**白盒测试**

深入理解白盒测试，总结其特点。

白盒测试是一种验证技术，软件工程师可以用它来检查他们的代码是否按预期运行。它也被称为结构测试、透明箱测试和玻璃箱测试。白盒测试是考虑到系统或组件的内部机制的测试。本章概述的白盒测试技术允许软件工程师设计测试用例，在一个模块或单元内行使独立的路径；在其真假两面行使逻辑决策；在其边界和操作范围内执行循环；以及行使内部数据结构以确保其有效性。有六种基本的测试类型：单元、集成、功能/系统、验收、回归和测试。白盒测试被用于这六种类型中的三种： 单元测试、集成测试和回归测试。

软件测试有六种基本类型：单元测试、集成测试、功能/系统测试、验收测试、回归测试和Beta测试。其中，白盒测试用于其中三种类型：

* 单元测试：测试单个硬件或软件单元或相关单元组（IEEE，1990）。单元是一个无法细分为其他组件的软件组件（IEEE，1990）。软件工程师编写白盒测试用例来检查单元是否编码正确。在将代码与其他代码集成之前，单元测试非常重要，以确保代码的稳定性。一旦代码集成到代码库中，查找观察到的故障的原因将变得更加困难。此外，由于软件工程师自己编写并运行单元测试，公司通常不会跟踪观察到的单元测试失败，使这些类型的缺陷成为最“私密”的缺陷。我们都更喜欢找到自己的错误，并有机会在别人不知情的情况下修复它们。大约65％的错误可以在单元测试中捕获（Beizer，1990）。
* 集成测试：将软件组件、硬件组件或两者组合并进行测试，以评估它们之间的交互（IEEE，1990）。编写的测试用例明确检查各个单元之间的接口。这些测试用例可以是黑盒测试用例，测试人员明确了解测试用例需要多个程序单元进行交互。或者可以编写白盒测试用例，明确执行测试人员已知的接口。
* 回归测试：选择性地重新测试系统或组件，以验证修改是否引起了意外效果，并且系统或组件仍符合其指定要求（IEEE，1990）。与集成测试一样，回归测试可以通过黑盒测试用例、白盒测试用例或两者的组合来进行。白盒单元测试和集成测试用例可以保存并作为回归测试的一部分重新运行。

白盒测试的相关特点以及总结:

1. 单元测试的重要性：白盒测试在单元测试中起着关键作用。单元测试是对单个软件或硬件单元进行测试的过程，以确保其正确性。白盒测试用例由软件工程师编写，旨在检查单元的正确编码。这种测试对于在将代码与其他代码集成之前确保代码的质量至关重要。
2. 私密性：由于软件工程师自己编写和运行白盒测试，通常不会跟踪观察到的单元测试失败。这使得白盒测试中发现的缺陷成为最私密的缺陷，因为软件工程师更愿意找到并修复自己的错误，而不让其他人知道。
3. 错误捕获率：根据Beizer的研究，大约65％的错误可以在白盒测试中捕获。这表明白盒测试是一种有效的测试方法，可以帮助发现并修复软件中的大多数错误。
4. 集成测试：白盒测试还可以用于集成测试，其中测试人员明确检查各个单元之间的接口。白盒测试用例可以专门测试已知接口的交互，以确保系统的各个组件之间的正确交互。
5. 回归测试的重要性：白盒单元测试和集成测试用例可以被保存并作为回归测试的一部分重新运行。回归测试是选择性地重新测试系统或组件的过程，以验证修改是否引起了意外效果，并确保系统或组件仍然符合其指定要求。

此外，白盒测试作为测试，其对于代码的覆盖率也有要求。覆盖率衡量了测试用例对于代码的执行路径、分支和逻辑的覆盖程度。它是评估测试的全面性和有效性的指标之一。

常见的白盒测试覆盖率指标包括以下几种：

1. 语句覆盖（Statement coverage）：衡量测试用例执行期间是否覆盖了代码中的每一条语句。语句覆盖率要求每个语句至少执行一次。
2. 分支覆盖（Branch coverage）：衡量测试用例是否覆盖了代码中的每个分支或决策点。分支覆盖率要求每个分支的两个可能结果（真和假）都至少执行一次。
3. 条件覆盖（Condition coverage）：衡量测试用例是否覆盖了代码中每个条件语句的所有可能结果。条件覆盖要求每个条件语句的每个可能取值至少执行一次。

选择适当的覆盖率指标取决于测试的目标和要求。较高的覆盖率通常表示测试用例更全面地测试了代码，但并不能保证发现所有的错误或缺陷。

|  |
| --- |
| Plain Text int a = 5; int b = 10; int c = 0; if (a > b) {  c = a - b; } else {  c = b - a; } int result = c \* 2; |

使用语句覆盖的测试用例要求至少执行一次每个语句。在这个例子中，至少需要设计一个测试用例，使得第4行的条件判断为真，执行语句块中的语句，并最终执行到最后一行。

分支覆盖（Branch coverage）是一种白盒测试的覆盖率指标，用于衡量测试用例是否覆盖了代码中的每个分支或决策点。在分支覆盖中，目标是确保每个分支的两个可能结果（真和假）都至少被执行一次。

继续上面的例子，分支覆盖要求至少设计两个测试用例，一个使得第4行的条件判断为真，另一个使得条件判断为假，以确保两个分支都得到执行。

条件覆盖（Condition coverage）是一种白盒测试的覆盖率指标，用于衡量测试用例是否覆盖了代码中每个条件语句的所有可能结果。在条件覆盖中，目标是确保每个条件语句的每个可能取值至少被执行一次。

继续上面的例子，条件覆盖要求至少设计两个测试用例，一个使得条件 (a > b) 为真，另一个使得条件为假，以确保两个可能的结果都得到覆盖。

总结来说，语句覆盖要求每个语句至少执行一次，分支覆盖要求每个分支的两个可能结果至少执行一次，条件覆盖要求每个条件语句的每个可能取值至少执行一次。这些覆盖率指标帮助测试人员评估测试用例对代码的覆盖程度，并提供有关测试全面性的信息。

同时，需要考虑用户可能对代码结构做出的破坏性的使用方式，导致程序出现不寻常的bug，以恶意的思维来考虑用户可能对程序进行的各种操作。

* 检查你所测试的代码中的每个输入。如果输入不正确、字体错误或太大（或太小），你是否能正确处理？
* 从安全的角度来看代码。用户是否可以溢出缓冲区，导致安全问题？
* 检查每个计算。它是否可能造成溢出？你是否防止了可能的除零错误？
* 通过检查输入、关注安全性、检查计算过程等方法，测试人员可以增加测试用例的覆盖范围，提高代码的质量和可靠性。

白盒测试是一种经过精确规划的验证和验证技术。通过明确的输入/输出组合，你可以运行测试用例，了解执行的代码行，并知道预期的结果。如果结果不正确，测试用例就会揭示问题（错误）。幸运的是，你可以根据失败的测试用例知道需要检查哪些代码行。由于有这种控制能力，相比开发周期中的后续阶段，单元测试中的缺陷修复更经济。所以应当尽量使用白盒测试发现并消除尽可能多的缺陷。