# 软件测试的定义和目的

1， 什么是软件测试

a) IEEE 定义为：使用人工和自动手段来运行或测试某个系统的过程，其目的在于检验它是否满足规定的需求或是弄清预期结果与实际结果之间的差别。

b) G.J.Myers 认为：1）程序测试是为了发现错误而执行程序的过程；2）好的测试方案是极可能发现迄今为止尚未发现的错误的测试方案；3）成功的测试是发现了至今为止尚未发现的错误测试。

（注：1）软件测试是一个过程，包含若干活动，运行软件进行测试只是活动之一；2）运行软件测试可以人工方式也可以借助于工具，3）进行软件测试可以运行软件也可以不运行软件；4）软件测试的目的不仅仅是为了发现错误。）

2， 软件测试的目的

人们对软件测试的目的的认识也经历了一个过程：

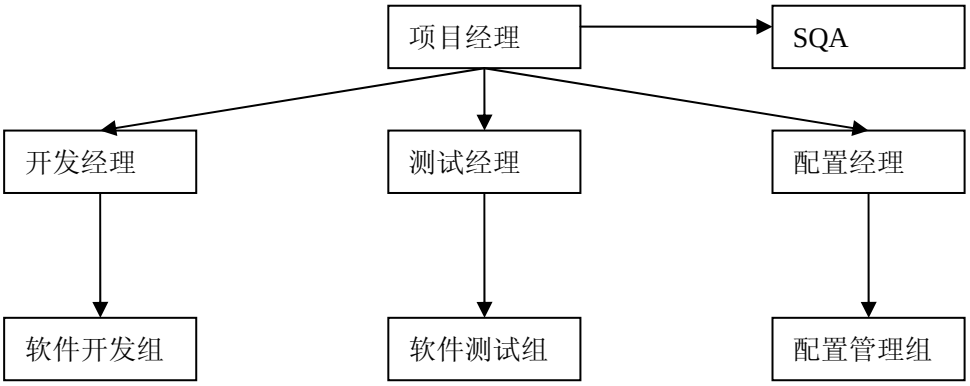
| 20 世纪 60 年代 | 20 世纪 70 年代中期 | 20 世纪 90 年代 |
| --- | --- | --- |
| 证明 | 检测 | 预防 |
| 表明软件能够工作 | 发现错误 | 管理质量 |

# 软件生命周期

计划 → 需求分析 → 设计 → 编码 → 测试 → 运行和维护

# 软件研发组织和流程

常见项目组架构




## 基本软件研发流程

1） 瀑布模型

2） 螺旋模型
3） RUP（Rational United Press）模型　所有工作流在各个阶段都有体现。（IBM 收购）
4） IPD（Integred Product Design）模型　从整个产品角度出发，不仅仅针对研发。（IBM）

# 软件中引入缺陷的原因

软件缺陷：既指静态存在于软件工作产品（文档，代码）中的错误，也指软件运行时由于这些错误被激发引起的和软件产品预期属性的偏离现象。
Bug ：代码中的缺陷。有时也被广泛指因软件产品内部的缺陷引起的软件产品最终运行时和预期属性的偏离。
（注：软件错误、软件缺陷、Bug 在实际工作中可以认为是一样。）

## 常见的引入缺陷的原因

1） 开发过程缺乏有效的沟通，或者没有进行沟通
2） 软件复杂度越来越高
3） 编程中产生的错误
4） 需求不断变更
5） 项目进度的压力
6） 不重视开发文档
7） 软件开发工具本身隐藏的问题
8） 。。。。。。。。。。。。。。。。。。。。。。

## 缺陷类型

1） 遗漏：规定的或者预期的需求未体现在产品中（可能未将规格说明全面实现，也可能需求分析阶段就遗漏了需求）
2） 错误：未将规格说明正确实现（可能设计错误、也可能编码错误）
3） 额外的实现：规格说明并未规定的需求被纳入了产品，得到实现。
（也可以用下面五种类型表示：
a) 产品未达到产品说明书中要求实现的功能
b) 产品出现了产品说明书中没有的功能
c) 产品没有实现产品说明书中虽未指明但要求实现的功能
d) 产品出现了说明书中明确规定不出现的功能
e) 测试人员或用户认为产品不应使用）

# 测试过程

## 测试阶段划分

### 单元测试（Unit Testing）

针对软件基本组成单元（软件设计的最小单位）来进行正确性检验的测试工作。（检测软件模块对《详细设计说明书（LLD）的符合度》）。

### 集成测试（Integration Testing）

在单元测试的基础上，将所有模块按照概要设计组装成为子系统或系统，验证组装后功能以及模块间接口是否正确的测试工作。（检测软件模块对《概要设计说明书（HLD）的符合度》）

### 系统测试（System Testing）

将已经集成好的软件系统，作为整个基于计算机系统的一个元素，与计算机硬件、外设、某些支持软件、数据和人员等其他元素结合在一起，在实际运行（使用）环境下，对计算机系统进行一系列的测试工作。（通过与《需求规格说明书（SRS）》作比较，发现软件与系统需求定义不符合或之矛盾的地方）

### 单元、集成、系统测试的比较

1） 测试方法不同
   单元测试属于白盒测试范畴
   集成测试属于灰盒测试范畴
   系统测试属于黑盒测试范畴
2） 考察范围不同
   单元测试主要测试单元内部的数据结构、逻辑结构、异常处理等
   集成测试主要测试模块之间的接口和接口数据传递关系，以及模块组合后的整体功能
   系统测试主要测试整个系统相对于需求的符合度
3） 评估基准不同
   单元测试主要是逻辑覆盖率
   集成测试主要是接口覆盖率
   系统测试主要是测试用例对需求规格的覆盖率

### 回归测试（Regression Testing）

目的：验证缺陷得到了正确的修复，同时对系统的变更没有影响以前的功能。
（注：回归测试可以发生在任何一个阶段）

回归测试策略
1） 完全重复测试
   重新执行所有在前期测试阶段建立的测试用例，来确认问题修改的正确性和修改的扩散局部影响性。

2） 选择性重复测试

即有选择地重新执行部分在前期测试阶段建立的测试用例，来测试被修改的程序

a) 覆盖修改法

即针对被修改的部分，选取或重新构造测试用例验证没有错误再次发生的用例选择方法

b) 周边影响法

该方法不但包含覆盖修改法确定的测试用例，还需要分析修改的扩散影响，对那些受到修改间接影响的部分选择测试用例验证它没有受到不良影响，该方法比覆盖修改法更充分一点。

c) 指标达成法

这是一种类似于单元测试的方法，在重新执行测试前，先确定一个要达成的指标，如修改的部分代码 100%的覆盖、与修改有关的接口 60%的覆盖等，基于这种要求选择一个最小的测试用例集合。

## 回归测试流程（适用于单元测试，集成测试，系统测试）

1） 在测试策略制定阶段，制定回归测试策略

2） 确定需要回归测试的版本

3） 回归测试版本发布，按回归测试策略执行回归测试

4） 回归测试通过，关闭缺陷跟踪单（问题单）

5） 回归测试不通过，缺陷跟踪单返回开发人员，开发人员重新修改问题，再次提交测试人员回归测试

（注：回归测试比较适合使用自动化工具）

## 其他测试阶段

1） 验收测试

a) 验收测试是以用户为主的测试，验收组应该由项目组成员，用户代表等组成

b) 在通过内部系统测试及软件配置审查后，就可以开始验收测试

c) 验收测试原则上在用户所在地进行，但经用户同意也可以在公司内模拟用户环境

d) 验收测试根据合同、《需求规格说明书》或《验收测试计划》对产品进行验证

e) 结果两种（接受与不接受）

2） Alpha 测试（属于验收测试）

由用户在开发环境下进行的测试，也可以是开发机构内部的用户在模拟实际操作环境下进行的测试。

目的主要是评价软件产品的 FLURPS（即功能、局域化、可用性、可靠性、性能和技术支持等）

3） Beta 测试（属于验收测试）

由软件的多个用户在一个或多个用户的实际环境下进行测试

Alpha 测试和 Beta 测试的区别

Alpha 测试过程可控，但是参与人数有限；Beta 测试参与人数巨大，但是过程不可控。

# 测试过程模型

## 测试过程阶段划分

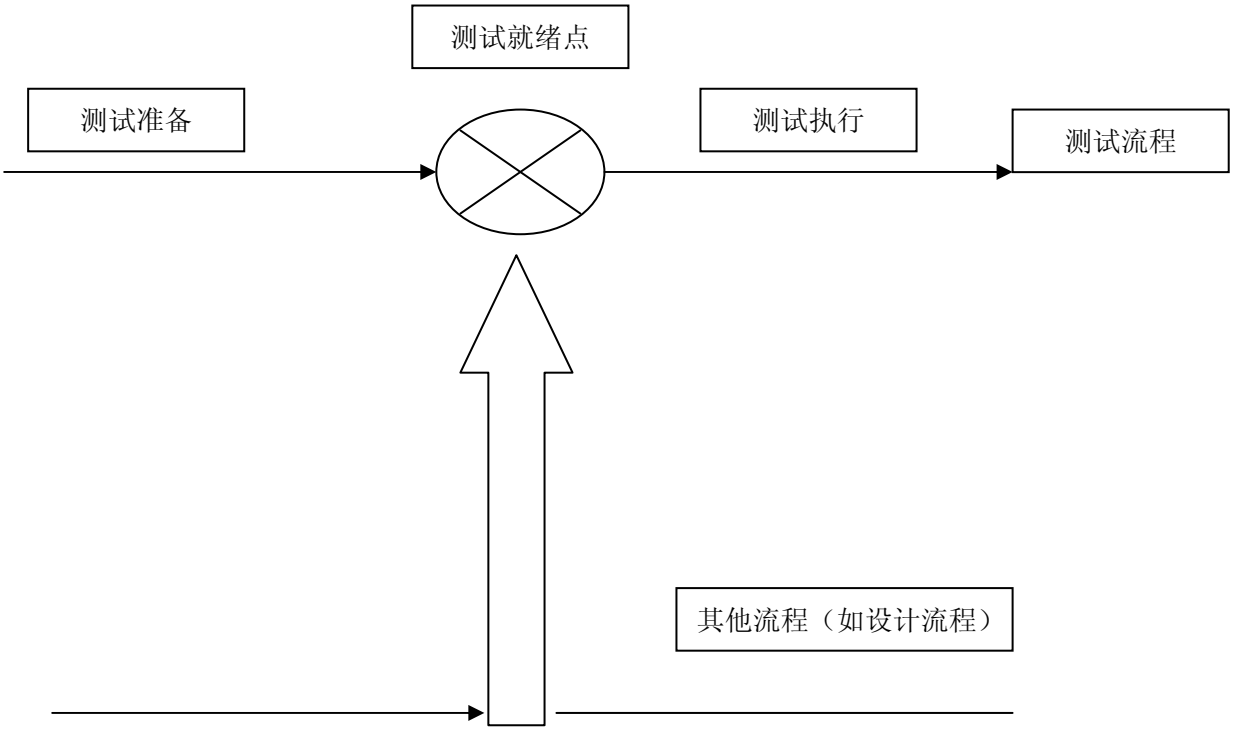
1） 测试计划阶段：测试计划
2） 测试设计阶段：测试方案
3） 测试实现阶段：测试用例、测试规程
4） 测试执行阶段：测试报告

## 主要测试文档

测试计划：指明测试范围、方法、资源、以及相应测试活动的时间进度安排表的文档。
测试方案：指明为完成软件或软件集成特性的测试而进行的设计测试方法的细节文档。
测试用例：指明为完成一个测试项的测试输入、预期结果、测试执行条件等因素的文档。
测试规程：指明执行测试时测试活动序列的文档。（后执行用例的输入是先执行用例的输出）
测试报告：指明执行测试结果的文档。（注：1）将工作过程表现出来 2）表明个人对测试对象的态度）
测试日报：每天测试执行情况的记录和总结。

## 常见的测试过程模型

1） 瀑布模型
缺陷：
a) 测试介入太晚
b) 工作效率低
c) 成本巨大
2） H 模型



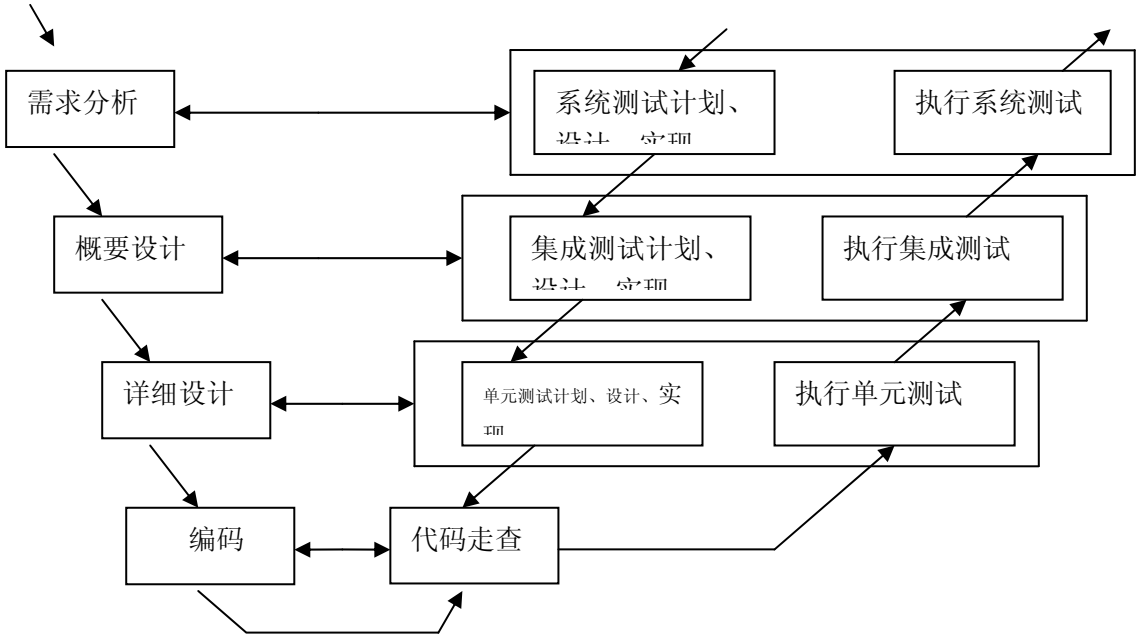

测试准备活动，包括测试需求分析、测试计划、测试设计、测试编码、测试验证

另一类是测试执行活动，包括测试运行、测试报告、测试结果分析等。

优点：

a) 测试与其他流程并发的进行
b) 测试准备和测试执行分开

3） V&V 模型




优点：

a) 测试与其他流程并发的进行
b) 测试准备和测试执行分开
c) 测试过程子阶段与开发过程子阶段一一对应。

V&V 的含义

验证（Verification）和确认（Validation）

验证：（“Are we building the product right?”）

1） 验证是保证软件正确地实现特定功能的一系列活动
2） 验证是检测每一阶段形成的工作产品是否与前一阶段定义的规格相一致

确认：（“Are we building the right product?”）

1） 确认是指保证所生产的软件可追溯到用户需求的一系列活动
2） 确认是检测每一阶段的工作产品是否与最初定义的软件需求规格相一致

# 测试过程规范

CMM 关于过程的要素

1） 角色（Roles）：人
2） 入口准则（Entry Criteria）：执行活动所必须满足的条件
3） 输入（Inputs）：完成某活动所需要加工或参考的资料、原材料
4） 活动（Activities）：流程由一系列有相互关系的活动组成
5） 输出（Outputs）：完成某活动后所提交的工作产品
6） 出口准则（Exit Criteria）：完成或退出某活动所必须满足的条件
7） 评审和审计（Reviews and Audits）
8） 可管理和受控的工作产品（Work Products Managed and Controlled）

9） 测量(Measurements)：客观指标（一组数据）

10） 书面规程（Documented Procedures）

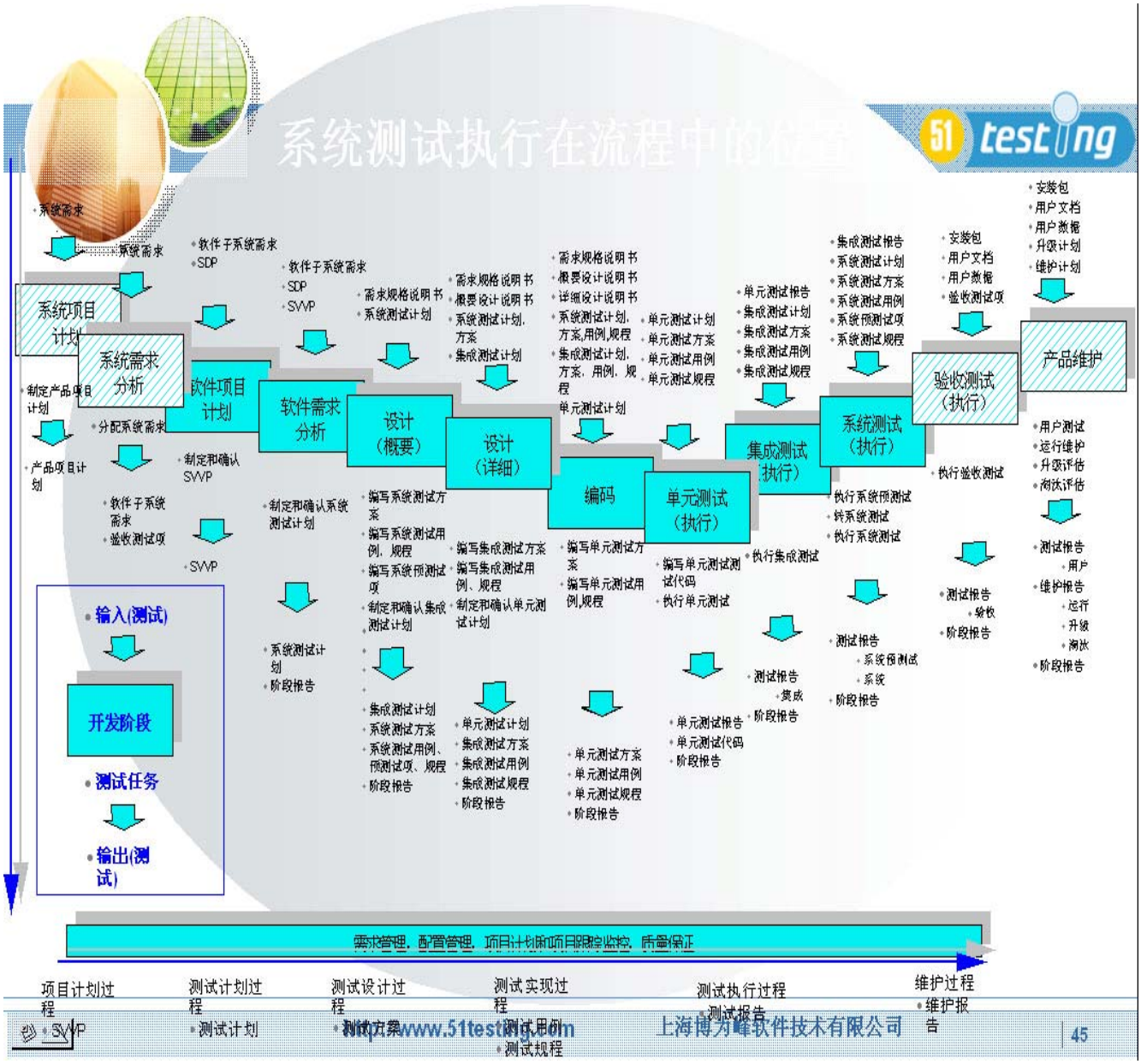
11） 培训（Training）：技术支持

12） 工具（Tools）：辅助说明

13） 职责：权责定义

14） 模板：标准格式

15） 检查表（Checklist）：要点列表



# 软件质量

## 软件质量的定义

质量：实体基于这些特性满足需求的程度。（一个实体的所以特性，基于这些特性可以满足明显的和隐含的需求）

软件质量的三个层次：（需求的分层导致质量也分层）

1） 符合需求规格：符合开发者明确定义的目标，即产品是不是在做让它做的事情。目标是开发者定义的，并且是可以验证的。

2） 符合用户显示需求（基于 SRS）：符合用户所明确说明的目标。目标是客户所定义的，符合目标即判断我们是不是在做我们需要做的事。

3） 符合用户的实际需求：实际需求包括用户明确说明的和隐含的需求。

影响软件质量的因素：（铁三角）

1） 流程

好处：将不可见的工作过程变得可见可控；使得整个工作过程有序并减少内耗，提高工作效率。

2） 技术（设计、开发、测试）

企业技术负载于人（现有职工的技术；企业是否重视技术积累）

技术与流程的关系：有技术，无流程不可能进行现代化的软件开发；有流程，无技术不可能生产高质量的产品

3） 组织（非直接的）

通过对流程和技术产生作用而间接对产品质量产生影响。

组织对流程的影响（组织应该将流程制度化，规范化以保证其执行效率；当流程执行中遇到阻碍时，组织应给予处理，保证流程顺畅执行）

组织对技术的影响（保证有能力的人去做合适的事情（资源调配）；组织重视并组织技术的积累，建立知识库（财富库））

## 软件质量管理体系

1） ISO9000

ISO9000 族 2000 版标准主要由 ISO9000、ISO9001、ISO9004 三个核心标准组成。

2000 版的八项质量管理原则：

a) 以客户为中心（在同一组织内部，顾客的定义是下游环节的人员是上游环节人员的顾客）
b) 领导作用（1 个制定，4 个确保，1 个创造，2 个决定，1 个评审）
c) 全员参与
d) 过程方法
e) 管理的系统方法
f) 持续改进
g) 基于事实的决策方法
h) 互利的供方关系

八项质量管理原则的意义：

a) 是质量管理的理论基础
b) 用高度概括，易于理解的语言所表述的质量管理的最基本、最通用的一般性规律
c) 为组织建立质量管理体系提供了理论依据
d) 是组织的领导者有效的实施质量管理工作必须遵循的原则。

2） CMM（Capability Maturity Model）/CMMI（Capability Maturity Model Integration）

评估软件承办商能力；协助软件组织改进过程，提高过程能力

基本术语：

a) KPA（Key Process Area）关键过程域（过程域简单的说就是做好一个事情的某个方面，对于软件开发而言就是做好软件开发的某个方面）
b) 如果该级别的全部 PA 达到要求了，就认为该级别达到了
c) 如果判断 PA 达到要求呢？（每个 PA 包含几个目标（Goal）；如果这几个目标都达到要求了，就认为该 PA 达到要求了）
d) 如何判断 Goal 达到要求呢？（每个 Goal 包含几个实践（Practice）；每个实践达到要求了，就认为该 Goal 达到要求了）

CMM/CMMI 用途

a) 可以识别组织的长处和弱处
b) 评估组织用以来评价软件承包商的能力和风险
c) 领导可以借此来进行过程改进，提高企业软件生产能力
d) 开发和技术人员参照 CMM/CMMI 进行执行过程改进

CMM/CMMI 的选择

a) 企业本身项目特点（软件开发用 CMM；有软件开发且包括硬件和采购用 CMMI）
b) 考虑企业自身的能力成熟度
c) 企业对经费的预算
d) 若企业只想在某个方面（如过程）提高进行改进（使用 CMMI）
e) CMM 向 CMMI 的转型

CMM/CMMI 区别

a) 降低了复杂度和规模；扩大了模型覆盖率；表达方式（CMM：阶段式表示；CMMI：阶段式（初始级、可重复级、已定义级、已管理级、优化级）、连续式（管理类、支持类、项目类、过程类））
b) CMMI 强调对需求的管理；加强对工程过程的重视，强调度量；加强了对风险的管理；CMM 中的“组间协调”在 CMMI 中作为“集成化项目管理”CMM 中的一个目标；中的 KPA“同行评审”在 CMMI 中抽象为 KPA“验证”；
c) CMM 是作为评估标准出现的，是“必要”是才能保证评估的标准；CMMI 是作为改进模型出现的，罗列了较多的最佳实践，易于过程改进。
d) CMM 主要是针对软件的

CMM/CMMI 的各级特点

a) 初始级（Initial）过程能力是不可预测的，过程是无序的。
b) 可重复级（Repeatable）过程能力是有纪律的。
c) 已定义级（Defined）过程能力为标准的和一致的。（SEPG 软件工程过程组）
d) 已管理级（Managed）过程能力为可预测的。
e) 优化级（Optimizing）过程能力的基本特征是不断改进，不断改善其项目的过程性能。

ISO9001 和 CMM 的关系

a) 最大的相似点（强调管理、过程、规范化和文档化）
b) 不同点（CMM 把焦点严格对准软件；ISO9001 的范围包括硬件、软件、流程性材料和服务）
c) 两者之间的联系（CMM2 和 ISO9001 强相关；CMM 的每个关键过程域至少按某种解释与 ISO9001 弱相关）

3） 六西格玛（本质是一个全面管理概念，而不仅仅是质量提高手段）

六西格玛管理法是以质量作为主线，以客户为中心，利用对事实和数据的分析，改进提升一个组织的业务流程能力，从而增强企业竞争力，是一套灵活的，综合性的管理方法体系。

六西格玛管理法原则：（与 ISO9000 族 2000 版的八大原则进行比较）

a) 注重客户
b) 注重流程
c) 全员参与
d) 预防为主
e) 事实依据的决定
f) 持续和突破性改进

六西格玛改进区域：

a) 周期时间（流程速度、回应能力）
b) 输出物的变差（产品或服务的直通率，缺陷成本降低，客户满意升高）
c) 营运效率（更低成本）

六西格玛的实施模式（DMAIC）

a) 定义（Define）提出问题，确定目标
b) 测量（Measure）收集资料，寻找原因
c) 分析（Analysis）研究资料，确定原因
d) 改进（Improve）优化解决方案
e) 控制（Control）推行控制系统

三大质量管理体系的区别：

ISO9000 是不分行业的质量管理体系；CMM/CMMI 只适用于软件行业的质量管理体系；六西格玛是考虑质量、成本、进程三方面的不分行业的质量管理体系。

# 软件质量模型

项目和产品的区别（依据需求来源不同）：

项目：由特定用户提出，以合同、契约为方式表现，企业需求人员获得；

产品：由企业内部的市场人员进行对潜在客户群进行分析后得出。

质量模型：一组特性及特性之间的关系，它提供规定质量需求和评价质量的基础。

a) 内部质量：从接收到用户的原始需求开始到产品交付用户之间的所有中间过程产品的质量（由开发与测试人员决定）（影响因素“铁三角”流程最主要）
b) 外部质量：软件系统作为一个整体运行时所体系出来的特性（系统测试-测试人员决定）
c) 使用质量：用户评价

软件质量模型

1） 软件功能性（核心）

当软件在指定条件下使用时，软件产品提供满足明确和隐含需求的功能的能力。

a) 适合性（Suitability）：软件产品为指定的任务和用户目标提供一组合适的功能的能力。
b) 准确性（Accuracy）：软件产品提供具有所需精确的正确或相符的结果或效果的能力。
c) 互操作性（Interoperabiblity）：软件产品与一个或更多的规定系统进行交互的能力。
d) 保密安全性（Security）：软件产品保护信息和数据的能力，以使未授权的人员或系统不能阅读或修改这些信息和数据，而不拒绝授权人员或系统对它们的访问。
e) 功能性的依从性

2） 软件可靠性

在指定条件下使用时，软件产品维持规定的性能级别的能力。

a) 成熟性（Maturity）：软件产品为避免由软件中错误而导致失效的能力。
b) 容错性（Fault Tolerance）：在软件出现故障或者违反指定接口的情况下，软件产品维持规定的性能级别的能力。
c) 易恢复性（Recoverability）：在失效发生的情况下，软件产品重建规定的性能级别并恢复受直接影响的数据的能力。（MTTR 平均恢复时间和恢复业务的程序）
d) 可靠性的依从性

3） 软件易用性

在指定条件下使用时，软件产品被理解、学习、使用和吸引用户的能力

a) 易理解性（Understandability）：软件产品使用用户能理解软件是否合适以及如何能将软件用于特定的任务和使用环境的能力。
b) 易学性（Learnability）：软件产品使用户能学习其应用的能力。
c) 易操作性（Operability）：软件产品使用户能操作和控制它的能力。
d) 吸引性（Attractiveness）：软件产品吸引用户的能力。
e) 易用性的依从性

4） 软件效率（性能测试重点）

在规定条件下，相对于所用资源的数量，软件产品可提供适当性能的能力。

a) 时间特性（Time Behavior）：在规定条件下，软件产品执行其功能时，提供适当的响应和处理时间以及吞吐率的能力。（即完成用户的某个功能需要的响应时间（响应时间是从发起请求到收到成功提示信息））
b) 资源利用性（Resource Utilization）：在规定条件下，软件产品执行其功能时，使用合适的资源数量和类别的能力。
c) 效率依从性

5） 软件维护性

软件产品可被修改（修正、改进或软件对环境、需求和功能规格说明变化的适应）的能力。

a) 易分析性（Analyzability）：软件产品诊断软件中的缺陷或失效的原因或识别待修改部分的能力。
b) 易改变性（Changeability）：软件产品使指定的修改可以被实现的能力。
c) 稳定性（Stability）：软件产品避免由于修改而造成意外结果的能力。
d) 易测试性（Testability）：软件产品使已修复软件能被确认的能力。
e) 维护性的依从性

6） 软件可移植性

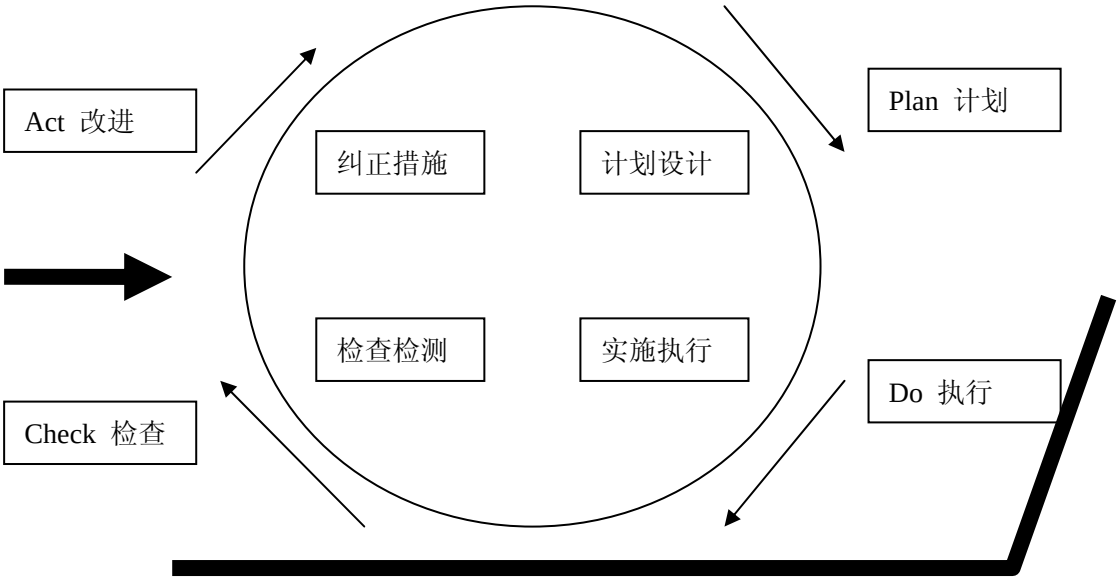
软件产品从一种环境迁移到另一种环境的能力。

a) 适应性（Adaptability）：软件产品无需采用有别于为考虑该软件的目的而准备的活动或手段就可能适应不同的指定环境的能力。

b) 易安装性（Installability）：软件产品在指定环境中被安装的能力（安装测试）。
c) 共存性（Co-existence）：软件产品在公共环境中同与其分享公共资源的其他独立软件共存的能力。
d) 易替换性（Replaceablity）：软件产品在同样环境下，替代另一个相同用途的指定软件产品的能力。
e) 可移植性的依从性

# 软件质量活动（软件质量保证（SQA）和测试）

SQA 和测试的关系：
a) SQA 从流程方面保证了软件的质量
b) 测试从技术方面保证了软件的质量
c) 只进行 SQA 活动或者只进行测试活动不一定能产生很好的软件质量。
SQA 的主要工作范围：（被称为老师，医生，警察）
a) 指导（指导项目成员执行过程，培训）并监督（过程的执行是否符合规范）项目安装过程实施
b) 对项目进行度量（度量数据的采集，度量使得不可见的智力过程变得可见）、分析，增加项目的可视性
c) 审核（审计过程产品是否符合相关模块，审计问题产生原因）工作产品，评价工作产品和过程质量目标的符合度
d) 进行缺陷分析（提出过程改进意见给 SEPG），缺陷活动预防，发现过程的缺陷，提供决策的参考，促进过程的改进
SQA 需要职能：
a) 软技能（个人素质、沟通能力）自我修炼
b) 掌握项目管理
c) 熟知软件工程
d) 了解业务知识
e) 熟练掌握过程改进体系
质量管理 PDCA 循环（螺旋式上升逐步实现质量目标）：
Plan 计划：制定计划，明确目标；基于目标的方法步骤
Do 执行：执行 Plan
Check 检查：检查实际执行结果与计划中预期目标的差距（目标的实现程度，若存在差距，分析原因）
Act 改进：根据分析原因给出明确方案并制定下一轮过程改进目标




软件度量的概念和目的：

度量：对事物属性的量化表示

软件度量：是指计算机软件中范围广泛的测度，包括对软件系统、构件或生命周期过程具有的某个给定属性的度的一个定量测量

目的：

a) 提高软件生产率，缩短产品研发周期，降低研发成本、维护成本（对于开发商）
b) 提高软件产品质量，提高用户满意度（对用户）
c) 为组织持续改进提供量化的指标和反馈（对开发商长远）

软件度量的作用：

a) 理解
b) 预测
c) 评估
d) 改进

软件度量分类：

四个基本度量项：

a) 规模（Size）：软件产品的大小
b) 工作量（Effort）：完成各软件工作产品和活动所用人时（或人天等）
c) 进度（Schedule）：各软件工作产品和活动开始和结束的时间
d) 质量（quality）-缺陷(defect)：在各软件工作产品和活动中产生的缺陷数

其他度量指标：

a) 缺陷密度：研发活动发现缺陷密度；研发活动引入缺陷密度；工作产品缺陷密度
b) 生产率：SRS、HLD、LLD 阶段文档生产率：页/人天；编码阶段生产率：KLOC/人天；UT、IT、ST 用例设计阶段生产率：用例/人天
c) 测试执行效率：执行用例数/人天
d) 用例密度：用例数/KLOC
e) 。。。。。。。。。。。。。。。。。。。。

# 测试方法

是否需要了解软件内部结构（黑盒测试和白盒测试）注灰盒测试
是否需要执行被测对象（静态测试和动态测试）
是否需要借助自动化脚本或工具进行测试（人工测试和自动化测试）

## 黑盒测试和白盒测试

什么是白盒测试（基于程序结构的逻辑驱动测试）？
白盒测试是依据被测软件分析程序内部构造，并依据内部构造设计用例，来对内部控制流程进行测试，可完全不顾程序的整体功能实现情况。
（玻璃盒测试，透明盒测试，开放盒测试，结构化测试，逻辑驱动测试）】
为什么进行白盒测试？

a) 白盒测试一般在测试前期进行，通过达到一定的逻辑覆盖率指标，使得软件内部逻辑控制结构上的问题能基本得到消除
b) 白盒测试能保证内部逻辑结构达到一定的覆盖程度，能够给予软件代码质量的更大保证
c) 白盒测试发现问题后解决问题的成本较低

白盒测试的常用技术：（静态分析和动态分析）

a) 静态分析：控制流分析、数据流分析、信息流分析等；

控制流分析

1） 相关概念

程序元素：一个程序元素通常是一个条件，一个简单的语句或者一块语句（多个连续语句）

控制流关系：一个程序的控制流关系叙述了程序元素和它们执行的次序之间的联系

控制流图：对应于控制流关系的图

控制流矩阵：由控制流图得到，反映相邻程序元素之间的先后顺序关系

2） 控制流分析步骤

确定所有程序元素；

根据程序元素之间的相互关系得到控制流图；

将控制流图转换成控制流矩阵；

通过数据结构的形式把控制流矩阵表示出来；

借助算法对控制流进行分析，找出存在的问题；

3） 控制流分析可以发现的问题

确保写出的程序不应包含：

转向并不存在的标号；

没有用的语句的标号；

从程序入口进入后无法达到的语句；

不能达到停机语句的语句。

数据流分析

1） 相关概念

数据流分析法关键是数据的定义和引用。

数据的定义：如果程序中某一语句执行时能改变某程序变量 V 的值，则称 V 是被该语句定义的。

数据的引用：如果一语句的执行引用了内存中变量 V 的值，则说该语句引用变量 V。

2） 数据流分析步骤

根据代码得到数据流表；

分析数据流表找到以下两种错误：（变量未定义但被引用；变量定义但未被引用）；

根据分析结果对代码进行修正和优化。

信息流分析

b) 动态分析：逻辑覆盖测试（分支测试、路径测试等）、程序插装等；

逻辑覆盖测试（分支测试、路径测试等）

程序插装

借助往被测程序中插入操作来实现测试目的的方法。（比如：打印语句，打印我们最关系的信息）。

白盒测试的特点：

a) 测试人员需要了解软件的实现；
b) 可以检测代码中的每条分支和路径；
c) 揭示隐藏在代码中的错误；
d) 对代码的测试比较彻底；
e) 实现代码结构上的优化；
f) 白盒测试投入大，成本高；
g) 白盒测试不验证规格的正确性。

什么是黑盒测试（基于规格的测试）？

黑盒测试把被测对象看成一个黑盒，只考虑其整体特征，不考虑其内部具体实现。

黑盒测试针对的被测对象可以是一个系统，一个子系统，一个模块，一个子模块，一个函数等。

常见的黑盒测试类型：

a) 功能性测试：一种是顺序测试每个程序特性或功能，另一种途径是一个模块一个模块的测试，即每个功能在其最先调用的地方被测试；
b) 容量测试：检测软件在处理海量数据时的局限性，能发现系统效率方面的问题；
c) 负载测试：检测系统在一个很短时间内处理一个巨大的数据量或执行许多功能调用上的能力；
d) 恢复性测试：主要保证系统在崩溃后能够恢复外部数据的能力。

常用的黑盒测试的方法：

等价类划分法；

边界值分析法；

因果图分析法；

判定表法；

状态迁移法；

。。。。。。。。。

黑盒测试的特点：

a) 对于更大的代码单元来说（子系统甚至系统）比白盒测试效率更高；
b) 测试人员不需要了解实现的细节，包括特定的编程语言；
c) 从用户的视角进行测试，很容易被大家理解和接受；
d) 有助于暴露任何规格不一致或有歧义的问题；
e) 没有清晰的和简明的规格，测试用例很难设计的；
f) 不能控制内部执行路径，会有很多内部程序路径没有被测试到；
g) 不能直接针对特定的程序段，这些程序可能非常复杂（因此可能隐藏更多的问题）。

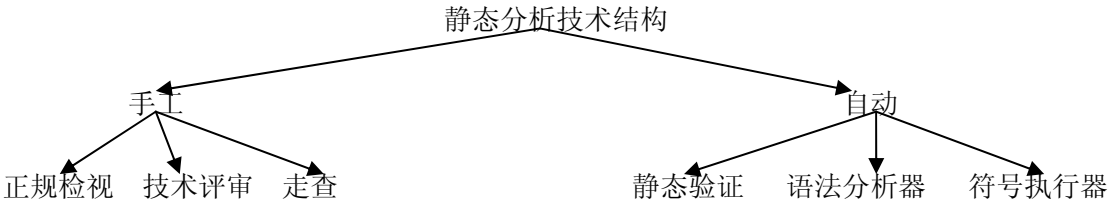
灰盒测试

利用被测对象的整体特性信息，采用黑盒测试方法；
利用被测对象的内部具体实现信息，采用白盒测试方法；
如果既使用被测对象的整体特性信息，又利用被测对象的内部具体实现信息，采用灰盒测试方法。两种信息占的比例不同，相应的灰度就不同。

## 静态测试和动态测试

静态测试：不运行被测试的软件系统，而是采用其他手段和技术对被测试软件进行检测的一种测试技术。（代码走读、文档评审、程序分析等）。
静态测试常用技术——静态分析技术：
定义：一种不通过执行程序而分析程序执行的技术。
功能：检查软件的表示和描述是否一致，没有冲突或者没有歧义，它描述的是纠正软件系统的描述、表示和规格上的错误，因此是任何进一步测试执行的前提。
主要有三种不同的程序测试可能性：

a) 考虑程序是否满足编码规则，语法上是否具有一致性和完整性；
b) 考虑文档描述是否规范、准确、便于查阅；
c) 考虑程序和文档之间的一致性。




手工静态分析（最重要的手工技术是同行评审（对象：计划、需求文档、设计图、代码等））：
根据同行评审形式正规的程度分为：

a) 正规检视：以某个方案的裁决为目的，形式比较严格，有固定的流程，多用于文档的评审；
b) 技术评审：以某个方案的裁决为目的，一般由企业高层技术人员和管理人员参与；
c) 走查：以发现软件产品中的缺陷为目的，没有严格规定，比较随意。

自动化静态分析

动态测试：按照预先设计的数据和步骤去运行被测软件系统，从而对被测软件系统进行检测的一种技术。
动态测试常用技术——动态分析技术：
定义：对软件系统运行行为进行分析，包含程序在受控的环境下使用特定的输入进行正式的运行，和期望的结果比较以检查系统运行是正确还是不正确。
常用的动态分析技术：
路径测试
分支测试
性能测试
。。。。。。。。

常用动态分析工具功能

| 动态分析类型 | 工具的功能 |
| --- | --- |
| 测试覆盖率分析（白盒） | 测试对代码的检测范围 |
| 跟踪（白盒） | 跟踪程序执行期间的所以路径，例如所有变量的值等 |
| 调整（黑盒） | 度量程序执行过程中使用的资源 |
| 模拟（黑盒） | 模拟系统的一部分，例如无法获得的代码或硬件 |
| 断言检查（白盒） | 测试在复杂逻辑结构中是否某个条件已经被给出 |

Logiscope 中的 Rulechecker(白盒静态技术)，Logiscope 中的 Testchecher（白盒动态技术）、TCL（白盒动态技术）。

# 人工测试和自动化测试

人工测试：测试活动（如评审、测试设计、测试执行等）由人工来完成，狭义上是指测试执行由人工完成，这是最基本的测试形式。
自动化测试：一般是指通过计算机模拟人的测试行为，替代人的测试活动，狭义上是指测试的执行由计算机完成。

自动化测试的意义：

a) 对程序新版本运行前一版本执行的测试，提高回归测试效率
b) 可以运行更多更频繁的测试，比如冒烟测试
c) 可以执行手工测试困难或不可能做的测试，比如大量的重复操作或者集成的测试
d) 更好地利用资源，比如测试仪器或者被测对象
e) 测试具有一致性和可重复性，即自动化测试的步骤和结果是完全一样的
f) 测试的复用性，即自动化测试脚本可以拆分开给其他测试脚本使用
g) 可以更快地将软件推向市场，软件发布前进行高效的回归测试，减少软件发布的时间
h) 增加软件的信任度，通过自动化测试提高了测试效率，把节约的时间拿出来做更多的测试。

自动化测试的限制：

a) 不能取代手工测试，自动化测试只能提供测试效率，不能提高测试的有效性，即不可能发现更多的缺陷
b) 手工测试比自动化测试发现的缺陷更多
c) 对测试设计依赖性极大，测试设计的不好会遗漏问题
d) 自动化测试对软件开发具有很大的依赖性，开发上出现变更可能导致前面的自动化测试完全失效
e) 工具本身并不具备想象力，工具不具有职能。

自动化测试的误区：
a) 不现实的期望，希望自动化能取代手工测试
b) 缺乏测试实践经验，手工测试都做不好，或者经验积累不够，就尝试自动化，很难成功
c) 期望自动化测试发现大量新的缺陷，自动化只能保证测试执行的效率，确保已有的问题不再发生，发现新缺陷不是其目的
d) 安全性错觉，认为进行自动化测试的软件就安全的，质量有保证的
（只有手工测试做好了，明确了测试观察点，才能把自动化测试做好，所以手工测试是自动化测试的一个基础）

# 需求管理

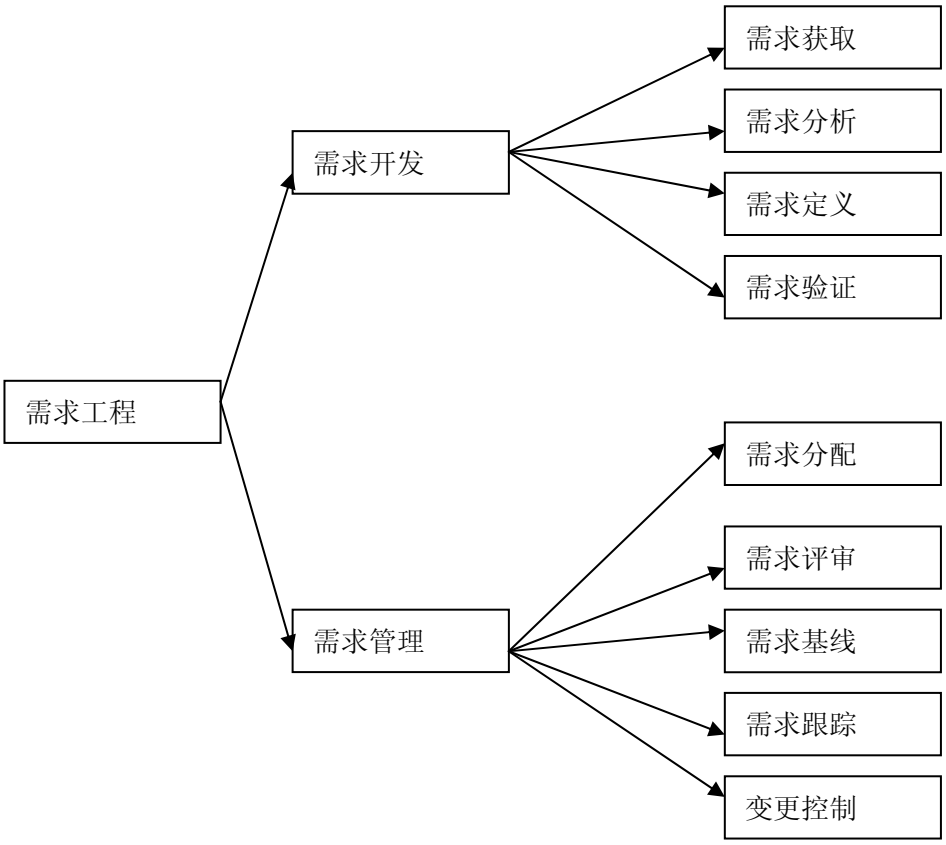
## 软件需求管理简介

需求（强调做什么（What）而不是如何做（How））的定义：
SRS 就是（1）解决用户问题或达到用户目标所需的条件或能力；2）为遵循合同、标准、规格或其他要求的正式文档，系统必须满足或拥有的条件或能力。）

需求管理的定义：使用户和项目团队之间，就不断变更的需求，达到并保持一致的过程。
（基线：
a) 评审后方可建立
b) 受控，需修改时必须经过 CCB（Change Control Board 变更控制委员会）批准
c) 是下一步开始工作的基础）

需求管理的要求：
a) 软件需求要基线化
b) 软件需求的实现要跟踪，要记录，要标示
c) 要测量软件需求
d) 要验证软件需求

# 软件需求工程介绍




需求开发：

a) 需求获取：根据需求来源的不同分为：项目和产品
b) 需求分析：1）根据用户提出的显示需求，充分的挖掘其背后隐藏的隐式需求；2）严格定义所以需求（显示和隐式）的规格（功能、性能、约束、质量、其他）
c) 需求定义：将所获取的所有需求以规范化的语言表示
d) 需求验证：验证需求是否得到实现

（需求陷阱：缺少用户参与；用户扁平需求；项目范围不断扩大；切忌需求没完没了；切忌模糊不清的需求等）

需求管理（CMM 二级的第一个 KPA）：

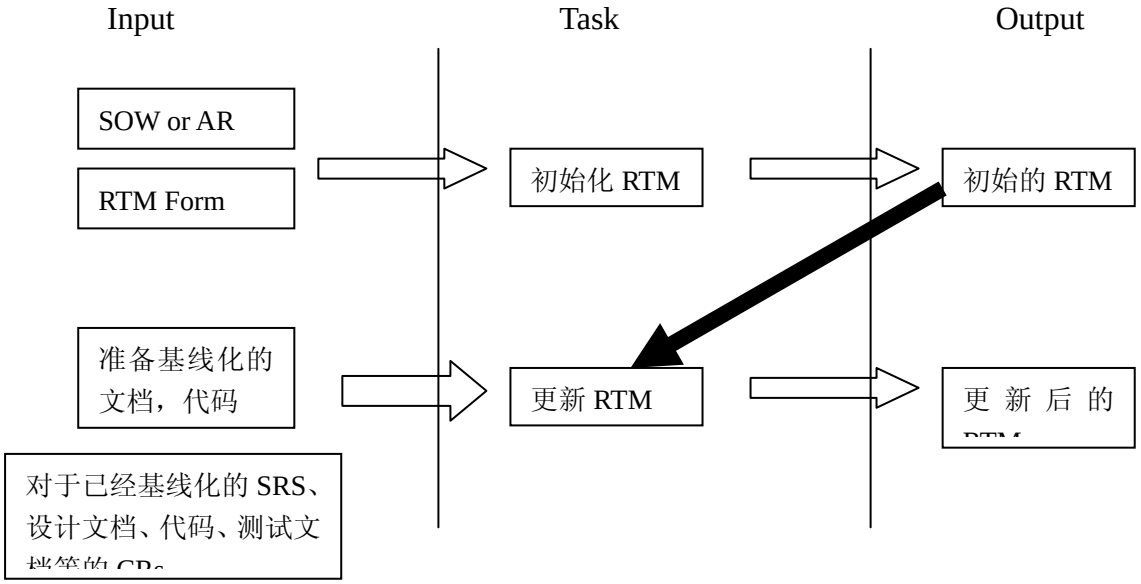
a) 需求分配：需求本身是分层次的（业务需求；用户需求；软件需求），并非所有项目都有需求分配；
b) 需求评审：评审已分配的需求（原始需求）；对已形成的 SRS 进行评审
c) 需求基线：建立需求基线使得 SRS 可控
d) 需求跟踪：贯穿于整个软件开发过程
e) 变更控制：需求变更的控制（失效的变更控制；没有进行变更控制（几乎不存在））

软件需求变更：

a) 需求变更可能发生在任意阶段
b) 要求变更的需求进行变更控制
c) 变更的基础是需求基线

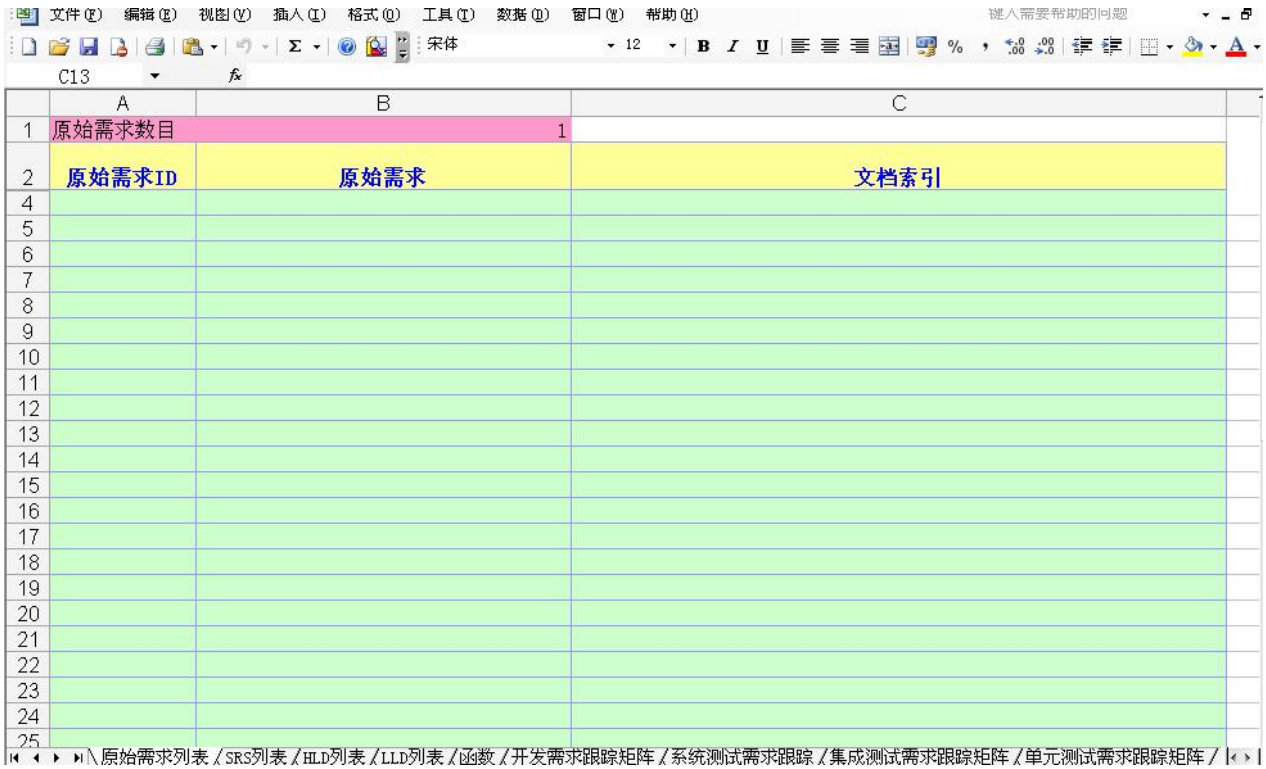
# 软件需求跟踪流程介绍

软件需求跟踪流程




SOW为工作说明书；AR为原始需求；CRs为需求变更

RTM（Requirement Truceablity Matrix）需求跟踪矩阵



（注意 Excel 最下排的不同需求列表）

RTM 初始化：

a) 将原始需求列入原始需求列表
b) 将 SRS 列入 SRS 列表
c) 在开发需求跟踪矩阵中建立 AR 与 SRS 的关系（在开发中确保需求被实现）
d) 在系统测试需求跟踪矩阵中建立 SRS 与 ST 的关系（在测试中确保需求被验证）

## 软件需求跟踪方法

需求项，概要设计项等的定义。

# 通用测试用例写作方法

## 软件测试用例格式

| | |
|---|---|
| 测试用例编号 | N3310_IT_FILEITF_READFILE_004 |
| 测试项目 | 测试模块 A 提供的文件接口 |
| 测试标题 | 文件 B 正在被其他进行执行读/写操作，通过 A 模块的文件接口读取文件 B 中的数据 |
| 重要级别 | 高 |
| 预置条件 | 进程 XProcess 被创建并启动 |
| 输入 | 。。。。 |
| 操作步骤 | 。。。 |
| 预期输出 | 。。。 |

## 测试用例的写作要点

a) 测试用例编号
   规则：由字符和数字组合成的字符串，用例编号应具有唯一性、易识别性
   约定：系统测试用例：产品编号-ST-系统测试项名-系统测试子项名-XXX
   。。。。。。。。。。。。。。。。。。。。。。。。
b) 测试用例项目
   规则：当前测试用例所属测试大类、被测需求、被测模块、被测单元等
   约定：系统：软件需求项；集成：集成后的模块名或接口名；单元：被测试的函数名
c) 测试用例标题
   规则：测试用例的简单描述，需要用概括的语言描述该用例的出发点和关注点，原则上每个用例的标题不能重复
d) 测试用例重要级别
   规则：高：保证系统基本功能、核心业务、重要特性、实际使用频率比较高的用例
   中：重要程度介于高和低之间的测试用例

低：实际使用频率不高、对业务功能影响不大的模块或功能的测试用例
（对于流程：基本流的级别为高；备选流的级别为低）

e) 测试用例预置条件
规则：执行当前测试用例需要的前提条件，如果这些前提条件不满足，则后面的测试步骤无法进行或者无法得到预期的结果

f) 测试用例输入
规则：用例执行过程中需要加工的外部信息。根据软件测试用例的具体情况，有手工输入、文件、数据库记录等等

g) 测试用例操作步骤
规则：执行当前测试用例需要经过的操作步骤，需要明确的给出每个步骤的描述，测试用例执行人员可以根据该操作步骤完成测试用例执行

h) 测试用例预期结果
规则：当前测试用例的预期输出结果，包括返回值的内容、界面的响应结果、输出结果的规则符合度等等

## 测试用例的写作检查规则

a) 测试用例标识是否按照测试方案的规则来编写
b) 是否每个测试用例的预置条件都被描述清楚？
c) 每个测试用例的“输入“中是否列出了所以测试的输入数据？
d) 测试用例的“预期结果“是否完整而且清晰？
e) 是否明确说明了每个测试用例或测试用例集的重要级别？
f) 是否明确说明了测试用例的执行顺序？

# 软件缺陷管理

## 软件缺陷管理基本概念

注意五个关于缺陷管理的名词：BUG，缺陷（Defect），错误（Error），故障（Fault），失效（Failure）
缺陷报告单（前面在缺陷类型中提到的五种缺陷都应写如其中）
缺陷管理的目的：

a) 保证信息的一致性
b) 保证缺陷得到有效的跟踪，解决
c) 获取正确的 Bug 信息，用作缺陷分析和产品度量

## 软件缺陷管理基本流程

Bug 的生命周期：一个缺陷从最初的提交，经过若干处理，最后关闭的过程。

缺陷的相关属性：

a) 缺陷发现人
b) 缺陷发现时间
c) 缺陷状态
d) 缺陷严重程度

严重性：软件缺陷对软件质量的破坏程度，即此软件缺陷的存在将对软件的功能和性能产生怎样的影响。

1） 致命：例如，软件的意外退出甚至操作系统崩溃，造成数据丢失。
2） 严重：例如，由于单功能失效导致多个相关功能均失效。
3） 一般：例如，软件的单功能失效。
4） 提示：软件界面的细微缺陷。例如，某个控件没有对齐，某个标点符合丢失等

依据（工作量、质量、进度、成本）修改缺陷的优先级分为：紧急的，恰当的，范围内的

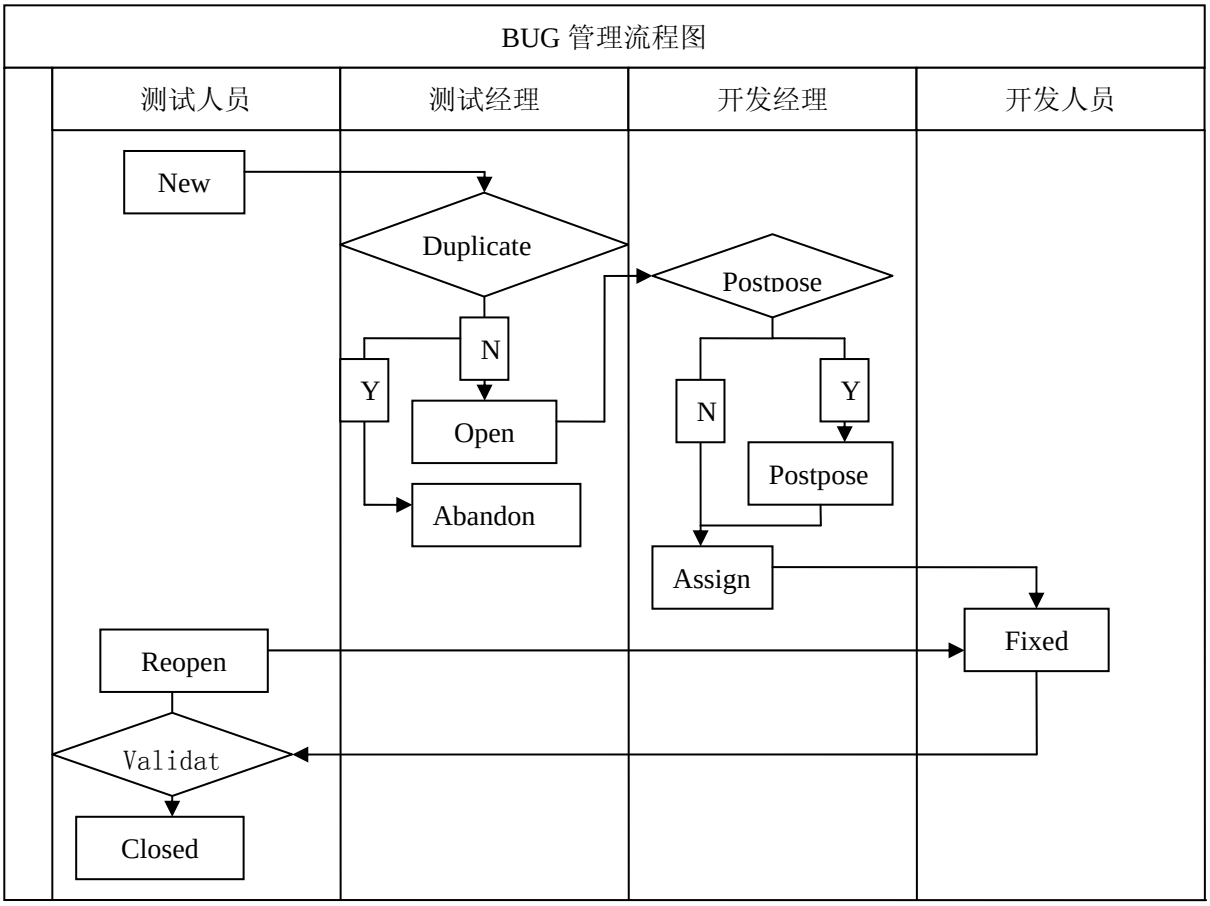
e) 缺陷所属版本
f) 缺陷修改日期

TD（QC）中常用软件缺陷状态列表：

New：缺陷的初始状态
Open：开发人员开始修改缺陷
Fixed：开发人员修改缺陷完毕
Closed：回归测试通过
Reopen：回归测试失败
Postpone：推迟修改
Rejected：开发人员认为不是程序问题，拒绝缺陷
Duplicate：与已经提交的 Defect 重复
Abandon：被 Rejected 和 Duplicate 的 Defect，测试人员确认后的确不是问题，将 Defect 置为此状态

软件测试缺陷管理流程：

a) 提交一个 Bug
b) 审查 Bug 并分配
c) 相关负责人员修改 Bug
d) 回归测试




# 缺陷跟踪单填写方法

缺陷跟踪单写作准则（5C）：

a) Correct（准确）：每个组成部分的描述准确，不会引起误解
b) Clear（清晰）：每个组成部分的描述清晰，易于理解
c) Concise（简洁）：只包含必不可少的信息，不包括任何多余的内容
d) Complete（完整）：包含复现该缺陷的完整步骤和其他本质信息
e) Consistent（一致）：按照一致的格式书写全部缺陷报告

缺陷跟踪单基本内容

| 缺陷项目 | 注意事项 |
|---|---|
| 简单描述 | 1、用一句话简单的描述清楚问题（Headline“看到什么“） |
| 详细描述 | 1、描述问题的基本环境，包括操作系统、硬件环境、网络环境、被测试软件的运行环境<br>2、用简明扼要的语言描述清楚软件出现异常的时候的测试人员的操作步骤和使用数据<br>3、如果从 GUI 界面上可以反映出软件的异常，采用拷屏的方式截屏，粘贴在问题单中<br>4、被测试软件运行时候的相关日志文件<br>5、测试人员根据上述信息可以给出对问题的简单的分析<br>6、被测试软件的版本 |

| | |
|---|---|
| | 7、状态、严重级别、优先级<br>8、提交日期、提交人 |
| 相关附件 | 1、GUI 的拷屏图片<br>2、被测试软件运行的相关日志文件 |

缺陷报告的写作要点：

a) 再现：一般是尽量三次再现故障，如果问题是间断的，那要报告问题发生频率
b) 初步定位：可能影响再现的变量，例如配置变化、工作流、数据库，这些都可能改变错误的特征
c) 推广：确定系统的其他部分是否可能出现这种错误，以及使用不同的数据时是否存在着这种问题等等，特别是这些可能存在更加严重特征的部分
d) 压缩：精简任何不必要的信息，特别是冗余的测试步骤
e) 去除歧义：使用清晰的语言，尤其是避免使用那些有多个不同或相反含义的词汇
f) 中立：公正的表达自己的意思，对错误及其特征的事实进行陈述，避免夸张、幽默和讽刺
g) 评审：至少有一个同行，最好是一个有经验的测试工程师或测试经理，在递交错误报告之前自己先阅读一遍。

# 测试覆盖率

## 覆盖率概念

a) 覆盖率是用来度量测试完整性的一个手段。覆盖率是测试技术有效性的一个度量。
   覆盖率=（至少被执行一次的 Item 数）/item 的总数
b) 覆盖率大体可划分为两大类：逻辑覆盖率和功能覆盖率
c) 测试用例设计不能一味追求覆盖率，因为测试成本随覆盖率的增加而增加。

## 逻辑覆盖率

主要类型：

a) 语句覆盖（Statement Coverage）
   在测试时运行被测程序后，程序中被执行到的可执行语句的比率：
   语句覆盖率=（至少被执行一次的语句数量）/（可执行的语句总数）
b) 判断覆盖（Decision Coverage）或分支覆盖（Branch Coverage）
   在测试时运行被测程序后，程序中所有判断语句的取真分支和取假分支被执行到的比率
   判断覆盖率=（判断结果被评价的次数）/（判断结果的总数）
c) 分支条件覆盖率（Branch Condition Coverage）或判断条件覆盖率（Decision Condition Coverage）
   在测试时运行被测程序后，所有判断语句中的每个条件的所有可能值（为真或为假）和每个判断本身的判定结果（为真或为假）出现的比率

分支条件覆盖率=（条件操作数值或判定结果至少被评价一次的数量）/（条件操作数值总数+判定结果总数）

d) 路径覆盖率（Path Coverage）

在测试时运行被测程序后，程序中所有可能的路径被执行过的比率

路径覆盖率=（至少被执行到一次的路径数）/（总的路径数）

1） 路径能否全面覆盖在软件测试中是个重要的问题，如果程序中的每一条路径都得到考验，才能说程序受到了全面检验

2） 即使对于路径数有限的程序已经做到了路径覆盖，仍然不能保证被测程序的正确性

## 其他覆盖率

a) 功能覆盖率（Function Coverage）：属于黑盒测试范畴

b) 面向对象的覆盖率

1） 继承上下文覆盖

2） 基于状态的上下文覆盖

3） 已定义用户上下文覆盖

c) 函数覆盖

d) 指令块覆盖

e) 判定路径覆盖

# 单元测试

## 单元测试的定义和目的

什么是单元测试？

a) 单元测试是对软件基本组成单元进行测试，如函数(function 或 procedure)或一个类的方法（method）

b) 单元具有一些基本属性，如：明确的功能、规格定义，明确的与其他部分的接口定义等，可清晰地同一程序的其他单元划分

c) 基本单元不一定是指一个具体的函数或一个类的方法

d) 在具体实现时，也可能对应的是多个程序文件中的一组函数

单元测试的目的

在于发现各模块内部可能存在的各种错误，主要是基于白盒测试

a) 验证代码是与设计相符合的；

b) 发现设计和需求中存在的错误

c) 发现在编码过程中引入的错误

# 单元测试关注的重点

a) 单元接口

对如下进行测试：

1） 被测单元的输入输出参数在个数、属性、顺序上是否和详细设计中的描述保持一致

2） 是否修改了只做输入用的形式参数

3） 约束条件是否通过形式参数来传送

b) 局部数据结构

检查一些各种错误：

1） 检查不正确或不一致的数据类型说明

2） 使用尚未赋值或尚未初始化的变量

3） 错误的初始值或错误的缺省值

4） 变量名拼写错误或书写错误

5） 不一致的数据类型

c) 独立路径

对基本执行路径和循环进行测试会发现大量的错误。设计测试用例查找由于错误的计算、不正确的比较或不正常的控制流而导致的错误。

1） 运算的优先次序不正确或误解了运算的优先次序

2） 运算的方式错误

3） 不同数据类型的比较

4） “差 1 错”。即不正确的多循环或少循环一次

5） 错误的或不可能的循环终止条件

6） 关系表达式中不正确的变量和比较符

7） 当遇到发散的迭代时不能终止的循环

8） 不适当地修改了循环变量等

d) 出错处理

比较完善的单元设计要求能预见出错的条件，并设置适当的出错处理，已使在程序出错时，能对出错程序重新做安排，保证其逻辑上的正确性

1） 出错的描述难以理解

2） 出错的描述不足以对错误定位和确定出错的原因

3） 显示的错误与实际的错误不符

4） 对错误条件的处理不正确

5） 在对错误进行处理前，错误条件已经引起系统的干预等

e) 边界条件

常见错误：

1） 在 N 次循环的第 N 次，取最大最小值时容易发生错误

2） 特别要注意数据流，控制流中刚好等于、大于、小于确定的比较值时出现错误的可能性

# 单元测试环境

必须为每个单元测试开发驱动单元和桩单元

a) 驱动单元（Driver）：所测函数的主程序，它接收测试数据，并把数据传送给所测试单元，

最后再输出实测结果。当被测试单元能完成相关功能时，也可以不要驱动单元。

1） 接收测试数据，包含测试用例输入和预期输出

2） 把测试用例输入传送给要测试的单元

3） 将被测单元的实际输出和预期输出进行比较，得到测试结果

4） 将测试结果输出到指定位置

b) 桩单元（Stub）：用例代替所测单元调用的子单元

1） 桩单元的功能是从测试角度模拟被调用的单元

2） 桩单元需要针对不同的输入，返回不同的期望值，模拟所代替单元的不同功能

3） 桩单元返回的期望值根据输入和被测模拟单元的详细设计来确定

# 单元测试策略

a) 孤立的测试策略

方法：不考虑每个模块与其他模块之间的关系，为每个模块设计桩模块和驱动模块。每个模块进行独立的单元测试。

优点：该方法是最简单，最容易操作的，可以到达高的结构覆盖率，它是纯粹的单元测试

缺点：桩函数和驱动函数工作很大，效率低

b) 自顶向下的单元测试策略

方法：先对最顶层的单元进行测试，把顶层所调用的单元做成桩模块。其次对第二层进行测试，使用上面已测试的单元做驱动模块。如此类推直到测试完所有模块。

优点：可以节省驱动函数的开发工作量，测试效率较高

缺点：随着被测单元一个一个被加入，测试过程将变得越来越复杂，并且开发和维护的成本将增加。

c) 自底向上的单元测试策略

方法：先对模块调用层次图上最低层的模块进行单元测试，模拟调用该模块的模块做驱动模块。然后再对上一层做单元测试，用下面已经被测试过的模块做桩模块。以此类推，直到测试完所有模块。

优点：可以节省桩函数的开发工作量，测试效率较高

缺点：不是纯粹的单元测试，底层函数的测试质量对上层函数的测试将产生很大的影响。

# 单元测试过程

单元测试的四个阶段

a) 单元测试计划阶段：完成单元测试计划

b) 单元测试设计阶段：完成单元测试方案

c) 单元测试实现阶段：完成单元测试用例、单元测试规程、单元测试脚本及数据文件

d) 单元测试执行阶段：执行单元测试用例，修改发现的问题并进行回归测试，提交单元测试报告

# 单元测试原则

a) 对全新的代码或修改过的代码进行单元测试
b) 单元测试根据单元测试计划和方案进行，排除测试的随意性
c) 必须保证单元测试计划、单元测试方案、单元测试用例等经过评审
d) 当测试用例的测试结果与预期结果不一致时，单元测试的执行人员需如实记录实际的测试结果
e) 只有当测试计划中的结束标准达到时，单元测试才能结束
f) 对被测试单元需达到的一定的代码覆盖率要求

# 单元测试执行过程

a) 单元测试执行时间安排
   编码完成之后，并且符合单元测试入口条件
b) 单元测试执行的活动
   1） 构造单元的测试环境
   2） 运行单元测试脚本，记录测试结果
   3） 修改错误，进行回归测试
   4） 单元测试分析和总结
   5） 提交单元测试报告
c) 单元测试执行的输入
   输入：单元测试计划、单元测试方案、单元测试用例和规程、驱动程序和桩模块、需求规格说明书、概要设计说明书、详细设计说明书
   输出：单元测试报告

# CppUnit自动化单元测试框架

CppUnit 是用于面向对象 c++程序的单元测试的框架，它提供了一系列的头文件和静态库。采用 CppUnit 所提供的单元测试框架，可以很方便的开发测试用例，并实现单元测试的“自测试”。
使用 CppUnit 开发出来的测试用例是固化的，对同一功能多次执行的测试用例是完全相同的，并实现了测试的自动化。
CppUnit 采用断言来判断测试用例的执行结果，并可将执行结果写入一个文本文件中。

CppUnit 基本概念：
a) 被测试类（CUT）：被测试的类，该类的一个实例对象或者类本身作为测试的对象
b) 被测试方法；被测试的类中的成员方法
c) 测试用例(test case)：是用来测试一个功能是否正确的一系列测试执行，是测试执行和统计的最基本单位
d) 测试工厂：一般给一个被测试类定义一个测试工厂，对该被测试类的被测试方法的测试用例和数据进行组装
e) 测试装置（test fixture）：容纳多个测试方法以及相关测试数据的类，它可以为多个测试

方法准备相关数据、测试上下文，进行公共的初始化和后处理

f) 测试套（test suite）：相关测试用例的集合，测试套之间可以相互嵌套，也就是说一个测试套可以包含一些测试用例，也可以包含其他测试套，一般来说测试套和被测试的方法对应，并且需要把测试套定义为测试装置类

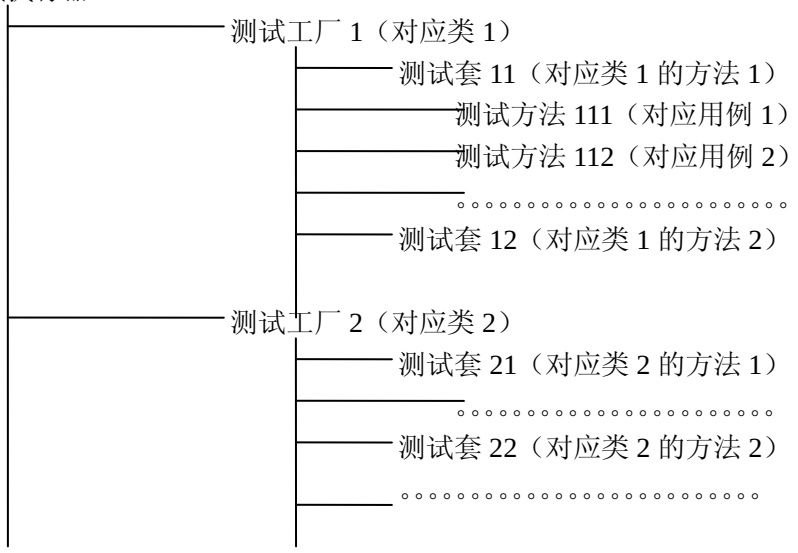
g) 测试方法（test method）：以一个函数形式表现出来的独立的测试执行，每个测试用例对应一个测试方法，但又不完全等同于测试方法。比如进行继承类的测试，虽然父类和子类有完全相同的测试方法，但是在父类和子类中对应于同样测试方法的测试用例是不相干的

h) 测试驱动（test driver）：就是给所有的测试用例提供执行环境的代码

i) 测试执行器（test runner）：负责实际执行测试驱动。在 CppUnit 中，一般在 main 函数或其他的入口点定义一个测试执行器，然后把要执行的测试套全部加入该执行器，调用该执行器的 run 函数，执行实际的测试

CppUnit 基本框架：

测试执行器




CppUnit 框架结果判断

主要依靠断言进行检查，判断测试用例执行是否成功；

如：CPPUNIT_ASSERT_EQUAL(para1,para2)

# 集成测试

## 集成测试的定义和目的

什么是集成测试（组装测试、联合测试等）？

1） 集成测试是在单元测试的基础上，将所有函数按照概要设计要求组装成为子系统或系统所进行的测试。

2） 集成测试和单元测试所关注的范围是不同的，因此，它们在发现问题的集合上包含有不

相交的区域，不能使用集成测试来代替单元测试，反之也是一样。

集成测试的目的：
集成测试的目的是确保各组件组合在一起后能够按既定意图协作运行，并确保增量的行为正确。属于灰盒测试。
a) 验证接口是否与设计相符合；
b) 发现设计和需求中存在的错误。

# 集成测试关注的重点

1） 单元间的接口（类型：函数接口、文件接口、消息接口、数据库接口和全局变量）
   a) 在把各个模块连接起来的时候，穿越模块接口的数据是否会丢失；
   b) 全局数据结构是否有问题，会不会被异常修改。
2） 基础后的功能
   a) 各个子功能组合起来，能否达到预期要求的父功能
   b) 一个模块的功能是否会对另一个模块的功能产生不利的影响；
   c) 单个模块的误差积累起来，是否会放大，从而达到不可接受的程度。

# 集成测试的层次

三个级别：
a) 模块内集成测试；
b) 子系统内集成测试；
c) 子系统间集成测试。

# 集成测试的策略

1) 大爆炸集成方式（Big Bang）
   首先对每个模块分别进行单元测试，然后再把所有单元组装在一起进行测试，最终得到要求的软件系统。
   a) 优点
      1） 大爆炸集成可以迅速完成集成测试，并且只要极少数的驱动和桩模块设计。它需要的测试用例也是最少的；
      2） 该方法比较简单；
      3） 多个测试人员可以并行工作，对人力，物力资源利用率较高
   b) 缺点
      1） 这种一次性组装方式试图在辅助模块的协助下，在模块单元测试的基础上，将所测模块连接起来进行测试。但是由于程序中不可避免地存在模块间接口、全局数据结构等方面的问题，所以一次试运行成功的可能性并不很大；
      2） 在发现错误的时候，其问题定位和修改都比较困难；
      3） 即使被测系统能够被一次性集成，但还是会有许多接口错误很容易的躲过测试而进入到系统测试范围内。

c) 适用范围

1） 一个维护型项目（或者能增强型项目），其以前的产品已经很稳定，并且新增的项目只有少数几个组件被增加或修改；

2） 被测系统比较小，并且它 的每个函数都经过了充分的单元测试。

2) 自顶向下集成方式（Top-Down）

首先集中于顶层的组件，然后逐步测试处于底层的组件；广度优先策略和深度优先策略

a) 优点

1） 自顶向下的集成方式在测试过程中较早地验证了主要的控制和判断点

2） 如果选用按深度方向组装的方式，可以首先实现和验证一个完整的软件功能

3） 功能可行性较早得到证实，还能够给开发者和用户带来成功的信心

4） 最多只需一个驱动，减少了驱动器的开发的费用

5） 支持故障隔离，定位比较容易

6） 控制结构得到较早验证。

b) 缺点

1） 桩的开发和维护是本策略的最大成本；

2） 底层组件行为的验证被推迟了；

3） 随着底层组件的不断增加，整个系统越来越复杂，导致底层组件的测试不充分，尤其是那些被重用的组件。

c) 适用范围

1） 产品控制结构比较清晰和稳定；

2） 产品的高层接口变化比较小；

3） 产品的底层接口未定义或经常可能被修改；

4） 产品的控制组件具有较大的技术风险，需要尽早被验证；

5） 希望尽早能够看到产品的系统功能行为。

3) 自底向上集成测试策略（Bottom-up）

从程序结构的最底层的组件开始组装和测试。对于一个给定层次的组件，它的子组件（包括子组件的所有下属组件）已经组装并测试完成，所以不再需要桩。

a) 优点

1） 允许对底层组件行为的早期验证。可以在任何一个叶子节点已经就绪的情况下进行集成测试；

2） 在工作的最初可能会并行进行集成，在这一点上比使用自顶向下的策略效率高；

3） 减少了桩的工作量，毕竟在集成测试中，桩的工作量远比驱动的工作量要大得多。但是为了模拟一些中断或异常，可能还是需要设计一定的桩；

4） 也支持故障隔离。

b) 缺点

1） 驱动的开发工作量也是很庞大的；

2） 对高层的验证被推迟到了最后，设计上的错误不能被及时发现，尤其对于那些控制结构在整个体系中非常关键的产品。

c) 适用范围

1） 底层接口比较稳定、变动较少的产品

2） 高层接口变化比较频繁的产品

3） 底层组件较早被完成的产品

4) 三明治集成策略（Sandwich）

把系统划分成三层，中间一层为目标层，测试的时候，对目标层上面的一层使用由顶向下的集成策略，对目标层下面的一层使用自底向上的集成策略，最后测试在目标层会合。

a) 优点

集合了自顶向下和自底向上两种策略的优点

b) 缺点

中间层在被集成前测试不充分

c) 适用范围

大部分软件开发项目都可以使用这种集成策略

5) 基干集成策略（Backbone）主要在嵌入式系统

在很多系统中，尤其在嵌入式系统中，一般可以划分成两个部分：内核部分（基干部分）和外围部分，这两部分经常会被不同的项目组并行开发。

其特点：

1） 内核部分提供了系统最基本的功能和服务；

2） 外围部分以内核为基础，不能脱离内核而独自使用；

3） 内核具有很高的耦合性，并且相当复杂，试图设置桩会是相当困难且成本很高的事情；

4） 外围部分可以分为应用子系统和控制子系统，应用子系统内耦合性不大，而控制子系统内具有较高的耦合性。

a) 基干集成步骤

1) 识别外围的应用子系统部分、控制子系统部分，基干部分；

2) 对基干中所有的组件进行大爆炸集成，形成基干子系统，并使用一个驱动检查经过大爆炸的基干；

3) 对控制子系统进行自顶向下的集成；

4) 对各应用子系统进行自底向上的集成；

5) 对基干子系统，控制子系统和各应用子系统进行大爆炸集成形成整个系统

b) 优点

具有三明治集成的优点，更适合于大型复杂项目的集成

c) 缺点

必须对系统的结构和相互依存性进行仔细的分析；

必须开发驱动和桩，并且由于被测系统的复杂性导致驱动和桩开发工作量的加大；

由于局部采用大爆炸的策略，因此有些接口可能测试不充分。

6) 分层集成策略（通信行业）

7) 基于功能的集成策略

8) 基于消息的集成策略

9) 基于进度的集成策略

10) 基于风险的集成策略

# 集成测试的过程

集成测试的四个阶段：

1） 集成测试计划阶段：完成集成测试计划

2） 集成测试设计阶段：完成集成测试方案

3） 集成测试实现阶段：完成集成测试用例、集成测试规程、集成测试脚本及数据文件

4） 集成测试执行阶段：执行集成测试用例、修改发现的问题并进行回归测试，提交集成测

试报告

# 系统测试

## 系统测试的定义和目的

什么是系统测试？
系统测试是将已经集成好的软件系统，作为整个基于计算机系统的一个元素，与计算机硬件、外设、某些支持软件、数据和人员等其他系统元素结合在一起，在实际运行（使用）环境下，对计算机系统进行一系列的测试活动。（系统可能是纯软件，不一定包含硬件）。
系统测试的目的

a) 通过与系统的需求定义做比较，发现软件与系统定义不符合或与之矛盾的地方。
b) 系统测试的测试用例应根据需求分析说明书来设计，并在实际使用环境下运行。

## 系统测试的对象

系统测试的对象是软硬件集合在一起的系统，不应该是独立的软件与硬件环境。当然具体操作、执行时可以根据实际情况来组织。
验证时尽可能模拟实际的运行环境与条件。

## 系统测试的类型

1） 功能测试
   a) 概念
      功能测试是根据产品的需求规格说明书和测试需求列表，验证产品的功能实现是否符合产品的需求规格
   b) 目标
      功能测试主要是为了发现以下几类错误：
      1.是否有不正确或遗漏的功能？
      2.功能实现是否满足用户需求和系统设计的隐藏需求？
      3.输入能否正确接受？能否正确输入结果？
2） 性能测试（Performance Testing）
   性能测试就是用来测试软件在集成系统中的运行性能的；性能测试的目标是度量系统相对于预定义目标的差距；性能测试必须要有工具支持，有一些专门用于 GUI 和 Web 的性能测试工具，如 Loadrunner，SilkPerformer，WebLoad 等。
   a) 压力测试：在一定的软硬件及网络环境下，模拟大量的用户，处理大量的数据业务，使得系统长时间在极限状态下运行，目的在于寻找系统的失效点。
   b) 负载测试：在一定的软硬件及网络环境下，模拟不同数量的用户运行一种或多种业务，查看系统的性能表现。
   c) 容量测试：根据数据库的存储空间，向数据库构造不同数量级的数据，分别执行一种或多种业务，查看 DB Server 的性能表现。

性能测试收集的信息

A， CPU 使用情况

B，IO 使用情况

C，内存使用情况

D， 信道使用情况

E， 每个模块执行时间百分比

F， 一个模块等待 IO 完工的百分比

G， 指令随时间的跟踪路径

H， 每一组指令页换入和换出的次数

I， 系统反应时间

J， 系统吞吐量，即每个时间单元的处理数量

K， 所有主要指令的单元执行时间

3） 压力测试（Stress Testing）

目的：调查系统在其资源超负荷的情况下的表现。尤其感兴趣的是这些对系统的处理时间有什么影响。这类测试在一种需要反常数量、频率或资源的方式下执行系统。

目标：通过极限测试方法，发现系统在极限或恶劣环境中自我保护能力。主要验证系统的可靠性。

例子：成千上万的用户在同一时间登陆到 Internet；同时引入大量操作。

4） 容量测试（Volume Testing）

目的：使系统承受超额的数据容量来发现它是否能够正确处理。容量测试是面向数据的，并且它的目的是显示系统可以处理目标内确定的数据容量。

例子：使用编译器编译一个极其庞大的源程序；

一个操作系统的任务队列被充满；

庞大的 Email 信息和文件充满了 Internet。

5）

测试：1 测试经理

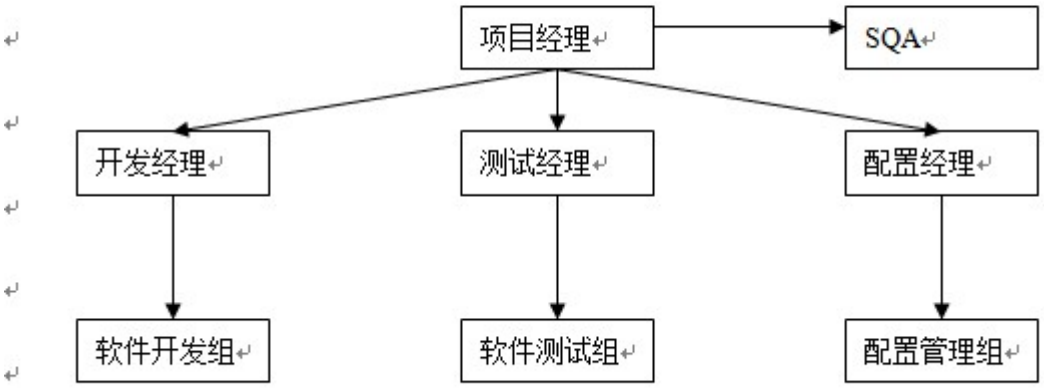
2 业务专家

3 测试工程师

流程 技术 组织、

## 软件研发组织和流程

常见项目组架构




## 软件生命周期

计划 → 需求分析 → 设计 → 编码 → 测试 → 运行和维护