**《软件测试》**

**软件测试实验五：基路径测试方法**

**姓 名： 胡国昌 学 号： 202210120518**

**院 系： 计算机与信息学院 专 业：计算机与科学**

**实 验 室： B1-119 实验日期： 2025.3.21**

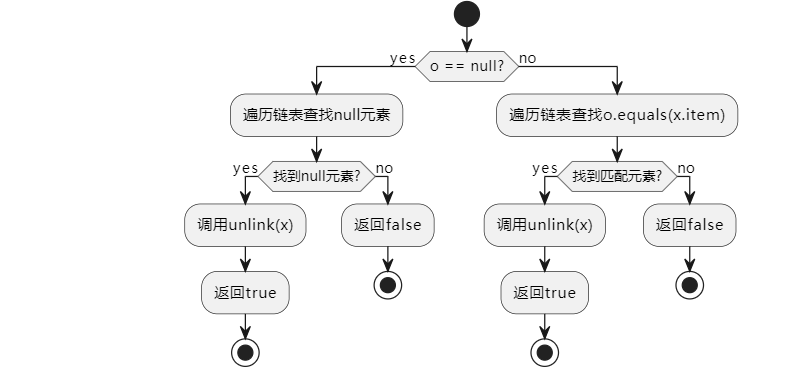
**总评成绩： 审阅教师： 杨青**

# 实验目的：

1. 学习逻辑覆盖准则
2. 学习绘制程序图
3. 学习基路径测试
4. 学习测试用例评估方法

# **实验要求**

1. 针对上面的remove方法，设计尽可能少的测试用例，要求达到 100% 语句覆盖率，参考remove方法的文档说明
2. 创建remove方法的程序图；
3. 根据创建的程序图，计算圈复杂度
4. 采用基路径测试方法，构造基路径集
5. 根据基路径方法，构造测试数据
6. 使用Junit5框架实现自动化测试，提交到码云仓库
7. 配置插件Pitest，评估测试用例有效性
8. 针对实验中的经验和教训，编写实验总结
9. 按照实验报告模板 编写实验报告，以“学号-姓名-软件测试实验五”命名，提交到雨课堂“软件测试实验五”



### 4. 基路径测试

**基路径集包括四条路径：**

|  |  |  |
| --- | --- | --- |
| o为null | 找到null元素 | 返回true |
| o为null | 未找到null元素 | 返回false |
| o不为null | 找到对应元素 | 返回true |
| o不为null | 未找到对应元素 | 返回false |

## 代码实现：

### Java部分

**LinkedListTest.java**

package cn.edu.ctgu;

import org.junit.jupiter.api.BeforeEach;

import org.junit.jupiter.api.DisplayName;

import org.junit.jupiter.api.Test;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import static org.junit.jupiter.api.Assertions.\*;

public class LinkedListTest

{

private static final Logger LOGGER = LoggerFactory.getLogger(LinkedListTest.class);

private LinkedList < String > list;@

BeforeEach

void setUp()

{

list = new LinkedList < > ();

}

/\* 基路径1: o=null且存在目标节点 \*/

@

Test@ DisplayName("移除存在的null元素")

void shouldRemoveNullWhenExists()

{

list.add(null);

list.add("dummy"); // 验证多节点遍历

LOGGER.debug(" List: {}", list);

assertTrue(list.remove(null), "应成功移除null");

LOGGER.debug(" list.contains: {}", list.contains(null));

assertFalse(list.contains(null), "移除后不应包含null");

LOGGER.debug(" List: {} ,list size : {}", list, list.size());

assertEquals(1, list.size(), "其他元素应保持存在"); // 新增完整性检查

}

/\* 基路径2: o=null且不存在目标节点 \*/

@

Test@ DisplayName("移除不存在的null元素")

void shouldNotRemoveNullWhenAbsent()

{

list.add("element1");

list.add("element2");

LOGGER.debug(" List: {}", list);

assertFalse(list.remove(null), "无null元素时应返回false");

LOGGER.debug(" List: {} ,list size : {}", list, list.size());

assertEquals(2, list.size(), "原始元素不应被修改");

}

/\* 基路径3: o≠null且存在目标节点 \*/

@

Test@ DisplayName("移除存在的非null元素")

void shouldRemoveExistingNonNullElement()

{

String target = "target";

list.add("dummy1");

list.add(target);

list.add("dummy2");

LOGGER.debug(" List: {}", list);

assertTrue(list.remove(target), "应成功移除存在的非null元素");

LOGGER.debug(" list.contains: {}", list.contains(target));

assertFalse(list.contains(target), "移除后不应包含目标元素");

LOGGER.debug(" List: {} ,list size : {}", list, list.size());

assertEquals(2, list.size(), "其他元素应保持存在");

}

/\* 基路径4: o≠null且不存在目标节点 \*/

@

Test@ DisplayName("移除不存在的非null元素")

void shouldNotRemoveNonExistingElement()

{

list.add("element1");

list.add("element2");

LOGGER.debug(" List: {}", list);

assertFalse(list.remove("ghost"), "不存在的元素应返回false");

LOGGER.debug(" List: {}", list);

LOGGER.debug(" List: {} ,list size : {}", list, list.size());

assertEquals(2, list.size(), "原始元素不应被修改");

}

/\* 覆盖空列表的特殊路径 \*/

@

Test@ DisplayName("从空列表移除null")

void shouldFailRemoveFromEmptyList\_NullCase()

{

LOGGER.debug(" List: {}", list);

assertFalse(list.remove(null), "空列表移除null应失败");

LOGGER.debug(" List: {} ,list size : {}", list, list.size());

}@

Test@ DisplayName("从空列表移除非null")

void shouldFailRemoveFromEmptyList\_NonNullCase()

{

LOGGER.debug(" List: {}", list);

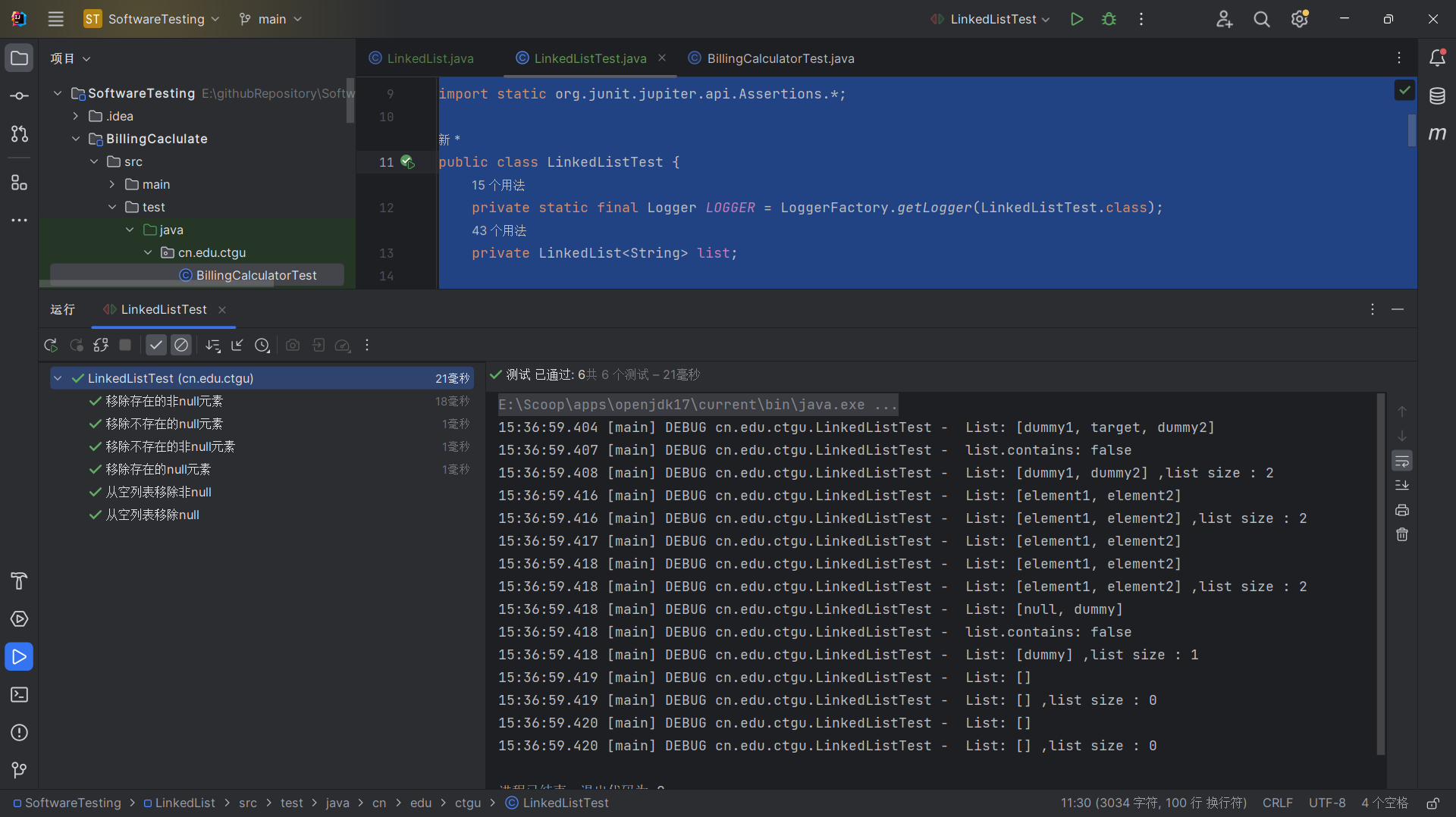
assertFalse(list.remove("any"), "空列表移除非null应失败");

LOGGER.debug(" List: {} ,list size : {}", list, list.size());

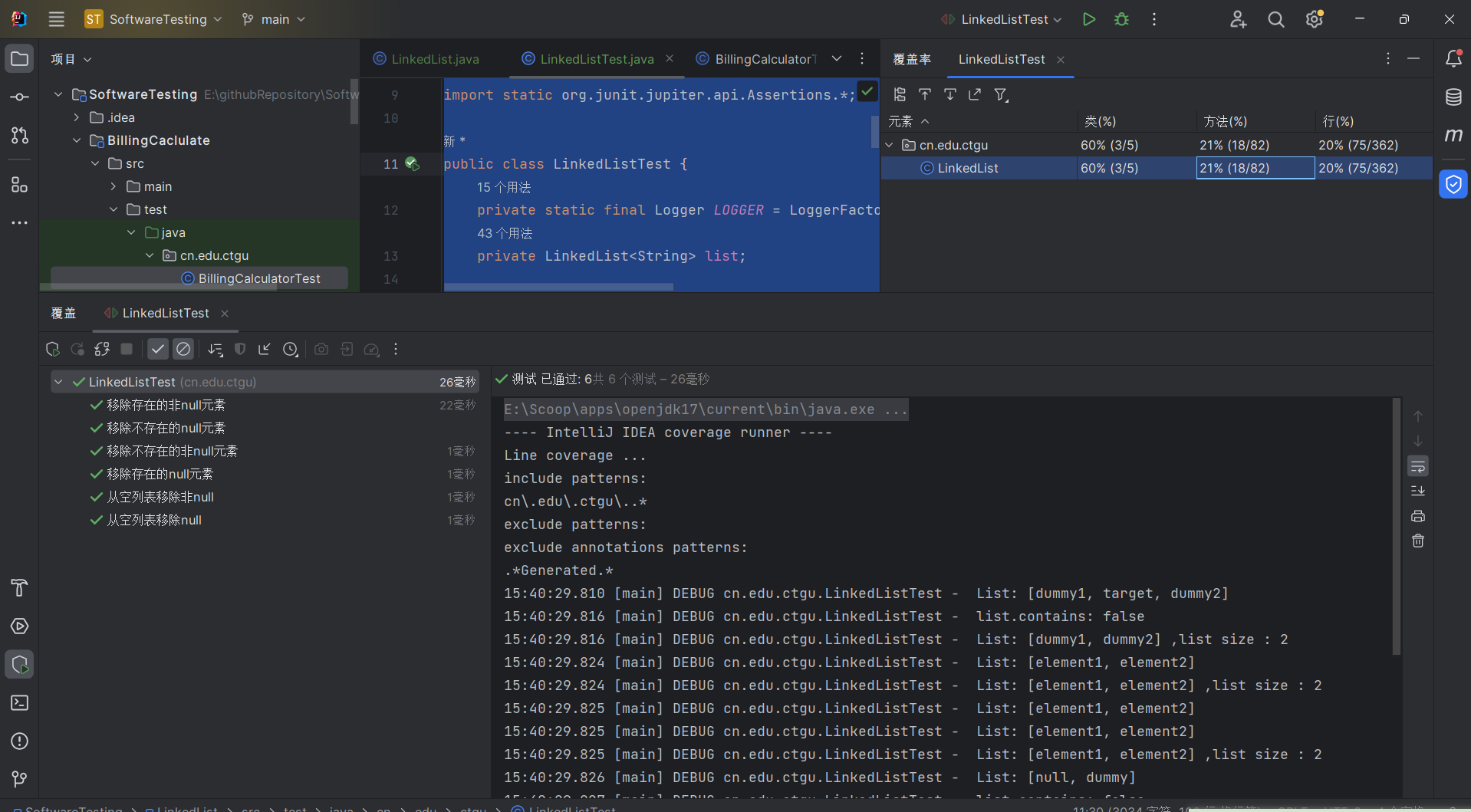
}

}

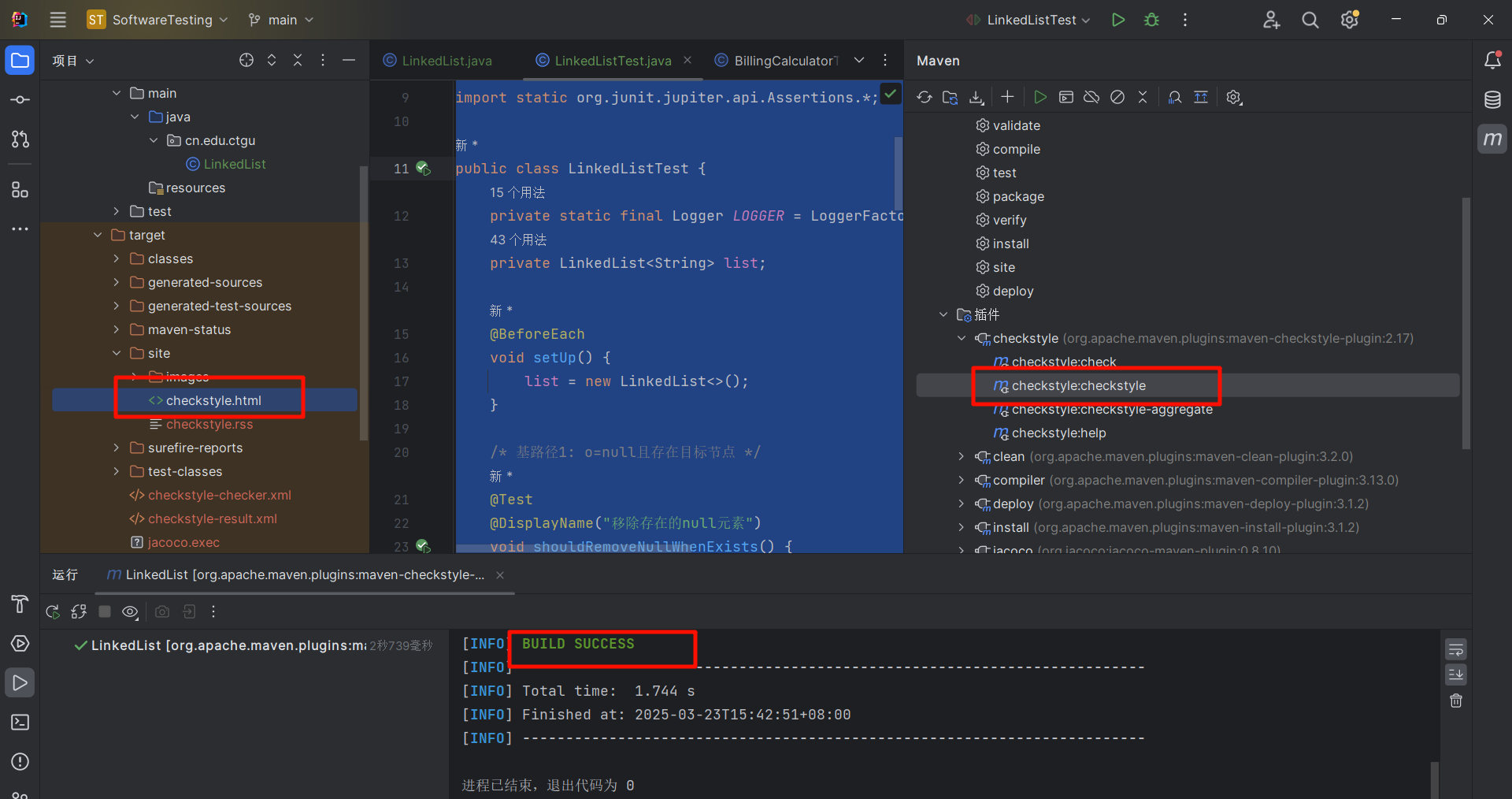
#### Idea工具测试用例 覆盖率结果 checkstyle jacoco

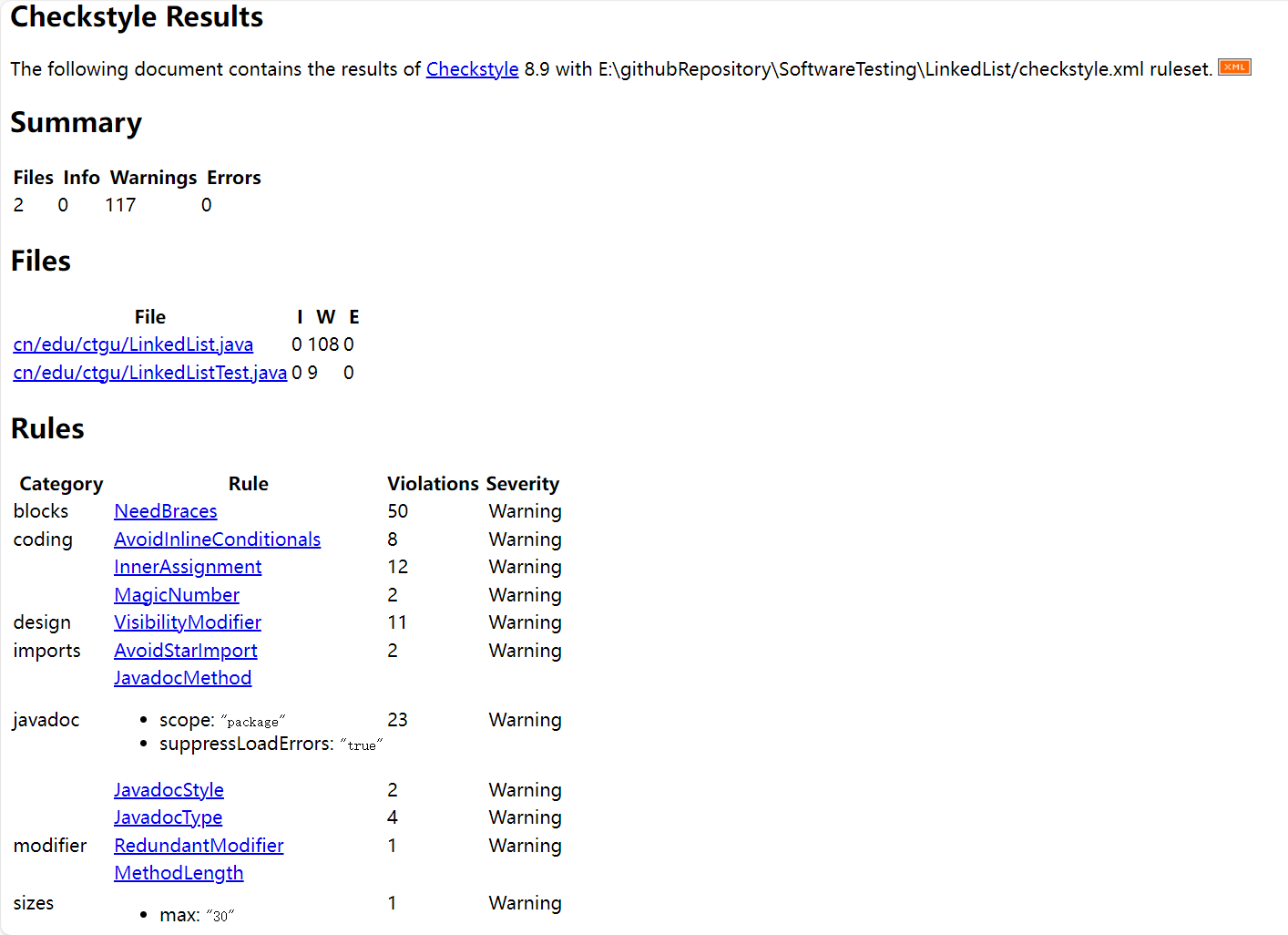


**覆盖率**

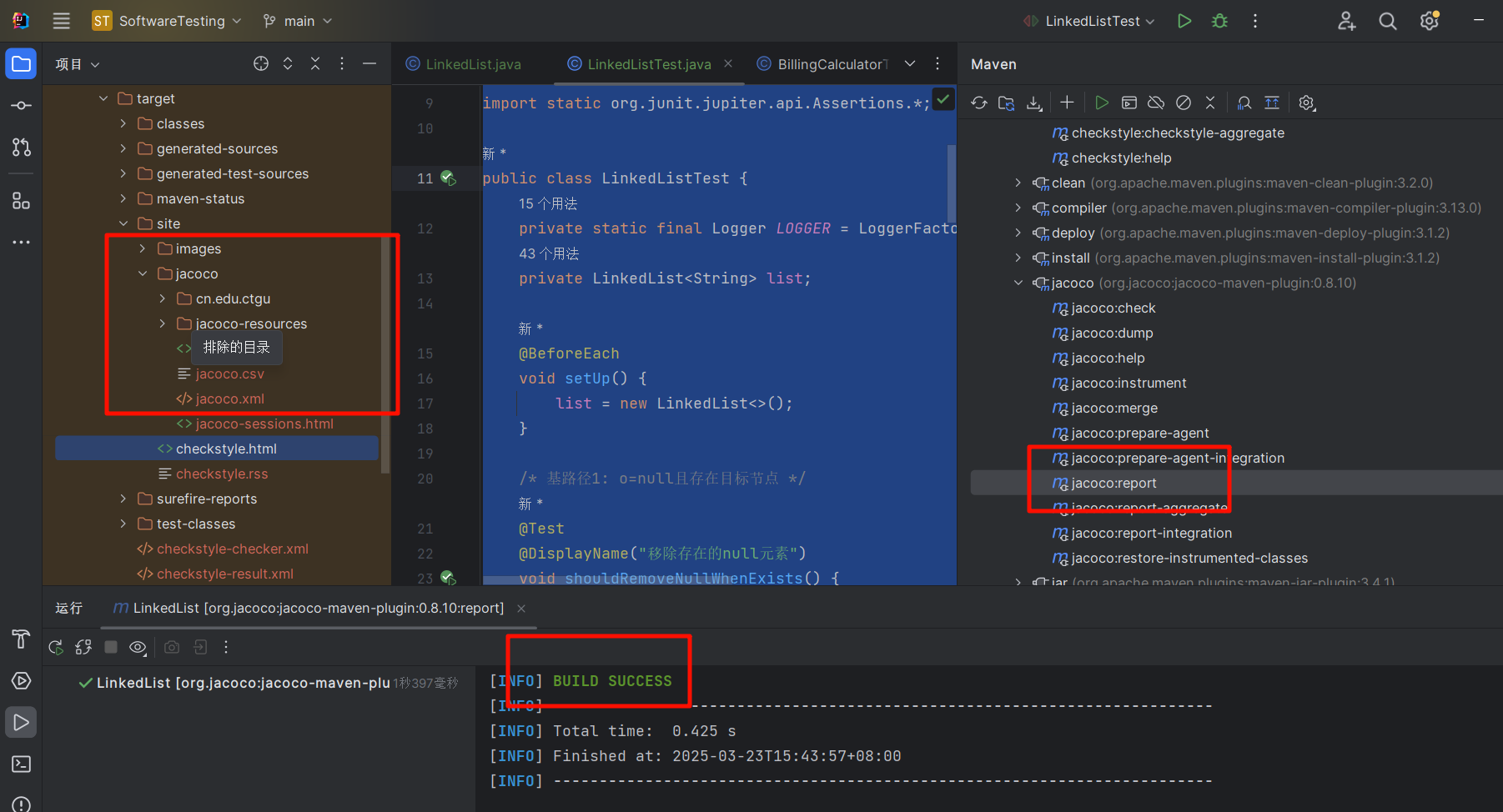


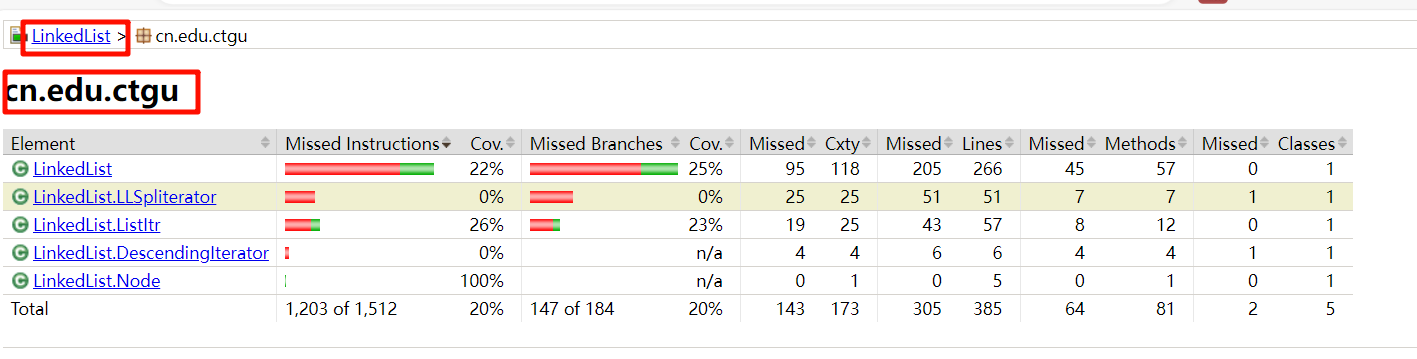
**Checkstyle**



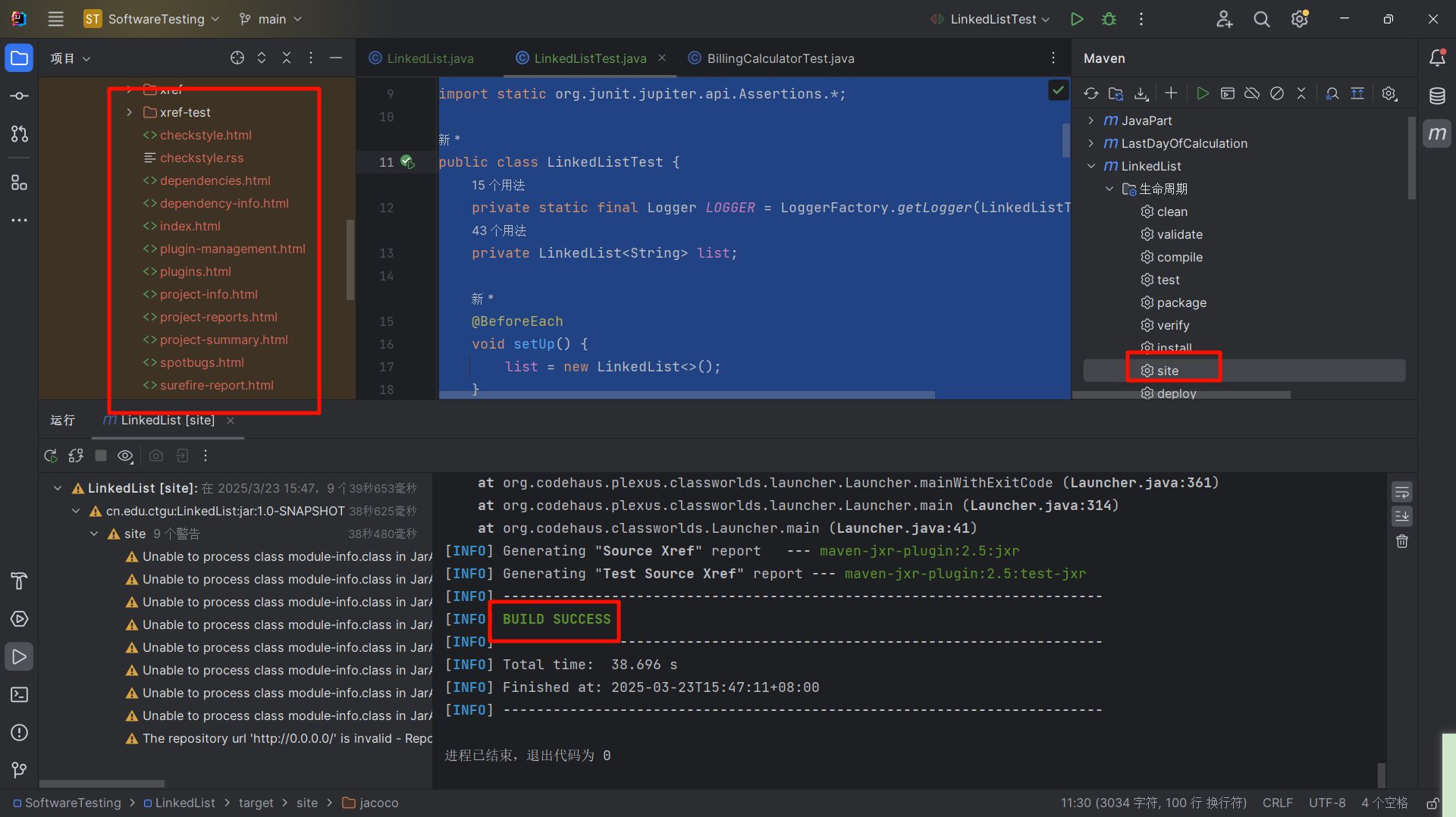


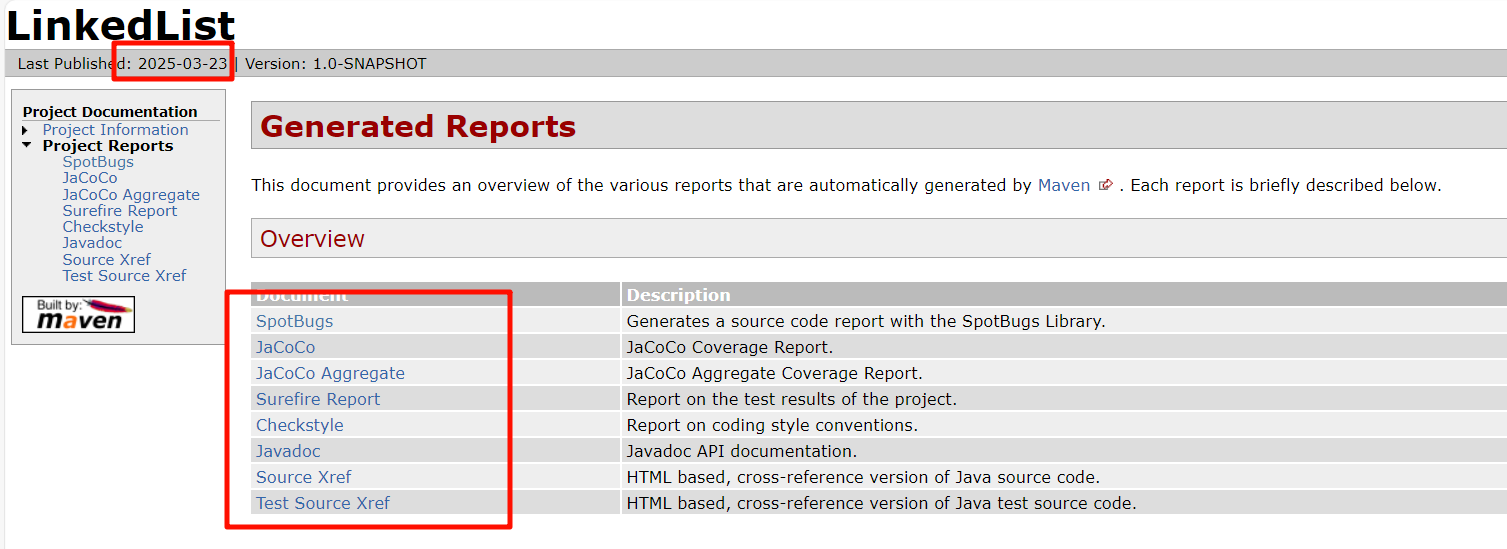
**Jacoco**



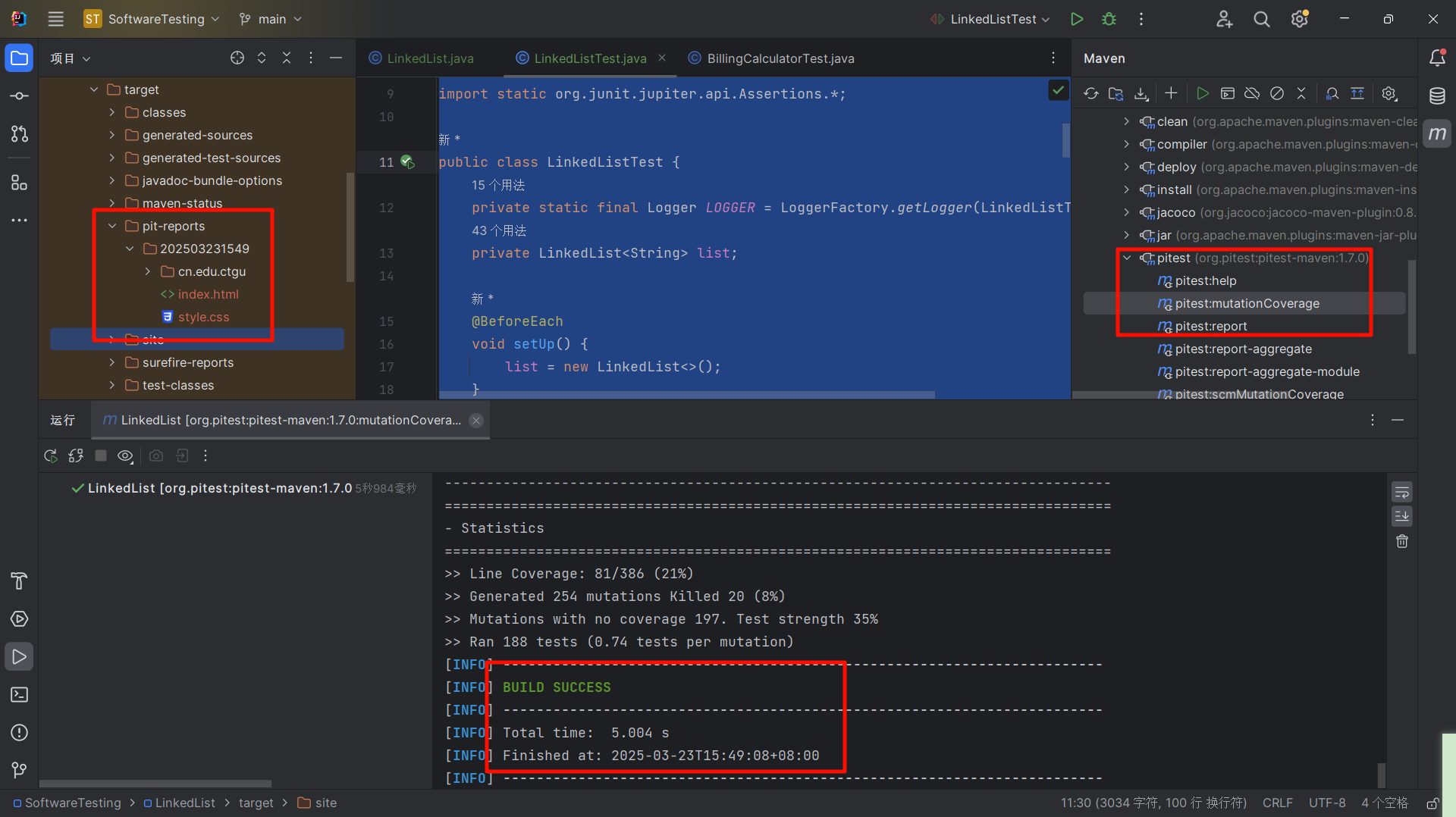


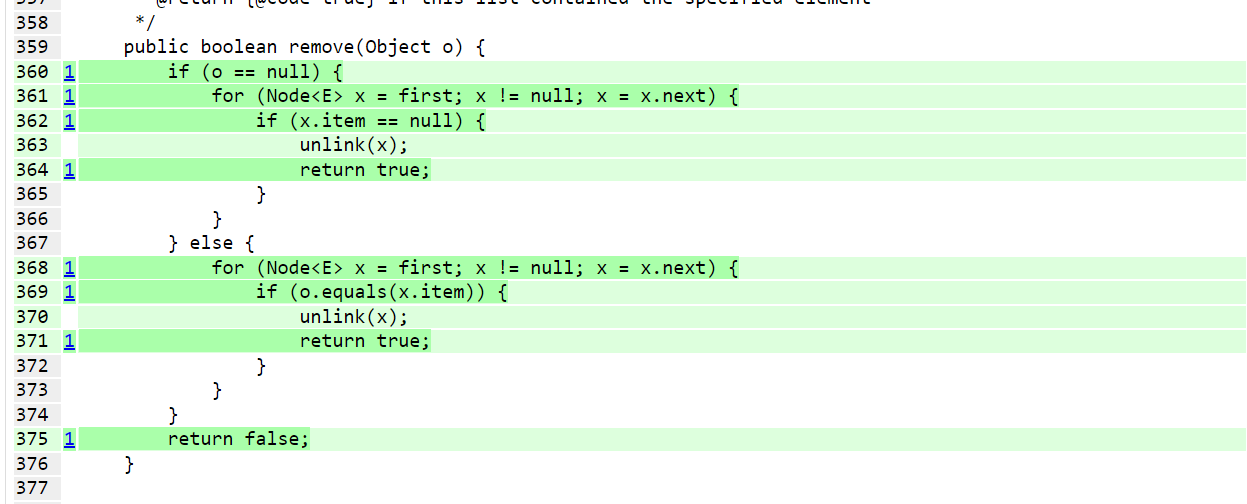
**spotbugs报告:**





**Pitest**





# 五、结论分析与体会

测试与报告的依赖关系：无论是 Go 还是 Java，生成报告的前提都是先运行测试。例如，单独执行 mvn jacoco:report 会失败，必须通过 mvn test jacoco:report 确保测试数据已生成。单独执行 mvn checkstyle:checkstyle会失败，必须通过 mvn test checkstyl:checkstyle确保测试数据已生成。

Java 的生态复杂性：Java 依赖 Maven/Gradle 等构建工具，通过插件（如 Surefire、Jacoco）实现测试和覆盖率统计。使用chcekstyle进行静态调整代码规范

spotbugs静态代码分析 发现潜在缺陷（空指针、资源泄漏、线程安全等）。识别代码异味（冗余逻辑、低效实现）。

Pitest 进行突变代码检测 在检测代码测试有着不一样的效果 在下载依赖的时候由于是jdk17 在很多地方 对老师的pom文件进行适当的修改来匹配jdk17

# 仓库地址

https://github.com/xieyangyuyue/SoftwareTesting

[xieyangyuyue/SoftwareTesting: 软件测试--java (github.com)](https://github.com/xieyangyuyue/SoftwareTesting)