桂林电子科技大学2023-2024学年 第3学期

**软件测试实践 实验报告**

**课号：2330183         任课老师：刘斯韵**

|  |  |  |  |
| --- | --- | --- | --- |
| 项目名称 | **实验二、静态代码分析** | | |
| 院系 | **计算机与信息安全学院** | 专业 | **软件工程** |
| 学号 | 2200310913 | 姓名 | **李昊戈** |
| 实验日期 | **2024年7月 5日** | | |

一． 实验目的

1、掌握静态代码分析的技术原理和作用。

2、掌握面向Java和C/C++等不同程序设计语言的典型静态代码分析工具的使用。

二． 实验内容

1、使用Idea或其插件提供的静态代码分析工具，对自己在以前的项目或实验中设计的Java或者C/C++程序代码进行静态代码分析（如果代码问题较少，也可试着设计一些问题），并给出分析结果（如分析工具的截图等）。

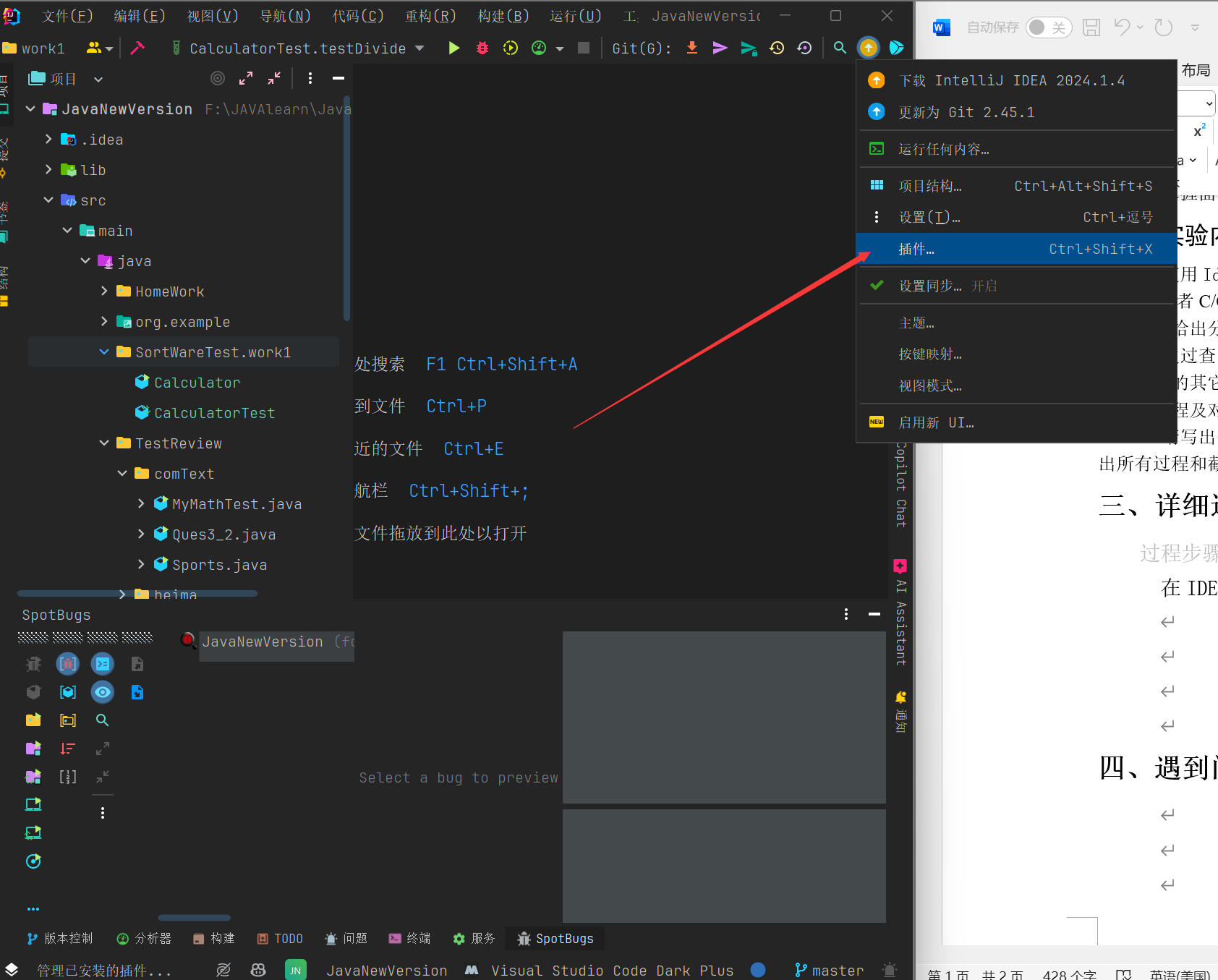
2、通过查阅相关资料，熟悉你所使用的静态代码分析工具的菜单中提供的除基本分析功能之外的其它各种功能以及设置，了解其更多功能，并在报告中给出至少3项以上的功能使用过程及对应的结果和分析总结。

3、请写出你使用静态代码分析工具安装和使用方法，如同指导书那样具体且详细地列出所有过程和截图。

**三、详细过程**

过程步骤和截图，以及代码分析

 在IDEA安装SpotBugs

 点击: 

直接搜索:SpotBugs

直接安装即可:

图形用户界面

描述已自动生成

然后右键你要检测的项目:

电脑萤幕画面

描述已自动生成

电脑萤幕画面

描述已自动生成

直接在出现的菜单显示使用即可:

图形用户界面

中度可信度描述已自动生成

 相关解释:

 **Analyze Selected File (Ctrl+Alt+Shift+F)**：分析当前在编辑器中打开的文件。

 **Analyze Class (Non-Anonymous) Under Cursor (Ctrl+Alt+Shift+C)**：分析光标所在的类（不包括匿名类）。

 **Analyze Package(s) Files**：分析选定的包中的所有文件。

 **Analyze Module Files Not Including Test Sources**：分析模块中的所有文件，但不包括测试代码。

 **Analyze Module Files Including Test Sources**：分析模块中的所有文件，包括测试代码。

 **Analyze Project Files Not Including Test Sources**：分析项目中的所有文件，但不包括测试代码。

 **Analyze Project Files Including Test Sources**：分析项目中的所有文件，包括测试代码。

 **Analyze Scope Files**：分析当前范围内的文件（范围可以是自定义的）。

 **Analyze All Modified Files (Alt+Shift+C)**：分析自上次提交以来所有修改过的文件。

 **Analyze Changelist Files (Alt+Shift+A)**：分析变更列表中的文件（变更列表是版本控制系统中的概念）。

 分析一个矩阵乘法的类:

电脑屏幕截图

描述已自动生成

相关方法:

文本

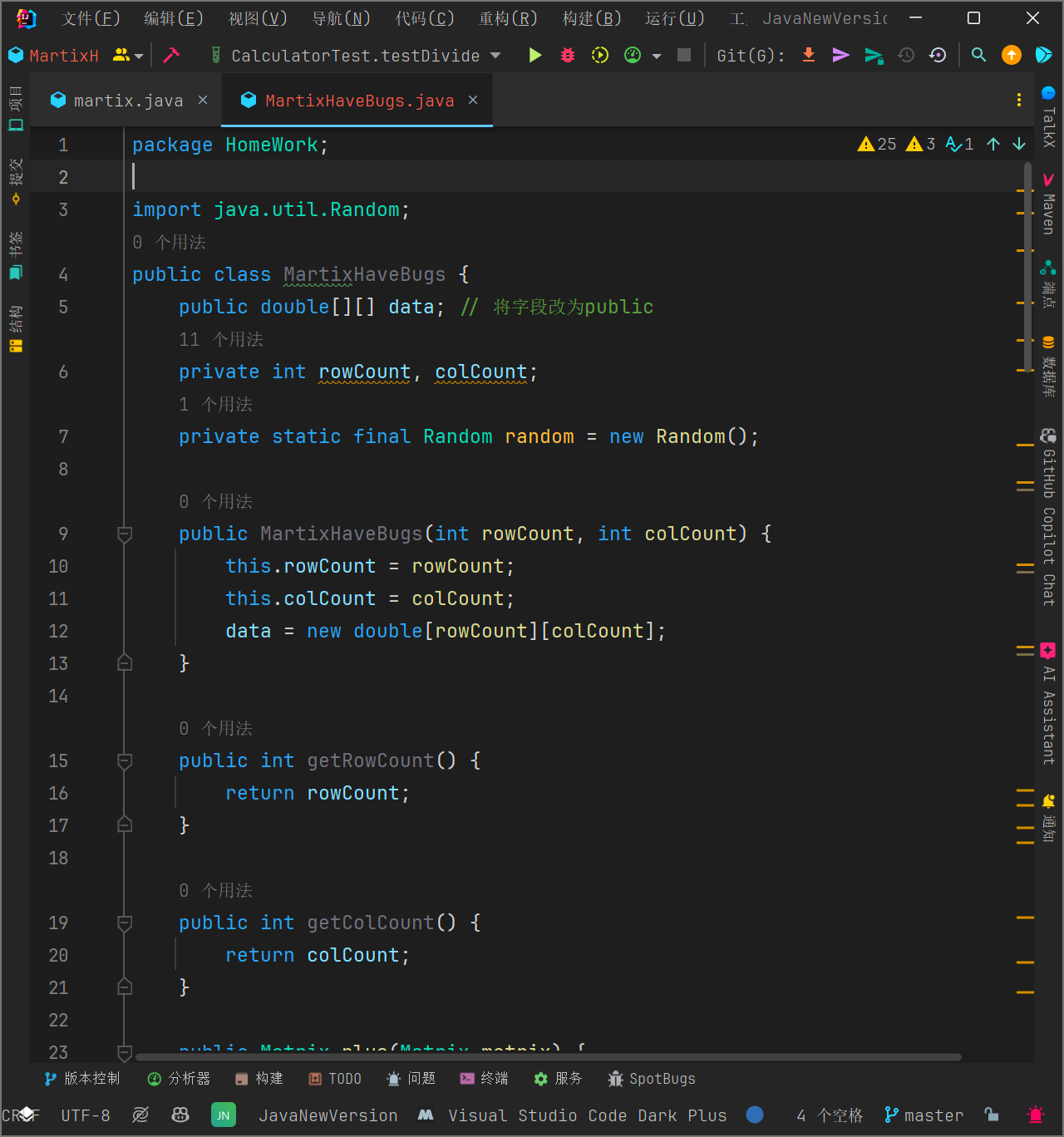
描述已自动生成

 使用spotBugs分析代码(实际是分析java的字节码.class)

图表

描述已自动生成

修改源文件,使得have bugs



电脑萤幕画面

描述已自动生成

文本

描述已自动生成

 data字段被设置为public，违反了封装原则。

 plus(Matrix matrix)方法中有一个数组越界的bug。

 plus(double scalar)方法直接返回null，没有实现功能。

 toString()方法使用了字符串拼接，而不是StringBuilder，可能导致性能问题。

 添加了一个未使用的私有方法unusedMethod()。

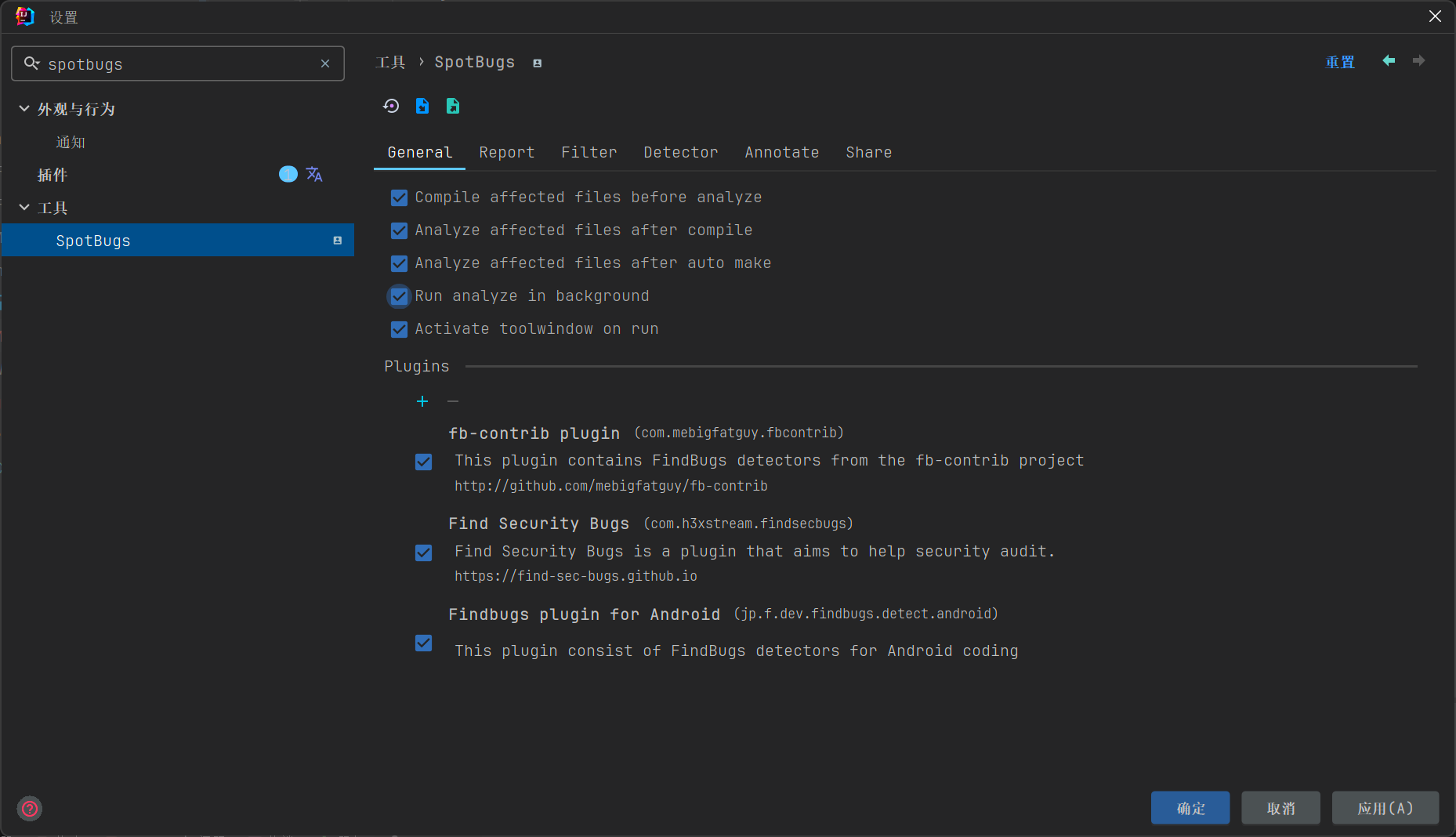
 添加了一个可能引发空指针异常的方法potentialNullPointer()。

上文是具体的bug

遗憾的是:spotBugs竟然没发现,麻了文本

描述已自动生成

可以在相关的设置里面给spotbugs添加插件:



然后再次分析:

文本

描述已自动生成

有点fw了,麻了

可以分析整个软件包的代码:

图形用户界面

描述已自动生成

文本

描述已自动生成

可以分析自己选中的但是跳过测试代码的:

文本

描述已自动生成

**四、遇到问题和解决办法**

 问题：SpotBugs 最初没有检测到一些故意引入的 bug。

 解决方法：在 SpotBugs 设置中，添加额外的插件以提高其检测能力。这改进了工具在代码中识别更多潜在问题的能力。

问题：一些特定于 Java 的问题，比如违反封装的公共字段，并没有立即被标记出来。

 解决方法：考虑使用额外的、专门为 Java 设计的静态分析工具，如 PMD 或 SonarQube，以补充 SpotBugs 并捕获更广泛的代码质量问题。

问题：分析结果有时会让人感到不知所措，特别是对于较大的代码库。

 解决方法：使用 SpotBugs 提供的各种过滤选项，专注于特定类型的 bug 或严重级别。这有助于优先处理最关键的问题。

**五、实验总结**

在本次实验中:我了解了静态代码分析的重要性和基本原理。

静态分析工具如SpotBugs可以在不运行代码的情况下识别潜在的bug和代码质量问题，这对于提高软件质量和可维护性至关重要。

而且我还学会了如何在IntelliJ IDEA中安装和使用SpotBugs插件。这个过程包括插件的安装、配置以及如何对不同范围的代码（从单个文件到整个项目）进行分析。

实验中，我尝试了SpotBugs的多种功能，包括分析选定文件、分析整个包、分析模块文件（包括和不包括测试源码）等。这些不同的分析范围使得工具更加灵活，能够适应不同的开发和审查需求。

通过故意引入一些常见的编程错误，如数组越界、不恰当的封装、潜在的空指针异常等，我更好地理解了静态分析工具的检测能力和局限性。这也强调了在实际开发中结合多种质量保证方法的重要性。

实验过程中遇到的问题，如某些bug未被检测到，让我意识到静态分析工具并非万能的。它们是开发过程中的有力辅助，但不能完全替代代码审查和测试等其他质量保证措施。

同时呢,这次实验增强了我对代码质量和最佳实践的认识。我认识到静态代码分析是持续集成和持续交付（CI/CD）流程中的重要一环，能够帮助开发团队更早地发现和修复潜在问题，从而提高软件的整体质量3002