I.INTRODUCTION

在当今的软件开发过程中，由于用户需求存在很大的变化波动，软件产品的开发过程是反复添加、删除、修改的过程。在过去的20年内，维护软件的花费占整个软件系统的花费，已经从50%增长到了90%。为了帮助减少维护费用，研究者提出了一些方法尝试降低软件的复杂性。在这里，我们尝试分析反模式对于软件的易改性（change-proneness）的影响。

在软件不断改进和完善的过程中，代码常常会变得越来越坏，影响系统的质量和可维护性。这些不良代码和低劣设计等被称为反模式或“代码坏味道”。反模式是对于设计和应用上的“坏”的解决方案。Coplien将反模式描述成“看上去是个好主意，实际应用起来却造成了很糟糕的不良后果”研究发现，很多软件系统中超过45%的类都存在至少一个反模式，对系统的质量造成了严重的影响。因此如何检测反模式，并且从反模式入手来提高软件质量已经成为学术界和工业界非常关心的内容。

在这篇文章中，我们分析反模式可以被用来指出和预见Java classes的改变。研究的目的是调查分析哪些反模式更易导致Java classes的修改。在分析和收集数据的过程中，我很会考虑这些一些代码修改的意义同时尝试过滤掉不重要的修改类型（如对注释的修改）。

为进行我们的研究分析，我们将通过回答以下两个研究问题来完成分析

* **RQ1：**被某一个反模式所影响的Java classes是否比没有被该反模式影响的Java classes更易被修改
* **RQ2：**被某两个反模式所影响的Java classes是否比没有被该反模式组合影响的Java classes更易被修改

这篇文章的大致内容如下，第二部分我们将描述获取分析所需要的数据的方法，第三部分呈现我们分析的方法，过程以及最后的结果；第四部分讨论我们的研究所用到的理论是否有些不可靠地因素，最后一部分讲述我们的结论和一些相关工作

II.DATA COLLECTION

在这一部分，我们将描述我们研究所需要的数据以及整合获得数据的方法。这里先简要的说明所需获得的数据和方法。

1. 第一步为获取开源软件各个release之间的修改信息，通过基于linux中diff命令的脚本获得各个release版本之间的Patch文件，可以得到各个release之间的差异和修改信息。
2. 第二步为对修改信息进行过滤和处理，将例如修改注释一类的非重要修改进行简单的过滤，同时获取各个Java classes的“有效”修改行数，哪些Java classes收到了修改，以及版本之间总共的“有效”修改行数。（记录在Change\_log文件中）
3. 第三步为探测在Java classes中出现的反模式。我们通过DÉCOR（Defect dEtection for CORrection）。DÉCOR通过一系列的规则集描述反模式，并从Java classes中探测出反模式，我们通过在我们分析的开源软件上运行DÉCOR探测反模式。在探测出的反模式中，我们选取了AntiSingleton，Blob，ClassDataShouldBePrivate，ComplexClass，LazyClass，LongParameterList，MessageChain，RefusedParentBequest，SwissArmyKnife进行分析。先对Antisingleton，Blob，ComplexClass，LazyClass，SwissArmyKnife进行分析（开源软件中先对Xalan进行分析）

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| System | #Antisingleton | Blob | ComplexClass | LazyClass | SwissArmyKnife |
| Xalan\_2.6 | 29 | 49 | 111 | 77 | 3 |
| Xalan\_2.7 | 2 | 54 | 116 | 110 | 2 |
| Xalan\_2.71 | 1 | 57 | 120 | 118 | 2 |

III.EMPIRICAL STUDY

实际研究的目的是为了研究反模式与Java类的change-proneness之间的联系。我们对两个广泛使用的Java开源系统作为实例进行研究。

1. Investigation of RQ1

RQ1的目标是分析JAVA类被一个反模式影响的易改性，我们将与没有被改反模式影响的JAVA类的易改性作比较。为了保证RQ1的测试我们提出以下的假设：

* H1：在两个发布的版本之间，被某个反模式影响的类们中，与没有被该反模式影响的类们中，被修改过的类所占的比例没有不同。

1. *分析方法*：我们使用odds ratio（ORs）去测量在两个releases(k,k+1)之间，如果Java class在release k中被至少一个反模式影响时，该Java class会受到修改的可能性。OR被定义为，比率p表示某个事件发生在一个组（实验组）中的概率，比率q表示该事件发生在另一个组（控制组）的概率。在我们的 研究中，实验组为至少被一个反模式所影响的class集合，控制组则为没有被该反模式影响的class集合。ORs大于1表示修改更易发生在被该反模式影响的class中，小于1反之，等于1则表示发生修改的可能性相同。

（分析方法中计划添加Fisher’s exact test，通过计算Fisher p-value是否小于0.01判断change-proneness与Java classes中antipatterns的关系

1. *结果：*在对SwissArmyKnife的分析中，xalan-j\_2\_6\_0中共有13个class受到该反模式的影响，其中5个class在下一个release版本中没有受到修改，共有881个class，522个作为有效修改过的class。被SwissArmyKnife影响的class更易受到修改

AntiSingleton分析类似，4个class无有效修改，共29个class受其影响

(Blob 和ComplexClass由于数量较多，正在写C++程序辅助分析)

|  |  |  |  |
| --- | --- | --- | --- |
| Antipattern\_Xalan | OR |  |  |
| AntiSingleton | 4.4623 > 1 |  |  |
| Blob |  |  |  |
| ComplexClass |  |  |  |
| SwissArmyKnife | * 1. 1 |  |  |

IV. THREATS TO VALIDITY

V. CONCLUSION AND RELATED WORK