
软件测试与质量保证

目录

简答/综述.....	2
1.测试分为哪些过程（阶段）？	2
2.按照不同类型的不同测试方法？	3
3.为什么要进行软件测试，即软件测试的意义：	4
4.什么是软件缺陷？它的表现形式有哪些？	4
5.不同角度分析缺陷怎么来的，软件缺陷产生的各种原因：	4
6.概念辨析	6
计算（综合）题.....	8
1.黑盒测试、白盒测试的常用方法	8
2.逻辑覆盖补充	8
3.情景	9
4.书写方法	12
实验考试	14
软件质量保证与测试-作业 1.....	23
软件质量保证与测试-作业 2.....	31
考后整理部分	33

这个考试会比较灵活

课上讲过的计算题、作业上的计算题，看

题型：简答题、计算题、论述题

不要空着

软件测试：技术、分类、测试分为哪些过程、测试方法有哪些，安装不同的类型分有不同方法，测试过程——黑盒测试、白盒测试、单元——安装不同的类型分有不同方法
为什么要有软件测试，因为软件有缺陷，缺陷怎么来的？从不同角度分析缺陷怎么来的

概念：测试、调试、qcqa, security/safety

计算题：肯定会有 pid(tip?) (pie?) 模型，首先会有找出错误，然后根据这个错误设计不同的测试用例，不同测试用例的哪些不同条件能触发（覆盖）错误，或者覆盖错误以后不会引发根(fail?)，已经讲过了

黑盒测试白盒测试：

黑盒：等价类和边界值分析，会直接给一个真实的场景，基于这个场景/界面自己用黑盒测试方法设计有效等价类、无效等价类、边界值分析的测试用例，根据不同的方法、边界值分析，要知道，给了你一个界面/案例/情景，知道等价类是怎么划分的，什么样的可以用边界值来做，什么样的不可以，对于可以的，会采取什么样的方式设计，进行边界值的划分。好好理解等价类和边界值的概念，以及具体的测试的策略。

白盒：逻辑覆盖，7 种，每一种都要好好看，实验/作业/ppt，满足 100%的逻辑覆盖：语句覆盖、条件覆盖、判断覆盖、修改/条件判定覆盖、条件组合覆盖……

需要设计测试用例实现这 100%覆盖，要画个表出来，告诉这个测试用例覆盖到了哪些条件、判定，是真？假？和课上的一样，可以用铅笔画表格再盖

调试：什么叫调试，调试包含了错误定位，题型相当于实验考试的题型，给一段小(伪)代码，先做一个变异测试，植入几个 bug，分别设计测试用例，执行的时候，有的测试用例跑得过，有的跑不过，基于程序频谱的错误定位方法，给代码行进行基于疑似值的排序，计算公式自己定，思想 cm cs us um，把这四项搞懂

环路复杂度基本路径，设计测试用例能够覆盖到这些基本路径

简答/综述

1.测试分为哪些过程（阶段）？

软件测试的过程可分为单元测试、集成测试、系统测试和验收测试四个主要阶段。
单元测试是针对每个程序单元程序代码的测试，以确保每个程序模块能正常工作为目标。

单元的粒度具体划分按不同的单位与不同的软件有不同，比如有具体到模块的测试，也有具体到类，函数的测试等。

单元测试对应的是代码开发，测试依据是详细设计。

实施者：开发编程人员

单元测试的方法以白盒方法为主，黑盒方法为辅。

集成测试是按照设计要求对已经通过单元测试的模块进行组装后再进行和测试。目的是检验与软件设计相关的程序结构问题，如各模块间组合后的功能实现情况、模块接口连接的成功与否、数据传递的正确性等。

原因是实践表明，有的模块虽然能够单独正常工作，但并不能保证多个模块组装起来也能正常工作。一些局部反映不出来的问题，在全局上很可能暴露出来。

集成测试对应的是程序模块集成，测试依据是概要设计

实施者：开发人员或白盒测试工程师

集成测试主要是白盒测试和灰/黑盒测试。集成方法可参考测试 1，8 题

系统测试是把软件系统搭建起来，检验软件产品能否与系统的其他部分（如硬件、操作系统、数据库等）协调工作，达到软件规格说明书中的功能、性能等方面要求。

系统测试对应的是系统集成和实施，测试依据是系统规格说明。

实施者：独立的测试小组。

系统测试完全采用黑盒测试技术。

验收测试是从用户的角度对软件产品进行检验和测试，看是否符合用户的要求。验收测试可以分成两类，针对具有大量用户的通用软件，可以采用 α 测试+ β 测试， α 测试是由用户在开发环境下完成的测试， β 测试是由用户在用户环境下完成的测试；而针对只有特定用户的专用软件，可以采用用户正式验收测试。

实施者：用户主导

	被测试对象	测试依据
单元测试	程序模块	详细设计
集成测试	装配好的多个软件模块	概要设计
系统测试	软件系统 (包括软件及其运行环境)	软件规格说明
验收测试	可运行的软件系统	软件需求说明 以及其他用户要求

软件测试中还经常提到回归测试。回归测试是指，再对软件进行修改之后，重新对其进行测试，以确认修改是正确的，没有引入新的错误，并且不会导致其他未修改的部分产生错误。

他不是第五个测试阶段。在软件测试的各个阶段都有可能会对软件进行修改，都需要进行回归测试。

2.按照不同类型的不同测试方法？

按过程分：

单元测试的方法以白盒方法为主，黑盒方法为辅。

集成测试主要是白盒测试和灰/黑盒测试。集成方法可参考测试 1，8 题

系统测试完全采用黑盒测试技术。

验收测试由用户主导，有🔨技术。可分为 α 测试和 β 测试。

按热门测试方法细分：

黑盒测试：等价类划分测试法，边界值测试法，判定表驱动法，因果图法，场景法，正交实验法。

白盒测试：静态白盒测试，逻辑覆盖，基本路径覆盖，程序变异测试，符号执行，程序插桩和调试。

还有作业 1，4 题的分法，如下。

分别解释什么是静态测试、动态测试、黑盒测试、白盒测试、人工测试和自动化测试。

所谓静态测试是指不运行被测软件，仅通过分析或检查等其他手段达到检测的目的。

所谓动态测试是指通过运行被测软件，检查运行结果与预期结果的差异，并分析运行效率和健壮性等性能。

黑盒测试是指在对程序进行的功能抽象的基础上，将程序划分成功能单元，然后对每个功能单元生成测试数据进行测试。用这种方法进行测试时，被测程序被当作打不开的黑盒，因而无法了解其内部构造，因此又称为功能测试。

白盒测试又称为结构测试或逻辑驱动测试，它是知道产品内部工作过程，可通过测试来检测产品内部动作是否按照规格说明书的规定正常进行，按照程序内部的结构测试程序，检验程序中的每条通路是否都有能按预定要求正确工作，而不顾它的功能。

广义上，人工测试是人为测试和手工测试的统称。人为测试的主要方法有桌前检查，代码审查和走查。手工测试指的是在测试过程中，按测试计划一步一步执行程序，得出测试结果并进行分析的测试行为。

自动化测试指的是利用测试工具来执行测试，并进行测试结果分析的测试行为。

3.为什么要进行软件测试，即软件测试的意义：

见作业 1，1 题。如下

随着计算机技术的迅速发展和广泛深入的应用，软件质量问题已成为开发和使用软件人员关注的焦点。而由于软件本身的特性，软件中的错误是不开避免的。不断改进的开发技术和工具只能减少错误的发生，但是却不可能完全避免错误。因此为了保证软件质量，必须对软件进行测试。软件测试是软件开发中必不可少的环节，是最有效的排除和防治软件缺陷的手段，是保证软件质量、提高软件可靠性的最重要手段。

4.什么是软件缺陷？它的表现形式有哪些？

见作业 1，2 题。如下

从产品内部看，软件缺陷是软件产品开发或维护过程中所存在的错误、毛病等各种问题；

从外部看，软件缺陷是系统所需实现的某种功能的失效或违背。它的表现形式主要有以下几种：

- (1) 软件未达到产品说明书中已经标明的功能；
- (2) 软件出现了产品说明书中指明不会出现的错误；
- (3) 软件未达到产品说明书中虽未指出但应当达到的目标；
- (4) 软件功能超出了产品说明书中指出的范围；
- (5) 软件测试人员认为软件难以理解、不易使用，或者最终用户认为该软件使用效果不良。

5.不同角度分析缺陷怎么来的，软件缺陷产生的各种原因：

有四角度：

软件自身的特点

(1) 软件需求不明确，或发生变化

软件本身的实际需求不清晰，导致设计目标偏离实际需求，从而引起功能或产品特征上的缺陷。

案例：某网上售票系统，一开始把系统的同时在线购票用户数量定位在十万数量级，但在实际应用中，同时在线购票用户数量可能会达到百万甚至千万数量级，这样就会引起负载或强度问题。系统过载会导致性能下降，而如果负载超过其强度极限，则可能会彻底瘫痪或崩溃。

(2) 软件系统结构非常复杂

系统结构非常复杂,而又无法设计成一个很好的层次结构或组件结构,结果导致意想不到的问题或系统维护、扩充上的困难;即使设计成良好的面向对象的系统,由于对象、类太多,很难完成对各种对象、类相互作用的组合测试,而隐藏着一些参数传递、方法调用、对象状态变化等方面问题。如庞大的 ERP 系统,数字化校园系统等。

(3) 精确的时间同步问题

实时应用需要精确的时间同步

(4) 软件运行环境复杂

系统运行环境的复杂,不仅用户使用的计算机环境千变万化,包括用户的各种操作方式或各种不同的输入数据,容易引起一些特定用户环境下的问题。如:美国迪斯尼公司狮子王游戏软件兼容性问题(游戏只在迪士尼自己开发的游戏平台可以玩,没有考虑普通 PC 用户的系统运行环境)

(5) 通信端口多、存取和加密手段的矛盾性等,会造成系统的安全性的或适用性差等问题。

团队合作

(1) 系统需求分析时对客户的需求理解不清楚,或者和用户的沟通存在一些困难。

(2) 不同阶段的开发人员相互理解不一致。

例如,软件设计人员对需求分析的理解有偏差,编程人员对系统设计规格说明书某些内容重视不够,或存在误解。

(3) 对于设计或编程上的一些假定或依赖性,相关人员没有充分沟通。

(4) 项目组成员技术水平参差不齐,新员工较多,或培训不够等。

设计和实现问题(技术问题)

(1) 系统结构设计不合理、算法选择不科学,造成系统性能低下。

(2) 没有考虑系统崩溃后的自我恢复或数据的异地备份、灾难性恢复等问题,从而存在系统安全性、可靠性的隐患。

(3) 对程序逻辑路径或数据范围的边界考虑不够周全,漏掉某些边界条件,造成容量或边界错误。

(4) 算法错误:在给定的条件下没能给出正确或准确的结果

(5) 语法错误:对于解释性语言程序,只能在测试运行时发现这类问题。

(6) 计算和精度问题:计算的结果达不到所需要的精度

(7) 接口参数传递不匹配,导致模块集成出现问题

(8) 新技术的采用可能涉及技术不成熟或系统兼容性问题。

项目管理问题

(1) 缺乏质量意识,不重视质量计划,对质量、资源、任务、成本等的平衡性把握不好,容易挤掉需求分析、评审、测试、等时间,遗留的缺陷会比较多。

(2) 开发流程不够完善和规范，存在太多的随机性和缺乏严谨的内审或评审机制，容易产生问题。

例如对需求变化、设计更改、代码修正等因素缺乏严格规范的管理机制，导致开发过程难以稳步推进。

(3) 开发周期短，需求分析、设计、编程、测试等各项工作不能完全按照定义好的流程来进行，工作不够充分，结果也就不完整、不准确，错误较多；周期短，还给各类开发人员造成太大的压力，引起一些人为的错误。

(4) 文档不完善，风险估计不足等

6.概念辨析

1.QC 与 QA 概念：

QA 即英文 QUALITY ASSURANCE 的简称，中文意思是质量保证 。

QC 即英文 QUALITY CONTROL 的简称，中文意义是质量控制。

QA 和 QC 的主要区别是：QC 是保证产品质量符合规定,QA 是建立体系并确保体系按要求运作,以提供内外部的信任。

同时 **QC 和 QA 又有相同点：**QC 和 QA 都要进行验证，如 QC 按标准检测产品就是验证产品是否符合规定要求,QA 进行内审就是验证体系运作是否符合标准要求；又如 QA 进行产品稽核和可靠性检测,就是验证产品是否已按规定进行各项活动,是否能满足规定要求,以确保交付的产品都是合格和符合相关规定的。

QC 是控制质量最基础的环节，包含了所有的检验工作。而 QA 中是包含 QC 的，其核心仍在于控制质量。但是，它超出了检测的范围，还包括像培训，文件管理和审核等相关工作。质量管理则是一个更广泛的范围，它不仅包含 QA 和 QC，还包括对质量体系的思考和管理。因此 QA 不仅要知道问题出在哪里，还要知道这些问题解决方案如何制订，今后该如何的预防。QC 要知道如何对问题进行控制，但不一定要知道为什么要这样去控制。

2.课程名概念

软件测试

使用人工或自动的手段来运行或测定某个软件系统的过程，其目的在于检验它是否满足规定的需求或弄清预期结果与实际结果之间的差别 。

软件质量保证

软件质量保证是为保证产品和服务充分满足消费者要求的质量而进行的有计划、有组织的活动，它贯穿于整个软件过程 。

3.测试与调试概念：

测试和调试的含义完全不同。

测试是去发现软件中的问题的过程，是一个可以系统进行的有计划的过程，可以事先确定测试策略，设计测试用例，可以把测试结果和预期的结果进行比较。测试发现的不一

定是错误本身，而可能只是错误的外部征兆或表现。

调试是在发现错误之后消除错误的过程。调试应充分利用测试结果和测试提供的信息，全面分析，先找出错误的根源和具体位置，再进行修正，将错误消除。

从职责上说，测试工作只需要发现错误即可，并不需要修正错误，而调试的职责就是要修正错误。软件开发者有时需要同时肩负这两种职责，对自己开发的程序进行测试，发现问题，并对其进行调试，修正错误

调试的方法主要有：回溯法、原因排除法、归纳法、演绎法。

4.safety 与 security

安全性(Security)是指系统向合法用户提供服务的同时能够阻止非授权用户使用的企图或者拒绝服务的能力；强调的是免受外部（其他）未经授权（非法）的攻击。如交通中红绿灯信号系统突然错乱。

safety 强调的是软件自身的错误（不安全），通过软件连接的设备引发危险（危害）。如黑客攻击。

5.PIE 模型

缺陷 Fault:

指静态存在于程序中的错误代码行。

错误 Error:

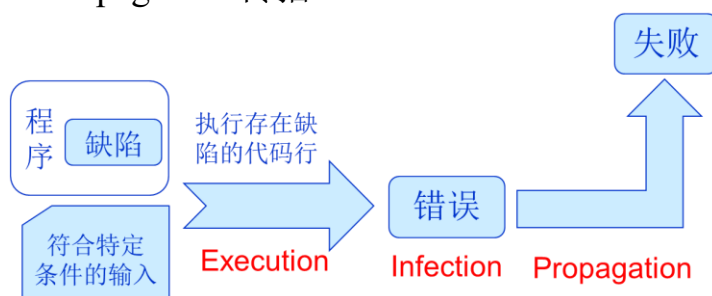
指执行错误代码后导致的内部错误状态。此时若无适当措施(容错)加以及时处理，便产生软件失败。

失败 Failure:

指错误状态传播到软件外部被外部感知。

PIE 模型告诉我们，就算一个程序中有缺陷，但要通过动态测试观察到这一缺陷的外部表现，还需要三个必要的条件：

- 1、程序执行路径必须通过错误的代码（Execution-执行）；
- 2、在执行错误代码的时候必须符合某个或者某些特定条件，从而触发出错误的中间状态（Infection-感染）；
- 3、错误的中间状态必须传播到最后输出，使得观测到输出结果与预期结果不一致（Propagation-传播）。



计算（综合）题

分为两部分，一部分是黑盒测试、白盒测试的常用方法

另一部分关于调试和各种抽象词汇，以 chatgpt-4o 对于实验考试的回答为主。（调试，调试包含的错误定位，变异测试，植入 bug，基于程序频谱的错误定位方法，基于疑似值排序，计算公式自己定之思想 cf cs us uf）

1.黑盒测试、白盒测试的常用方法

在课本《软件质量保证与测试（慕课版）》，看书就行，没书可以看 ppt。书上各类不同的等价类/边界值都要掌握，具体知识这里就不写了。请在掌握书上知识后再看这部分。

等价值划分——31-36（作业 1，11 题）

边界值测试——36-39（作业 1，11 题）

逻辑覆盖——71-76（作业 1，12 题）

基本路径覆盖 77-84（作业 1，13 题）

其他应该不会在计算题考

2.逻辑覆盖补充

根据覆盖目标的不同，逻辑覆盖一般可以分为：

- 1.语句覆盖(SC)
- 2.判定覆盖(DC)
- 3.条件覆盖(CC)
- 4.条件/判定覆盖(C/DC)
- 5.修正条件判定覆盖(MC/DC)
- 6.条件组合覆盖(MCC)
- 7.判定组合覆盖(MDC)

对书上的补充：修改（正）的条件/判定覆盖

修改的条件/判定覆盖比一般的条件/判定覆盖更为严格。

设计足够多的测试用例，使得：

（1）判定表达式中每个条件的真/假取值至少都出现一次，并且每个判定表达式自身的真/假取值也都要至少出现一次

（2）每个条件都显示能独立影响判定结果：

对于条件 C，除 C 外其他条件取值不变

当条件 C 的取值相反时，判定表达式的结果与原结果相反

例题：对于表达式 $(x > 0 \text{ or } y > 0)$ ，请设计满足修改的条件/判定覆盖测试用例集
修改的条件/判定覆盖情况如表所示。 $(x > 0 \text{ or } y > 0)$

x	y	条件表达式 $x > 0$	条件表达式 $y > 0$	判定表达式 ($x > 0$ or $y > 0$)	测试用例
1	0	T	F	T	$x=1, y=0$
0	0	F	F	F	$x=0, y=0$
0	1	F	T	T	$x=0, y=1$
0	0	F	F	F	$x=0, y=0$

附加：判定组合覆盖

设计足够多的测试用例，使得程序中所有判定间的各种组合都至少出现一次。

对于程序段 P1，有两个判定：

D1: ($x > 0$ OR $y > 0$) , D2: ($x < 10$ AND $y < 10$)。

所以，如果要达到判定组合覆盖，需要 4 个测试用例来覆盖 D1 和 D2 的 4 种条件组合。
设计的测试用例如下：

D1 取 Y，D2 取 Y: $x=8, y=8$;

D1 取 Y，D2 取 N: $x=8, y=11$;

D1 取 N，D2 取 Y: $x=-1, y=-1$;

D1 取 N，D2 取 N: 没有满足条件的测试用例；

因此，对于程序 P1 无法生成满足 100%判定组合覆盖的测试用例。

判定组合覆盖虽然标准最高，但由于程序中可执行路径的特点限制，判定组合覆盖一般难以满足。

3.情景

作业、实验中主要有三类，仅供参考

1.给 xyz 的取值范围

就是作业 1，11 题的样式和实验考试的样式。

如：

某程序包含 x，y，z 三个参数，其约束条件如下：

$1 \leq x \leq 6$

$7 \leq y \leq 14$

$20 \leq z \leq 24$

请设计满足如下条件的测试用例：

(1) 分别将 x 划为 3 个有效等价类，y 划为 2 个有效等价类，z 划为 2 个有效等价类

(2) 根据 (1) 的划分设计满足弱一般等价类的测试用例

剩下的略

参考：

$1 \leq x \leq 3$; $3 < x \leq 4$; $4 < x \leq 6$

$7 < y \leq 10$; $10 < y < 14$

$20 < z \leq 22$; $22 < y \leq 24$

$1 \leq x \leq 3 \rightarrow x1$

$3 < x \leq 4 \rightarrow x2$

$4 < x \leq 6 \rightarrow x3$

$7 < y \leq 10 \rightarrow y1$

$10 < y < 14 \rightarrow y2$

$20 < z \leq 22 \rightarrow z1$

$22 < z \leq 24 \rightarrow z2$

(1) 弱一般等价类测试用例个数=各个被测变量中的最大有效等价类个数=3
e.g., $\langle x1, y1, z1 \rangle \langle x2, y2, z2 \rangle \langle x3, y1, z2 \rangle$

2.NextDay

划分等价类

比如这一问题：

测试以下程序：该程序有三个输入变量 month、day、year（month、day 和 year 均为整数值，并且满足： $1 \leq \text{month} \leq 12$ 、 $1 \leq \text{day} \leq 31$ 和 $1900 \leq \text{year} \leq 2050$ ），分别作为输入日期的月份、日、年份，通过程序可以输出该输入日期在日历上隔一天的日期。例如，输入为 2004 年 11 月 30 日，则该程序的输出为 2004 年 12 月 1 日。

输入数据	有效等价类	无效等价类
年	(1) 1900到2050内的闰年整数	(10) year<1900
	(2) 1900到2050内的平年整数	(11) year>2050
月	(3) 1,3,5,7,8,10,12内的整数	(12) 其他输入
		(13) month<1
	(4) 4,6,9,11内的整数	(14) 12<month
	(5) 2	(15) 其他输入
日	(6) 1~28	(16) day<1
	(7) 29	(17) year为闰年 month=2时, 29<day
	(8) 30	(18) year为非闰年 month=2时, 28<day
	(9) 31	(19) month=1,3,5,7,8,10,12时, 31<day
		(20) month=4,6,9,11时, 30<day
		(21) day>31
		(22) 其他输入

按这样划分，就是说有三个“变量”，年、月、日，这样看弱一半等价类数按最多的“日”算，有 4 个；显然强健壮等价类会非常非常多……

3.triangle

比如说给个三角形三边参数 abc，判断 等边/等腰/不构成三角形/不等腰三角形。

划分等价类比如

输入数据	有效等价类
a	$1 \leq a < 4$ $4 \leq a < 8$
b	$1 \leq b < 4$ $4 \leq b < 8$
c	$1 \leq c < 4$ $4 \leq c < 8$

CSDN @萍萍睡不醒

为什么这样划分我也不知道呢

或者按这个

题目：

一个程序读入 3 个整数，把这三个数值看做一个三角形的 3 条边的长度值
这个程序要打印信息，说明这个三角形是不等边的、是等腰的、还是等边的
使用等价类划分法，写出测试用例

题解：

等价类表

编号	功能项	有效等价类	编号	无效等价类	编号
1	三角形类型判断	输入三个大于0的整数，任意两个输入的和大于另外一个输入，且其中两个数相等	1	输入三个大于0的整数，存在两个输入的和小于等于另外一个输入	4
		输入三个大于0的整数，任意两个输入的和大于另外一个输入，且三个数不相等	2	输入三个大于0的小数，存在两个输入的和大于另外一个输入	5
		输入三个大于0的整数，任意两个输入的和大于另外一个输入，且三个数相等	3	三个输入中包含汉字	6
				三个输入中包含空值	7
				三个输入中包含特殊字符	8
				输入项>3	9
				输入项<3	10
				输入项包含负数	11
				输入项包含0	12

CSDN @wz0111

测试用例

测试用例编号	输入数值	所属等价类	预期输出
1	3, 3, 5	1	等腰的
2	3, 4, 5	2	不等边的
3	3, 3, 3	3	等边的
4	3, 1, 1	4	输出错误信息
5	3.1, 3.3, 5.1	5	输出错误信息
6	你好, 3, 3	6	输出错误信息
7	null, 3, 3	7	输出错误信息
8	&, 3, 3	8	输出错误信息
9	3, 3, 3, 3	9	输出错误信息
10	3, 3	10	输出错误信息
11	-3, 3, 3	11	输出错误信息
12	0, 3, 3	12	输出错误信息

不过这种很难进行等价类的组合吧……我觉得 2、3 情景应该不会考毕竟太复杂了。

4.书写方法

主要是设计测试用例，有两种题目/书写方法，一种是作业 1，11 题和 12 题。

如划分有效等价类

划分 x 有效等价类

$1 \leq x \leq 3$; $3 < x \leq 4$; $4 < x \leq 6$

$1 \leq x \leq 3 \rightarrow x1$

$3 < x \leq 4 \rightarrow x2$

$4 < x \leq 6 \rightarrow x3$

yz 略

(1) 弱一般等价类测试用例个数=各个被测变量中的最大有效等价类个数=3
e.g., $\langle x1, y1, z1 \rangle \langle x2, y2, z2 \rangle \langle x3, y1, z2 \rangle$

(2) 强一般等价类测试用例个数=各个被测变量有效等价类数的乘积= $3 \times 2 \times 2 = 12$

$\langle x1, y1, z1 \rangle \langle x1, y1, z2 \rangle$

$\langle x1, y2, z1 \rangle \langle x1, y2, z2 \rangle$

$\langle x2, y1, z1 \rangle \langle x2, y1, z2 \rangle$

$\langle x2, y2, z1 \rangle \langle x2, y2, z2 \rangle$

$\langle x3, y1, z1 \rangle \langle x3, y1, z2 \rangle$

$\langle x3, y2, z1 \rangle \langle x3, y2, z2 \rangle$

边界值划分

(5) 一般边界值测试用例个数（只考虑有效区间）= $4 \times \text{被测变量个数} + 1 = 4 \times 3 + 1 = 13$

对于 x: xmin-, xmin, xmin+, xnor, xmax-, xmax, xmax+
 对于 y: ymin-, ymin, ymin+, ynor, ymax-, ymax, ymax+
 对于 z: zmin-, zmin, zmin+, znor, zmax-, zmax, zmax+
 <xmin, ynor, znor> <xmin+, ynor, znor> <xmax-, ynor, znor> <xmax, ynor, znor>
 <xnor, ymin, znor> <xnor, ymin+, znor> <xnor, ymax-, znor> <xnor, ymax, znor>
 <xnor, ynor, zmin> <xnor, ynor, zmin+> <xnor, ynor, zmax-> <xnor, ynor, zmax>
 <xnor, ynor, znor>

画表格（白盒测试）

(1) 判定覆盖 (程序中每个判断的真值结果都至少出现一次)

x	y	z	(x > y) & (z > 0)	(x < z)
2	1	0	Y	N
1	0	1	N	Y

(2) 条件覆盖 (判定表达式中的每一个条件都要至少取得一次真值和一次假值)

x	y	z	x > y	z > 0	x < z
2	1	1	Y	Y	N
2	3	1	N	Y	N
2	0	1	N	N	Y

(3) 条件判定覆盖 (判定表达式中每个条件的真/假取值都至少出现一次, 并且每个判定表达式自身的真/假取值也都至少出现一次)

x	y	z	x > y	z > 0	判1	判2
2	1	1	Y	Y	N	Y
1	2	2	N	Y	Y	N
2	1	1	Y	Y	N	Y

一种是实验中用的 Junit

用 assertEquals(expected,actual)来设计测试用例

```
public class NestDayTest {
    //弱一般等价类
    @Test//注解,表明这是一个单元测试方法
    Public void testWeakNormalEquivalenceClass(){
        assertEquals("2003-01-31",NextDay.getNextDay("2003-01-30"));
        //这行代码测试了输入日期 "2003-01-30" 的下一天是否是 "2003-01-31"。
        //判定语句是 NextDay.getNextDay(currentDate)
        assertEquals("2003-03-01",NextDay.getNextDay("2003-02-29"));
        assertEquals("2004-02-29",NextDay.getNextDay("2004-02-28"));
        assertEquals("2004-05-01",NextDay.getNextDay("2004-04-31"));
    }
    //强一般等价类
    @Test
    Public void testStrongNormalEquivalenceClass(){
        略
    }
}
```

assertEquals (expected,actual) 最常用方法 (断言), 判断测试执行后的实际取值 actual 与预期取值 expected 是否相等, 相等便通过。

如果测试方法中任一行错了, 该方法判断定为失败; 其他测试方法仍然会执行, 且不受影响。

测试方法里面也可以定义参数、对象

#Triangle

```
public class TriangleTest {  
    Triangle T1 = new Triangle(4, 6, 10);  
    .....  
    @Test  
    public void testIsTriangle() {  
        assertEquals(false, T1.isTriangle(T1));  
        .....  
    }  
}
```

还有的断言

```
assertTrue(NextDay.isValidDate("2024-04-28"));
```

//assertTrue 如果 NextDay.isValidDate("2024-04-28")为 true, 则 success, 反之 Failure

```
assertFalse(NextDay.isValidDate("2024-04-28")); //false 则 success
```

实验考试

现有如下所示的程序段, 需要对其进行相应的测试和调式分析(满分 100 分)

```
public int get_result(int a, int b, int c){  
    int result;  
    if (a >= b || c <= 20)  
        { result = a; }  
    else  
        { result = b; }  
    if ( a >= c && c < a+b)  
        { result = c; }  
    return result;  
}
```

- (1) 设计满足100%代码覆盖率的测试用例集并提供相应的覆盖率截图 (5分)
- (2) 设计满足100%分支 (Branch) 覆盖率的测试用例集并提供相应的覆盖率截图 (10分)
- (3) 对于上述程序段中的两个判定语句, 任选一个设计满足100%MCDC覆盖率的测试用例集并给出详细的设计过程 (15分)
- (4) 对上述代码段运用变异算子手动植入2个不同的bug (在不同代码行同时植入), 并针对植入的2个bug, 设计10个包含failed和successful的测试用例(20分)

- (5)根据Spectrum-based fault localization思想, 自行设计或者套用现有代码疑似值的计算公式并根据one-bug-at-a-time的调试机制, 先做第一轮的错误定位, 生成基于代码行的疑似值排序, 并定位出排名最高的bug代码行, 展示第一轮排序的结果和bug定位结果(25分)
- (6)根据one-bug-at-a-time的调试机制, 修复第一个被定位的bug, 重新运行测试用例集, 开始第二轮的错误定位。生成基于代码行的疑似值排序, 并定位出排名最高的bug代码行, 展示第二轮排序的结果和bug定位结果(25分)

说明: 考试极可能和这个类似。这个答案是 gpt 写的, 看看就行, 主要是提供一个回答格式。

123 就是我们熟悉的哪些常见测试用例方法

4 植入 bug 就是让某两行变成错的, 比如 result = a; 改成 result = b;

这个最好改不太容易被执行的地方, 然后设计测试用例的时候刻意设计几个执行它。

(如果改很容易被执行的行比如第一行定义数的, 不好看出来有问题, 它错了其他全错; 测试用例不执行它搞不好就没错的, 可疑度全是 0 了)

5 就是设计 10 个测试用例, 画出来像下行的东西

(结果处: 失败就是预测结果与实际结果不符, 比如通过代码后, 预期最终的返回值为 1, 结果实际最终的返回值为 2, 这就是失败)

测试用例	结果	可以加上 预期与实际 2 列	行 1	行 2	行 3	行 4
Test1	失败		X	X		
Test2	成功		X		X	
Test3	失败			X		X
Test4	成功		X		X	X
可疑度			0.33	1	0	0.5

可疑度最高的最可能有 bug

6 改了一个 bug 后再仿照 5 过一遍

意思差不多就行, 主要是理解

题目:

现有如下所示的程序段, 需要对其进行相应的测试和调试分析(满分 100 分)

```
public int get_result(int a, int b, int c){
```

```

int result;
if (a >= b || c <= 20)
    { result = a; }
else
    { result = b; }
if ( a >= c && c < a+b)
    { result = c; }
return result;
}

```

(7)设计满足100%代码覆盖率的测试用例集 (5分)

100%代码覆盖率：代码覆盖率有多种标准，gpt 似乎用的行覆盖率，就是测试用例走过了所有行。

(8)设计满足100%分支（Branch）覆盖率的测试用例集 (10分)

(9)对于上述程序段中的两个判定语句，任选一个设计满足100%MCDC覆盖率的测试用例集并给出详细的设计过程 (15分)

(10) 对上述代码段运用变异算子手动植入2个不同的bug（在不同代码行同时植入），并针对植入的2个bug，设计10个包含failed和successful的测试用例(20分)

变异算子：变异算子是用来引入错误或改变代码以生成变异体的方法，通常用于测试代码的鲁棒性和错误检测能力。常见的变异算子包括改变条件、替换操作符、注释掉代码行等。

(11) 根据Spectrum-based fault localization思想，自行设计或者套用现有代码疑似值的计算公式并根据one-bug-at-a-time的调试机制，先做第一轮的错误定位，生成基于代码行的疑似值排序，并定位出排名最高的bug代码行，展示第一轮排序的结果和bug定位结果(25分)

人话：自创疑似值的计算公式并对改了的代码进行疑似值排序，可疑度最高的就是bug。

基于程序频谱的缺陷定位：SBFL(Spectrum-based Fault Localization)：给定一组测试用例，有些用例通过有些用例失败。根据某一代码行是否更多地被通过（或失败）的测试用例所覆盖的基本想法，SBFL 可以给出每一行代码的出错概率。

计算疑似值（可疑值）：

在 SBFL 中，cf、cs、us 和 uf 分别代表：

- cf：通过的测试用例覆盖该代码行的数量。
- cs：失败的测试用例覆盖该代码行的数量。

- us: 通过的测试用例未覆盖该代码行的数量。
- uf: 失败的测试用例未覆盖该代码行的数量。

通过这些数据，可以计算出每行代码的疑似值（出错概率），常用的公式有 Tarantula、Ochiai 等。例如 Tarantula 公式是：

$$[可疑度 = \frac{\frac{cf(s)}{cf(s)+uf(s)}}{\frac{cs(s)}{cs(s)+us(s)} + \frac{cf(s)}{cf(s)+uf(s)}}]$$

$$suspiciousness(e) = 1 - hue(e) = \frac{\frac{failed(e)}{totalfailed}}{\frac{passed(e)}{totalpassed} + \frac{failed(e)}{totalfailed}}$$

第二个公式的 e 表示行

这两个公式一个意思，理解意思就行，就是：

测试失败/总测试失败 比 （测试成功/总测试成功）+ （测试失败/总测试失败）
举个例子就是这样

测试用例	结果	行1	行2	行3	行4
Test1	失败	X	X		
Test2	成功	X		X	
Test3	失败		X		X
Test4	成功	X		X	X

在这个例子中：

- 行1被Test1、Test2、Test4覆盖
- 行2被Test1、Test3覆盖
- 行3被Test2、Test4覆盖
- 行4被Test3、Test4覆盖

接下来，我们可以计算每一行的可疑度：

1. 行1

- 失败的测试用例数量（包含行1）：1（Test1）
- 成功的测试用例数量（包含行1）：2（Test2, Test4）
- 总失败的测试用例数量：2（Test1, Test3）
- 总成功的测试用例数量：2（Test2, Test4）

$$[\text{Suspiciousness}(\text{行 1}) = \frac{\frac{1}{2}}{\frac{1}{2} + \frac{2}{2}} = \frac{0.5}{0.5 + 1} = \frac{0.5}{1.5} = \frac{1}{3} \approx 0.333]$$

2. 行2

- 失败的测试用例数量（包含行2）：2（Test1, Test3）
- 成功的测试用例数量（包含行2）：0
- 总失败的测试用例数量：2（Test1, Test3）
- 总成功的测试用例数量：2（Test2, Test4）

```
[
\text{Suspiciousness}(\text{行2}) = \frac{\frac{2}{2}}{\frac{2}{2} + \frac{0}{2}} = \frac{1}{1 + 0} = 1
]
```

$$[\text{Suspiciousness}(\text{行 2}) = \frac{\frac{2}{2}}{\frac{2}{2} + \frac{0}{2}} = \frac{1}{1 + 0} = 1]$$

最终

行号	可疑度
行1	0.333
行2	1.0
行3	0.0
行4	0.5

3、4 行同理。行 2 可疑度最大，它是 bug 的概率最大。

one-bug-at-a-time：一种调试策略，指在解决问题时一次只专注于一个bug。

错误定位：错误定位是指在程序找出导致故障的具体代码行或代码片段的过程。常用的方法有基于代码覆盖的技术（如SBFL），动态分析和静态分析等。

植入bug：指的是在代码中有意地加入若干错误，以模拟实际开发过程中可能存在的问题，用于测试和验证错误定位方法的有效性。

(12) 根据one-bug-at-a-time的调试机制，修复第一个被定位的bug，重新运行测试用例集，开始第二轮的错误定位。生成基于代码行的疑似值排序，并定位出排名最高的bug代码行，展示第二轮排序的结果和bug定位结果(25分)

人话：改了1个bug后再按5的流程走一遍。

gpt-4o 回答如下，看看就行，主要是了解一下格式：

(1) 设计满足 100%代码覆盖率的测试用例集 (5 分)

为了实现 100%的代码覆盖率，我们需要确保每条语句都被执行至少一次。下面是一个测试用例集：

```
java
@Test
public void testGetResult() {
    assertEquals(10, get_result(10, 5, 15)); // a >= b, c > 20
    assertEquals(5, get_result(5, 10, 25)); // a < b, c > 20
    assertEquals(10, get_result(10, 5, 20)); // a >= b, c <= 20
    assertEquals(20, get_result(5, 10, 20)); // a < b, c <= 20
}
```

(2) 设计满足 100%分支 (Branch) 覆盖率的测试用例集 (10 分)

为了实现 100%的分支覆盖率，我们需要确保每个条件分支都被执行至少一次。以下是测试用例：

Test Case	a	b	c	Expected Result	Branches Covered
TC1	10	5	15	10	if (a >= b
TC2	5	10	25	10	if (a >= b
TC3	10	5	20	10	if (a >= b
TC4	5	10	20	20	if (a < b && c <= 20), if (a >= c)

(3) 设计满足 100%MCDC 覆盖率的测试用例集并给出详细的设计过程 (15 分)

我们选择第一个判定语句 if (a >= b || c <= 20) 来设计测试用例集。

Test Case	a	b	c	Condition a >= b	Condition c <= 20	Decision (a >= b c <= 20)	Expected Result
TC1	10	5	30	T	F	T	10
TC2	5	10	15	F	T	T	5
TC3	5	10	25	F	F	F	10
TC4	10	10	25	T	F	T	10

(4) 植入 2 个不同的 bug 并设计测试用例集 (20 分)

Bug 1: 在 if (a >= c && c < a + b) 语句中，将 && 改为 ||。 **Bug 2:** 在 if (a >= b || c <= 20) 语句中，将 <= 改为 <。

修改后的代码段:

```
java
public int get_result(int a, int b, int c) {
    int result;
    if (a >= b || c < 20) // Bug 2
        { result = a; }
    else
        { result = b; }
    if (a >= c || c < a + b) // Bug 1
        { result = c; }
    return result;
}
```

测试用例集:

Test Case	a	b	c	Expected Result	Actual Result	Pass/Fail
TC1	10	5	15	10	10	Pass
TC2	5	10	25	10	25	Fail
TC3	10	5	20	10	10	Pass
TC4	5	10	20	20	20	Pass
TC5	15	10	5	15	5	Fail
TC6	8	12	30	12	30	Fail
TC7	10	5	15	10	10	Pass
TC8	5	15	10	5	10	Fail
TC9	10	5	25	10	25	Fail
TC10	5	15	5	5	5	Pass

(5) 第一轮错误定位和排序结果 (25 分)

根据 Spectrum-based fault localization 的思想, 我们可以计算怀疑值。我们使用 Tarantula 公式计算怀疑值:

$$[Suspiciousness = \frac{\frac{failed_{executions}}{total_{failed}}}{\frac{failed_{executions}}{total_{failed}} + \frac{passed_{executions}}{total_{passed}}}]$$

我们针对上述代码段计算每行代码的怀疑值:

Line of Code	Passed Executions	Failed Executions	Suspiciousness Score
int result;	10	5	0.333
`if (a >= b	注：这里 gpt 应该输出错行了	c < 20)`	7
{ result = a; }	5	2	0.285
else { result = b; }	3	3	0.5
`if (a >= c		c < a + b)`	5
{ result = c; }	3	3	0.5
return result;	10	5	0.333

排序结果显示，行 `else { result = b; }`，`if (a >= c || c < a + b)` 和 `{ result = c; }` 具有最高的怀疑值。

第一轮定位出怀疑度最高的 bug 行：`if (a >= c || c < a + b)`。

(6) 第二轮错误定位和排序结果 (25 分)

修复第一个被定位的 bug，改回 `&&`：

java

```
public int get_result(int a, int b, int c) {
    int result;
    if (a >= b || c < 20) // Bug 2 remains
        { result = a; }
    else
        { result = b; }
    if (a >= c && c < a + b) // Bug 1 fixed
        { result = c; }
    return result;
}
```

重新运行测试用例集，并计算怀疑值：

Line of Code	Passed Executions	Failed Executions	Suspiciousness Score
int result;	10	1	0.090
`if (a >= b		c < 20)`	7

{ result = a; }	5	1	0.166
else { result = b; }	3	0	0
if (a >= c && c < a + b)	5	0	0
{ result = c; }	3	0	0
return result;	10	1	0.090

第二轮定位出怀疑度最高的 bug 行：if (a >= b || c < 20)。

修复第二个被定位的 bug，改回 <=:

java

```
public int get_result(int a, int b, int c) {
    int result;
    if (a >= b || c <= 20) // Bug 2 fixed
        { result = a; }
    else
        { result = b; }
    if (a >= c && c < a + b) // Bug 1 fixed
        { result = c; }
    return result;
}
```

现在，所有测试用例应该通过，并且代码中的 bug 已经修复。（可疑度应该都是 0 了）

软件质量保证与测试-作业 1

1.简述软件测试的意义。(5 分)

随着计算机技术的迅速发展和广泛深入的应用，软件质量问题已成为开发和使用软件人员关注的焦点。而由于软件本身的特性，软件中的错误是不开避免的。不断改进的开发技术和工具只能减少错误的发生，但是却不可能完全避免错误。因此为了保证软件质量，必须对软件进行测试。软件测试是软件开发中必不可少的环节，是最有效的排除和防治软件缺陷的手段，是保证软件质量、提高软件可靠性的最重要手段。

2.什么是软件缺陷？它的表现形式有哪些？（5 分）

从产品内部看，软件缺陷是软件产品开发或维护过程中所存在的错误、毛病等各种问题；

从外部看，软件缺陷是系统所需实现的某种功能的失效或违背。它的表现形式主要有以下几种：

- (1) 软件未达到产品说明书中已经标明的功能；
- (2) 软件出现了产品说明书中指明不会出现的错误；
- (3) 软件未达到产品说明书中虽未指出但应当达到的目标；
- (4) 软件功能超出了产品说明书中指出的范围；
- (5) 软件测试人员认为软件难以理解、不易使用，或者最终用户认为该软件使用效果不良。

3.“软件测试能够保证软件的质量”这句话对吗？软件测试和软件质量之间是什么关系？（5 分）

不对。软件测试是保障软件质量的手段之一，但不是唯一手段。测试是产品高质量的必要非充分条件，软件测试不能决定软件质量。

4.分别解释什么是静态测试、动态测试、黑盒测试、白盒测试、人工测试和自动化测试。（5 分）

所谓静态测试是指不运行被测软件，仅通过分析或检查等其他手段达到检测的目的。

所谓动态测试是指通过运行被测软件，检查运行结果与预期结果的差异，并分析运行效率和健壮性等性能。

黑盒测试是指在对程序进行的功能抽象的基础上，将程序划分成功能单元，然后对每个功能单元生成测试数据进行测试。用这种方法进行测试时，被测程序被当作打不开的黑盒，因而无法了解其内部构造，因此又称为功能测试。

白盒测试又称为结构测试或逻辑驱动测试，它是知道产品内部工作过程，可通过测试来检测产品内部动作是否按照规格说明书的规定正常进行，按照程序内部的结构测试程序，检验程序中的每条通路是否都有能按预定要求正确工作，而不顾它的功能。

广义上，人工测试是人为测试和手工测试的统称。人为测试的主要方法有桌前检查，代码审查和走查。手工测试指的是在测试过程中，按测试计划一步一步执行程序，得出测

试结果并进行分析的测试行为。

自动化测试指的是利用测试工具来执行测试，并进行测试结果分析的测试行为。

5.在单元测试中，所谓单元是如何划分的？单元测试的主要任务是什么？（5分）

单元测试的对象通常是软件设计的最小逻辑单元，单元的划分在面向过程的结构化程序中一般是函数或子过程，在面向对象的程序中可以是类或类的成员函数。

单元测试的主要任务是：模块接口测试；局部数据结构测试；路径测试；错误处理测试；边界测试。

6.如果开发时间紧迫，是否可以跳过单元测试而直接进行集成测试？为什么？（5分）

不可以。因为没有经过单元测试的模块会遗留大量的缺陷到集成测试阶段，而在集成测试阶段对这些缺陷定位困难，导致后续工作展开困难，修复缺陷成本成指数级增长。

7.什么是回归测试？什么时候进行回归测试？（5分）

回归测试就是重新运行现有测试用例测试原有功能，以便确定变更是否达到了预期的目的，检查变更是否损害了原有的正常功能。每当软件发生变化时就应进行回归测试。

8.集成测试有哪些不同的集成方法？简述不同方法的特点。（5分）

集成测试通常有一次性集成、自顶向下集成、自底向上集成和混合集成 4 种集成方法。

一次性集成方法需要的测试用例数目少，测试方法简单、易行。但是由于不可避免存在模块间接口、全局数据结构等方面的问题，所以一次运行成功的可能性不大；如果一次集成的模块数量多，集成测试后可能会出现大量的错误，给程序的错误定位与修改带来很大的麻烦；即使集成测试通过，也会遗漏很多错误进入系统测试。

自顶向下集成在测试的过程中，可以较早地验证主要的控制和判断点；一般不需要驱动程序，减少了测试驱动程序开发和维护的费用；可以和开发设计工作一起并行执行集成测试，能够灵活的适应目标环境；容易进行故障隔离和错误定位。但是在测试时需要为每个模块的下层模块提供桩模块，桩模块的开发和维护费用大；桩模块不能反映真实情况，重要数据不能及时回送到上层模块，导致测试不充分；涉及复杂算法和真正 I/O 的底层模块最易出问题，在后期才遇到导致过多的回归测试。

自底向上集成可以尽早的验证底层模块的行为；提高了测试效率；一般不需要桩模块；容易对错误进行定位。但是直到最后一个模块加进去之后才能看到整个系统的框架；驱动模块的设计工作量大；不能及时发现高层模块设计上的错误。

混合集成具有自顶向下和自底向上两种集成策略的优点，但是在被集成之前，中间层不能尽早得到充分的测试。

9.分析比较面向对象的软件测试与传统的软件测试的异同。（5分）

传统的单元测试的对象是软件设计的最小单位——模块。当考虑面向对象软件时，

单元的概念发生了变化,此时最小的可测试单位是封装的类或对象,而不再是个体的模块。传统单元测试主要关注模块的算法实现和模块接口间数据的传递,而面向对象的单元测试主要考察封装在一个类中的方法和类的状态行为。

面向对象软件没有层次的控制结构,因此传统的自顶向下和自底向上集成策略就不再适合,它主要有以下两种集成策略: 基于类间协作关系的横向测试;基于类间继承关系的纵向测试。

系统测试一般不考虑内部结构和中间结果,因此面向对象软件系统测试与传统的系统测试差别不大。

面向对象软件测试的整体目标和传统软件测试的目标是一致的,即以最小的工作量发现尽可能多的错误,但是面向对象测试的策略和战术有很大不同。测试的视角扩大到包括复审分析和设计模型,此外,测试的焦点从过程构件(模块)移向了类。

10. (10 分) 有程序段如下:

```
public int get_max(int Age1, int Age2,int Age3){  
int max;  
if(Age1 >= Age2)  
    { max = Age1; }  
else  
    { max = Age2; }  
if( Age3 > Age1 )  
    { max = Age3; }  
return max;  
}
```

(1)试分析该程序段有何逻辑错误;

(2)设计 1 个测试数据,使得执行该测试时会执行到 **fault** 但不会触发 **error**;

(3)设计 1 个测试数据,使得执行该测试时会执行到 **fault** 并触发 **error**,但不会引起 **Failure**;

(4)设计 1 个测试数据,使得执行该测试时会执行到 **fault**,触发 **error**,并引起 **Failure** (**feiljør**)。

(1) $\text{if}(\text{Age3} > \text{Age1}) \Rightarrow \text{if}(\text{Age3} > \text{max})$

(2) $\text{Age1}=30, \text{Age2}=20, \text{Age3}=10$ ($\text{Age3} \leq \text{Age1}$ 即可);

(3) $\text{Age1}=20, \text{Age2}=10, \text{Age3}=30$ (Age3 最大即可);

(4) $\text{Age1}=10, \text{Age2}=30, \text{Age3}=20$ ($\text{Age1} < \text{Age3} < \text{Age2}$ 即可)。

11. (15 分) 某程序包含 x, y, z 三个参数,其约束条件如下:

$1 \leq x \leq 6$

$7 \leq y \leq 14$

$20 \leq z \leq 24$

请设计满足如下条件的测试用例:

(1) 分别将 x 划为 3 个有效等价类, y 划为 2 个有效等价类, z 划为 2 个有效等价类

(2) 根据 (1) 的划分设计满足弱一般等价类的测试用例

- (3) 根据 (1) 的划分设计满足强一般等价类的测试用例
- (4) 根据 (1) 的划分设计满足弱健壮等价类的测试用例
- (5) 根据 (1) 的划分设计满足强健壮等价类的测试用例
- (6) 设计满足一般边界值的测试用例
- (7) 设计满足一般最坏情况边界值的测试用例
- (8) 设计满足健壮边界值的测试用例
- (9) 设计满足健壮最坏情况边界值的测试用例

参考:

$1 \leq x \leq 3$; $3 < x \leq 4$; $4 < x \leq 6$

$7 < y \leq 10$; $10 < y < 14$

$20 < z \leq 22$; $22 < y \leq 24$

$1 \leq x \leq 3 \rightarrow x1$

$3 < x \leq 4 \rightarrow x2$

$4 < x \leq 6 \rightarrow x3$

$7 < y \leq 10 \rightarrow y1$

$10 < y < 14 \rightarrow y2$

$20 < z \leq 22 \rightarrow z1$

$22 < y \leq 24 \rightarrow z2$

(1) 弱一般等价类测试用例个数=各个被测变量中的最大有效等价类个数=3
e.g., $\langle x1, y1, z1 \rangle$ $\langle x2, y2, z2 \rangle$ $\langle x3, y1, z2 \rangle$

(2) 强一般等价类测试用例个数=各个被测变量有效等价类数的乘积= $3 \times 2 \times 2 = 12$

$\langle x1, y1, z1 \rangle$ $\langle x1, y1, z2 \rangle$

$\langle x1, y2, z1 \rangle$ $\langle x1, y2, z2 \rangle$

$\langle x2, y1, z1 \rangle$ $\langle x2, y1, z2 \rangle$

$\langle x2, y2, z1 \rangle$ $\langle x2, y2, z2 \rangle$

$\langle x3, y1, z1 \rangle$ $\langle x3, y1, z2 \rangle$

$\langle x3, y2, z1 \rangle$ $\langle x3, y2, z2 \rangle$

(3) 弱健壮等价类测试用例个数=各个被测变量中的最大有效等价类个数+各个被测变量无效等价类个数之和= $3+2+2+2=9$; 相当于弱一般等价类+各个被测变量的无效等价类个数之和

弱一般 3 个 $\langle x1, y1, z1 \rangle$ $\langle x2, y2, z2 \rangle$ $\langle x3, y1, z2 \rangle$

变量 x 无效 2 个 $\langle -x1, y1, z1 \rangle$, $\langle x3+, y1, z1 \rangle$

变量 y 无效 2 个 $\langle -y1, y1, z1 \rangle$, $\langle y2+, y1, z1 \rangle$

变量 z 无效 2 个 $\langle -z1, y1, z1 \rangle$, $\langle z2+, y1, z1 \rangle$

(4) 强健壮等价类测试用例个数=各个被测变量的等价类总数 (有效+无效) 的乘积=

$$(3+2) \times (2+2) \times (2+2) = 5 \times 4 \times 4 = 80$$

$$\begin{bmatrix} x1- \\ x1 \\ x2 \\ x3 \\ x3+ \end{bmatrix} \times \begin{bmatrix} y1- \\ y1 \\ y2 \\ y2+ \end{bmatrix} \times \begin{bmatrix} z1- \\ z1 \\ z2 \\ z2+ \end{bmatrix}$$

对于 x: xmin-, xmin, xmin+, xnor, xmax-, xmax, xmax+

对于 y: ymin-, ymin, ymin+, ynor, ymax-, ymax, ymax+

对于 z: zmin-, zmin, zmin+, znor, zmax-, zmax, zmax+

(5) 一般边界值测试用例个数（只考虑有效区间）=4×被测变量个数+1=4×3+1=13

<xmin, ynor, znor> <xmin+, ynor, znor> <xmax-, ynor, znor> <xmax, ynor, znor>
 <xnor, ymin, znor> <xnor, ymin+, znor> <xnor, ymax-, znor> <xnor, ymax, znor>
 <xnor, ynor, zmin> <xnor, ynor, zmin+> <xnor, ynor, zmax-> <xnor, ynor, zmax>
 <xnor, ynor, znor>

(6) 一般最坏情况边界值测试用例个数=5×5×5=125

$$\begin{bmatrix} x \text{ min} \\ x \text{ min}+ \\ xnor \\ x \text{ max}- \\ x \text{ max} \end{bmatrix} \times \begin{bmatrix} y \text{ min} \\ y \text{ min}+ \\ ynor \\ y \text{ max}- \\ y \text{ max} \end{bmatrix} \times \begin{bmatrix} z \text{ min} \\ z \text{ min}+ \\ znor \\ z \text{ max}- \\ z \text{ max} \end{bmatrix}$$

(7) 健壮边界值测试用例个数（同时考虑有效和无效区间）=6×被测变量个数+1=6×3+1=19

<xmin-, ynor, znor> <xmin, ynor, znor> <xmin+, ynor, znor> <xmax-, ynor, znor> <xmax, ynor, znor> <xmax+, ynor, znor>

<xnor, ymin-, znor> <xnor, ymin, znor> <xnor, ymin+, znor> <xnor, ymax-, znor> <xnor, ymax, znor> <xnor, ymax+, znor>

<xnor, ynor, zmin-> <xnor, ynor, zmin> <xnor, ynor, zmin+> <xnor, ynor, zmax-> <xnor, ynor, zmax> <xnor, ynor, zmax+>

<xnor, ynor, znor>

(8) 健壮最坏情况边界值测试用例测试用例个数=7×7×7=343

$x_{\min-}$	$y_{\min-}$	$z_{\min-}$
x_{\min}	y_{\min}	z_{\min}
$x_{\min+}$	$y_{\min+}$	$z_{\min+}$
x_{nor}	y_{nor}	z_{nor}
$x_{\max-}$	$y_{\max-}$	$z_{\max-}$
x_{\max}	y_{\max}	z_{\max}
$x_{\max+}$	$y_{\max+}$	$z_{\max+}$

12. (15 分) 对于两个判定表达式: 1、 $(x>y) \&\& (z>0)$ 和 2、 $(x<z)$ 请生成满足下列覆盖的测试用例集:

(1) 判定覆盖

(1) 判定覆盖: (程序中每个判断的真值均出现一次).

x, y, z	$(x>y) \&\& (z>0)$	$(x<z)$	
3 2 1	Y	N	$\Rightarrow x>y, x>z$
-1 0 1	N	Y	$\Rightarrow z<0$ 或 $x<y$, $z>x$

(2) 条件覆盖

(2) 条件覆盖: (判定表达式中的每个条件都取到一次真值一次假值).

x, y, z	$x>y$	$z>0$	$x<z$	
2 1 -1	Y	N	N	$\Rightarrow x>y, z<0, x>z$
2 3 1	N	Y	N	$\Rightarrow x<y, z>0, x>z$
-2 0 -1	N	N	Y	$\Rightarrow x<y, z<0, x<z$

(3) 条件判定覆盖

3. 条件判定覆盖: (判定表达式中每个条件的真/假值均出现一次, 且每个判断表达式的自身, 至少取到真/假一次)

x, y, z	$(x>y)$	$(z>0)$	$x<z$	$(x>y) \&\& (z>0)$	$x<z$
2 1 -1	Y	N	N	N	N
2 0 1	Y	Y	N	Y	N
-2 0 -1	N	N	Y	N	Y

(4) 修改的条件判定覆盖

(4). 修改的条件覆盖 (满足(3), 且每个条件都显式地影响判定结果).

x	y	z	(x>y)	(z>0)	x<z	(x>y) && (z>0)	x<z
1	0	2	Y	Y	Y	Y	Y
2	3	1	N	Y	N	N	N
1	0	-1	Y	N	N	N	N

(5) 条件组合覆盖

(5) 条件组合覆盖: 每个判定条件取值的各种组合都至少出现一次

x	y	z	(x>y)	(z>0)	x<z	(x>y) && (z>0)	x<z
-2	0	-1	N	N	Y	N	Y
2	3	1	N	Y	N	N	N
0	1	-1	Y	N	Y	N	Y
2	0	1	Y	Y	N	Y	N

(6) 判定组合覆盖

(6). 判定组合覆盖 (所有判定间的各种组合都至少出现一次).

x	y	z	(x>y)	(z>0)	(x<z)	(x>y) && (z>0)	x<z
0	1	-1	N	N	N	N	N
-2	0	-1	N	N	Y	N	Y
2	0	1	Y	Y	N	Y	N
2	1	3	Y	Y	Y	Y	Y

13. (15 分) 有程序模块 My_program 代码如下:

```

1 public int My_program (int num, int cycle, boolean flag)
2 {
3   int ret = 0;
4   while( cycle > 0 )
5   {
6     if( flag == true )
7     {
8       ret = num - 10;
9       break;
10    }
11    else
12    {
13      if( num%2 == 0 )

```

```

14    {
15    ret = ret * 10;
16    }
17    else
18    {
19    ret = ret + 1;
20    }
21    }
22    cycle--;
23 }
24 return ret;
25 }

```

请为程序模块 **My_program**

(1) 画出程序控制流图，计算控制流图的环路复杂度

(2) 导出基本路径

(3) 设计测试用例实现基本路径覆盖

(1) 环路复杂度 $V(G)=4$

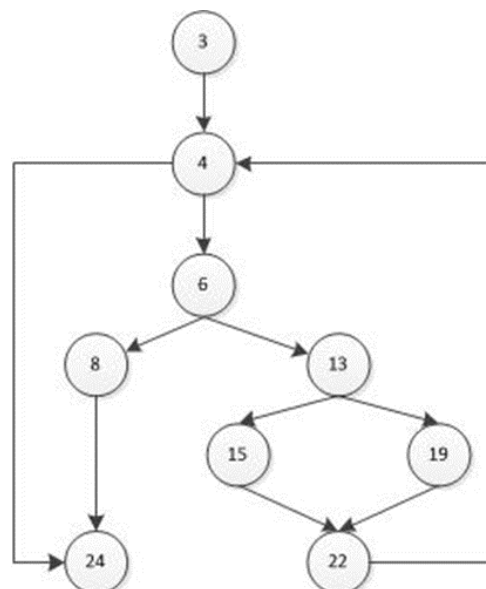
(2) 导出基本路径

A. 3-4-24

B. 3-4-6-8-24

C. 3-4-6-13-15-22-4-24

D. 3-4-6-13-19-22-4-24



(3) 设计基本路径覆盖测试用例

测试用例(num, cycle, 覆盖基本
flag) 路径

(10,0, true) A

(100, 10,true) B

(10,1,false) C

(5,1,false) D

软件质量保证与测试-作业 2

1.用种子模型法估算软件中存在错误数的困难在哪里？请举例说明。（10 分）

- (1) 人为植入错误较为困难
- (2) 错误被发现的难易程度不一样，被插入的错误并不一定能代表各种可能的错误，估算结果不一定准确。

例子：在开始测试前向被测软件人为插入 10 个 bug，均是已知类型的常见错误。经过一段时间测试以后这 10 个错误都被发现了，这样一来根据种子模型估算方法，软件中隐藏的错误的估算为 0，显然是不合理的。

2.两个小组独立地测试同一个程序，第一组发现 35 个错误，第二组发现 23 个错误，在两个小组发现的错误中有 16 个是共同的。（20 分）

- (1) 根据 Hyman 估算法可以估计程序中的错误总数是多少个？
- (2) 运用 Hyman 估算法时需要注意什么问题？

- (1) $35 \times 23 \div 16 \approx 50$
- (2) 当两组测试人员具有较高相关性时，Hyman 估算方法会有较大误差。

3.软件产品质量特性是什么？什么是软件质量保证？其主要任务是什么？（20 分）

软件产品质量特性指的是软件产品满足使用要求的程度，就是高质量的软件系统能够准时交付给用户，所耗费的成本不超出预算，且能够正常地运行。

软件质量保证（Software Quality Assurance, SQA）就是保证软件质量的一系列系统性的活动，它是建立一套有计划，有系统的方法，向管理层保证拟定出的标准、步骤、实践和方法能够正确地被所有项目所采用。

软件质量保证的主要任务是向所有有关的人提供证据以确立软件产品的质量功能按照需求运行，它不负责高质量软件的编制，其主要职责是审计软件经理和软件工程组的质量活动，并鉴别活动中出现的偏差。

4.质量保证部门与测试部门的职责是否一样？归纳它们的共同点和不同点。（20 分）

不一样。

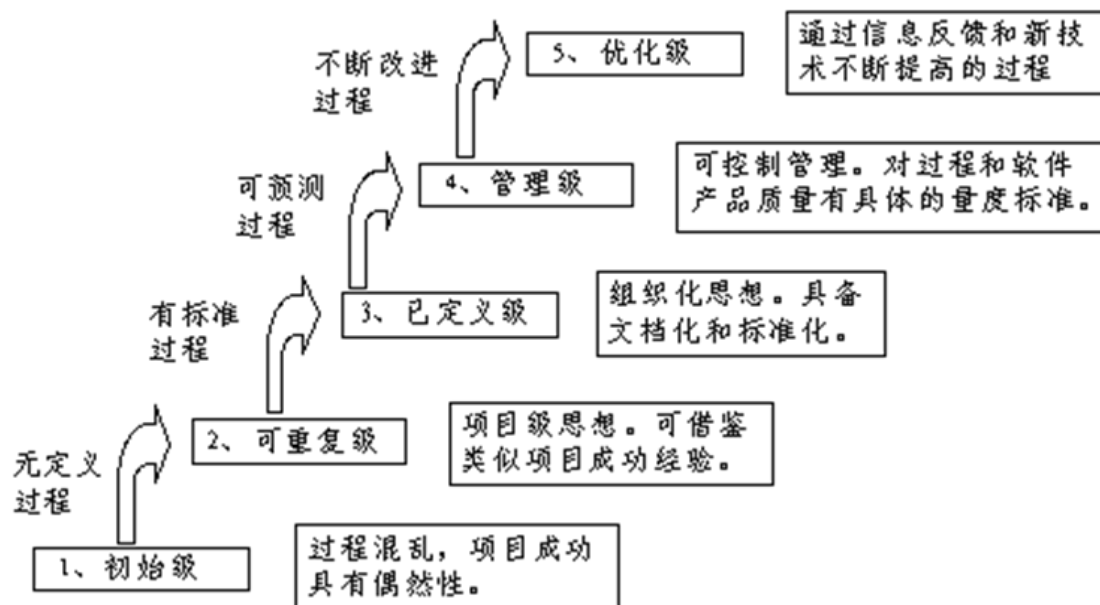
从**共同点**的角度看，软件测试部门和软件质量保证部门的目的是尽力确保软件产品满足需求，从而开发出高质量的软件产品。两个流程都贯穿在整个软件开发生命周期中。

二者的**不同**之处在于软件质量保证部门的工作侧重对软件开发流程中的各个过程进行管理与控制，杜绝软件缺陷的产生。而测试部门工作则是对已产生的软件缺陷进行修复。

5.简述测试计划工作的目的是什么？测试计划工作的内容都包括什么？测试文档的含

义和内容是什么？（20 分）

CMM 将软件过程能力成熟度划分为 5 个等级，如下图所示。每个等级定义一组过程能力目标，并且表述了能够达到这些过程能力目标的实践活动。



考后整理部分

1、软件测试的方法

从是否需要执行程序的角度来区分：静态测试，动态测试

从是否需要知道程序的内部结构来区分：黑盒测试，白盒测试

从测试执行者来区分：手工测试，自动化测试

黑盒测试：又称功能测试、数据驱动测试或基于规格说明的测试。测试者只知道该程序输入和输出之间的关系或程序的功能。

白盒测试：又称结构测试、逻辑驱动测试或基于程序的测试。白盒测试依赖于对程序内部结构的分析，针对特定条件设计测试用例，对软件的逻辑路径进行测试。

2、测试过程

软件测试项目可按照测试需求分析、测试计划、测试设计、测试开发、测试执行、测试总结这样的环节来组织实施。

以下内容编辑于 2024/7/4

原题：

简单题：缺陷产生原因，看课本

第一个计算题：给一个代码，然后题目与（1）~（4）一模一样

```
public int get_max(int Age1, int Age2, int Age3){  
    int max;  
    if(Age1 >= Age2)  
        { max = Age1; }  
    else  
        { max = Age2; }  
    if(Age3 > Age1)  
        { max = Age3; }  
    return max;  
}
```

(1)试分析该程序段有何逻辑错误；

(2)设计 1 个测试数据，使得执行该测试时会执行到 fault 但不会触发 error；

(3)设计 1 个测试数据，使得执行该测试时会执行到 fault 并触发 error，但不会引起 Failure；

(4)设计 1 个测试数据，使得执行该测试时会执行到 fault，触发 error，并引起 Failure (feiljör)。

第二个计算题：

黑盒测试白盒测试，等价类边缘测试之类的，作业题类似但需给出具体的测试用例

第三个计算题：

对下面的代码植入一个 bug，然后求 CF，CS，UF，US，用 $Susp = (CF + US) / (CS + UF)$ 计算疑似值

```
public int get_max(int Age1, int Age2, int Age3){  
    int max;  
    if(Age1 >= Age2)  
        { max = Age1; }  
    else  
        { max = Age2; }  
    if(Age3 > Age1)  
        { max = Age3; }  
    return max;  
}
```

论述题：有三个相近的词语，解释区别，有一个是代码测试和代码调试的区别