《软件测试技术》复习大纲

第一章　引论

1 软件测试学科的发展（重点）

1957年之前，调试为主 (Debugging Oriented)

1957～1978年，以功能验证为导向，测试是证明软件是正确的（正向思维）。

1978～1983年，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）。

1983～1987年，以质量评估为导向，测试是提供产品的评估和质量度量。

1988年起，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。

- 1957年，Charles Baker在他的一本书中对调试和测试进行了区分：
- **调试(Debug)：确保程序做了程序员想它做的事情**
- **测试(Testing)：确保程序解决了它该解决的问题**
- 这个时期测试的主要目就是确认软件是满足需求的，也就是我们常说的“做了该做的事情”。

- 1972年，软件测试领域的先驱Bill Hetzel博士，提出了对软件测试的定义：“**就是建立一种信心，认为程序能够按预期的设想运行。Establish confidence that a program does what it is supposed to do.**” 为表明软件正确而进行测试

- 1979年Glenford J. Myers（代表论著《The Art of Software Testing》）提出了对软件测试的定义：“**测试是为发现错误而执行的一个程序或者系统的过程。**”
- 测试的目的是寻找错误，并且是尽最大可能找出最多的错误。这个观点较之前证明为主的思路，是一个很大的进步。我们不仅要证明软件做了该做的事情，也要保证它没做不该做的事情，这会使测试更加全面，更容易发现问题。

- 1983年，《软件测试完全指南》（Bill Hetzel著）定义：测试是以评价一个程序或者系统属性为目标的任何一种活动。测试是对软件质量的度量
- 软件测试定义发生了改变，测试不单纯是一个发现错误的过程，而且包含软件质量评价的内容。制定了各类标准。
- 预防为主是当下软件测试的主流思想之一。
- 20世纪90年代，软件测试开始迅猛发展。软件测试工具开始盛行，极大地提升了软件测试的能力，同时自动化测试技术也开始迅猛发展，
- 各种对软件测试系统的评估方法也开始被提出，如1996年提出的“测试成熟度模型（TMM）”“测试能力成熟度模型（TCMM）”等。软件测试体系日益成熟完善。
- 2002年，Rick和Stefan在《系统的软件测试》一书中对软件测试做了进一步定义：“测试是为了度量和提高被测软件的质量，对测试软件进行工程设计、实施和维护的整个生命周期过程。”这一定义进一步丰富了软件测试的内容，扩展了软件测试的外延。
- STEP(Systematic Test and Evaluation Process)是最早的一个以预防为主的生命周期模型
- STEP认为测试与开发是并行的，整个测试的生命周期也是由计划，分析，设计，开发，执行和维护组成，也就是说，测试不是在编码完成后才开始介入，而是贯穿于整个软件生命周期。
- 没有100%完美的软件，零缺陷是不可能的，所以我们要做的是：尽量早的介入，尽量早的发现这些明显的或隐藏的bug，发现得越早，修复起来的成本越低，产生的风险也越小。

2 正向测试与反向测试的定义，关系

定义：

（1）正向测试：软件测试就是为程序或系统能够按预期设想运行而建立信心的过程；是一系列活动以评价一个程序或系统的特性或能力并确定是否达到预期的结果;是为了验证软件是否符合用户需求，即验证软件产品是否能正常工作

（2）反向测试：软件测试就是为发现错误而执行的一个程序或者系统的过程；测试是为了证明程序有错，而不是证明程序无错误。

关系：

（1）互补性：正向测试和反向测试是互补的，两者结合可以全面验证系统的功能和稳健性。正向测试确保系统在正常条件下工作良好，反向测试确保系统在异常条件下处理得当。

（1）覆盖面：正向测试关注正常操作路径和功能的正确性。反向测试关注异常情况和错误处理路径，验证系统在处理异常数据时的行为。

（3）测试数据：正向测试使用的是符合需求规范的有效数据。反向测试使用的是违反需求规范的无效数据。

（4）测试结果：正向测试期望系统返回预期的正确结果。反向测试期望系统能够识别错误并返回适当的错误处理结果。

3 从从经济视角认知软件测试（重点）

测试的经济观点就是以**最小的代价获得最高的软件产品质量**。经济观点也要求软件测试尽早开展工作，发现缺陷越早，返工的工作量就越小，所造成的损失就越小。测试的成本< 缺陷造成的损失，测试才有意义。

4 SQA，与软件测试关系

SQA 定义：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。

联系：

a）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。

b）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。

区别：

a）SQA 是一项管理工作，侧重于对流程的评审和监控

b）测试是一项技术性的工作，侧重对产品进行评估和验证

第二章 软件测试基本概念

1 缺陷定义，现象，判定准则

（1）定义：
（a） 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。
（b） IEEE（1983）729 软件缺陷一个标准的定义：
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）现象：
（a）功能、特性没有实现或部分实现
（b）设计不合理，存在缺陷
（c）实际结果和预期结果不一致
（d）运行出错，包括运行中断、系统崩溃、界面混乱
（e）数据结果不正确、精度不够
（f）用户不能接受的其他问题，如存取时间过长、界面不美观
（3）判定准则：
（a）Test Oracle 就是决定一项测试是否通过的（判断）的一种机制。Test Oracle 的使用会要求将被测试系统的实际输出与所期望的输出进行比较，从而判断是否有差异，即是否为缺陷。
判断依据：
1）需求规格说明书和其它需求、设计规范文档
2）竞争对手的产品启发式测试预言（Heuristic oracle）
3）统计测试预言（Statistical oracle）
4）一致性测试预言（Consistency oracle）
5）基于模型的测试预言（Model-based oracle）
6）人类预言（Human oracle）

2 软件缺陷产生的原因有哪？

技术问题
算法错误，语法错误，计算和精度问题，接口参数传递不匹配
团队工作
沟通不充分，误解
软件本身
文档错误、用户使用场合(user scenario)，
时间上不协调、或不一致性所带来的问题
系统的自我恢复或数据的异地备份、灾难性恢复等问题

3 产品质量的内容，内部，外部，使用质量

（1）产品质量的内容
定义：是人们**实践产物的属性和行为**，是可以认识的，可以科学地描述的。并且可以通过一些方法和人类活动，来改进质量
ISO/IEC 定义：产品质量是指在特定的使用条件下产品满足明示的和隐含的需求所明确具备

能力的全部固有特性。

质量模型： McCall 模型，Boehm 模型，ISO 9126 模型

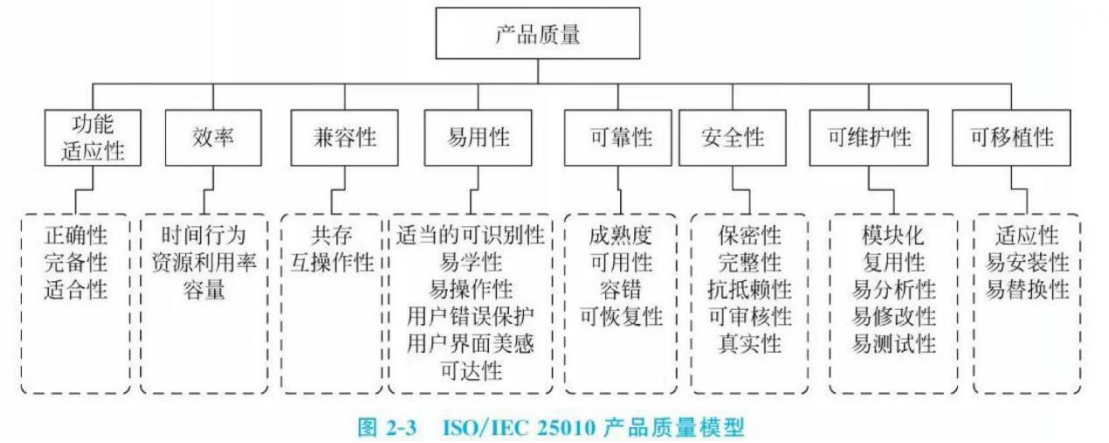


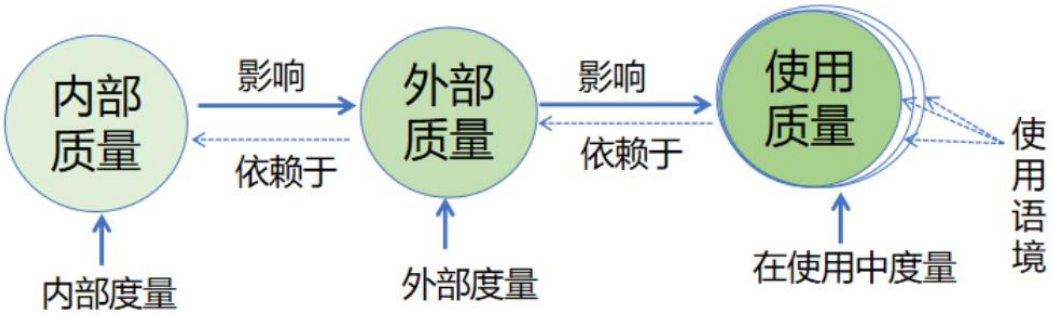
图 2-3 ISO/IEC 25010 产品质量模型

（2）内部，外部，使用质量

内部质量：是指软件系统本身的质量特性，这些特性主要涉及**软件代码和设计的内部结构**。它通常由开发人员和技术人员评估，关注代码的可维护性、可读性、可扩展性等方面。

外部质量：是指软件系统在**运行过程**中对用户和外部环境**表现出来的质量特性**。这些特性主要涉及软件的功能性、性能、可靠性、可用性等方面，通常由最终用户和客户评估。

使用质量：是指软件在**实际使用环境**中的质量特性，主要关注用户在使用过程中体验到的满意度、效率、效果等。这些特性通常由最终用户在实际操作中评估，重点在于用户的整体使用体验和业务目标的达成。



4 如何理解软件规格说明书缺陷

软件规格说明书缺陷最多

① 开发与用户沟通存在困难，对产品理解不一样

② 靠想象描述系统的实现结果，有些特性不清楚

③ 需求变化的不一致性，用户的需求不断变化，不能及时在需求中得到正确描述

④ 对规格说明书不够重视，对其投入人力，时间不足

⑤ 沟通不够

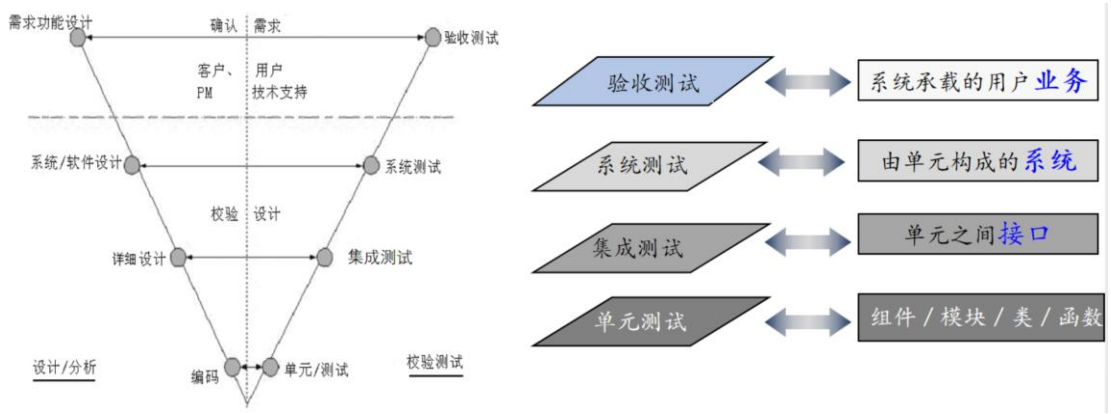
5 verification,validation

软件测试是由“验证（Verification）”和“有效性确认（Validation）”活动构成的整体

Verification：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。

Validation：是确认所开发的软件是否满足用户真正需求的活动。

6 软件测试不同层次测试的对象和任务



（1）底层测试：单元测试

（2）接口层次：集成测试，完成系统内单元之间接口和单元集成为一个完整系统的测试。

（3）系统层次：系统测试，针对已集成的软件系统进行测试。

（4）用户/业务层次：验收测试，验证是否为用户真正所需要的产品特性，验收测试关注用户环境、用户数据，而且用户也参与测试过程。

Alpha 测试是在开发人员现场由潜在用户/客户或独立测试团队进行的模拟或实际操作测试，是一种内部验收测试形式。

Beta 测试是在 Alpha 测试之后进行的。软件的 Beta 版本会发布给编程团队以外的有限用户进行测试。

7 静态测试的内容包括什么？开展相关活动时采用的形式有哪些？

（1）静态测试包括对软件产品的需求和设计文档的评审、对程序代码的审查和对代码的静态分析。

（2）静态测试的内容：需求审查、设计评审、文档检查、代码审查、静态代码分析

（3）开展相关活动时采用的形式：互为评审 (Peer review)、走查 (walk-through)、会议评审 (Inspection)、静态分析

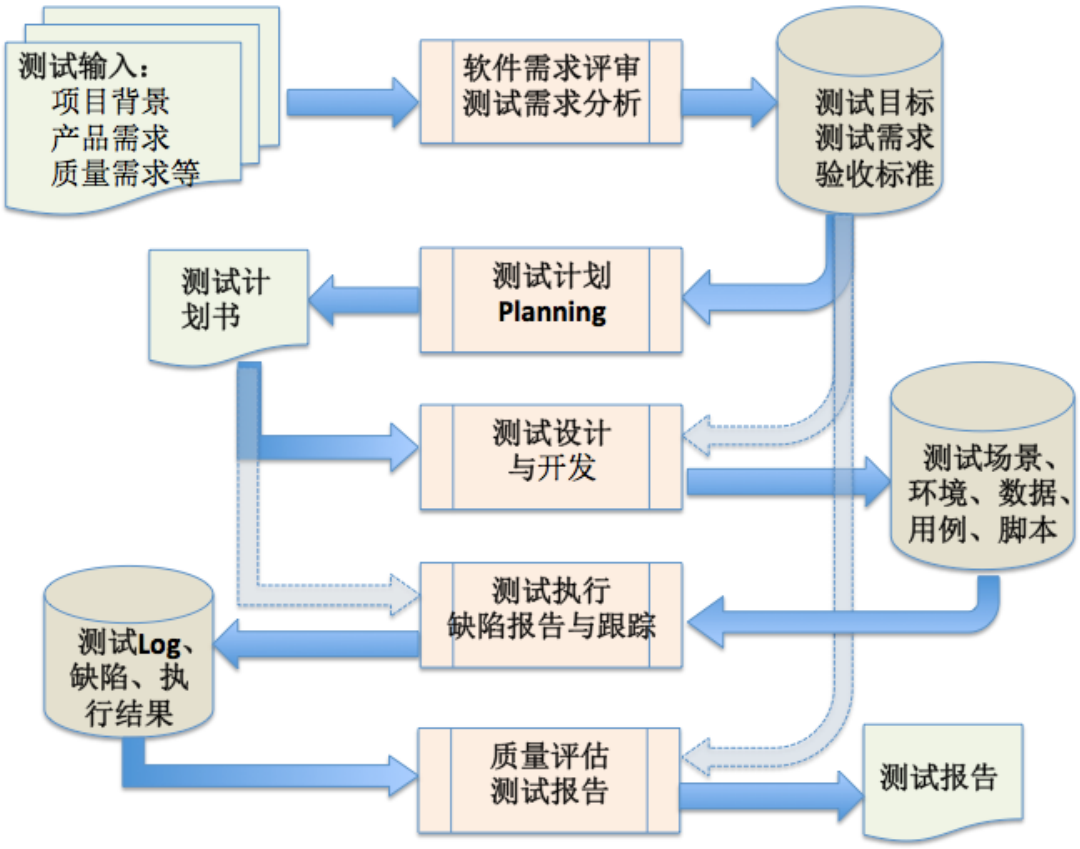
静态分析：

①人工检测：人工检测偏重于**编码风格、质量**的检验，对设计、代码进行分析，有效地发现逻辑设计和编码错误。

②计算机辅助静态分析：利用静态分析工具对被测程序进行特性分析，从程序中提取一些信息，以便检查**程序逻辑**的各种缺陷和可疑的程序构造。

8 测试工作的流程

测试输入——>软件需求评审、测试需求分析——>测试计划——>测试设计与开发——>测试执行、缺陷报告与跟踪——>质量评估、测试报告

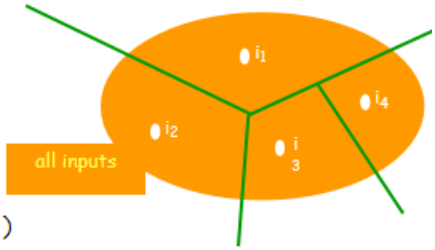


9 软件测试的工作范畴

软件测试工程过程：需求评审、设计评审、单元测试、集成测试、系统测试、验收测试。
软件测试管理过程：测试需求分析、测试策略指定、测试计划、测试设计、测试执行、测试结果、过程评估。

1 等价类

- 等价类是某个输入域的子集，在该子集中每个输入数据的作用是等效的.
- 将输入数据分成若干个等价类，从每个等价类选取一个代表性的数据作为测试用例
- 等价类分为有效等价类（合理的数据）和无效等价类（异常的数据）



3. 示例

有一报表处理系统，要求用户输入处理报表的日期。假设日期限制在 2000 年 1 月至 2020 年 12 月，即系统只能对该段时期内的报表进行处理。如果用户输入的日期不在这个范围内，则显示错误信息。并且此系统规定日期由年月的 6 位数字组成，前 4 位代表年，后两位代表月。检查日期时，可用表 3-1 进行等价类划分和编号。

表 3-1　等价类划分的一个实例

输　　入	合理等价类	不合理等价类
报表日期	① 6 位数字字符	② 有非数字字符 ③ 少于 6 个数字字符 ④ 多于 6 个数字字符
年份范围	⑤ 2000～2020	⑥ 小于 2000 ⑦ 大于 2020
月份范围	⑧ 1～12	⑨ 等于 0 ⑩ 大于 12

在进行功能测试时，只要对有效等价类和无效等价类测试进行测试，覆盖①、⑤、⑧三个有效等价类测试，只要用一个值 201006 即可；对无效等价类的测试则要分别输入 7 个非法数据，如 200a0b、20102、1012012、198802、203011、200000、202013。合起来只要完成 8 个数据的输入就可以了。如果不用等价类划分法，其测试的输入值则高达几百个，可见等价类划分法提高了测试效率。

2 边界值法

- 很多错误发生在输入或输出范围的边界上，因此针对各种边界情况设置测试用例，可以更有效地发现缺陷。
- 设计方法：
  - ✧ 确定边界情况（输入或输出等价类的边界）
  - ✧ 选取正好等于、刚刚大于或小于边界值作为测试数据

3 决策表，因果图法

判定表

- 在实际应用中，许多输入是由多个因素构成，而不是单一因素，这时就需要多因素组合分析。
- 对于多因素，有时可以直接对**输入条件（成立或不成立）**进行组合设计，不需要进行因果分析，这时就采用判定表方法。
- 判定表由“**条件和活动**”两部分组成，**即列出一个测试活动执行所需的条件组合**，所有可能的条件（输入）组合定义了一系列的选择，而测试活动（**结果输出**）需要考虑每一个选择。

- **条件桩**：问题的所有条件
- **动作桩**：针对问题所采取的动作
- **条件项**：所列条件的具体赋值（输出）
- **动作项**：在条件项组合情况下应采取的动作（输出）
- **规则**：任何一个条件组合的特定取值及其相应的动作

最后每一列需要设计一条测试用例覆盖，有几条规则（组合）就有几条测试用例

表 3-5　初始化的判定表

序　　号		1	2	3	4	5	6	7	8
条件	驱动程序是否正确	1	0	1	1	0	0	1	0
	是否有纸张	1	1	0	1	0	1	0	0
	是否有墨粉	1	1	1	0	1	0	0	0
动作	打印内容	1	0	0	0	0	0	0	0
	提示驱动程序不对	0	1	0	0	0	0	0	0
	提示没有纸张	0	0	1	0	1	0	1	1
	提示没有墨粉	0	0	0	1	0	1	0	0

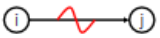
表 3-6　优化后的判定表

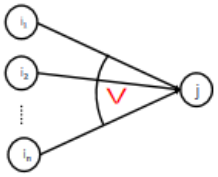
序　　号		1	2	4/6	3/5/7/8
条件	驱动程序是否正确	1	0	—	—
	是否有纸张	1	1	1	0
	是否有墨粉	1	1	0	—
动作	打印内容	1	0	0	0
	提示驱动程序不对	0	1	0	0
	提示没有纸张	0	0	0	1
	提示没有墨粉	0	0	1	0

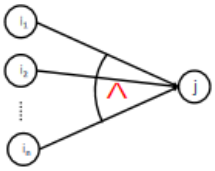
因果图

- 针对**更为复杂的**“多种输入条件、产生多种结果”设计组合测试用例。
- **设计方法：**
  - ✧ 分析软件Spec描述的哪些是原因（输入条件）、哪些是结果（输出），给每个原因和结果赋予一个标示符。
  - ✧ 找出原因与结果、原因与原因之间的对应关系，画出因果图
  - ✧ 在因果图上标上哪些不可能发生的因果关系，表明约束或限制条件
  - ✧ 根据因果图，创建（转化为）判定表，将复杂的逻辑关系转化为简单的组合矩阵
  - ✧ 把判定表的每一列转化为测试用例。

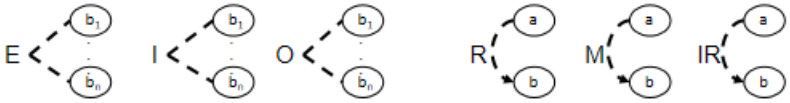
因果图的基本符号

- 有因必有果关系 
- 非-关系：只有当因 i 不存在时，果 j 才出现 
- **或-关系：**如果因 i_1 **或**因 i_2 **或**……因 i_n 存在时，结果 j 才出现。
- **与-关系：**只有当因 i_1 **与**因 i_2 **与**……因 i_n 同时存在时，结果 j 才出现。

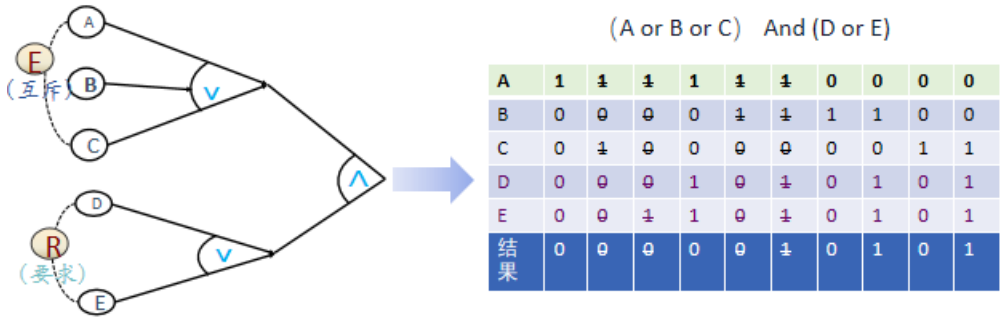




条件之间的关系



E（互斥）：最多只能有一个条件被满足
I（包含）：至少有一个条件被满足
O（唯一）：正好（只能一个）条件满足
R（要求）：满足了条件 a 要求满足条件 b
M（屏蔽）：满足条件 a 隐含的要求不满足条件 b
IR（无关）：如果满足了条件 a，条件b 就无关重要了



例如，某个软件规格说明中包含以下的要求：第一列字符必须是 A 或 B，第二个字符必须是一个数字，在此情况下进行文件的修改；但如果第一列字符不正确，则输出信息 L；如果第二列字符不是数字，则给出信息 M。采用因果图方法进行分析，可根据表 3-7 获得图 3-3 的各种组合，其中∧表示“与”、∨表示“或”、∽表示“非”的关系。

表 3-7　因果关系表

编号	原因(Cause)	编号	结果(Effect)
C1	第一列字符是 A	E1	修改文件
C2	第一列字符是 B	E2	给出信息 L
C3	第二列字符是一个数字	E3	给出信息 M
11	中间原因		

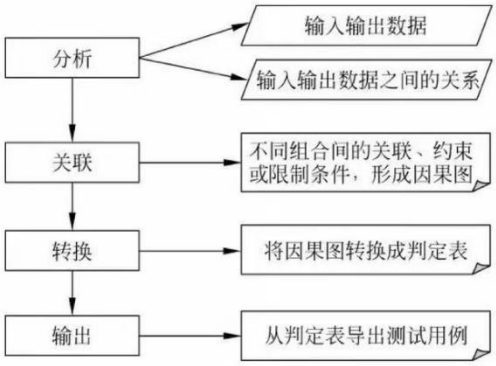


图 3-2　因果图法示例

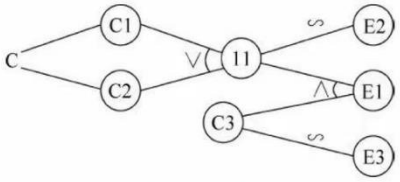


图 3-3　因果图表示

根据图 3-3 可以制定一张判定表，3 个因素共有 8 种组合，考虑到 C1(首字符是 A)成立时，C2(首字符是 B)就不能成立，就变成 6 种组合，如表 3-8 所示。可以根据判定表来设计测试用例，每一列就代表一个测试用例，共设计 6 个测试用例。但实际上，还可以进一步优化，由于第二个字母不是数字时，第一个字母不管是 A 或是 B，或 A、B 都不是，结果都一样，即 E3 成立。所以表 3-8 可进一步优化为 4 种组合，即 4 个测试用例，如表 3-9 所示。

表 3-8　上述例子的判定表

序　　号		1	2	3	4	5	6
原因	C1	1	0	0	1	0	0
	C2	0	1	0	0	1	0
	C3	1	1	1	0	0	0
结果	E1	1	1	0	0	0	0
	E2	0	0	1	0	0	0
	E3	0	0	0	1	1	1

表 3-9　优化后的判定表

序　　号		1	2	3	4/5/6
原因	C1	1	0	0	—
	C2	0	1	0	—
	C3	1	1	1	0
结果	E1	1	1	0	0
	E2	0	0	1	0
	E3	0	0	0	1
用例		首字符为 A，第二个字符为数字	首字符为 B，第二个字符为数字	首字符为 x，第二个字符为数字	首字符为 A 或 B 或 x，第二个字符不是数字

4 各种逻辑覆盖法

逻辑覆盖：以程序或系统的内部逻辑结构为基础，分为语句覆盖、判定覆盖、判定-条件覆盖、条件组合覆盖等；

（1）语句覆盖

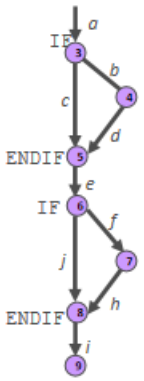
设计若干测试用例，运行被测程序，使程序中的每个可执行语句至少被执行一次

程序源代码

```
1.  dim a, b as integer
2.  dim c as double
3.  if (a >0 and b > 0) then
4.      c = c / a
5.  end if
6.  if (a > 1 or c > 1) then
7.      c = c + 1
8.  end if
9.  c = b + c
```

(a, b ,c)= (1, 1, 2)

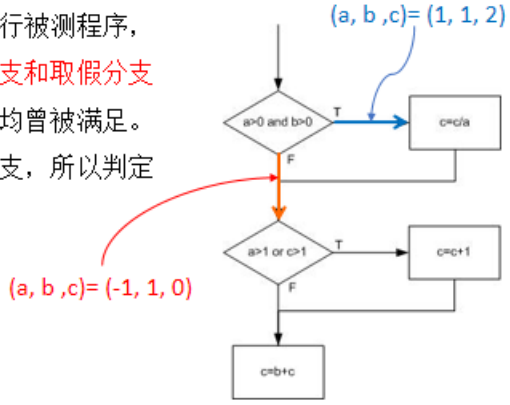
覆盖



程序控制流图

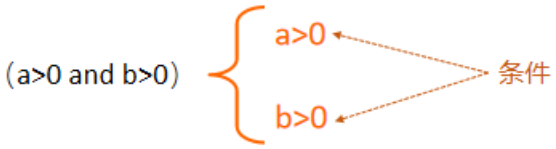
（2）判定（分支）覆盖

- **判定覆盖**：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。
- 一个判定代表着程序的一个分支，所以判定覆盖也被称为**分支覆盖**。

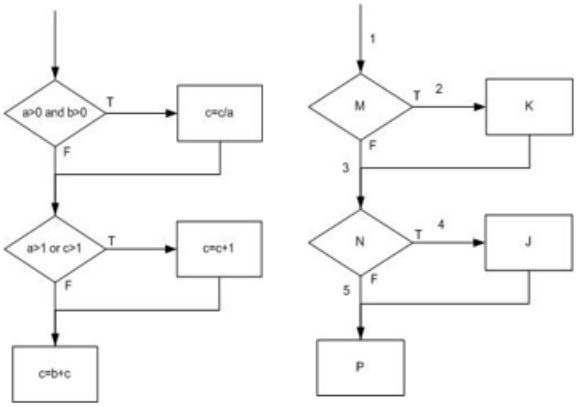


（3）条件覆盖

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



示例：列出所有条件



判定M:

•条件a>0：取.T时为T1
取.F时为F1
•条件b>0：取.T时为T2
取.F时为F2；

判定N:

•条件a>1：取.T时为T3
取.F时为F3
•条件c>1：取.T时为T4
取.F时为F4

满足条件覆盖不一定满足分支覆盖，满足分支覆盖不一定满足条件覆盖

（4）判定/条件覆盖

- 判定-条件覆盖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，使得判断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

条件	取值	覆盖条件	分支	覆盖判定
a>0, b>0, a>1，c>1	(a, b ,c)= (2, 1, 2)	T1，T2，T3，T4	a>0 AND b>0	M=.T. N=.T.
a<=0, b<=0, a<=1, c<=1	(a, b ,c)= (-1, 0, 1)	F1，F2，F3，F4	a>1 OR c>1	M=.F. N=.F.

（5）条件组合覆盖

- 条件组合覆盖的基本思想是设计足够的测试用例，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地要求每个条件都出现“真”与“假”两种结果，而是要求让这些结果的**所有可能组合都至少出现一次**

(a>0 and b>0)
- .T. .T. (1, 1)
- .T. .F. (1, -1)
- .F. .T. (-1, 1)
- .F. .F. (-1, -1)

（6）MC/DC 覆盖

- 每个判定的所有可能结果至少能取值一次；
- 判定中的每个条件的所有可能结果至少取值一次；
- 一个判定中的每个条件独立地对结果产生影响；
- 每个入口和出口至少执行一次

表 3-16 必要的测试条件组合

AND 关系	OR 关系
(1) . T. and . T. → . T.	(1) . T. or . F. → . T.
(2) . T. and . F. → . F.	(2) . F. or . T. → . T.
(3) . F. and . T. → . F.	(3) . F. or . F. → . F.

5 路径覆盖法

（1）基本路径覆盖的设计过程

a）依据代码绘制流程图

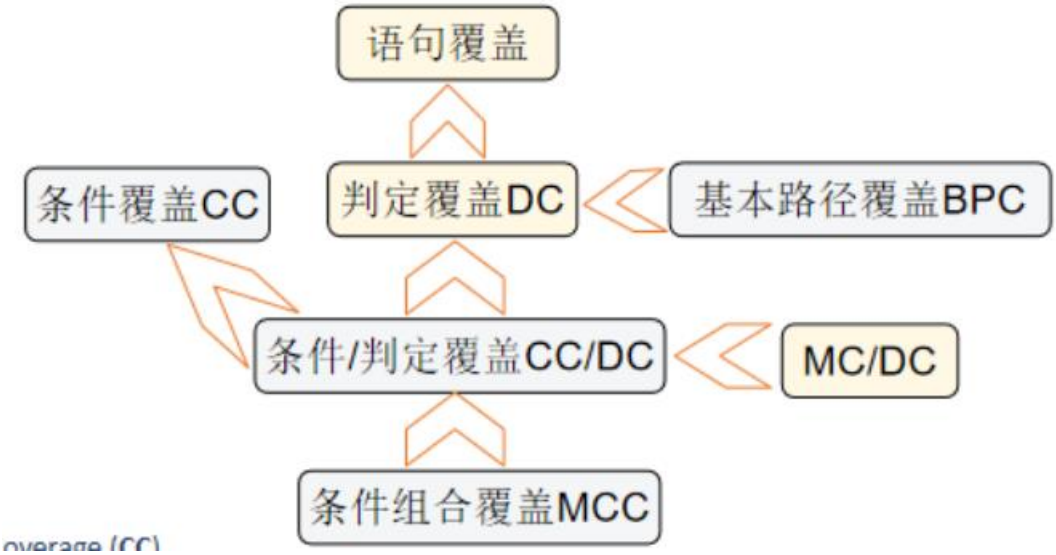
b）确定流程图的圈复杂度（cyclomatic complexity ）

c）确定线性独立路径的基本集合(basis set)

d）设计测试用例覆盖每条基本路径

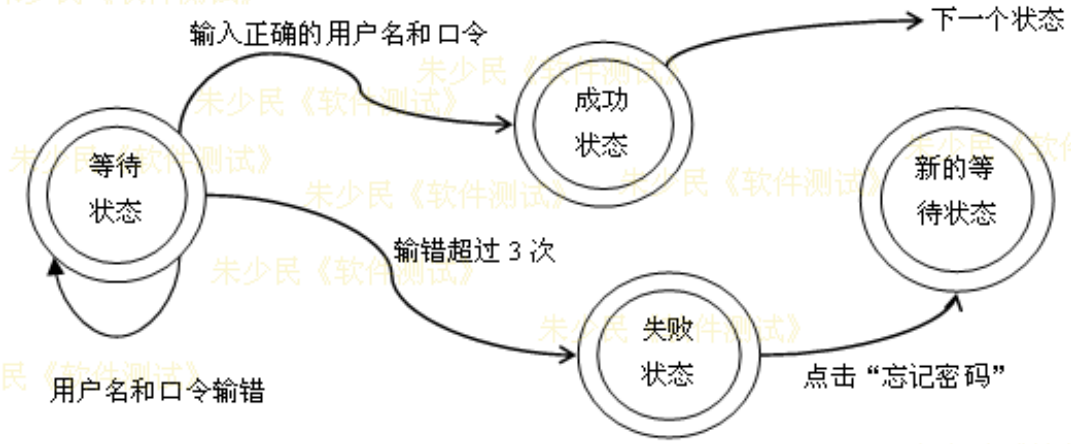
（2）确定流程图的圈复杂度的 3 种方法

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
V(G)是基本路径集合条数的上限
（3）确定独立路径集合
a）独立路径：至少引入一系列新的处理语句或条件的任何路径，即覆盖了其他路径没覆盖到的
b）基本集：由独立路径构成的集合
c）由基本集导出的测试用例，保证每行代码语句至少被执行一次
d）基本集合不一定唯一



6 功能图法，EFMS

（1）基于模型的测试 (MBT, Model-based testing)：通过构建**能够正确描述被测软件系统功能特性**的模型，然后基于这个模型产生测试用例并执行这些测试用例的过程
（2）MBT 实施过程
a. 为被测试系统（SUT）建模
b. 基于模型产生测试用例
c. 将抽象的测试具体化使测试用例具有可执行性
d. 执行测试
e. 分析测试结果
（4）功能图法
a. 功能图法就是为了解决动态说明问题的一种测试用例的设计方法
b. 功能图由状态迁移图（state transition diagram，STD）和逻辑功能模型（logic function model， LFM）构成
c. 状态迁移图，描述系统状态变化的动态信息——动态说明，由状态和迁移来描述，状态指出数据输入的位置（或时间），而迁移则指明状态的改变



d. 功能图法是综合运用黑盒方法和白盒方法来设计测试用例，即**整体上**选用白盒方法——**路径覆盖、分支和条件覆盖**等，而**局部上**选用的是黑盒方法——**决策表或因果图**方法

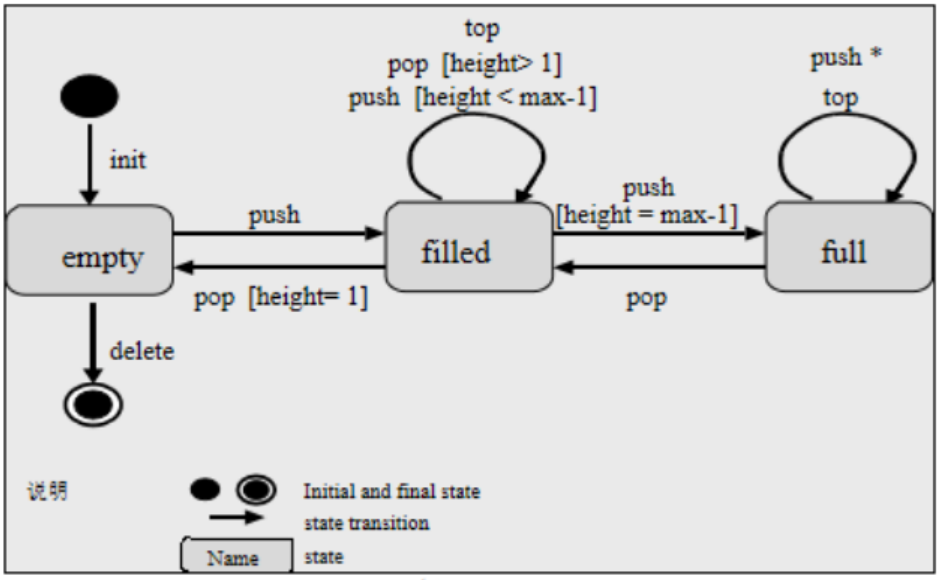
（5）模糊测试：在一个被测试程序中附加上随机数据（fuzz）作为程序的输入。如果被测试程序出现问题（例如 Crush ,或者异常退出），就可以定位程序的缺陷。

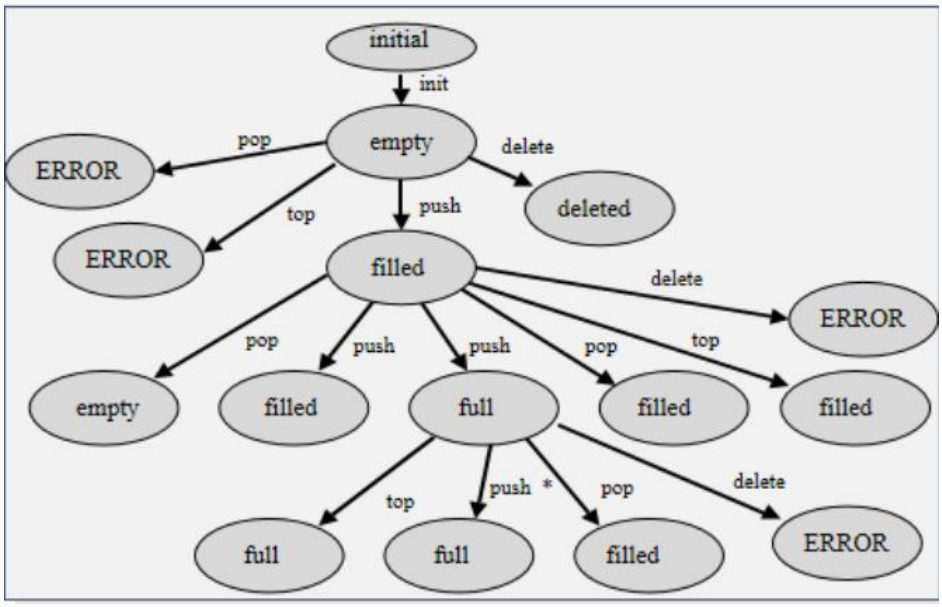
（6）变异测试（Mutation Testing）是一种在细节方面改进程序源代码的软件测试方法。变异操作是模拟典型应用错误（定位代码的弱点）、或强制产生有效地测试

（7）形式化方法:基于数学的方法（数学表示、精确的数学语义）来描述目标软件系统属性的一种技术。

（8）**有限状态机**（ Finite State Machine ，FSM）是对象行为建模的工具，以描述对象在其生命周期内所经历的状态序列，以及如何响应来自外界的各种事件。

一个堆栈的状态图(state diagram)





（9）扩展有限状态机（Extended Finite State Machine,EFSM）模型:在 FSM 模型基础上增加了动作和转移条件，以处理系统的数据流问题，但 FSM 模型只能处理系统的控制流问题。

第四章 软件测试流程和规范

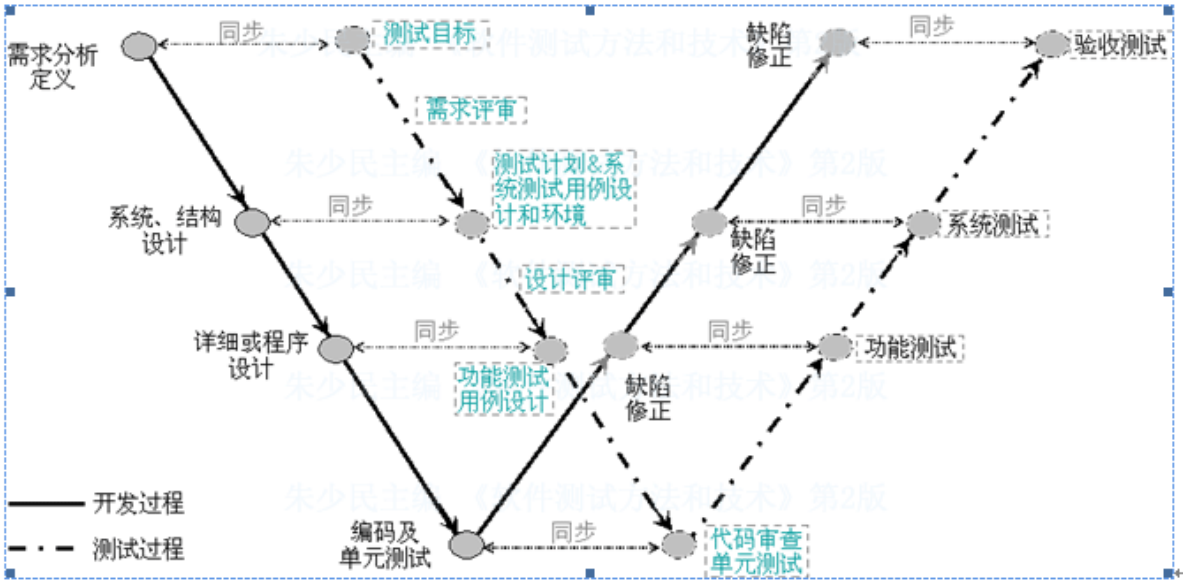
1 测试左移和右移，贯穿全生命周期的测试思想

（1）测试左移：不仅让**开发人员做更多的测试**，而且需要做需求评审、设计评审，验收测试驱动开发（ATDD），测试计划和测试设计可以在更早的时候开始

（2）测试右移：是**在线测试**（Test in Production，TiP），包括在线性能监控与分析、A/B测试和日志分析等，可以和现在流行的 DevOp 联系起来

（3）测试贯穿全生命周期，不仅可以在第一时间内发现缺陷，降低缺陷带来的成本，而且能有效地预防的产生，构建更好的软件产品质量。

2 w 模型



（1）由图 4-2 可以看出,软件分析、设计和实现的过程,同时伴随着软件测试、验证和确认的过程，而且包括软件测试目标设定、测试计划和用例设计、测试环境建立等一系列测试活动的过程。

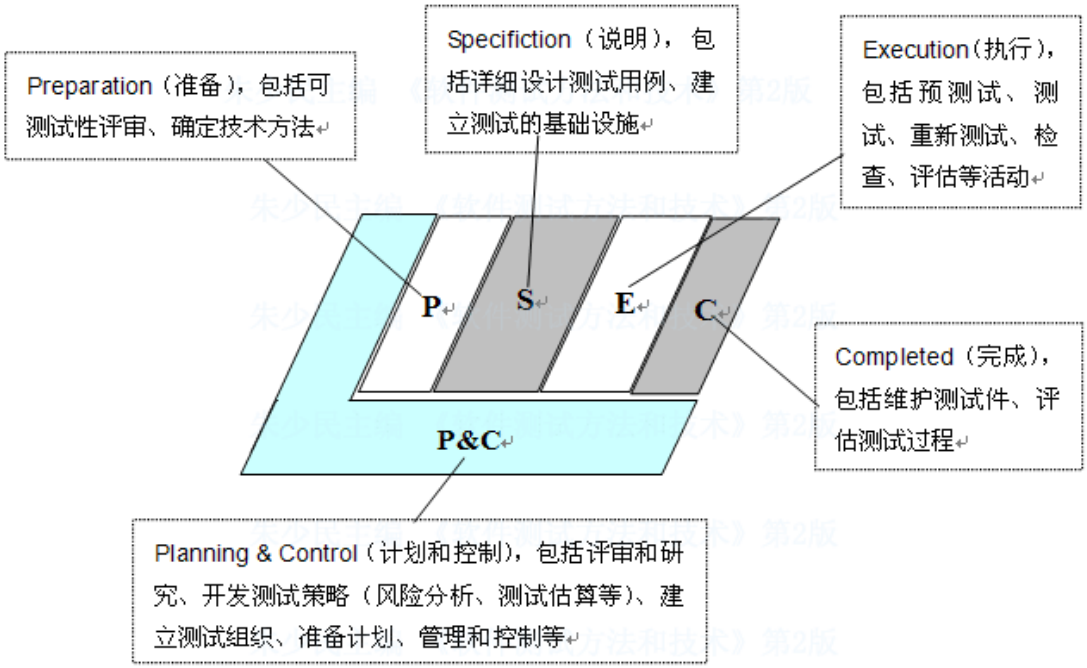
（2）测试过程和开发过程都贯穿软件过程的整个生命周期它们是相辅相成、相互依赖的关系。

- 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,所以两者是相互依赖的。前期,测试过程更多地依赖于开发过程,后期,开发过程更多地依赖于测试过程。
- 测试过程中的工作重点和开发工作的重点可能是不一样的,两者有各自的特点。不论在资源管理,还是在风险管理,两者都存在着差异。

3 TMAP 定义，几个阶段，模型基石及关系

（1）定义：TMap (Test Management Approach，**测试管理方法**)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。

（2）TMap 所定义的测试生命周期由**计划和控制**、**准备**、**说明**、**执行**和**完成**等阶段组成。

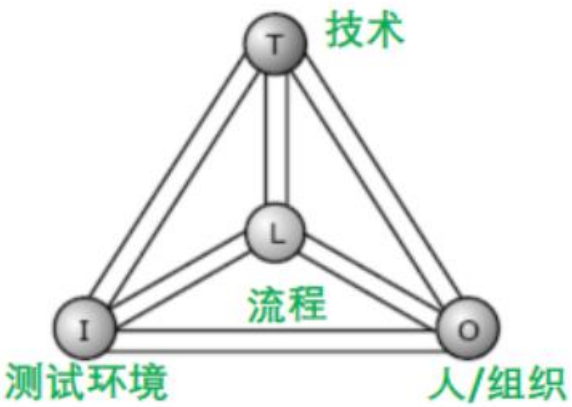


- 计划和控制阶段覆盖测试生命周期,涉及测试计划的创建、执行、调整和测试过程监控。在测试过程中,通过定期和临时的报告,客户可以经常收到关于产品质量和风险的更新。
- 准备阶段决定软件说明书质量是否足以实现说明书和测试执行的成功。
- 说明阶段涉及定义测试用例和构建基础设施。一旦测试目标确定,测试执行阶段就开始。
- 执行阶段需要分析预计结果和实际结果的区别,发现缺陷并报告缺陷。
- 完成阶段包括对测试资料的维护以便于再利用,创建一个最终的报告以及为了更好地控制将来的测试过程对测试过程进行评估。

（3）TMap 三大基石

与软件开发生命周期一致的测试活动生命周期（L）

- 坚实的组织融合（O）
- 正确的基础设施和工具（I）
- 可用的技术（T）



4 SBTM 基本要素，结果，原理

（1）定义：SBTM 基于会话的测试管理，管理 ET

（2）SBTM 基本要素

- Session(会话)是一段不受打扰的测试时间（通常是 90 分钟），是测试管理的最小单元。
- 每个 session 关联一个特定的、目标明确的测试任务（mission）
- Charter （章程，即测试指导）：对每个 session 如何执行进行简要的描述，相当于每个 session 需要一个简要的计划（提纲）

一系列 Session 相互支持，有机地组合在一起，周密地测试了整个产品。

（3）结果

- A session sheet（测试报告）：相当于**测试报告**，供第三方（如测试经理、ScrumMaster 等）进行检查的材料。它最好能被工具扫描、解析和合成。
- Debriefing（听取口头报告）：**口头汇报**，更准确地说，是测试人和其 lead/Manager 之间的对话。

（4）原理

探索性测试（Exploratory Testing）、结构化管理（Structured Management）、时间管理（Time Management）、持续改进（Continuous Improvement）

5 测试几个学派的特点

1)**分析学派**：认为**测试是严格的和技术性的**，在学术界有许多支持者，侧重于使用类似 UML 工具进行分析和建模。例子：代码覆盖率、结构化测试。

2)**标准学派**：将测试视为**衡量进度**的一种方法，强调成本和可重复的标准。

3)**质量学派**：测试是过程的**质量控制、揭示项目质量风险**的活动。**强调过程规范性**，监督开发人员并充当质量的看门人。

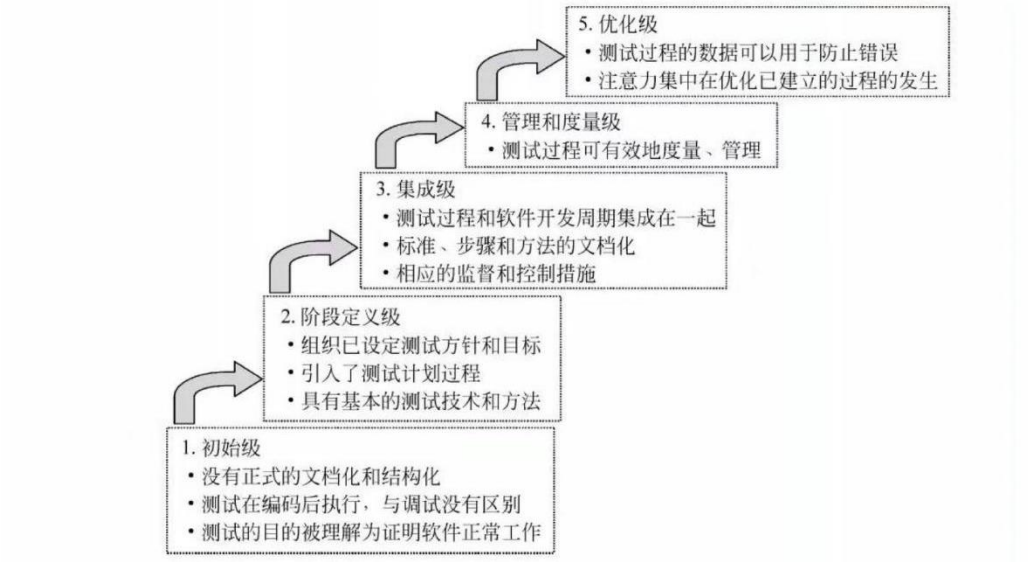
4)**上下文驱动学派**：认为**软件是人创造的**，测试所发现的每一个缺陷都和相关利益者密切相关；测试是一种有技巧的心理活动；强调**人的能动性**和**启发式测试思维**。例子：探索性测试。

5)**敏捷学派**：认为软件就是持续不断的对话，而测试就是**验证开发工作是否完成**，强调自动化测试。例子：TDD。

6 TMM，TPI，CTP，STEP 定义，特点

（1）TMM(Testing Maturity Model integration)测试成熟度模型

1) 定义：TMM 是一个框架，用于评估和改进组织的测试过程的成熟度。它基于 CMM（Capability Maturity Model），旨在通过一系列逐步改进的阶段，帮助组织优化其测试过程。
2) 特点：TMM 也将测试过程成熟度分为 5 个等级--初始级、阶段定义级、集成级、管理和度量级、优化级。每一个等级都包括**已定义的过程域**，组织在升级到更高一个等级之前，需要完全满足前一个等级的过程域。要达到特定的等级需要实现一系列的预先定义好的**成熟度目标和附属目标**。这些目标根据活动、任务和责任等进行定义，并依据管理者、开发人员、测试人员和客户或用户的特殊需求来进行。



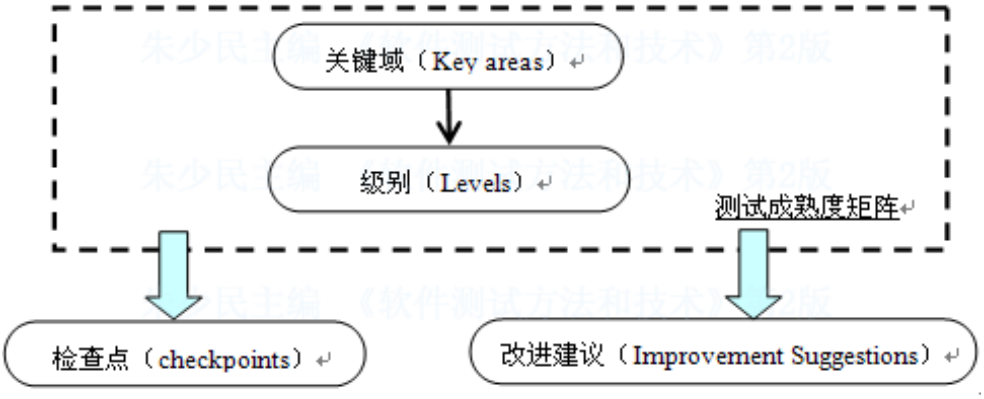
3） TMM 的建立，得益于以下 3 点：

- 充分吸收 CMM 的精华
- 基于历史演化的测试过程
- 业界的最佳实践

（2）TPI (Test Process Improvement)测试过程改进

1） 定义：TPI 是业务驱动的、基于连续性表示法的**测试过程改进**的参考模型，是在**软件控制**、**测试知识**以及**过往经验**的基础上开发出来的。

2） 特点：TPI 模型用于支持测试**过程的改进**,包括了一系列的关键域、生命周期、组织、基础设施、工具及技术,并可以用于**了解**组织内测试**过程的成熟度**。在这个基础上,该模型有助于定义渐进的、可控的改进步骤。



- 20 个关键域
- 级别：为了了解过程在每个关键域所处的状态，即对关键域的评估结果，通过级别是来体现。模型提供了 4 个级别，由 A 到 D，A 是最低级。根据测试过程的可视性改善、测试效率的提高、或成本的降低以及质量的提高，级别会有所上升。
- 检查点：每个级别都有若干个检查点，测试过程只有在满足了这些检查点的要求之后，才意味着它达到了特定的级别。
- 建议：**检查点**帮助我们**发现**测试过程中的**问题**，而**建议**会帮助我们**解决问题**，最终改进测试过程。建议不仅包含对如何达到下个级别的指导,而且还包括一些具体的操作技巧、注意事项等。

（3）CTP (Critical Test Process)关键测试过程

1) 定义：CTP 是一种强调在测试过程中关注关键过程的模型。它帮助组织识别和优化那些对测试成功至关重要的过程。

2) 特点：

- 使用 CTP 的过程改进，始于对现有测试过程的评估，通过评估以识别过程的强弱，并结合组织的需要提供改进的意见。
- 计划（Plan）、准备（Prepare）、执行（Perform）和完善 (Perfect)；计划和完善主要是管理工作，准备和执行是实践工作。
- 12 个关键过程

（4）STEP (Systematic Test & Evaluation Process)系统化测试和评估过程

1) 定义：STEP 是一个内容参考模型，认定测试是一个生命周期活动，在明确需求后开始直到系统退役。

2) 特点：STEP 强调度量，量化的因子包括：已定义的测试过程使用，客户满意度

第五章

1 代码评审（代码审查）的形式及各自特点

代码中 60%以上的缺陷可以通过代码审查发现

（1）互查

特点：代码互查是日常工作中使用最多的一种代码评审方式,比较容易开展,相对自由。两个人之间互查

（2）走查

定义：采用讲解、讨论和模拟运行的方式进行的查找错误的活动。

特点：是一种相对比较正式的代码评审过程，程序员讲解，小组讨论，提出测试用例。

（3）会议评审

定义：以会议形式，制定目标、流程和规则按**缺陷检查表**(不断完善)逐项检查发现问题适当记录，避免现场修改发现重大缺陷，**改正后会议需要重开**。

特点：是一种最为正式的检查和评估方法。需要开发者自查，还要组织代码检查小组进行代码检查。

（4）走查和审查的比较

	走 查	审 查
准备	通读设计和编码	事先准备Spec、程序设计文档、源代码清单、代码缺陷检查表等
形式	非正式会议	正式会议
参加人员	开发人员为主	项目组成员包括测试人员
主要技术方法	无	缺陷检查表
生成文档	会议记录	静态分析错误报告
目标	代码标准规范 无逻辑错误	代码标准规范 无逻辑错误

2 单元测试定义，作用，目标

（1）定义：单元测试是对软件**基本的组成单元**（如函数、类的方法等）进行独立的测试。

（2）作用：

① 尽早发现错误，节省测试时间

② 规范行为和记录代码，检查代码是否符合设计和规范，有利于将来代码的维护

③ 支持扩展

④ 引导更好的设计

⑤ 支持变化

⑥ 避免回归缺陷

⑦ 保证工作的平稳步调

（3）目标

- **单元模块被正确实现**(主要指编码)，包括功能、性能、安全性等，但一般

主要介绍单元功能测试

- (参数)**输入**是否正确传递和得到保护(容错)，**输出**是否正常
- **内部数据**能否保持其完整性，包括变量的正确定义与引用、内存及时释放、全局变量的正确处理和影响最低
- **覆盖率**：**代码行、分支覆盖**或**MC/DC**达到要求，如高于80%或95%

（4）任务

- 单元独立执行路径的测试
- 单元局部数据结构的测试
- 单元接口测试
- 单元边界条件的测试
- 单元容错性测试
- 内存分析

3 桩程序，驱动程序

运行被测试单元,为了隔离单元,根据被测试单元的接口,开发相应的驱动程序(Driver)和桩程序(Stub)。

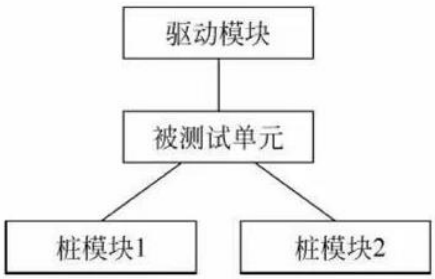


图 5-5 单元测试中驱动程序和桩程序

(1) 驱动程序(Driver),也称驱动模块,用以**模拟被测模块的上级模块,能够调用被测模块**。在测试过程中,驱动模块接收测试数据,调用被测模块并把相关的数据传送给被测模块。

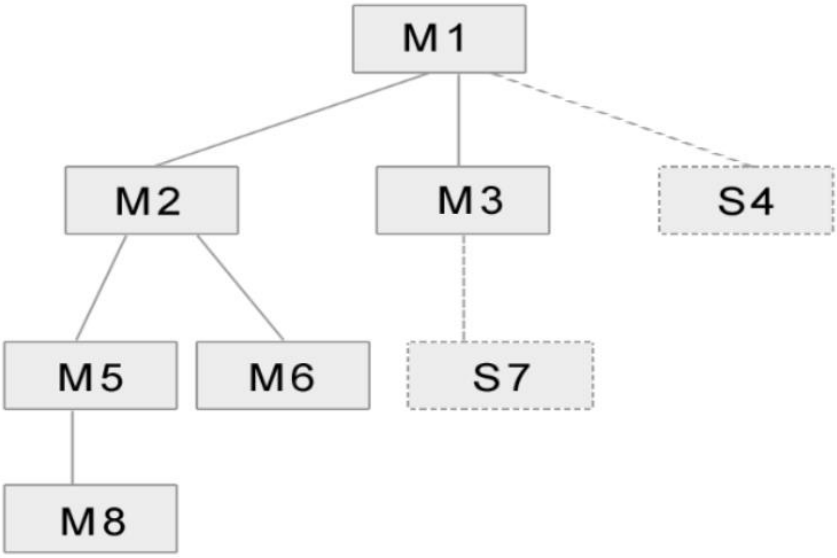
(2) 桩程序(Stub),也称桩模块,用以**模拟被测模块工作过程中所调用的下层模块**。桩模块由被测模块调用,它们一般只进行很少的数据处理,如打印入口和返回,以便于检验被测模块与其下级模块的接口。

4 集成测试的概念

定义：集成测试是将已分别通过测试的单元按设计要求集成起来再进行的测试，以检查这些单元之间的接口是否存在问题，包括接口参数的一致性引用、业务流程端到端的正确性等。

5 单一系统的集成测试的集成模式及优缺点

(1) 自顶向下



深度优先：M1→M2→M5→M8→M6→M3→S7→S4
宽度优先：M1→M2→M3→S4→M5→M6→S7→M8

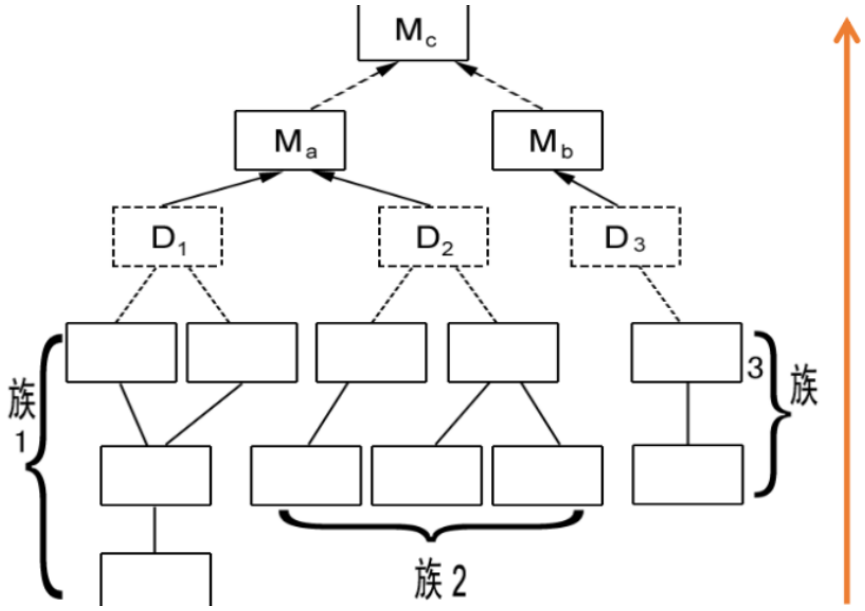
优点：

① 不需要测试**驱动程序**
② 能够在测试阶段的**早期**实现并**验证系统**的主要功能
③ 能在**早期发现**上层模块的**接口错误**

缺点：

① 需要**桩程序**,可能遇到与此相联系的测试困难
② **低层**关键模块中的**错误发现较晚**
③ 用这种方法在早期**不能充分展开人力**

(2) 自底向上



缺点：

① 需要测试**驱动程序**
② 不能够在测试阶段的**早期**实现并**验证系统**的主要功能

③ 不能在**早期发现**上层模块的**接口错误**

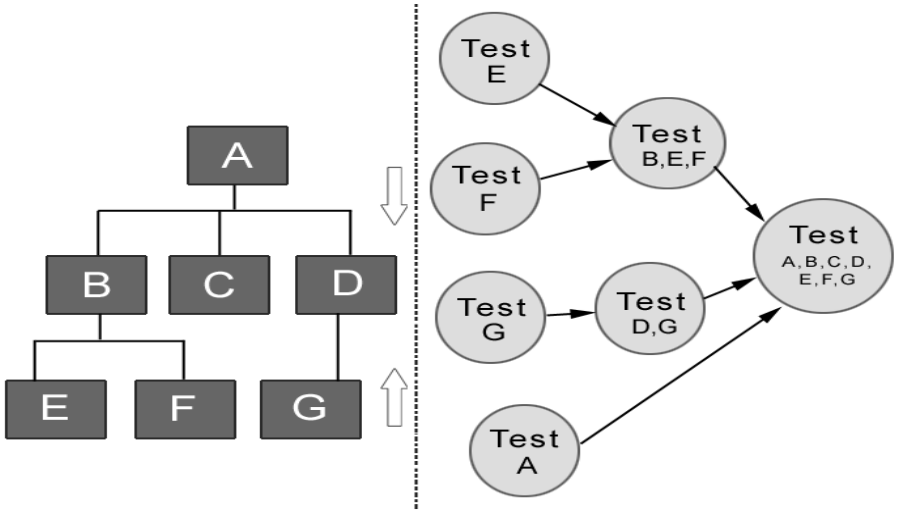
优点：

① 不需要**桩程序**,可能遇到与此相联系的测试困难

② **低层**关键模块中的**错误发现较早**

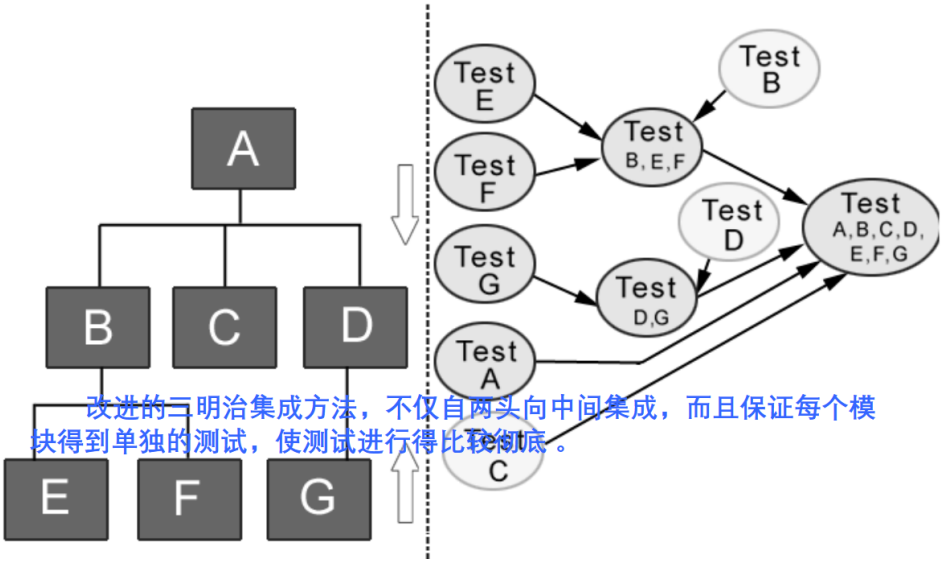
③ 用这种方法在早期**能充分展开人力**

(3) 三明治集成法



优点：它将自顶向下和自底向上的集成方法有机地结合起来,**不需要写桩程序**,因为在测试初自底向上集成已经验证了底层模块的正确性。

缺点：在真正集成之前每一个独立的模块都没有完全测试过。



6 微服务特点，测试目标？测试基本步骤？

（1）微服务特点：独立部署、独立开发、模块化、弹性伸缩、容错性、持续交付、技术多样性、松耦合

（2）测试目标：为了验证一个**子系统或功能模块**和**外部组件**之间的正常通信。外部组件包括其他的**微服务**或者**外部数据存储系统**。

（3）测试基本步骤

一个微服务和**外部服务**的集成测试的测试步骤如下：

① **启动**被测微服务和外部服务的实例
② **调用被测微服务**，该服务会通过外部服务提供的API读取响应数据
③ **检查**被测微服务是否能正确解析**返回结果**

一个微服务和**外部数据存储**的集成测试的测试步骤如下：

④ **启动**外部数据库
⑤ **连接**被测应用到**数据库**
⑥ **调用被测微服务**，该服务会往数据库写数据
⑦ 读取数据库，**查看**期望的数据是否被写到了数据库里

7 集成测试的目标

目标：确保系统中各个模块或组件在一起工作时，能够正确地协同运作。集成测试主要关注模块之间的接口、交互和数据流，确保整个系统在集成后仍然符合预期的功能和性能要求。

持续测试的特点

足够快

整个测试过程要快，一方面高度自动化测试，另方面业务端到端的探索式测试

准确且有效

被测系统往往很复杂，不可能做全回归测试，而是要推行精准测试

平滑有序

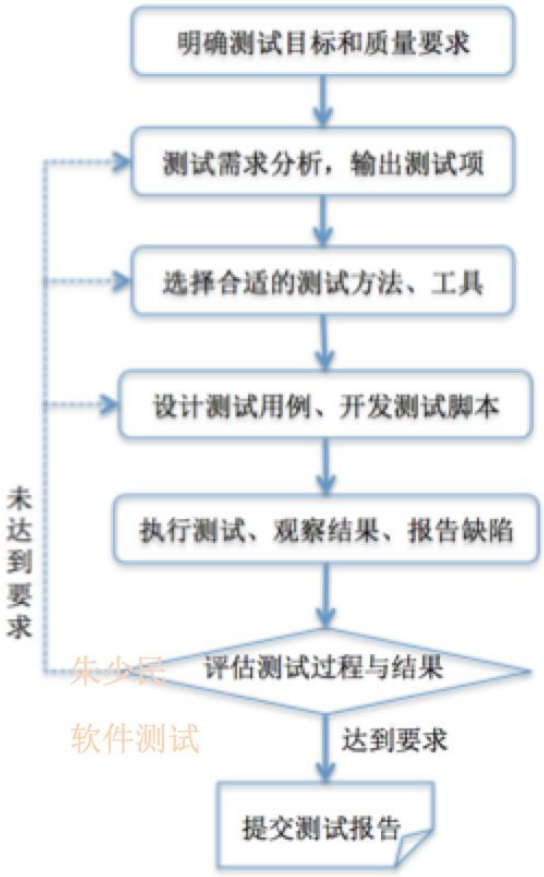
打通整个测试过程：测试左移到测试右移、单元测试到系统测试、静态测试到动态测试

与CI/CD集成

与研发的持续构建、持续集成、持续部署、运维等环境的集成

第六章　系统功能测试

1 功能测试的基本思路



(1) 首先要了解质量要求和测试目标，例如功能基本正常运行就可以了，还是要求完完全全地正常运行?
(2) 基于测试目标，阅读需求文档及相关文档，了解需求，分析测试重点、难点和风险点，对被测功能进行分析。确定功能的边界、测试所涉及的范围，然后逐条列出测试项，并根据质量风险和功能的价值(对用户的重要性)标记测试优先级。
(3) 针对所测功能的特性和实现，采用合适的测试设计方法、工具。
(4) 了解本功能测试所给定的时间，决定相应的测试策略。
(5) 根据所选择的方法和工具,设计测试用例(包括功能点用例和测试场景)或开发测试脚本,进行调试,确保能够被执行。
(6) 准备测试数据、搭建测试环境,执行测试用例(人员)或测试脚本(工具)。
(7) 分析测试结果,评估测试风险和测试的充分性,确认是否达到测试目标。如果测试还不够充分或没有达到测试目标,需要补充测试或进一步加强测试。
(8) 最后,编写测试报告,给出该版本的产品质量评估意见。

2 回归测试需要解决的问题。回归测试的策略和方法

（1）回归缺陷 (Regression bug)：原来正常工作的功能，没有发生需求变化，而由于受**其它改动**影响而产生的问题。
（2）回归测试就是**为了发现回归缺陷**而进行的测试。如果没有回归测试，产品就带着回归

缺陷被发布出去了，造成严重后果。

（3）回归测试的策略和方法

① **再测试全部用例**。选择测试用例库中的全部测试用例构成回归测试包，这是一种**最安全**的方法，具有最低的遗漏回归错误的风险，但**测试效率最低**，即**测试成本最高**。这种策略不需要进行用例分析和取舍，但是随着软件开发的不断迭代，回归测试用例不断增多而带来越来越大的工作量，最终无法在限定的进度下完成。

② **基于风险选择测试**。这里的风险是指容易**发生回归缺陷的风险**，更准确地讲，是指**受改动代码的影响风险**。这种策略主要是根据经验判断或分析未改动的区域受改动代码的影响可能性（概率），越有可能被影响的区域，其对应的测试用例越要被选入回归测试集，有助于尽早发现回归缺陷，而忽视那些受影响的可能性低或不受影响的测试用例。在允许的条件下，回归测试尽可能覆盖受到影响的部分。

③ **基于操作剖面选择测试**。软件操作剖面主要是指**软件功能点被用户的使用程度，**这种策略意味着最重要或最频繁使用功能的测试用例需要被选择、被执行。例如，根据 80/20 原则，20%的功能是用户 80%时间使用的功能，这部分 20%的功能所对应的测试用例需要被选择、被执行，保证基本功能不存在缺陷，如有缺陷，能被回归测试发现。

④ **再测试修改的部分**。当开发人员和测试人员对局部的代码修改有足够的信心时，可以通过代码的依赖性分析或根据丰富的经验判断这次修改的影响，将回归测试局限于被改变的模块和它的接口上，相当于没有进行回归测试，而只是**针对被修改的或新加的代码进行测试**。这种策略**效率最高**，但风**险也是最大的**。

第七章　专项测试

1 性能测试目标

- 获取系统**性能**某些**指标数据**
- 为了**验证**系统是否达到**用户提出的性能指标**
- **发现**系统中存在的**性能瓶颈**，**优化**系统的**性能**

2 什么是性能测试？

性能测试（performance test）就是**为了发现系统性能问题**或**获取系统性能相关指标**而进行的测试。一般在真实环境、特定负载条件下，通过**工具模拟**实际软件系统的运行及其操作，同时**监控性能各项指标**，最后对测试结果进行**分析**以**确定**系统的**性能状况**。

3 系统性能表现

系统性能指标包括：系统资源的使用率和系统性能表现。
系统性能表现的指标常见有：

(1) **请求响应时间**:客户端浏览器向 Web 服务器**提交一个请求**到**收到响应**之间的间隔时间。有些测试工具将请求响应时间表示为 TLLB(即 TimetoLast Byte),解释为从发起一个请求开始到客户端接收到最后 1 字节所耗费的时间。

(2) **事务响应时间**:事务可能由一系列请求组成,事务的响应时间就是**这些请求完成处理所花费的时间**。它是针对用户的业务而设置的,容易被用户理解。

(3) **数据吞吐量**:**单位时间内客户端和服务器之间网络上传输的数据量**，对于 Web 服务器，数据吞吐量可以理解为单位时间内 Web 服务器成功处理的 HTTP 页面或 HTTP 请求数量。

4 按照测试目的分，性能测试类型

(1) 性能**基准**测试：在**系统标准配置**下获得有关的**性能指标数据**，作为将来**性能改进**的**基线**(Baseline)

(2) 性能**验证**测试：**验证**系统是否达到**事先已定义**的**系统性能指标**、能否**满足**系统的**性能需求**

(3) 性能**规划**测试：在多种特定的环境下，获得**不同配置**的系统**性能指标**，从而**决定**系统部署的**软硬件配置**选型

(4) 容量测试可以看作性能的测试一种，因为系统的**容量**可以看作是**系统性能指标之一**

5 性能测试的基本过程

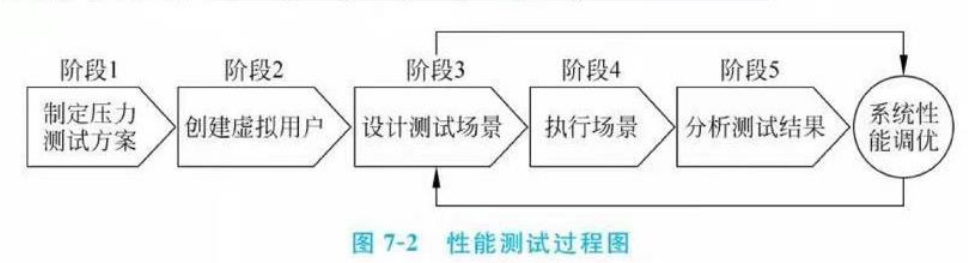


图 7-2　性能测试过程图

(1)**确定性能测试需求**，包括确定哪些性能指标要度量的，以及系统会承受哪些负载。一般来说,在某些关键业务操作情况下,系统的性能问题更容易出现。这些关键业务场景的确定可以看作性能测试的测试用例设计。如果这些测试用例通过了,也就说明这个系统的负载测试通过了。
(2)根据测试需求,**选择测试工具和开发相应的测试脚本**。一般针对选定的关键业务操作来开发相应的自动化测试脚本,并进行测试脚本的数据关联(如建立客户端请求和系统响应指标之间的关联)和参数化(把脚本中的某些请求数据替换成变量)。
(3)**建立性能测试负载模型**,就是确定并发虚拟用户的数量、每次请求的数据量、思考时间、加载方式和持续加载的时间等。设计负载模型通常不会一次设计到位,是一个不断迭代完善的过程,即使在执行过程中,也不是完全按照设计好的测试用例来执行,需要根据需求的变化进行调整和修改。
(4)**执行性能测试**。通过多次运行性能测试负载模型,获得系统的性能数据。一般要借助工具对系统资源进行监控和分析,帮助发现性能瓶颈,定位应用代码中的性能问题,切实解决系统的性能问题或在系统层面进行优化。
(5)**提交性能测试报告**,包括性能测试方法、负载模型和实际执行的性能测试、性能测试结果及其分析等。

6 什么是安全性测试

定义：安全性测试是一种软件测试方法，旨在识别和修复系统、应用程序或网络中的安全漏洞，以确保系统能够抵御恶意攻击、数据泄露和其他安全威胁。

7 渗透测试实施策略

（1）定义：采用探索式测试方式，**模拟黑客**可能使用的**攻击技术**和**漏洞发现技术**，对目标系统的安全做深入的探测，发现系统最脆弱的环节，直观地发现问题。
（2）实施策略

- **全程监控**：采用类似 wireshark 的嗅探软件进行全程**抓包**嗅探
- **择要监控**：对扫描过程不进行录制，仅仅在数据分析后，准备**发起渗透前**才**开启**软件进行**嗅探**。
- **主机监控**：仅监控**受测主机的存活状态**
- **指定攻击源**：**多方监控指定源**进程、网络连接、数据传输等
- **对关键系统**，可以采用对**目标的副本**进行渗透测试

8 软件安全性测试有哪两种？有什么关系和区别？

（1）**安全功能测试** (Security Functional Testing)：安全功能测试是指**验证**软件系统中实现的**安全功能**是否符合设计规范和安全需求，确保系统能够有效地提供预期的安全保护。此类测试主要关注软件的**安全特性**和**控制措施**。主要目标有数据机密性、完整性、可用性、不可否认性、身份认证、授权、访问控制、审计跟踪、委托、隐私保护、安全管理等。
（2）**安全漏洞测试** (Security Vulnerability Testing)：从攻击者的角度，以**发现**软件的**安全漏洞**为目的。安全漏洞是指系统在设计、实现、操作、管理上存在的可被利用的缺陷或弱点。
（3）关系

共同目标：两者都旨在提高软件系统的安全性，保护系统免受恶意攻击。
相辅相成：安全功能测试确保系统安全功能的正确实现，安全漏洞测试发现和修复实际存在的漏洞。两者结合使用能够全面提升系统的安全性。
（4）区别

- 测试目的：

安全功能测试： 验证安全功能的正确性和有效性。
安全漏洞测试： 发现并修复系统中的安全漏洞。

- 测试视角：

安全功能测试： 从开发和设计的角度出发，确保系统实现预期的安全功能。
安全漏洞测试： 从攻击者的角度出发，发现系统可能被利用的缺陷和弱点。

- 测试方法：

安全功能测试： 使用功能测试方法，验证各个安全特性的实现。
安全漏洞测试： 使用静态分析、动态分析、渗透测试和漏洞扫描等方法，模拟攻击和发现漏洞。

- 测试内容：

安全功能测试： 主要测试身份认证、授权、数据保护、审计等安全机制。
安全漏洞测试： 主要测试 SQL 注入、XSS、CSRF、缓冲区溢出等常见漏洞。

9 安全性测试的任务

(1) 全面检验软件在需求规格说明中规定的防止**危险状态**措施的有效性和在每一个危险状态下的反应。
(2) 对软件设计中用于**提高安全性的结构、算法、容错、冗余、中断处理等方案**，进行针对性测试。
(3) 在**异常条件**下测试软件，以表明不会因可能的单个或多个输入错误而导致不安全状态。
(4) 对**安全性关键的**软件单元、组件，单独进行加强的测试，以确认其满足安全性需求。

10 安全性测试方法按内外部分为哪两种？

(1) 基于**威胁**的方法：从软件**外部**考察其安全性，**识别**软件面临的安全**威胁**并测试其是否能够发生。
(2) 基于**漏洞**的方法：从软件**内部**考虑其安全性，**识别**软件的安全**漏洞**，如借助特定的漏洞扫描工具。

11 什么是 XSS 攻击和 sql 注入攻击，如何进行测试和防范

（1）XSS（Xcross-site Scripting 跨站点脚本攻击）：XSS 可以让**攻击者**在**页面访问者**的**浏览器**中**执行 JavaScript 脚本**,从而可以**获得**用户会话的**安全信息、插入恶意的信息**或**植入病毒**等。
（2）Sql 注入攻击：从客户端**提交特殊的代码**,**收集程序及服务器的信息**,从而**获取**必要的**数据库信息**,然后基于这些信息,可以注入某些参数,绕过程序的保护,针对**数据库服务器**进行攻击。

12 web 安全性测试可从哪些方法开展

(1) **数据加密**。某些数据需要进行信息加密和过滤后才能在客户端和服务器之间进行传输,

包括用户登录密码、信用卡信息等。数据加密的安全性还包括加密的算法、密钥的安全性。
(2) **登录或身份验证**。一般的应用站点都会使用登录或者注册后使用的方式,因此,必须对用户名和匹配的密码进行校验,以阻止非法用户登录。身份验证还包括调用者身份、数据库的身份、用户授权等，并区分公共访问和受限访问,受限访问的资源。
(3)**输入验证**。在进行 Web 安全性测试时,每个输入域都需要用标准的机制验证,长度、数据类型等符合设定要求,不允许输入 JavaScript 代码,包括验证从数据库中检索的数据、传递到组件或 Web 服务的参数等。
(4)**SQL 注入漏洞检测**。
(5)**超时限制验证**。Web 应用系统一般会设定“超时”限制,当用户长时间(如 15min)不做任何操作时,需要重新登录才能打开其他页面。会话(session)的安全性还包括交换会话标识符、会话存储状态等的安全性。
(6)**目录安全性**。Web 的目录安全也是不容忽视的。如果 Web 程序或 Web 服务器的处理不适当,可以通过简单的 URL 替换和推测,使整个 Web 目录暴露出来,带来严重的安全隐患。可以采用某些方法将这种隐患降低到最小程度,如每个目录下都存在 index.htm,以及严格设定 Web 服务器的目录访问权限。
(7)**操作留痕**。为了保证 Web 应用系统的安全性,**日志文件**是至关重要的。需要测试相关信息是否写进入了日志文件,是否可追踪。

13 什么是软件可靠性？可从哪几个指标度量？各自的定义

（1）软件可靠性：在规定的一段时间和条件下，软件能**维持其性能水平的能力**有关的一组属性。
（2）指标：可用**成熟性、容错性、易恢复性**三个基本子特性来度量。
（3）**成熟性**：是指**系统在规定时间内能够正常运行的能力**，即系统处于可用状态的时间比例。通过错误发现率 DDP（Defect Detection Percentage）来表现。DDP 越小，软件越成熟。DDP=测试发现的错误数量/已知的全部错误数量
（4）容错性：是指系统在**面对错误或异常情况**时，仍能保持**稳定运行**的能力，能够忽略或纠正错误，保证系统的可靠性和稳定性。
（5）**易恢复性**：是指系统在**遭受故障或错误**后，能够迅速**恢复到正常运行状态的能力**，最小化系统停机时间和数据丢失。

14 容错测试的要点？

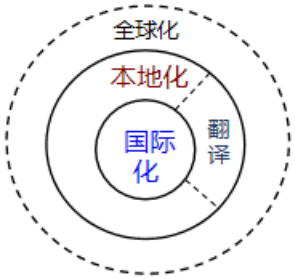
定义：容错测试是一种**对抗性的测试过程**。在这种测试中，通过各种手段让软件**强制**性地**发生故障**，或将**把应用程序或系统置于（模拟的）异常条件下**，以产生故障，例如设备输入/输出（I/O）故障或无效的数据库指针和关键字等
15 什么是 A/B 测试？有什么特点
α 测试是指**软件开发公司组织内部人员模拟**各类**用户**行为对即将面市软件产品（称为 α 版
本）进行测试，试图发现错误并修正。
经过 α 测试调整的软件产品称为 β 版本。紧随其后的 β 测试是指软件开发公司**组织**各
方面的**典型用户**在日常工作中实际使用 β 版本，并**要求用户报告**异常情况、提出批评意见。
然后软件开发公司再对 β 版本进行改错和完善。

第八章　软件本地化测试

1 软件本地化，国际化，全球化，相互关系

（1）**软件本地化（L10N）**：软件本地化是将一款软件产品**按特定国家或语言市场的需要**进行**全面定制**的过程，它包括翻译，重新设计，功能调整及测试，是否符合各个地方的习俗、文化背景、语言和方言的验证等。
（2）**软件国际化（I18n）**：指为保证所开发的软件**能适应全球市场的本地化工作**而**不需要对程序做任何系统性或结构性变化**的特性，这种特性通过特定的系统设计、程序设计、编码方法来实现。也就是说，完全符合国际化的软件产品，在对其进行本地化工作的时候，只要进行一些配置和翻译工作，而不需要修改软件的程序代码。
（3）**软件全球化（G11N）**：是一个概念化产品的过程，它基于全球市场考虑，以便**一个产品只做较小的改动就可以在世界各地出售**。全球化可以看作**国际化和本地化两者合成的结果**。
（4）关系

- I18N是L10N的基础和前提，为L10N做准备
- L10N是I18N向特定本地语言环境的转换
- I18N 是软件产品源语言开发的一部分，属于Engineering
- L10N 可以独立于Engineering，可由第三方完成



2 unicode 与 utf-x 关系，特点

字符集是操作系统中所使用的字符映射表
（1）Unicode：是一个**国际标准**，采用双字节对字符进行编码，提供了在世界主要语言中通用的字符，所以也称为基本多文种平面。Unicode 以明确的方式表述文本数据，简化了混合平台环境中的数据共享。
（2）UTF-X：UCS 只是规定如何编码，并没有规定如何传输、保存编码。所以有了 **Unicode 实用的编码体系**，如 UTF-8、UTF-16、UTF-32。UTF-8(UCS Transformation Format)和 ISO 8859-1 完全兼容，**解决了 Unicode 编码在不同的计算机之间的传输、保存**，使得双字节的 Unicode 能够在现存的处理单字节的系统上正确传输。

3 软件本地化基本步骤

(1) 建立配置管理体系，跟踪目标语言各个版本的源代码
(2) 创造和维护术语表
(3) 源语言代码和资源文件分离、或提取需要本地化的文本
(4) 把分离或提取的文本、图片等翻译成目标语言
(5) 把翻译好的文本、图片重新检入目标语言的源代码版本
(6) 如果需要，编译目标语言的源代码
(7) 测试翻译后的软件，调整 UI 以适应翻译后的文本

(8) 测试本地化后的软件，确保格式和内容都正确

4 本地化测试主要有哪些工作

(1) **功能性测试**，所有基本功能、安装、升级等测试；
(2) **翻译测试**，包括语言完整性、术语准确性等的检查；
(3) **可用性测试**，包括用户界面、度量衡和时区等；
(4) **兼容性调试**，包括硬件兼容性、版本兼容性等测试；
(5) **文化、宗教、喜好等适用性测试**
(6) **手册验证**，包括联机文件、在线帮助、PDF 文件等测试

5 软件本地化测试完整路线

(1) I18n 测试
(2) L10n 测试
(3) 语言/翻译的 测试
(4) 美观/界面 测试
(5) 功能 测试
(6) 发布 测试

第九章　软件测试自动化及其框架

1 自动化测试与测试自动化

（1）**自动化测试**：是指使用**自动化工具和脚本**来执行**测试用例**，以验证软件系统的功能、性能和稳定性。是把**以人为驱动的测试行为**转化为**机器执行**的一种过程，即**模拟**手工测试步骤，通过**执行**由程序语言编制的测试**脚本**，自动地完成软件的单元测试，功能测试、负载测试或性能测试等全部工作。
（2）**测试自动化**：意味着**测试全过程**的自动化和**测试管理工作**的自动化。

2 如何理解测试自动化

3 测试自动化实现的原理，几种技术

(1) **代码分析**：类似于高级编译系统，在工具中定义类/对象/函数/变量等定义规则、语法规则等，在分析时对代码进行语法扫描，找出不符合编码规范的地方。
(2) **脚本技术**：线性脚本、结构化脚本、数据驱动脚本、关键字驱动脚本
(3) **对象识别**：Windows 对象 、Mac 对象、Web DOM 对象
(4) **接口调用**
(5) **自动比较技术**：静态比较和动态比较，简单比较和复杂比较，敏感性测试比较和健壮性测试比较，比较过滤器
(6) **测试自动化系统的构成**：测试工具的分类、测试工具的选择、测试自动化普遍存在的问题、自动化测试的引入和应用

4 自动化测试的流程

(1) **测试环境**的搭建和设置，如自动上传软件包到服务器并完成安装。
(2) 基于模型实现**测试设计**的自动化，或基于软件设计规格说明书实现测试用例的自动生成。
(3) **脚本**自动生成，如根据 UML 状态图、时序图等生成可运行的测试脚本。
(4) **测试数据**的自动产生，例如，通过 SQL 语句在数据库中产生大量的数据记录，用于测试。
(5) **测试操作步骤**的自动执行，包括软件系统的模拟操作、测试执行过程的监控。
(6) **测试结果**分析，实际输出和预期输出的自动对比分析。
(7) **测试流程**(工作流)的自动处理，包括测试计划复审和批准、测试任务安排和执行、缺陷生命周期等自动化处理。
(8) **测试报告**自动生成功能等。

5 几种脚本技术

(1) **线性脚本**：是录制手工执行的测试用例得到的脚本，这种脚本包含所有的击键、移动、输入数据等，所有录制的测试用例都可以得到完整的回放。
(2) **结构化脚本**：类似于结构化程序设计，具有各种逻辑结构，包括选择性结构、分支结构、

循环迭代结构,而且具有函数调用功能。结构化脚本具有很好的可重用性、灵活性,所以结构化脚本易于维护。

(3) **数据驱动脚本**:将测试脚本和数据分离开来,测试输入数据存储在独立的(数据)文件中,而不是存储在脚本中。针对某些功能测试时,操作步骤是一样的,而输入数据是不一样的,相当于一个测试用例对应一种输入组合。

(4) **关键字驱动脚本**:是**数据驱动脚本**的**逻辑扩张**,实际上是**封装**了各种基本的操作,每个操作由相应的函数实现,而在开发脚本时,不需要关心这些基础函数,直接使用已定义好的关键字。这样的好处是脚本编写的效率会有很大的提高,脚本维护起来也很容易。而且,关键字驱动脚本构成简单,脚本开发按关键字来处理,可以看作是业务逻辑的文字描述,每个测试人员都能开发。

6 自动化功能测试基本构成

(1) 构建、存放程序软件包和测试软件包的**文件服务器**,在这个服务器上进行软件包的构建,并使测试工具可以存取这些软件包。

(2) 存储测试用例和测试结果的**数据库服务器**,提高过程管理的质量,同时生成统计所需要的数据。

(3) 执行测试的**运行环境**--测试试验室,一组测试用的服务器或 PC。单元测试或集成测试可能多用单机运行。但对于系统测试或回归测试,就极有可能需要多台机器在网络上同时运行。

(4) **控制服务器**,负责测试的执行、调度,从服务器读取测试用例,向测试环境中代理(Agent)发布命令。

(5) **Web 服务器**负责显示测试结果、生成统计报表、结果曲线;作为测试指令的转接点,接受测试人员的指令,向控制服务器传送。同时,根据测试结果,自动发出电子邮件给测试或开发的相关人员。Web 服务器,有利于开发团体的任何人员都可以方便地查询测试结果,也方便测试人员在自己办公室就可以运行测试。

(6) **客户端程序**,测试人员在自己机器安装的程序,许多时候,要写一些特殊的软件来执行测试结果与标准输出的对比工作或分析工作,因为可能有部分的输出内容是不能直接对比的,就要用程序进行处理。

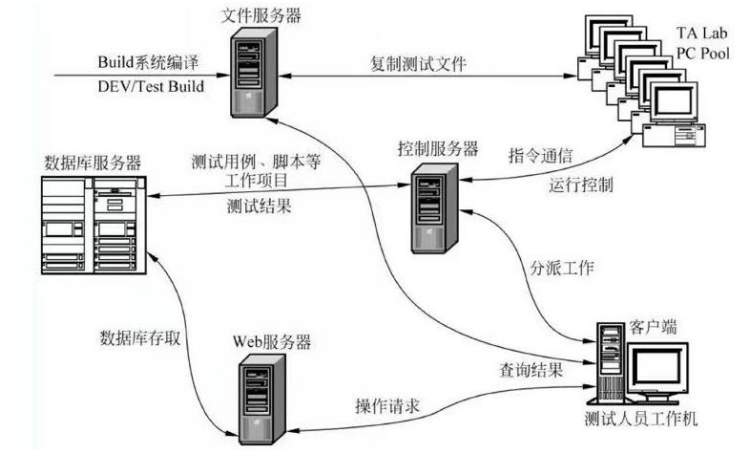


图 9-2 测试自动化的基本结构

7 TA 框架的构成及各部分特点

(1) Harness/IDE:TA 框架的**核心**,相当于“夹具”,其他 TA 框架的组成部分都能作为插件与

之集成,而且承担脚本的创建、编辑、调试和管理等。
(2) TA **脚本**的管理：包括公共脚本库、项目归类的脚本库,这部分可以与 GitHab 这样的代码管理工具集成。
(3) 测试**资源**管理：增加、删除和配置相应的测试设备(软硬件资源),并根据它们的使用状态来分配测试资源。
(4) 测试**数据**管理：测试数据的自动生成、存储、备份和恢复等。
(5) **开放的接口**：提供给其他持续集成环境,开发环境或其他测试环境的集成接口
离\旱
(6) **代理(Agents)**：负责 Harness 与工具的通信,控制测试工具的运行。
(7) **任务安排(Scheduler)**：安排和提交定时任务、事件触发任务等,以便实现无人值守的自动化测试执行。
(8) **数据统计分析**：针对测试结果(含测试工具运行产生的日志),生成可读性良好的**测试报告**(如 HTML 格式的测试结果)

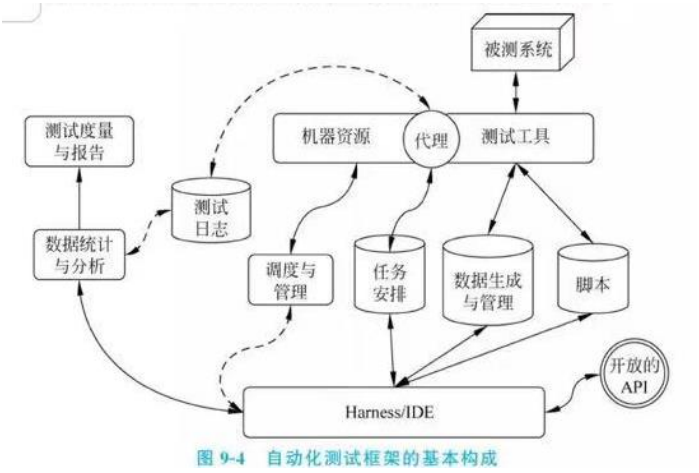


图 9-4 自动化测试框架的基本构成