ZJUI学院2022年暑期科研成果展
100号队伍：王杰、朱杭刚、蔡奕涛
指导老师：万志远（软件工程学院）

软件工程角度分析开源软件项目的架构反模式

Architectural Antipatterns Research Based on Software Engineering

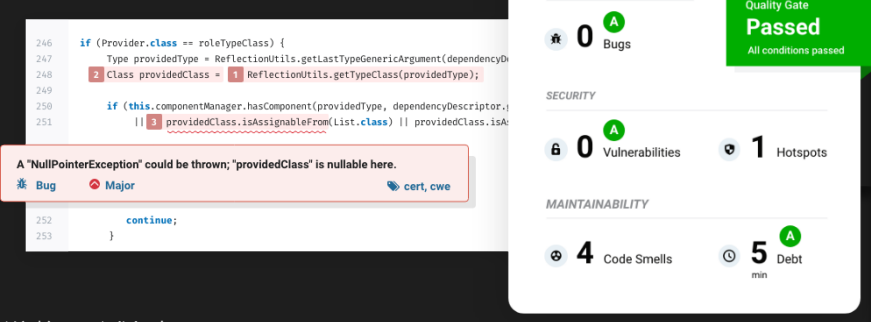
summer_intern > build > poster.jpg

#define Introduction

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, tools to detect smell and help maintenance are necessary. Our team analyzed five tools' performance, producing well-defined architecture smell lists, which can help developers quickly set up the tool they need.

#define Keyword

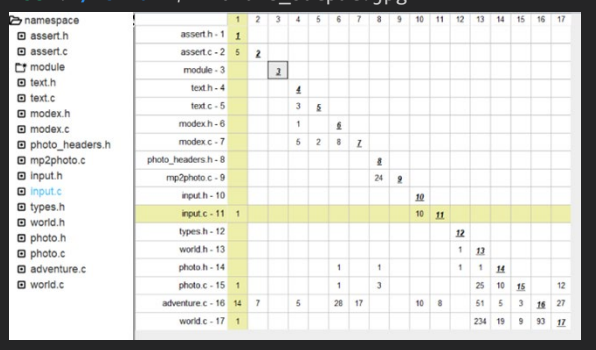
Architecture smell, architecture antipatterns, maintenance, smell detection, open-source.



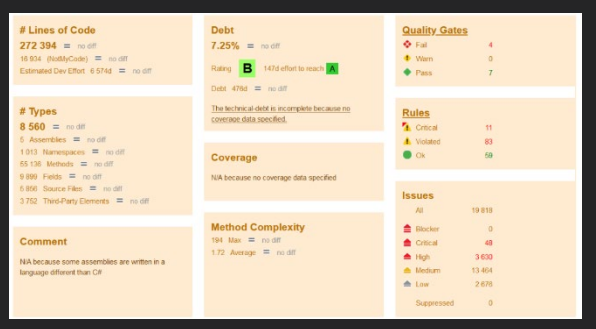

```c
/* Representative anti-patterns */
switch (anti_pattern){
case Cyclic_Dependency:
        项目发展过程中，包之间往往需要一定的依赖关系。但实际开发过程中时常不自觉构成环形依赖，形成复杂的依赖关系。随着项目的发展，复杂的依赖关系不利于项目维护，导致不断增加的技术债。
    break;
case Unstable_Dependency:
        面向对象开发的项目，不同的文件和包有不同的重要程度和修改频率。当一个子系统依赖于更加不稳定的子系统时，项目会更加不稳定，极大增强项目的维护成本。此反模式违反了稳定依赖原则。
    break;
case 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。
    break;
case 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。若一个成分包含过多的子部分，新增功能时将难以分解出其功能，造成额外的维护成本。
    break;
case Scattered_Functionality:
        一个功能依赖过多的分散子功能，导致功能混杂时，同样会增加项目维护成本。一次改动会引起大量其他功能的改动，造成不健康的改动链。
    break;
case Dense_Structure:
        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。这同样会对架构上增加技术债以及可能的安全漏洞。
    break;
}

/* Properties of different Code Quality Analysis Tools */
report_t Designite(const char *code){
        免费开源的代码质量分析工具，可以检测C#和Java两种语言。依赖环境较为简单，安装方便。能给出详细的项目质量分析评估，和异味分析的原由，但没有较好的可视化，且无法直接定位问题代码位置。
}
report_t SonarQube(const char *code){
        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。}
report_t Sonargraph(const char *bytecode){
        更注重与可视化依赖关系。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，可以帮助开发者梳理大型项目依赖关系，排查解决环形依赖关系的反模式。
}
report_t Arcan(const char *code, log_t *commit_history){
        免费开源的代码分析工具，输出表格数据，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。支持环形依赖、不稳定依赖、高扇入扇出结构、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。
}
report_t DV8(const char *code, log_t *commit_history, bug_t issue_tracker){
        DV8 支持循环依赖、不稳定依赖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，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。
}

/* Summery */
    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。
    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。
```

```
ZJUI:~/workdir$ tree smell_category
Smell_category
├── Architecture_Smells
│   ├── Maintenance
│   │   ├── change_based
│   │   │   ├── Duplicate_Functionality
│   │   │   └── Logical_Coupling
│   │   ├── concern_based
│   │   │   ├── Concern_overload
│   │   │   └── Scattered_parasitic_functionality
│   │   ├── dependency_based
│   │   │   ├── Dependency_cycle
│   │   │   └── Link_Overload
│   │   └── interface_based
│   │       ├── Ambiguous-Interface
│   │       ├── Brick_Functionality_Overload
│   │       ├── Lego_Symdrome
│   │       ├── Sloppy_delegation
│   │       ├── Unused_Brick
│   │       └── Unused_Interface
│   └── Others
├── Code_Smells
└── Design_Smells
ZJUI:~/workdir$ viu sonargraph_output.jpg
```

```
ZJUI:~/workdir$ viu dv8_output.jpg
```



```
ZJUI:~/workdir$ viu Ndepend_output.jpg
```



/* More Information */

WSL: Ubuntu-22.04 | master | ⊗ 0 ⚠ 0 | Ln 233, Col 1 | Tab Size: 4 | UTF-8 | LF | C | Linux