浙江大学

# ZJUI 学院 2022 年暑期科研项目总结材料

中文论文题目 ： 从软件工程角度分析开源软件项目的架构反模式

英文论文题目： **Architectural Antipatterns Research Based on Software Engineering**


成员姓名：王杰 （3200112404）

朱杭刚 （3200110457）

蔡奕涛 （3200110985）

指导教师：万志远

指导教师所在学院：软件工程学院（玉泉）


暑期研究起止日期：2022 年 6 月 30 日-2022 年 8 月 14 日

## 摘要/ Abstract

Nowadays, open-source ecology brings prosperity to every dimension of software development. However, unlike projects owned and managed by commercial corporations, open-source projects are instinctively decentralized. Architectural anti-patterns are rooted deeply, which reduces development efficiency, increases maintenance costs, and causes potential architectural debt that limits those projects. Therefore, the tools to detect smell and help maintainence is necessary. Our team analyzed five tools' performance, producing well-defined architecture smell lists, which can help developers quickly set up the tool they need.



## 关键词/ Keyword

*Architecture smell, architecture anitpattern, maintainence, smell detection, open-source*


# 1介绍/Introduction

随着时代发展，人类小步快跑进入了互联网时代，大大小小的代码项目如雨后春笋般兴起。而开源生态便是其中最璀璨可贵的一部分，开发者众人拾柴火焰高，以集市的形式做出了远超大教堂的成果。代码开源是软件研发和计算机产业的良好生态的优秀策略，良好的开源协议能保护 开发者相关权益的同时促进计算机领域的交流和发展。对开源生态的研究，能够把握新时代下计算机领域研究方向和行业热点。

然而，与传统的大公司的开发形势不同，因为开源项目天然的分散性与自由性，实践中很多架构并未经过科学充分的需求工程分析，存在大量的软件架构性缺陷（architectural smell）和反模式（antipattern）,这不但导致开放出的代码效率与稳定性下降了，同时项目存在隐性的架构性隐患，这导致了软件项目的能力和未来都受到了限制。甚至有很多项目因为最开始的架构过于简单粗暴，形成规模后难以继续改进，使项目逐渐失去生命

因此，能够分析架构反模式的工具如Arcan, Sonarqube,Designate应运而生，通过多种技术路径，可以不同程度地辅助开源程序员进行架构设计和后期重构。我们的研究项目就是通过软件工程的学科视角，研究最有效果的工具及使用方法，以便工程人员参照快速上手这些工具。

# 2 背景/Background

## 2.1反模式(anti-pattern)是什么？

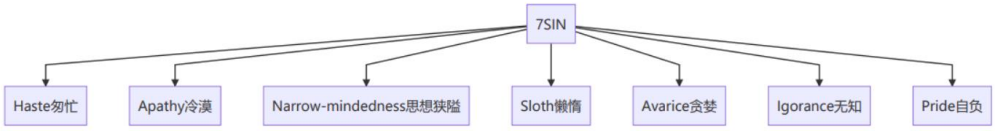
反模式是一种文字记录形式,描述了对某个问题必然产生消极后果的常见解决方案。由于管理人员或者开发人员不知道更好的解决方法,缺乏解决特定类型问题所需的知识或经验,或者是在不适当的环境中应用了一个完美的好模式,这些都会造成反模式。在采用适当文字记录下来的时候,反模式描述了一个一般的形式、其主要原因、典型症状、后果,以及一个如何把该反模式改变成更健康状态的重构方案。

反模式是把一般情况映射到一类特定解决方案的有效方法。反模式的一般形式为它所针对的那类问题提供了一个易于辨识的模板。此外,它还清楚地说明了与该问题相关联的症状以及导致这一问题的典型内在原因。这些模板元素完整地说明了特定反模式存在的情况。这个一般形式可以减少使用设计模式时最常见的问题:把特定设计模式应用于不正确的环境。

反模式为识别软件行业反复出现的问题提供了实际经验,并为大多数常见的困境提供了详细的补救措施。反模式突出了软件行业所面临的最常见问题,并提供了一些工具使你能够识别这些问题并确定其内在原因。此外,反模式为逆转这些内在原因的影响、实现有生产率的解决方案提供了一个详尽的计划。反模式有效地说明了可以在不同层次上采取的措施,以便改善应用的开发。

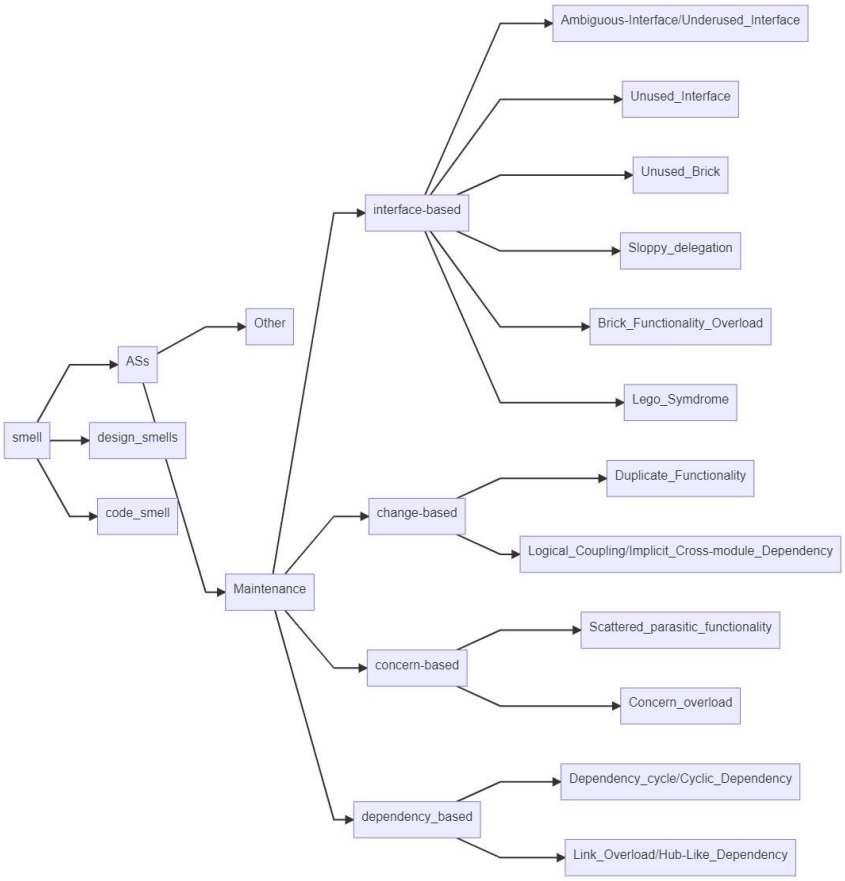
## 2.2 反模式的根源?

恰如《反模式：危机中软件、架构和项目的重构》所说， 根源就是在软件开发中会导致项目失败、成本超支、进度落后和无法满足业务需要等问题的错误[Mowbray 1997]。而这是非常普遍的:所有软件项目中有1/3被取消;每6个软件项目中有5个被认为是不成功的[Johnson 1995]。不幸的是,面向对象技术并没有改变这一总体趋向。实际上,每次新技术潮流(例如客户机/服务器技术)都会趋向于增加软件风险,增加这些根源导致失败的可能性。项目失败的根源被归结为“七宗罪”。这个类比成功地指出了那些低效的实践方法[Bates 1996],它得到了广泛的接受：




图一：造成软件开发反模式的根源性因素

具体来讲，他们都是【架构师-开放人员-管理人员】三角中各个环节会遇到的问题，而不同的疏忽便会导致最终软件架构灾难性的后果。而经过数十年的软件工程研究，人们将软件反模式细分为smell，其中architecure smell 的类型我们总结为以下思维导图：




图二：系统性缺陷的思维导图

实际上，这些架构性缺陷广泛存在于大中小项目中，而且在开源项目尤为突出，因此，如何解析并且利用工具去改进现有项目架构就是一个重要而有意义的问题。

## 2.3 Docker：轻量级的高效软件配置平台

在研究过程中，我们选择以跨平台，高性能，自动化，强稳定的Docker 作为配置软件的首要选项。因为Docker可以同时支持多个不同系统，编程语言的多个版本同时并行使用，命令行的系统也更

贴近Linux的使用体验，大多数的分析软件都可以一行命令下载配置。这为我们同时测试多个不同版本，不同操作系统的软件项目及对应工具提供了方便。

# 3 成果与分析/Result & Analysis

对于架构反模式的定义以及分类，不同学者有不同的定义，总的来说，架构反模式或者架构异味是在软件开发中因为架构以及设计等各个层面产生的问题，并且影响项目的整体质量和可维护性。反模式可以分为架构层面，设计层面以及代码层面，对架构层面的反模式主要在模块性和分层性两方面体现出来。在这次研究中，我们主要关注与架构反模式的分析,并且利用当前已有的工具在一些开源项目上实践，分析不同工具的检测效果。

## 3.1 架构反模式分析

尽管反模式这个术语经常被使用，但不同的人又不同的标准，我们根据一些检测工具的分析和论文的分析，主要将架构反模式分为一下几类,并且主要研究了其中几类

### 3.1.1 Cyclic Dependency

环形依赖关系，又被称为Package Cycle，通常是最常见的架构反模式. 在项目发展过程中，为了方便开发，不同包之间往往需要又一定的依赖关系，但实际开发过程中时常形成复杂的依赖关系。直接依赖关系，即两个包直接互相依赖，往往比较容易发现，间接依赖关系，也即两个包虽然没有之间的依赖关系，但它们所依赖的包以及依赖的包的所依赖的包中等等，在不自觉中就形成了环形依赖，这类依赖关系往往不易可见，并且容易形成十分复杂的依赖关系，开发者并不容易能够查找出相关异味。但随着项目的发展，复杂的依赖关系并不易于项目的维护，对修补所需要的技术债也会不断增加。

### 3.1.2 Unstable Dependency

在面对对象开发的过程中，不同的类以及包等有依赖于继承关系，不同的文件也有不同的重要程度于修改频率，当一个子系统依赖于一个更加不稳定的子系统时，会导致项目整体更加不稳定，并且极大增强了项目的变化倾向，并不利于项目的维护。此项反模式违反了稳定依赖原则。

### 3.1.3 Ambiguous Interface

源于过度抽象的设计接口，在添加旨在适应潜在的未来需求的方法上，容易造成不明确的接口。在开发过程中，由于大量的方法抽象或者需要扩展的类抽象需要实现。此外，由于“一般目的”方法用于定义抽象，使得一些设计接口只有少数使用方法，并不利于项目维护。

### 3.1.4 God Component

良好的项目应做到每个成分实现独立的功能，在模块性上有明确的分工，以此实现不同功能实现的独立化。当一个成分包含过多的子部分或者过多的代码行数时，就会容易造成God Component. 对于God Component, 当需要实现一个独立功能或者需要新加功能时，并不能够容易分解出功能，因此对项目的维护会造成额外的成本。

### 3.1.5 Scattered Functionality

在软件设计时，一般从高层次的功能抽象开始，再到底层的技术细节实现，和God Component相反，当实现一个功能时依赖于过多的分散子功能，导致功能混杂时，同样会增加项目维修成本。再设计一次改动时，往往会涉及其他方面的改动，从而形成一条不健康的改动链

### 3.1.6 Dense Structure

又被称为Hub-Like Dependency。每一个类都有一定的扇入（输入以及使用）和扇出（输出以及被使用），当一个类涉及过多的扇入和扇出时，说明此类与其他抽象类会有极大的交互，也另一层次

上说明了此高扇入扇出类设计不合理，形成了Dense Structure，同样会对架构上增加技术债以及可能的安全漏洞

## 3.2 数据集的搭建

为了能够充分测验不同工具的检测效果，我们收集了20个Java项目以及20个C/C++项目作为工具检测的数据集，对每个项目，收集其发行以来最初版本至今最新版本。对于Java项目，我们从Apache Software Foundation Distribution选择20个，保证该项目在Github上显示Java语言至少80%，且有带有完善的版本记录，仍在更新维护的项目，包括了各大小项目，代码行数在5万行至50万行，涉及大数据，网络服务，网络框架等多种领域。对于C/C++项目，我们收集了OpenEuler开源操作系统的源码，以及各种开源嵌入式系统的源码。

## 3.3 检测工具实践

针对软件开发过程中出现的反模式，有许多不同的检测工具来帮助开发人员解决此类问题。在此次项目中我们主要关注与Designite，Sonargraph，Sonarqube，Arcan，DV8等检测工具的分析，下面列举了各个检测工具的简略介绍

### 3.3.1 Designite

Designite 是一个免费开源的代码分析质量工具，用于检测architecuture smell，design smell，implementation smell 以及其他各种度量。可以检测C#和Java两种语言。 DesigniteJava很大的一个优势是易安装易使用,无需复杂的环境搭建，并且能够给出详细的分析结果，同时DesigniteJava可检测数据规模能够有几十万行，根据电脑配置的不同和项目的不同能检测的代 码大小也不同。 虽然Designite能够给出较为详细的分析报告，并且有每个异味分析的缘由，但并没有很好的可视化，且并没有很好地方法能够定位问题所在之处，实际上对开发者维护项目帮助并不大，更多的是让开发者了解当前项目的整体质量以及重构必要等等

### 3.3.2 Sonarqube

Sonarqube是一个检测项目漏洞以及项目反模式一个全面的分析工具，支持几乎所有常见编程语言。能够明确分析出问题所在之处，可能存在的安全漏洞等等，且支持多版本检测纵向分析，技术债估计，给出项目总体质量评价等等。Sonarqube检测依赖于各种设计规则，用户可以自定义选择应用规则，更加具有针对性。但Sonarqube免费版本与商业版本功能差别较大，免费版本功能有限，同时只支持代码层次的异味检测，并不支持架构异味。Sonarqube更适合帮助开发者找出个别技术细节上的问题，帮助开发人员开发代码时解决一些问题，但不适合架构分析。

### 3.3.3 Sonargraph

Sonargraph是更注重与可视化依赖关系的一个工具。Sonargraph将一个类或者一个包抽象为一个节点，依赖关系抽象为边，从而给出更为直观的依赖关系，不仅帮助开发者梳理大型项目依赖关系，而且也更易解决环形依赖关系的问题。但也有明确的缺点，例如支持个别参数的度量以及Cyclic Dependency的检测，在检测架构反模式上十分有限

### 3.3.4 Arcan

Arcade是类似于Designite的分析工具，同样给出表格数据，且支持额外的可视化功能，支持Java，C/C++语言的检测，也支持多版本纵向分析。支持Cyclic Dependency，Unstable Dependency，Dense Structure，God Component等反模式的分析。和其他工具相比，具有更加完备的功能特点，但实际检测效果并没有其他工具准确。

### 3.3.5 DV8

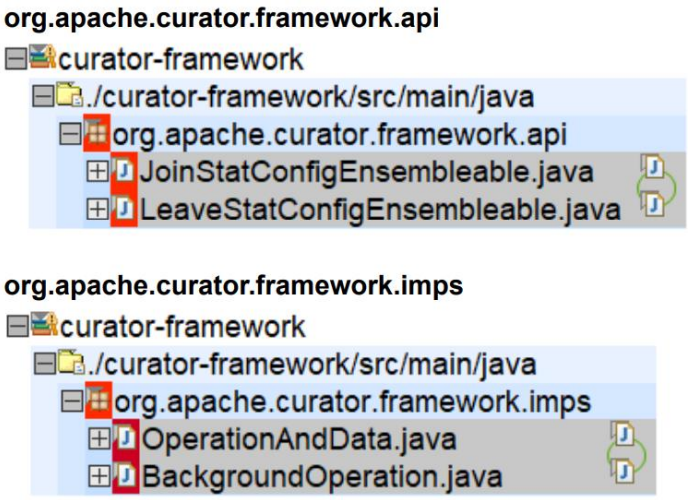
DV8 支持 Cyclic Dependency，Unstabe Dependency 等不同反模式的检测，并且将依赖关系转化为设计结构矩阵(Design Structure Matrix),以此直观显示不同包的依赖关系以及依赖种类等等。

除此之外，DV8还提供了项目分析报告，提供解耦参数等等。DV8同样能够直观表示出异味的具体体现，但是相比之下基本度量较为欠缺。

## 3.4 检测结果与原理分析

虽然每个tool都能够检测出架构反模式，但实际上不同工具对同名称反模式的定义并不一定相同，对同一个反模式检测的原理的和方法也不一定相同，对一个反模式的计数标准也不一定相同。并没有完全的统一标准，因此工具检测的准确性并没有十足的保证,无法保证哪一个工具最真实反映实际情况。

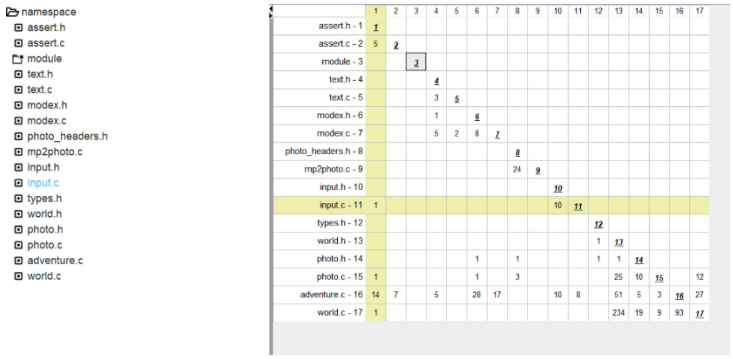
以对分析apache-curator-2.3版本项目为例，对于Cyclic Dependency Designite给出org.apache.curator.framework.api，org.apache.curator.framework，org.apache.curator.framework.imps，org.apache.curator.framework.state， 存在自我依赖。但Sonarqube给出的结果如下：




图三：Sonarqube生成的依赖关系图

同时sonargraph无关于org.apache.curator.framework,org.apache.curator.framework.state但sonargraph有关于org.apache.curator.framework.api.transaction的依赖关系，而designite 并没有此smell。

实际上，这些工具的易用性让他们可以立刻实践于我们的作业当中：以下图片是我们秋学期的课程ECE391:操作系统 中的MP2项目的分析结果：可以看到DV8用关系矩阵很清晰的展示了C语言各种包的逻辑层次关系，很好的帮助我们进一步理解lumetta教授在编写架构时的智慧及我们编写代码时要注意的架构细节。




图四：ECE391:MP2,adventure game

# 4 讨论与困难/Discussion and Difficulty

## 4.1 项目推进遇到的问题

1. 团队自身能力

本次暑研涉及的领域为软件工程，虽然小组三人都是电子与计算机工程专业，但是在软件开发和大型项目中并无实战经验，也并未选修软件工程的相关课程。项目初期的学习成本较高，导致项目初期进度缓慢。

2. 反模式划分不明确

尽管软件工程是一个基本完善的学科，但对于反模式的具体划分，学界并无统一标准。这导致相关论文和资料中对反模式的分类标准相差甚远，相应的代码质量分析工具所采用的检测方式和生成结果也并不一致。对于一些项目，不同的代码分析工具给出的评估结果可能截然相反，但对于其结果信度也因为标准不同难以比较和确定。

3. 分析工具本身可靠性有限

对于市场上已有的代码质量分析工具，可分为商业软件和开源项目。优秀的代码分析工具往往是商业软件，目标客户多为大型软件公司，因此软件使用价格较高。而开源的代码质量分析工具，则局限性较大，一般只能分析Java代码，且无法分析项目版本记录，导致结果信效度不高。

## 4.2 Future Work

在本次研究项目中，我们更多的注重的是实践，分析不同工具对架构反模式的检测效果，但要具体分析不同工具的不同之处，需要理解工具的检测原理，检测依据和检测方法等等。因此，我们项目在推进关于工具检测原理分析的研究，同时也在分析各种常见的架构反模式，立志于帮助统一化反模式的定义与分类。

# 5 结论/Summary

开源项目任重道远，有趁手好用的架构分析工具将会为广大开源开发者提供如虎添翼的效果。通过这次研究，我们也更加了解到软件开发过程中的各种问题，并且学习如何解决他们。关于架构反模式，也认为当前业界对反模式的定义不统一，不同检测工具的检测效果也大不相同，反模式也值得未来进一步研究。