本周小结（第三周）

恭喜完成《软件测试与质量》课程第三周的学习。

本周我们讨论了课程的第二部分 技术篇，并主要围绕第 3 章 白盒测试技术展开讨论。

本周，我们主要回答了如下的问题。

1 什么是白盒测试？

白盒测试就是基于程序的源代码，已知产品的内部工作过程，主要对程序内部结构展开测试，关注程序实现细节的测试方法。

白盒测试的基本原理如图 1.1 所示：

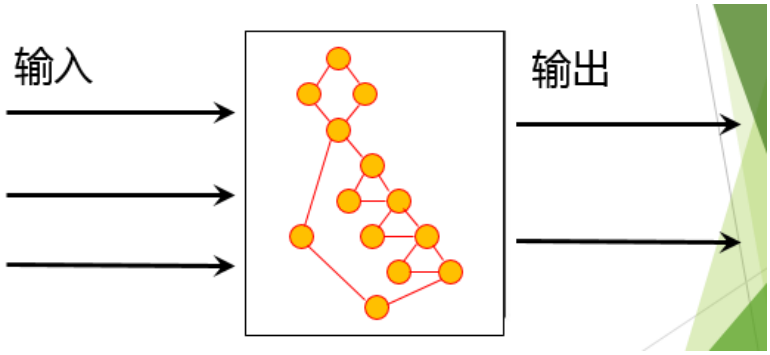

图 1.1 白盒测试的基本原理

2 什么是控制流分析技术？

导致程序结构变得复杂的主要因素，以及控制程序执行流程发生变化的主要因素是判定节点。控制流分析方法的核心就是围绕判定节点来设计测试用例，展开测试。

控制流分析可从三个方面展开：

- 关注判定表达式。利用不同的覆盖指标，对判定表达式中所包含的数据变量、子表达式等进行检查，分析可能的程序分支，即对判定的测试。
- 关注路径。判定节点的引入对程序的执行路径带来不同程度的影响，需要分析路径风险，优选路径进行测试，保证测试效率，即独立路径测试。
- 关注循环体。即针对循环的测试。

3 如何对判定节点展开测试用例设计？

不同的覆盖指标之间的比较如表 3.1 所示。

表 3.1 不同覆盖指标的比较

名称	含义	特点及不足
语句覆盖	设计测试用例时，需要保证程序中每一条可执行语句至少应执行一次。	控制流图中的点覆盖。 是最弱的覆盖指标。 仅关注可执行语句，而非判定节点； 对隐式分支无效。
判定覆盖	设计测试用例时，应保证程序中每个判定节点取得每种可能的结果至少一次。	控制流图中的边覆盖。 满足语句覆盖。 仅关心表达式的整体取值，不能覆盖到每个子条件的所有取值情况。
条件覆盖	设计测试用例时，应保证程序中每个复合判定表达式中，每个简单判定条件的取真和取假情况至少执行一次。	不一定满足判定覆盖。即局部全覆盖不等于整体全覆盖。
判定/条件覆盖	测试用例设计应满足判定节点的取真、取假分支至少执行一次，且每个简单判定条件的取真和取假情况也至少执行一次。	判定覆盖+条件覆盖。
条件组合覆盖	测试用例的设计应满足每个判定节点中，所有简单判定条件的所有可能的取值组合情况应至少执行一次。	方法简单，但测试用例太多，冗余严重。
修正的判定/条件覆盖	在满足判定/条件覆盖的基础上，每个简单判定条件都应独立地影响到整个判定表达式的取值。	判定覆盖+条件覆盖+独立影响性。 步骤如下： (1) 列出所有的简单判定条件； (2) 构建真值表； (3) 对每个简单判定条件，找到独立影响对； (4) 抽取最少独立影响对。

请注意：上述指标在实际使用中并不需要全部采用，通常使用较多的是判定覆盖指标。

另外，白盒测试用例的设计并不等同于调试程序的过程中，通过简单输入一些预先设想的数据，看看程序能够返回正确结果。从代码的层面来说，测试用例设计的更多目的是为了支持自动化单元测试。以及我们通常所谈的白盒测试方法也同样适用于在功能测试中使用，关于这一点，我们在第 4 周的路径测试中会进一步体会到。

4 如何执行同行评审？

静态白盒测试不需要实际运行被测软件，而是直接对软件形式和结构进行分析。静态白盒测试主要包括代码检查、静态结构分析、代码质量度量等。

代码检查主要是通过同行评审来发现缺陷，主要以评审会议为形式，通过多人对软件交付物进行检查，从而发现缺陷或获得改进优化的机会。同行评审方法遵循的评审流程大同小异，但随着这些方法的正式程度不同，适用的对象、评审形式等方面也存在一定的差异（表格略）。

同行评审的一般流程如图 4.1 所示。详细的流程说明见课程视频。

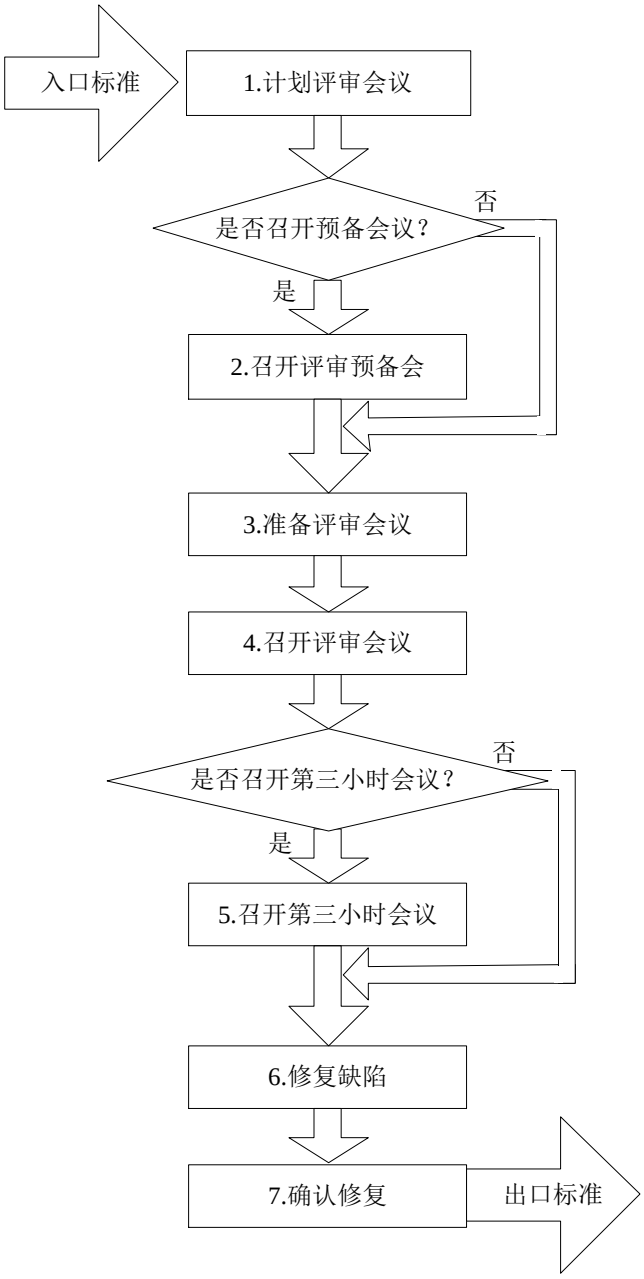

图 4.1 同行评审的一般流程

同行评审通常有三类评审结果。

（1）正常：评审专家做好了评审准备，评审会议顺利进行，达到了预期目的，达成明确的评审结论，不需要再次评审。

（2）延期：30%以上的评审专家并未做好评审准备，会议无法正常进行，需要重新安排评审日程。

（3）取消：初审阶段就发现工作产品中存在太多问题，需要作者进行修复，然后再进行第二次同行评审。

注意：同行评审的有效性取决于评审流程的规范性、评审专家对工作产品和评审流程的熟悉程度，以及所有参与人员的态度等多方面因素。在缺乏规范、缺乏有效组织、缺乏责任心和检查工具的情况下，同行评审只是在浪费时间。所以，不妨从结对编程开始，从遵循行业已有的规范开始，尝试有监督的开发。

5 如何展开初步的程序结构分析？

静态结构分析就是通过分析程序相关的图表，从而快速了解程序设计和结构，更好地理解源代码，找到程序设计的缺陷和代码优化的方向。

5.1 看函数调用图

从函数调用图看程序结构包括如下方面。

（1）看函数调用层次。层次太深，将增大集成测试的负担，造成风险。对栈造成压力，容易导致溢出。要在函数调用层次与单个函数复杂度之间达到平衡。

（2）看函数调用关系，标识高风险节点。调用层次深的节点，根节点，出度大的节点，入度大的节点，都是高风险节点。

（3）看递归调用。如果存在递归调用，尽量改为循环。

（4）看孤立节点。尽量避免孤立节点。

5.2 看控制流图

从控制流图看程序结构包括如下方面：

（1）看孤立节点。从控制流图看是否存在孤立节点。

（2）看出口节点。多出口带来高复杂度，应尽量避免多出口。

（3）看环复杂度。环复杂度应控制在 10 以内。

（4）非结构化设计。应尽量避免非结构化设计。

如何改进结构设计不合理的函数？

（1）避免孤立节点；

（2）尽量避免多次使用 return 语句，尽量将有效性校验前置；

（3）应将完成单一功能的语句块改为函数调用的方式，降低单个函数复杂度；

（4）尽量不使用强制跳转或强制结束语句，避免非结构化设计。

6 下周预告

下周，我们将继续讨论第三章 白盒测试技术，看看如何对路径设计测试用例，并来解决第二章 黑盒测试技术中场景爆炸的问题。