软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2018,29(10):3021−3050 [doi: 10.13328/j.cnki.jos.005613]

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

持续集成测试用例集优化综述研究*

李英玲[1,3], 王 青[1,2,3]

[1](中国科学院 软件研究所 互联网实验室,北京 100190)

[2](计算机科学国家重点实验室(中国科学院 软件研究所),北京 100190)

[3](中国科学院大学,北京 100049)

通讯作者: 王青, E-mail: wq@itechs.iscas.ac.cn

摘 要: 基于互联网的软件开发要求产品快速迭代,同时保持产品的质量,其关键的环节就是持续集成.持续集成通过自动化测试来保证集成到主干的代码质量.持续集成时的测试用例选择是一个非常有挑战性的问题.如果运行所有的测试用例,需要消耗大量的计算资源,并造成测试反馈周期过长.如果选择的测试用例集不合适,又不足以覆盖必要的测试代码以保证待集成代码的质量.持续集成测试用例集优化的目的是平衡测试资源和测试质量,在尽可能不影响测试质量的情况下,减少持续集成的资源需求.对近年来国内外学者在该领域的研究工作进行了系统的分析、提炼和总结.为此,首先从研究主题、影响因子、研究方法、研究对象以及性能评价等方面提出 5 个研究问题;然后从电子文献数据库搜索最近 10 年的研究工作,经过仔细审查和筛选后选择 39 篇文献作为研究对象;最后,从选择文献中收集数据,通过定量分析和可视化展示来回答提出的研究问题.总的来说,回顾了持续集成测试用例集优化的研究进展,为该领域的研究者提供了一些有用的发现,并总结了面临的问题和挑战.

关键词: 持续集成;自动测试;测试用例选择;测试用例排序;测试用例生成;测试套件减少

中图法分类号: TP311

中文引用格式: 李英玲,王青.持续集成测试用例集优化综述研究.软件学报,2018,29(10):3021−3050. http.//www.jos.org.cn/1000-9825/5613.htm

英文引用格式: Li YL, Wang Q. Test set optimization in continuous integration: A systematic literature review. Ruan Jian Xue Bao/Journal of Software, 2018,29(10):3021−3050 (in Chinese). http.//www.jos.org.cn/1000-9825/5613.htm

Test Set Optimization in Continuous Integration: A Systematic Literature Review

LI Ying-Ling[1,3], WANG Qing[1,2,3]

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

[2](State Key Laboratory of Computer Sciences (Institute of Software, The Chinese Academy of Sciences), Beijing 100190, China)

[3](University of Chinese Academy of Sciences, Beijing 100049, China)

Abstract: Currently, many software applications are rapidly developed and deployed. Rapid iterations are used to implement new requirements while keeping high quality. Continuous integration (CI) is an important activity to make it work. CI tests the code automatically before it will be integrated into the master branch to ensure its quality. The big challenge is how to select an appropriate test case set for continuous integration. Running all test cases will consume a large amount of resource and get feedback in a long cycle, but selecting an improper test case subset will lose some required test, and therefore increase quality risk. The goal of test set optimization is to select an appropriate test case subset which can satisfy the test requirement for each unit or iteration test while reducing the test resource as much as possible. In this paper, a systematic literature review is performed to reveal the current status of CI-test set optimization based on existing literature. More specifically, the related studies published between 2007 and 2017 from four databases of

* 基金项目: 国家自然科学基金(61432001, 61602450)
Foundation item: National Natural Science Foundation of China (61432001, 61602450)
收稿时间: 2018-01-11; 修改时间: 2018-05-01; 采用时间: 2018-06-18

electronic literature are searched to finally select 39 important studies for careful review. Five research questions are presented in order to achieve the above goal, and data from the selected studies are extracted to analyze and answer the questions. The analysis results contribute some useful findings and new challenges for researches and practitioners.

Key words: continuous integration; automated testing; test case selection; test case prioritization; test case generation; test suite reduction

敏捷(agile)是一种迭代以及快速响应的理念应用于软件产品开发和交付,如何能在不断变化的需求中快速适应和保证软件的质量显得尤为重要.持续集成正是解决这一问题的软件开发实践,持续集成的理念倡导开发人员频繁地集成变更的代码,甚至每天都可能发生多次.每次集成都是通过自动化编译、构建和测试,从而尽快地发现缺陷,保证集成的代码处于充分测试的状态,以便产品可以随时发布[1].已有研究表明,持续集成在不降低测试质量的情况下,可改进开发团队的生产力,降低开发成本,并在开源社区和工业界上逐步得到广泛的应用[2−4].近年来,有研究者在敏捷基础上,提出了 DevOps 方法论[5,6],并在学术界和工业界得到了普遍关注.DevOps 以 IT 自动化以及持续集成(CI)、持续部署(CD)为基础,优化软件开发、测试、系统运维等所有环节;它主要用于搭建持续集成到持续交付的桥梁,并通过开发和运营团队的合作来支持软件产品(IT 服务)的迭代开发和持续交付.DevOps 的出现和广泛应用,使得持续集成越来越受到重视.

在持续集成过程中,自动化测试是一个重要的组成部分.当有新的变更代码提交合并时,需要回归测试检测是否引入新的缺陷,重新运行所有的测试是普遍适用的策略,但它会导致较高的测试成本,例如较高的测试资源需求和较长的测试反馈周期.调研文献 S8(见后文表 2)指出,即使拥有巨大的计算资源,谷歌的持续集成自动化测试,仍然不能满足开发人员以平均每秒的速度提交一个集成请求.在持续集成过程中,由于运行大量的测试需要较长的测试时间,为了缓解这个问题,部分企业选择在夜间运行当天的集成构建自动化测试,开发人员第 2 天清晨得到测试的反馈.即使这样,调研文献 S6(见后文表 2)通过访谈一系列工业界的开发人员后证实,较长的测试反馈周期严重影响了开发效率.例如,当耗时几个小时的持续构建失败时,开发人员需要切换上下文去修复缺陷,这将导致至少花费半个小时来切换上下文.因此,为了支持频繁的代码集成和不影响开发效率,持续集成测试的快速反馈至关重要.在有限的资源约束和不降低测试质量的前提下,通过优化测试用例集来减少持续集成时间,已成为软件工程领域关注的热点.

在过去的 10 年,已经出现了很多研究工作来攻克持续集成测试用例集优化面临的挑战,但最近几年还没有出现系统综述研究.为了更好地理解该领域面临的核心问题和研究进展,对这一领域的研究进行系统的调研变得十分必要.本研究通过系统性调研来回答以下问题:(1) 调研文献主要围绕哪些研究主题来优化持续集成测试用例集?(2) 调研文献使用了哪些度量元(影响因子)研究持续集成测试用例集优化?不同研究主题使用的影响因子是否存在偏好?(3) 调研文献,针对不同的研究主题,主要采用了哪些研究方法来优化持续集成测试集?不同方法使用的影响因子是否存在偏好?(4) 在已有研究中,研究对象的应用场景、数据来源及规模如何?(5) 不同研究主题,主要采用哪些评价指标来评价方法的有效性?各方法的效果和性能如何?

基于以上问题,首先搜索近 10 年相关的研究工作,通过人工仔细审查后选择 39 篇文献作为研究对象;然后从研究主题、影响因子、研究方法、研究对象和性能评价等方面提取研究文献的数据,并通过定量分析和可视化展示来回答提出的研究问题.本文通过以上过程调研了该领域的研究现状,提出了一些重要的发现,最后总结了该领域面临的问题和挑战.

本文第 1 节给出研究方法介绍,包括研究计划、研究问题和搜索策略.第 2 节是研究结果展示,回答提出的研究问题.第 3 节讨论研究的局限性.第 4 节通过对已有研究的总结,提出该领域存在的问题和我们的建议.第 5 节总结全文.

1 研究方法

本文的综述研究遵循 Kitchenham 和 Charters[7]提出的方法,系统性地调研持续集成测试用例集优化的研究进展.

1.1 研究计划

本研究通过制定研究计划定义一个可控的程序来进行,以减少研究人员主观的可能性.该研究计划包括研究问题和搜索策略.其中,搜索策略包括文献的搜索过程、论文筛选、质量评估、数据提取和合成.第一作者制定计划后,需要进行实验搜索和数据提取以评价和改进计划.最后由合著作者对计划进行评审.

通过上述过程得到了最终的研究计划.在下面章节中,我们将详细介绍该计划的不同组成部分.

1.2 研究问题

本文从研究主题、影响因子、研究方法、研究对象和性能评价等方面设定研究问题,调研持续集成测试用例集优化的研究进展.表 1 列出了各研究问题及动机.

Table 1 The research questions and motivation

表 1 研究问题和动机

研究问题	问题描述	动机
RQ1	调研文献主要围绕哪些研究主题来优化持续集成测试用例集?	识别持续集成测试用例集优化的当前研究主题
RQ2	已有研究使用了哪些度量元(影响因子)研究持续集成测试用例集优化?不同研究主题的影响因子是否存在偏好?	识别调研文献中的影响因子,分析不同研究主题使用的影响因子偏好和影响因子随着时间的变化
RQ2.1	已有研究使用的影响因子有哪些?	
RQ2.2	不同研究主题使用的影响因子是否存在偏好?	
RQ2.3	影响因子随着时间有哪些变化?	
RQ3	针对已有研究不同的研究主题,主要采用了哪些研究方法来优化持续集成测试集?不同方法使用影响因子是否存在偏好?	识别已有研究方法,分析不同研究主题使用的研究方法以及不同方法使用的影响因子偏好
RQ3.1	在调研文献中,常用的研究方法有哪些?	
RQ3.2	不同研究主题使用哪些研究方法?	
RQ3.3	不同方法使用影响因子是否存在偏好?	
RQ 4	在调研文献中,研究对象的应用场景、数据来源及规模如何?	分析研究对象的应用场景、数据来源和数据集规模
RQ4.1	已有研究主要应用于哪些编程语言的场景?	
RQ4.2	调研文献的数据来源主要有哪些?	
RQ4.3	调研文献的数据集规模怎么样?	
RQ 5	已有研究常用的评价指标有哪些?不同研究主题,主要采用哪些评价指标来评价方法的有效性?已有研究方法的效果和性能如何?	识别常用评价指标,分析不同研究主题使用的评价指标和比较已有研究方法的性能
RQ5.1	在调研文献中,常用的评价指标有哪些?	
RQ5.2	不同研究主题使用的评价指标有哪些?	
RQ5.3	已有研究方法的效果和性能如何?	

1.3 搜索策略

基于以上的研究问题,本文定义研究的搜索策略,包括搜索术语、文献资源、搜索过程、文献筛选、质量评估、数据提取和合并等.

1.3.1 搜索术语

本节首先收集研究主题相关的术语来构建查询串,然后用连接词“and”和“or”组成查询串:(continuous integration OR continual OR automated build) and (test OR testing OR test case OR test suite) and (selection OR order OR prioritization OR permutation OR reduction OR minimization OR augmentation OR optimization OR improvement).

1.3.2 文献资源

Kitchenham 等人[7]指出,自动搜索文献和人工搜索文献都存在一定的缺陷,本研究以自动搜索为主,人工搜索为辅的方式:首先选择电子数据库进行自动搜索,然后通过文献引用关系或作者的关联关系进行人工搜索.本文选择的用于自动搜索的电子数据库包括:

- IEEE Xplore Digital Library
- ACM Digital Library
- Science Direct Digital Library

● Springer Link Digital Library

选择以上数据库的原因是,这几个数据集基本上覆盖了持续集成测试用例集优化领域发表的所有论文.

1.3.3 搜索过程

本研究主要通过以下两个阶段的搜索,以防止重要研究文献的遗漏.

● 第 1 个阶段:首先使用查询串,分别在 4 个数据库中基于标题、摘要和关键字自动搜索得到主要的候选研究文献,同时限制搜索的文献发表在 2007 年 1 月 1 日~2017 年 8 月 20 日期间.接着合并搜索结果、移除重复文献,得到自动搜索的结果.

● 第 2 个阶段:为了识别更多相关的文献,在第 1 个阶段自动搜索的基础上再执行滚雪球式[8]的操作.当发现一篇高度相关文章时,将它加入到重要相关研究集.

1.3.4 论文筛选

研究文献的筛选过程如图 1 所示.

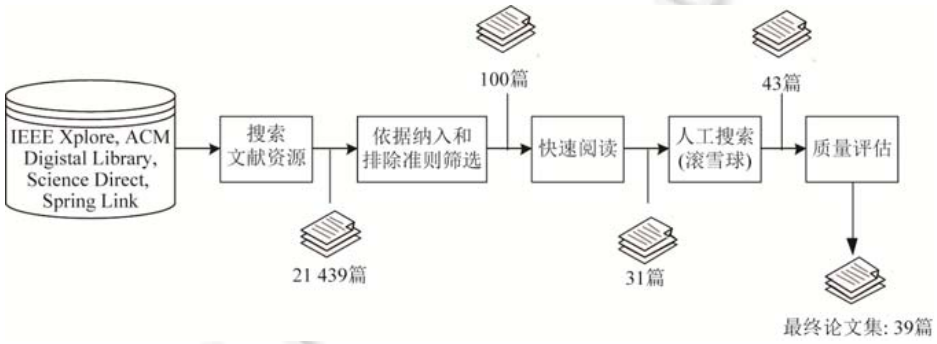

Fig.1 The study search and selection process

图 1 文献搜索及筛选过程

本文定义纳入和排除标准,用于对基于标题、摘要和关键字搜索得到的研究文献进行初步筛选.其中,纳入准则内容包括:

● 研究属于期刊和会议的长文和短文.由于部分短文的内容也具有一定的参考价值,将这些短文列入研究范围.

● 研究用英文发表.

● 研究属于软件工程领域.

● 标题和摘要是持续集成测试相关.

排除准则的内容包括:

● 出版类型是书、灰色文献.灰色文献一般指不作商业出版的出版物,其目的是为了传播信息,如技术报告、学位论文和正在进行的工作[7].

● 重复的研究工作.

如图 1 所示,文献的筛选过程如下:第 1 阶段搜索得到 21 439 篇候选研究文献;依据纳入和排除准则,初步筛选后得到 100 篇候选论文;然后快速阅读全文,进一步识别最相关的文献,最终保留 31 篇重要文献集.第 2 阶段,审查 31 篇重要文献的引用,识别出相关的且在第 1 阶段没有出现的文献.通过人工搜索,新增 12 篇文献到重要文献集,以确保对相关工作的覆盖度.最后,对 43 篇重要文献进行质量评估,最终选择 39 篇文献作为研究集.

另外,为了确保文献选择的质量,合著作者需要对纳入和排除准则进行评审;在选择的过程中,依据纳入和排除准则对候选文献进行分类,符合纳入准则的文献才被纳入候选集,与研究主题不太匹配的候选文献,需要与合著作者一起评审来决定是否保留.

1.3.5 质量评估

本研究采用质量评估来提高选择文献的质量,并设计了 5 个评估问题来评估已选重要文献的严谨性和可

信度,问题如下所示.

- QA1:是否定义明确的研究目标?
- QA2:是否有清晰的研究内容的描述?
- QA3:是否提出较好的研究方案?
- QA4:是否包含研究限制的分析?
- QA5:文献对本文的研究内容是否有价值?

对于每个评审问题,有 3 种可选的答案:“有”“部分有“和”没有”,这 3 种答案对应的分值是“有”=1,“部分有”=0.5,“没有”=0.对每个重要文献,它的质量分值由5个问题的得分总和而得.已选的重要文献的质量评分由第 1 作者和两名学生分别完成,任何有分歧的评分都需要一起讨论来决定.最后,收集由自动搜索和滚雪球式人工搜索得到的 43 篇文献的质量评分,并按质量水平对文献进行分类:高(分值=5)、中(3=<分值<5)、低(分值<3).选择质量水平属于中高等的 39 篇文献,作为最后的研究集(见表 2).

Table 2 The selected studies and their scores of quality assessment

表 2 最终的文献列表及对应质量评分

研究编号	对应参考文献编号	研究主题	质量评分	研究编号	对应参考文献编号	研究主题	质量评分
S1	文献[19]	测试用例选择	4.5	S21	文献[39]	测试用例选择	3.5
S2	文献[20]	测试用例选择	5	S22	文献[40]	测试用例生成	3.5
S3	文献[21]	混合优化研究	5	S23	文献[41]	测试用例排序	3.5
S4	文献[22]	测试用例排序	5	S24	文献[42]	测试用例选择	5
S5	文献[23]	混合优化研究	5	S25	文献[43]	测试用例选择	5
S6	文献[24]	测试套件减少	4.5	S26	文献[44]	混合优化研究	3
S7	文献[25]	测试套件减少	4.5	S27	文献[45]	测试用例选择	4.5
S8	文献[26]	混合优化研究	4.5	S28	文献[46]	测试用例排序	5
S9	文献[27]	混合优化研究	4	S29	文献[47]	测试用例排序	4
S10	文献[28]	测试用例选择	5	S30	文献[48]	测试用例选择	4
S11	文献[29]	测试用例排序	4	S31	文献[49]	测试用例排序	3
S12	文献[30]	测试用例排序	4	S32	文献[50]	测试用例排序	3.5
S13	文献[31]	测试用例选择	4	S33	文献[51]	测试用例排序	3.5
S14	文献[32]	混合优化研究	4	S34	文献[52]	测试用例排序	4
S15	文献[33]	混合优化研究	5	S35	文献[53]	测试用例排序	4
S16	文献[34]	混合优化研究	5	S36	文献[54]	测试用例排序	5
S17	文献[35]	测试用例排序	4	S37	文献[55]	测试套件减少	4
S18	文献[36]	测试用例排序	4	S38	文献[56]	测试用例选择	5
S19	文献[37]	测试用例生成	4	S39	文献[57]	测试用例选择	4.5
S20	文献[38]	测试用例选择	4				

1.3.6 数据提取和合成

本研究从最终的文献集中提取数据,并创建了数据表来记录提取信息:第 1 个数据表记录文献的基本信息,例如标题、作者姓名、摘要总结等,用于纳入/排除的筛选[9];第 2 个数据表记录从文献提取的信息,用于回答研究问题.数据提取工作由第 1 作者完成,在该过程中,任何的不确定性都需要与第 2 作者讨论来决定.表 3 展示了研究属性和研究问题之间的映射关系.接着,我们对调研文献中提取的数据进行合成,然后用可视化的工具(例如条形图、折线图、气泡图等)进行展示,用于回答 RQ1~RQ5 的研究问题.

Table 3 The mapping between data properties and research questions

表 3 数据提取与研究问题的对应关系

属性	描述	研究问题编号
研究问题	收集论文题目、解决的问题、作者、发表年份、发表刊物或者会议等信息	RQ1
研究主题	主要研究内容、主题、采用的技术、影响因子、应用场景	RQ1,RQ2,RQ4
影响因子	收集影响因子,包括代码覆盖、需求覆盖、当前测试与历史失败测试距离等信息	RQ2
研究方法	收集采用的研究方法、方法的粒度、方法的优缺点和局限性	RQ3
数据集	收集项目来源、应用场景、代码行数、代码类个数、测试用例数、功能点数等信息	RQ4
性能评价	收集评价指标、对比的基线、评价结果	RQ5

2 研究结果

2.1 调研文献的总体情况

经过文献检索与筛选,最终得到持续集成测试用例集优化领域自 2007 年 1 月 1 日~2017 年 8 月 20 日的学术论文 39 篇,其中会议长文 23 篇、会议短文 6 篇、期刊 6 篇和专题讨论 4 篇.

图 2 展示了自 2007 年以来该领域相关文献的分布,可以看到文献数量逐年缓步上升,最近 3 年的文献数量最多:2016 年 9 篇,2017 年有高于 9 篇的趋势(因为搜索截止 2017 年 8 月,2017 年的文章可能不完整).这说明该领域正趋活跃状态;另外,2008 年没有任何文献出现,暂时没有发现特殊原因.

图 3 展示了 CCF 不同级别的文献分布情况,可以看出,67%的调研文献属于 CCF 级别的论文.这个分布说明了持续集成测试用例集优化领域在国际顶级会议和期刊趋于较活跃状态,是较热门的研究问题.

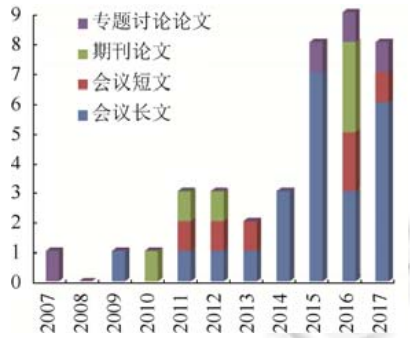

Fig.2 The distribution of the selected studies by year

图 2 研究文献按年份的分布情况

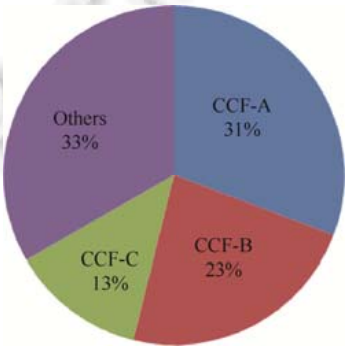

Fig.3 The distribution of the selected studies by CCF

图 3 研究文献按级别的分布情况

表 4 列出了在持续集成测试用例集优化的调研文献中,CCF 级别的文献共 26 篇,其中,CCF-A 类论文共 12 篇(会议 9 篇、期刊 3 篇),发表在 FSE、ICSE、ASE、JSS、TSE 上;CCF-B 类论文共 9 篇(会议 8 篇、期刊 1 篇),发表在 ISSTA、ICST、ICSME、ISSRE、ESE 上;CCF-C 类论文共 5 篇(会议 4 篇、期刊 1 篇),发表在 ICST、COMPSAC、QRS、SQJ.

Table 4 The distribution of the selected studies by publication types

表 4 研究文献按出版分布列表

类别	CCF 级别	会议/期刊	文献编号	数量
会议论文	A	{FSE,ICSE,ASE}	{S4,S5,S7,S8,S13,S22,S25,S27,S38}	9
	B	{ISSTA,ICST,ICSME,ISSRE}	{S3,S9,S10,S11,S12,S15,S16,S24}	8
	C	{ICST,COMPSAC,QRS}	{S17,S23,S28,S39}	4
期刊论文	A	{JSS,TSE}	{S18,S26,S29}	3
	B	{ESE}	{S1}	1
	C	{SQJ}	{S2}	1

2.2 RQ1 研究主题

本节主要调研持续集成测试用例集优化的研究主题.调研文献按研究主题类型分为 5 类:测试用例排序、测试用例选择、测试套件减少、测试用例生成、混合优化方法,该分类参考 Yoo 等人的研究[10]中提出的测试用例排序、测试用例选择和测试套件减少;并根据研究文献的研究主题加入测试用例生成和混合优化方法.

- 测试用例排序(TCP):文献 S32 指出,测试用例排序是减少持续集成回归测试工作量的一种有效方法.它旨在找出能够尽快检测出缺陷的最优用例顺序,从而提高软件的发布频率.
- 测试用例选择(RTS):文献 S39 指出,测试用例选择是一个典型的测试技术,该技术是在有限的测试资源和时间期限内,选择一个测试子集来测试变更的代码.
- 测试套件减少(TSR):测试套件减少或最小化,是指移除冗余测试,识别出测试套件中不可替代的测试来

减少测试规模,从而减少测试成本[10].

● 测试用例生成(TCG):主要是指持续集成过程中,依据代码变化、设计文档、UML 图等自动生成测试用例.

● 混合优化方法(Mix):组合测试用例排序、测试用例选择、测试套件减少和测试用例生成研究主题中,两个或两个以上的主题进行持续集成测试用例集优化.

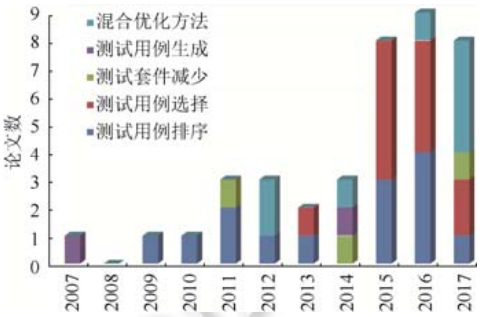

Fig.4 The evolution of research topics by years

图 4 研究主题随着年份的演化

图 4 展示了研究主题随着年份的演化情况,可以看出:持续集成测试用例集优化的研究在最近 3 年增长迅速,主要以测试用例排序、测试用例选择和混合优化研究为主.在调研文献中,以测试用例排序为主题的文献共 14 篇(占所有文献的 35.9%),以测试用例选择为主题的文献共 12 篇(占 30.77%),以测试套件减少为主题的文献共 3 篇(占 7.69%),以测试用例生成为主题的文献共 2 篇(占 5.13%),以混合优化为主题的文献共 8 篇(占 20.51%).

2.2.1 小 结

在持续集成测试用例集优化的研究领域,已有研究主要从测试用例选择、测试用例排序、测试套件减少和测试用例自动生成 4 个方面来优化测试用例集,从近 3 年的数据看,测试用例选择、排序以及混合优化方法呈现上升的趋势.总体上,以测试用例排序和测试用例选择的研究为主,其次是混合优化研究,关注测试套件减少和测试用例生成的研究比较少.

2.3 RQ2 影响因子

影响因子是指在持续集成测试用例优化的研究案例中,研究者标识出来用于研究实验的度量元.本节抽取调研文献中不同研究主题的影响因子,然后对其分类,并分析不同研究主题的影响因子偏好.

2.3.1 RQ2.1 影响因子及分类

参考 Yoo[10]和 Rafaqut[11]已有的影响因子分类,包括制品变化、成本因素、缺陷检测能力因素、测试历史、依赖关系,考虑持续集成测试集优化研究中其他的影响因子类型,包括文本相似度、变化代码与测试距离、测试用例间距离、用例风险值、领域知识等,将调研文献涉及的影响因子分为以下几类.

● 制品变化(change):与项目制品变化相关的影响因子,包括文本变化、代码变化、代码修改频率等.

● 覆盖信息:该分类假设,高覆盖的测试用例具有高的缺陷检测能力,包括代码覆盖(CC)和非代码覆盖(NCC)两类.其中,代码覆盖包括代码行覆盖、函数覆盖、类覆盖、方法覆盖、分支覆盖等;非代码覆盖包括数据库覆盖、变异覆盖、配置覆盖、需求覆盖、交叉功能覆盖等.

● 成本因素(cost):在调研文献中,部分研究考虑了测试成本的因素,例如总时间约束、测试执行成本等.

● 缺陷检测能力(fail-det):这是指用于度量测试用例的缺陷检测能力的影响因子,包括用例的缺陷检测能力分值、失败影响值、失效频率.

● 测试历史(*T*-history):从测试历史信息中提取的影响因子,包括上次执行的时间间隔、测试用例的修改、历史测试结果、测试执行时间、历史测试失败窗口、历史测试执行窗口、历史失败测试、历史执行数量、最近优先级、静态优先级、测试用例年龄等.

● 依赖关系(depend):文献 S1,S2,S25 指出,按获取依赖关系方法的不同,依赖关系可分为静态依赖关系和动态依赖关系.这两种依赖关系可以进一步区分不同的粒度,具体来讲,静态依赖关系包括静态依赖关系_方法级别、静态依赖关系_类级别、静态依赖关系_文件级别、静态依赖关系_模块级别、代码变化和测试代码相关性、代码变化和测试失败历史相关性、组件之间依赖关系、项目之间依赖关系、历史构建信息与测试依赖关系;动态依赖关系包括动态依赖关系_方法级别、动态依赖关系_文件级别等.

● 文本相似度(text-sim):这是指文本变化(例如配置文件、数据集)与测试用例相似度,例如文献 S4 考虑了文本变化与测试用例的相似度进行测试用例集优化.

● 代码变化与测试距离(chatest-dis):这是指将代码变化和测试代码的距离作为影响因子进行测试用例集优化.例如文献 S8 基于有向非循环图的依赖关系,来计算代码变化与测试代码距离进行测试用例集优化.

● 测试用例间距离(test-dis):指的是度量测试用例之间距离的影响因子,可用于比较当前用例与历史用例的相似距离,例如文献 S11 计算当前用例与上次失败用例的 Basic counting、Hamming distance 和 Edit distance 相似度距离作为影响因子;文献 S12 和 S39 基于用例间字符串距离来度量用例多样性.

● 用例风险值(risk):通过聚类测试历史信息来计算测试用例的风险值.文献 S12 指出在测试历史信息中,测试用例检测到的缺陷越多,则风险值越高.

● 领域知识(domain):使用特定领域知识作为测试用例集优化的影响因子.文献 S36 基于历史测试失败信息、时间约束和特定领域的启发,进行持续集成测试用例排序.

● 其他(others):指的是不常使用的影响因子,包括保留测试 kept tests、服务交互类型、测试目标的启发式近似距离等.

图 5 展示了各影响因子使用频率的排序.可以看出制品变化类中的代码变化是使用频度最高的因子,覆盖信息类的代码行覆盖、测试历史中的历史失败测试和测试执行时间、成本因素中的时间约束在使用频度排序中并列第二;最后是使用 1 次的影响因子包括代码覆盖类中的分支覆盖、变化制品中的代码修改频率等.

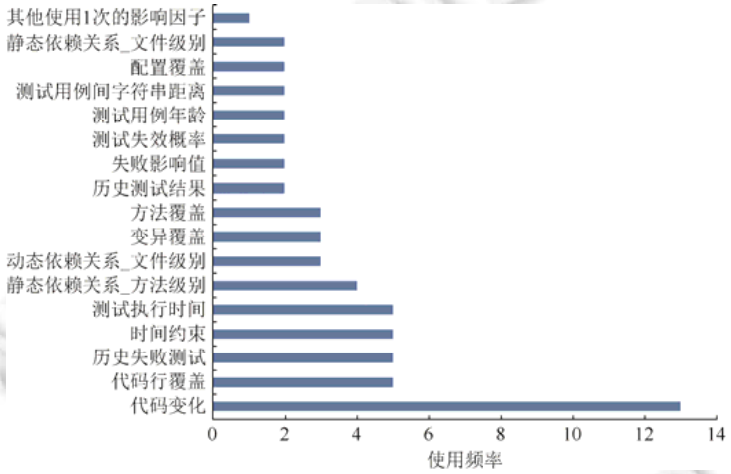

Fig.5 The order of factors by frequency

图 5 影响因子使用排序

2.3.2 RQ2.2 不同研究主题使用影响因子的偏好

本节主要分析不同的研究主题在识别和使用影响因子时是否存在不同的偏好.表 5 给出了各研究主题使用的影响因子及因子类型.

图 6 展示各研究主题的影响因子类型使用频率,气泡大小表示频率的高低.可以看到,测试用例排序主要采用测试历史 *T*-history、非代码覆盖 NCC 和代码覆盖 CC 类的影响因子;测试用例选择则更多采用依赖关系和制品变化类的影响因子.而制品变化类因子在所有研究主题都有使用,NCC、CC、*T*-history、depend 等类的因子也在大多数研究主题中使用.所以我们进一步分析在这些共同使用的类型中,不同主题对具体因子的偏好.

如图 6 所示,制品变化类的因子在各个研究主题都有使用,但从表 6 所示的信息中,可以发现“代码变化”在除 TCG 外的所有主题都使用,TCG 主题的偏好则是“提交信息”.在非代码覆盖类的因子中,“变异覆盖”是 Mix、RTS、TCG 主题都使用的因子,但 TCP 使用的是“主题覆盖”“交叉功能覆盖”和“模块依赖覆盖”等因子.在代码覆盖类的因子中,“代码行覆盖”在 RTS、TCP 和 Mix 主题都有用到.在测试历史类中,TCP 主要使用的是“历史失败测试”“测试执行时间”和“测试用例年龄”等因子,RTS 主题的偏好则是“历史测试结果”和“测试失败概率”,而“历史失败测试”和“测试执行时间”是 TCP 和 Mix 主题都使用的因子.

Table 5 The factors of different research topics

表 5 各研究主题使用的影响因子及类型

研究主题		影响因子分类	影响因子	频率
TCP		测试历史	{测试执行时间,静态优先级,测试用例年龄,历史失败测试,测试失效概率,测试历史时间窗口,在缺陷检测中历史的有效性,历史失败服务,历史执行数量,最近优先级}	9
		非代码覆盖	{模块依赖覆盖,配置覆盖,交叉功能覆盖,主题覆盖}	4
		代码覆盖	{功能点覆盖,代码行覆盖,方法覆盖,测试覆盖分值}	3
		制品变化	{文本变化,代码变化}	2
		成本因素	{时间约束}	2
		用例间距离	{当前用例与上次失败用例相似度距离_Basic counting,当前用例与上次失败用例相似度距离_Edit distance,当前用例与上次失败用例相似度距离_Hamming distance}	1
		缺陷检测能力	{失败影响值,失效频率}	1
		依赖关系	{项目之间依赖关系}	1
		多样性	{测试用例间字符串距离}	1
		文本相似度	{文本变化和测试代码相似度}	1
		风险值	{测试风险值}	1
		其他	{服务交互类型,特定领域知识}	1
RTS		依赖关系	{静态依赖关系_方法级别,代码变化和测试失败历史相关性,动态依赖关系_文件级别,代码变化和测试代码相关性,组件之间依赖关系,静态依赖关系_文件级别,静态依赖关系_类级别,静态依赖关系_模块级别}	10
		制品变化	{代码变化}	8
		代码覆盖	{类覆盖,方法覆盖,分支覆盖,代码行覆盖}	2
		测试历史	{历史测试结果,测试失效概率}	1
		用例间距离	{测试用例间字符串距离}	1
		成本因素	{测试执行成本}	1
		非代码覆盖	{变异覆盖}	1
TCG		其他	{异常信息,测试目标的启发式近似距离}	2
		非代码覆盖	{变异覆盖}	1
		制品变化	{提交信息}	1
TSR		依赖关系	{历史构建信息与测试依赖关系,静态依赖关系_方法级别}	2
		制品变化	{代码变化}	1
		其他	{保留测试集 kept tests}	1
Mix	TCP+ RTS	测试历史	{上次成功的缺陷检测,测试用例的修改,测试执行时间,测试执行窗口,上次执行的时间间隔,历史测试结果,历史测试失败窗口}	3
		成本因素	{时间约束}	3
		制品变化	{代码修改频率,代码变化}	2
		代码覆盖	{过程级别覆盖,代码行覆盖 }	2
		非代码覆盖	{变异覆盖,数据库覆盖}	2
		代码变化和测试的距离	{代码变化和测试代码的距离}	1
		缺陷检测能力	{用例的缺陷检测能力分值}	1
	TCP+ TSR	非代码覆盖	{配置覆盖,需求覆盖}	1
		测试历史	{历史失败测试}	1
		缺陷检测能力	{失败影响值}	1

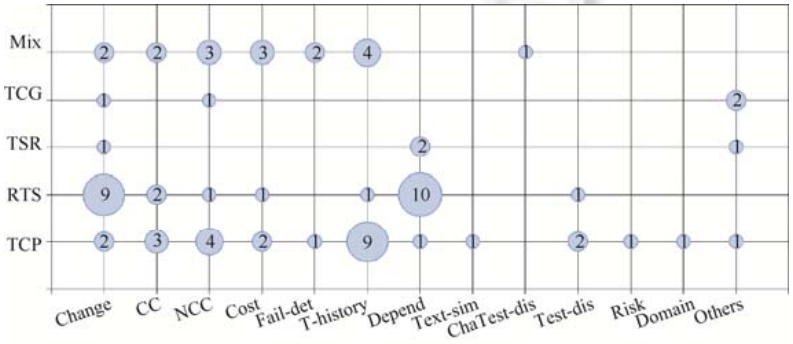

Fig.6 The frequency of factor types in different research topics

图 6 各研究主题的影响因子类型使用频率

Table 6 The frequency of factors within each factor type in different topic

表 6 同一分类下影响因子在各研究主题下的使用频率

影响因子分类	研究主题	频率	影响因子
制品变化 Change	RTS	9	{代码变化}
	Mix	2	{代码变化}
	TCP	1	{文本变化,代码变化}
	Mix	1	{代码修改频率}
	TCG	1	{提交信息}
	TSR	1	{代码变化}
代码覆盖 CC	RTS	2	{代码行覆盖}
	TCP	2	{代码行覆盖,方法覆盖}
	Mix	1	{代码行覆盖,过程级别覆盖}
	RTS	1	{分支覆盖,方法覆盖,类覆盖}
	TCP	1	{测试覆盖分值}
非代码覆盖 NCC	Mix	1	{配置覆盖,需求覆盖,变异覆盖,数据库覆盖}
	RTS	1	{变异覆盖}
	TCG	1	{变异覆盖}
	TCP	1	{主题覆盖,配置覆盖,交叉功能覆盖,模块依赖覆盖}
成本因素 Cost	Mix	3	{时间约束}
	TCP	2	{时间约束}
	RTS	1	{测试执行成本}
缺陷检测能力 Fail-det	Mix	1	{用例的缺陷检测能力分值,失败影响值}
	TCP	1	{失效频率,失败影响值}
测试历史 *T*-history	TCP	4	{历史失败测试,测试执行时间}
	TCP	2	{测试用例年龄}
	Mix	1	{历史测试执行窗口,测试执行时间,测试用例的修改,历史失败测试,历史测试结果,历史测试失败窗口,上次成功的缺陷检测,上次执行的时间间隔}
	RTS	1	{历史测试结果,测试失效概率}
	TCP	1	{测试失效概率,在缺陷检测中历史的有效性,静态优先级,历史失败服务,历史失败时间窗口大小,历史执行数量,最近优先级}
依赖关系 Depend	RTS	3	{动态依赖关系_文件级别,静态依赖关系_方法级别}
	RTS	2	{静态依赖关系_文件级别}
	RTS	1	{静态依赖关系_模块级别,组件之间依赖关系,动态依赖关系_方法级别,静态依赖关系_类级别,代码变化和测试失败历史相关性,代码变化和测试代码相关性}
	TCP	1	{项目之间依赖关系}
	TSR	1	{历史构建信息与测试依赖关系,静态依赖关系_方法级别}
文本相似度 Text-sim	TCP	1	{修改文本和测试相似度}
代码变化与测试距离 ChaTest-dis	Mix	1	{代码变化和测试代码的距离}
用例间距离 Test-dis	RTS	1	{测试用例间字符串距离}
	TCP	1	{测试用例间字符串距离,测试风险值,当前用例与上次失败用例相似度距离 Edit distance、Basic counting 和 Hamming distance}
领域知识 Domain	TCP	1	{特定领域知识}
其他 Others	TCG	1	{测试目标的启发式近似距离,异常信息}
	TCP	1	{服务交互类型}
	TSR	1	{保留测试}

类似地,依赖关系、成本因素等分类的影响因子在各研究主题中既存在共同使用的因子,也存在不同的偏好.另外,缺陷检测能力、文本相似度、代码变化与测试距离、领域知识等类别因子使用频率都比较低,并且只是在少数研究主题中使用.

本节分析不同的研究主题使用影响因子的偏好,发现不同研究主题使用的影响因子类型存在不同的偏好,但也存在共同使用的因子类型;在这些共同使用的分类中,不同主题对具体因子也存在不同的偏好.

2.3.3 RQ2.3 影响因子随着时间的变化

本研究定义影响因子的出现时间为因子第 1 次出现的文献发表时间,进而分析影响因子随着时间的变化.

图 7 列出了不同因子类型随着年份的变化情况,气泡大小表示新增因子数量的大小.

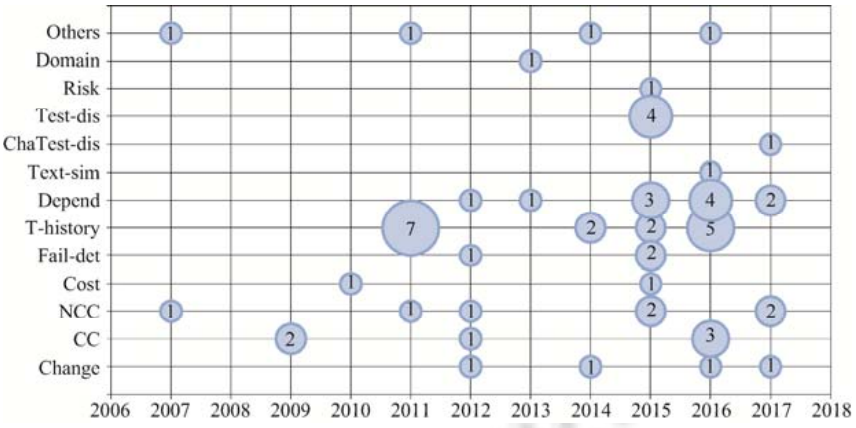

Fig.7 The evolution of the factor types by year

图 7 影响因子类型随着年份的变化

可以看出:早期的研究中,影响因子类型主要包括测试历史、代码覆盖和非代码覆盖;最近 3 年,逐渐出现一些新的因子类型,包括测试用例间距离、用例风险值、代码变化与测试间距离,传统的影响因子类型,如代码和非代码覆盖、测试历史、依赖关系也有新的影响因子提出.具体变化如下.

- 2015 年提出“测试用例间距离”影响因子类型,包括测试用例间字符串距离,当前用例与上次失败用例相似度距离 Edit distance、Hamming distance 和 Basic counting 影响因子.
- 代码覆盖的分类,在 2016 年新增的影响因子包括分支覆盖、类覆盖、测试覆盖分值.
- 非代码覆盖的分类,在 2015 年新增的影响因子包括交叉功能覆盖、主题覆盖;在 2017 年新增的影响因子包括配置覆盖、需求覆盖.
- 测试历史的分类,在 2015 年新增的影响因子包括测试失效概率、历史测试结果;在 2016 年新增的影响因子包括上次执行的时间间隔、历史失败服务、测试用例的修改、上次成功的缺陷检测、历史失败窗口大小.
- 依赖关系的分类,在 2015 年新增的影响因子包括代码变化和测试失败历史相关性、动态依赖关系_文件级别、代码变化和测试代码相关性;在 2016 年新增依赖关系的影响因子包括组件之间依赖关系、静态依赖关系_文件级别、静态依赖关系_类级别、动态依赖关系_方法级别.

随着 Web、服务应用等程序的广泛使用,有研究者指出这类系统与传统软件存在两方面的差异[12]:异质性和环境依赖.异质性是由于系统组件可能来自不同的源,而这些源又由不同的编程语言实现,并且以不同的形式(例如源代码、二进制代码或远程接口)提供.环境依赖是指系统的行为不仅受代码影响,还在一定程度上受其他因素(例如数据库、配置文件、项目之间依赖、网络布局等)影响.而传统的以代码为中心的方法,忽视了异构环境下的特定因素可能导致测试不完整.

文献 S31 指出,已有的项目代码通常在孤立环境下测试,没有考虑复杂环境下的因素(例如多个项目之间的依赖),可能造成测试不完整和发布软件质量不高的风险.在调研文献中,只有一项研究考虑了项目之间的依赖,这种依赖提高了持续集成自动测试的效率.另外,已有的测试用例集优化技术主要考虑代码制品和测试历史相关的影响因子,考虑了非代码制品(例如配置文件、数据库)对测试用例集优化影响的研究只有 1 项.

本节分析影响因子随时间的变化情况,发现早期研究主要基于测试历史、代码覆盖和非代码覆盖来研究持续集成测试用例集优化问题;最近 3 年,逐渐出现一些新的因子类型,并且传统的影响因子类型也提出了新的影响因子.但在调研文献中,只有两项研究考虑到了复杂异构环境下的项目之间依赖和非代码制品的影响因子.

2.3.4 小 结

本节识别调研文献中常用的影响因子并将其分类,然后调研不同研究主题使用的影响因子偏好,发现不同

研究主题使用的影响因子类型存在不同的偏好,但也存在共同使用的因子类型;在这些共同使用的因子类型中,不同研究主题对具体的因子也存在不同的偏好.最后分析影响因子随时间的变化情况,发现在最近 3 年逐渐出现一些新的影响因子类型,例如测试用例间距离、用例风险值等,传统的影响因子类型,包括代码和非代码覆盖、测试历史、依赖关系也提出了新的影响因子.

另外,已有文献指出,随着 Web、服务应用等程序的广泛使用,这类复杂异构系统与传统软件存在两方面的差异:异质性和环境依赖.但是,已有研究主要考虑传统的影响因子,考虑复杂异构环境下特定影响因子(例如项目之间依赖、数据库、配置文件、网络布局等)的研究只有两项.但文献 S31 指出,忽视这些因子的影响可能会带来测试不完整和发布软件质量不高的风险.

2.4 RQ3 研究方法

这一节调研不同研究主题采用的研究方法,对其进行分类,并分析各方法的优缺点和不同类型研究方法使用的影响因子偏好.

2.4.1 RQ3.1 研究方法及分类

我们识别调研文献的研究方法,将调研文献的研究方法分为静态方法和动态方法两大类,其中静态方法包括:

- 基于依赖关系(depend-relation)的方法:包括静态依赖关系的方法和动态依赖关系的方法.前者无需运行程序,通过静态分析来获取代码的依赖关系;而后者动态收集测试用例执行时的调用关系进行测试用例集优化.
- 主题(topic)模型:采用主题模型的方法来进行测试用例集优化.
- 测试用例相似度(test-similarity):基于历史测试用例与失败用例间相似距离进行测试用例排序.
- 其他静态方法(other-static):包括基于测试历史计算用例的权重值进行排序的方法、基于测试成本的测试用例选择策略 THEO 等.

参考 Yoo[10]和 Qiu[13]对动态方法的已有分类,包括贪婪算法、搜索算法、整数规划、自适应随机算法;考虑调研文献中其他的动态方法,包括多目标优化、机器学习算法、基于测试历史时间窗口进行测试选择和排序等,将动态方法分为以下几类.

- 贪婪(greedy)算法:主要包括基于覆盖、total 和 additional 贪婪算法,也有研究者提出重叠感知贪婪算法.
- 搜索(search)算法:调研文献中,部分研究使用搜索算法求解用例集优化问题,例如文献 S9 提出基于搜索的算法 Flower 减少测试套件.
- 多目标优化(multiobjects-optim)算法:使用多目标优化的算法进行测试用例集优化,例如文献 S36 以最大化可执行测试数量和最小化测试时间为目标,提出测试用例排序算法 ROCKET.
- 整数规划(IP):用整数规划的方法求解测试用例集优化问题.
- 机器学习(ML):用机器学习算法优化测试用例集,包括二元决策树 BDFs、强化学习、线性 SVM 算法、凝聚式层次聚类算法、基于贝叶斯网络排序等.
- 其他动态方法(other-dynamic):包括基于测试历史时间窗口进行测试用例选择和排序、基于随机和演化变异的测试用例生成方法.

另外,部分调研文献使用组合算法,该类方法是组合以上 2 个或 2 个以上分类的方法.例如文献 S12 组合主题模型、贪婪算法和聚类算法进行测试用例集优化.

2.4.2 RQ3.2 不同研究主题的研究方法

本节调研不同研究主题使用的研究方法是否存在偏好.表 7 展示了各研究主题使用的研究方法类型及对应的文献.表中显示不同研究主题采用不同类型的研究方法,例如 TCP 主要采用多目标优化算法、其他静态方法和组合算法类的方法;而 RTS 主要采用基于依赖关系的方法.

Table 7 The methods and literature in different research topics

表 7 各研究主题使用的研究方法及对应文献

研究主题		研究方法分类	文献编号	文献数量
TCP		多目标优化算法	S23,S32,S34,S35,S36	5
		其他静态方法	S17,S33	2
		贪婪算法+机器学习	S28,S29	2
		主题模型+贪婪算法+机器学习	S12	1
		机器学习算法	S4	1
		贪婪算法	S18	1
		测试用例相似度	S11	1
RTS		基于依赖关系的方法	S1,S10,S2,S20,S21,S24,S25,S27,S30,S38	10
		多目标优化算法	S39	1
		其他静态方法	S13	1
TSR		基于依赖关系的方法	S6	1
		贪婪算法	S7	1
		贪婪算法+机器学习	S37	1
TCG		搜索算法	S22	1
		其他动态算法	S19	1
Mix	TCP+RTS	贪婪算法	S14,S15,S16	3
		机器学习算法	S3,S8	2
		其他动态方法	S5	1
		贪婪算法+整数规划	S26	1
	TCP+TSR	多目标优化+搜索算法	S9	1

下面几节将详细介绍不同研究主题的研究方法.

2.4.2.1 测试用例排序(TCP)

测试用例排序共有 14 篇文献,文献 S31 只是提出解决方案,没有通过实验来验证.其他研究的排序方法主要包括多目标优化算法、其他静态方法、组合算法、机器学习算法、贪婪算法和测试用例相似度方法.

(1) 多目标优化算法.在 2011 年,文献 S35 以最大化依赖覆盖、缺陷检测能力和最小化测试时间为目标,对提交前的测试用例排序.接着,在 2015 年,文献 S23 基于时间约束、缺陷检测能力和失败影响,通过计算每个测试用例权重值进行排序.文献指出,该方法与人工测试比较,排序后的测试更快、更有效.2017 年,文献 S34 基于历史和配置覆盖,最大化每个测试用例的配置覆盖、最大化缺陷检测能力和最快测试反馈进行测试用例排序.另外,文献 S36 和文献 S32 分别在 2013 年和 2016 年基于测试历史信息和特定领域知识,最大化测试执行数量和最小化测试时间,通过计算测试用例的权重进行排序.文献指出该方法只需测试历史信息,能应用到大型集成项目中.

(2) 其他静态算法.文献 S17 和文献 S33 分别在 2011 年和 2016 年基于测试历史信息计算当前测试用例的权重,并依据权重进行排序,包括 ExtFaz 和 AFSAC 算法.

(3) 组合算法.有 2 项研究基于贪婪算法和机器学习算法进行测试用例排序,有 1 项组合主题模型、贪婪算法和机器学习算法.早在 2009 年,文献 S28 提出组合贪婪算法和凝聚式层次聚类算法进行测试用例排序.接着,文献 S29 在 2010 年设置控制实验,分析基于代码覆盖排序和基于贝叶斯网络排序方法.最后,文献 S12 在 2015 年提出基于覆盖、基于多样性和风险驱动,组合主题模型、聚类算法和贪婪算法进行测试用例排序.文献指出,该方法考虑历史风险值进行排序比随机排序方法在快速发布的环境下更有效,但在不同版本之间,方法的性能波动比较大.

(4) 机器学习算法.2016 年,文献 S4 综合已有的特征,用线性 SVM 模型进行用例排序.文献指出,该方法的缺陷检测能力比单个特征的方法要好.

(5) 贪婪算法.2016 年,文献 S18 基于测试历史信息,用 additional 贪婪算法进行测试用例排序.文献指出,该方法在大部分情况下比传统贪婪算法性能要好,但没有考虑不同测试运行时间的差异,并且当不同程序版本动态执行不同的测试套件时,方法的性能有所下降.

(6) 测试用例相似度.文献 S11 在 2011 年通过计算测试用例与上次识别用例的相似度来进行测试用例排

序,但文献也指出,该方法对只更改文件名称、内容不更新的情况不适用.

测试用例排序的研究方法以多目标优化算法为主,其次是组合算法和其他静态方法.另外,有研究者提出其他的排序方法,包括机器学习方法、贪婪算法等.

2.4.2.2 测试用例选择(RTS)

测试用例选择的文献共 12 篇,研究方法包括基于依赖关系的方法、多目标优化算法和搜索算法.

(1) 依赖关系方法.依据调研文献 S1,S2 对依赖方法的已有分类,将基于依赖关系优化测试集的方法分为静态依赖关系的测试选择和动态依赖关系的测试选择.

有 6 项研究采用静态依赖关系的测试选择,只是依赖的粒度分别在方法级别、类级别、文件级别、模块级别、组件级别和包等不同级别.2013 年,Soetens 等人基于代码变化和测试的方法级别依赖关系实现测试用例选择,并完成工具 ChEOPSJ.但是文献指出,该方法假设调用和被调用是一对一的关系,选择子集不太安全,多态情况下会遗漏相关用例.接着,文献 S10 和文献 S30 在 2015 年基于测试历史分别分析包级别和模块级别代码变化和测试历史的相关性进行测试用例选择.但是方法粒度较粗,精度比较低.在 2016 年,文献 S25 基于静态依赖关系,实现了方法级别和类级别测试用例选择,并与动态的 Ekstazi 方法进行对比.文献指出,类级别选择技术与 Ekstazi 的性能差不多,偶尔不安全;改进静态方法,也可能获得比较准确的测试覆盖,并且成本效益比动态方法要好.接着,Vöst 等人在 2016 年基于组件级别依赖关系的代码变化测试用例选择. 2017 年,Marko 等人在测试执行前,提取静态文件级别和模块级别的依赖关系进行测试用例选择,该方法是扩展动态测试用例选择工具 Ekstazi 在.Net 环境的版本.

在动态依赖关系的测试选择中,2 项研究基于文件级别,另外 2 项研究基于方法级别.具体来讲,2015 年,Gligoric 等人基于动态文件依赖关系进行测试用例选择,并实现工具 Ekstazi.文献指出,该方法比方法级别和类级别效率要高,方法级别测试用例选择虽然选择的测试用例集小,但却降低了测试用例选择前分析的速度,而类级别方法加快了测试用例选择分析的速度,但是选择的测试用例集较大.随后,2017 年,文献 S38 解决异构环境下跨语言和多个 Java 处理器情况下,通过截取操作系统内核的系统文件的调用关系来获取文件级别测试依赖.文献指出,该方法比 Ekstazi 能够收集更多的依赖信息,跳过更多的测试,节约更多的时间.另外,2016 年文献 S1 和文献 S2 在方法级别上,比较了基于静态和动态依赖关系的测试用例选择.研究指出,静态依赖方法比动态方法易操作,耗时短,但偶尔会遗漏相关测试;动态依赖关系方法选择精度较高,但运行时间长、第三方非预期中断等可能导致不安全测试选择,对实时系统和大型系统不适用.

(2) 多目标优化算法.S39 在 2015 年以最大化覆盖、最大化多样性和最小化测试执行时间为优化目标,使用 Additional 贪婪算法和 NSGA-II 搜索算法进行测试用例选择.

(3) 其他静态方法.文献 S13 在 2015 年提出基于历史测试估算测试执行成本的测试选择策略 THEO.文献指出,该方法是动态自适应的测试选择策略,能够依据测试执行上下文自动调整测试的成本估算,并且只需要测试历史数据,无需测试覆盖或者依赖等信息,适用于大型项目;但是跳过的测试可能造成缺陷被延迟发现.

测试用例选择主要采用静态依赖关系和动态依赖关系的方法.这两种方法各有其优缺点,粒度不同,性能存在差异.例如,在动态依赖关系中,文件级别方法比方法级别和类级别方法效率要高.另外,改进静态依赖测试选择方法,也可能获得比较准确的测试覆盖,并且成本效益更好.此外,研究者逐渐考虑采用多目标优化算法、基于测试成本模型等方法进行测试用例选择.

2.4.2.3 测试套件减少(TSR)

测试套件减少共有 3 篇文献,不同的文献使用不同的研究方法.研究方法包括基于依赖关系方法、贪婪算法和组合算法(贪婪算法+机器学习算法).具体介绍如下.

2011 年,文献 S37 采用二元决策森林 BDFs 和贪婪算法,通过识别冗余测试来减少测试用例数和降低集成异构系统的测试成本.文献指出,该方法优于传统的统计学方法,能够识别测试成本高的冗余测试用例,但在系统全生命周期的数据集上可能精度不高.后来,文献 S6 在 2014 年采用静态的依赖关系,降低冗余测试来减少测试套件.文献指出,该方法能够解决静态依赖关系方法在多态情况下依赖收集的问题,并且提高了测试选择精

度.但是选择更大的测试集,增加了总测试时间.随后,文献 S7 在 2017 年采用贪婪算法减少与代码变化无关的依赖,从而减少测试.文献指出,该方法只需历史构建信息,但仅仅降低了 17.09%的测试,性能有待提升.以上研究方法,有 2 项只关注了减少测试规模或者缺陷检测能力,只有 S6 同时考虑了减少测试规模和缺陷检测能力.可以参考已有研究[14,15],在减少测试用例时,这些研究还考虑了测试执行成本、缺陷检测率和测试覆盖,并且取得了较好的效果.

测试套件减少主题的研究只有 3 项,其研究方法包括基于依赖关系方法、贪婪算法和组合算法.

2.4.2.4 测试用例生成(TCG)

测试用例自动生成包括 2 篇文献,使用的研究方法包括搜索算法和其他动态算法.具体介绍如下.

文献 S19 在 2007 年基于随机测试和演化变异,通过测试代理自动生成测试用例,从而减少持续集成过程中人工测试的工作量.随后,文献 S22 在 2014 年使用类的上下文信息,采用搜索算法生成测试用例.文献指出,该方法改进了分支覆盖和未申明的异常,节约 83%的测试时间;但只是在原型系统实现,很多地方还需要改进.

测试用例自动生成只有 2 项研究,研究方法包括搜索算法和基于随机和演化变异生成测试用例.可以尝试使用自动测试用例生成来提高持续集成测试效率.例如,Qu[16]基于代码变化自动生成单元测试用例,并在此基础上进行测试用例选择和排序,该方法提高了持续集成测试集优化的效率,并且降低了开发人员工作量.

2.4.2.5 混合优化方法(Mix)

混合优化共有 8 篇文献,其中混合测试用例选择和排序的研究有 7 篇,混合测试用例排序和测试套件减少的研究有 1 篇.该主题的研究方法包括贪婪算法、机器学习、组合算法和其他动态算法.具体介绍如下.

(1) 贪婪算法.文献 S15 在 2012 年基于代码变化和覆盖信息,最大化代码覆盖为目标进行测试用例选择和排序.文献指出,该方法适用于 C/C++代码环境,操作简单,但精度不高,部分相关测试被遗漏.随后,文献 S16 在 2016 年基于时间约束和测试历史,使用贪婪算法进行测试用例选择和排序.文献指出,该方法在不降低测试质量情况下,减少夜间持续构建时间,但没有考虑测试用例年龄因素.后来,文献 S14 在 2017 年采用重叠感知贪婪算法进行测试用例选择和排序,文献指出,该方法考虑了覆盖重叠,比传统方法高于 50%的覆盖率,但是选择的测试用例集不太安全,可能遗漏相关测试.

(2) 机器学习.首先,文献 S3 在 2017 年基于测试历史信息,采用强化学习方法实现测试用例排序.该方法是轻量级的,只需测试历史数据.同年,文献 S8 在 2017 年通过计算代码变化和测试用例距离来排除不太可能出错的测试用例,从而减少谷歌持续集成测试时间.但文献指出,该研究可能不能推广到其他软件组织,因为不同的组织有不同的测试准则、质量控制流程(例如提交前和提交后策略).

(3) 组合算法.文献 S26 在 2012 年基于非代码覆盖和缺陷检测能力,提出用整数线性规划进行测试用例选择,用贪婪算法对选择测试用例的排序.文献指出,该方法在度量缺陷检测率时,没有考虑不同技术在不同平台上测试的运行时间.随后,文献 S9 在 2017 年基于缺陷检测能力和基于需求覆盖,使用多目标优化排序算法 Rocket 和搜索算法 Flower 优化测试集.文献指出,该方法相比工业实践,提高了缺陷检测能力和需求覆盖,并缩小了测试套件规模,但是没有考虑减少后测试套件的执行时间.

(4) 其他动态算法.文献 S5 在 2014 年针对传统的基于代码覆盖方法,对于大型系统而言,因为系统代码量大而且变化太频繁,存在动态地收集覆盖信息不可行的问题,据此提出了基于测试历史时间窗口进行测试用例选择和排序的方法.该方法只需要测试历史信息,适用于大型项目的测试用例集优化.

混合优化研究,主要采用贪婪算法、机器学习算法和组合算法类型的研究方法.另外,研究提出,传统的基于覆盖的测试用例集优化方法,在大型系统动态地收集覆盖信息变得不可行.有研究者提出基于测试历史的方法(例如基于测试历史时间窗口进行测试选择和排序),来解决大型项目的测试用例集优化问题.

2.4.3 RQ3.3 不同研究方法使用影响因子的偏好

图 8 展示了不同类型的研究方法使用的影响因子类型频率,气泡大小表示频率的高低.可以看出,不同类型的研究方法主要使用的影响因子类型存在不同的偏好.例如,基于依赖关系的方法主要使用代码变化和依赖关系类的影响因子;贪婪算法类的方法主要使用代码覆盖、成本因素和测试历史类的影响因子;多目标优化算法

类的方法主要使用测试历史和非代码覆盖类的影响因子.另外,不同类型的方法覆盖的因子类型数量也不一样,基于依赖关系的方法共使用了 4 种类型的因子,贪婪算法和多目标优化算法类的方法分别使用了 7 种类型的因子,机器学习算法使用了 6 种类型的因子.但测试历史、非代码覆盖、成本因素、代码覆盖和代码变化是各种类型的方法共同使用的因子类型.

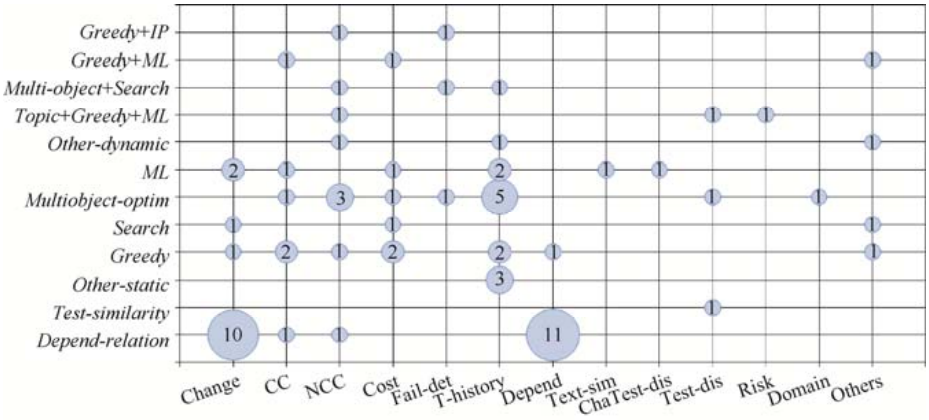

Fig.8 The frequency of factor types in different methods

图 8 不同类型的研究方法使用的影响因子类型频率

本节分析不同研究方法使用的影响因子类型偏好,可以得到:不同类型的方法使用的影响因子类型存在不同偏好,但也存在各类型的研究方法共同使用的因子类型,包括测试历史、非代码覆盖、成本因素、代码覆盖和代码变化.

2.4.4 小 结

本节首先识别调研文献中的研究方法并对其归类;然后调研不同研究主题使用的研究方法偏好及方法优缺点,发现不同研究主题使用不同类型的研究方法.总结如下:已有研究指出,传统的基于代码覆盖的方法,在大型系统因为系统代码量大而且变化太频繁,动态地收集覆盖信息变得不可行,有研究者提出,用基于测试历史的方法来解决大型系统的持续集成测试用例集优化问题是一个可行的途径.另外,静态依赖测试选择和动态依赖测试选择各有其优缺点,但研究表明,改进静态依赖测试选择方法,也可能获得比较准确的测试覆盖,并且成本效益更好.最后,分析研究方法使用的影响因子类型偏好,发现不同类型研究方法使用不同类型的影响因子;但也存在各种类型的研究方法共同使用的因子类型.

2.5 RQ4 研究对象

本节主要讨论调研文献的研究对象,包括应用场景、数据来源和数据集规模.本研究把数据来源分为:工业数据和开源数据,遵循 Ghani[11]和 Engstr[17]中提出的根据代码行数、代码类个数和功能点数来判断数据集的规模,将数据集规模分为以下 3 类.

- 小型数据集(S):代码行数≤2Kloc,或代码类个数≤100,或功能点数<1000;
- 中型数据集(M):2Kloc<代码行数<100Kloc,或 100<代码类个数<1000,或 1000<功能点数<50000;
- 大型数据集(L):代码行数≥100Kloc,或代码类个数≥1000,或功能数≥50000.

2.5.1 RQ4.1 应用场景的编程语言

在调查文献中,持续集成测试用例集优化应用场景的编程语言包括 Java、C\C++、多语言.图 9 展示了研究文献中应用场景的编程语言分布情况.其中,没有说明编程语言环境的研究有 13 篇,占总文献的 33%.这个分布说明,除了没有语言说明的文献,Java 环境下应用研究是主要关注点;其次是应用在大型复杂集成环境中的多语言持续集成测试用例集优化.

在多语言的研究中,文献 S4 解决多种编程语言(包括 Java,PL/SQL,JavaScript,Apex,XML)开发项目的持续

集成测试用例排序问题,但仅仅在 salesforce 项目(中型)上进行了验证,可能在大型集成系统中不适用;文献 S7 考虑在 C#、C++、Powershell 等场景下,减少模块级别无用的测试依赖来降低测试套件,应用在微软云构建系统;文献 S5 和文献 S8 在谷歌多语言环境(包括 config,C,Go,GWT,Java,Python,Shell 和 WEB)下,优化持续集成测试用例集.但该研究的结论可能不能推广到其他软件组织,因为不同组织的测试准则、质量控制流程存在差异.文献 S3 和文献 S13 是基于测试历史的测试用例集优化方法,该方法可应用到多语言环境,但主要考虑测试历史的影响因子,方法的性能有待提升.综上所述,多语言环境已有 6 项研究,有 4 项研究仅在单个数据集上进行了验证,可能不能推广到其他组织;另外 2 项研究方法的性能有待提升.

图 10 展示了持续集成测试研究中,不同编程语言应用随着年份的演化情况.可以看到:在最近 4 年中,Java 环境下的应用研究数量逐年增长,并且 2017 年有高于 4 篇的趋势(搜索截止到 8 月份,2017 年的文章可能不完整);多语言环境下的研究在近 3 年呈现逐年增加的趋势;其他单语言(例如 C/C++、.Net)的研究每年最多只有 1 项.

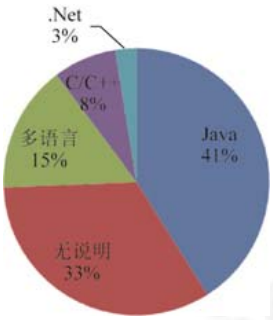

Fig.9 The distribution of programming languages in application scenarios of the selected studies

图 9 在研究文献中应用场景的编程语言分布

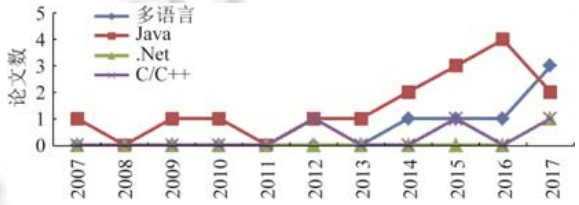

Fig.10 The evolution of programming languages in the application scenarios

图 10 应用场景的编程语言随年份的演化情况

除了没有语言说明的文献,调研文献应用场景的编程语言以 Java 为主,其次是多语言;其他语言的研究关注得较少.多语言环境下的研究在近 3 年呈现逐年增长的趋势,但已有的研究仍然面临着一些问题:67%的研究仅在单个数据集上进行了验证,可能不能推广到其他组织;另外 33%的研究性能有待提升.

2.5.2 RQ4.2 数据来源

图 11 和图 12 分别展示了在调研文献中工业数据和开源数据的来源以及使用频率的排序.这个排序说明,持续集成测试用例集优化在主流的 IT 企业(例如谷歌、微软、思科等)和开源社区(例如 Github、Apache、SIR 等)得到了广泛关注,该研究问题是目前工业界和开源社区共同面临的挑战.

Fig.11 The order of the industrial data sources

图 11 工业数据的来源

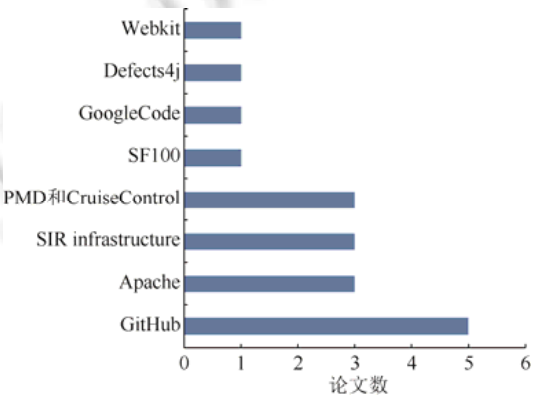

Fig.12 The order of the open-source data sources

图 12 开源数据的来源

在调研文献中,使用工业数据的研究,包括工业研究和非工业研究.工业研究指的是在工业环境中开展的研究,包括收集工业数据、在工业环境中验证提出的研究方法;非工业研究是仅仅使用工业界提供的公共数据集进行的研究,并且未在工业环境中加以验证.

表 8 列出了在调研文献中使用工业数据的研究分布情况.可以看出,在工业数据的研究中,有 24 项(占92.3%)属于工业研究,只有 2 项研究属于非工业研究.在工业研究中,由学术界和工业界合作研究占总文献比例的 58.3%,工业界单独完成的研究占总文献比例的 33.3%,学术界单独完成并在工业界验证的研究占总文献比例的 9.4%.另外,从时间分布来看,最近 3 年由学术界和工业界合作研究方式呈现增长的趋势,到 2017 年达到顶峰,有 5 篇文献.

Table 8 The distribution of research using industrial data

表 8 使用工业数据的研究分布

研究类型	研究者来源	年份	文献编号	数量	合计
工业研究	工业界和学术界	2011	S35,S17	2	14
		2014	S5	1	
		2014	S22	1	
		2015	S10,S30	2	
		2016	S18,S4,S16	3	
		2017	S8,S7,S3,S14	4	
		2017	S27	1	
	工业界	2007	S19	1	8
		2013	S36	1	
		2015	S23,S13	2	
		2016	S32,S2	2	
		2017	S34,S9	2	
	学术界	2011	S37	1	2
		2016	S21	1	
非工业研究	学术界	2015	S12	1	2
		2016	S1	1	

本节分析研究数据的来源,发现持续集成测试用例集优化问题得到了主流的 IT 企业(谷歌、微软和思科等)和开源社区(Github、Apache、SIR 框架等)的广泛关注,该问题是工业界和开源社区共同面临的挑战.另外,在使用工业数据的研究中,92.3%的研究属于工业研究,并且以工业界和学术界合作为主,这种研究方式在最近 3 年呈现逐年增长的趋势.

2.5.3 RQ4.3 数据集规模

在案例研究/实验中,数据集是评估方法有效性的重要因素,本文调研了已有研究的数据集规模.图 13 展示了调研文献中数据集的分布情况,该分布主要以工业项目为主,其次是开源项目,8%的研究同时在开源项目和工业项目上进行了验证;另外,有 1 篇文献(占 2%)只提出了研究方法,没有在数据集上进行验证.

图 14 展示了在调研文献中数据集的规模.可以看出,仅有 7 项(占总文献比例的 18%)研究使用大型工业数据集,其他在中小型数据集上验证的研究,可能在大型集成项目上并不适用.例如文献 S2 指出,动态依赖关系的方法精度高,但运行时间长,对大型系统不适用.

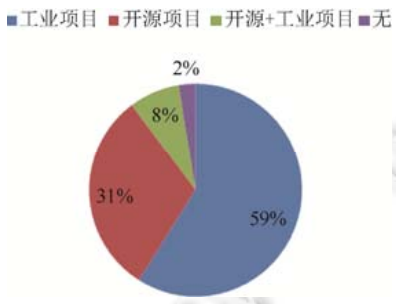

Fig.13 The distribution of the datasets

图 13 调研文献的数据集分布情况

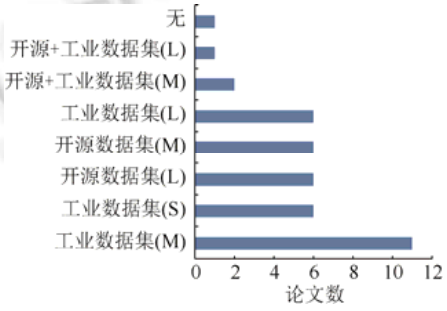

Fig.14 The size of the datasets in the literature

图 14 调研文献的数据集规模

已有研究指出,部分研究工作仅在单个数据集上进行验证,可能不能推广到其他组织.另外,持续集成过程中频繁地构建,产生的数据需占用大量资源,一般只保留近期的构建记录.因此,研究数据集的收集需要一段时间的累积,应当鼓励研究者提供公开数据集,为其他研究者提供比较的数据.例如,谷歌提供了公开数据集GSDTSR,该数据集包含多个项目在 30 天内共 350 万条持续集成测试历史数据.

最后,本节分析了不同研究主题选择的数据集规模偏好(如图 15 所示),气泡大小表示频率的高低.可以看出:不同研究主题选择的数据集规模存在不同的偏好,例如测试用例排序主要选择中型数据集;测试用例选择主要选择大型和中型数据集.

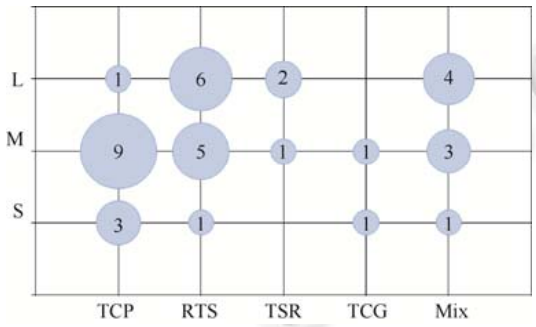

Fig.15 The mapping between the research topics and the sizes of the datasets

图 15 研究主题和数据集规模的关系

本节分析调研文献的数据集分布和规模发现,已有研究主要集中在工业界,其次是开源社区;但是只有 18%的研究使用大型工业数据集,其他的研究在中小型数据集上进行了验证,可能不能推广到其他大型集成项目.另外,部分研究工作仅在单个数据集上进行验证,可能不能推广到其他组织.最后,可以看出,不同研究主题在选择数据集的规模上存在不同的偏好.

2.5.4 小 结

调研已有研究的应用场景发现,以 Java 应用场景为主,多语言环境下的研究近 3 年呈现逐年增长的趋势,但总体所占的比值(15%)仍然比较低,并且已有的多语言研究仍然存在一些问题.另外,从数据集来看,已有的研究主要集中在工业界,其次是开源社区;但只有 18%的研究在大型工业数据集上进行了验证,其他研究仅在中小型数据集上进行了验证,可能在大型工业集成项目上并不适用.

2.6 RQ5 性能评价

本节识别已有研究的评价指标,对其分类,然后分析不同研究主题使用的评价指标并评价已有研究方法的性能.在 39 篇调研文献中,有 36 篇文献对研究方法进行了评价.

2.6.1 RQ5.1 评价指标及分类

我们提取调研文献中的评价指标,遵循 Ghani[11]和 Engstr[17]中评价指标的已有分类,将评价指标归为以下 4 类:测试成本、缺陷检测能力、测试覆盖和混成指标.表 9 展示了评价指标及分类信息.

(1) 测试成本.在调研文献中,针对成本的评价,包括总测试时间、减少测试规模等指标.

(2) 缺陷检测能力.在收集缺陷检测能力相关评价指标的基础上,参考 Catal 等人[18]对测试用例排序的缺陷检测能力的分类,包括 APFD、ASFD、NAPFD,缺陷检测比率 TPFD,基于成本的缺陷检测比率 APFDc 和其他;另外,考虑持续集成测试用例集优化领域特定的评价指标,包括 FDR 和 Inclusiveness 等,对缺陷检测能力相关的指标进行归类.其中,文献 S17 自定选择测试集的缺陷检测效率 Eff_{dec}=发现缺陷数/执行测试用例数.

(3) 测试覆盖(coverage).基于覆盖信息评价研究方法的有效性,包括变异覆盖、代码覆盖率、分支覆盖率、失效测试覆盖等;测试覆盖类中使用最多的是变异覆盖.

(4) 混成指标.一些研究直接提出了一些综合测试成本和缺陷检测能力的混成指标.包括文献 S29 自定义成

本和效益评价指标 EVOMO,S26 提出平均缺陷检测率 vs.运行测试数的评价指标.

从表 9 可以看出,总测试时间是受关注最多的评价指标,测试成本和缺陷检测能力两类中有部分指标受关注也较多.此外,可以看到,一些研究采用了多个评价指标,例如文献 S1 关注了减少测试规模和总测试时间、失效测试覆盖和优化后测试集的变异覆盖,这些指标从测试成本、测试覆盖两方面评价了测试优化的性能.

Table 9 The categories of evaluation metrics in the selected studies

表 9 调研文献评价指标及分类

评价指标分类	评价指标	文献编号	频率	比例(%)
测试成本 Cost	总测试时间	S1,S5,S10,S13,S14,S15,S22,S34,S35,S36,S39,S7	12	28.21
	减少测试规模	S1,S6,S7,S9,S21,S24	6	15.38
	测试选择率	S2,S5,S15,S25,S27	5	12.82
	end-to-end time	S24,S25,S27,S38	4	10.26
	跳过测试数量	S13,S38	2	5.13
	总收支成本	S13	1	2.56
	减少测试用例	S20	1	2.56
	检测单个缺陷时间	S5	1	2.56
	减少测试结果的人工审查	S13	1	2.56
	夜间持续集成的持续时间	S16	1	2.56
	减少测试团(test targets)	S8	1	2.56
缺陷检测能力 Fail-detect	缺陷检测率 Fault Detection Ratio(FDR)	S26,S34,S39,S5,S9,S32	6	15.38
	平均缺陷检测率 APFD	S12,S17,S18,S32,S4	5	12.82
	遗漏的缺陷	S13,S16,S37	4	10.26
	F-measure	S10,S2,S30	3	7.69
	召回率 Recall	S20,S4	2	5.13
	精度 Precision	S20	2	5.12
	APDF	S33	1	2.56
	基于缺陷严重程度的平均缺陷检测率 APFDsv	S18	1	2.56
	选择测试集中包含失效用例比例 Inclusiveness	S15	1	2.56
	归一化的平均缺陷检测率 NAPFD	S3	1	2.56
	遗漏的依赖项	S38	1	2.56
	检测的缺陷数量	S6	1	2.56
	损失率	S37	1	2.56
	发现缺陷的时间间隔	S19	1	2.56
	安全性和精度违规比例	S25	1	2.56
	缺陷检测效率 Eff_{dec}	S17	1	2.56
	与排序位置相关的检测缺陷比例	S16	1	2.56
	识别未定义的异常数	S22	1	2.56
测试覆盖 Coverage	优化后测试集的变异覆盖	S1,S19,S20	3	7.69
	代码覆盖率	S14	1	2.56
	分支覆盖率	S22	1	2.56
	失效测试覆盖 Failing test coverage	S1	1	2.56
	配置覆盖率	S34	1	2.56
	需求覆盖率	S9	1	2.56
混成指标	基于成本的平均缺陷检测率 APFDc	S23,S36	2	5.13
	成本和效益评价指标 EVOMO	S29	1	2.56
	平均缺陷检测率 vs.运行测试数	S26	1	2.56

2.6.2 RQ5.2 不同研究主题使用的评价指标

图 16 给出了各研究主题中各文献使用的评价指标类型,调研文献中的混成指标都是综合考虑了测试成本和缺陷检测能力.如图 16 所示,TCP 主题中只关注缺陷检测能力类的文献有 6 篇,同时关注测试成本和缺陷检测能力类的文献有 3 篇,只关注测试成本和同时关注 3 种类型的文献各有 1 篇;RTS 主题中同时关注测试成本和缺陷检测能力类的文献有 6 篇,只关注测试成本的文献有 3 篇,只关注缺陷检测能力、同时关注成本和缺陷检测能力、同时关注 3 种类型的文献各有 1 篇;TSR 主题中的研究只关注测试成本或缺陷检测能力、同时关注成本和缺陷检测能力的文献各 1 篇;TCG 主题中只关注缺陷检测能力和同时关注 3 种类型的文献各有 1 篇;在 Mix 混合主题中同时关注测试成本和缺陷检测能力类的文献有 4 篇,使用其他 3 种评价方法的文献各 1 篇.由此可见,每类研究主题都关注了这 3 类评价指标,TCP、RTS 以及 Mix 混合主题中关注较多的是测试成本和缺陷检

测能力,没有研究单独评价测试覆盖.但 TCP、RTS、TCG 这 3 个主题,以及 Mix 中混合 TCP 和 TSR 都有研究综合考虑了测试成本、缺陷检测能力和测试覆盖.

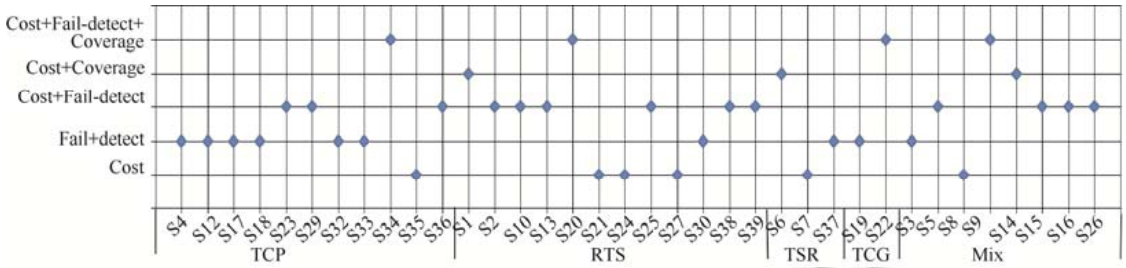

Fig.16 The evaluation methods of the studies in different research topics

图 16 各研究主题中文献的评价指标类型

另外,可以看到,有 16 项研究(占总文献的 41%)只关注了单个类型的指标,16 项研究采用 2 个指标类型的评价,还有 4 篇文献考虑全部 3 种类型.只关注单个类型的评价会存在一定的局限性,不能全面地反映研究方法的性能.例如文献 S21 在评价测试用例选择的效果时,只评价了缩小测试规模,没有评价减少测试后,其缺陷检测能力是否有所下降.但在 Mix 混合主题中只有 2 项研究使用单个类型的评价,并且在混合 RTS 的研究中,有 5 项(占 71.4%)在减少测试后,考虑了缺陷检测能力和测试覆盖.

通过本节的分析可以看出,各研究主题都关注了 3 类评价指标,其中对测试成本和缺陷检测能力的关注较多.调研文献中有 41%的研究只关注了 1 类指标,只有 10%的研究同时考虑了 3 种类型的评价指标.此外,TSR 主题的研究没有综合考虑 3 类指标的评价;而 TCG 主题虽然只有 2 篇文献,但其中一篇综合考虑了 3 类指标的评价.

2.6.3 RQ5.3 比较已有研究方法的性能

本节分析使用频率较高的评价指标.图 17 展示了评价指标按使用频率的排序.图中展示使用频率最高的评价指标是测试成本类中的总测试时间;其次是测试成本类中的减少测试规模和缺陷检测能力类的 FDR;测试成本类中的测试选择率和缺陷检测能力类中的 APFD 在使用频率排序中并列第三;测试覆盖类中使用最多的是优化后测试集的变异覆盖;一些只使用 1 次的评价指标包括缺陷检测能力类中的交叉功能覆盖、分支覆盖和功能点覆盖等.

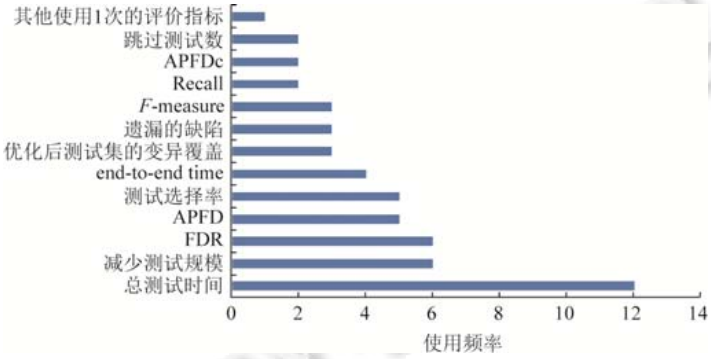

Fig.17 The order of evaluation metrics by frequency in the research literature

图 17 调研文献中常用的评价指标排序

我们分别在 3 类评价指标中选取了使用频率较高的指标进一步加以分析,其中,测试成本和缺陷检测能力类各 2 项,测试覆盖类 1 项,一共 5 个指标.在图 17 中排在第五和第六的两个指标:测试选择率和 end-to-end time 都属于测试成本类,我们在选择时略过了这两个评价指标.图 18 展示了已有研究方法按已选择评价指标的性能排序情况.

(1) 测试成本类的总测试时间.在收集调研文献的总测试时间时,发现文献 S34 没有总测试时间的原始数据,只有与基线对比后改进的数据;类似地,文献 S7 只给出了方法节约 38 天的持续集成服务器运行时间.因此,在比较已有方法的总测试时间时,排除了这两项研究.从该指标来看,RTS 主题中文献 S10 的基于 Narrow 策略静态相关性方法性能最好,该研究方法还关注了缺陷检测能力类的评价,召回率是 79%,但精度只有 7.4%;其次是文献 S1 基于方法级别静态依赖关系的 RTS 方法,该方法还关注了测试覆盖的评价,失效测试覆盖是 50%和变异覆盖只有 39%;TCG 主题中文献 S22 的搜索算法和 Mix 混合主题中文献 S5 基于历史时间窗口进行测试用例选择和排序算法并列排在第三,文献 S22 的研究方法同时关注了 3 种类型的评价指标,在减少了 83%的总测试时间的同时,识别出 69%的未定义异常,并达到 58%的分支覆盖率;文献 S5 的方法在减少测试成本和缺陷检测能力两方面都取得比较好的效果,不仅减少总测试时间性能比较好,而且选择的测试集小,缺陷检测率也比较高.

性能排在最后 3 个的方法依次是文献 S36 基于测试历史和特定领域知识的多目标优化算法 Rocket 进行测试用例排序、文献 S14 基于重叠感知算法进行测试用例选择和排序以及文献 S1 基于方法级别动态依赖关系的 RTS 方法.文献 S36 的方法从总测试时间和基于成本的平均缺陷检测率 APFDc 来评价,与人工测试和随机测试相比,单位测试时间内缺陷检测率更高;文献 S14 的方法从测试成本和测试覆盖来评价,虽然只减少了 13.6%的总测试时间,但是代码行覆盖率可达到 97%;文献 S1 的动态方法关注了测试成本和测试覆盖的评价,虽然只减少 8%的总测试时间,但该方法对失效测试覆盖率能达到 100%,变异覆盖 29%.

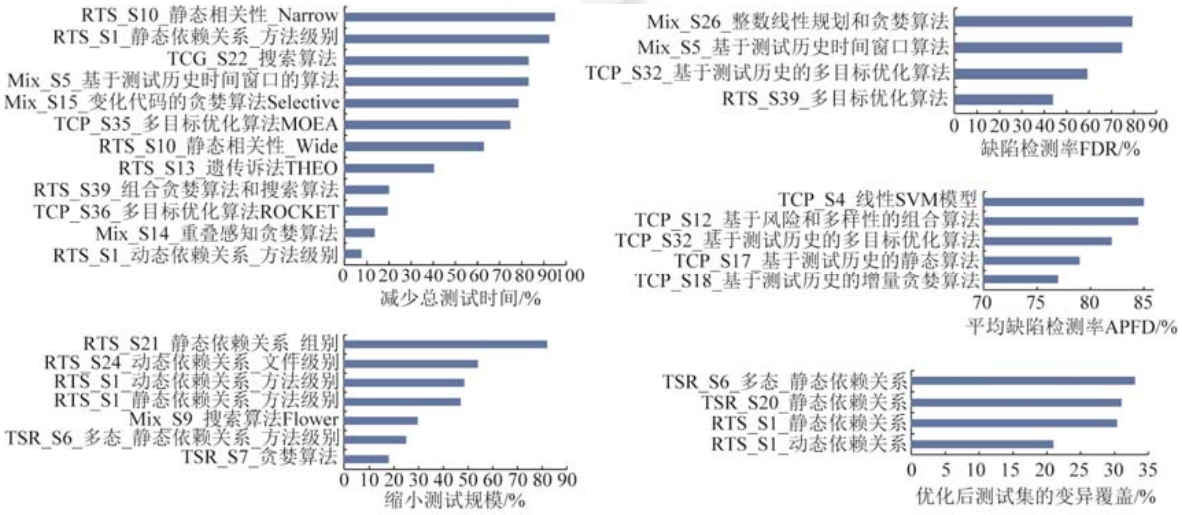

Fig.18 The performance order of methods by the selected evaluation metrics

图 18 已有研究方法按已选择评价指标的性能排序

可以看出,静态依赖关系的方法比动态依赖关系的方法减少测试时间的比例较大,但是缺陷检测能力和覆盖率较低;而动态依赖方法覆盖率高,但总测试时间长.文献 S2 指出,静态依赖关系方法无需运行程序,静态地分析变化代码和测试代码来提取依赖信息,测试时间成本低;而动态依赖关系方法通过测试执行时的调用关系获得依赖信息,导致测试时间成本高,该方法对大型系统和实时软件不适用.因此,静态依赖方法和动态依赖方法各有优缺点,研究者需要依据测试需求在测试成本、缺陷检测能力和测试覆盖方面取得平衡.

(2) 测试成本类的减少测试规模.从该指标看,文献 S21 的基于组件级别静态依赖关系的 RTS 方法性能最好,但该方法仅从减少测试规模来评价,没有验证减少后的缺陷检测能力和测试覆盖;其次是文献 S24 基于文件级别动态依赖关系的 RTS 方法,该方法比方法级别和类级别的方法效率要高,选择的测试用例集小,但是测试时间(end-to-end time)长,并且也没有评价减少后的缺陷检测能力和测试覆盖.排在第三的是文献 S1 基于方法级别

动态依赖关系的 RTS 方法,该方法在减少测试规模和失败测试覆盖率方面的性能比较好,但在减少总测试时间的性能上排在最后.可以看出,这 3 种 RTS 方法都是基于依赖关系的,只是方法的粒度分别在组件、文件和方法等不同级别上,并且方法的粒度越粗,减少测试规模的比例越大;但性能最好的两种研究方法都只关注了测试成本类的评价指标,没有验证测试减少后的缺陷检测能力和测试覆盖.

另外,RTS 主题的 3 种方法比测试套件减少的 3 种方法性能要好.这 3 种 TSR 的方法依次是 Mix 混合主题中文献 S9 使用搜索算法 Flower 来减少测试套件、TSR 主题中文献 S6 基于方法级别的静态依赖关系方法和 TSR 主题中文献 S7 的贪婪算法.文献 S9 同时关注 3 种类型的评价指标,从减少测试规模、FDR 和需求覆盖率来评价方法,该方法提高了缺陷检测能力和需求覆盖,但减少测试规模的比例不大,并且没有考虑减少后测试时间成本.文献 S6 关注测试成本和测试覆盖,从减少测试规模和变异覆盖指标来评价,虽然减少测试规模比例不大,但该方法解决了多态情况下依赖关系的收集问题.文献 S7 只关注测试成本,从总测试时间和减少测试规模来评价,没有考虑减少后的缺陷检测能力和测试覆盖,并且仅降低 17.9%的测试,减少了集成服务器运行时间.进一步分析没有使用减少测试规模的其他研究后发现,混合 TSR 的研究比单个 TSR 研究减少测试比例较大,并且从测试成本、缺陷检测能力和测试覆盖 3 种类型来评价方法.

(3) 缺陷检测能力类的缺陷检测率 FDR.在收集调研文献的缺陷检测率时,发现有两篇文献没有 FDR 的原始数据,只有与基线对比后 FDR 改进的数据,例如文献 S34 提出的研究方法与工业实践相比 FDR 增加 31%~41%,但没有给出方法的 FDR 值.因此,在比较已有方法的 FDR 时,排除了这两项研究.Mix 混合主题中文献 S26 组合整数线性规划和贪婪算法进行测试用例选择和排序的方法性能最好,该方法关注了测试成本和缺陷检测能力,从 FDR 和运行测试数量评价方法,但没有考虑测试时间成本.其次是 Mix 混合主题中文献 S5 的方法,该方法在测试成本和缺陷检测能力的性能上都比较好,并且在减少总测试时间的性能上排在第三.排序第三的方法是 TCP 主题中文献 S32 调整测试历史时间窗口来改进多目标优化算法排序的效果,该方法只关注了缺陷检测能力的评价指标,包括缺陷检测率和 APFD.性能最差的方法是文献 S39 基于测试历史进行多目标优化的 RTS 方法,该方法综合考虑了测试成本和缺陷检测能力的评价,提高了缺陷检测率,但仅降低了 20%的总测试时间.可以看出,Mix 混合主题中研究方法的 FDR 性能比单个主题的研究方法性能要好.

(4) 缺陷检测能力类的平均缺陷检测率 APFD,主要用于评价 TCP 主题的方法性能.可以看出,文献 S4 的线性 SVM 模型的性能最好,该方法只关注了缺陷检测能力的评价,APFD 和 recall 的性能都比较好,用 3%的测试用例可检测出 75%的缺陷;其次是文献 S12 基于测试用例的风险值和多样性组合算法的排序方法,该方法比随机排序方法在快速发布的环境下更有效,但在不同版本之间方法的性能波动比较大,并且方法只评价了 APFD 的性能;排序第三的方法是文献 S32 调整测试历史时间窗口来改进多目标优化算法的排序效果,该方法在 FDR 的性能也排在第三,但是只关注了缺陷检测能力的评价.性能最差的是文献 S18 基于测试历史的 Additional 贪婪算法,该方法从 APFD 和基于缺陷严重程度的平均缺陷检测率 APFDsv 两个方面来评价,但是该方法没有考虑测试时间成本和测试覆盖.另外,基于测试历史排序的 3 种方法 APFD 的性能相对较低,并且这 3 种方法都只关注了缺陷检测能力的评价.

可以看出,TCP 主题中使用 APFD 的研究都只关注了缺陷检测能力的评价,没有考虑测试时间成本和测试覆盖的评价.我们进一步调研该主题的其他文献,发现该主题有 6 项(占 43%)研究都只关注了缺陷检测能力类指标,而在包含 TCP 的混合研究中,有 6 项(占 75%)都综合考虑了测试成本和缺陷检测能力的评价.

(5) 测试覆盖类的优化后测试集的变异覆盖.从该指标来看,RTS 主题中文献 S1 基于方法级别静态依赖关系的方法性能最好,该方法改进了文献 S20 的方法,在减少总测试时间的性能上排在第二,但是失效测试覆盖不如动态依赖关系的方法;其次是 TSR 主题中文献 S6 基于多态情况下的静态依赖关系方法,该方法还关注了测试成本的评价,虽然减少测试规模比例不大,但减少后的变异覆盖性能比较好;排在第三的是 RTS 主题中文献 S20 基于静态依赖关系的方法,该方法同时关注 3 种类指标的评价,召回率是 68%,精度能达到 85%,并且在 75%的情况下能减少测试套件中 99%测试用例;排在最后的是 RTS 主题中文献 S1 基于方法级别动态依赖关系的方法,该方法在减少总测试时间性能上排在最后,但在减少测试规模上排在第三,并且失效覆盖率可达到 100%.可以

看出,静态依赖关系方法比动态依赖关系方法变异的覆盖性能更好,但这 4 种方法都是基于方法级别依赖关系的 RTS 方法,并且选择后的变异覆盖性能都不高.

在测试成本类的评价指标中,除了总测试时间和减少测试规模,测试选择率和 end-to-end time 也有一定的关注.在 end-to-end time 方面,文献 S38 基于文件级别动态依赖关系的 RTS 方法性能最好,该方法考虑收集异构环境下跨语言和多个 Java 处理器情况下的依赖关系,相比文献 S24 的方法能收集更多依赖信息,跳过更多测试,节约更多时间.在测试选择率方面,文献 S5 的方法性能最好,并且在减少总测试时间和缺陷检测率的性能也比较好.其次是文献 S27 基于模块级别静态依赖关系 RTS 方法,该方法在测试选择率和 end-to-end time 上都排序第二,只选择 28.5%的测试,并且减少 46.6%的 end-to-end time,但没有考虑选择测试集的缺陷检测能力和测试覆盖.接着,文献 S25 基于类级别静态依赖关系的 RTS 方法,在测试选择率和 end-to-end time 上都排序第三,该方法比方法级别静态依赖关系方法精度要高,选择测试集小,但是偶尔不安全,并且只关注了测试成本和缺陷检测能力.

在以上的评价指标中,至少在两个指标的性能同时排在前三的研究有 5 项:Mix 混合主题中文献 S5 基于测试历史时间窗口进行测试用例选择和排序,该方法在减少总测试时间、测试选择率和缺陷检测率方面的性能都比较好;有 2 项研究虽然在一些指标上性能比较好,但只关注了单个类型的指标,例如 TCP 主题中文献 S32 调整测试历史时间窗口来改进多目标优化算法在 FDR 和 APFD 上的性能都比较好,但只关注了缺陷检测能力;其他 2 项研究都是在测试成本、缺陷检测能力和测试覆盖中取得平衡,例如文献 S1 基于方法级别静态依赖关系的 RTS 方法,该方法在减少总测试时间和变异覆盖方面的性能都比较好,但是失效测试覆盖率不如动态依赖关系的方法.可以看出,在这 5 种方法中,只有文献 S5 的方法在各评价指标的性能上都比较好,其他 4 种方法中一些只关注单个类型的指标,另一些是在 3 种类型评价中取得平衡.

在对比已有方法的性能后,我们进一步调研性能排在前三的研究方法使用的影响因子(见表 10).

Table 10 The factors of the top 3 studies in the performance order

表 10 性能排在前三的研究方法使用的影响因子

评价指标	研究方法	影响因子及类型
减少总测试时间	Top1:RTS_基于相关性_Narrow	制品变化{代码变化},依赖关系{代码变化和测试代码相关性}
	Top2:RTS_静态依赖关系	代码覆盖{类覆盖,方法覆盖,分支覆盖,代码行覆盖},非代码覆盖{变异覆盖},依赖关系{静态依赖关系_方法级别}
	Top3:TCG_搜索算法	制品变化{提交信息},其他{异常信息}
	Top3:基于测试历史时间窗口 RTS 算法	测试历史{历史测试失败窗口,历史测试执行窗口}
减少测试规模	Top1:RTS_静态依赖关系_组件	制品变化{代码变化},依赖关系{组件之间依赖关系}
	Top2:RTS_动态依赖关系_文件级别	制品变化{代码变化},依赖关系{动态依赖关系_文件级别}
	Top3:RTS_动态依赖关系_方法级别	依赖关系{动态依赖关系_方法级别},代码覆盖{类覆盖,方法覆盖,分支覆盖,代码行覆盖},非代码覆盖{变异覆盖}
缺陷检测率	Top1:Mix_整数线性规划的非贪婪算法	非代码覆盖{变异覆盖},缺陷检测能力{用例的缺陷检测能力分值}
	Top2:Mix_基于测试历史时间窗口算法	测试历史{历史测试失败窗口,历史测试执行窗口}
	Top3:RTS_基于测试历史的多目标优化算法	成本因素{时间约束},测试历史{测试失效概率,测试失效时间窗口}
APFD	Top1:线性 SVM 模型	代码覆盖{测试覆盖分值},制品变化{文本变化},文本相似度{修改文本和测试相似度},测试历史{测试用例年龄}
	Top2:基于风险驱动和多样性排序	用例间距离{测试用例间字符串距离},用例风险值{测试风险值},非代码覆盖{主题覆盖}
	Top3:基于测试历史的启发式算法	测试历史{历史失败窗口大小,测试失效概率},成本因素{时间约束}
优化后测试集的变异覆盖	Top1:RTS_静态依赖关系_方法级别	代码覆盖{类覆盖,方法覆盖,分支覆盖,代码行覆盖},非代码覆盖{变异覆盖},依赖关系{静态依赖关系_方法级别}
	Top2:TSR_多态_静态依赖关系_方法级别	制品变化{代码变化},依赖关系{静态依赖关系_方法级别}
	Top3:RTS_静态依赖关系_方法级别	制品变化{代码变化},依赖关系{静态依赖关系_方法级别}

(1) 减少总测试时间排在前三的研究方法可以看出,不同的研究方法使用了不同的影响因子.

(2) 减少测试规模上排在前三的研究方法使用的影响因子非常相近,这 3 种方法都采用了测试对象的依赖关系,只是依赖的粒度分别在组件、文件或者方法等不同级别.

(3) 缺陷检测率排在前三的研究方法可以看出,排在第二和第三的研究方法都使用了测试历史类的影响因子;性能最好的研究方法则使用非代码覆盖和缺陷检测能力类的影响因子.

(4) 对比 APFD 排在前三的研究方法可以看出,考虑的影响因子类型越多,缺陷检测性能越趋于良好.例如,线性 SVM 模型综合考虑测试覆盖、制品变化、文本相似度和测试历史等类型的因子,方法性能最好.另外,这 3 种研究方法使用了不同的影响因子.

(5) 对比优化后测试集的变异覆盖排在前三的研究方法可以看出,这 3 种研究方法都使用了静态依赖关系类的影响因子.

本节选取测试成本、缺陷检测能力和测试覆盖中的使用频率较高的评价指标,比较已有研究方法的性能以及分析性能排在前三的研究方法使用的影响因子,总结如下:在基于依赖关系的 RTS 方法中,静态依赖关系的方法和动态依赖关系的方法在减少测试成本和测试覆盖上各有优缺点,研究者需要根据测试需求在这两者中取得平衡;并且依赖关系的粒度越粗,减少测试套件的比例越大;从 APFD 来看,TCP 主题中 43%的研究只关注了缺陷检测能力;考虑异构环境下跨语言和多个 Java 处理器情况下的依赖关系的方法,在减少测试成本和缺陷检测能力的性能都比较好.另外,从 FDR 来看,Mix 混合主题的研究方法比单个主题的研究方法性能要好;并且 75%的混合研究都从测试成本和缺陷检测能力等方面评价了方法的有效性.最后发现,性能排在前三的研究方法,在大部分情况下,都使用了不同类型的影响因子.

2.6.4 小 结

本节首先识别调研文献中常用评价指标,并进行归类.接着分析不同研究主题使用的评价指标,发现各研究主题都关注缺陷检测能力和测试成本类的指标,但各个研究主题的评价方法各有侧重,TCP 主题的研究主要关注缺陷检测能力的评价,RTS 和 Mix 研究主题的大部分研究同时关注测试成本和缺陷检测能力的评价;另外,41%的研究只关注了单个类型的评价指标.最后,我们选取 3 种类型中使用频率较高的 5 个评价指标,比较已有研究方法的性能,并分析性能排在前三的研究方法使用的影响因子.具体来讲,

- 在测试成本类的减少总测试时间方面,文献 S10 基于 narrow 策略的静态相关性 RTS 方法的性能最好,并且召回率比较高,但精度低;文献 S1 动态依赖关系 RTS 方法的性能最差,但失效测试覆盖率高.通过分析得到,静态依赖关系方法无需运行程序,通过分析代码来提取依赖信息,比动态依赖关系方法减少总测试时间比例大及变异覆盖率高,但是失效测试覆盖率较低;而动态依赖关系的方法通过测试执行时的调用关系获得依赖信息,导致总测试时间长和变异覆盖率低,但失效测试覆盖率高.另外,排在前三的研究方法使用了不同的影响因子.
- 在测试成本类的减少测试规模方面,性能最好的两种研究方法只关注了测试成本类的评价指标,没有验证测试减少后的缺陷检测能力和测试覆盖.另外,RTS 主题的 3 种方法比测试套件减少的 3 种方法性能要好,RTS 的方法都采用了测试对象的依赖关系,只是方法的粒度分别在组件、文件和方法等不同级别上,并且依赖的粒度越粗,减少测试规模的比例越大;混合 TSR 的研究比 TSR 主题的研究减少测试比例大,并且综合测试成本、缺陷检测能力和测试覆盖来评价方法,而 TSR 主题的研究都没有综合考虑 3 类指标的评价.
- 在缺陷检测能力类的缺陷检测率方面,Mix 混合主题中文献 S26 的组合整数线性规划和贪婪算法性能最好,但该方法没有评价优化后的总测试时间;RTS 主题中文献 S39 的多目标优化算法性能最差,但该方法提高了缺陷检测率;而排在第二的 Mix 混合主题中文献 S5 的方法在测试成本和缺陷检测能力两方面都取得比较好的性能.可以看出,Mix 混合主题中研究方法的 FDR 比单个主题的研究方法性能要好.另外,不同研究方法使用了不同的影响因子.
- 在缺陷检测能力类的平均缺陷检测率方面,线性 SVM 模型排序方法的 APFD 和 recall 都比较高,基于测试历史排序的 3 种方法的性能相对较低,并且这 3 种方法只使用了缺陷检测能力类的指标;进一步分析发现,TCP 主题中 43%的研究都只关注了缺陷检测能力的评价,而在包含 TCP 的混合研究中,75%的研究都综合考虑了测试成本和缺陷检测能力的评价.另外,可以看出,使用的影响因子类型越多,方法的缺陷检测能力越趋于良好;排在前三的研究方法使用了不同的影响因子.
- 在测试覆盖类的优化后测试集的变异覆盖方面,RTS 主题中文献 S1 基于方法级别静态依赖关系的方法

性能最好,文献 S1 动态依赖关系的方法性能最差;对比已有研究发现,静态依赖关系方法比动态依赖关系方法的变异覆盖性能要好.另外,测试覆盖类中使用最多的评价指标是变异覆盖,并且调研文献中关注变异覆盖的研究都是基于依赖关系的 RTS 方法,但优化后测试集的变异覆盖性能都不高.

基于以上的评价指标,性能同时比较好的研究方法只有 1 项;在其他研究中,一些只关注了部分类型的评价指标,另一些是在测试成本、缺陷检测能力和测试覆盖 3 类中取得平衡.

3 研究的局限性

本文的系统调研存在两方面的限制:(1) 搜索文献可能存在误差;(2) 提取数据可能不够准确.

首先,为了确保在选择文献过程的误差,前期制定了一个搜索策略,并提出研究问题.基于这些问题,设定关键字符串来查找相关的文献.但是,基于关键词的搜索是主观的,可能存在遗漏一些文献的情况.因此,本文进一步采用滚雪球式的方法,人工地搜索出相关的文献作为自动搜索的补充.尽管如此,在本文的研究范围内,可能还会遗漏一些相关的文献.

其次,提取数据的过程可能存在一些不准确的情况.在数据抽取过程中,本研究预定义提取表单,该表单定义需提取的字段信息与研究问题的对应关系(见表 3).另外,数据提取工作由本文第一作者和两名学生分别完成,然后进行交叉检查,任何不确定性或不一致的数据都进行讨论并达成一致.但在这个过程中,由于缺乏足够的信息导致数据提取可能不够完整.更具体地表现为:实验方法没有充分的说明,验证方法存在的问题也没有介绍,对实验结果的分析没有很好地解释等,这些都会影响数据提取的准确性.

4 问题与建议

本文采取系统文献调研的方法,回顾了持续集成测试用例集优化的研究进展,希望为该领域的研究者提供一些有用的发现.基于前面的分析和总结,我们给出了该领域当前面临的问题和我们的建议,包括:

(1) 不同研究主题面临的问题

测试用例排序 TCP 研究主题,该主题的文献共 14 篇,占所有文献的 35.9%.在该主题的研究中,综合考虑已有影响因子的线性 SVM 模型,在平均缺陷检测率 APFD 和召回率 recall 方面的性能都比较好.研究指出,传统的基于代码覆盖的方法,在大型系统因为系统代码量大并且变化频繁、动态地收集覆盖信息变得不可行.有研究者提出基于测试历史的方法来解决这个问题,但基于测试历史方法的 APFD 相对较低,并且没有考虑测试时间成本和测试覆盖.因此,大型项目的测试用例排序仍然是该领域面临的重要挑战.另外,TCP 主题中 43%的研究都只关注了缺陷检测能力类的指标,没有考虑对测试成本和测试覆盖的评价.

测试用例选择 RTS 研究主题,该主题的文献共 12 篇,占所有文献的 30.77%.已有研究指出,动态依赖关系方法通过测试执行时的调用关系获得依赖信息,能够得到一个比较准确的测试集,但需要长时间的运行测试,导致较高的分析成本;另外,非确定性的中断引起不安全的测试选择,这些限制了该技术在实际中的应用,特别是在实时系统和大型项目上的应用;而静态依赖关系方法无需运行测试,静态地分析变化代码和测试代码来提取依赖信息,分析成本低,并且比动态依赖测试选择更易于操作,但是偶尔会遗漏测试.另外,从减少测试规模来看,性能最好的两种 RTS 方法都没有考虑选择测试集的缺陷检测能力和测试覆盖.

测试套件减少 TSR 研究主题,目前只有 3 项研究,占所有文献的 7.69%.该主题的研究都没有综合考虑测试成本、缺陷检测能力和测试覆盖的评价.

测试用例生成 TCG 研究主题,研究者关注的比较少,目前只有 2 项研究,占所有文献的 5.13%,但其中一篇综合考虑了 3 类指标的评价.

混合优化 Mix 研究主题,该主题的文献共 8 篇,占所有文献的 20.51%.从缺陷检测率 FDR 来看,Mix 混合主题的研究方法比单个主题的研究方法性能要好,例如文献 S5 提出基于测试历史时间窗口进行测试用例选择和排序,在减少测试成本和缺陷检测率方面的性能都比较好.

从评价方面来看,TCP 主题的研究主要关注缺陷检测能力的评价,RTS 和 Mix 主题的大部分研究同时关注

测试成本和缺陷检测能力的评价;在调研文献中,41%的研究只关注了单个类型的评价指标,但测试成本、缺陷检测能力和测试覆盖任意一种类型的评价都不够充分度量测试的质量,而在 Mix 混合主题中,75%的研究关注两个及两个以上类型的评价.

建议:

● 持续集成测试用例集优化的已有研究积累了较多的研究成果,在不同研究主题的研究取得了突破,但仍然存在不足,诸如时间成本太高、遗漏一些必要的测试,目前还少有研究在减少测试成本的同时,考虑如何保证缺陷的检测能力和测试的覆盖度.因此,综合考虑这三者的性能,研究具有较好成本效益的持续集成测试集优化方法,具有非常好的价值和应用前景.

● 已有研究提出的方法对大型项目,由于耗时、复杂等原因,限制了它的有效应用,但大型项目由于系统复杂、庞大,无选择地运行所有集成测试非常不经济,甚至不可能,所以更需要测试用例的优化选择.因此,大型项目下持续集成测试集优化仍然是目前该领域所面临的挑战.

● 动态方法追求精准测试选择,但由于耗时、第三方非预期中断等可能导致不安全或者不完整的测试;静态方法由于分析方法的局限性,有可能导致过多或者过少的测试,两者都有优缺点.但近年的研究 S25 表明,改进静态方法也可能获得全面的测试覆盖,并且成本效益更好,所以基于静态技术测试优化选择,可能是未来一个有价值的研究方向.另外,我们认为融合静态和动态的方法,譬如以静态技术搜索全局,以动态技术优化局部,也可能是一个值得研究的方向.

(2) 异构环境下的研究:研究文献指出,随着 Web、服务应用等程序的广泛使用,这类异构系统与传统软件存在两方面的差异:异质性和环境依赖.在调研文献中,多语言环境下的研究占调研文献的 15%,并且已有研究仍然存在一些问题.另外,已有研究中只有两项考虑了复杂异构环境下特定影响因子(例如数据库、配置文件、项目之间依赖、网络布局等).但研究指出,传统的以代码为中心的研究方法,忽视了复杂环境下特定因子,可能造成测试不完整和发布软件的质量不高的风险.

建议:随着 Web 服务应用等程序的广泛使用,复杂异构环境下特定的持续集成测试集优化,是面临的新的挑战.因此,除了传统的影响因子(包括测试历史、覆盖信息、代码变化和依赖关系等),还应当考虑复杂异构环境下特定的影响因子,提出一些新的研究方法,例如基于多语言环境下持续集成测试优化、非代码制品的文本检索、主题模型方法进行测试用例排序.

(3) 已有研究方法在工业界的验证:在已有研究中,只有 18%的研究在大型工业数据集上进行了验证,其他研究仅在中小型工业数据集或开源数据集上进行了验证,可能在大型集成项目上并不适用.此外,一些研究工作只是在单个数据集上验证了方法的有效性,这些研究的验证应该额外增加真实的场景或标准的实例.

建议:首先,应当鼓励研究者与工业界合作,把已有的研究应用到大型的工业项目中,是当务之急也是挑战.另外,应当鼓励研究者提供公开数据集,为其他相关研究提供比较数据.最后可能缺少一个公共技术框架,用来比较常见的测试用例集优化方法.

5 总 结

DevOps 的出现和广泛应用,使得持续集成越来越受到重视.在有限的资源约束和不降低测试质量的前提下,通过优化测试用例集来减少持续集成时间,已成为学术界和工业界研究的热点.

本文的研究从 4 个电子数据库中检索近十年的文献,共选取 39 篇重要文献,调研持续集成测试用例集优化的研究进展.从研究主题、影响因子、研究方法、研究对象和性能评价 5 个方面提取数据,定量分析回答设定的研究问题.总之,持续集成测试用例集优化的已有研究积累了较多的研究成果,但仍然存在一些问题,目前还少有研究在减少测试成本的同时,保证测试的缺陷检测能力和覆盖度.此外,大型复杂异构环境下的持续集成测试优化、已有研究在工业界的推广仍然是目前工业界和学术界面临的重大挑战.

References:

[1] Hilton M, Nelson N, Tunnell T, Marinov D, Dig D. Trade-Offs in continuous integration: Assurance, security, and flexibility. In: Proc. of the Joint Meeting on Foundations of Software Engineering. 2017. 197–207.

[2] Vasilescu B, Yu Y, Wang H, Devanbu P, Filko V. Quality and productivity outcomes relating to continuous integration in GitHub. In: Proc. of the Joint Meeting on Foundations of Software Engineering. 2015. 805–816.

[3] Hilton M, Tunnell T, Huang K, Marinov D, Dig D. Usage, costs, and benefits of continuous integration in open-source projects. In: Proc. of the IEEE/ACM Int'l Conf. on Automated Software Engineering. 2016. 426–437.

[4] Martensson T, Stahl D, Bosch J. Continuous integration impediments in large-scale industry projects. In: Proc. of the IEEE Int'l Conf. on Software Architecture. IEEE, 2017. 169–178.

[5] Virmani M. Understanding DevOps & bridging the gap from continuous integration to continuous delivery. In: Proc. of the 5th Int'l Conf. on Innovative Computing Technology (INTECH). 2015. 78–82.

[6] Rathod N, Surve A. Test orchestration a framework for continuous integration and continuous deployment. In: Proc. of the Int'l Conf. on Pervasive Computing. 2015. 1–5.

[7] Kitchenham BA, Charters S. Guidelines for performing systematic literature reviews in software engineering. EBSE Technical Report, EBSE-2007-01, ICSE IEEE Computer Society, 2007. 1051–1052.

[8] Badampudi D, Wohlin C, Kai P. Experiences from using snowballing and database searches in systematic literature studies. In: Proc. of the Int'l Conf. on Evaluation and Assessment in Software Engineering. 2015. 1–10.

[9] Kitchenham BA, Dyba T, Jorgensen M. Evidence-Based software engineering. In: Proc. of the Int'l Conf. on Software Engineering. IEEE Computer Society, 2004. 273–281.

[10] Yoo S, Harman M. Regression testing minimization, selection and prioritization: A survey. Software Testing, Verification and Reliability, 2012,22(2):67–120.

[11] Kazmi R, Jawawi DNA, Mohamad R, Ghani I. Effective regression test case selection: A systematic literature review. ACM Computing Surveys, 2017,50(2):29.

[12] Nanda A, Mani S, Sinha S, Harrold MJ, Orso A. Regression testing in the presence of non-code changes. In: Proc. of the 4th IEEE Int'l Conf. on Software Testing, Verification and Validation. 2011. 21–30.

[13] Qiu D, Li B, Ji S, Leung H. Regression testing of Web service: A systematic mapping study. ACM Computing Surveys, 2015,47(2):21.

[14] Stratis P, Rajan A. Test case permutation to improve execution time. In: Proc. of the IEEE/ACM Int'l Conf. on Automated Software Engineering. 2016. 45–50.

[15] Huang CY, Chen CS, Lai CE. Evaluation and analysis of incorporating fuzzy expert system approach into test suite reduction. Information and Software Technology, 2016,79(C):79–105.

[16] Qu X, Cohen MB, Woolf KM. Combinatorial interaction regression testing: A study of test case generation and prioritization. In: Proc. of the IEEE Int'l Conf. on Software Maintenance. 2007. 255–264.

[17] Engström E, Runeson P, Skoglund M. A systematic review on regression test selection techniques. Information and Software Technology, 2010,52(1):14–30.

[18] Catal C, Mishra D. Test case prioritization: A systematic mapping study. Software Quality Journal, 2013,21(3):445–478.

[19] Soetens QD, Demeyer S, Zaidman A, Pérez J. Change-Based test selection: An empirical evaluation. Empirical Software Engineering, 2016,21(5):1990–2032.

[20] Blondeau V, Etien A, Anquetil N, Cresson S, Croisy P, Ducasse S. Test case selection in industry: An analysis of issues related to static approaches. Software Quality Journal, 2016, 1–35.

[21] Spieker H, Gotlieb A, Marijan D, Mossige M. Reinforcement learning for automatic test case prioritization and selection in continuous integration. In: Proc. of the 26th ACM SIGSOFT Int'l Symp. on Software Testing and Analysis. 2017. 12–22.

[22] Busjaeger B, Xie T. Learning for test prioritization: An industrial case study. In: Proc. of the 24th ACM SIGSOFT Int'l Symp. on Foundations of Software Engineering. 2016. 975–980.

[23] Elbaum S, Rothermel G, Penix J. Techniques for improving regression testing in continuous integration development environments. In: Proc. of the 22nd ACM SIGSOFT Int'l Symp. on Foundations of Software Engineering. 2014. 235–245.

[24] Parsai A, Soetens QD, Murgia A, Demeyer S. Considering polymorphism in change-based test suite reduction. In: Proc. of the Int'l Conf. on Agile Software Development. Cham: Springer-Verlag, 2014. 166–181.

[25] Shi A, Thummalapenta S, Lahiri SK, Bjorner N, Czerwonka J. Optimizing test placement for module-level regression testing. In: Proc. of the IEEE/ACM Int'l Conf. on Software Engineering. IEEE Computer Society, 2017. 689–699.

[26] Memon A, Gao Z, Nguyen B, Dhanda S, Nickell E, Siemborski R, Micco J. Taming Google-scale continuous testing. In: Proc. of the 39th Int'l Conf. on Software Engineering. 2017. 233–242.

[27] Marijan D, Liaaen M, Gotlieb A, Sen S, Ieva C. TITAN: Test suite optimization for highly configurable software. In: Proc. of the IEEE Int'l Conf. on Software Testing, Verification and Validation (ICST). 2017. 524–531.

[28] Ekelund ED, Engstrom E. Efficient regression testing based on test history: An industrial evaluation. In: Proc. of the IEEE Int'l Conf. on Software Maintenance and Evolution. 2015. 449–457.

[29] Noor TB, Hemmati H. A similarity-based approach for test case prioritization using historical failure data. In: Proc. of the 26th IEEE Int'l Symp. on Software Reliability Engineering (ISSRE). 2015. 58–68.

[30] Hemmati H, Fang Z, Mantyla MV. Prioritizing manual test cases in traditional and rapid release environments. In: Proc. of the 8th IEEE Int'l Conf. on Software Testing, Verification and Validation (ICST). 2015. 1–10.

[31] Herzig K, Greiler M, Czerwonka J, Murphy B. The art of testing less without sacrificing quality. In: Proc. of the IEEE/ACM Int'l Conf. on Software Engineering. 2015. 483–493.

[32] Bach T, Andrzejak A, Pannemans R. Coverage-Based reduction of test execution time: Lessons from a very large industrial project. In: Proc. of the IEEE Int'l Conf. on Software Testing, Verification and Validation Workshops. IEEE, 2017. 3–12.

[33] Beszédes Á, Gergely T, Schrettner L, Jász J, Langó L, Gyimóthy T. Code coverage-based regression test selection and prioritization in WebKit. In: Proc. of the 28th IEEE Int'l Conf. on Software Maintenance (ICSM). 2012. 46–55.

[34] Strandberg PE, Sundmark D, Afzal W, Ostrand TJ, Weyuker EJ. Experience report: Automated system level regression test prioritization using multiple factors. In: Proc. of the 27th IEEE Int'l Symp. on Software Reliability Engineering (ISSRE). 2016. 12–23.

[35] Engström E, Runeson P, Ljung A. Improving regression testing transparency and efficiency with history-based prioritization-an industrial case study. In: Proc. of the 4th IEEE Int'l Conf. on Software Testing, Verification and Validation (ICST). 2011. 367–376.

[36] Srikanth H, Cashman M, Cohen MB. Test case prioritization of build acceptance tests for an enterprise cloud application: An industrial case study. Journal of Systems and Software, 2016,119:122–135.

[37] Nguyen CD, Perini A, Tonella P, Kessler FB. Automated continuous testing of multiagent systems. In: Proc. of the 5th European Workshop on Multi-Agent Systems (EUMAS). 2007.

[38] Soetens QD, Demeyer S, Zaidman A. Change-Based test selection in the presence of developer tests. In: Proc. of the 17th IEEE European Conf. on Software Maintenance and Reengineering (CSMR). 2013. 101–110.

[39] Vöst S, Wagner S. Trace-Based test selection to support continuous integration in the automotive industry. In: Proc. of the Int'l Workshop on Continuous Software Evolution and Delivery. ACM, 2016. 34–40.

[40] Campos J, Arcuri A, Fraser G, Abreu R. Continuous test generation: Enhancing continuous integration with automated test generation. European Journal of Oral Sciences, 2010,109(4):241–248.

[41] Marijan D. Multi-Perspective regression test prioritization for time-constrained environments. In: Proc. of the IEEE Int'l Conf. on Software Quality, Reliability and Security (QRS). 2015. 157–162.

[42] Gligoric M, Eloussi L, Marinov D. Practical regression test selection with dynamic file dependencies. In: Proc. of the 2015 Int'l Symp. on Software Testing and Analysis. ACM, 2015. 211–222.

[43] Legunsen O, Hariri F, Shi A, Yu YF, Zhang LM, Marinov D. An extensive study of static regression test selection in modern software evolution. In: Proc. of the ACM SIGSOFT Int'l Symp. on Foundations of Software Engineering. 2016. 583–594.

[44] Mirarab S, Akhlaghi S, Tahvildari L. Size-Constrained regression test case selection using multicriteria optimization. IEEE Trans. on Software Engineering, 2012,38(4):936–956.

[45] Vasic M, Parvez Z, Milicevic A, Gligoric M. File-Level vs. module-level regression test selection for .NET. In: Proc. of the Joint Meeting on the Foundations of Software Engineering. ACM, 2017. 848–853.

[46] Jiang B, Zhang Z, Tse TH, Chen TY. How well do test case prioritization techniques support statistical fault localization. In: Proc. of the 33rd Annual IEEE Int'l Computer Software and Applications Conf. 2009,1:99–106.

[47] Do H, Mirarab S, Tahvildari L, Rothermel G. The effects of time constraints on test case prioritization: A series of controlled experiments. IEEE Trans. on Software Engineering, 2010,36(5):593–617.

[48] Knauss E, Staron M, Meding W, Söder O, Nilsson A, Castell M. Supporting continuous integration by code-churn based test selection. In: Proc. of the IEEE Int'l Workshop on Rapid Continuous Software Engineering. 2015. 19–25.

[49] Dösinger S, Mordinyi R, Biffl S. Communicating continuous integration servers for increasing effectiveness of automated testing. In: Proc. of the 27th IEEE/ACM Int'l Conf. on Automated Software Engineering. 2012. 374–377.

[50] Marijan D, Liaaen M. Effect of time window on the performance of continuous regression testing. In: Proc. of the IEEE Int'l Conf. on Software Maintenance and Evolution. 2016. 568–571.

[51] Cho Y, Kim J, Lee E. History-Based test case prioritization for failure information. In: Proc. of the 23rd IEEE Asia-Pacific Software Engineering Conf. (APSEC). 2016. 385–388.

[52] Marijan D, Liaaen M. Test prioritization with optimally balanced configuration coverage. In: Proc. of the IEEE Int'l Symp. on High Assurance Systems Engineering. IEEE, 2017. 100–103.

[53] Yoo S, Nilsson R, Harman M. Faster fault finding at Google using multi objective regression test optimization. In: Proc. of the 8th European Software Engineering Conf. and the ACM SIGSOFT Symp. on the Foundations of Software Engineering (ESEC/FSE 2011). Szeged, 2011.

[54] Marijan D, Gotlieb A, Sen S. Test case prioritization for continuous regression testing: An industrial case study. In: Proc. of the 29th IEEE Int'l Conf. on Software Maintenance (ICSM). 2013. 540–543.

[55] Biswas S, Blanton RD. Reducing test execution cost of integrated, heterogeneous systems using continuous test data. IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems, 2011,30(1):148–158.

[56] Celik A, Vasic M, Milicevic A, Gligoric M. Regression test selection across JVM boundaries. In: Proc. of the 11th Joint Meeting on Foundations of Software Engineering. 2017. 809–820.

[57] Mondal D, Hemmati H, Durocher S. Exploring test suite diversification and code coverage in multi-objective test case selection. In: Proc. of the IEEE Int'l Conf. on Software Testing, Verification and Validation. 2015. 1–10.

李英玲(1984－),女,湖南衡阳人,博士生,主要研究领域为持续集成测试,生产力分析.

王青(1964－),女,博士,研究员,博士生导师,CCF 高级会员,主要研究领域为软件过程方法与技术,软件知识工程,经验软件工程.