软件测试综述

软件测试综述

软件测试是一个必不可少的活动，是对软件需求分析、设计规约和编码的最终复审；是软件质量保证的关键步骤。

软件测试是根据软件开发各阶段的规约和软件的内部结构，精心设计一批测试用例（包括输入数据及其预期的输出结果），并利用这些测试用例去运行程序，以发现软件中不符合质量特性要求（即缺陷或错误）的过程。

软件测试综述

- 软件测试的背景
- 软件测试概述
- 软件测试的模型

软件测试的背景

- 软件错误和软件失效的案例
- 软件缺陷是什么
- 软件缺陷分类
- 软件缺陷的产生
- 软件缺陷修复的代价
- 软件可靠性问题
- 软件错误数估算

软件错误和软件失效的案例

- 火星登陆事故 1999年12月3日，缺少集成测试
- 爱国者导弹防御系统, 1991，多哈，死亡28人，缺少稳定性测试
- 英特尔奔腾芯片缺陷，1994，对待缺陷的态度

  （4195835 / 3145727）×3145727– 4195835
- 美迪斯尼公司的狮子王游戏软件bug 1994年圣诞节前，缺少兼容性测试

- 记事本 联通

软件错误和软件失效

- “错误”术语

  “错误”这一术语。在没有特别加以说明的情况下，这是一个泛用的、模糊的概念。

  它指的可能是bug(差错)、 fault(故障)、error(错误)、failure(失效)、crash(重大事故)、problem(疑问)等。

  在汉译中，这些术语的使用更加混乱。

软件缺陷是什么？

- 软件出错机理可描述为:软件错误，软件缺陷，软件故障，软件失效。
  - 软件错误(error)：是指在软件生存期内的不希望或不可接受的人为错误，其结果是导致软件缺陷的产生。
  - 软件缺陷(bug)：是存在于软件（文档、数据、程序）之中的那些不希望或不可接受的偏差。其结果是软件运行于某一特定条件时出现软件故障，这时称软件缺陷被激活。
  - 软件故障(fault)：是指软件运行过程中出现的一种不希望或不可接受的内部状态。此时若无适当措施（容错）加以及时处理，便产生软件失效。
  - 软件失效(failure)：是指软件运行时产生的一种不希望或不可接受的外部行为结果。

软件缺陷是什么？

- 错误(error)可能转化为缺陷(bug)，也可能不会。缺陷可能导致系统故障(fault)或失效(failure) ，也可能不会。举例：

  有一行代码：if a>0 then do…,程序员犯了错误(error) ，写成了 if a>=0 then do…, 但是由于某些外部限制，a=0的情况不可能出现，所以这个错误也就不具备变成缺陷(bug)的条件。

  另有一行代码：if a=0.83975, then do…, 程序员犯了错误，写成了 if a=0.93975, then do…, 且输入值完全有可能是0.83975 或 0.93975， 所以具备条件成了缺陷（bug），但由于出现该输入值的几率非常之小，以至于一都未发所生过，也就不能成为故障(fault)或失效(failure)。

软件缺陷激活条件

- 符合下列五种情况之一就可认为是软件缺陷（出错）：

  1）软件未达到软件产品需求说明书指明的要求。

  2）软件出现了软件产品需求说明书指明不会出现的错误。

  3）软件功能超出软件产品需求说明书指明的范围。

  4）软件未达到软件产品需求说明书虽未指明但应达到的要求。

  5）软件测试人员认为难以理解、不易使用、运行速度缓慢、或者最终用户认为不好的问题。

- 软件缺陷的特征
  - “看不到” ——软件的特殊性决定了缺陷不易看到
  - “看到但是抓不到”

    ——发现了缺陷，但不易找到问题发生的原因所在

软件缺陷分类

- 可从不同角度对软件缺陷进行分类：
  - 按错误的影响和后果分类
  - 按错误的性质和范围分类
  - 按软件生存期阶段分类

软件错误分类-按错误的影响和后果分类

- 按错误的影响和后果分类
  1. 较小错误：只对系统输出有一些非实质性影响。如，输出的数据格式不合要求等。
  2. 中等错误：对系统的运行有局部影响。如输出的某些数据有错误或出现冗余。
  3. 较严重错误：系统的行为因错误的干扰而出现明显不合情理的现象。如开出了0.00元的支票，系统的输出完全不可信赖。
  4. 严重错误：系统运行不可跟踪，一时不能掌握其规律，时好时坏。
  5. 非常严重的错误：系统运行中突然停机，其原因不明，无法软启动。
  6. 最严重的错误：系统运行导致环境破坏，或是造成事故，引起生命、财产的损失。

软件错误分类-按错误的性质和范围分类

- 按错误的性质和范围分类

  B.Beizer从软件测试观点出发，把软件错误分为5类。

  1.功能错误

  1）规格说明错：规格说明可能不完全，有二义性或自身矛盾。

  2）功能错误：程序实现的功能与用户要求的不一致。这常常是由于规格说明中包含错误的功能、多余的功能或遗漏的功能所致。

  3）测试错误：测试的设计与实施发生错误。软件测试自身也可能发生错误。

  4）测试标准引起的错误：对软件测试的标准要选择适当，若测试标准太复杂，则导致测试过程出错的可能就大。

软件错误分类-按错误的性质和范围分类

2.系统错误

1）外部接口错误：外部接口指如终端、打印机、通信线路等系统与外部环境通信的手段。

2）内部接口错误：内部接口指程序之间的联系。

3）硬件结构错误 ：不能正确理解硬件如何工作而导致的出错。

4）操作系统错误：主要是由于不了解操作系统的工作机制而导致出错。当然，操作系统本身也有错误。

5）软件结构错误：由于软件结构不合理或不清晰引起的错误。

6）控制与顺序错误：如存在不正确的处理步骤等。

7）资源管理错误：由于不正确使用资源而产生的。

软件错误分类-按错误的性质和范围分类

3.加工错误

1）算术与操作错误：指在算术运算、函数求值和一般操作过程中发生的错误。

2）初始化错误：如，忘记初始化工作区、寄存器和数据区；对循环控制变量赋错初值；用不正确的格式，数据或类型进行初始化等。

3）控制和次序错误：与系统级同名错误类似，但它是局部错误。如遗漏路径；不可达到的代码等；

4）静态逻辑错误：主要包括：不正确地使用CASE语句；在表达式中使用不正确的条件（例如用“＞”代替“＜”的否定）；对情况不适当地分解与组合；混淆“或”与“异或”等。

软件错误分类-按错误的性质和范围分类

4.数据错误

1）动态数据错误 ：动态数据是在程序执行过程中暂存的数据。

2）静态数据错误：静态数据直接或间接地出现在程序或数据库中，在内容和格式上都固定。一般由预处理出错造成。

3）数据内容错误：由内容被破坏或被错误地解释而造成的错。

4）数据结构错误：数据结构错误主要包括结构说明错误及把一个数据结构误当做另一类数据结构使用的错误。

5）数据属性错误：如错把整数当实数，允许不同类型数据混合运算而导致的错误等。

软件错误分类-按错误的性质和范围分类

5.代码错误

主要包括：语法错误；打字错误；对语句或指令不正确理解所产生的错误。

软件错误分类-按软件生存期阶段分类

- 把软件的逻辑错误按生存期不同阶段分为4类。

1.问题定义（需求分析）错误

在软件定义阶段，分析员研究用户的要求后所编写文档中出现的错误。换句话说，这类错误是由于问题定义不满足用户的要求而导致的错误。

软件错误分类-按软件生存期阶段分类

2.规格说明错误

这类错误是指规格说明与问题定义不一致所产生的错误。它们又可以细分成：

1）不一致性错误：规格说明中功能说明与问题定义发生矛盾。

2）冗余性错误：规格说明中某些功能说明与问题定义相比是多余的。

3）不完整性错误：规格说明中缺少某些必要的功能说明。

4）不可行错误：规格说明中有些功能要求是不可行的。

5）不可测试错误：有些功能的测试要求是不现实的。

软件错误分类-按软件生存期阶段分类

3.设计错误

设计阶段产生的错误，它使系统的设计与需求规格说明中的功能说明不相符。它们又可以细分为：

1）设计不完全错误：某些功能没有被设计，或设计得不完全。

2） 算法错误：算法选择不合适。主要表现为算法的基本功能不满足功能要求、算法不可行或者算法的效率不符合要求。

3） 模块接口错误 ：模块结构不合理 ；模块与外部数据库的界面不一致，模块之间的界面不一致。

4） 控制逻辑错误 ：控制流程与规格说明不一致 ；控制结构不合理。

5） 数据结构错误：数据设计不合理；与算法不匹配；数据结构不满足规格说明要求。

软件错误分类-按软件生存期阶段分类

4.编码错误

多种多样，大体归为以下几种：

数据说明错、数据使用错、计算错、比较错、控制流错、界面错、输入/输出错，及其它的错误。

在不同的开发阶段，错误的类型和表现形式不同，故应采用不同的方法和策略来进行检测。

软件缺陷的产生

- 软件缺陷的产生

  比较容易造成软件缺陷的主要因素,归纳如下:

  - 技术问题

    算法错误、语法错误、计算和精度问题、系统结构不合理、接口参数不匹配。

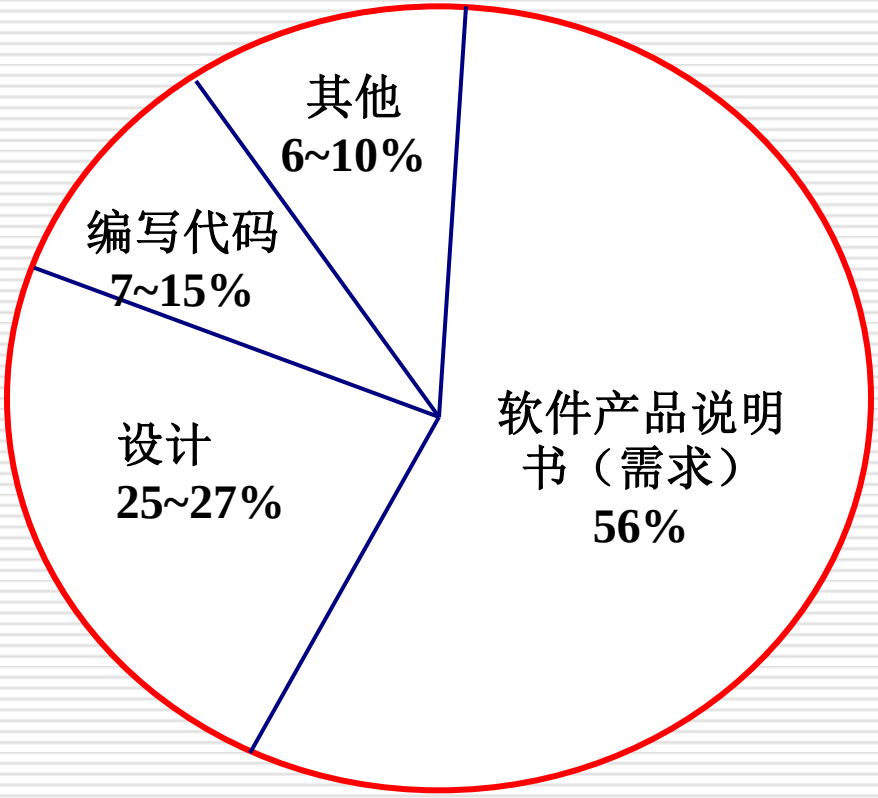
  - 团队工作

    需求不清、开发人员相互理解不一致、设计或编程上的假定或依赖性没得到充分沟通。

  - 软件本身

    文档错误、大数据量错误、边界错误、时序错误、数据恢复能力没有或不够、软硬件上的错误、软件开发标准或过程上的错误。

软件缺陷的构成



软件错误产生的原因分布

软件缺陷的产生

- 软件需求说明书为什么是软件缺陷存在最多的地方，主要原因有：
  - 用户的计算机知识较少
  - 要开发产品的特性不够清晰
  - 需求变化的不一致
  - 对需求说明书不重视
  - 项目组成员间缺少沟通

软件缺陷的产生

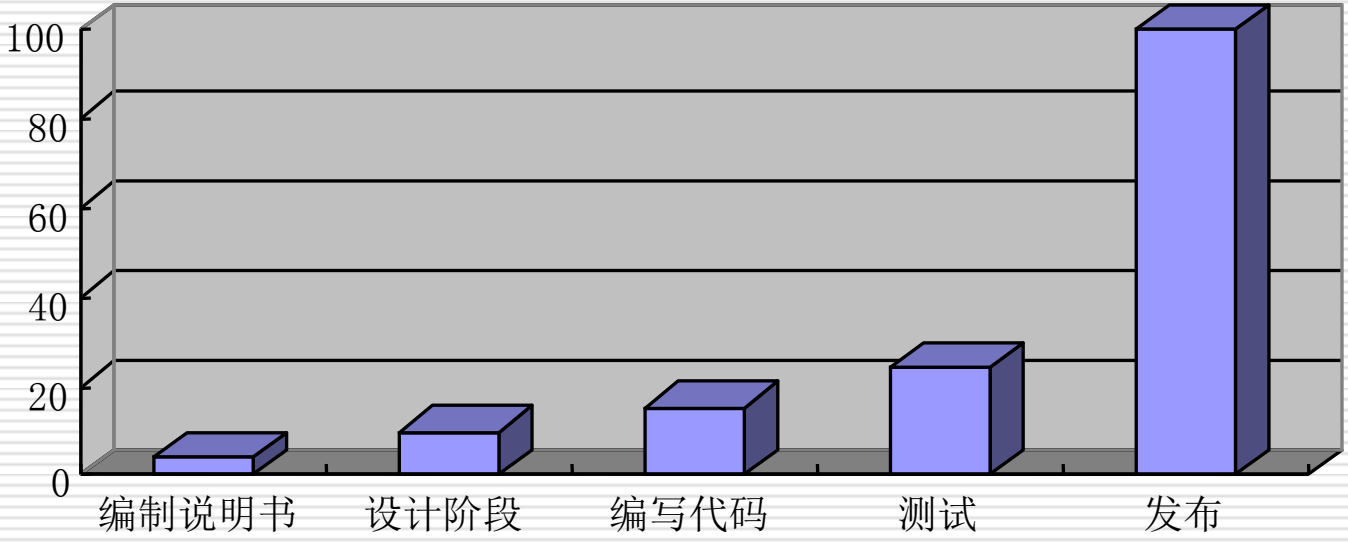
软件缺陷的状态

为便于跟踪和管理某产品的缺陷，可以定义不同的软件缺陷状态：

- 激活状态(Active或Open)：问题还没解决，测试员报的bug、或验证后bug仍然存在。
- 已修正状态(Fixed或Resolved)：开发人员针对缺陷，修改程序，认为已解决问题，或通过单元测试。
- 关闭或非激活状态(Close或Inactiv)：测试员验证Fixed bug后，确认bug已改的状态。
- Hold状态：第三方产品引起的、或是无法解决的bug。
- Differed状态：不需解决或准备在下版中解决的bug。

软件错误修复的代价

- 软件在从计划、编制、测试、一直到交付用户公开使用的过程中，都有可能产生和发现错误。随着整个开发过程的时间推移，修复软件的费用呈几何级数的增长。下图是软件错误在不同阶段发现时修改的费用示意图



软件可靠性问题

- 软件可靠性定义

IEEE将软件可靠性定义为：在给定时间间隔内和特定的环境下，软件按规格说明成功运行的概率。

给定的时间间隔：在定义中，一般采用“运行时间” t 作为时间的尺度。

环境条件：指的是软件的使用环境。无论是什么软件，如果不对它的使用环境加以限制，都会失效。这种失效的数据，不能用来度量软件的可靠性。

成功地运行：指不仅程序能正确运行，满足用户对它的功能要求， 而且当程序一旦受到意外的伤害，或系统故障时，能尽快恢复，仍能正常地运行。

软件可靠性问题

- 软件可靠性的主要指标

借用硬件可靠性的定量度量方法来度量软件的可靠性：

MTBF:平均故障间隔时间

MTTF:平均故障时间

$$MTTF = \frac{1}{n}\sum_{i=1}^{n} t_i$$

t1，t2，.....，tn:失效时间

软件可靠性问题

- 因软件设计故障与因计算机硬件设计故障而引发的系统 失效的比例大约是：10:1
- 运行软件的驻留故障密度（每千行代码的故障数目）：

——要求很高的关键财务或财产软件为：每千行代码 1～10个故障

——关键的生命软件为：每千行代码0.01～1个故障

软件可靠性是对软件在设计、开发以及所预定的环境下具有能力的置信度的一个度量，是衡量软件质量的主要参数之一。

软件测试则是保证软件质量、提高软件可靠性的最重要手段。

软件错误数估算

- 植入故障法估算（ 捕获—再捕获抽样法）

  利用植入故障法估算程序中原有故障总数E_T

  设N_s 是在测试前人为地向程序中植入的故障数(称播种故障)，

  n_s 是经过一段时间测试后发现的播种故障的数目，

  n 是在测试中又发现的程序原有故障数。

  设测试用例发现植入故障和原有故障的能力相同，则程序中原有故障总数 N ($=E_T$)估算值为

$$N = \frac{N_s}{n_s} n$$

软件错误数估算

- Hyman分别测试法

由两个测试员同时互相独立测试同一程序的两个副本，用 t 表示测试时间，记

t＝0时，程序中原有故障总数是 B0；

t＝t1 时，测试员甲发现的故障总数是 B1；测试员乙发现的故障总数是 B2；其中两人发现的相同故障数目是 bc；两人发现的不同故障数目是 bi。

在大程序测试时，开始两个测试员测试的结果应当比较接近，bi 不是很大。这时有

$$B_0 = \frac{B_1 \cdot B_2}{bc}$$

软件错误数估算

- Hyman分别测试法（续）

  如果bi很大，应当每隔一段时间，由两个测试员再分别测试，分析测试结果，估算B0。如果bi减小，或几次估算值的结果相差不多，则B0作为原有错误总数的估算值。

软件测试概述

- 软件测试的发展
- 软件测试的定义
- 软件测试的重要性
- 软件测试的分类
- 软件测试的目的和原则
- 软件测试活动
- 软件测试技术概要
- 软件测试误区
- 测试员应有的素质

软件测试的发展

- 软件测试发展史上的几个重要事件
  - 1972年6月，首次软件测试会议

    1972年6月，Bill Hetzel（代表论著《The Complete Guide to Software Testing》）在美国的北卡罗来纳（North Carolina）大学组织了首次以软件测试为主题的会议。
  - 1973年， Bill Hetzel定义软件测试概念

    就是建立一种信心，认为程序能够按预期的设想运行。

    1983年，Bill Hetzel将定义修订为：评价一个程序和系统的特性或能力，并确定它是否达到预期的结果。软件测试就是以此为目的的任何行为。他还把软件的质量定义为“符合要求”。

软件测试的发展

- 软件测试发展史上的几个重要事件（续）
  - 1979年，Glenford Myers：《The Art of Software Testing》出版。

    这本书是软件测试方面的圣经。Myers定义及诠释的测试方法论已成为软件测试的基本模块。提出测试的目的是证伪。
  - 1983、1990年，IEEE/ANSI标准定义软件测试概念

    1990年的IEEE/ANSI标准将软件测试进行了这样的定义：在既定的状况条件下，运行一个系统或组建，观察记录结果，并对其某些方面进行评价的过程。

    这里所谓“既定的状况”可理解为需求或设计。

软件测试的发展

- 软件测试发展史上的几个重要事件（续）
  - 1996年提出：测试能力成熟度TCMM（Testing Capability Maturity Model）、测试支持度TSM（Testing Support Model）、测试成熟度（Testing Maturity Model）。

软件测试的发展

- 软件测试发展趋势
  - 测试与质量保证体系的融合
  - 测试方法越来越细分

    测试方法的细分，如网站测试、安全性测试等；
  - 测试技术不断发展
    - 软件验证技术方面
    - 软件静态测试方面
    - 测试用例的选择方面
    - 自动化测试方面
  - 测试走向专业化道路

软件测试的定义

- 一般定义
  - IEEE的定义
  - Myers的定义
- 广义定义
- 相关术语

一般软件测试的定义

- 定义1：

  1983年，IEEE提出的软件工程标准术语中给软件测试下的定义：

  - 软件测试是使用人工的或自动的手段来运行或检测某个系统的过程, 其目的在于检验它是否满足约定的需求或是比较预期结果与实际结果之间的差别。

  1990年的IEEE/ANSI标准将软件测试进行了这样的定义：(IEEE/ANSI, 1990 [Std 610.12-1990])

  - 就是在既定的状况条件下，运行一个系统或组建，观察记录结果，并对其某些方面进行评价的过程。(这里所谓“既定的状况”也可理解为需求或设计)

  这一定义非常明确地提出了软件测试以检验是否满足需求为目标。

一般软件测试的定义

- 定义2：

  Glenford J.Myers在其1979年《软件测试技巧》（“The Art of Software Testing”）一书中对软件测试的定义是：

  - 软件测试是为了发现错误而运行程序的过程。

  这一定义明确指出软件测试的目的是“发现错误”。

广义软件测试的定义

- 广义的软件测试是由确认、验证、测试3方面组成。
  - 确认：评估将要开发的软件产品是否“正确无误”、可行和有价值的。确认意味着确保一个开发软件是“正确无误的”，是对软件开发构想的检测。
  - 验证：检测软件开发的每个阶段、每个步骤的结果是否“正确无误”，是否与软件开发各阶段的要求或期望的结果相一致。验证意味着确保软件会“正确无误”地实现软件的需求，开发过程是沿着正确的方向进行的。
  - 测试：与狭隘的测试概念统一。

软件测试的定义 -相关术语

- 测试用例

  所谓测试用例是为特定的目的而设计的一组测试输入、执行条件和预期的结果；测试用例是执行测试的最小实体。

- 测试步骤

  测试步骤详细规定了如何设置、执行、评估特定的测试用例。

- 精确和准确

  准确是指得到的测试结果与真实值之间的接近程度。

  精确是指同样环境下重复测测试所得到的结果之间的重现性。

- 确认和验证

  确认是保证软件符合产品说明书的过程。

  验证是保证软件满足用户要求的过程。

软件测试的对象

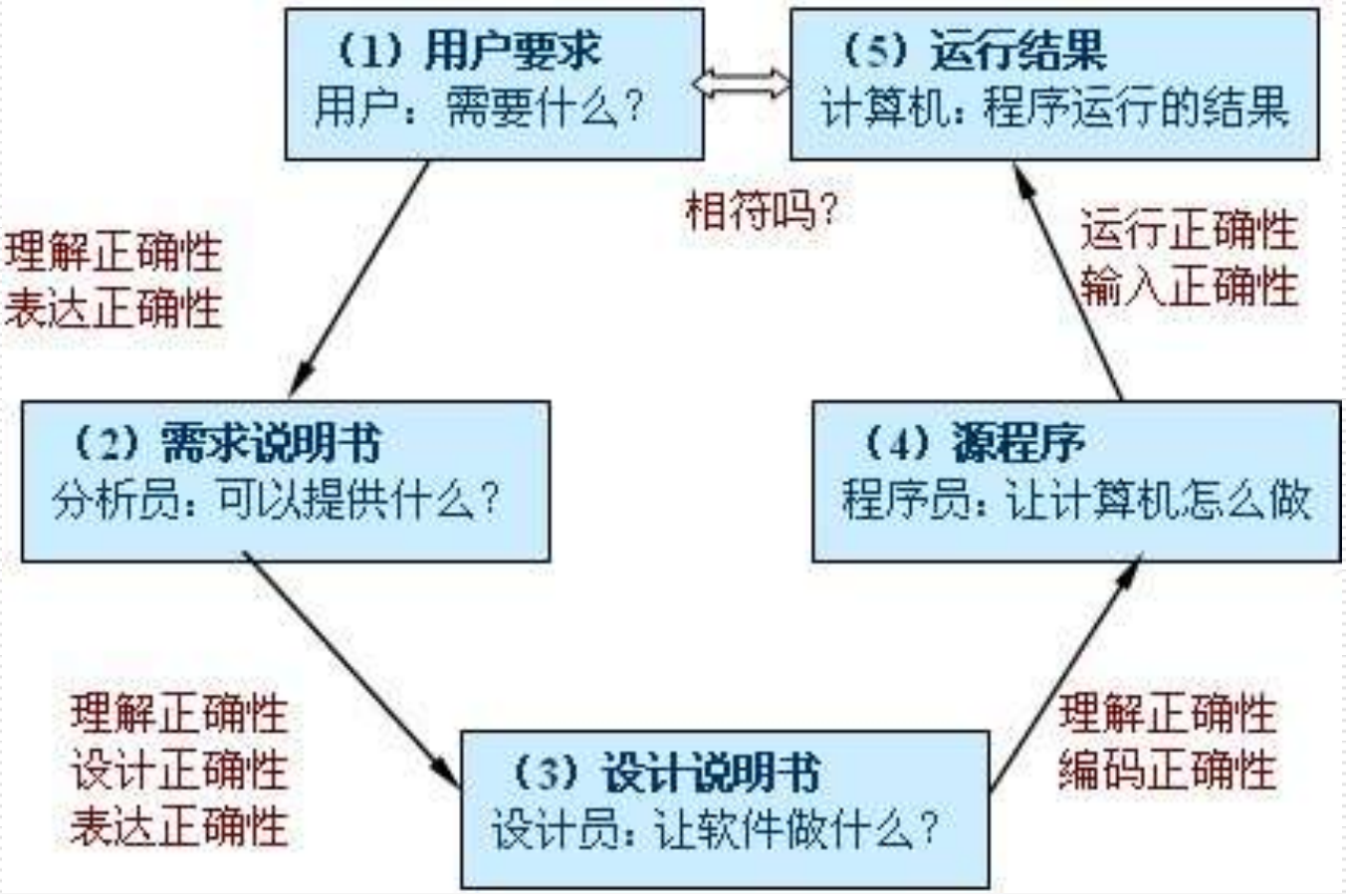
- 软件测试的对象

  软件测试并不等于程序测试。

  软件测试应该贯穿整个软件定义与开发整个期间。

  因此需求分析、概要设计、详细设计以及程序编码等各阶段所得到的文档，包括需求规格说明、概要设计规格说明、详细设计规格说明以及源程序，都应该是软件测试的对象。

软件测试的对象


（1）用户要求
用户：需要什么？
（5）运行结果
计算机：程序运行的结果
相符吗？
理解正确性
表达正确性
运行正确性
输入正确性
（2）需求说明书
分析员：可以提供什么？
（4）源程序
程序员：让计算机怎么做
理解正确性
设计正确性
表达正确性
（3）设计说明书
设计员：让软件做什么？
理解正确性
编码正确性

软件测试的对象

上图中，在对需求理解与表达的正确性、设计与表达的正确性、实现的正确性以及运行的正确性的验证中，任何一个环节发生了问题都可能在软件测试中表现出来。

软件测试的重要性

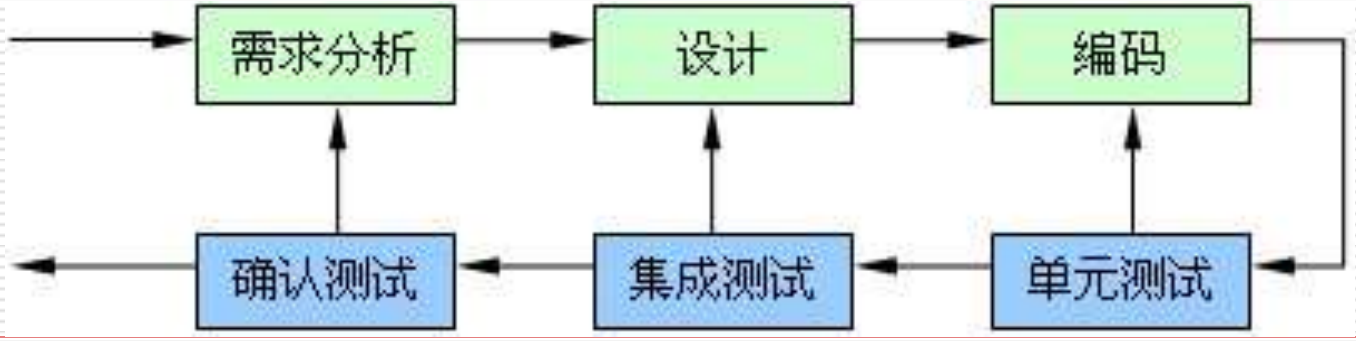
- 软件错误不可避免

  由于人的思维局限性，再加上开发的系统具有的复杂性，因此决定了在开发过程中出现软件错误是不可避免的。

  软件错误并不一定是由编码所引起的，很可能是详细设计，概要设计阶段，甚至于是需求分析阶段的问题引起的。

  若能及早排除开发中的错误，就可排除给后期工作带来的麻烦，降低出错代价。

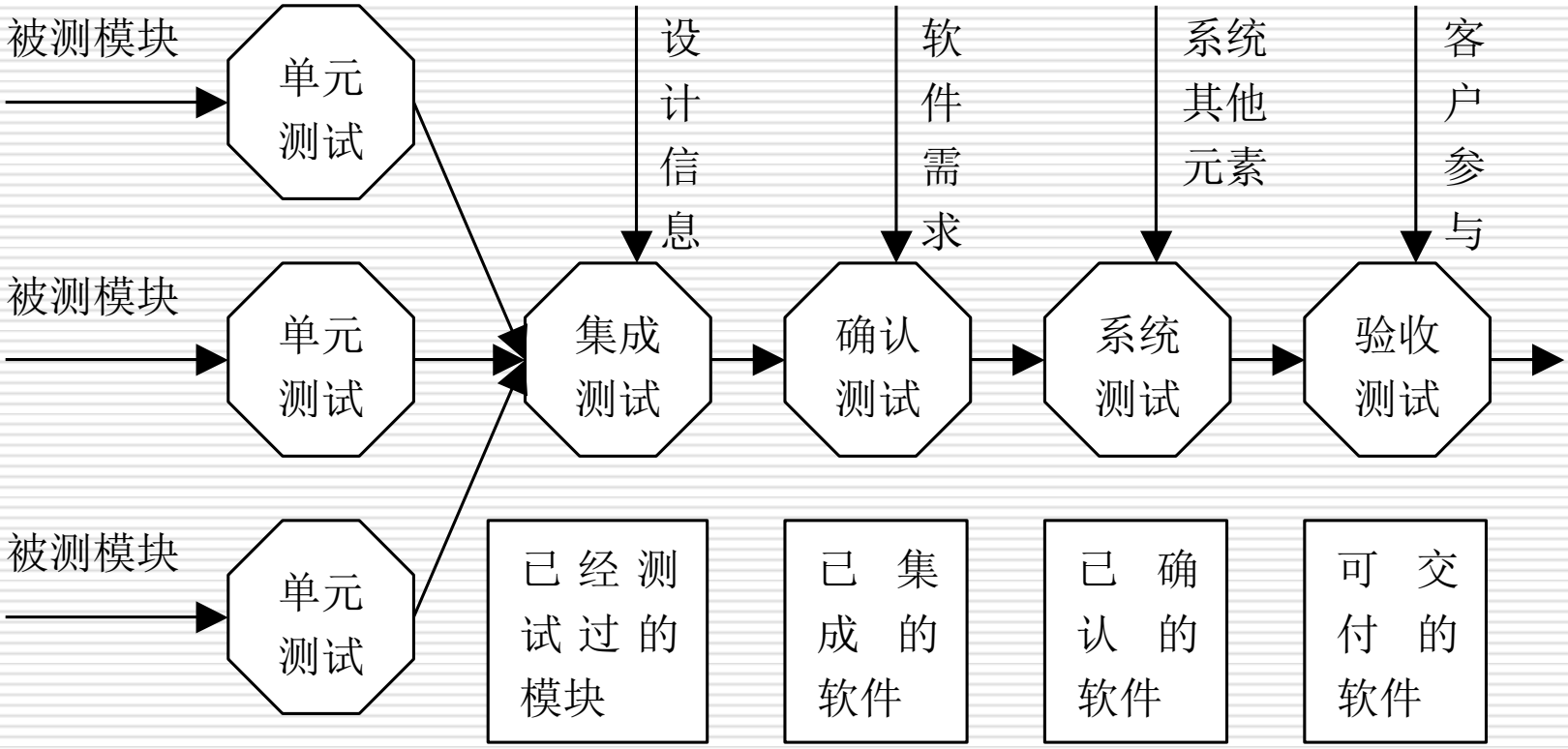
- 软件测试应无处不在



软件测试的分类

- 按测试过程 （开发阶段）
  - 单元测试：又称模块测试，是针对软件设计的最小单位——程序模块进行正确性检验的测试工作。
  - 集成测试：又称组装测试，是将模块按照设计要求组装起来进行测试，主要目标是发现与接口有关的问题。
  - 确认测试：验证软件的功能和性能及其它特性是否与用户的要求一致。
  - 系统测试：是在集成测试通过后进行，目的是充分运行系统，验证各子系统是否都能正常工作并完成设计的要求。
  - 验收测试：用户为主，开发人员参与，以规格说明书为蓝本的测试。

软件测试的分类


被测模块
单元测试
被测模块
单元测试
被测模块
单元测试
设计信息
软件需求
系统其他元素
客户参与
集成测试
确认测试
系统测试
验收测试
已经测试过的模块
已集成的软件
已确认的软件
可交付的软件

软件测试的分类

- 按测试用例设计方法
  - 白盒测试

    也称结构测试或逻辑驱动测试。它是从程序的控制结构出发进行的测试，测试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。
  - 黑盒测试

    又称功能测试、数据驱动测试或基于规格说明书的测试，是一种从用户观点出发的测试，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。
  - 灰盒测试

    是介于白盒测试与黑盒测试之间的测试，灰盒测试关注输出对于输入的正确性，同时也关注内部表现，但这种关注不象白盒那样详细、完整，只是通过一些表征性的现象、事件、标志来判断内部的运行状态。

软件测试的分类

- 按实施对象：
  - Alpha测试（企业内部测试）：是由用户在开发环境下进行的测试，也可以是公司内部的用户在模拟实际操作环境下进行的受控测试。
  - Beta测试（最终用户测试）：是软件的多个用户在实际使用环境下进行的测试。
  - 第三方测试（独立测试）

软件测试的分类

- 按执行方式
  - 人工测试：手工执行的测试；
  - 自动化测试：希望能够通过自动化测试工具或其他手段,按照测试工程师的预定计划进行自动的测试。如：负载测试、性能测试、可靠性测试等。

软件测试的分类

- 按测试方式划分
  - 静态测试

    静态测试方法的主要特征是在用计算机测试源程序时，计算机并不真正运行被测试的程序，只对被测程序进行特性分析。

    静态测试常称为“分析”，静态分析是对被测程序进行特性分析的一些方法的总称。

  - 动态测试

    动态测试方法的主要特征是计算机必须真正运行被测试的程序，通过输入测试用例，对其运行情况(输入/输出的对应关系)进行分析。

软件测试的分类

- 按测试形态(Testing Types)(台湾许育诚的一种分法)：
  - 建构性测试(Construction Testing)：当程序还是处于建设阶段时所进行的测试；是属于前置性的测试,它主要是偏重于程序端的功能测试,以确保程序执行运行正常。
  - 系统测试(System Testing) :是针对系统的行为来做测试；是属于中后期的整合测试,所进行的测试是以使用者的观点为主,也就是模拟外界世界的使用者会如何的使用产品。
  - 特殊测试(Special Testing) :根据产品的本质特性来安排或剔除特殊测试

软件测试的分类

建构性测试包括：

单步测试、尝试性测试、单元测试、组件测试、集成测试等

系统测试包括：

集成测试、前哨测试、功能测试、设置测试、发行测试、验收测试等

特殊测试包括：

回归测试、压力测试、兼容性测试、性能测试、Alpha和Beta测试

软件测试的目的

- 软件测试的目的决定了如何去组织测试。

  如果测试的目的是为了尽可能多地找出错误，那么测试就应该直接针对软件比较复杂的部分或是以前出错比较多的位置。

  如果测试目的是为了给最终用户提供具有一定可信度的质量评价，那么测试就应该直接针对在实际应用中会经常用到的商业假设。

软件测试的目的

- Grenford J. Myers在《The Art of Software Testing》一书中的观点：

  ① 软件测试是程序的执行过程，目的在于发现错误；

  ② 测试是为了证明程序有错，而不是证明程序无错误。

  ③ 一个好的测试用例是在于它能发现至今未发现的错误；

  ④ 一个成功的测试是发现了至今未发现的错误的测试。

软件测试的目的

- 对 Myers 观点的解释

  测试要以查找错误为中心，而不是为了演示软件的正确功能。但发现错误并不是软件测试的唯一目的。没有发现错误的测试也是有价值的，完整的测试是评定测试质量的一种方法。详细而严谨的可靠性增长模型可以证明这一点。例如 Bev Littlewood发现一个经过测试而正常运行了n小时的系统有继续正常运行n小时的概率。

软件测试的原则

- 七大测试原则
  - 原则1-测试只是展示缺陷

    测试只能表明缺陷存在，却不能证明没有缺陷。测试能降低未发现缺陷留存的概率，却不能证明软件是绝对正确的。

    正如某些数学命题，你可以穷举1~ n，证明其正确，却依然无法证明对于n+1仍然正确。
  - 原则2-穷尽测试是不可能的

    测试所有的输入和条件组合是不可能的，除非是极其简单的情况。可以取而代之的是基于风险和优先级的测试。

    当不懂装懂的老板要求你彻底测试一个软件的时候，这是你反驳的最好支持，当然要说的委婉一点。

    （续...）

软件测试的原则

- 原则2-穷尽测试是不可能的（续）

  可能的原因：

  （1）测试所需要的输入量太大

  （2）测试的输出结果太多

  （3）软件实现的途径太多

  （4）软件规格说明没有一个客观标准

  软件测试是有风险的行为，要针对风险做出抉择：

  就是如何将无边无际的可能性减小到一个可控的范围，以及如何针对软件风险做出恰当选择，去粗存精，找到最佳的测试量，使得测试工作量不多也不少，既能达到测试的目的，又能较为经济。

  （续）

软件测试的原则

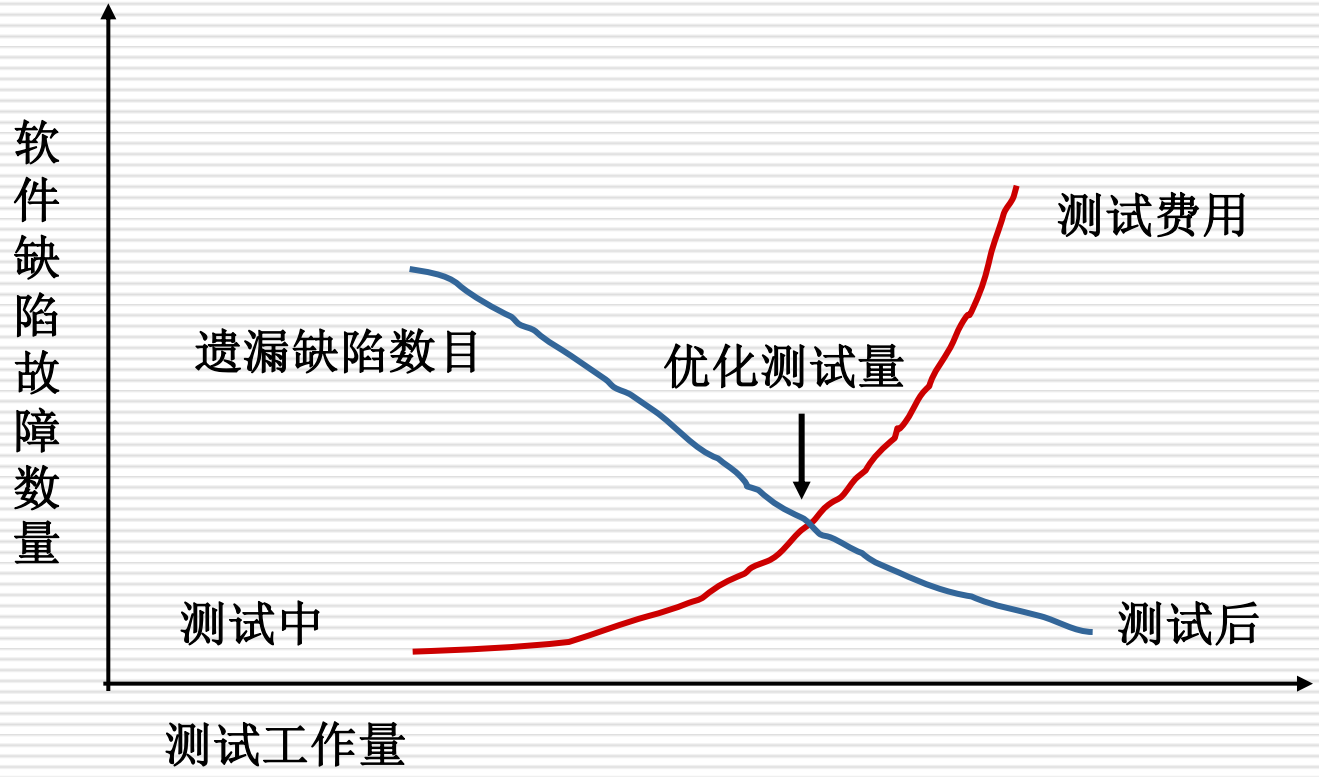


图　测试工作量和软件缺陷数量之间的关系

软件测试的原则

- 原则3-早期测试

要较早发现缺陷，就要在软件周期尽可能早的时候开始测试，而且要专注于已定义的测试目标。

尽早开始测试！这句话估计早就把大家的耳朵磨起茧了。为什么要早？因为越早发现问题，解决的代价就越小。

- 原则4-缺陷簇生

要对缺陷发现率高的模块投入更多的测试。少量的模块往往隐藏了大部分的缺陷。

这不仅仅是所谓的物以类聚。缺陷发现率高的模块往往于需求不清，设计不当，编码复杂度高，程序员情绪不佳、心情不好、往往犯同样的错误等内在原因关联，所以从风险的角度来看必然较高，多花些时间绝对值得。

软件测试的原则

- 原则5-杀虫剂悖论

相同的测试再重复多次后就无法再找到缺陷了。要克服“杀虫剂悖论”，测试用例要不断评审修改，不断添加新的和不同的测试，就有可能找到更多缺陷。

随着对系统的加深理解，必然会有更多的测试用例产生。另外缺陷本身也是新用例的很好来源。

- 原理6-测试是上下文相关的

测试在不同上下文环境中的执行是不同的。比方说安全关键系统(scrfety criticcl system)和电子商务网站的测试方法就有很大不同。

这个原理相对难理解。这里其实强调的是不能用相同的态度和手段来测试不同类型的系统。

软件测试的原则

- 原理7-无错谬论

假如建立的系统不稳定或不能满足用户需要和期望，那么发现和修复缺陷就毫无帮助了。

缺陷数量往往用来评估某软件的质量，但要是系统本身背离了用户要求，那就算缺陷再少也没用，因为没有人会去用它。

所以测试时要注意验证(verification)和确认(validation)的区别。需求规格说明和其他文档只是需求的不完全载体。文字说明必然有遗漏和偏差，而各人的理解更有可能出错。要不断通过各种途径保证所生产的的确就是用户需要的。

常用的方式就是邀请领域专家或用户尽可能多地参与到开发活动来，特别是需求评审和演示(Demo)。

软件测试的原则

- 其它测试原则
  - 测试用例应由测试输入和与之对应的预期输出结果两部分组成。
  - 程序员应避免检查自己的程序。（注意不是指对程序的调试）
  - 在设计测试用例时，应当包括合理的输入条件和不合理的输入条件。不合理的输入条件是指异常的，临界的，可能引起问题异变的输入条件。
  - 严格执行测试计划，排除测试的随意性。
    测试计划应包括：所测软件的功能，输入和输出，测试内容，各项测试的进度安排，资源要求，测试资料，测试工具，测试用例的选择，测试的控制方法和过程，系统的组装方式，跟踪规则，调试规则，以及回归测试的规定等等以及评价标准。

软件测试的原则

- 其它测试原则
  - 妥善保存测试计划，测试用例，出错统计和最终分析报告，为维护提供方便。
  - 应当对每一个测试结果做全面的检查。
  - 对测试结果要有一个确认过程。

    A测出的错误由B确认。严重的错误可召开评审会进行讨论和分析。
  - 软件测试必须基于“质量第一”的思想去开展各项工作，当时间和质量冲突时，时间服从质量。

软件测试的原则

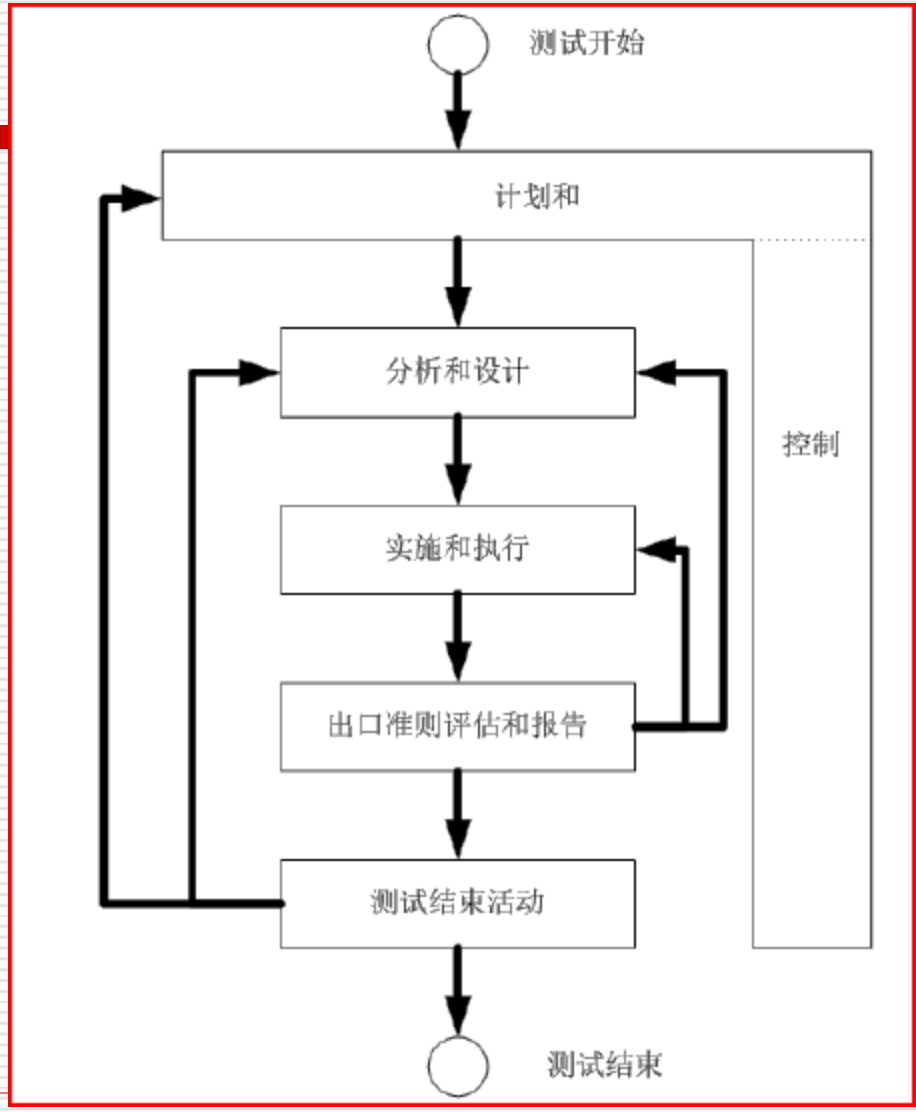
- 其它测试原则
  - 并非所有软件缺陷都要修复

原因：没有足够的时间进行修复；修复的风险较大； 不值得修复；可不算做故障的一些缺陷；“杀虫剂现象”。

结论：关键是要进行正确判断、合理取舍，根据风险分析决定哪些故障必须修复，哪些故障可以不修复。

基本测试过程

- 基本测试过程由以下测试活动构成：
  - 计划与控制
  - 分析与设计
  - 实施与执行
  - 评估出口准则和报告
  - 测试结束活动



基本测试过程

- 测试计划和控制

测试计划是定义测试目标及测试活动规格说明以满足特定目标和使命的过程。

这是一个过程，而不是完成一份计划文档，需要所有相关人员的参与并达成共识，否则提交的文档没有任何价值。

有一些快速开发的小型项目没有翔实的计划文档，但所有项目成员在一起进行了细致的讨论，确定了项目开展的细节，那这样的计划活动效果要远好过测试经理闷头熬夜赶出来的计划文档。

要计划的内容有测试方法，人员，策略，时间表，度量等等。简言之，计划应当包含所有与测试执行相关的信息，但是切忌大而空，内容要有用，提供的评估和度量要具有可行性。

有人把计划总结为：**什么人，在什么时间内，根据什么，做什么。** IEEE829测试计划规格说明书提供了指导性建议。

基本测试过程

- 测试计划和控制（续）

  测试控制是持续比较实际进展和计划的活动，并报告状态，比如实际和计划的偏离。

  这也包括采取一定的措施来符合项目使命和目标。为了能有效进行测试控制，我们需要在整个项目周期内对测试活动进行一定监控。测试计划也应当纳入测试监控的反馈。

  用更加易于理解的话来说，测试控制就是持续追踪测试活动的进度，及时发现与计划的偏差，并采取措施消除偏差或修改计划。

基本测试过程

- 测试分析和设计

  测试分析和设计是将抽象测试目标转化为具体有形测试条件和测试用例的过程。 测试分析和设计有以下主要任务：

  - 评审测试依据(test bcrsis)（例如需求文档，软件完整性级别（风险级别），风险分析报告，构架设计，设计，界面规格说明书）。
  - 评估测试依据和目标的可测试性(testcrbility)
  - 基于测试项，规格说明，行为和软件结构进行分析并确定测试条件(test conditions)以及优先级
  - 设计高级测试用例(high level test ccse)并确定优先级
  - 确定必要的测试数据(test datc)来支持测试条件和测试用例

基本测试过程

- 测试分析和设计（续）
  - 设计测试环境(test environment)并确定所需的基础设施(infrcrstructure)和工具
  - 建立测试依据和测试用例间的双向可追溯性(trc cecrbility)

测试设计是一个很严谨的过程，很容易发现细节上的问题。如，系统需求：“此系统用户密码加密存储，且在数据库中相同密码显示的密文不能相同。” 此需求从安全的角度来看很合理，描述也很清晰。但当你设计测试用例的时候就会发现问题了：它具有可测试性吗？ 对应于一个密码来说，期待的加密结果是什么？如何测试来保证相同密码加密后不会相同？

基本测试过程

- 测试实施和执行

  测试实施和执行是结合测试用例确定测试规程(test procedure)，脚本（test script）以及执行顺序，建立测试环境并运行测试的过程。

  可以把该定义分成两部分来理解：测试实施是搭环境，选用例，而测试执行就是所说的“跑测试”。

  注意测试规程，脚本和用例的区别：测试用例可以是高级（抽象）的，也可以是具体可遵照执行的。 而测试用例中可以遵照执行的动作序列就是测试规程。 测试脚本是特指自动化脚本。

基本测试过程

- 测试实施和执行

  测试实施和执行包含以下主要任务：

  - 确定，实旋测试用例（包括确定测试数据），以及制定优先级
  - 开发测试规程及制定优先级，创建测试数据，和准备测试用具(test hcrrness，指的是一个执行环境或框架)及自动化脚本（非必须）
  - 根据测试规程创建测试套件以达到高效测试的目的
  - 验证测试环境搭建正确
  - 验证和更新测试依据和测试用例间的双向可追溯性
  - 根据预定的执行次序使用手工方式或工具执行测试规程
  - 记录测试执行的输出，被测试软件的版本或标示，测试工具和测试件

基本测试过程

- 测试实施和执行
  - 比较真实结果和预期结果
  - 将（真实结果和预期结果）比较的差异作为事件来报告，并且分析其原因（比如代码缺陷，测试数据，测试文档或测试的错误执行方法等）
  - 重复执行引起执行差异的测试活动，比如重测试上次失败的测试来确认缺陷是否被修复了（确认测试），执行改正后的或原有的测试以确定修复缺陷的代码没有弓|入或揭示新的缺陷（回归测试）

基本测试过程

- 分析出口准则（测试退出条件）和报告

评估出口准则是指将测试执行与先前定义目标做比较评估的活动。这对于每个测试级别都有效。

分析出口准则包括以下主要任务：

- 将测试日志和测试计划中规定的出口准则做比较
- 确定是否需要更多的测试或者修改规定的出口准则
- 为相关方提供测试总结报告

测试出口准则不是一成不变的，根据实际情况要不断修正。比如原来规定的是所有bug都要修复，但在项目进行当中发现时间不够，可以通过某些正式的流程将出口准则修改为修复“普通”以上级缺陷。

基本测试过程

测试结束活动

测试结束活动指的是从已完成的测试活动中整理经验，测试件和事实数据等。

测试结束活动发生于某些项目里程碑，如当软件系统发布，软件项目完成（或取消），一个里程碑达成，或一个维护版本发布。

测试结束活动包括以下主要任务：

- 检查计划交付物都已经交付
- 关闭时间报告或提交曼改记录
- 编写系统验收文档
- 完成和打包测试件，测试环境和测试基础设施，以便未来重用

基本测试过程

- 测试结束活动
  - 将测试件交付给维护组
  - 分析经验教训以确定将来项目所需要的改进
  - 用收集的信息来增进测试成熟度

测试独立性(independence)

一定程度的独立性常使测试人员能更有效地发现缺陷和失效。但是独立性也不能取代对系统的熟悉。这里从低到高定义了独立性级别：

- 被测试软件开发者来设计测试（最低独立性）
- 其他人（比如其他开发人员）来设计测试
- 同组织内其他组的人员（比如独立测试组）或测试专业人员（比如可用性或性能测试员）来设计测试
- 不同组织或公司的人员（外包或外部授权体）来设计溅试

测试信息流程

测试信息流程如图所示。测试过程中需要三类输入：软件配置、测试配置和测试工具。

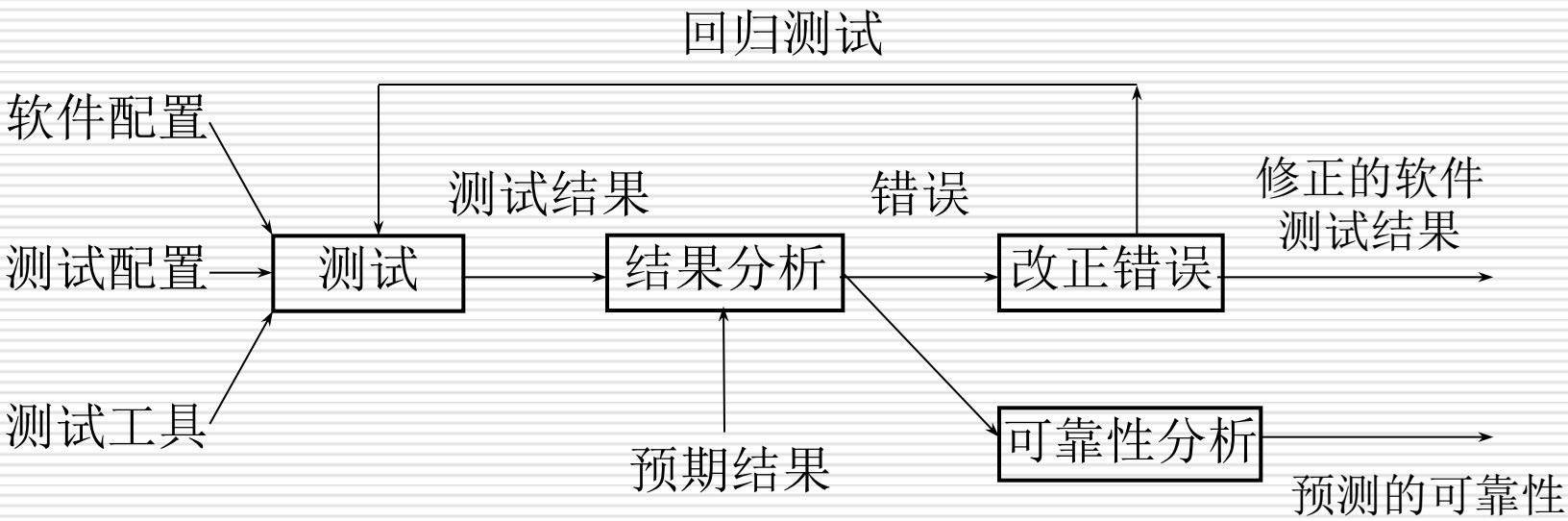


图　测试信息流程

软件测试的周期性

软件测试的周期性是“测试->改错->再测试->再改错”这样一个循环过程，如下图所示。

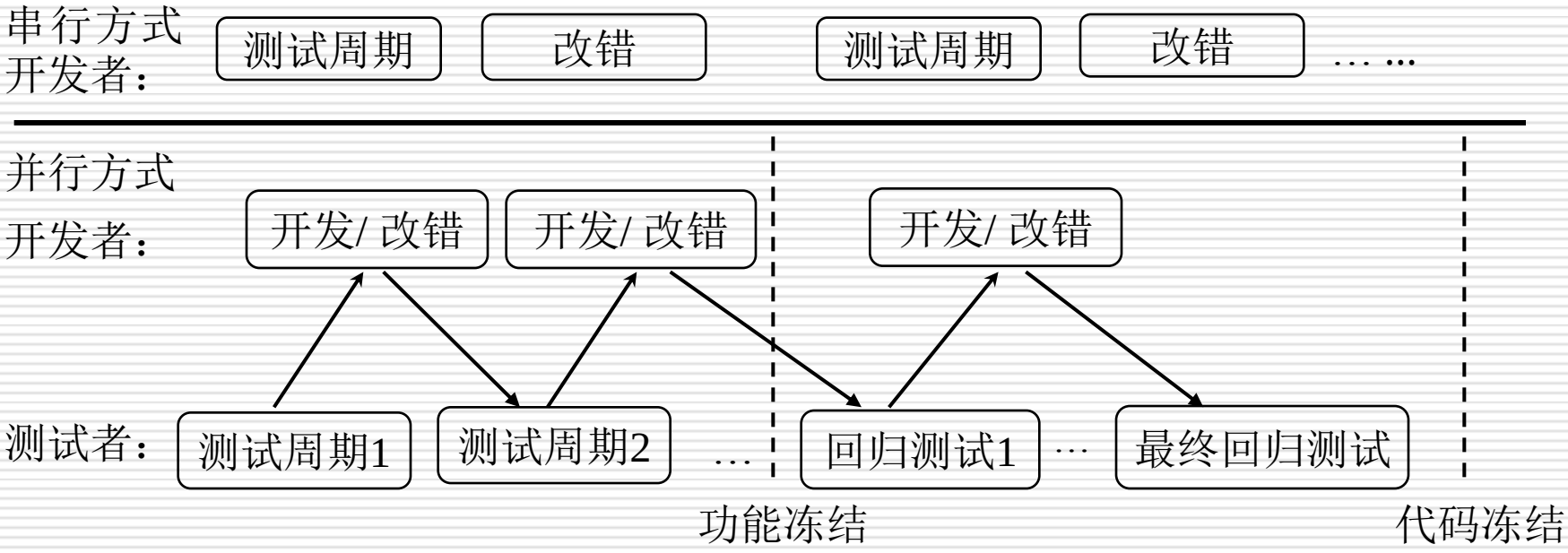


图 软件测试的周期性

软件测试技术概要

- 软件测试的策略：就是测试将按照什么样的思路和方式进行。通常，软件测试要经过单元测试、集成测试、确认测试、系统测试以及验收测试。
- 软件测试技术：

（1）白盒测试和黑盒测试

（2）静态测试和动态测试

（3）传统测试方法和面向对象测试的方法

（4）特定环境及应用的测试

软件测试误区

- 如果发布的软件有质量问题，那是软件测试人员的错。
- 软件测试技术要求不高，至少比编程容易多了。
- 有时间就多测试一些，来不及就少测试一些。
- 软件测试是测试人员的事，与开发人员无关。
- 根据瀑布模型，测试是开发后期的一个阶段。

软件测试人员应具备的素质

- 软件测试员在开发团队中“讨人厌”
- 软件测试员的目标：　　发现潜在的软件缺陷
- 保持团队和谐的建议
  - 尽可能早的找出缺陷
  - 进行合作而不是争斗：控制情绪,提醒所有人质量更好的系统才是共同目标
  - 对事不对人，围绕事实基础，进行中立客观的沟通，而不是批评相关人，比方说我们可以写下目标和实际事件报告评审测试发现的问题
  - 沟通很重要：不要总是报告坏消息；尽力理解其他入的感受和反应；确认他们理解你所说的，你也理解他们所说的

软件测试人员应具备的素质

- 软件测试员应具备的素质：

(1)探索精神

(2)故障排除能力

(3)不懈努力

(4)创造性

(5)追求完美

(6)判断准确

(7)老练稳重

(8)说服力

道德规范

参与测试的个人会接触到保密和需要权限才能了解的信息。一套职业道德规范可以保证这些敏感信息不会被滥用。

- 公共：认证软件测试员的行为应当与公共利益一致
- 客户和雇主：认证软件测试员的行为应当符合客户和雇主的最大利益，并与公共利益一致
- 产品：认证软件测试员应当保证他们的提交物（测试的产品和系统）符合可能的最高职业标准
- 判断：认证软件测试员的职业判断应当具有正直的品格和独立性
- 管理：认证软件测试经理应当在软件测试管理中遵循和推广符合职业道德的工作方式

道德规范

- 职业：认证软件测试员应当不断推进本身的职业正直品格和声誉，并符合公共利益
- 同事：认证软件测试员应当公平对待并支持他们的同事，并增进与软件开发人员的合作
- 自身：认证软件测试员应当参与与之I作实践相关的终身学习，并提高工作时间的职业道德工作方式

目前中国在软件测试职业道德方面的规范还很欠缺，且职业道德规范本身大多不具有强制性，并且与所处的文化环境相关联。而真正和大多数测试人员关系比较密切的是不能违反相关的保密隐私信息法律法规。有不少公司为图方便把真实用户信息用于测试，这会损害用户利益。测试人员有责任通过一定途径提出劝告。

习题

1. 下列哪个是测试计划阶段的主要任务之一？

   A. 为测试分析和设计安排时间

   B. 初始化更正活动（如修改缺陷）

   C. 监控测试进度和覆盖度

   D. 度量和分析测试结果

2. 调试活动通常由谁完成？

   A. 开发人员.　　B. 分析人员.

   C. 测试人员.　　D. 事件管理人员，

3. 测试退出准则是在哪个测试过程中确定的？

   A. 测试计划　　B. 测试准则评估和报告.

   C. 测试结束活动.　　D. 测试控制.

习题

4．以下哪个是测试实施和执行阶段的主要任务之一？

A．度量和分析结果　　B．作为事件报告测试差异

C．发现测试条件或测试需求

D．评估是否需要添加更多测试

5．效率测试（比如性能测试）套件在基本测试过程的哪个阶段创建？

A．实施和执行　　B．计划和控制

C．分析和设计　　D．测试结束活动

习题

6．关于确认测试和回归测试以下哪个表述是正确的？

A．确认测试是测试一组缺陷的修复而回归测试是测试代码变更是否引入新的缺陷

B．回归测试测试是测试一组缺陷的修复而确认测试是测试代码变更是否引入新的缺陷

C．确认测试和回归测试都是测试代码变更是否引入新的缺陷

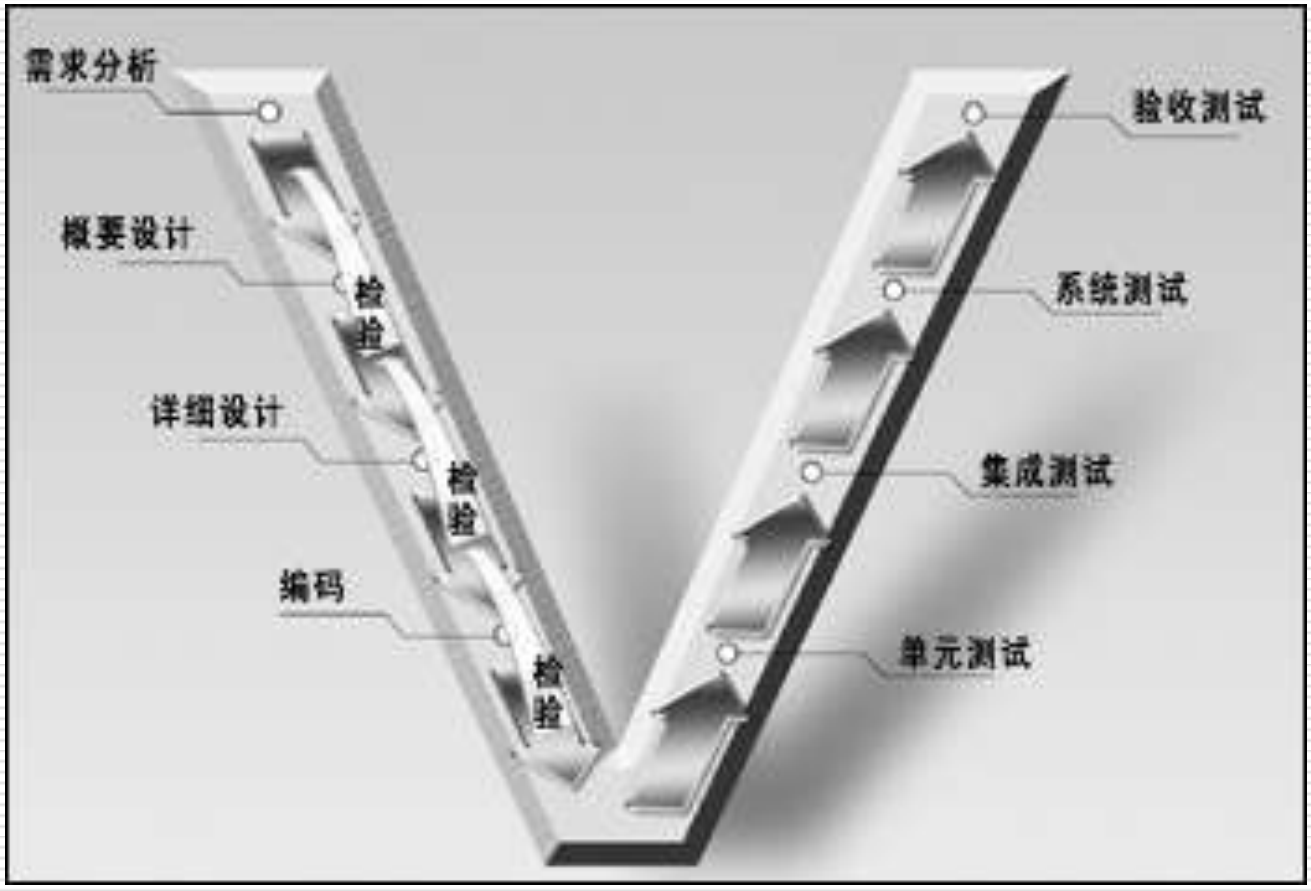
D．确认测试和回归测试都是测试一组缺陷的修复

软件测试的模型

软件测试模型则是软件测试的工作框架,用于指导软件测试过程.

- V模型
- W模型
- H模型
- X模型
- 前置模型
- 测试模型的使用

软件测试的模型 - V模型



软件测试的模型 - V模型

- V模型描述了不同的测试级别

  V模型描述了一些不同的测试级别,并说明了这些级别所对应的生命周期中不同的阶段。

  如模型图中所示,左边下降的是开发过程各阶段,与此相对应的是右边上升的部分，即各测试过程的各个阶段。

  在模型图中的开发阶段一侧，先从定义业务需求开始，然后要把这些需求不断地转换到概要设计和详细设计中去，最后开发为程序代码。在测试执行阶段一侧，执行先从单元测试开始，然后是集成测试、系统测试和验收测试。

- 成功应用V模型的关键因素是设计测试案例的时机

软件测试的模型 - V模型

- V模型的价值

  它非常明确地标明了测试过程中存在的不同级别，并且清楚地描述了这些测试阶段和开发过程期间各阶段的对应关系：

  - 单元测试的主要目的是针对编码过程中可能存在的各种错误，例如用户输入验证过程中的边界值的错误。
  - 集成测试主要目的是针对详细设计中可能存在的问题尤其是检查各单元与其它程序部分之间的接口上可能存在的错误。
  - 系统测试主要针对概要设计，检查了系统作为一个整体是否有效地得到运行，例如在产品设置中是否达到了预期的高性能。
  - 验收测试通常由业务专家或用户进行，以确认产品能真正符合用户业务上的需要。

软件测试的模型 - V模型

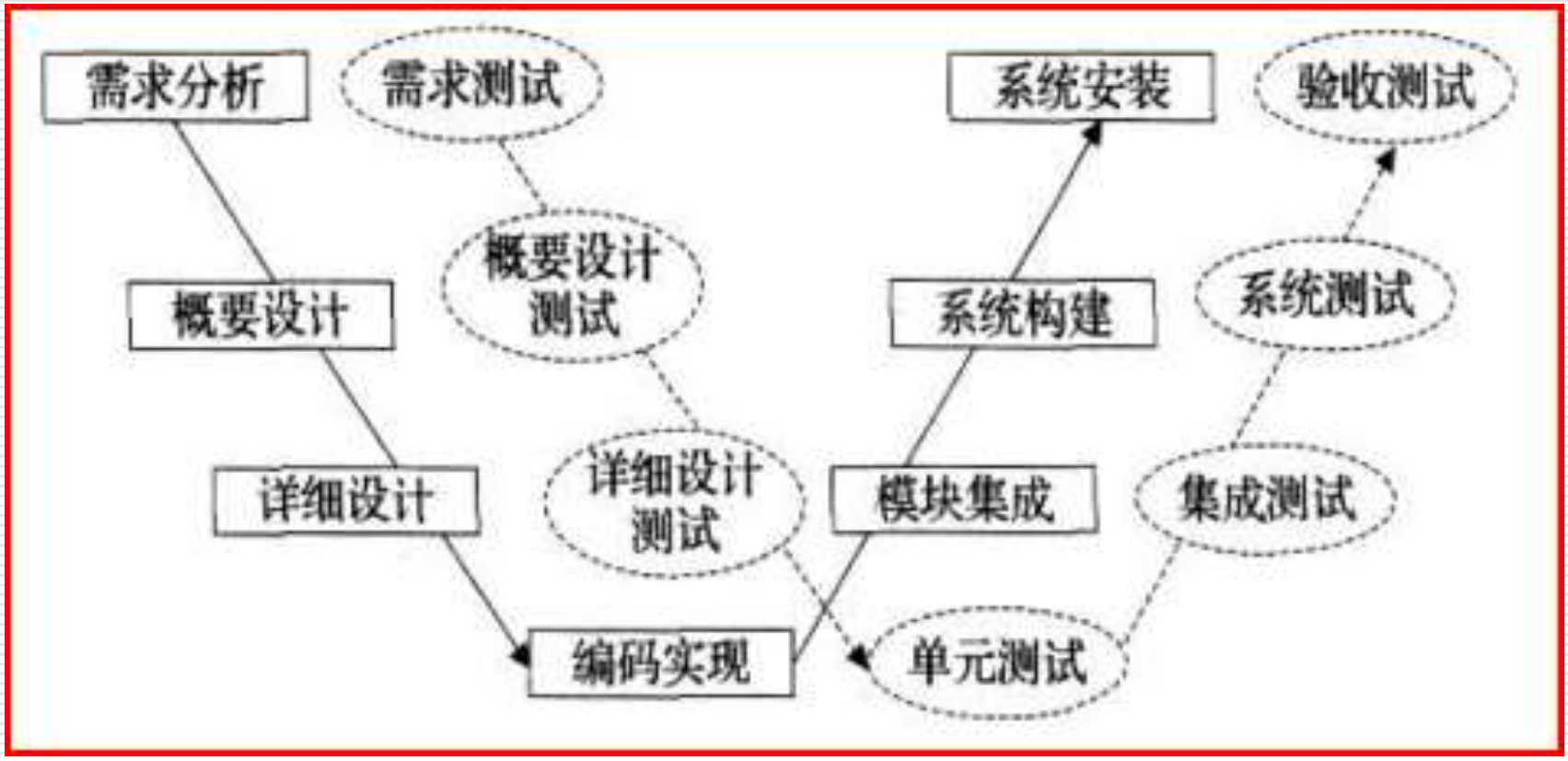
- V模型问题
  - 测试是开发之后的一个阶段
  - 测试的对象是程序本身
  - 易导致需求阶段的错误一直到最后系统测试阶段才被发现

    如果问题不能及时被发现，这些隐含的问题也被带到下一个工序，正确的设计被编码，错误的设计也同时被编码。

软件测试的模型 - W模型

- V模型测试的改进，在概要设计、详细设计和编码每个步骤都要进行检测。尽量把问题及时发现、及时消灭。
- W模型是基于IEEE std 1012-1998《软件验证和确认（V&V）》原则提出。此原则主要思想是“尽早地和不断地进行软件测试”。

软件测试的模型 - W模型


需求分析
需求测试
系统安装
验收测试
概要设计
测试
概要设计
系统构建
系统测试
详细设计
详细设计
测试
模块集成
集成测试
编码实现
单元测试

软件测试的模型 - W模型

- 测试伴随整个开发周期。相应开发活动完成，即可对相应开发活动进行测试
- 测试对象不仅是程序，还包括需求和设计
- 优点

  强调了测试计划等工作的先行和对系统需求和设计的测试。
- 缺点

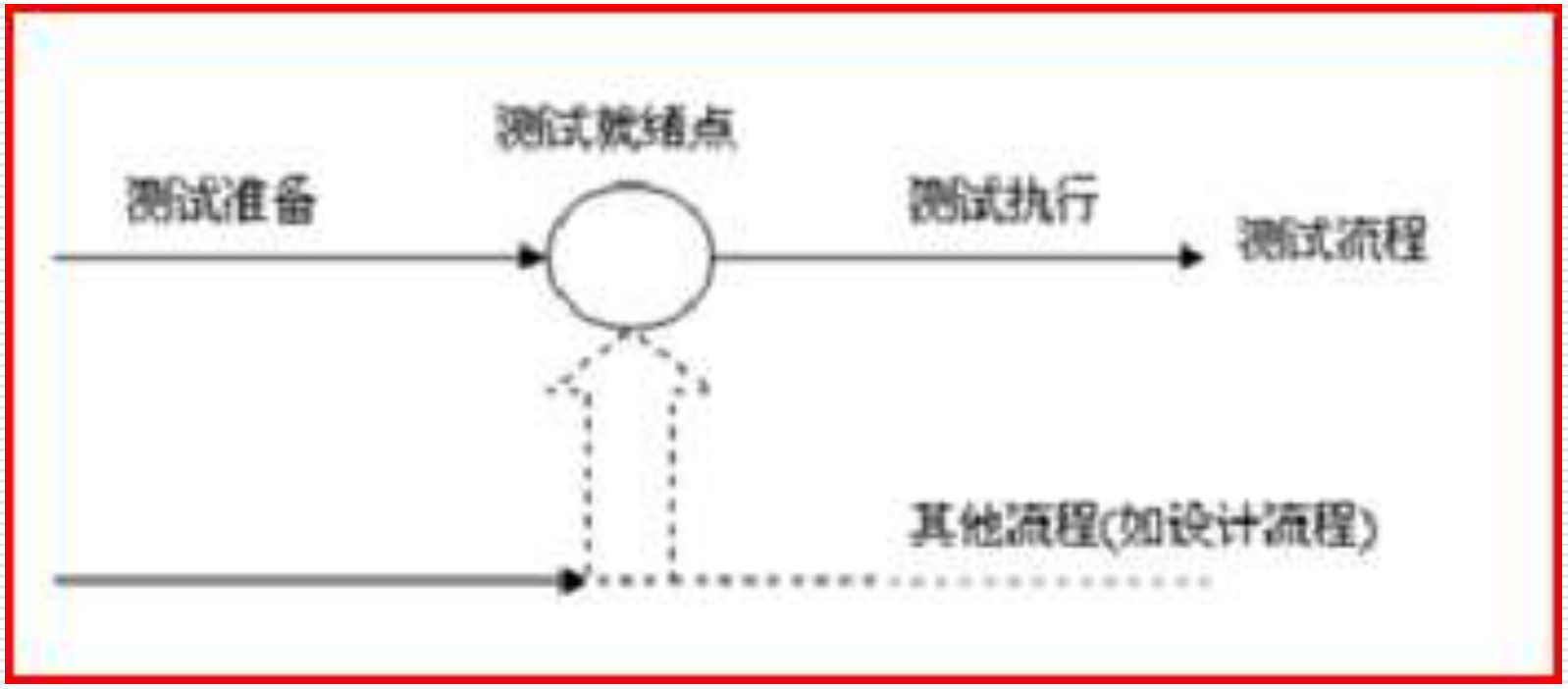
  没有对软件测试流程予以说明

软件测试的模型 - H模型

- V模型和W模型的局限
  - 软件开发被视为一系列串行活动。实际上，大部分时间可并发。
  - 软件开发中，严格的阶段划分只是一种理想状态。实际，只要测试条件满足，就可进行测试。不同层次测试之间除了先后关系外，还有触发、反复、迭代和增量关系。
  - 没有很好地表示测试流程的完整性。测试流程大致可分为测试准备活动（包括测试需求分析、测试计划、测试设计、测试编码和测试验证）和测试执行活动（包括测试运行、测试分析和测试报告）

软件测试的模型 - H模型

- H模型将测试作为一个独立流程，贯穿整个开发周期，与其他流程并行，同时测试准备和测试执行分离。



软件测试的模型 - H模型

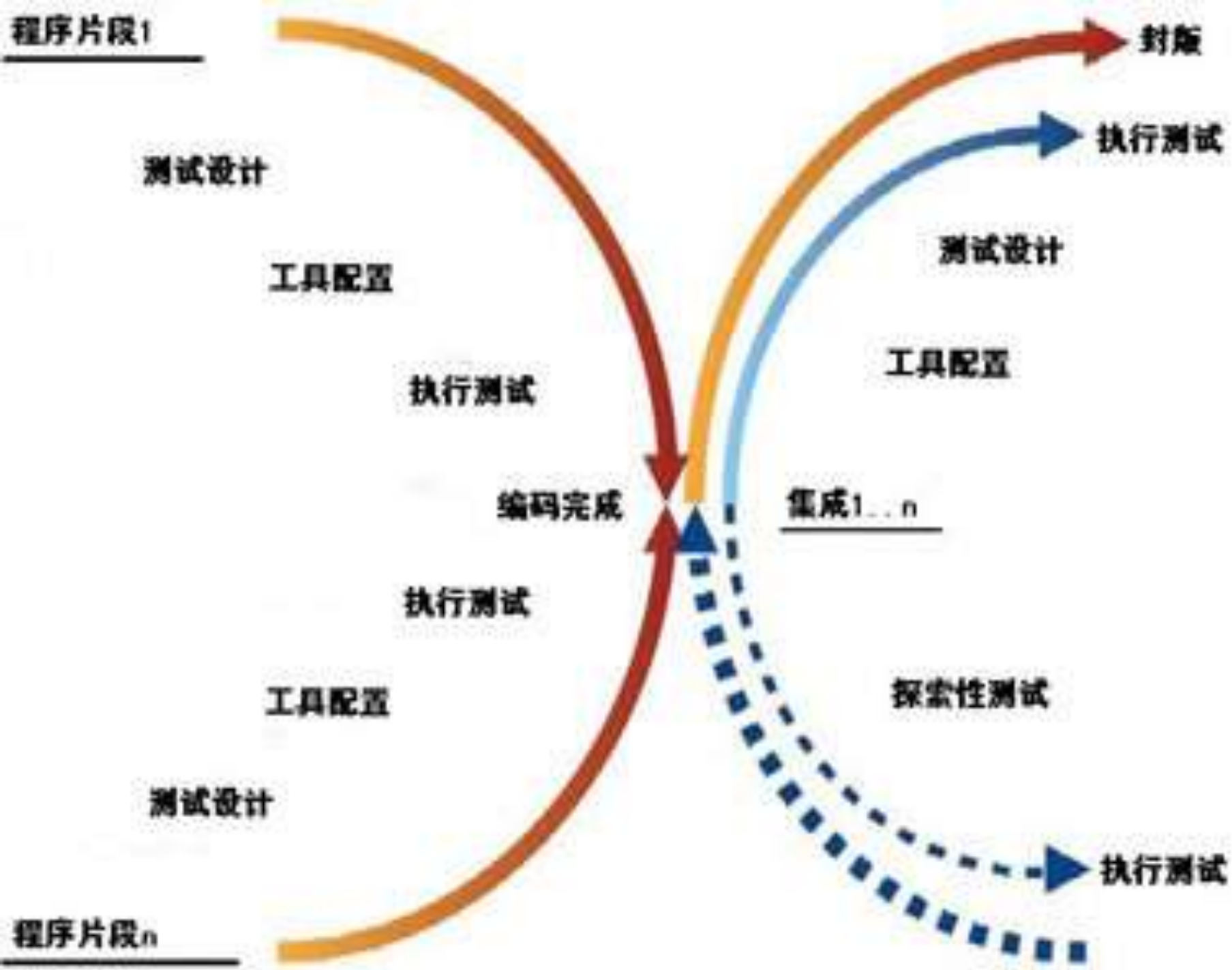
- H模型特性
  - 测试不仅仅指测试的执行，还包括许多其他活动;
  - 测试是一个独立流程，贯穿产品整个生命周期；与其它流程并发进行
  - 测试要尽早准备，尽早执行
  - 测试是根据被测对象的不同而分层进行。
- 意义
  - 测试准备和测试执行分离，有利于资源调配，降低成本，提高效率
  - 充分体现测试过程（不是技术）的复杂性
  - 有组织、有结构化的独立流程，有助于跟踪测试投入的流向

软件测试的模型 - X模型

- X模型

  X模型基本思想由Brian Marick（软件子系统测试的作者） 提出，Robin F.Goldsmith（Go项目管理咨询公司的总裁 ）命名。

- Brian Marick对V模型的质疑主要有：
  - V模型无法引导项目的全过程。他认为一个模型应能处理开发的所有方面，包括交接，频繁重复的集成，以及需求文档的缺乏等。
  - V模型基于一套必须按照一定顺序严格排列的开发步骤，而这很可能并没有反映实际的实践过程。
  - 质疑了单元测试和集成测试的区别，因为在某些场合人们可能会跳过单元测试而热衷于直接进行集成测试。按照V模型所指导的步骤进行工作，某些做法并不切合实用。


程序片段1
测试设计
工具配置
执行测试
编码完成
执行测试
工具配置
测试设计
程序片段n
集成1..n
封版
执行测试
测试设计
工具配置
探索性测试
执行测试

软件测试的模型 - X模型

- X模型的左边描述的是针对单独程序片段所进行的相互分离的编码和测试
- 此后将进行频繁的交接,通过集成最终合成为可执行的程序。(右上半部分）,这些可执行程序还需要进行测试。已通过集成测试的成品可以进行封版并提交给用户，也可以作为更大规模和范围内集成的一部分。多根并行的曲线表示变更可以在各个部分发生。

软件测试的模型 - X模型

- X模型还定位了探索性测试（右下方）。

  这是不进行事先计划的特殊类型的测试， 诸如“我这么测一下结果会怎么样？”，这一方式往往能帮助有经验的测试人员在测试计划之外发现更多的软件错误.X模型及其探索性测试的提倡也是为了避免把大量时间花费在测试文档编写上面，那样的话，真正用于测试的时间就减少了。

软件测试的模型 - X模型

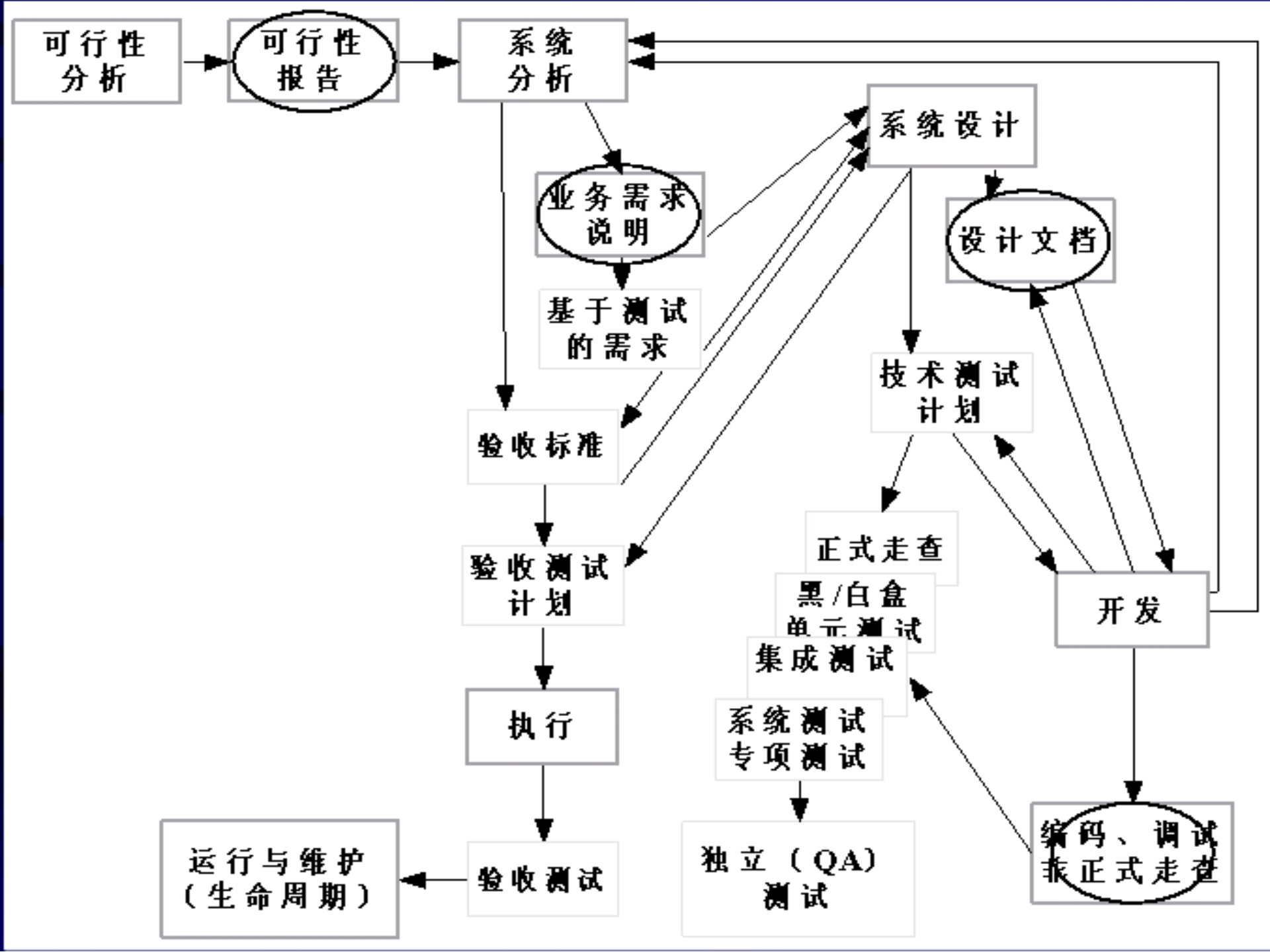
- V模型的一个强项是它明确的需求角色的确认，而X模型没有这么做，这是X模型的一个不足之处。
- X模型并不要求在进行作为创建可执行程序（图中右上方）的一个组成部分的集成测试之前， 对每一个程序片段都进行单元测试（图中左侧的行为）。但X模型没能提供是否要跳过单元测试的判断准则。

软件测试的模型 - 前置测试模型

- 前置测试模型

由Robin F. Goldsmith ， Dorothy Graham提出，是一个将测试和开发紧密结合的模型，该模型提供了轻松的方式，可以使你的项目加快速度。

V模型和X模型是当前被测试专家所推崇的主要的测试模型。前置测试从V模型和X模型中汲取其中精华，并设法弥补了它们的不足之处。虽然前置测试也不是完美的，但它可以带来明显的益处。


可行性分析
可行性报告
系统分析
系统设计
业务需求说明
设计文档
基于测试的需求
技术测试计划
验收标准
正式走查
验收测试计划
黑/白盒
集成测试
开发
执行
系统测试
专项测试
运行与维护（生命周期）
验收测试
独立（QA）测试
编码、调试
非正式走查

软件测试的模型 - 前置测试模型

- 开发和测试相结合

  前置测试模型将开发和测试的生命周期整合在一起，标识了项目生命周期从开始到结束之间的关键行为。并且表示了这些行为在项目周期中的价值所在。如果其中有些行为没有得到很好的执行，那么项目成功的可能性就会因此而有所降低。

  如果有业务需求，则系统开发过程将更有效率。业务需求最好在设计和开发之前就被正确定义。

软件测试的模型 - 前置测试模型

- 对每一个交付内容进行测试：每一个交付的开发结果都必须通过一定的方式进行测试。

  源程序代码并不是唯一需要测试的内容。在图中的圆圈表示了其它一些要测试的对象， 包括可行性报告、业务需求说明，以及系统设计文档等。这同V模型中开发和测试的对应关系是相一致的， 并且在其基础上有所扩展，变得更为明确。

软件测试的模型 - 前置测试模型

- 在设计阶段进行计划和测试设计：设计阶段是做测试计划和测试设计的最好时机。

  很多组织要么根本不做测试计划和测试设计，要么在即将开始执行测试之前才飞快完成测试计划和设计。在这种情况下，测试只是验证了程序的正确性，而不是验证整个系统本该实现的东西。

  在V模型中，验收测试最早被定义好，并在最后执行，以验证所交付的系统是否真正符合用户业务的需求。与V模型不同的是，前置测试模型认识到验收测试中所包含的3种成份，其中的2种都与业务需求定义相联系：即定义基于需求的测试，以及定义验收标准。但是，第三种则需要等到系统设计完成，因为验收测试计划是由针对按设计实现的系统来进行的一些明确操作定义所组成，这些定义包括：如何判断验收标准已经达到，以及基于需求的测试已算成功完成。

软件测试的模型 - 前置测试模型

技术测试主要是针对开发代码的测试，例如V模型中所定义的动态的单元测试，集成测试和系统测试。另外，前置测试还提示我们应增加静态审查。还有特别测试，并把该名称作为很多测试的一个统称，这些测试包括负载测试、安全性测试、可用性测试等等。　　对技术测试最基本的要求是验证代码的编写和设计的要求是否相一致。一致的意思是系统确实提供了要求提供的，并且系统并没有提供不要求提供的。技术测试在设计阶段进行计划和设计，并在开发阶段由技术部门来执行。

软件测试的模型 - 前置测试模型

- 测试和开发结合在一起：前置测试将测试执行和开发结合在一起，并在开发阶段以编码-测试-编码-测试的方式来体现.也就是说，程序片段一旦编写完成，就会立即进行测试。

  普通情况下，先进行的测试是单元测试，因为开发人员认为通过测试来发现错误是最经济的方式。但也可参考X模型，即一个程序片段也需要相关的集成测试,甚至有时还需要一些特殊测试。对于一个特定的程序片段，其测试的顺序可以按照V模型的规定，但其中还会交织一些程序片段的开发，而不是按阶段完全地隔离。

软件测试的模型 - 前置测试模型

- 让验收测试和技术测试保持相互独立：验收测试应该独立于技术测试，这样可以提供双重的保险，以保证设计及程序编码能够符合最终用户的需求。

  验收测试既可以在实施阶段的第一步来执行，也可以在开发阶段的最后一步执行。

  前置测试模型提倡验收测试和技术测试沿循2条不同的路线来进行，每条路线分别地验证系统是否能够如预期的设想进行正常工作。这样，当单独设计好的验收测试完成了系统的验证, 我们即可确信这是一个正确的系统。

软件测试的模型 - 前置测试模型

- 反复交替的开发和测试：开发和测试需要一起反复交替地执行。

  在项目中从很多方面可以看到变更的发生,例如需要重新访问前一阶段的内容,或者地跟踪并纠正以前提交的内容，修复错误，排除多余的成分,以及增加新发现的功能,等等。

  模型并没有明确指出参与的系统部分的大小。这一点和V模型中所提供的内容相似。不同的是，前置测试模型对反复和交替进行了非常明确的描述。

软件测试的模型 - 前置测试模型

- 发现内在的价值：前置测试能给需要使用测试技术的开发人员、测试人员、项目经理和用户等带来很多不同于传统方法的内在的价值。

  与以前的方法中很少划分优先级所不同的是，前置测试用较低的成本来及早发现错误，并且充分强调了测试对确保系统的高质量的重要意义。前置测试代表了整个对测试的新的不同的观念。在整个开发过程中，我们反复使用了各种测试技术以使开发人员、经理和用户节省其时间，简化其工作。

前置测试模型内在的价值

通常情况下，开发人员会将测试工作视为阻碍其按期完成开发进度的额外的负担。然而，当提前定义好该如何对程序进行测试以后，可能会使开发人员将节省至少20%的时间。虽然开发人员很少意识到他们的时间是如何分配的，也许他们只是感觉到有一大块时间从重新修改中节省下来可用来进行其它的开发。保守地说，在编码之前对设计进行测试可以节省总共将近一半的时间，这可以从以下方面体现出来：

前置测试模型内在的价值

- 针对设计的测试编写是检验设计的一个非常好的方法，由此可以及时避免因为设计不正确而造成的重复开发及代码修改。通常情况下，这样的测试可以使设计中的逻辑缺陷凸显出来。另一方面，编写测试用例还能揭示设计中比较模糊的地方。总的来说，如果你不能勾画出如何对程序进行测试，那么程序员很可能也很难确定他们所开发的程序怎样才算是正确的。
- 测试工作先于程序开发而进行，这样可以明显地看到程序应该如何工作，否则，如果要等到程序开发完成后才开始测试，那么测试只是查验开发人员的代码是如何运行的。而提前的测试可以帮助开发人员立刻得到正确的错误定位。

前置测试模型内在的价值

- 在测试先于编码的情况下，开发人员可以在一完成编码时就立刻进行测试。而且，会更有效率，在同一时间内能够执行更多的现成的测试，开发人员的思路也不会因为去搜集测试数据而被打断。
- 即使是最好的程序员，从他们各自的观念出发，也常常会对一些看似非常明确的设计说明产生不同的理解。如果他们能参考到测试的输入数据及输出结果要求，就可以帮助他们及时纠正理解上的误区，使其在一开始就编写出正确的代码。
- 前置测试定义了如何在编码之前对程序进行测试设计，开发人员一旦体会到其中的价值，就会对其表现出特别的欣赏。前置方法不仅能节省时间，而且可以减少那些令他们十分厌恶的重复工作。

测试模型的使用

- 任何模型都不完美，不应为了使用模型而照搬。
- 实测中，应灵活运用各模型优点，通常在W模型框架下，运用H模型的思想进行独立测试。当有变更时，按X模型和前置模型的思想进行处理。
- 测试和开发密不可分，寻找恰当的就绪点开始测试，并反复进行迭代测试，以达目标。