软件测试最终押题

L.C.

一、名词解释

1. **软件缺陷**（software Bug）：任何程序、系统中的问题，和产品设计书的不一致性，不能满足用户的需求。

   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要实现的某种功能的失效或违背。

2. **系统测试**（特征测试）：检验系统所有元素之间协作是否合适，整个系统的性能和功能是否达到要求。其测试内容包括：功能测试，非功能测试与回归测试等。

3. **软件本地化**：将一个软件产品按特定国家/地区或语言市场 的需要进行加工，使之满足特定市场上的用户 对语言和文化的特殊要求的软件生产活动。包括功能性测试，翻译测试， 可用性测试， 兼容性调试， 文化、宗教、喜好等适用性测试，手册验证。

4. **单元测试**：单元测试是对软件基本组成单元（如函数、类的方法等）进行的测试。

5. **TMAP**：是一种结构化的、基于风险策略的测试方法体系, 目的能更早地发现缺陷，以最小 的成本、有效地、彻底地完成测试任务，以减少软件发布后的支持成本。

6. **自动化测试**：相对手工测试而存在的一个概念，由手工逐个地运行测试用例的操作过程被测试工具自动执行的过程所代替。
   “一切可以由计算机系统自动完成的测试任务都已经由计算机系统或软件工具、程序来承担并自动执行”

7. **I18N（软件国际化）**：是通过功能设计和代码实现中软件系统有能力处理多种语言和不同文化，使创建不同语言版本时，不需要重新编写代码的软件工程方法。

二、简答

1. **简述桩程序和驱动程序**

   驱动模块: 代替上级模块传递测试用例的程序
   桩模块：代替下级模块的仿真程序

2. **软件测试和SQA的关系**

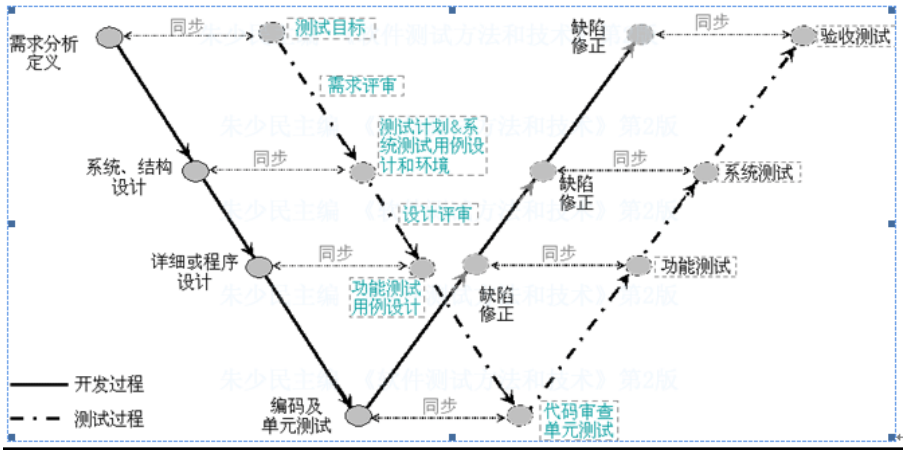
   SQA(软件质量保证)活动是通过对软件产品有计划地进行评审和审计来验证软件是否合乎标准的系统工程，通过协调、审查和跟踪以获取有用信息，形成分析结果以指导软件过程。
   a）SQA和软件测试之间相辅相成，既有包含又有交叉的关系。
   b）SQA指导、监督软件测试的计划和执行，督促测试工作的结果客观、准确和有效，并协助测试流程的改进。
   c）软件测试是SQA重要手段之一，为SQA提供所需的数据，作为质量评价的客观依据。
   d）相同点：都是贯穿整个软件开发生命周期的流程。
   e）不同点：SQA是一项管理工作，侧重于对流程的评审和监控。测试是一项技术性的工作，侧重对产品进行评估和验证。

3. **比较验证和有效性确认**

   软件测试是由“验证（Verification）”和“有效性确认（Validation）”活动构成的整体
   “验证”是检验软件是否已正确地实现了产品规格书所定义的系统功能和特性。
   “有效性确认”是确认所开发的软件是否满足用户真正需求的活动。

4. **画出W模型，解释开发和测试的关系**

   增加了软件开发阶段中应同步进行的验证和确认活动。
   测试过程和开发过程是同时开始，同时结束的，两者保持同步的关系。测试过程是对开发过程中阶段性成果和最终的产品进行验证的过程，所以两者相互依赖。前期，测试过程更多依赖开发过程，后期开发过程依赖于测试过程。



5. **ST、ET的优缺点比较**

   脚本测试（ST）使用手工测试的测试用例（Test case）和自动化的测试脚本（Test script）；先设计后执行（过程：分析->设计->执行->报告）；阶段性明显，属于较传统的测试方式。

   探索性测试（ET） 强调测试学习，设计和执行同时展开；没有测试用例，一边想（在头脑中设计）一边测试；缺乏良好的系统性、复用性。

6. **简述比较集成测试的不同模式、简述集成测试的方法**

   集成测试是将软件集成起来，对模块之间的接口进行测试。顾名思义，集成测试是将软件集成起来后进行测试。集成测试又叫子系统测试、组装测试、部件测试等。
   模块内的集成，主要是测试模块内各个接口间的交互集成关系；子系统内的集成，测试子系统内各个模块间的交互关系；系统内的集成，测试系统内各个子系统和模块间的集成关系。

   非渐增式测试模式：先分别测试每个模块，再把所有模块按设计要求放在一起结合成所要的程序，如大棒模式。
   渐增式测试模式：把下一个要测试的模块同已经测试好的模块结合进来进行测试，测试完后再把下一个应该测试的模块结合起来测试。渐增式测试又可以根据每次添加模块的路线分为自顶向下测试、自底向上测试和混合测试等方式。
   集成测试的测试依据：概要设计书，详细设计说明书，主要是概要设计说明书。

   大爆炸集成、自底向上集成、自顶向下集成和三明治集成

7. **比较4种导向**

   1)以功能验证为导向，测试是证明软件是正确的（正向思维）。
   2)以破坏性检测为导向，测试是为了找到软件中的错误（逆向思维）。
   3)以质量评估为导向，测试是提供产品的评估和质量度量。
   4)以缺陷预防为导向，测试是为了展示软件符合设计要求，发现缺陷、预防缺陷。

三、综合题

1. **等价类、判定表、因果图**

某单位财务管理系统中计算出差补助的方法是：
当员工办理长期出差时，不论是否出差，出差到哪里，每月固定补助1000；
当员工未办理长期出差时，如果出差省会城市，每月补助1500，非省会城市每月补助800，其他情况为0；
试用判定表法设计测试用例，测试系统的出差补助计算功能

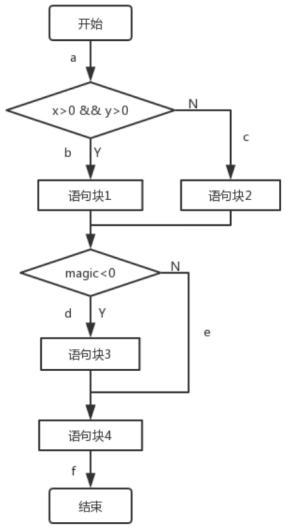
	序号	1	2	3	4
条件桩	是否长期出差	1	0	0	0
	是否出差省会	-	1	0	0
	是否出差非省会	-	-	1	0
	其他	-	-	-	1
动作桩	补助1500		1		
	补助1000	1			
	补助500			1	
	补助0				1

测试用例编号	输入条件	预期输出
1	办理长期出差	补助 1500
3	未办理长期出差，出差省会	补助 1000
4	未办理长期出差，不出差省会	补助 800
5	未办理出差	补助 0

2. 条件、判定、条件-判定、条件组合覆盖，设计/判断测试用例。

```
if(x>0 && y>0)
{
    magic = x+y+10; // 语句块1
}
else
{
    magic = x+y-10; // 语句块2
}

if(magic < 0)
{
    magic = 0;        // 语句块3
}
return magic;       // 语句块4
```



语句覆盖可以很直观地从源代码得到测试用例。无须细分每条判断每条表达式。由于这种测试方法仅仅针对程序逻辑中显示存在的语句，对于隐藏的条件和可能达到的隐式逻辑分支是无法测试的。并且语句逻辑对于多分支的逻辑运算是无法全面反映的。

（2）测试用例

当（x=3,y=3），　执行语句块：语句块1、语句块4，　所走的路径：a-b-e-f

当（x=-3,y=0），　执行语句块：语句块2、语句块3，　所走的路径：a-c-d-f

(3)测试充分性

假设判断条件 if(x>0 && y>0)中的"&&"被程序员错误的写成了"||"，即 if(x>0 || y>0)，使用上面的一组测试用例进行测试，任然可以达到100%的语句覆盖，所以语句覆盖无法发现上述的逻辑错误。**在六种逻辑覆盖标准中，语句覆盖标准是最弱的。**

判断条件实际上就是将前两种方法结合起来的设计方法，它是判定和条件覆盖设计方法的交集，即设计足够的测试用例，使得判定条件中的所有条件可能取值至少执行一次，同时，所有判断的可能结果至少执行一次。

（2）测试用例

数据	C1（x>0）	C2(y>0)	C3(magic<0)	P1(x>0 && y>0)	P2（magic <0）	路径
{x=3,y=3}	T	T	T	T	F	a-b-e-f
{x=-3,y=0}	F	F	F	F	T	a-c-d-f

（3）测试的充分性

达到100%判断-条件覆盖标准一定能够达到100%条件覆盖，100%判定覆盖和100%语句覆盖。

判断覆盖也叫分支覆盖，即设计若干用例，运行被测程序，使得程序中每个判断的取真分支和取假分支至少经历一次，即判断真假值均曾被满足。

判定覆盖比语句覆盖要多几乎一倍的测试路径，当然也就比语句覆盖具有更强的测试能力。

（2）测试用例

数据	P1（x>0 && y>0）	P2（magic < 0）	路径
{x=3,y=3}	T	F	a-b-e-f
{x=-3,y=0}	F	T	a-c-d-f

两个判断的取真或取假分支都已被执行过，所以满足了判断覆盖的标准。

假设判断条件 if(x>0 && y>0)中的"&&"被程序员错误的写成了"||"，即 if(x>0 || y>0)，使用上面的一组测试用例进行测试，任然可以达到100%的语句覆盖，所以判断覆盖无法发现上述的逻辑错误。

与语句覆盖相比，由于可执行语句要么在判断的真分支，要么在假分支上，所以满足了判定覆盖标准就一定满足了语句覆盖标准，反之则不然。因此，**判断覆盖比语句覆盖更强。**

条件覆盖的思想是设计若干测试用例，执行被测程序后，要使每个判断中每个条件的可能取值至少满足一次。

条件覆盖与判断覆盖相比较，增加了对符合判定情况的测试，增加了判定路径，要达到条件覆盖，需要足够多的测试用例，但条件覆盖并不能保证判断覆盖。

（2）测试用例

数据	C1（x>0）	C2(y>0)	C3(magic<0)	P1(x>0 && y>0)	P2（magic <0）	路径
{x=3,y=3}	T	T	T	T	F	a-b-e-f
{x=-3,y=0}	F	F	F	F	T	a-c-d-f

三个条件的各种可能取值都满足了一次，因此达到了100%条件覆盖标准。

（3）测试的充分性

上面的测试用例在达到条件覆盖的同时也达到了100%的判断覆盖标准。但是并不能保证达到100%条件覆盖标准的测试用例都能达到100%判断覆盖标准，

条件组合覆盖的基本思想是设计足够多的测试用例，使得判断中每个条件的所有可能至少出现一次，并且每个判断本身的判定结果也至少出现一次。**它与条件覆盖的差别是它不是简单的要求每个条件都出现"真"与"假"两种结果，而是让这些结果的所有可能组合都出现一次。**

注意：a.条件组合只针对同一个判断语句内存在多个条件的情况。让这些添加的取值进行笛卡尔乘积组合。

b.不同的判断语句内的条件取值之间无需组合。

c.对于单条件的判断语句，只需要满足自己的所有取值即可。

（2）测试用例

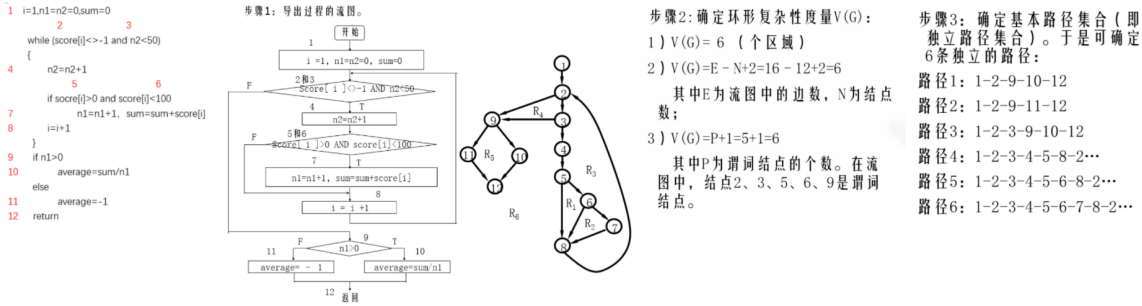
数据	C1（x>0）	C2(y>0)	C3(magic<0)	P1(x>0 && y>0)	P2（magic <0）	路径
{x=-3,y=0}	F	F	T	F	T	a-c-d-f
{x=3,y=3}	T	T	F	T	F	a-b-e-f
{x=-3,y=3}	F	T	T	F	T	a-c-d-f
{x=3,y=0}	T	F	F	F	T	a-c-d-f

基本路径覆盖就是设计所有的测试用例，来覆盖程序中所有可能的独立的执行路径。

（2）测试用例

数据	C1（x>0）	C2(y>0)	C3(magic<0)	P1(x>0 && y>0)	P2（magic <0）	路径
{x=-3,y=0}	F	F	T	F	T	a-c-d-f
{x=3,y=3}	T	T	F	T	F	a-b-e-f
{x=-3,y=14}	F	T	F	F	F	a-c-e-f
这条路径存在						a-b-d-f

3. **画出单条件的控制流程图；计算环路复杂度；写出基本路径集合；路径覆盖测试用例，输出结果，环路复杂度是基本路径条数（圈复杂度是基本路径集中的独立路径条数的上限）。说法是否正确，并说明理由。**



步骤4：为每一条独立路径各设计一组测试用例，以便执行程序沿着该路径至少一次。

1）路径1(1-2-9-10-12)的测试用例：

score[k]=有效分数值，当k＜i；

score[i]=－1，2≤i≤50；

期望结果：根据输入的有效分数算出正确的分数个数n1、总分sum和平均分average。

2）路径2(1-2-9-11-12)的测试用例：

score[1]= － 1 ；

期望的结果：average = －1，其他量保持初值。

3）路径3(1-2-3-9-10-12)的测试用例：

输入多于50个有效分数，即试图处理51个分数，要求前51个为有效分数；

期望结果：n1=50、且算出正确的总分和平均分。

4）路径4(1-2-3-4-5-8-2…)的测试用例：

score[i]=有效分数，当i<50；

score[k]<0,k< i ；

期望结果：根据输入的有效分数算出正确的分数个数n1、总分sum和平均分average。

5）路径5的测试用例：

score[i]=有效分数,当i<50；

score[k]>100,k< i ；

期望结果：根据输入的有效分数算出正确的分数个数n1、总分sum和平均分average。

6）路径6(1-2-3-4-5-6-7-8-2…)的测试用例：

score[i]=有效分数，当i<50；

期望结果：根据输入的有效分数算出正确的分数个数n1、总分sum和平均分average。

如果基本路径必须是可行的，则逻辑关系会压缩基本路径集合数量，因此环复杂度是包含起始点和终止点的基本路径数目的上限。

计算圈复杂度的两种方法，为什么基本路径测试法要计算圈复杂度，简述导出测试路径的算法。

V(G) = 区域数量(由节点、连线包围的区域，包括图形外部区域)
V(G) = 连线数量 - 节点数量 + 2
V(G) = 判定节点数量 + 1

原因：代码逻辑复杂度的度量，提供了被测代码的路径数量。复杂度越高，出错的概率越大。

步骤：1、画出控制流图；2、计算圈复杂度；3、导出测试用例 (一条独立路径是指，和其他的独立路径相比，至少引入一个新处理语句或一个新判断的程序通路。V(G)值正好等于该程序的独立路径的条数。)；4、准备测试用例

4. **年月日问题，大概意思为输入一个日期，一个程序输入year、month、day日期，功能是计算1天后的日期，写判定表和测试用例。**

选项	1	2	3	4	5
年：Y	平年	闰年			
月：M	1,3,5,7,8,10	4,6,9,11	12	2	
日：D	1-27	28	29	30	31
年月不变.日加1					
年不变.月加1.日归1					
年加1.月归1.日归1					

序号		1	2	3	4	5	6	7	8	9	10-11	12	13	14	15
条件	year	-	-	-	-	-	-	-	-	-	-	-	Y1	Y2	Y2
	month	-	M1	M1	M1	M1	M2	M2	M2	M3	M3-M3	M3	M4	M4	M4
	day	D1	D2	D3	D4	D5	D2	D3	D4	D2	D3-D4	D5	D2	D2	D3
动作	年月不变.日加1	√	√	√	√		√	√		√	√-√			√	
	年不变月加1日归1					√			√				√		√
	年加1月归1日归1											√			

5. **状态图、状态表、状态生成树**

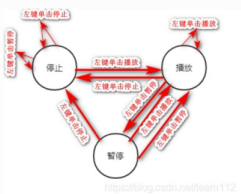
1. 将软件需求规格说明书划分成需求子片段
2. 分析需求子片段，找出状态，跳转条件
3. **设定一个初始状态**，以圆圈（代表状态）为节点，以箭线（带跳转条件）为迁移方向画出状态迁移图
4. 通过状态迁移图得到状态事件转换表（表头为：上一状态，跳转条件，下一状态，输出结果）
5. 通过状态事件转换表得出状态事件转换树（以矩形框为节点）

例子

测试某播放器的播放功能：

状态：播放；停止；暂停

跳转条件：左键单击播放；左键单击停止；左键单击暂停



将迁移图转换为状态事件转换表：

上一状态	跳转条件	下一状态	输出结果
停止	左键单击播放	播放	音乐开始播放
停止	左键单击停止	停止	音乐停止播放
停止	左键单击暂停	停止	音乐停止播放
播放	左键单击播放	播放	音乐重新播放
播放	左键单击停止	停止	音乐停止播放
播放	左键单击暂停	暂停	音乐暂停播放
暂停	左键单击播放	播放	音乐重新播放
暂停	左键单击停止	停止	音乐停止播放
暂停	左键单击暂停	播放	音乐继续播放

表转换为树结构：