**会议记录表**

|  |  |  |  |
| --- | --- | --- | --- |
| 时间 | 2023年6月2日 | 星期 | 第十六周 |
| 队伍 | 啊对对队 | 会议地点 | 线上会议 |
| 进行时长 | 1h |
| 参会人员 | 孟小凡、尹国泰、黄泰达、吴百川 | | |
| 会议主题 | 软件测试 | | |
| 会议内容 | 吴百川  6月2日 10:26  黑盒测试概念  黑盒测试法也称功能测试或数据驱动测试，它是在已知产品所应具有的功能，通过测试来检测每个功能是否都能正常使用，在测试时，把程序看作一个不能打开的黑盒子，在完全不考虑程序内部结构和内部特性的情况下，测试者在程序接口进行测试，它只检查程序功能是否按照需求规格说明书的规定正常使用，程序是否能适当地接收输入数锯而产生正确的输出信息，并且保持外部信息（如数据库或文件）的完整性。  黑盒测试主要发现以下类型的错误：  基于规格说明的功能错误  基于规格说明的构件或系统行为错误  基于规格说明的性能错误  面向用户的使用错误  黑盒接口错误  “黑盒”法着眼于程序外部结构、不考虑内部逻辑结构、针对软件界面和软件功能进行测试。“黑盒”法是穷举输入测试，只有把所有可能的输入都作为测试情况使用，才能以这种方法查出程序中所有的错误。实际上测试情况有无穷多个，人们不仅要测试所有合法的输入，而且还要对那些不合法但是可能的输入进行测试。  黑盒测试注重于测试软件的功能性需求，也即黑盒测试使软件工程师派生出执行程序所有功能需求的输入条件。黑盒测试并不是白盒测试的替代品，而是用于辅助白盒测试发现其他类型的错误。  总体来说，黑盒测试有以下特点：  对于更大的代码单元来说（子系统甚至系统级）比白盒测试效率要高；  测试人员不需要了解软件的实现细节；  从用户的视角进行测试，很容易被理解和接受；  有助于暴露任何规格不一致或有歧义的问题；  没有清洗和简明的规格，测试用例很难设计；  不能控制内部执行路径，会有很多内部程序路径没有被测试到；  不能直接针对特定的程序段，这些程序可能非常复杂（因此可能隐藏更多的问题）  黑盒测试用例设计方法有：  1.等价划分法；  2.边界值分析法；  3.错误推测法；  4.场景法；  5.因果图法；  6.判定表法；  7.正交试验设计法  黑盒测试类型包括：  1.系统测试；  2.验收测试；  3.性能测试；  3.Alpha测试；  4.Beta测试；  5.接口测试；  6.自动化测试等等  孟小凡  白盒测试也称结构测试，透明盒测试。主要用于单元测试阶段，代码和逻辑的测试，重点复杂的测试，是一种测试用例设计方法，不同于黑盒测试，白盒测试是可以看到内部代码如何运作的，可通过测试来检测产品内部是否符合规定正常运行。  优点：  ①可以检测代码中的每条分支和路径  ②揭示隐藏在代码中的错误  ③对代码的测试比较彻底  ④让软件最优化  缺点：  ①投入成本高昂  ②覆盖所有代码路径难度大  ③不能替代集成测试  白盒测试动态方法  动态方法是白盒测试人员主要参与的环节。主要有以下两大类方法：  1、逻辑覆盖法  原则：以程序内部的逻辑结构为基础设计测试用例。  逻辑覆盖包括语句覆盖、判定覆盖、条件覆盖、判定／条件覆盖、条件组合覆盖和路径覆盖。  以上六种覆盖标准发现错误的能力呈由弱到强变化：  ①语句覆盖每条语句至少执行一次。  ②判定覆盖每个判定的每个分支至少执行一次。  ③条件覆盖每个判定的每个条件应取到各种可能的值。  ④判定条件覆盖同时满足判定覆盖条件覆盖。  ⑤条件组合覆盖每个判定中各条件的每一种组合至少出现一次。  ⑥路径覆盖使程序中每一条可能的路径至少执行一次。  之所以六种覆盖标准发现错误的能力有差异，是因为使用每种覆盖标准所设计的测试用例对程序内部逻辑的覆盖率不同。  （覆盖率是用来度量测试完整性的一个指标）  1.1、语句覆盖（SC）  语句覆盖：设计足够多的测试用例，使得运行这些测试用例时，被测程序的每一个语句至少执行一次，其覆盖标准无法发现运算中的逻辑关系错误。  1.2、判定覆盖（DC）  判定覆盖：设计足够多的测试用例，使得程序中的每一个判断至少获得一次“真”和一次“假”，即使得程序流程图中的每一个真假分支至少被执行一次。但若程序中的判定是有几个条件联合构成时，未必能发现每个条件的错误。  1.3、条件覆盖（CC）  条件覆盖：设计足够多的测试用例，使得运行这些测试用例时，使得判定中的每个条件至少有一次取真值，有一次取假值。但未必能覆盖全部分支。  1.4、判定／条件覆盖（DCC）  判定／条件覆盖：设计足够多的测试用例，使得被测试程序中的每个判断本身的判定结果（真假）至少满足一次，同时，每个逻辑条件的可能值（真假）也至少被满足一次。即同时满足100%判定覆盖和100%条件覆盖的标准。  1.5、条件组合覆盖（BCCC）  条件组合覆盖：设计足够多的测试用例，使得被测试程序中的每个判定中条件结果的所有可能组合至少执行一次。显然，满足“条件组合覆盖”的测试用例是一定满足“判定覆盖”、“条件覆盖”和“判定/条件覆盖”的。  1.6、路径覆盖  路径覆盖：设计足够多的测试用例，覆盖被测试程序中的所有可能路径，是最强的覆盖准则。  2、基本路径测试法  理想情况下，路径覆盖需要覆盖程序中所有可能的路径。但在路径数目很大时，真正做到完全覆盖是很困难的，必须把覆盖路径数目压缩到一定限度。例如程序中的循环体只执行一次。所以，基本路径测试法可以理解为压缩后的路径覆盖。  尹国泰  符号测试（Sign test）：符号测试用于比较两个相关样本之间的差异是否存在显著性。它基于样本的符号差异，即将两个相关样本的差值取符号（正差和负差），然后计算正差的数量。如果正差的数量显著大于负差的数量或显著小于负差的数量，那么可以得出两个相关样本存在显著差异的结论。   1. 差分测试（Wilcoxon signed-rank test）：差分测试也用于比较两个相关样本之间的差异是否存在显著性。它基于样本的秩次差异，即将两个相关样本的差值的绝对值按照大小排序，并赋予秩次，然后计算秩次和正负号的乘积。通过对秩次和的正负号乘积进行统计检验，可以判断两个相关样本是否存在显著差异。   黄泰达  差分测试（Differential Testing）是软件工程中常用的一种测试技术，用于发现软件系统中的错误、漏洞和不一致性。差分测试的基本原理是同时运行多个实现或版本的软件，并比较它们的输出来检测差异。  下面是差分测试的一般流程：   1. 选择测试对象：确定要进行差分测试的软件系统或模块。通常选择具有多个实现或版本的软件。 2. 生成测试输入：创建一组测试用例，这些测试用例可以作为输入被同时输入到各个实现或版本中。 3. 运行测试：将相同的测试输入同时输入到不同的实现或版本中，并记录每个实现的输出。 4. 比较输出：对每个实现的输出进行比较，检测是否存在差异。差异可能表明其中一个实现存在错误或漏洞。 5. 分析和修复：分析差异的原因，并进行相应的修复操作。修复可以包括对错误进行修补或对实现进行调整，以使其输出与其他实现一致。   差分测试可以应用于不同层次的软件系统，包括操作系统、编译器、库和应用程序等。它可以帮助发现实现之间的行为差异，包括功能性差异、性能差异和安全性差异等。  通过使用差分测试，可以提高软件系统的质量和可靠性，减少错误和漏洞的存在，并增强对软件变更的鲁棒性和稳定性。 | | |
| 备注 | 无 | | |