|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 论文题目 | 代码坏味研究综述 | | | |
| Paper URL | [http://www.jos.org.cn/1000-9825/6431.htm](http://www.jos.org.cn/1000-9825/6431.htm" \t "http://html.rhhz.net/rjxb/html/_blank) | | | |
| Project URL |  | | | |
| 综述/背景介绍 | 发展状况 | 原因 | 意义 | 关键词（速记词汇、信息索引词汇） |
| 发展状况：  代码坏味是软件工程领域的一个研究热点,并且 相关研究方向众多、时间跨度大、研究成果丰富.究人员多次尝试对相关的研究进行文献综述，以梳理相关研究成果和研究思路,更好地了解和掌握研究现状。学术界对于代码坏味的研究持续发展，进一步推进了深度学习在其他坏味检测上的应用.此外学术界提出了众多新的代码坏味拓展了坏味目录，并且涌现出很多新的重构解决方案。  原因：  现有的文献综述研究论文发表时间较早,没能覆盖最近几年日新月异的研究进展.此外,这些综述论文各有侧重点,但都未能解决关键代码坏味遴选、工业 界应用状况分析、特点方向发展趋势等关键问题.为此,本文针对所有代码坏味相关的研究论文进行全面的分析和梳理。  意义：  本文综述了现有代码坏味相关的研究,围绕代码坏味的持续演化,关键代码坏味及研究热点,和代码坏味在工业界的研究现状3个方面展开了研究,完成了对代码坏味发展趋势以及研究热点的分析,提出一种代码坏味关键程度的量化方案,以分析其应该受关注和付出努力的优先级.最后从工业界实际情况出发,对实际开发过程中用于代码坏味检测的静态代码检查工具进行调查,分析了学术界和工业界对于代码坏味关注的一致性,并分析其可能存在的原因,为研究人员和致力于改善软件质量的从业人员给出研究机会指明了研究机会. | | | 代码坏味;软件重构;软件质量;度量;缺陷 |
| 假设 |  | | |  |
| 方法描述(含图) | 本次综述按照Kitchenham的系统文献综述的标准步骤进行.具体综述方法包括搜索范围与关键词、纳入 排除规则、人工审查标准等方面.具体论文收集过程如下：首先选择搜索数据库，本次综述选择ACM digital library、IEEE Xplore、Springer link、Google Scholar、Web of Science、Science Direct、中国知网、万方数据库、 中国科技论文在线、中图链接等文献数据库,对国内外针对代码坏味研究的相关文献进行检索。其次进行论文筛选，对于得到的搜索结果需由人工分析论文标题、摘要以及关键字等内容,并应用纳入排除规则进行论文筛选, 得到需要进一步研究的论文数据集.由于初始数据集过大,在具体实施时只考虑了综述性论文的参考文献,并且只进行一次滚雪球操作,最终得到了1129篇论文作为下一阶段筛选的初始数据集.最后进行人工审查，阅读论文并记录相关信息，排除初始数据集中可能包含仅涉及代码坏味但并未深入研究的论文.结果得到了用于本次研究的339篇论文及其数据。  重点关注以下4个研究问题.  RQ1:人们是否持续提出新的代码坏味?新的代码坏味主要关注哪些领域/问题?  RQ2:代码坏味相关研究的目的可以包含哪些类型?哪些类型的研究是目前的主流和热点?  RQ3:学术界重点关注哪些代码坏味?  RQ4:工业界重点关注哪些代码坏味?工业界的关注点是否与学术界一致? | | |  |
| 实验设计 |  | | |  |
| 数据处理 | 输入 | 筛除特例 | 处理方式 | 关键词（速记词汇、信息索引词汇） |
|  |  |  |  |
| 结论 | 代码坏味仍然是软件工程领域的研究热点.本文对1990年至2020年6月间发表的代码坏味相关的339篇论文进行了系统地分析和归类。回答了重点关注的4个研究问题。  研究问题RQ1探索了新定义代码坏味的发展趋势.定义新的代码坏味呈现出先下降而后一直增长的趋势,将代码坏味的概念拓展到其他领域,并定义新的代码坏味成为现阶段的研究热点.  研究问题RQ2展示了相关研究的研究目的.研究通过主题分析得到了彼此正交的10个研究目的类型,并对每种类型的研究进行了详细的说明.  研究问题RQ3解决了代码坏味关键程度的量化.利用以往研究中有关代码坏味广泛性或严重性的结论性描述,作为已被验证的补充信息来反映代码坏味的关键程度.  研究问题RQ4分析比较了学术界与工业界对代码坏味关注点的差异. | | | |
| 局限性分析 |  | | | |

1.论文总结：

（论文名中文）：代码坏味研究综述

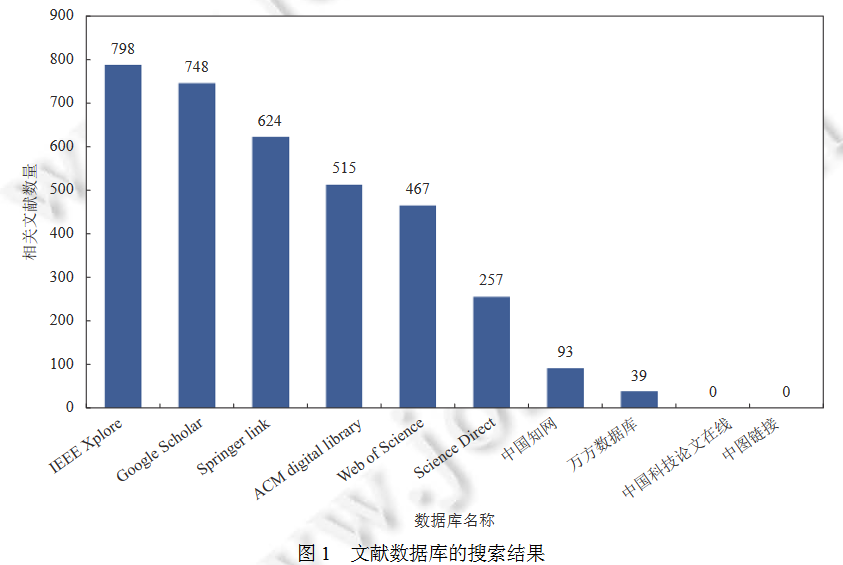
（论文名英文）：Survey on Code Smells

（论文URL）：[http://www.jos.org.cn/1000-9825/6431.htm](http://www.jos.org.cn/1000-9825/6431.htm" \t "http://html.rhhz.net/rjxb/html/_blank)

（总结）：代码坏味(code smells)是于1999年提出的一个概念,用于表示低质量的急需重构的软件代码片段.由于受到交付日期的压力或者开发人员的疏漏等因素的影响,软件开发和演化过程中有可能引入代码坏味,从而降低了代码的可理解性和可维护性.本文首先利用获得的数据集研究了学术界对代码坏味的研究状态,揭示了近年来软件工程领域的研究人员对代码坏味日益增长的研究兴趣,以及不断定义新的代码坏味的研究趋势。研究问题RQ1研究过程中发现,目前仍然缺少对代码坏味进行分类的标准.在将来的研究中,应该考虑定义一种完备、一致且准确的代码坏味分类标准,以规范代码坏味的定义和拓展,进一步提高代码坏味被关注的程度.研究问题RQ2研究过程中发现，①对于代码坏味重构的研究热度虽然有所下降,但是目前对于较高层次代码坏味,以及一些非面向对象代码中的代码坏味的自动重构解决方案相对较少,缺少更加深入的研究.因此建议研究人员针对不同的层次或者成熟的编程语言,研究和开发相应的代码坏味重构方案;②其次对于其他研究目的的论文,比如出现原因、优先级等,代表了对代码坏味不同特性的分析,有助于推进该领域的研究。研究问题RQ4中研究发现,这些经常用于实际开发中的工具较多的关注于能够直接或间接影响软件质量指标的代码坏味,并且一些新颖的代码坏味检测技术并未应用于工具检测过程中。而是更多采用模式匹配等方法,以达到支持用户自定义规则的要求因此在研究代码坏味的定义以及检测时,明确代码坏味带来的影响,并且提供选择软件质量指标进行检测的理由,将有利于提高开发人员对代码坏味接受程度。

（截图）：与方法相关的重要截图

在不同的文献数据库上应用搜索规则检索后,得到的数据库中相关的文献数量如图1 所示.



完整的论文数据收集执行流程如图2所示.

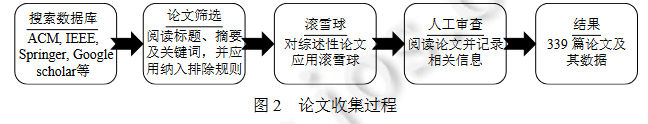


图3展示了从首次出现代码坏味的相关研究至2020年,每年发表的相关研究的数量,并且依照研究发表刊物所属的CCF等级,对每年发表的研究进行区分.由于本研究论文数据只统计到2020年6月,因此2020年对应的论文数仅代表部分数据。

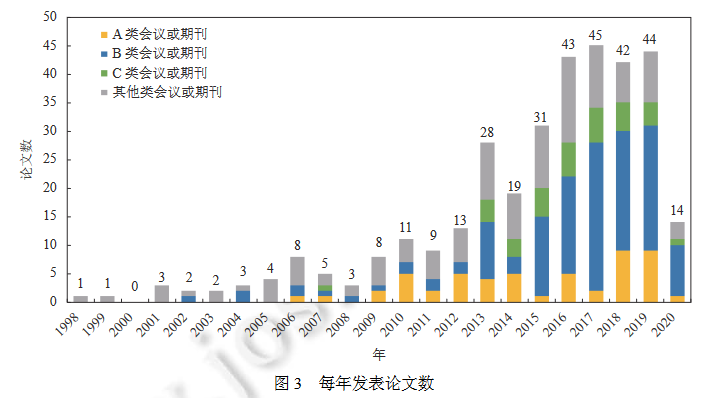


图4表示每年发表的论文中新定义代码坏味的数量。结果表明,自1998年至2004年在代码坏味被提出和讨论的初始时期,定义新代码坏味的热情逐渐下降,直至2004年发表的3篇论文中只新定义了1种代码坏味.而在之后的一段时间里,对代码坏味的研究主要集中在已 有的代码坏味上,新定义代码坏味的数量一直保持在较低的水平.而对该阶段的论文进一步分析发现,对于代码坏味的研究主要专注于面向对象代码,未将代码坏味的概念引申到其他领域。随着研究的深入,直至2013年研究人员对代码坏味进行拓展并完善代码坏味列表的兴趣骤 然上升,虽然受每年发表论文数的影响,每年研究人员定义新的代码坏味的数量有所波动,例如2013–2015年新定义代码坏味数有所下降.但是其总体呈上升趋势。

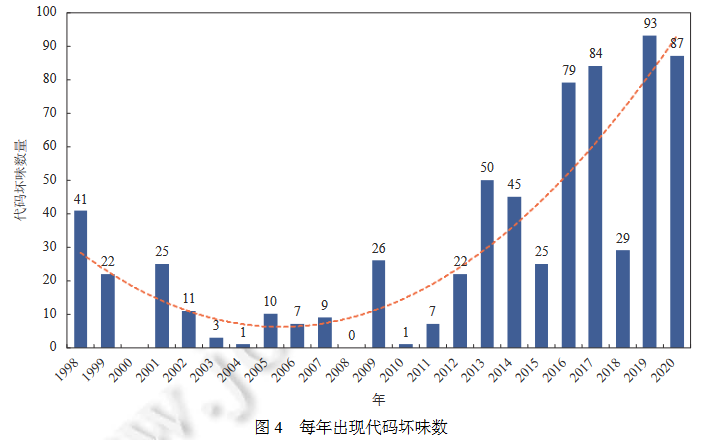
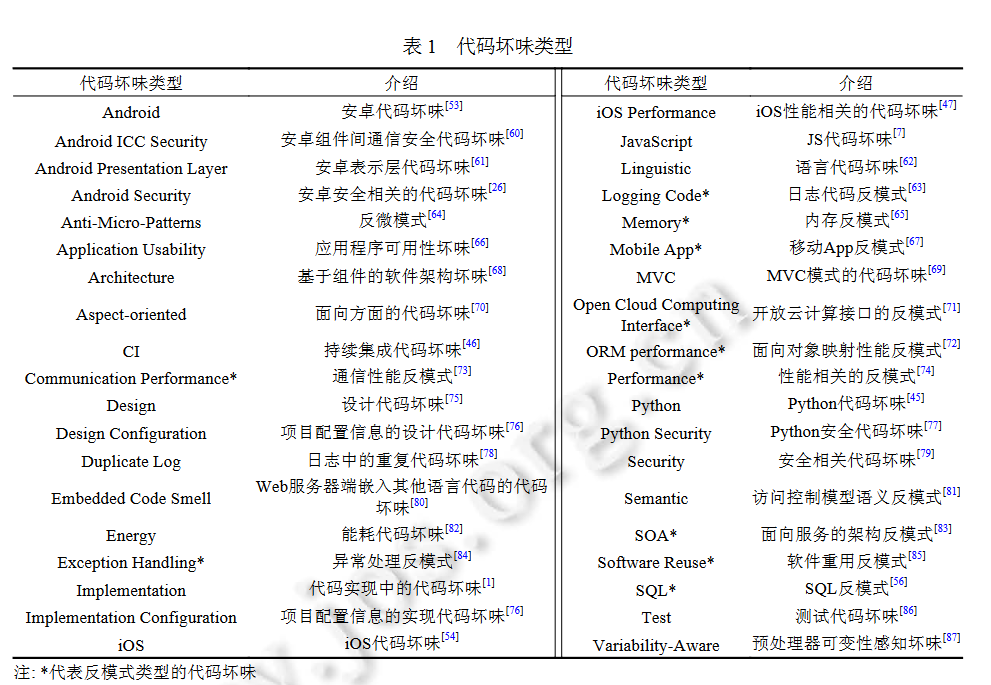
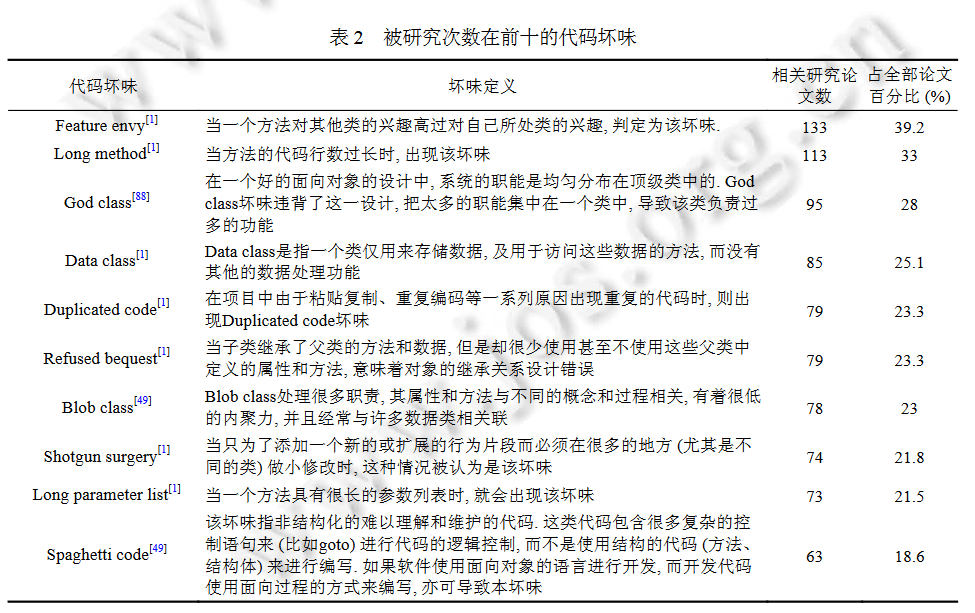


表1是本研究得到的38个不同的代码坏味类型的目录和对应的解释.





Feature Envy (依恋情节/功能嫉妒)某个函数为了计算某个值，从另一个对象那里调用几乎半打的取值函数。通俗点讲，就是一个函数使用了大量其他类的成员，有人称之为红杏出墙的函数。

Long Method (长函数)长函数是指一个函数方法几百行甚至上千行，可读性大大降低，不便于理解。

God class（上帝类）

Data Class （纯数据类）类很简单，仅有公共成员变量，或简单操作函数。

Duplicated Code （重复代码）重复代码就是不同地点，有着相同的程序结构。

Refused Bequest （拒绝继承）子类应该继承父类的数据和函数。子类继承得到所有函数和数据，却只使用了几个，那就是继承体系设计错误/（继承过多）父类里面方法很多，子类只用有限几个。

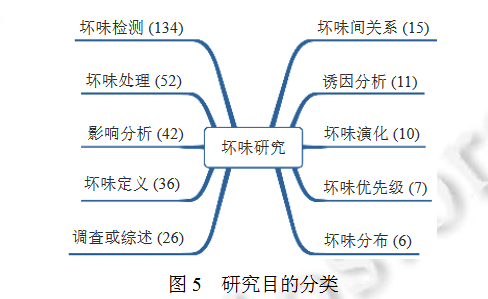
Blob class（复杂类）

Shotgun Surgery（散弹式修改）当你实现某个小功能时，你需要在很多不同的类做出小修改。

Long Parameter List (过长参数列)方法参数数量过多的话，可读性很差。如果有多个重载方法，参数很多的话，有时候你都不知道调哪个呢。并且，如果参数很多，做新老接口兼容处理也比较麻烦。

Spaghetti code(面条式代码) 是软件工程中反模式的一种，是指一个代码的控制结构复杂、混乱而难以理解，尤其是用了很多GOTO、例外、线程、或其他无组织的分歧架构。其命名的原因是因为程序的流向就像一盘面一样的扭曲纠结。

使用主题分析提取并确定最终的研究目的类型，得到最终的10种研究目的类型如图5所示



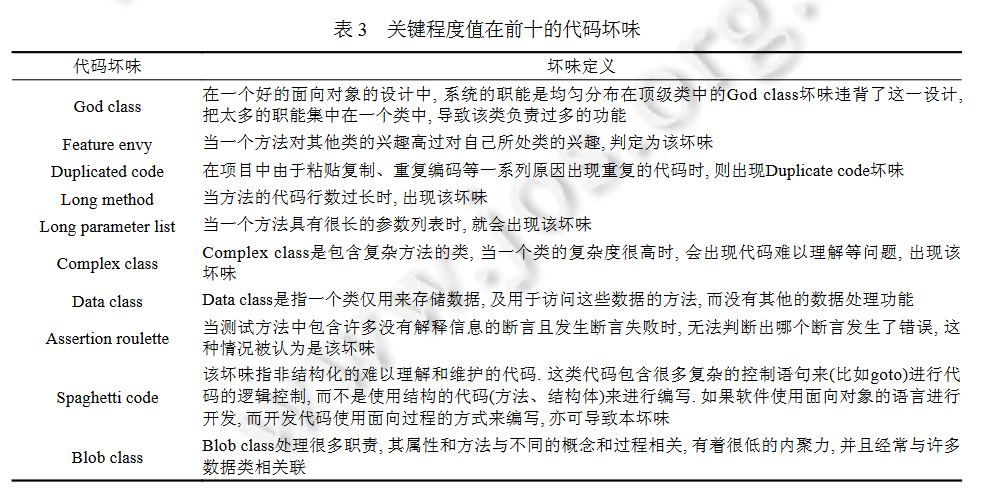
使用代码坏味的关键程度值(Key Score of Code Smell)进行量化判断，代码坏味如果在相关研究中中出现次数较多, 并且在结论性描述中也出现次数较多, 则该认为该代码坏味是关键的。

关键程度值的计算通过对坏味在文献中的出现频数NO(number of occurrences)和加权后的代码坏味结论性描述频数NC(number of conclusive description )求和得到.没有对应结论性描述的坏味,相应的NC值设为0.

其中, NCSum代表所有坏味在结论性描述中出现的频数之和, NOSum代表所有坏味在论文中出现的频数之和. 利用两数的商作为结论性描述频数的权重, 以加强这些结论性额外信息的影响.



表3展示了根据关键程度值进行排序后的前十位代码坏味. 结果发现按照关键程度排序后的前十个代码坏味, 与单纯按照频数排序后的结果有很大差别. 而由于关键程度考虑了额外的代码坏味广泛性和严重性信息, 因此能更好地评估代码坏味的关键程度.



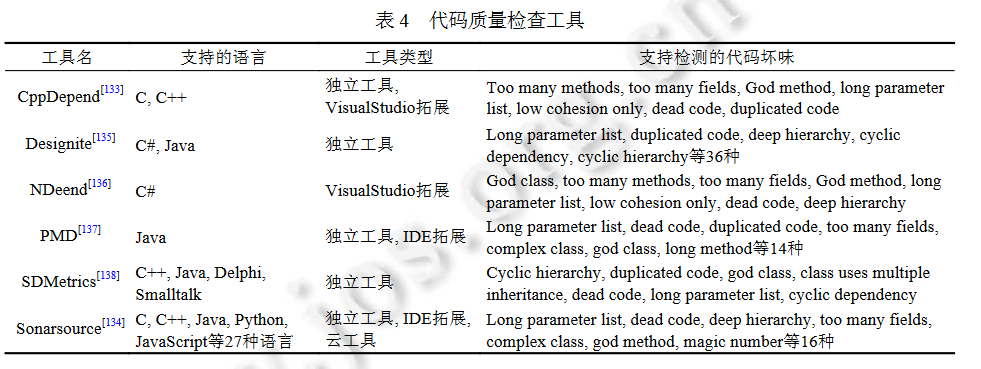
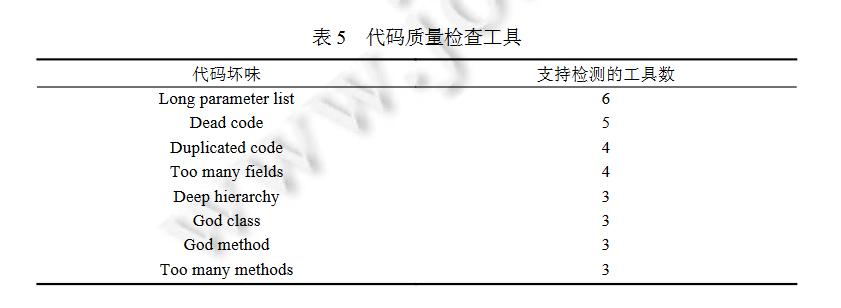


图5显示了受到3种及以上工具支持的代码坏味,可以看到long parameter list代码坏味能够被所有6种工具检测,并且同样是关键程度值在前10的代码坏味.除此之外, God class、duplicated code同样也是受到工具支持较多的且关键程度较高的代码坏味.从总体上看,关键程度值在前10代码坏味在以上6种工具的检测规则中只有1种未受到支持,即blob class.除此之外,代码质量检查工具还关注于其他5种关键程度值并不高的代码坏味,如dead code、God method等.



总而言之,代码坏味在项目实际开发中会作为质量问题被开发人员关注并检测,对于关键程度值较高的代码坏味,学术界和工业界的关注点是一致的.开发过程中的代码质量检查工具对其提供了很好的支持.关键程度值最高的10个代码坏味,有9个得到了工业界的重点关注和认可,代码坏味blob class未受到支持的可能原因是其表现形式与God class过于相同, 因此只提供了God class的检测规则.