特征请求识别方法, 用于从应用评论中抽取特征请求. 基于规则匹配方法的优点在于可以精准地提取出特征请求. 但是需要人为总结规则, 人力成本较高. 此外, 人工总结的规则, 难以覆盖所有情况.

3.2.2 基于监督学习的方法

最常见的获取特征请求的方法是基于监督学习的分类的方法. 根据分类对象的不同, 用户反馈分类可以分为两类: 句子级别和文本级别的分类, 表 3 中分别列出了现有研究中句子级别和文本级别的分类. 文本级别分类的对象是每条用户反馈, 通过分类器为每一条用户反馈贴标签. 文本级别的分类类别一般包括: 缺陷报告、特征请求、赞赏、使用体验、询问以及其他. 句子级别的分类是将每条用户反馈根据句子划分, 针对每一个句子预测一个标签. 句子的标签一般包括: 信息提供、信息搜索、特征请求、问题发现、解释说明、解决方法和其他等. 文本级别分类可以有效地识别出特征请求, 从而将不涉及特征请求的用户评论排除. 句子级别可以更细粒度地定位用户反馈中的特征请求, 可以有选择地将与特征请求无关信息进一步排除.

文本分类常见的流程如图 9 所示. 输入一般是大量的用户反馈 (包括应用评论、问题报告等), 用户反馈来源于图 7 中的开源平台. 首先采用自然语言处理方法预处理, 一般包括: 分词、去除标点符号、去除停用词、词形还原、提取词根等. 然后将经过预处理的文本向量化. 常用的向量化方法包括: 基于 bag-of-words (BOW) 的方法 (如 TF-IDF、TF 等), 基于 n-gram 的方法, 基于词嵌入的方法 (如 Word2Vec[54]、GloVe[55]等). 除此以外, 一些分类方法中将用户反馈的元知识 (metadata) 作为部分特征[17,18,21,26,33,42,45]. 经过相关研究验证的有效元知识有: 评论主题、评分等级、文本长度、句子位置、应用大小、安装数量、评论者数量等. 除了文本特征之外, 一些研究还挖掘辅助特征以便于分类器更好地学习特征. 如: 关键词[18,21,37,42]、句子情感分析[14,17,18,21,33,36,39,40]、语义规则[18,37]、标识语句的启发式方法[39,40,48]、词汇特征[40]、词性标注[21,45]等. 最后, 基于机器学习或者深度学习训练分类模型预测文本类别. 在分类模型中, 由于用户评论中不同类别分布不均衡, 采用重采样方法以均衡数据集在一些研究中被验证有助于分类器更好地学习类别特征[26,28,45].

图 9 常见文本分类流程

根据分类方法采用的模型的不同, 用户反馈分类可以分为基于传统机器学习的分类和基于深度学习的分类. 早期主要是基于传统机器学习分类算法的方法, 比如朴素贝叶斯 (naïve Bayes, NB)、多层朴素贝叶斯 (multilayer naïve Bayes, MNB)、支持向量机 (support vector machines, SVM)、逻辑回归 (logistic regression, LR)、决策树 (decision tree, DT)、随机森林 (random forest, RF)、最大熵模型 (MaxEnt) 等. 其采用的向量化方法一般基于 BOW 的 TF-IDF、TF、以及 *n*-gram 等. 随着深度学习在文本分类问题上的应用, 深度模型逐渐被用于用户评论分类, 如多层感知机 (multilayer perception, MLP)、卷积神经网络 (convolutional neural networks, CNN)[56]、文本循环神经网络 (text recurrent neural networks, TextRNN)[57]、长短记忆神经网络 (long-short term memory, LSTM)[58], 以及 fastText[59]等. 对应的向量化的方法一般是基于词嵌入方法: Word2Vec、Glove、BERT[60]等. 现有文献中采用的分类算法及其年份分布如图 10 中所示, 横坐标代表了不同的分类算法, 纵坐标代表了不同的年份, 圆圈中的数字代表论文的引用序号. 从图中可以看出基于传统机器学习的分类器最常被采用, 其中 NB 和 DT 使用频率最高, 其次是 RF 和 SVM. 深度模型于 2019 年开始被应用于提取特征请求, 并且有增长的趋势, 其中 CNN 是目前为止使用最广泛的分类模型. 在不同的情况下, 不同的分类器分类效果各有千秋. Guzman 等人[17]基于 NB、LR、NN 和 SVM 分类器的实验发现, 集成不同的分类器, 表现比单个分类器效果好. Stanik 等人[25]通过对比不同的传统机器学习分类器和深度学习分类器发现, 传统机器学习分类器更适用于小样本分类, 而深度学习分类器在大样本上表现更优. Ali Khan 等人[28]的研究发现深度学习在小样本上会产生过拟合. 因此, 在实际应用中, 需要根据实际情况, 选择合适的分类器. 大多数情况下, 用户反馈各个类别之间分布是不均衡的, 常见的策略是重采样. Young 等人[34]探究了重采样技术对用户反馈分类的影响, 采用随机欠采样、SMOTE 过采样和随机过采样, 以及 SMOTE-Tomek 将过采样和欠采样技术相结合, 分别在 5 种机器学习分类器上进行实验. 结果发现一般情况下过采样会取得更好的性能, 少数情况下, 欠采样技术表现更好.

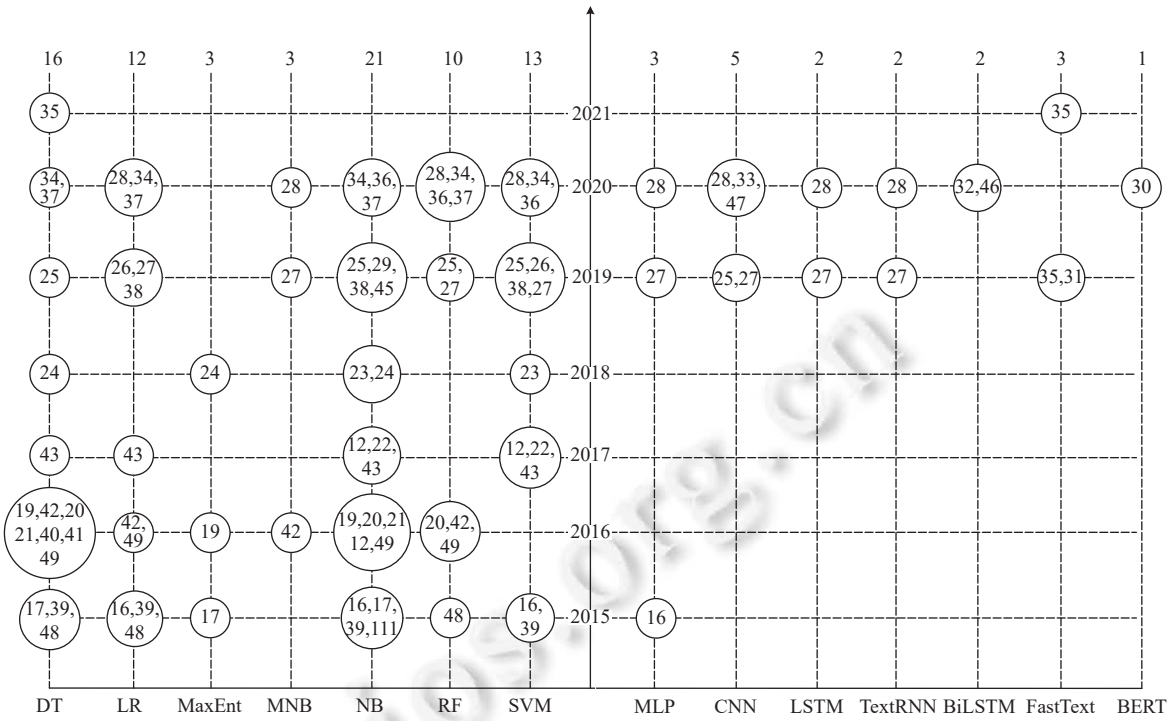

图 10 常见文本分类算法

除了特征请求的文本描述, 许多研究提出使用基于自然语言处理提取的其他的辅助特征. 比如关键词、情感分析、时态、语义规则、词性标注等特征. 基于关键词的辅助特征被证实效果显著. 例如, 缺陷报告与特征请求有着不同的关键词汇, 因此基于关键词的辅助特征可以帮助识别出特征请求[18,21,42]. 而情感分析, 一般是基于SentiStrength[61]、SnowNLP (https://pypi.python.org/pypi/snownlp)、Senti4SD[62]等工具对用户评论进行情感分类, 将用户反馈的情感分类为积极、消极以及中立. Peng 等人研究发现结合了情感分类的模型, 可以更好地识别出用户评论中的特征请求[21]. 而 Williams 与 Mahmoud[14]对来源于推特的用户反馈的研究中发现, 情感分析对分类没有明显的效果. 其原因可能是推特中用户反馈倾向于叙述性说明, 没有明显的情感区别, 因此对于分类缺乏指导性意义, 而应用商店中的用户反馈倾向于抱怨等.

3.2.3 基于半监督学习的方法

基于监督学习的方法需要有大量的人工标记数据, 尤其是深度学习算法, 模型往往依赖大量的标记数据学习不同类别的特征, 才能取得较高的分类准确度. 然而实际中, 大量的标记数据需要大量的人工成本. 为了减少人工标记数据成本, 并且取得同等的分类效果, 半监督学习算法被应用于特征请求提取.

基于半监督学习的特征请求抽取的方法主要包括自训练 (self-training)[63]、Rasco[64]、Rel-Rasco[65]以及主动学习 (active learning)[66]等. Deocadez 等人采用包括自训练、Rasco 和 Rel-Rasco 在内的 3 种半监督算法将应用评论分类为缺陷、请求和其他 3 类, 并基于 4 种基本分类器, KNN、C4.5、SMO 和 NB, 进行实验. 实验结果表明半监督方法可以大大减少人工标记数据, 仅需要监督学习 30% 的标记数据就可以取得类似的分类效果[51]. Dhinakaran 等人[52]将主动学习用于用户评论中的特征请求抽取. 主动学习基于 3 种不确定性采样策略: 置信度最低、边缘采样和熵方法选择最具信息性的标注数据加入训练集, 结果表明在达到与监督机器学习到达同等分类效果时, 主动学习可以显著减少数据标注的人力消耗.

3.2.4 其他方法

Rahimi 等人[67]提出一种新颖的、半自动化的方法, 用于从在线论坛中提取功能请求. 首先利用增量扩散聚类

算法 (incremental diffusive clustering, IDC) 将用户评论按主题归类, 然后采用 Apriori 关联规则挖掘多个利益相关者所关注的主题分组. 为每个主题确定性能、安全性和可用性等属性相关的质量问题. 最后推荐可用于创建个人角色的主题组, 创建角色. 该角色作为帮助了解项目早期阶段的架构问题可以指导架构设计过程, 并最大限度地减少后期成本高昂的重构的需要.

Zhang 等人[50]提出一种基于相似性的方法来将问题报告自动化标记为缺陷报告、特征请求和其他. 该方法采用混合相似度算法, 分别计算未标记的问题报告与用户评论之间的文本相似度, 以及与已标记问题报告之间的文本相似度. 两个相似度指标归一化, 然后分配贡献权重混合计算结果完成分类.

Shi 等人[32]提出一种 FRMiner 方法, 基于深度孪生网络从聊天信息中提取特征请求. 孪生网络将两个实例作为输入, 通过学习两个实例之间的相似度判断两个实例是否属于同一类.

Panichella 等人[39]提出一种组合了规则匹配和文本分类的算法. 手动总结 246 条语言模式, 并使用启发式算法自动识别. 并结合情感分析和文本分析使用 J48 分类算法进行分类, 将用户评论分类为: 信息提供、信息搜索、特征请求、问题发现和其他等.

Gribkov 等人[68]提出 Trans-LSTM 的神经网络模型, 用于从用户评论中提取信息表达. 首先将用户评论分句, 然后对每一个句子经过抽象自动机并提取出 (主题, 描述) 的表达, 并将表达的信息类别分为缺陷报告、特征请求、积极特征、消极特征. 该方法使用 Trans-CNN、Trans-LSTM 来建立输入上下文表示. 结果表明该模型在提取信息表达的效果优于其他模型.

Shi 等人[53]提出了一种自动化解决方案, 通过利用语义序列模式从开发电子邮件中发现功能请求. 首先使用之前研究中提出的 81 条模糊规则[69]来标记电子邮件中的句子. 然后在 2-gram 模型中用电子邮件的上下文信息表示语义序列. 从 317 封 Ubuntu 社区随机采样的电子邮件中生成了 10 个语义序列模式. 实验结果表明该方法可以有效地识别来自电子邮件的功能请求.

3.3 特征请求特点

Hoon 等人[70]从评论长度、评论数量增长、评论评分分布演化以及评论大小分布演化等角度对 iOS 系统 22 个类别应用的用户评论展开统计调研. 研究结果发现: (1) 22 个类别应用中, 评论长度有显著差异. 评分较高的软件其评论较短; 评分较低的软件其评论较长. 在 5 个评分等级上, 评论长度有显著差异. (2) 不同类型的应用, 评论数量也有差异. 与付费应用相比, 免费应用收到的评论往往更多. 而信息类的应用, 如天气预报等, 只在短时间内得到用户的关注. 复杂的应用如医疗等, 此类用户相对较少, 因此评论也较少. (3) 评论数量的增长通常在新版本发布时达到峰值.

特征请求来源多种多样, 这就导致其有以下的特征: (1) 数量大: 特征请求可以由不同的用户随时随地地提出, 短时间内会产生大量的用户请求. (2) 非结构化: 由于特征请求是不同的用户提出的, 用户对于软件缺乏整体的把握, 因此产生重复的特征请求概率很大. 此外, 不同的特征请求之间, 可能存在结构上的依赖或者互斥关系. 因此, 特征请求是无结构化的. (3) 来源多样化: 特征请求的来源可能是企业人员、用户以及开发者等. Noll 等人[71]调研了开源软件开发的需求来源, 结果表明大多数的需求来源是开发者, 超过一半的需求是由核心团队以外的参与者提出的. 用户作为项目的贡献者的参与是开源开发的一个必要方面. (4) 质量参差不齐: 由于来源的多样性 (来自企业、用户、开发者等)、受教育程度的不同、语言以及专业领域的不同等, 导致特征请求的差异性[72]. 用户的自然语言描述可能包括错误的拼写、口语化的表达等. 导致特征请求的质量与传统的需求规格说明书相比有较大的差距. 特征请求的这些特点使得其区别于传统的需求, 因此对其处理也要有所不同. 传统需求工程的处理流程和方法可能不再适用于特征请求, 下文将继续讨论处理特征请求的具体方法, 并与其他传统需求工程活动对比.

4 需求分析

需求分析是软件计划阶段的重要活动, 也是软件生存周期中的一个重要环节. 需求分析一般是需求工程师经过调研和分析, 准确理解用户对项目的功能、性能、可靠性等具体要求, 经过分析与整理将用户非形式化的需求

表述转化为描述完整、清晰与规范的文档, 从而确定软件需要实现哪些功能、完成哪些工作的过程. 分析的活动主要包括识别、定义和结构化[73]. 传统需求分析阶段的子活动包括需求梳理、需求优先级排序、需求建模等. 近来, 自然语言处理技术和机器学习技术使得特征请求的自动化分析得以实现, 因此许多研究被提出. 总结已有的针对特征请求分析的研究, 我们将其分为: 软件特征抽取、分析梳理、优先级排序. 本节将整理每一种子活动的相关研究. 其中目前针对特征请求建模的研究仍有欠缺, 因此本文不单独设立特征请求建模相关章节.

4.1 软件特征抽取

根据 Kang 等人的定义, 软件特征是软件系统一个突出的用户可见的层面、质量或特性[74]. 软件特征抽取就是采用文本挖掘的方法将这些层面、质量或特性从软件需求中抽离出来. 软件特征可以作为相似产品开发中的早期特征, 被提供给领域分析师, 也可以用于构建软件特征模型[75]. 提取特征请求中的特征对于理解用户的请求、挖掘不同特征之间的依赖关系也非常重要. Bakar 等人[5]针对 2015 年之前的从自然语言需求描述中抽取软件特征的研究做了综述调研. 研究总结了从不同的自然语言需求中提取软件特征, 如: 需求规格说明书、产品描述等. 在本节, 主要对如何从用户反馈和特征请求中抽取出用户所期望的软件特征进行调研.

现有研究中, 除了人工标注方法, 一般采用自然语言处理技术抽取软件特征. 首先对文本进行预处理, 过滤掉无关内容. 然后词法分析, 确定单词的词性. 基于预先定义的过滤规则, 提取出匹配规则的内容, 作为软件特征. 通用的特征抽取流程如图 11 所示. 表 4 总结了现有研究中软件特征抽取的来源、过滤规则等. 从表中可以看出, 特征的来源主要为应用评论 (包括专家评论[75]和用户评论[76−79]) 和应用描述[77], 过滤规则一般基于单词或词语的词性, 筛选出句子中的名词、动词、形容词或其组合, 或者是基于人工定义的词性组合等. 抽取得到的软件特征可被用于需求复用[75]、分析用户对特征的满意度[76]、软件维护[77]、优先级排序[78]以及竞品分析[79]等.

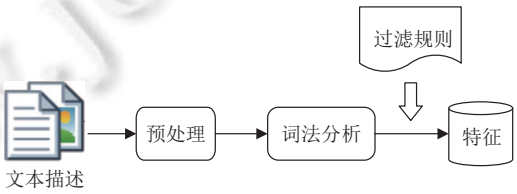

图 11 常见软件特征抽取流程

表 4 软件特征抽取研究总结

文献	特征来源	过滤规则	应用	年份
[75]	软件专家评论	<形容词, 名词> 或<名词, 形容词>, <动词, 形容词> 或<形容词, 动词> 以及<动词, 名词, 形容词>	需求复用	2016
[76]	用户评论	名词、动词、形容词	用户对软件满意度分析	2017
[77]	应用描述和用户评论	POS模式和句式	软件维护	2017
[78]	应用评论	名词	优先级排序	2017
[79]	用户评论	<名词, 名词>, <名词, 形容词>, <名词, 动词>, <形容词, 名词>, <形容词, 动词>, <动词, 名词>	竞品分析	2019

Shah 等人[80]调研和分析了现有研究中在用户评论中标注软件特征的标注指南, 并以提高培训和标注的质量为目标, 提出新的标注指南.

在自动化抽取软件特征研究方面, Bakar 等人[75]提出一种自动化的 FENL (feature extraction for reuse of natural language requirements) 方法, 首先预处理软件专家评论, 选择名词, 动词和形容词, 然后通过聚类排除掉边缘文档. 最终从评论中提取代表产品特征的短语, 即, 以<形容词, 名词>或<名词, 形容词>, <动词, 形容词>或<形容词, 动词>以及<动词, 名词, 形容词>形式出现的短语组合. 与手动特征抽取相比, 该自动化方法能提取出更多的表示软件特征的短语.

Bakiu 等人[76]提出使用搭配算法从用户评论中提取软件特征. 首先过滤掉评论文本中名词, 动词和形容词之

外的其他词, 然后再过滤掉出现次数较少或者不符合规定距离的特征. 最后还使用词汇情感分析揭示用户对于特定特征的满意度.

Johann 等人[77]提出 SAFE (a simple approach for feature extraction), 手动建立 18 个 POS (part-of-speech) 模式和 5 个句式, 用于从应用描述和用户评论中提取软件特征. 该方法不需要大量的训练数据和配置数据. 并且已经被广泛应用于特征抽取[44,81].

Licorish 等人[78]通过提取应用评论中的名词提取特征, 利用该特征进行预测特征请求的处理优先级顺序.

Dalpiaz 等人[79]首先对用户评论中的单词或词语进行词性标注, 然后提取其中的<名词, 名词>, <名词, 形容词>, <名词, 动词>, <形容词, 名词>, <形容词, 动词>, <动词, 名词> 组合, 来从竞品应用中挖掘软件特征. 对比产品与竞品的特征、加强优势、克服弱点, 以提高软件的竞争力.

整体来看, 目前最常见的软件特征抽取的方法是基于词性标注, 提取名词、动词和形容词及它们之间不同的搭配. 因为名词、动词和形容词是常见的描述特征的词性[76]. 更具体地, Johann 等人[77]提出 18 个 POS 模式和 5 个句式. Shah 等人[80]提出的指南建议: (1) 仅标注由连续单词组成的特征; (2) 不要标注对应用程序本身的引用; (3) 仅标注包含名词的特征; (4) 将标注的特征的长度限制为最多 3 个单词. 因此, 在未来的研究中, 可以采用基于词性标注的方法来提取用户评论尤其是特征请求中的软件特征. 但是单纯提取出 3 种词性的不同搭配, 提取的结果中可能包含不代表软件特征的冗余信息.

4.2 结构化分析

需求分析的目的是挖掘出信息的本质含义, 准确理解用户的意图, 在对需求信息内容的理解上与用户达成一致. 而特征请求由于其来源以及提取过程的特殊性, 往往是非结构化的、口语化的表达, 这对需求分析人员准确分析并理解特征请求带来了障碍. 现有研究中, 分析特征请求的研究可以分为 3 类: 用户意图识别、特征请求分类、摘要生成. 各研究主题的具体内容如下所示.

4.2.1 用户意图识别

特征请求的来源是终端用户, 用户在向开发人员提出请求的时候, 为了使自己的请求更容易被理解和接受, 往往会解释请求的背景、原因, 以及实现请求可能的解决策略等信息. 用户意图识别即挖掘出用户特征请求中描述用户意图的内容, 用户意图识别有助需求分析人员更好地理解用户的意图.

Shi 等人[69]提出一种基于模糊规则理解特征请求的自动化方法. 该方法基于自然语言处理技术分别从词汇、句法和语义 3 个层面提取经常出现的词汇、句子结构以及语义模式, 总结出 81 条规则. 并采用 RF、NB、J48、LR 和 SVM 等机器学习分类算法将特征请求从句子层面分类为意图、解释、优点、缺点、举例以及其他类型. 实验结果表明将模糊规则应用于机器学习方法, 可以显著提高分析性能. Mu 等人[82]基于 Shi 等人[69]总结的 81 条规则, 提出一个 NERO (content annotation and smelly feature requests detection) 工具. 基于规则匹配, 将每条特征请求的句子分类为意图、解释、优点、缺点、举例和其他.

4.2.2 特征请求分类

需求分类是需求分析的重要活动之一. 不同于需求获取阶段采取的分类算法, 需求分类是指按照既定的分类维度, 根据需求所描述的软件产品不同层面的属性, 将众多软件需求分门别类. 用户提出的特征请求涉及软件功能及性能的方方面面, 如安全性、可靠性等. 将描述不同方面的特征请求分门别类既有助于需求分析师分析, 又有助于结构化存储和管理用户特征请求. 本小节从分类方法、分类特征以及分类类别 3 个角度总结现有的特征请求分类的相关研究.

表 5 中总结了目前所有的特征请求分类研究的分类类别以及采用的分类方法. 虽然各个研究的分类类别均不相同, 但主要是从功能性和非功能性角度, 以及细化的非功能特性进行区分. 从分类方法上看, 对特征请求分类的方法可以分为基于监督学习[83,84]方法、基于无监督学习[19,85], 和基于关键词共现矩阵[86]的方法等. 基于监督学习的方法主要是基于机器学习分类模型的方法, 普遍使用的有: KNN[84,87], NB[84,87], SVM[84], RF[83], J48[87], Bagging[87]. 基于无监督学习的方法主要使用聚类方法, 包括: DBSCAN[19,83], Spherical K-means[85]. 早期 (2016 年前) 研究主要采用无监督的聚类算法, 而之后的研究则主要采用机器学习分类算法.

表 5 现有研究中分类类别

文献	类别	方法	年份
[85]	根据软件特征聚类	Spherical K-means聚类	2009
[19]	无监督聚类	DBSCAN聚类	2016
[83]	缺陷报告, 功能请求, 性能请求, 安全请求, 能耗请求, 可用性请求, 其他	基于RF的机器学习分类	2017
[84]	安全性, 可靠性, 性能, 生命周期, 可用性, 易用性, 系统界面	基于KNN, NB以及SVM的机器学习分类	2018
[87]	可用性, 可靠性, 可移植性, 性能	基于NB, Bagging, DT和KNN的机器学习分类	2018
[86]	系统, 资金, 垃圾邮件, 隐私, 其他	基于关键词的词语共现矩阵	2020

为了将特征请求进行分类, 除了将特征请求文本向量化之外, 一些研究还利用其他的特征进行辅助分类器分类. 常用的辅助特征包括: 项目关键词以及启发式属性[84], 用户评论的一些元属性 (用户评分、评论的文本长度以及应用的类别)[83], 以及变更日志[87]等. 其中项目关键词被证实是最为有效的辅助特征[84]. 因为在特征请求领域, 描述同一层面性能的特征请求可能共享相同的关键词语料库.

从分类类别来看, 不同于第 2.2 节中的机器学习分类, 特征请求分类是根据特征请求描述需求的不同层面, 如安全性、可用性、易用性等, 将特征请求进行分门别类. 常见的需求工程将需求分为功能性和非功能性需求. 功能性需求针对软件的功能方面提出需求. 非功能性需求是指软件产品为满足用户业务而必须具有且除功能需求以外的特性. ISO 25010 将软件产品质量分为功能性以及非功能性, 进一步地非功能性包括: 功能适用性、性能效率、兼容性、可用性、可靠性、安全性、可维护性和可移植性[88]. Li 等人[84]将特征请求分类为安全性、可靠性、性能、生命周期、可用性、易用性和系统界面等. Scalabrino 等人[83]将用户评论中的信息性评论分为缺陷报告、功能请求、性能请求、安全请求、能耗请求、可用性请求和其他等. Tao 等人[86]将用户评论中安全层面的评论分为系统 (system)、资金 (finance)、垃圾邮件 (spam)、隐私 (privacy) 和其他 (other). Wang 等人[87]将用户评论中的需求分类为可用性、可靠性、可移植性以及性能.

4.2.3 摘要生成

区别于用户故事明确的叙述模板, 特征请求一般是口语化的表达, 不便于理解. 为了规范化特征请求的语言描述, 有研究提出采用摘要生成的方法, 提取特征请求的主题, 并规范化表达.

Jha 等人[89]介绍了一种 MARC 2.0, 利用框架语义对应用商店的用户评论进行分类和总结. 首先使用 FrameNet 进行语义角色标记 (semantic role labelling), 给文本中的单词和短语加上框架元素的注释. 然后使用 4 种总结算法 Hybrid TF、Hybrid TF-IDF、 SumBasic 和 LexRank 分别将缺陷报告和特征请求生成摘要总结. 通过将自动生成的摘要总结与人工生成的进行对比, 结果发现 SumBasic 总结算法能够在很大程度上生成与人工一致的摘要总结.

Williams 等人[14]将推特上的软件用户评论自动化分类为缺陷报告、用户需求. 然后分别采用 Hybrid TF、Hybrid TF-IDF 以及 SumBasic 生成摘要总结. SumBasic 摘要算法的底层冗余控制机制允许一些较低频率的单词的偶尔出现, 其摘要总结的结果与人工生成的摘要最为接近.

4.3 优先级排序

在实践中, 项目资源有限, 不能保证满足所有的用户需求, 因此项目通常采用分阶段的开发方式. 在项目开始阶段, 权衡成本和收益, 在既定成本下实现利益最大化. 因此, 要确定用户需求的优先级, 优先迭代高优先级的需求, 保证项目最终最大程度满足了用户的需求.

4.3.1 传统需求优先级排序方法

优先级排序任务通常是依据某种排序标准 (如重要性、成本等) 对需求或特征进行排序, 以确定下一版本中要包含的需求或特征. 传统需求常用的确定优先级的方法有层次分析法 (analytical hierarchy process, AHP)[90], KANO 模型分析方法[91], 二进制搜索树 (binary search tree, BST)[92], 累计投票制 (cumulative voting, CV)[93], 规划策略 (planning game, PG)[94], MoSCoW[95], CBRank (case based rank) 方法[96], 优先级组 (priority group) 方法[97]. 表 6 总结了这几种方法及其具体描述, 以及是否适用于大量的特征请求. 其中 KANO 方法和 CBRank 方法依赖于向用

户征集对需求的看法. AHP 和 CV 方法随着数据量增多, 计算量急剧增大, 因此不适用于大量的特征请求的优先级排序.

表 6 传统需求工程优先级排序方法

排序方法	描述	大量特征请求适用性
AHP	AHP采用人为判断需求两两之间的相对优先级高低, 假设待排序的需求为n, 其计算复杂度为$n\times(n-1)/2$. 随着需求数量的增加, 其计算消耗越来越大, 因此, 该方法不适用于大量的特征请求优先级排序	不适用
KANO	KANO主要通过问卷调查的方式, 分别调研需求被满足和不被满足时, 用户的满意度和不满意度. 并将功能、服务的态度属性分为6类, 分别是魅力属性A, 期望属性O, 必备属性M, 无差异属性I, 反向属性R和可疑结果Q	—
BST	二进制搜索树(BST)方法将重要的需求插入到搜索树的右边, 不重要的需求插入到搜索树的左边. 通过对构建的树进行深度优先遍历, 便可以生成优先的需求列表. 二进制搜索树方法适用于较大需求集的优先级排序	适用
CV	累计投票制(CV)方法给被调查者一定数量的假想单元(比如金钱和时间)来分配到需求之间, 需求越重要, 其得到的单元应该越多. 被调查者可以根据个人偏好以任何方式分配	不适用
PG	规划策略(PG)以故事卡的形式, 用户根据需求重要性将其分为3个组. 程序员按风险和成本分类. 假设待排序的需求为n, 那么对比的次数为n	适用
MoSCoW	MoSCoW方法将待开发的需求分为MUST, COULD, SHOULD, WON'T 或者WOULD, 这些不同的等级代表了需求在下一版本中被开发的重要性	适用
CBRank	CBRank方法通过收集用户的偏好信息等, 演化出需求优先级整合规则, 从而可以计算获得所有需求的优先级结果	—
Priority Group	优先级组方法是将待排序的需求按优先级高低进行分组. 最常见的分组是分为高、中、低3个优先级. 高优先级是下一版本中需要实现的关键需求; 中优先级是可以稍后实现的系统必要的需求; 低优先级是增强功能或特征, 实现了会更好	适用

4.3.2 特征请求优先级排序方法

传统的需求与特征请求之间存在很大的不同, 尤其是数据量方面, 因此特征请求排序方法与已有方法上存在不同. 以下部分调研了目前针对特征请求排序方法研究.

Laurent 等人[98]探讨了以供应商为基础的开源项目的产品需求的获取和优先级排序过程. 以供应商为基础的开源项目主要采用 3 种方法来确定用户需求的优先级顺序: (1) 用户对功能请求进行投票支持, 票数高的优先级高; (2) 在虚拟世界游戏中, 贡献高的利益相关者为功能请求赋予一个优先级; (3) 没有明确的优先级划分的论坛, 更依赖用户在评论中设定优先级.

Chen 等人[99]提出的 AR-Miner 工具首先利用主题建模技术 (latent Dirichlet allocation (LDA) 和 ASUM) 将信息性的用户评论进行分组. 然后针对分组结果分别进行组间和组内排序. 组间排序采用 3 个指标组成的线性函数进行评估, 3 个指标包括: 每组包含的评论数量、组内评论的时间跨度、组内评论的平均评分. 组内排序使用比例、重复项、概率、评分、时间戳等指标的线性组合进行排序. 最后采用直观的可视化方法呈现最具有信息性的组别.

Keertipati 等人[100]为每一个特征请求句子确定 4 个属性: 频率 (句子中出现重要特征的频率)、用户评分 (评分越低优先级越高)、负面情感 (软件特征的负面情绪越高, 优先级越高) 和情态动词 (情态动词数量越大优先级越高). 设计 3 种排序方法: 基于独立属性的方法 (独立地考虑每一种属性)、基于权重的方法 (为每种属性设定权重值)、基于回归的方法 (回归函数), 在用户评论数据上对 3 种优先级方法进行评估. 结果表明基于独立属性的方法允许开发人员选择重要的属性进行优先级排序, 或者允许人为设定重要功能决定优先考虑的功能. 基于权重的方法依赖于开发人员的经验和专业知识, 为不同的属性设置不同的权重值. 基于回归的方法便于发现一些微观洞察, 建议作为引导机制来弥补缺乏专业知识的情况.

Villarroel 等人[19]提出一种 CLAP (crowd listener for release planning) 的工具, 首先使用 RF 算法将用户评论分类为新特征 (new feature)、缺陷报告和其他; 然后使用 DBSCAN 聚类算法聚类相似的用户评论; 针对聚类的每一簇提取以下特征: 簇中包含的评论的数量、簇中用户评论的平均评分、簇的平均评分与 APP 的评分之间的差别、

用户给当前簇中用户评论的评分与以往的用户评论的评分之间的差别、簇中包含的硬件设备的数量, 最后使用 RF 算法给每一簇标记为高、中、低 3 种优先级.

Morales-Ramirez 等人[101]结合策略决策者和终端用户的意见来共同决策需求的优先级. 他们首先利用自然语言处理技术 (例如 POS 标签、LDA、情感分析、句法分析等) 提取用户反馈中的特征 (包括标题、描述、软件特征、主题、情感、目的、情态、长度等), 然后为需求找到一组相似的用户反馈, 使用用户反馈的属性的平均值计算需求的属性: 情感极性、目的、严重程度等. 同时由策略决策者选择新的需要排序的需求, 提取出决策者的偏好. 将用户反馈的信息与决策者信息结合计算出当前的需求优先级.

Scalabrino 等人[83]在 Villarroel 等人[19]的基础上, 进一步完善 CLAP 工具. 首先监听来自用户的评论, 并将非功能性信息自动化分类为安全问题报告、性能问题报告、能耗过多报告以及对可用性的改进请求等. 然后采用 DBSCAN 聚类算法将每一类中的评论进行聚类. 最后采用 RF 算法对簇优先级排序为高和低. 分类算法采用的特征包括: 簇中评论数量、簇的平均评分、簇平均分与应用平均分的差距以及簇中不同设备的数量. 实验评估结果表明该方法比 AR-Miner[99]方法更加优越.

Licorish 等人[78]采用优先级组的思想将用户评论分为 4 类: NAI (没有提供可操作的信息)、MS (对改进提出了温和的建议)、CFP (应用程序核心功能的功能和性能问题) 以及 SP (报告严重问题). 抽取用户评论中的软件特征名词的频率、评分、消极情感以及情态动词作为特征, 并构建多层回归模型进行分类, 预测用户评论修复的紧迫性. 实验结果表明, 该方法可以结构化捕捉用户评论的有用信息, 为开发人员节省宝贵的时间.

Etaiwi 等人[102]选择了 3 种共识算法 (ExactAlgorithm、BioConcert 和 Kwiksort) 来生成最优共识排名. 他们使用 CLAP 工具将用户评论分为 8 类 (缺陷报告、功能请求、性能请求、安全请求、能耗请求、可用性请求和其他), 然后在每一类中聚类, 最后利用共识算法产生一个最优共识排名, 以决定在应用的下一个版本中要实现的改进. 他们采用自定义的评论属性 (基数、最早日期、平均评分和类别) 独立地对 CLAP 产生的评论集群进行排名.

刘名成等人[103]提出了面向产品设计的用户需求重要度分析方法. 运用数据挖掘技术确定用户需求, 综合用户关注度和用户满意度. 用户满意度越低, 关注度越高, 需求期望实现程度越高, 用户需求的重要程度及潜在的创新价值就越高. 该方法结合 TextRank 算法、词典和规则, 构建用户需求重要度评价要素池. 综合用户关注度和满意度指标参数, 采用 TextRank 算法, 构建评价语料池, 建立重要度综合评测参数计算模型, 进而指导设计者选择科学有效的设计方向.

4.3.3 现有研究的分析

传统需求工程优先级排序算法从分组的角度可以分为两类: 分组方法和非分组方法. 分组方法最终产生的优先级结果是按照优先等级划分的组 (如 KANO、MoSCoW、优先级组)[19,84,99]. 非分组方法最终产生的结果是经过两两对比最终得到的优先级列表 (如 AHP、BST、CBRank) 或者每一个需求赋予的不同权重 (如规划策略、CV)[100−103]. Kakar 等人[104]对比了 KANO 模型方法和优先级组方法, KANO 模型方法对每一个功能请求使用一个正向问题和一个反向问题, 同时捕捉到满足的奖励和不满足的惩罚, 从潜在需求的超集中学习了丰富的客户偏好知识, 可以驱动软件需求的优先级排序. 但是 KANO 模型从正反两个角度考虑特征请求被实现的重要性, 因此其效率会降低. 优先级组方法从一个角度进行评鉴, 即用户对一个功能实现的满意度, 相较于 KANO 方法有相对高的效率. 但是仅包含一个维度, 不能更加全面地获取用户的看法. 传统的 KANO 模型方法和优先级组方法均是考虑用户对给定特征请求被实现的满意度, 需要较多的人为参与, 可以获得相对准确的优先级结果. 但是当特征请求数量较多时, 这种方法不实际.

总结已有特征请求的优先级排序方法, 可以发现有以下特征: (1) 特征请求的数量规模不定, 因此自动化优先级排序非常重要, 并已经成为发展趋势. 自动化排序可以节省大量的人力物力. (2) 为了实现特征请求的自动化排序, 通用方法是提取特征请求的属性. 常用的用于排序的文本属性包括: 软件特征出现的频率、用户评分、情感分类、情态动词的数量等. 为了提升排序的效率, 一些研究会将类似的特征请求聚类, 根据聚类结果进行组间排序和组内排序. 组间排序采用的属性一般包括: 组内评论的数量、组内用户评分的平均值、评论的时间跨度等. (3) 用户请求数量与开发响应数量之间有显著的强相关性. Licorish 等人[105]通过调研安卓社区记录的问题报告以及谷歌

的改进措施与用户请求之间的相关性, 发现开发团队普遍对社区用户提出的高频特征请求进行了回应. 因此用户请求数量可以作为优先级排序的重要指示性指标.

相较于传统的需求, 特征请求数量规模较大, 一些传统的需求优先级算法不再适用. 比如 KANO 主要依赖人工回答问卷调研. 当特征请求数量较大时, 该方法可以获得较为准确的结果, 但是人力消耗很大. 采用一些指示性指标, 比如用户请求的数量, 以及用户对软件特征的评分等判断优先级, 这样的策略可以帮助开发人员筛选出用户呼声高的请求. 但是这样的方法也存在一定的缺陷: (1) 可能会忽略对于系统关键但是用户关注度不高的特征; (2) 一些早期提出的请求可能随着时间流逝逐渐被忽略. 此外, 基于用户角度的指示性指标来指导优先级排序, 缺乏从项目利益相关者角度的参考指标, 没有考虑开发成本以及时间. 一种好的优先级排序策略应该同时考虑用户角度的重要性以及项目角度的开发成本, 与预期收益的权衡. 特征请求优先级排序未来研究的方向应该在自动化优先级排序过程的同时, 综合考虑用户需求、项目开发成本以及预期收益.

5 需求管理

需求管理是需求工程中非常关键的一步, 从需求分析开始, 贯穿软件项目的整个生命周期. 需求管理直接关乎软件整体的质量, 甚至软件项目的成败[106]. 传统的需求管理过程主要包括: 需求评审、需求跟踪、变更控制等活动, 图 12 描述了需求收集与需求管理各个活动, 以及与需求开发之间的关系. 特征请求提出之后, 需要对其进行评审, 以确定是否进行开发. 变更管理是为了对新提出的特征请求进行管理, 这依赖于需求跟踪. 三者之间没有绝对的界限, 而是互相依赖, 相辅相成的关系. 本节将从这 3 方面对特征请求的管理活动进行调研, 在对研究工作划分时, 为了契合本文框架的描述合理性, 部分研究工作的划分可能会有所调整.

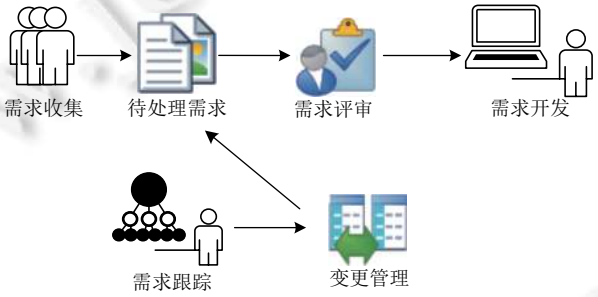

图 12 需求管理活动

5.1 特征请求评审

需求评审是指对需求的质量、正确性等进行审查. Wiegers 提出的需求评审的检查清单中对需求规格说明书的组织、完整性、正确性、质量属性和可跟踪性等进行细化[107]. 需求评审是需求分析过程中的“过滤器”, 可以在开发之前过滤掉低质量的特征请求, 以减少开发过程中的错误和缺陷. 特征请求的生命周期如图 13 所示, 用户提出新的特征请求, 特征请求的状态处于开放状态. 然后需求分析人员对特征请求进行需求分析、质量评估, 拒绝低质量 (如重复、冗余、歧义等) 的特征请求. 被拒绝的特征请求可以经过修改重新被提出或者直接结束. 被接受的特征请求被交付给开发人员开发, 开发成功的特征请求最后处于成功状态, 开发失败的特征请求处于失败状态. 特征请求评审发生于质量评估阶段, 需求分析人员评估特征请求的质量并进行筛选.

目前针对特征请求质量评估主要依赖于定义质量标准, 并按照质量标准对特征请求的质量评估, 进而过滤出低质量的特征请求.

在质量标准方面, Heck 等人[108,109]提出一种针对 JIT (just-in-time) 需求的特征请求质量评估框架, 该框架指出 JIT 需求需要满足的 3 个质量标准包括: 完整性、统一性以及一致性和正确性. Aversano[110]对企业开源系统的问题报告的质量调研. 从是否包含细化条目、是否包含附件、评论和可读性 4 个指标评估问题报告的质量. Shi 等人[111]调研了开源社区的特征请求被拒绝的原因, 特征请求被拒绝的主要原因包括冗余 (23.37%)、重复 (18.90%)、

冲突 (4.67%)、不完整 (3.86%) 等 19 种类别. Heck 等人[112]调研了开源项目上特征请求的重复率, 其中 Mozilla Firefox 的特征请求重复率高达 36%. 总结现有研究中关于特征请求的质量标准, 我们得到了如图 14 所示的质量模型, 可以用于评估特征请求的质量.

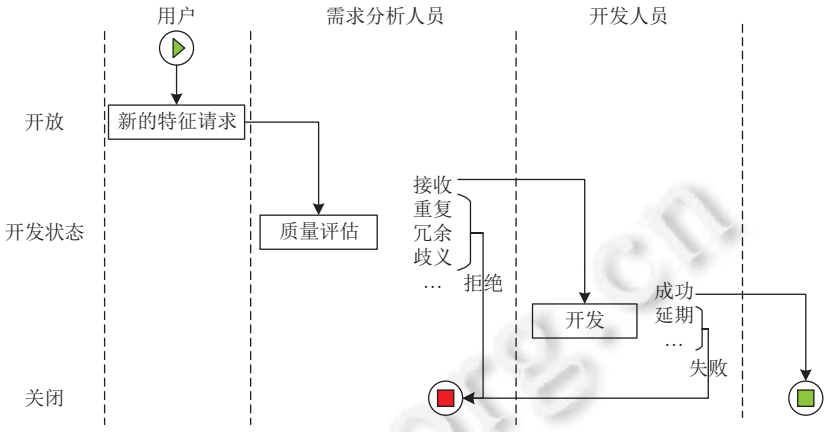

图 13 特征请求的生命周期图

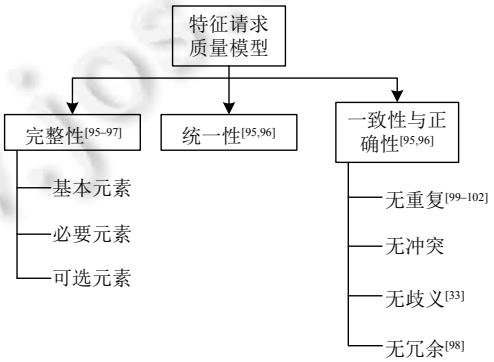

图 14 特征请求质量模型

现有研究主要集中于识别重复、冗余和歧义的特征请求. 重复与冗余识别任务之间的区别在于: 重复检测主要对比新提出特征请求与现有特征请求之间是否重复. 冗余检测主要识别新提出的特征请求与已有软件特征之间是否相同. 冗余检测需要维护软件产品已有特征, 已有软件特征可以基于用户手册、发行说明以及拉请求 (pull request) 进行挖掘. 有歧义的特征请求因其请求难以确定, 也会被需求工程师拒绝.

Gill 等人[33]总结了自然语言的 3 种层次的歧义: 词法歧义、句法歧义和语义歧义. 为了处理有歧义的特征请求, 他们建议使用自动化工具和技术, 并结合特定领域的知识、术语和观点, 以减轻人工识别和筛选的工作量, 并避免人工识别可能造成的错误.

Shi 等人[111]提出一种基于软件特征树模型的方法来自动识别冗余软件特征. 该方法分别利用用户手册、发行公告以及拉请求构建软件的特征树. 计算特征请求与软件特征树的子节点之间的相似性, 如果相似性超过一定阈值, 则认为特征请求是冗余的. 自动识别冗余特征可以减少需求分析师的工作负担.

Gu 等人[113]提出一种自动推荐算法, 当用户提交问题报告时, 自动推荐潜在重复问题报告. 该方法利用问题报告的摘要和描述, 预处理后转化为 TF-IDF 向量, 然后使用无监督的聚类算法聚类相似报告, 为用户推荐可能重复的问题报告. 推荐结果的 recall 达到 66%–100%, 可以在问题报告产生之时有效减少重复的问题报告.

Li 等人[114]使用基于文本相似性的方法为新提出的特征请求推荐候选重复特征请求, 该方法分别计算新提出特征请求与已有特征请求的总结 (summary) 和描述 (description) 的向量空间的 cosine 相似性, 二者的算术平均作

为特征请求之间的相似性. 相似性超过一定阈值的作为重复特征请求候选集.

Heck 等人[115]总结了特征请求重复的原因, 包括: 提出了重复的解决方案、部分重复、对特征请求提交补丁、同一个作者多次提交、特征请求的重要属性不同、措辞描述不同、没有进行重复检查等. 此外, 他们还总结了难以通过文本相似性识别出的重复类型: (1) 提出了重复的解决方案; (2) 部分重复; (3) 措辞描述不同; (4) 同一作者多次提交; (5) 没有进行重复检查.

5.2 特征请求跟踪

Gotel 等人将可追溯性定义为“可追溯性是将存储在某种人工制品中的数据相关联的潜力, 以及检查这种关系的能力”[116]. 在软件开发过程中, 随着业务和技术不断变化, 用户不断提出新的请求, 导致需求不断变化. 需求跟踪就是“一种描述和跟踪整个需求生命周期 (包括前向和后向) 的能力”[117].

需求跟踪可以分为横向跟踪和纵向跟踪. 横向跟踪指的是在需求工程同一阶段, 同一抽象层次的产物之间的关联, 比如需求之间. 纵向跟踪指的是对处于不同阶段、不同抽象层次制品间的跟踪, 如从软件需求到软件代码之间的跟踪. 在实际的需求跟踪过程中, 主要有需求跟踪矩阵和链表法两种策略实现需求跟踪. 接下来分别从横向和纵向两个角度对特征请求的跟踪研究调研.

5.2.1 横向跟踪

特征请求的横向跟踪就是将同一层面的需求产物之间建立联系. 特征请求建立横向跟踪有助于项目开发人员以及用户了解项目的当前状态以及实现新的特征, 有助于尽早发现需求之间的冲突, 理解软件需求的实现意义, 更好地处理需求之间的逻辑相关性, 避免重复或者不一致的工作[115], 以减少软件开发过程中的软件失败[118].

现有研究中, 与特征请求相关的构建横向跟踪的需求产物包括: 顾客期望与产品需求之间[119], 不同的特征请求之间[115], 以及不同的问题报告之间[120]. 横向跟踪关系包括: 重复、依赖、冲突等. Heck 等人[115]将特征请求之间的关联类型定义为: 依赖 (一个应该在另一个之前解决)、重复和参考. Merten 等人[120]将问题报告之间的横向跟踪关系定义为重复 (两个问题报告描述相同的软件特征) 和泛型 (两个问题报告参考软件的同一个特征). 然而构建不同产物之间的横向跟踪的实现是一项费时费力的任务, 因此目前的研究中主要采用基于文本相似性的自动化方法.

Natt 等人[119]基于向量空间模型计算顾客期望和产品需求之间的 cosine 相似性, 根据相似度的大小判断顾客期望和产品需求之间是否存在链接. 评估结果表明 63% 的链接的产品需求, 其所有的链接都可以在 10 个之内的建议的客户愿望的列表中找到, 并且可以节省 65% 的时间.

Heck 等人[115]分别使用 TF-IDF 和 LSA 构建特征请求的词向量, 然后计算特征请求之间的 cosine 相似性. 选择相似度值为前 50 的特征请求对, 认为两个特征请求之间相关. 实验结果对比发现 TF-IDF 有助于检测新的关联功能请求, 该方法可以计算特征请求功能网络, 用于未来添加新的功能请求.

Merten 等人[120]采用基于相似度的信息检索算法: (1) TF-IDF+cosine; (2) LSI+cosine; (3) BM25、BM25+、BM25L, 判断两个问题报告之间是否存在链接. 将问题报告的题目、描述、评论以及代码分别设置不同的权重进行对比. 采用人工构建的黄金准则进行评估. 实验结果表明对题目增加权重和对评论降低权重有很好的效果.

5.2.2 纵向跟踪

特征请求的纵向跟踪是指建立起特征请求与其他阶段的产物之间的联系. 特征请求的纵向跟踪有助于开发者了解项目的状态, 开发人员在维护特征请求时可以高效地定位软件各阶段对应的制品, 更加便捷地执行维护活动.

Licorish 等人[121]调研了安卓社区的用户请求, 开发者回应以及相关代码之间的关联. 以安卓社区的问题跟踪器上的问题报告为研究对象, 并挖掘安卓代码库, 调研代码更改与问题报告和发行公告中名词性短语之间的关系. 结果发现用户请求和发行公告之间有强相关性. 开发者的回应与对应的源代码更改之间有强相关性. 用户请求与源代码更改之间没有明显的联系. 结果说明安卓社区开发团队通过发行公告响应了用户的强烈的请求, 这些响应与他们实际实施的软件开发一致.

Seiler 等人[122]针对特征请求的管理提出一种 TAFT (tagging approach to support feature management) 方法, 该

方法严格记录软件特征和软件增强特征知识之间的合理跟踪. 该方法对软件的每一个特征使用一个标签, 然后将软件特征的描述, 需求和工作项与软件特征通过特征的标签联系起来. 该标签为特征提供了概述, 以及建立特征管理所需的跟踪的轻量级机制. 早期的评估表明 TAFT 可以是一个对于管理特征有益的、可行的方法. 但是软件特征的标签是人为分配和维护的, 找到一个有代表性的标签来描述一个特性, 难以解释不同的特性知识, 以及保证标签的可用性和一致性.

Palomba 等人[123]提出了一种 CRISTAL (crowdsourcing reviews to support apps evolution) 工具, 首先他们采用 AR-Miner[99]从用户评论中筛选出有用信息, 然后提取在用户评论提出到下一版本发布之间的问题报告和提交记录. 使用非对称的 Dice 相似性系数, 分别构建用户评论与提交记录, 用户评论与问题报告, 以及问题报告与提交记录之间的链接. 将用户评论与相关联的问题报告的提交记录之间创建链接. 链接建立完成后, 就可以用来跟踪开发人员是否实现了用户评论, 也可以追踪用户对已处理的评论的反应.

特征请求的纵向跟踪主要是建立起特征请求与软件开发其他阶段的构件之间的关联, 如代码等[115]. 有助于需求分析师评估需求变更时的产生的影响, 包括需要修改的组件、文档等, 评估需求变更的风险, 明确执行需求变更需要执行的任务[73].

5.3 特征需求变更管理

在软件开发过程中, 需求变更是难以避免的. 后期频繁的需求变更会给开发团队带来巨大的工作量. 在美国权威的第三方机构 Standish Group 的 CHAOS 报告中, 将需求变更列为困扰软件开发团队、导致项目失败的 5 大原因之一[124]. 变更管理的目的是为了控制变更, 以减少变更对开发工作的影响[118]. 良好的特征管理可以有效地减少特征请求管理的工作量[122]. 从本质上看, 特征请求的变更管理依赖于需求跟踪. 需求跟踪建立需求与开发代码、开发人员、开发文档等之间的联系, 该关联关系可以用于后续的变更管理.

通常一个特征请求被提交后, 需求分析师首先分析变更请求的必要性和合理性, 以决定是否接受该请求. 对于接受的特征请求, 需求分析师会为其分配开发者, 开发者定位其关联的构件等, 修改构建代码, 最后发行新版本. 根据此过程, 可以将对特征请求的变更管理分为 4 种任务, 即: 接受预测、推荐开发者、定位代码和代码推荐. 图 15 表示了几种任务之间的关系. 每种任务的右侧是在现有研究中所参考的背景知识. 接下来将从这 4 种任务的角度对现有工作进行调研.

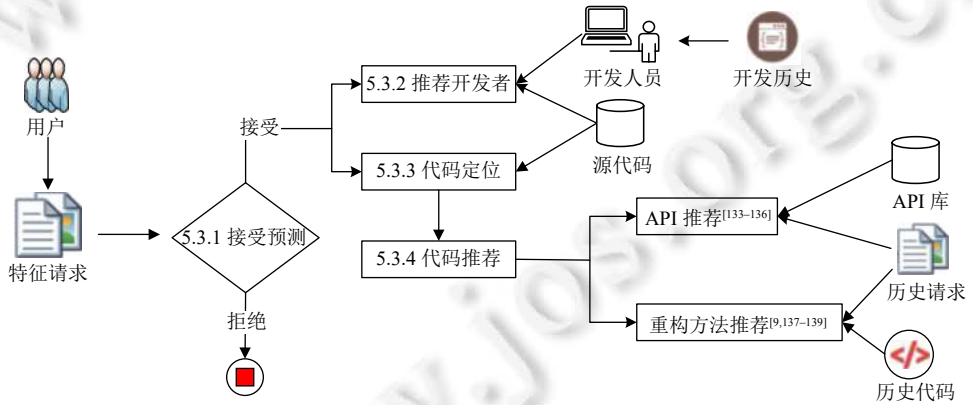

图 15 变更管理

5.3.1 接受预测

接受预测任务就是需求分析师分析特征请求的必要性和合理性以确定特征请求是否会被接受并开发. 用户提出的特征请求代表了用户的需求, 应用程序的开发者应该确保有价值的用户评论在合理的时间内得到及时的处理. Erne 等人[125]提出只有当开发者更快地回应用户评论时, 用户评论主导的创新的影响才是显著的. 因此, 开发者要在短时间内确定要开发的特征请求, 并对其进行开发.

现有研究中为预测特征请求是否会被接受, 普遍采用基于机器学习的分类方法, 将特征请求向量化并通过分类模型将特征请求分类为接受和拒绝. 不同的方法的区别在于分类模型的不同以及提取的文本特征的不同.

熊文军等人[126]提出一种基于 LR 模型的自动化预测一条变更请求 (包含缺陷报告和特征请求) 是否会被处理的方法. 该方法基于变更请求的标题长度、内容行数、含有附件数、开发者参与讨论次数以及等待生存时间等指标进行预测.

Nizamani 等人[127]提出了一种基于 MNB 的预测增强报告能否被接受的自动化模型. 并与不同的机器学习算法 (NB、SVM、RF、LR) 进行对比, 最终基于 MNB 的分类器准确率高达 89.25%. 在 Nizamani 等人[127]研究的基础上, Umer 等人[128]使用情感分类 Senti4SD 首先分析增强报告的情感, 然后将情感分析结果作为辅助特征, 结合 Umer 等人的模型进行预测. 实验结果表明加入情感分析作为辅助特征, 预测准确率有显著提升.

Arshad 等人[129]提出一种基于深度学习的技术, 使用 Word2Vec 和基于深度学习的分类器学习增强报告的单词之间的句法和语义信息, 结合情感分析作为辅助特征, 自动化地将增强报告分类为接受和拒绝. 实验结果表明该方法优于 Nizamani 等人[127]和 Umer 等人[128]的方法.

在开发过程中遇到故障的成本要远远高于在需求阶段发现故障的成本, 尽早发现特征请求中存在的故障, 尽早解决, 有助于有效地节约成本[130,131]. Fitzgerald 等人[130,131]提出一种利用机器学习分类算法预测软件故障 (failure) 的方法, 总结了特征请求的 13 种属性作为输入, 采用 DT、NB、LR 以及 M5P-tree 作为分类算法, 预测特征请求是否会造成软件故障. 实验结果发现 M5P-tree 和 LR 取得最好的表现结果. 特征请求的 BOW 和 TF-IDF 向量化特征在预测故障类型上表现的比较好, 说明讨论的内容比其他的属性要可靠.

为了减轻需求工程师的工作负担, 研究提出基于机器学习分类模型自动预测特征请求是否会被接受, 或者是否会造成软件故障. 然而由于特征请求涉及软件方方面面的特征, 已不能仅依赖关键词作为预测模型的输入特征. 基于 Bugzilla 数据的研究发现, 情感分析可以显著提升预测模型的效果[127−129]. 对基于 Google Play 的用户评论的研究发现负面词汇对于预测用户评论是否会被修复并没有效果. 在实际研究中, 情感分析的有效性有待实际验证. Heppler 等人[132]经过调研发现来自开发人员的增强报告成功实现的可能性是来在外部人员请求的两倍. 表 7 中总结了现有研究提出的预测模型中采用的分类特征以及分类模型, 从表中可以看出文本描述是最常用的特征, 此外情感分类也常被用于分类特征. 早期研究主要采用基于传统机器学习的分类模型, 2021 年深度模型首次被用于接受预测. 预测特征请求可能产生的软件故障只出现在早期研究中, 近年来的研究主要集中于预测特征请求是否会被接受. 在未来的研究中可以调研文本之外的其他特征, 对于预测特征请求能否被接受的作用, 并合理地组合有效的特征.

表 7 预测模型采用的特征

文献	采用的分类特征	分类模型	预测目标	年份
[126]	标题长度, 内容行数, 含有附件数, 开发者参与讨论次数, 等待生存时间	LR	接受预测	2017
[127]	文本描述	MNB		2018
[128]	文本描述, 情感分类	SVM		2019
[129]	文本描述, 情感分类	CNN		2021
[130,131]	文本描述参与讨论的人数, 提出者发布帖子的数量	M5P-tree, LR	软件故障	2011, 2012

5.3.2 推荐开发者

已被接受的特征请求, 需求分析师需要分配合适的开发人员去开发实现. 而不同的开发人员的技能有所不同. 为了提高特征请求的开发效率, 降低失败的可能性, 分配合适的开发人员显得尤为重要. 为特征请求推荐开发者主要基于假设“代码的原作者或者有类似需求开发经验的开发者应该是最佳推荐人选”. 因此现有研究方法可以归为两类: (1) 直接从所有开发者中挑选与特征请求最为接近的开发者, 并根据其相似性为候选开发者排序[133]; (2) 筛选与特征请求最为接近的源代码和历史开发记录, 获取其开发人员作为候选开发者, 并根据相似性等为候选开发者排序[134−139]. 或者同时结合二者[140,141]. 常见方法的框架图如图 16 所示.

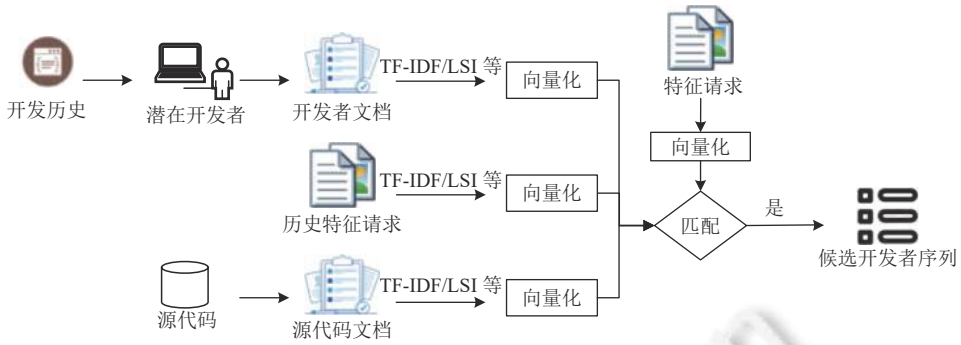

图 16 推荐开发者方法概览

Canfora 等人[133]提出一种为变更请求推荐最佳候选开发者的方法. 该方法假设最佳候选开发者是过去解决过类似变更请求的开发人员. 根据开发者解决过的变更请求的描述, 以及在 CVS 提交操作中的注释为开发者创建文档. 新提出的变更请求作为查询, 基于匹配算法与所有开发者的文档进行匹配, 从而得到一个候选开发者推荐列表. 但是该方法需要基于成熟软件大量的历史数据, 而没有大量历史记录的软件, 将难以得到准确的推荐结果.

Kagdi 等人[134,135]基于“过去对源代码中某一特定部分做出实质性修改的开发者, 很可能对它当前或未来的变更提供最佳协助”这样的假设, 通过提取每个源代码单元 (文件、类和方法) 中的注释和标识符, 为每个源代码单元创建相应的文档, 该文档通过 LSI 转化为向量作为源代码单元的索引. 将新提出的变更请求作为查询, 计算查询与语料库中文档的语义相似度, 从而获取与查询最为相似的相关源代码单元的推荐列表. 然后计算相关源代码的开发者曾经提交的代码向量与文件变化向量之间的欧氏距离来衡量开发者的贡献. 依据开发者的贡献生成变更请求开发者的推荐列表. 经过在开源项目上的评估, 所提出的技术能够以相当高的精度识别相关的开发人员, 为团队负责人或开发人员在维护任务中提供有价值的帮助.

Linares-Vásquez 等人[136]基于“源代码的作者是最有能力处理其变更的人”的假设, 首先从软件构建一个语料库, 基于源代码单元 (文件、类以及方法) 创建一个对应的文档, 然后使用 LSI 为语料库中的文档创建索引, 每个文档对应一个向量. 然后将变更请求的描述作为查询, 计算查询与语料库中文档的相似度排序. 获取相似度最高的几个文档, 得到源代码. 将源代码转化为 srcML 表示, 并从头注释中提取源代码的开发者. 从所有开发者中选出出现频率最高的 m 个开发者作为变更请求的推荐列表. 与其他方法不同, 该方法不需要挖掘软件仓库, 也不需要从历史问题报告中训练. 实验结果表明该方法优于已有的对比方法.

Hossen 等人[137]提出一种 iMacPro 方法, 为新提交的变更请求分配具有专业技能的最合适的开发者. 该方法首先利用 LSI 计算变更请求与源代码单元 (文件、方法或类) 之间的相似性, 为变更请求匹配相似的源代码单元. 将检索到的相关源代码单元根据历史维护中的变更倾向进行排序, 然后提取源代码的开发者和维护者, 将开发者和维护者均作为推荐的开发者. 实验结果表明该方法在召回率上比 iA[136]和 iAcPro 方法更好, recall 有 30% 到 180% 的提升.

Zanjani 等人[138]提出一种 iHDev 方法. 该方法从软件中创建一个语料库, 解析源代码单元 (文件、类和方法), 提取标识符、注释和表达式, 为每个源代码单元生成文档. 预处理源代码单元文档, 并使用 TF-IDF 向量化, 作为源代码单元的索引. 然后使用 KNN 搜索与变更请求最相似的 k 个文件, 采用余弦相似性来衡量两个文件向量的相似度. 分析上述产生的源代码单元的交互历史, 并推荐最有经验和处理这些单元方面有重大贡献的开发人员的列表. 该方法与 xFinder[135]和 iMacPro[137]对比, 总体结果表明, iHDev 在召回率和 MRR 方面的表现普遍优于 xFinder 和 iMacPro. 使用交互历史通常会导致准确性的提高, 不仅能更经常地识别出正确的开发者, 而且在推荐的候选人名单中也有更为准确的位置.

Cavalcanti 等人[139]提出一种半自动化的变更请求分配策略. 该方法整合了一个基于规则的专家系统 (RBES) 和一个机器学习模型. 前者用于实现组织特定的分配规则, 后者使用历史信息来匹配开发人员和变更请求. 在第 1 阶段, 输入变更请求和所有可用开发者, 基于规则进行分配. 如果第 1 个阶段不起作用, 则第 2 阶段, 应用机器学习模型找到过去解决过类似变更请求的开发者, 执行新的 RBES. 这一阶段, 使用不同的变量: 可用性、开发人员的

工作量和经验、变更请求的严重程度、重新分配模式等, 与建议的开发者列表结合. 如果第 1、第 2 阶段均无用, 就进入第 3 阶段: 分析员手动分配变更请求. 实验结果表明该方法比单纯的机器学习方法表现要好. 使用规则可以提高准确性.

根据开发人员的专业程度、开发历史与特征请求的匹配程度推荐开发者的方法可以为项目选出最有经验的、最合适的开发者. 然而这种方法偏向于选择开发经验丰富的开发者, 对于开发经验少的新手不友好. 因此, Yang 等人[140]和 Sun 等人[141]提出一种 EDR_SI 方法, 该方法不仅考虑开发人员的专业性和开发习惯, 并为开发人员提供了个性化源代码文件, 开发网络以及源代码更改历史等作为附加信息, 以便于开发人员参考. 首先预处理问题报告和历史提交记录, 然后通过问题报告与提交记录之间的相似性, 分析与问题报告相关的提交记录, 从提交记录中提取作者信息, 作为问题报告的潜在开发者. 在获取潜在开发者的同时, 获取与提交记录相关的源代码更改. 最后, 将这些源代码信息和作者信息作为输入, 使用协同主题模型 (collaborative topic modeling, CTM) 来为每一个候选开发者推荐相关的源代码文件, 以便在开发时使用.

在实际中, 开发人员的选择是多方面因素权衡利弊的结果, 如: 开发人员的时间、技能、对项目的熟悉度以及任务的难易程度等. 为难度低的任务推荐最有经验的年长的开发者, 可以加速任务完成进度, 但是会造成人力资源的浪费. 仅根据开发经验推荐开发者, 容易给有经验的开发者分配过多的任务, 不能合理给新手开发者锻炼机会. 因此在未来的推荐算法中, 应该更好地权衡多方面的因素. 如 Claytor 和 Servant[142]提出理解开发人员的业余技能, 并将此运用于开发者推荐. 以期达到项目和人员的合理分配安排, 实现成本最小化, 收益最大化.

5.3.3 代码定位

特征请求一般是在现有程序的基础上提出新的功能请求或者对现有功能的改进. 实现特征请求需要追溯源代码, 并做相应修改. 确定特征请求对应的源代码的位置, 称为代码定位. 目前普遍采用的方法是计算特征请求与源代码类之间的相似性, 相似度高的代码作为与请求对应的或需要修改的代码.

Palomba 等人[143]提出一种 CHANGEADVISOR 方法, 使用 ARdoc[40]方法分析用户评论的句子结构、语义和情感, 从软件维护角度提取有用的用户反馈, 然后分别采用 LDA、LDA-GA (the application of genetic algorithms to LDA) 和 HDP (hierarchical Dirichlet process) 对用户反馈进行聚类. 测量用户反馈簇与源代码类之间的非对称 Dice 相似度系数. 将与用户反馈簇相似度超过一定阈值的源代码根据相似度排序作为有序推荐列表. 研究结果表明, CHANGEADVISOR 在聚类相似的用户变更请求和识别变更影响的代码组件方面有较高的准确性. 将用户反馈簇与源代码构建关联的精度和召回率都比基线方法更准确.

Ciurumelea 等人[144]提出一种 URR 方法分析用户评论, 并为用户评论推荐需要修改的代码位置. 该方法首先分析用户评论, 基于 GBRT (gradient boosted regression trees) 将用户评论进行 6 个高层次和 12 个低层次两种粒度的分类. 然后为源代码单元 (文件、类和方法) 定义一套结构类别以及相应的规则识别源文件, 为每个源文件创建一个结构类别作为其索引. 完成索引后, 使用 Apache lucene API 进行相关性搜索, 并返回得分最高的源文件.

Kato 等人[145]基于 Web 集成开发环境提出一种新的交互设计 user-generated variables (UGV), 允许用户在不阅读或理解源代码的情况下, 在现有程序中声明要调整的参数, 然后将其转化为源代码相关部分的变量声明.

5.3.4 代码推荐

目前针对特征请求推荐代码的研究可以分为推荐 API 或者推荐重构方法. 一种方法是学习历史的特征请求采用的 API 或者重构记录, 从而预测新的推荐方法. 另一种方法是对比特征请求与 API 或者重构方法的相似性, 基于相似性推荐.

Thung 等人[146]提出一种自动化的方法, 为特征请求推荐开发者可以使用的 API 方法. 该方法从其他软件系统的变化记录中学习, 并将特征请求的文本描述与各种 API 方法的文本描述进行对比. 在 500 个特征请求上进行评估, 实验结果表明, 该方法可以基于 10 个库为特征请求推荐正确的方法.

Xu 等人[147]提出一种 MULAPI (method usage and location for API) 方法, 首先计算新特征请求与源代码文件和历史特征请求之间的相似性来获取特征相关的文件. 然后计算 API 文档与推荐的特征相关文件之间的相似性, 获取推荐的 API 方法. 接下来计算 API 文档与特征请求之间的相似性, 获取推荐的 API 方法. 新特征请求与历史

特征请求之间的相似性获取依据历史特征请求推荐的 API 方法. 将这 3 种推荐的 API 方法赋予不同的权重作为最终的推荐方法. Sun 等人[148]对 Xu 等人[147]的方法进行提升, 将参数从 17 个减少到 2 个, 降低计算的复杂度, 并采用 CNN_NPL 计算相似性来提升准确度. 该改进可以将 MULAPI 的 Hit@5 和 Hit@10 分别提升 29.7% 和 9.6%.

Li 等人[149]对 API 推荐的效用进行评估, 结果显示匹配的 API 严重重叠, 10% 的 API 构成了 80% 以上的匹配结果. 他们提出, 在衡量 API 推荐的正确性时, API 的效用也应该被考虑.

为了提高软件的可维护性, 在提出新的软件需求时, 需要对软件进行重构. Niu 等人[150]提出基于需求跟踪关系定位特征请求对应的代码位置, 判断代码的味道 (smell), 根据代码味道确定重构方法. Nyamawe 等人[9,151]提出一种为特征请求推荐重构的方法. 该方法首先识别过去实现特征请求的重构方法, 然后训练一个二分类器用于判断特征请求是否需要重构, 再基于特征请求与重构的历史记录训练 SVM 多标签分类器, 用于为特征请求推荐重构方法.

重命名是最经常和最广泛使用的重构操作之一. Nyamawe 等人[152]提出一种新的方法, 通过特征请求预测重命名的机会. 这篇文章基于假设: 如果源代码语义发生了改变, 那么它对应的源代码名字也应该被修改. 因此, 基于被修改的源代码和相关的特征请求之间的相似性来认定是否需要重命名. 分别预处理特征请求和源代码, 使用 TF-IDF 向量化并计算两个向量之间的相似性. 相似性超过一定阈值, 认为是需要重命名的. 实验结果表明, 该方法能够以平均 66% 的精准度和 72% 的召回率识别标识符重命名的需要.

6 特征请求公开研究资源

6.1 特征请求处理工具

为便于更好地将特征请求应用于软件开发, 许多工具被提出. 在本节中对此汇总整理, 以便后续研究人员参考利用. 表 8 中总结了现有研究中提出的特征请求处理工具、使用场景的简单描述、研究主题以及研究的年份等信息.

表 8 特征请求应用工具

工具名称	描述	研究主题	文献	年份
MARA	一个从在线评论中自动化获取移动应用特征请求的工具	获取	[13]	2013
AR-Miner	从用户评论中提取有用的信息, 然后主题模型技术分组, 最后根据分组结果组内排序和组间排序	获取, 分类, 优先级排序	[99]	2014
CRISTAL	将用户反馈与提交记录之间建立链接, 并根据链接分析对开发过程的影响	纵向跟踪	[123]	2015
SURF	对用户评论进行句子级别分类, 识别用户意图, 总结评论主题	获取, 生成摘要	[41]	2016
ARdoc	从应用程序中提取有用反馈	获取	[40]	2016
DECA	自动分类开发邮件, 识别句子意图	获取	[49]	2016
CLAP	分类评论, 聚类, 优先级排序	分类, 聚类, 优先级排序	[83]	2017
CHANGEADVISOR	分析用户评论并从中提取有用的用户反馈, 对用户评论分组, 然后根据建议的软件修改确定要维护的代码工件	获取, 分组, 代码定位	[143]	2017
URR	分析用户评论, 分类并推荐要修改的代码	分类, 代码定位	[144]	2017
MARC2.0	先分类应用评论, 然后生成摘要	分类, 生成摘要	[89]	2018
MULAPI	根据历史特征请求推荐API方法	API推荐	[147]	2018
REVSUM	从用户评论中识别缺陷报告和特征请求, 基于SAFE从评论中自动提取特征	获取, 特征抽取	[44]	2019
Ticket Tagger	将GitHub的issue标记为缺陷报告和特征请求	获取, 特征抽取, 聚类相同主题	[31]	2019
SAFE	基于规则的用于从用户评论中提取特征的工具, 首先提取评论的主题, 然后根据意图分类, 将相同主题的句子聚类	获取, 特征抽取, 聚类相同主题	[77]	2019
NERO	自动理解和分析特征请求的语义, 并判断特征请求的smell	分析理解	[82]	2020

6.2 公开数据集

在需求工程研究领域, 最大的阻碍是公开数据集的匮乏. 由于软件需求是公司的重要资产, 普通的科研人员没

有权限获得并使用. 而基于开源的特征请求解决了这一科研难题. 数据集可以通过爬虫技术, 轻易地从开源平台上爬取. 但是不同的科研人员, 不同的研究任务, 往往采用不同的数据集. 在这一部分, 旨在总结不同任务下的公开数据集, 以便后继科研的复用. 表 9 中总结了现有的公开数据集及其主要描述.

表 9 公开数据集

研究主题	文献	链接	描述	年份
需求获取	[13]	http://goo.gl/cL00vd	从用户评论中获取特征请求	2013
	[18]	https://mast.informatik.uni-hamburg.de/app-review-analysis/		2015
	[20]	https://mast.informatik.uni-hamburg.de/app-review-analysis/		2016
	[42]	https://zenodo.org/record/56907#.X-AES8gzabg	从问题报告中获取特征请求	2016
	[50]	https://github.com/heulaoyoutiao/bugtag2		2019
	[75]	https://www.dropbox.com/sh/mzd6yu1n8nckaod/AABi4wnmf8VR_90dc6C706cxa?dl=0	从用户评论中提取软件特征	2016
	[80]	https://alt.qcri.org/semeval2014/task4/index.php?id=data-and-tools		2018
需求分析	[105]	https://goo.gl/4qLVcf	应用评论优先级排序	2015
	[100]	https://play.google.com/store/apps/details?id=com.google.android.maps.mytracks		2016
	[83]	https://dibt.unimol.it/reports/clap/		2017
需求管理	[123]	http://www.cs.wm.edu/semeru/data/ICSME15-cristal	用户评论与代码跟踪	2015
	[151]	https://github.com/nyamawe/FR-Refactor	推荐重构方法	2020
	[131]	http://sre-research.cs.ucl.ac.uk/Intueri/	特征请求早期失败预测	2012
	[143]	https://sites.google.com/site/changeadvisormobile/	特征请求代码定位	2017

7 机遇与挑战

随着软件项目的不断发展, 项目的特征请求规模不断扩大, 研究人员也愈发重视该领域的研究. 目前针对特征请求的研究主要集中在: 特征请求的获取、软件特征抽取、结构化分析、优先级排序、特征请求评审、特征请求跟踪以及变更管理, 如图 2 所示, 涵盖特征请求从提出到分析、开发、管理的整个流程. 由于特征请求不同于传统的软件需求, 其特点在于数量多, 人工分析费时费力. 目前的研究趋向于自动化处理特征请求. 本节总结了各个主题下的研究进展, 并为未来研究指明了方向.

7.1 特征请求获取

目前特征请求的来源广泛, 研究人员从用户评论、推特、开发邮件等多种渠道获得特征请求, 极大地丰富了特征请求的内容. 但随之而来的, 是特征请求质量良莠不齐, 需求知识偏向于碎片化、非结构化. 低质量的特征请求会增加分析、开发的成本. 因此, 在未来研究中应该努力寻找更高质量的特征请求来源, 高质量的需求可以为高质量软件奠定基石.

现有研究中从用户反馈中获取特征请求的方法主要包括基于无监督学习、半监督学习以及监督学习等. 基于无监督学习的方法主要依赖于规则匹配, 该方法可以取得较高的精准度, 但是需要人工定义抽取规则. 人工定义规则的过程要求高, 需要对特征请求有很深的理解, 而且抽取的规则的完备性难以得到验证, 很难以覆盖所有的情况, 并且需要消耗大量的人力成本. 目前普遍采用的方法是基于监督学习的方法, 通过模型自动学习类别特征, 减少大量的人力、物力和资源消耗但是基于监督学习的方法需要人工标记数据, 并且方法的准确性依赖于标注数据的数量和质量, 尤其是基于深度模型的方法, 需要基于大量的样本学习类别特征. 降低人工成本的同时, 取得较高的抽取准确度, 是未来研究方向之一. 半监督学习方法旨在通过选择覆盖特征最广泛的最优训练集, 来实现最好的预测效果. 选择最优训练集的采样策略显得尤为重要. 在实际应用中, 需要在人力消耗和精准度之间进行衡量取舍.

7.2 特征请求分析

在特征请求分析阶段, 目前的研究热点主要为软件特征抽取、结构化分析与优先级排序. 其中结构化分析包

括用户意图识别、分类和摘要生成.

现有的软件特征抽取任务主要依赖于自然语言处理的词性标注技术, 获得每个单词的词性, 然后提取名词、动词、形容词等. 然而, 基于这种方法提取出的软件特征只是特定词性单词的组合, 软件特征的可读性不高. 因此, 在未来的研究中可以致力于提升软件特征的可读性, 如基于序列标注的方法, 提取出特征请求中连续的软件特征描述, 以提高提取的精准度.

在特征请求分类研究中, 现有的分类研究的类别不相同, 缺乏统一的分类标准. 此外, 由于特征请求受软件的领域知识的影响, 分类模型效果较差. 未来可以构建特征请求领域的预训练模型, 或者加入领域知识以提升模型的学习能力和分类水平.

特征请求的优先级排序是另一研究热点, 即如何从数量众多的特征请求中确定下一阶段要开发的用户需求, 不仅需要考虑用户请求, 还要结合开发成本, 项目收益等. 未来可以考虑提出量化标准, 用于评估特征请求的成本、预期收益以及用户的需求迫切程度, 以便于根据量化的数据判断优先级.

7.3 特征请求管理

从用户角度收集的特征请求质量参差不齐, 不利于特征请求的分析、开发以及管理. 因此, 未来可以针对提升特征请求的质量展开研究. 现有研究主要依赖于构建质量模型, 依据质量模型评审特征请求的质量. 现有研究自动化识别特征请求中的重复、冗余、歧义等问题, 但是缺乏其他角度的质量检查, 比如完整性、统一性, 缺乏量化的标准评价特征请求的质量. 此外, 目前仅能识别出低质量的特征请求, 无法自动化提升低质量的特征请求, 未来可以考虑自动化提升特征请求的质量.

在特征请求的横向跟踪研究中, 常见方法是计算不同的特征请求之间的相似性, 以识别重复、冗余以及参考等关系. 由于特征请求知识的碎片化, 并且不同特征请求之间并不是绝对独立的, 有可能存在依赖关系等. 因此构建特征知识网络, 增加可溯性工具支持非常重要, 可以构建特征树来记录现有特征. 由于特征请求语言描述的多样性, 同一个软件特征可能由不同的词汇描述, 用户可能使用不同的句法结构描述相同的特征请求, 导致文本相似度难以识别. 因此未来研究方法, 可以从词法、语法和句法的角度对比文本相似性, 以提高识别的准确性. 此外, 基于文本相似性的方法仅能识别描述同一个软件特征的特征请求, 但是无法进一步细化特征请求的关系为重复、依赖还是冲突. 若能对该问题提出有效的解决方法, 将进一步提升特征请求横向跟踪的建立效率.

特征请求的纵向跟踪主要建立特征请求与不同构件 (如, 源代码单元、开发者以及后续的软件开发过程的构件) 之间的连接, 可以极大地便利开发者推荐以及代码定位等任务. 然而特征请求与其他开发阶段的中间件, 如测试文档、测试用例以及后续的发行说明、用户手册等之间的链接也是未来研究的方向之一. 良好的特征请求跟踪关系网络对于变更管理起着至关重要的作用.

现有关于特征请求管理的研究主要可以分为: 接受预测、推荐开发者、定位代码以及代码推荐. 由于人工判断是否接受特征请求费时费力, 因此现有研究主要采用机器学习分类算法预测特征请求是否被接受. 在类别上, 预测的类别为接受或者拒绝. 拒绝的原因可能包括重复、冗余、冲突、表述有歧义等. 但是现有研究无法预测拒绝的具体研究. 因此未来研究可以进一步细化预测的类别. 在预测的方法上, 现有研究主要采用传统机器学习分类算法, 结合人工定义的特征构建分类模型. 未来可以发掘更多深度分类模型的应用, 并结合更多的特征, 如特征请求的作者是否为开发者等信息. 在开发者推荐方面, 现有算法倾向于为所有任务分配给有开发经验的开发者, 而不考虑任务的难易程度, 导致能者多劳, 新手开发者缺乏锻炼的机会. 因此未来的开发者推荐算法应该权衡更多的因素, 如: 开发人员的时间、技能、对项目的熟悉度、任务的难易程度等, 合理分配安排, 争取实现成本最小化、收益最大化. 为开发者推荐特征请求涉及的代码定位以及推荐相关代码的研究主要适用于成熟项目, 需要有大量的代码积累以及历史特征请求. 对于小规模项目以及开发资源积累不足的项目不再适用.

尽管目前针对特征请求的研究很多, 涉及获取、分析以及管理的方方面面, 但是特征请求建模、规格化与验证方面的研究几近空白. 特征请求知识碎片化、非结构化, 如何从众多的特征请求中, 识别不同请求之间的依赖关系、构建模型等, 是未来的研究难点之一. 如何将众多的特征请求转化为规范的、结构良好的需求规格说明书, 是未来需要解决的问题. 如何验证特征请求, 确保其符合用户的期望, 也有待研究.

整体看来, 现有相关研究的迅猛发展, 离不开自然语言处理技术的大力支持. 自然语言处理技术与机器学习技术的发展极大地激发了特征请求分析处理的自动化. 词性标注技术可以为特征请求中的每个单词标注其词性, 因此通常被用于提取描述特征的词组. 情感分类技术识别用户对一条特征请求的情感态度, 可以用于优先级排序、接受预测、竞品分析等任务. 基于无监督的聚类方法可以在不需要标注数据的前提下将相似的特征请求聚为一簇, 以便于结构化管理. 文本分类被广泛应用于各种分类任务, 如: 从用户反馈中提取特征请求、特征请求评审、接受预测、特征请求分类、优先级排序、跟踪、推荐开发者、推荐代码定位以及代码推荐等任务. 此外, 文本相似性技术也被广泛地应用于聚类分析、横向以及纵向跟踪的建立. 一方面, 这些自动化分析技术依赖于文本向量化, 准确的向量化标准是这些方法可靠性的保证, 因此未来针对特征请求的研究可以提出一种软件工程领域的向量化方法. 另一方面, 近年来深度神经网络技术的发展, 使得各种任务取得更高的准确度. 然而, 机器学习往往需要从大量标注数据中学习特征. 因此标注数据集限制了目前机器学习在特征请求分析与处理中的发展. 在需求工程领域, 缺乏大规模、可靠的数据集是目前公认的研究难题, 为了解决这一研究难题, 研究人员可以公开数据集, 共享资源. 同时也可以寻求主动学习、迁移学习等方法降低对大量数据集的依赖.

若能针对上述研究现状对特征请求的研究进行提升和补全, 将能进一步提升特征请求的质量以及提升特征请求的利用价值, 从而提升软件质量.

8 总结与展望

随着软件项目及其用户评论数量的指数增长, 用户评论受到研究人员的关注并将其应用于软件开发. 本文对121 篇用户特征请求分析与处理相关研究工作做了综述调研. 在传统需求工程框架下, 本研究调研了特征请求的不同来源, 总结并分析了需求获取、需求分析以及需求管理等不同需求活动下的研究进展. 针对需求工程研究领域缺乏公开数据集的公认短板, 还汇总了现有研究中提出的特征请求处理工具以及相关研究中的公开数据集. 最后对所有研究进行总结并提出其研究机遇与挑战. 未来研究中, 我们将持续关注特征请求领域相关的研究, 并及时更新综述内容. 此外, 本文只对特征请求进行了综述调研, 在未来将会融入缺陷报告相关的研究, 在需求工程视角下探究特征请求与缺陷报告分析与处理的异同. 同时, 目前针对特征请求规格化与验证的研究仍有较大缺口, 未来我们也会在该主题下进行深入探究.

References:

[1] Martin W, Sarro F, Jia Y, Zhang YY, Harman M. A survey of App store analysis for software engineering. IEEE Trans. on Software Engineering, 2017, 43(9): 817–847. [doi: 10.1109/TSE.2016.2630689]

[2] Cavalcanti YC, da Mota Silveira Neto PA, do Carmo Machado I, Vale TF, de Almeida ES, de Lemos Meira SR. Challenges and opportunities for software change request repositories: A systematic mapping study. Journal of Software: Evolution and Process, 2014, 26(7): 620–653. [doi: 10.1002/smr.1639]

[3] Tavakoli M, Zhao LP, Heydari A, Nenadić G. Extracting useful software development information from mobile application reviews: A survey of intelligent mining techniques and tools. Expert Systems with Applications, 2018, 113: 186–199. [doi: 10.1016/j.eswa.2018.05.037]

[4] Wang C, Daneva M, van Sinderen M, Liang P. A systematic mapping study on crowd sourced requirements engineering using user feedback. Journal of Software: Evolution and Process, 2019, 31(10): e2199. [doi: 10.1002/smr.2199]

[5] Bakar NH, Kasirun ZM, Salleh N. Feature extraction approaches from natural language requirements for reuse in software product lines: A systematic literature review. Journal of Systems and Software, 2015, 106: 132–149. [doi: 10.1016/j.jss.2015.05.006]

[6] Wohlin C. Guidelines for snowballing in systematic literature studies and a replication in software engineering. In: Proc. of the 18th Int'l Conf. on Evaluation and Assessment in Software Engineering. London: ACM, 2014. 38. [doi: 10.1145/2601248.2601268]

[7] Herzig K, Just S, Zeller A. It's not a bug, it's a feature: How misclassification impacts bug prediction. In: Proc. of the 35th Int'l Conf. on Software Engineering. San Francisco: IEEE, 2013. 392–401. [doi: 10.1109/ICSE.2013.6606585]

[8] Hu MQ, Liu B. Mining and summarizing customer reviews. In: Proc. of the 10th ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining. Washington: ACM, 2004. 168–177. [doi: 10.1145/1014052.1014073]

[9] Nyamawe AS, Liu H, Niu N, Umer Q, Niu ZD. Automated recommendation of software refactorings based on feature requests. In: Proc.

of the 27th IEEE Int'l Requirements Engineering Conf. Jeju: IEEE, 2019. 187–198. [doi: 10.1109/RE.2019.00029]

[10] Merten T, Mager B, Hübner P, Quirchmayr T, Paech B, Bürsner S. Requirements communication in issue tracking systems in four Open-Source projects. In: Proc. of the 21st Int'l Conf. on Requirements Engineering: Foundation for Software Quality. Essen: CEUR-WS.org, 2015. 114–125.

[11] Kakar AK. Separating the wheat from the chaff: Extracting business value from feature requests posted in user forums. Journal of Organizational and End User Computing, 2016, 28(2): 8. [doi: 10.4018/JOEUC.2016040108]

[12] Jha N, Mahmoud A. Mining user requirements from application store reviews using frame semantics. In: Proc. of the 23rd Int'l Working Conf. on Requirements Engineering: Foundation for Software Quality. Essen: Springer, 2017. 273–287. [doi: 10.1007/978-3-319-54045-0_20]

[13] Iacob C, Harrison R. Retrieving and analyzing mobile Apps feature requests from online reviews. In: Proc. of the 10th Working Conf. on Mining Software Repositories. San Francisco: IEEE, 2013. 41–44. [doi: 10.1109/MSR.2013.6624001]

[14] Williams G, Mahmoud A. Mining twitter feeds for software user requirements. In: Proc. of the 25th Int'l Requirements Engineering Conf. Lisbon: IEEE, 2017. 1–10. [doi: 10.1109/RE.2017.14]

[15] Carreño LVG, Winbladh K. Analysis of user comments: An approach for software requirements evolution. In: Proc. of the 35th Int'l Conf. on Software Engineering. San Francisco: IEEE, 2013. 582–591. [doi: 10.1109/ICSE.2013.6606604]

[16] Lv HY, Fan K, Yang JL. Research on software feature mining for App user reviews. Library Theory and Practice, 2019(7): 106–112 (in Chinese with English abstract). [doi: 10.14064/j.cnki.issn1005-8214.2019.07.025]

[17] Guzman E, El-Haliby M, Bruegge B. Ensemble methods for App review classification: An approach for software evolution (N). In: Proc. of the 30th IEEE/ACM Int'l Conf. on Automated Software Engineering. Lincoln: IEEE, 2015. 771–776. [doi: 10.1109/ASE.2015.88]

[18] Maalej W, Nabil H. Bug report, feature request, or simply praise? On automatically classifying App reviews. In: Proc. of the 23rd Int'l Requirements Engineering Conf. Ottawa: IEEE, 2015. 116–125. [doi: 10.1109/RE.2015.7320414]

[19] Villarroel L, Bavota G, Russo B, Oliveto R, Di Penta M. Release planning of mobile Apps based on user reviews. In: Proc. of the 38th Int'l Conf. on Software Engineering. Austin: IEEE, 2016. 14–24. [doi: 10.1145/2884781.2884818]

[20] Maalej W, Kurtanović Z, Nabil H, Stanik C. On the automatic classification of App reviews. Requirements Engineering, 2016, 21(3): 311–331. [doi: 10.1007/s00766-016-0251-9]

[21] Peng ZL, Wang J, He KQ, Tang MD. An approach of extracting feature requests from App reviews. In: Proc. of the 12th Int'l Conf. on Collaborate Computing: Networking, Applications and Worksharing. Beijing: Springer, 2017. 312–323. [doi: 10.1007/978-3-319-59288-6_28]

[22] McIlroy S, Ali N, Khalid H, Hassan AE. Analyzing and automatically labelling the types of user issues that are raised in mobile App reviews. Empirical Software Engineering, 2016, 21(3): 1067–1106. [doi: 10.1007/s10664-015-9375-7]

[23] Nayebi M, Cho H, Ruhe G. App store mining is not enough for App improvement. Empirical Software Engineering, 2018, 23(5): 2764–2794. [doi: 10.1007/s10664-018-9601-1]

[24] Guo T, Bin G, Ouyang Y, Yu ZW. Mining and analyzing user feedback from app reviews: An econometric approach. In: Proc. of the 2018 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computing, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI). Guangzhou: IEEE, 2018. 841–848. [doi: 10.1109/SmartWorld.2018.00155]

[25] Stanik C, Haering M, Maalej W. Classifying multilingual user feedback using traditional machine learning and deep learning. In: Proc. of the 27th IEEE Int'l Requirements Engineering Conf. Workshops. Jeju: IEEE, 2019. 220–226. [doi: 10.1109/REW.2019.00046]

[26] Arya D, Wang WT, Guo JLC, Cheng JH. Analysis and detection of information types of open source software issue discussions. In: Proc. of the 41st Int'l Conf. on Software Engineering. Montreal: IEEE, 2019. 454–464. [doi: 10.1109/ICSE.2019.00058]

[27] Khan JA. Mining requirements arguments from user forums. In: Proc. of the 27th Int'l Requirements Engineering Conf. Jeju: IEEE, 2019. 440–445. [doi: 10.1109/RE.2019.00059]

[28] Khan JA, Liu L, Wen LJ, Ali R. Conceptualising, extracting and analysing requirements arguments in users' forums: The CrowdRE-Arg framework. Journal of Software: Evolution and Process, 2020, 32(12): e2309. [doi: 10.1002/smr.2309]

[29] Tizard J. Requirement mining in software product forums. In: Proc. of the 27th Int'l Requirements Engineering Conf. Jeju: IEEE, 2019. 428–433. [doi: 10.1109/RE.2019.00057]

[30] Xiao JM, Chen SZ, Feng ZY, Liu PL, Xue X. An automatic analysis of user reviews method for App evolution and maintenance. Chinese Journal of Computers, 2020, 43(11): 2184–2202 (in Chinese with English abstract). [doi: 10.11897/SP.J.1016.2020.02184]

[31] Kallis R, Di Sorbo A, Canfora G, Panichella S. Ticket tagger: Machine learning driven issue classification. In: Proc. of the 2019 Int'l

Conf. on Software Maintenance and Evolution. Cleveland: IEEE, 2019. 406–409. [doi: 10.1109/ICSME.2019.00070]

[32] Shi L, Xing MZ, Li MY, Wang YW, Li SB, Wang Q. Detection of hidden feature requests from massive chat messages via deep siamese network. In: Proc. of the 42nd Int'l Conf. on Software Engineering. Seoul: IEEE, 2020. 641–653.

[33] Aslam N, Ramay WY, Xia KW, Sarwar N. Convolutional neural network based classification of App reviews. IEEE Access, 2020, 8: 185619–185628. [doi: 10.1109/ACCESS.2020.3029634]

[34] Dhammajoti, Young JC, Rusli A. A comparison of supervised text classification and resampling techniques for user feedback in Bahasa Indonesia. In: Proc. of 5th Int'l Conf. on Informatics and Computing (ICIC). Gorontalo: IEEE, 2020. 1–6. [doi: 10.1109/ICIC50835.2020.9288588]

[35] Kallis R, Di Sorbo A, Canfora G, Panichella S. Predicting issue types on GitHub. Science of Computer Programming, 2021, 205: 102598. [doi: 10.1016/j.scico.2020.102598]

[36] Yadav A, Sharma R, Fard FH. A semantic-based framework for analyzing app users' feedback. In: Proc. of the 27th Int'l Conf. on Software Analysis, Evolution and Reengineering. London: IEEE, 2020. 572–576. [doi: 10.1109/SANER48275.2020.9054843]

[37] Song R, Li T, Ding ZM. Automatically identifying requirements-oriented reviews using a top-down feature extraction approach. In: Proc. of the 27th Asia-Pacific Software Engineering Conf. Singapore: IEEE, 2020. 450–454. [doi: 10.1109/APSEC51365.2020.00054]

[38] Dąbrowski J, Letier E, Perini A, Susi A. Finding and analyzing App reviews related to specific features: A research preview. In: Proc. of the 25th Int'l Working Conf. on Requirements Engineering: Foundation for Software Quality. Essen: Springer, 2019. 183–189. [doi: 10.1007/978-3-030-15538-4_14]

[39] Panichella S, Di Sorbo A, Guzman E, Visaggio CA, Canfora G, Gall HC. How can I improve my App? Classifying user reviews for software maintenance and evolution. In: Proc. of the 2015 Int'l Conf. on Software Maintenance and Evolution. Bremen: IEEE, 2015. 281–290. [doi: 10.1109/ICSM.2015.7332474]

[40] Panichella S, Di Sorbo A, Guzman E, Visaggio CA, Canfora G, Gall HC. ARdoc: App reviews development oriented classifier. In: Proc. of the 24th Int'l Symp. on Foundations of Software Engineering. Singapore: ACM, 2016. 1023–1027. [doi: 10.1145/2950290.2983938]

[41] Di Sorbo A, Panichella S, Alexandru CV, Shimagaki J, Visaggio CA, Canfora G, Gall HC. What would users change in my App? Summarizing App reviews for recommending software changes. In: Proc. of the 24th Int'l Symp. on Foundations of Software Engineering. Singapore: ACM, 2016. 499–510. [doi: 10.1145/2950290.2950299]

[42] Merten T, Falis M, Hübner P, Quirchmayr T, Bürsner S, Paech B. Software feature request detection in issue tracking systems. In: Proc. of the 24th Int'l Requirements Engineering Conf. Beijing: IEEE, 2016. 166–175. [doi: 10.1109/RE.2016.8]

[43] Di Sorbo A, Panichella S, Alexandru CV, Visaggio CA, Canfora G. SURF: Summarizer of user reviews feedback. In: Proc. of the 39th Int'l Conf. on Software Engineering Companion. Buenos Aires: IEEE, 2017. 55–58. [doi: 10.1109/ICSE-C.2017.5]

[44] Shah FA, Sirts K, Pfahl D. Using App reviews for competitive analysis: Tool support. In: Proc. of the 3rd Int'l Workshop on App Market Analytics. Singapore: ACM, 2019. 40–46. [doi: 10.1145/3340496.3342756]

[45] Tizard J, Wang HC, Yohannes L, Blincoe K. Can a conversation paint a picture? Mining requirements in software forums. In: Proc. of the 27th Int'l Requirements Engineering Conf. Jeju: IEEE, 2019. 17–27. [doi: 10.1109/RE.2019.00014]

[46] Li MY, Shi L, Yang Y, Wang Q. A deep multitask learning approach for requirements discovery and annotation from open forum. In: Proc. of the 35th Int'l Conf. on Automated Software Engineering. Melbourne: IEEE, 2020. 336–348.

[47] Huang Q, Xia X, Lo D, Murphy GC. Automating intention mining. IEEE Trans. on Software Engineering, 2020, 46(10): 1098–1119. [doi: 10.1109/TSE.2018.2876340]

[48] Di Sorbo A, Panichella S, Visaggio CA, Di Penta M, Canfora G, Gall HC. Development emails content analyzer: Intention mining in developer discussions (T). In: Proc. of the 30th IEEE/ACM Int'l Conf. on Automated Software Engineering. Lincoln: IEEE, 2015. 12–23. [doi: 10.1109/ASE.2015.12]

[49] Di Sorbo A, Panichella S, Visaggio CA, Di Penta M, Canfora G, Gall H. DECA: Development emails content analyzer. In: Proc. of the 38th Int'l Conf. on Software Engineering Companion. Austin: IEEE, 2016. 641–644.

[50] Zhang T, Li HM, Xu Z, Liu J, Huang RB, Shen YR. Labelling issue reports in mobile Apps. IET Software, 2019, 13(6): 528–542. [doi: 10.1049/iet-sen.2018.5420]

[51] Deocadez R, Harrison R, Rodriguez D. Preliminary study on applying semi-supervised learning to App store analysis. In: Proc. of the 21st Int'l Conf. on Evaluation and Assessment in Software Engineering. Karlskrona: ACM, 2017. 320–323. [doi: 10.1145/3084226.3084285]

[52] Dhinakaran VT, Pulle R, Ajmeri N, Murukannaiah PK. App review analysis via active learning: Reducing supervision effort without compromising classification accuracy. In: Proc. of the 26th Int'l Requirements Engineering Conf. Banff: IEEE, 2018. 170–181. [doi: 10.1109/RE.2018.00026]

[53] Shi L, Chen CL, Wang Q, Boehm B. Automatically detecting feature requests from development emails by leveraging semantic sequence mining. Requirements Engineering, 2021, 26(2): 255–271. [doi: 10.1007/s00766-020-00344-y]

[54] Mikolov T, Chen K, Corrado G, Dean J. Efficient estimation of word representations in vector space. In: Proc. of the 1st Int'l Conf. on Learning Representations. Scottsdale, 2013. [doi: 10.48550/arXiv.1301.3781]

[55] Pennington J, Socher R, Manning CD. GloVe: Global vectors for word representation. In: Proc. of the 2014 Int'l Conf. on Empirical Methods in Natural Language Processing. Doha: Association for Computational Linguistics, 2014. 1532–1543. [doi: 10.3115/v1/D14-1162]

[56] Lecun Y, Bottou L, Bengio Y, Haffner P. Gradient-based learning applied to document recognition. Proc. of the IEEE, 1998, 86(11): 2278–2324. [doi: 10.1109/5.726791]

[57] Lai SW, Xu LH, Liu K, Zhao J. Recurrent convolutional neural networks for text classification. In: Proc. of the 29th AAAI Conf. on Artifcial Intelligence. Austin: AAAI Press, 2015. 2267–2273.

[58] Hochreiter S, Schmidhuber J. Long short-term memory. Neural Computation, 1997, 9(8): 1735–1780. [doi: 10.1162/neco.1997.9.8.1735]

[59] Grave E, Bojanowski P, Gupta P, Joulin A, Mikolov T. Learning word vectors for 157 languages. In: Proc. of the 11th Int'l Conf. on Language Resources and Evaluation. Miyazaki: European Language Resources Association, 2018. [doi: 10.48550/arXiv.1802.06893]

[60] Devlin J, Chang MW, Lee K, Toutanova K. BERT: Pre-training of deep bidirectional transformers for language understanding. In: Proc. of the 2019 Conf. of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies. Minneapolis: Association for Computational Linguistics, 2019. 4171–4186. [doi: 10.18653/v1/N19-1423]

[61] Thelwall M, Buckley K, Paltoglou G. Sentiment strength detection for the social Web. Journal of the American Society for Information Science and Technology, 2012, 63(1): 163–173. [doi: 10.1002/asi.21662]

[62] Calefato F, Lanubile F, Maiorano F, Novielli N. Sentiment polarity detection for software development. Empirical Software Engineering, 2018, 23(3): 1352–1382. [doi: 10.1007/s10664-017-9546-9]

[63] Yarowsky D. Unsupervised word sense disambiguation rivaling supervised methods. In: Proc. of the 33rd Annual Meeting on Association for Computational Linguistics. Cambridge: Association for Computational Linguistics, 1995. 189–196. [doi: 10.3115/981658.981684]

[64] Wang J, Luo SW, Zeng XH. A random subspace method for co-training. In: Proc. of the 2008 IEEE Int'l Joint Conf. on Neural Networks (IEEE World Congress on Computational Intelligence). Hong Kong: IEEE, 2008. 195–200. [doi: 10.1109/IJCNN.2008.4633789]

[65] Yaslan Y, Cataltepe Z. Co-training with relevant random subspaces. Neurocomputing, 2010, 73(10–12): 1652–1661. [doi: 10.1016/j.neucom.2010.01.018]

[66] Bowring JF, Rehg JM, Harrold MJ. Active learning for automatic classification of software behavior. ACM SIGSOFT Software Engineering Notes, 2004, 29(4): 195–205. [doi: 10.1145/1013886.1007539]

[67] Rahimi M, Cleland-Huang J. Personas in the middle: Automated support for creating personas as focal points in feature gathering forums. In: Proc. of the 29th ACM/IEEE Int'l Conf. on Automated Software Engineering. Vasteras: ACM, 2014. 479–484. [doi: 10.1145/2642937.2642958]

[68] Gribkov EI, Yekhlakov YP. Neural network model for user request analysis during software operations and maintenance phase. Business Informatics, 2020, 14(1): 7–18. [doi: 10.17323/2587-814X.2020.1.7.18]

[69] Shi L, Chen CL, Wang Q, Li SB, Boehm B. Understanding feature requests by leveraging fuzzy method and linguistic analysis. In: Proc. of the 32nd IEEE/ACM Int'l Conf. on Automated Software Engineering. Urbana: IEEE, 2017. 440–450. [doi: 10.1109/ASE.2017.8115656]

[70] Hoon L, Vasa R, Schneider JG, Grundy J. An analysis of the mobile App review landscape: Trends and implications. Melbourne: Faculty of Information and Communication Technologies, Swinburne University of Technology, 2013.

[71] Noll J, Liu WM. Requirements elicitation in open source software development: A case study. In: Proc. of the 3rd Int'l Workshop on Emerging Trends in Free/Libre/Open Source Software Research and Development. Cape Town: ACM, 2010. 35–40. [doi: 10.1145/1833272.1833279]

[72] Gill KD, Raza A, Zaidi AM, Kiani MM. Semi-automation for ambiguity resolution in open source software requirements. In: Proc. of the 27th IEEE Canadian Conf. on Electrical and Computer Engineering. Toronto: IEEE, 2014. 1–6. [doi: 10.1109/CCECE.2014.6900955]

[73] Luo B, Ding EY. Requirements Engineering—Software Modeling and Analysis. Beijing: Higher Education Press, 2009. 194 (in Chinese).

[74] Kang KC, Cohen SG, Hess JA, Novak WE, Peterson AS. Feature-oriented domain analysis (FODA) feasibility study. Technical Report,

Pittsburgh: Software Engineering Institute, Carnegie Mellon University, 1990.

[75] Bakar NH, Kasirun ZM, Salleh N, Jalab HA. Extracting features from online software reviews to aid requirements reuse. Applied Soft Computing, 2016, 49: 1297–1315. [doi: 10.1016/j.asoc.2016.07.048]

[76] Bakiu E, Guzman E. Which feature is unusable? Detecting usability and user experience issues from user reviews. In: Proc. of the 25th Int'l Requirements Engineering Conf. Workshops (REW). Lisbon: IEEE, 2017. 182–187. [doi: 10.1109/REW.2017.76]

[77] Johann T, Stanik C, Alizadeh BAM, Maalej W. SAFE: A simple approach for feature extraction from App descriptions and App reviews. In: Proc. of the 25th Int'l Requirements Engineering Conf. Lisbon: IEEE, 2017. 21–30. [doi: 10.1109/RE.2017.71]

[78] Licorish SA, Savarimuthu BTR, Keertipati S. Attributes that predict which features to fix: Lessons for App store mining. In: Proc. of the 21st Int'l Conf. on Evaluation and Assessment in Software Engineering. Karlskrona: ACM, 2017. 108–117. [doi: 10.1145/3084226.3084246]

[79] Dalpiaz F, Parente M. RE-SWOT: From user feedback to requirements via competitor analysis. In: Proc. of the 25th Int'l Working Conf. on Requirements Engineering: Foundation for Software Quality. Essen: Springer, 2019. 55–70. [doi: 10.1007/978-3-030-15538-4_4]

[80] Shah FA, Sirts K, Pfahl D. Simulating the impact of annotation guidelines and annotated data on extracting App features from App reviews. In: Proc. of the 14th Int'l Conf. on Software Technologies. Prague: SciTePress, 2019. 384–396.

[81] Shah FA, Sirts K, Pfahl D. Is the SAFE approach too simple for App feature extraction? A replication study. In: Proc. of the 25th Int'l Working Conf. on Requirements Engineering: Foundation for Software Quality. Essen: Springer, 2019. 21–36. [doi: 10.1007/978-3-030-15538-4_2]

[82] Mu FW, Shi L, Zhou W, Zhang YZ, Zhao HX. NERO: A text-based tool for content annotation and detection of smells in feature requests. In: Proc. of the 28th Int'l Requirements Engineering Conf. Zurich: IEEE, 2020. 400–403. [doi: 10.1109/RE48521.2020.00056]

[83] Scalabrino S, Bavota G, Russo B, Di Penta M, Oliveto R. Listening to the crowd for the release planning of mobile Apps. IEEE Trans. on Software Engineering, 2019, 45(1): 68–86. [doi: 10.1109/TSE.2017.2759112]

[84] Li CY, Huang LG, Ge JD, Luo B, Ng V. Automatically classifying user requests in crowdsourcing requirements engineering. Journal of Systems and Software, 2018, 138: 108–123. [doi: 10.1016/j.jss.2017.12.028]

[85] Cleland-Huang J, Dumitru H, Duan C, Castro-Herrera C. Automated support for managing feature requests in open forums. Communications of the ACM, 2009, 52(10): 68–74. [doi: 10.1145/1562764.1562784]

[86] Tao CQ, Guo HJ, Huang ZQ. Identifying security issues for mobile applications based on user review summarization. Information and Software Technology, 2020, 122: 106290. [doi: 10.1016/j.infsof.2020.106290]

[87] Wang C, Zhang F, Liang P, Daneva M, van Sinderen M. Can App change logs improve requirements classification from App reviews?: An exploratory study. In: Proc. of the 12th Int'l Symp. on Empirical Software Engineering and Measurement. Oulu: ACM, 2018. 43. [doi: 10.1145/3239235.3267428]

[88] ISO. ISO/IEC 9126-1: 2001 Software engineering—Product quality—Part 1: Quality model. ISO, 2001.

[89] Jha N, Mahmoud A. Using frame semantics for classifying and summarizing application store reviews. Empirical Software Engineering, 2018, 23(6): 3734–3767. [doi: 10.1007/s10664-018-9605-x]

[90] Saaty TL. The Analytic Hierarchy Process: Planning, Priority Setting, Resource Allocation. New York: McGraw-Hill, 1980.

[91] Kano N, Seraku N, Takahashi F, Tsuji SI. Attractive quality and must-be quality. Journal of the Japanese Society for Quality Control, 1984, 14(2): 147–156. [doi: 10.20684/quality.14.2_147]

[92] Karlsson J, Wohlin C, Regnell B. An evaluation of methods for prioritizing software requirements. Information and Software Technology, 1998, 39(14–15): 939–947. [doi: 10.1016/S0950-5849(97)00053-0]

[93] Leffingwell D, Widrig D. Managing Software Requirements: A Unified Approach. Reading: Addison-Wesley, 2000.

[94] Beck K. Extreme Programming Explained. 7th ed., Reading: Addison-Wesley, 2000. 417–432.

[95] Tudor D, Walter GA. Using an agile approach in a large, traditional organization. In: Proc. of the 2006 AGILE Conf. Minneapolis: IEEE, 2006. 367–373. [doi: 10.1109/AGILE.2006.60]

[96] Avesani P, Bazzanella C, Perini A, Susi A. Facing scalability issues in requirements prioritization with machine learning techniques. In: Proc. of the 13th Int'l Conf. on Requirements Engineering. Paris: IEEE, 2005. 297–305. [doi: 10.1109/RE.2005.30]

[97] Sillitti A, Succi G. Requirements engineering for agile methods. In: Aurum A, Wohlin C, eds. Engineering and Managing Software Requirements. Berlin, Heidelberg: Springer, 2005. 309–326. [doi: 10.1007/3-540-28244-0_14]

[98] Laurent P, Cleland-Huang J. Lessons learned from open source projects for facilitating online requirements processes. In: Proc. of the 15th Int'l Working Conf. on Requirements Engineering: Foundation for Software Quality. Amsterdam: Springer, 2009. 240–255. [doi: 10.1007/978-3-642-02050-6_21]

[99] Chen N, Lin JL, Hoi SCH, Xiao XK, Zhang BS. AR-miner: Mining informative reviews for developers from mobile App marketplace.

In: Proc. of the 36th Int'l Conf. on Software Engineering. Hyderabad: ACM, 2014. 767–778. [doi: 10.1145/2568225.2568263]

[100] Keertipati S, Savarimuthu BTR, Licorish SA. Approaches for prioritizing feature improvements extracted from App reviews. In: Proc. of the 20th Int'l Conf. on Evaluation and Assessment in Software Engineering. Limerick: ACM, 2016. 33. [doi: 10.1145/2915970.2916003]

[101] Morales-Ramirez I, Muñante D, Kifetew F, Perini A, Susi A, Siena A. Exploiting user feedback in tool-supported multi-criteria requirements prioritization. In: Proc. of the 25th Int'l Requirements Engineering Conf. Lisbon: IEEE, 2017. 424–429. [doi: 10.1109/RE.2017.41]

[102] Etaiwi L, Hamel S, Guéhéneuc YG, Flageol W, Morales R. Order in chaos: Prioritizing mobile App reviews using consensus algorithms. In: Proc. of the 44th Annual Computers, Software, and Applications Conf. Madrid: IEEE, 2020. 912 –920. [doi: 10.1109/COMPSAC48688.2020.0-151]

[103] Liu MC, Zhan HF, Shi X, Wang YX. Analytical approach of importance of user's needs for product design. Machinery, 2020, 58(12): 1–9 (in Chinese with English abstract). [doi: 10.3969/j.issn.1000-4998.2020.12.002]

[104] Kakar AK, Smith R, Hale D, Hale J. IS Product enhancements: An exploration of two competing techniques for evaluation of user featiure requests. In: Proc. of the 7th Int'l Research Workshop on Information Technology Project Management. Orlando, 2012. 16–24.

[105] Licorish SA, Tahir A, Bosu MF, MacDonell SG. On satisfying the Android OS community: User feedback still central to developers' portfolios. In: Proc. of the 24th Australasian Software Engineering Conf. Adelaide: IEEE, 2015. 78–87. [doi: 10.1109/ASWEC.2015.19]

[106] Li H, Yan DH. Analysis of software requirements management based on project requirements engineering theory. China Science and Technology Information, 2011(16): 92–93 (in Chinese with English abstract).

[107] Wiegers KE. Software Requirements. 2nd ed., Redmond: Microsoft Press, 2003.

[108] Heck P, Zaidman A. Quality criteria for just-in-time requirements: Just enough, just-in-time? In: Proc. of the Workshop on Just-in-time Requirements Engineering. Ottawa: IEEE, 2015. 1–4. [doi: 10.1109/JITRE.2015.7330170]

[109] Heck P, Zaidman A. A framework for quality assessment of just-in-time requirements: The case of open source feature requests. Requirements Engineering, 2017, 22(4): 453–473. [doi: 10.1007/s00766-016-0247-5]

[110] Aversano L. Issue Reports analysis in enterprise open source systems. In: Proc. of the 21st Int'l Conf. on Enterprise Information Systems. Heraklion: SciTePress, 2019. 337–344. [doi: 10.5220/0007757803370344]

[111] Shi L, Chen CL, Wang Q, Boehm B. Is it a new feature or simply "don't know yet"? On automated redundant OSS feature requests identification. In: Proc. of the 24th Int'l Requirements Engineering Conf. Beijing: IEEE, 2016. 377–382. [doi: 10.1109/RE.2016.65]

[112] Heck P, Zaidman A. An analysis of requirements evolution in open source projects: Recommendations for issue trackers. In: Proc. of the 2013 Int'l Workshop on Principles of Software Evolution. Saint Petersburg: ACM, 2013. 43–52. [doi: 10.1145/2501543.2501550]

[113] Gu HY, Zhao L, Shu C. Analysis of duplicate issue reports for issue tracking system. In: Proc. of the 3rd Int'l Conf. on Data Mining and Intelligent Information Technology Applications. Macao: IEEE, 2011. 86–91.

[114] Li ZX, Yin G, Yu Y, Wang T, Wang HM. Detecting duplicate pull-requests in GitHub. In: Proc. of the 9th Asia-Pacific Symp. on Internetware. Shanghai: ACM, 2017. 20. [doi: 10.1145/3131704.3131725]

[115] Heck P, Zaidman A. Horizontal traceability for just-in-time requirements: The case for open source feature requests. Journal of Software: Evolution and Process, 2014, 26(12): 1280–1296. [doi: 10.1002/smr.1678]

[116] Gotel O, Cleland-Huang J, Hayes JH, Zisman A, Egyed A, Grünbacher P, Dekhtyar A, Antoniol G, Maletic J, Mäder P. Traceability fundamentals. In: Cleland-Huang J, Gotel O, Zisman A, eds. Software and Systems Traceability. London: Springer, 2012. 3–22. [doi: 10.1007/978-1-4471-2239-5_1]

[117] Gotel OCZ, Finkelstein CW. An analysis of the requirements traceability problem. In: Proc. of the 1994 IEEE Int'l Conf. on Requirements Engineering. Colorado Springs: IEEE, 1994. 94–101. [doi: 10.1109/ICRE.1994.292398]

[118] Xu F. Best Practices of Software Requirements: Principles and Applications of SERU Process Framework. Beijing: Publishing House of Electronics Industry, 2008 (in Chinese).

[119] Natt och Dag J, Gervasi V, Brinkkemper S, Regnell B. Speeding up requirements management in a product software company: Linking customer wishes to product requirements through linguistic engineering. In: Proc. of the 12th Int'l Requirements Engineering Conf. Kyoto: IEEE, 2004. 283–294. [doi: 10.1109/ICRE.2004.1335685]

[120] Merten T, Krämer D, Mager B, Schell P, Bürsner S, Paech B. Do information retrieval algorithms for automated traceability perform effectively on issue tracking system data? In: Proc. of the 22nd Int'l Working Conf. on Requirements Engineering: Foundation for Software Quality. Gothenburg: Springer, 2016. 45–62. [doi: 10.1007/978-3-319-30282-9_4]

[121] Licorish SA, Zolduoarrati E, Stanger N. Linking user requests, developer responses and code changes: Android OS case study. In: Proc. of the 22nd Int'l Conf. on Evaluation and Assessment in Software Engineering. Christchurch: ACM, 2018. 79–89. [doi: 10.1145/

3210459.3210467]

[122] Seiler M, Paech B. Using tags to support feature management across issue tracking systems and version control systems. In: Proc. of the 23rd Int'l Working Conf. on Requirements Engineering: Foundation for Software Quality. Essen: Springer, 2017. 174–180. [doi: 10.1007/978-3-319-54045-0_13]

[123] Palomba F, Linares-Vásquez M, Bavota G, Oliveto R, Di Penta M, Poshyvanyk D, De Lucia A. User reviews matter! Tracking crowdsourced reviews to support evolution of successful Apps. In: Proc. of the 2015 Int'l Conf. on Software Maintenance and Evolution. Bremen: IEEE, 2015. 291–300. [doi: 10.1109/ICSM.2015.7332475]

[124] The Standish Group. Standish group 2015. The Standish Group Int'l Inc., 2015.

[125] Erne M, Jiang ZY, Liu V. Investigating the effect of user reviews on mobile Apps: The role of customer led innovation. In: Proc. of the 2020 IFIP WG 8.6 Int'l Working Conf. on Transfer and Diffusion of IT. Tiruchirappalli: Springer, 2020. 193–200. [doi: 10.1007/978-3-030-64849-7_18]

[126] Xiong WJ, Zhang X, Wang X, Li T, Yin CL. Prediction on closed-probability of change request report for issue tracking system. Computer Science, 2017, 44(11): 146–155 (in Chinese with English abstract). [doi: 10.11896/j.issn.1002-137X.2017.11.022]

[127] Nizamani ZA, Liu H, Chen DM, Niu ZD. Automatic approval prediction for software enhancement requests. Automated Software Engineering, 2018, 25(2): 347–381. [doi: 10.1007/s10515-017-0229-y]

[128] Umer Q, Liu H, Sultan Y. Sentiment based approval prediction for enhancement reports. Journal of Systems and Software, 2019, 155: 57–69. [doi: 10.1016/j.jss.2019.05.026]

[129] Arshad MA, Huang ZQ, Riaz A, Hussain Y. Deep learning-based resolution prediction of software enhancement reports. In: Proc. of the 11th Annual Computing and Communication Workshop and Conf. Las Vegas: IEEE, 2021. 492–499. [doi: 10.1109/CCWC51732.2021.9375841]

[130] Fitzgerald C, Letier E, Finkelstein A. Early failure prediction in feature request management systems. In: Proc. of the 19th Int'l Requirements Engineering Conf. Trento: IEEE, 2011. 229–238. [doi: 10.1109/RE.2011.6051658]

[131] Fitzgerald C, Letier E, Finkelstein A. Early failure prediction in feature request management systems: An extended study. Requirements Engineering, 2012, 17(2): 117–132. [doi: 10.1007/s00766-012-0150-7]

[132] Heppler L, Eckert R, Stuermer M. Who cares about my feature request? In: Proc. of the 12th IFIP Int'l Conf. on Open Source Systems: Integrating Communities. Gothenburg: Springer, 2016. 85–96. [doi: 10.1007/978-3-319-39225-7_7]

[133] Canfora G, Cerulo L. Supporting change request assignment in open source development. In: Proc. of the 2006 ACM Symp. on Applied Computing. Dijon: ACM, 2006. 1767–1772. [doi: 10.1145/1141277.1141693]

[134] Kagdi H, Poshyvanyk D. Who can help me with this change request? In: Proc. of the 17th Int'l Conf. on Program Comprehension. Vancouver: IEEE, 2009. 273–277. [doi: 10.1109/ICPC.2009.5090056]

[135] Kagdi H, Gethers M, Poshyvanyk D, Hammad M. Assigning change requests to software developers. Journal of Software: Evolution and Process, 2012, 24(1): 3–33. [doi: 10.1002/smr.530]

[136] Linares-Vásquez M, Hossen K, Dang H, Kagdi H, Gethers M, Poshyvanyk D. Triaging incoming change requests: Bug or commit history, or code authorship? In: Proc. of the 28th Int'l Conf. on Software Maintenance. Trento: IEEE, 2012. 451–460. [doi: 10.1109/ICSM.2012.6405306]

[137] Hossen K, Kagdi H, Poshyvanyk D. Amalgamating source code authors, maintainers, and change proneness to triage change requests. In: Proc. of the 22nd Int'l Conf. on Program Comprehension. Hyderabad: ACM, 2014. 130–141. [doi: 10.1145/2597008.2597147]

[138] Zanjani MB, Kagdi H, Bird C. Using developer-interaction trails to triage change requests. In: Proc. of the 12th IEEE/ACM Working Conf. on Mining Software Repositories. Florence: IEEE, 2015. 88–98. [doi: 10.1109/MSR.2015.16]

[139] Cavalcanti YC, do Carmo Machado I, da Motal S. Neto PA, de Almeida ES. Towards semi-automated assignment of software change requests. Journal of Systems and Software, 2016, 115: 82–101. [doi: 10.1016/j.jss.2016.01.038]

[140] Yang H, Sun XB, Li B, Hu JJ. Recommending developers with supplementary information for issue request resolution. In: Proc. of the 38th IEEE/ACM Int'l Conf. on Software Engineering Companion. Austin: IEEE, 2016. 707–709.

[141] Sun XB, Yang H, Xia X, Li B. Enhancing developer recommendation with supplementary information via mining historical commits. Journal of Systems and Software, 2017, 134: 355–368. [doi: 10.1016/j.jss.2017.09.021]

[142] Claytor L, Servant F. Understanding and leveraging developer inexpertise. In: Proc. of the 40th Int'l Conf. on Software Engineering: Companion Proc. Gothenburg: ACM, 2018. 404–405. [doi: 10.1145/3183440.3195029]

[143] Palomba F, Salza P, Ciurumelea A, Panichella S, Gall H, Ferrucci F, De Lucia A. Recommending and localizing change requests for mobile Apps based on user reviews. In: Proc. of the 39th IEEE/ACM Int'l Conf. on Software Engineering. Buenos Aires: IEEE, 2017. 106–117. [doi: 10.1109/ICSE.2017.18]

[144] Ciurumelea A, Schaufelbühl A, Panichella S, Gall HC. Analyzing reviews and code of mobile Apps for better release planning. In: Proc. of the 24th Int'l Conf. on Software Analysis, Evolution and Reengineering. Klagenfurt: IEEE, 2017. 91–102. [doi: 10.1109/SANER.2017.7884612]

[145] Kato J, Goto M. User-Generated variables: Streamlined interaction design for feature requests and implementations. In: Proc. of the Companion to the 1st Int'l Conf. on the Art, Science and Engineering of Programming. Brussels: ACM, 2017. 28. [doi: 10.1145/3079368.3079403]

[146] Thung F, Wang SW, Lo D, Lawall J. Automatic recommendation of API methods from feature requests. In: Proc. of the 28th IEEE/ACM Int'l Conf. on Automated Software Engineering. Silicon Valley: IEEE, 2013. 290–300. [doi: 10.1109/ASE.2013.6693088]

[147] Xu CY, Sun XB, Li B, Lu XT, Guo HJ. MULAPI: Improving API method recommendation with API usage location. Journal of Systems and Software, 2018, 142: 195–205. [doi: 10.1016/j.jss.2018.04.060]

[148] Sun XB, Xu CY, Li B, Duan YC, Lu XT. Enabling feature location for API method recommendation and usage location. IEEE Access, 2019, 7: 49872–49881. [doi: 10.1109/ACCESS.2019.2910732]

[149] Li HD, Xie RS, Kong XL, Wang LL, Li BX. An analysis of utility for API recommendation: Do the matched results have the same efforts? In: Proc. of the 20th Int'l Conf. on Software Quality, Reliability and Security. Macao: IEEE, 2020. 479–488. [doi: 10.1109/QRS51102.2020.00067]

[150] Niu N, Bhowmik T, Liu H, Niu ZD. Traceability-enabled refactoring for managing just-in-time requirements. In: Proc. of the 22nd Int'l Requirements Engineering Conf. Karlskrona: IEEE, 2014. 133–142. [doi: 10.1109/RE.2014.6912255]

[151] Nyamawe AS, Liu H, Niu N, Umer Q, Niu ZD. Feature requests-based recommendation of software refactorings. Empirical Software Engineering, 2020, 25(5): 4315–4347. [doi: 10.1007/s10664-020-09871-2]

[152] Nyamawe AS, Bakhti K, Sandiwarno S. Identifying rename refactoring opportunities based on feature requests. Int'l Journal of Computers and Applications, 2021: 1–9. [doi: 10.1080/1206212X.2021.1922151]

附中文参考文献:

[16] 吕宏玉, 樊坤, 杨建林. 面向App用户评论的软件特征挖掘研究. 图书馆理论与实践, 2019(7): 106–112. [doi: 10.14064/j.cnki.issn1005-8214.2019.07.025]

[30] 肖建茂, 陈世展, 冯志勇, 刘朋立, 薛霄. 一种基于用户评论自动分析的APP维护和演化方法. 计算机学报, 2020, 43(11): 2184–2202. [doi: 10.11897/SP.J.1016.2020.02184]

[73] 骆斌, 丁二玉. 需求工程——软件建模与分析. 北京: 高等教育出版社, 2009. 194.

[103] 刘名成, 战洪飞, 石幸, 王雨潇. 面向产品设计的用户需求重要度分析方法. 机械制造, 2020, 58(12): 1–9. [doi: 10.3969/j.issn.1000-4998.2020.12.002]

[106] 李虹, 闫德恒. 基于项目需求工程理论的软件需求管理浅析. 中国科技信息, 2011(16): 92–93.

[118] 徐锋. 软件需求最佳实践: SERU过程框架原理与应用. 北京: 电子工业出版社, 2008.

[126] 熊文军, 张璇, 王旭, 李彤, 尹春林. 面向Issue跟踪系统的变更请求报告关闭可能性预测. 计算机科学, 2017, 44(11): 146–155. [doi: 10.11896/j.issn.1002-137X.2017.11.022]

牛菲菲(1996－), 女, 博士生, 主要研究领域为软件工程, 业务过程管理, 自然语言处理.

葛季栋(1978－), 男, 博士, 副教授, CCF 高级会员, 主要研究领域为软件工程, 分布式计算与边缘计算, 业务过程管理, 自然语言处理.

李传艺(1991－), 男, 博士, 助理研究员, CCF 专业会员, 主要研究领域为软件工程, 业务过程管理, 自然语言处理.

骆斌(1967－), 男, 博士, 教授, 博士生导师, CCF 杰出会员, 主要研究领域为软件工程, 人工智能.