软件测试

授课教师：张剑波

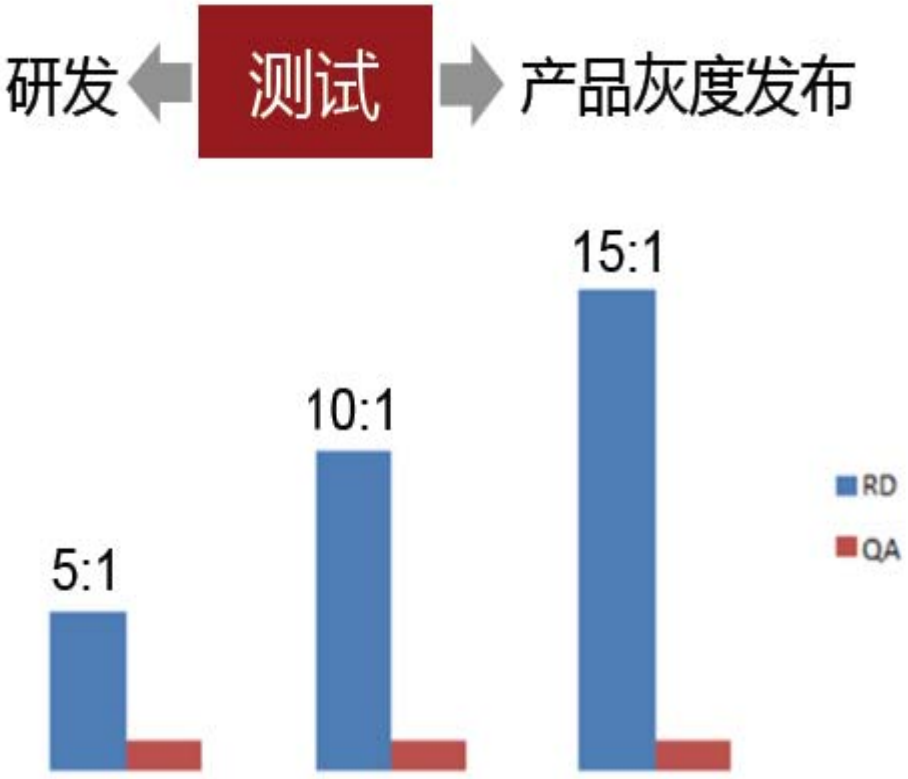
授课班级：111181-3班

2021年2月

Chapter1 软件测试概述

PM、RD(FE)、QA、OP

互联网公司：传统测试职能的价值被削弱

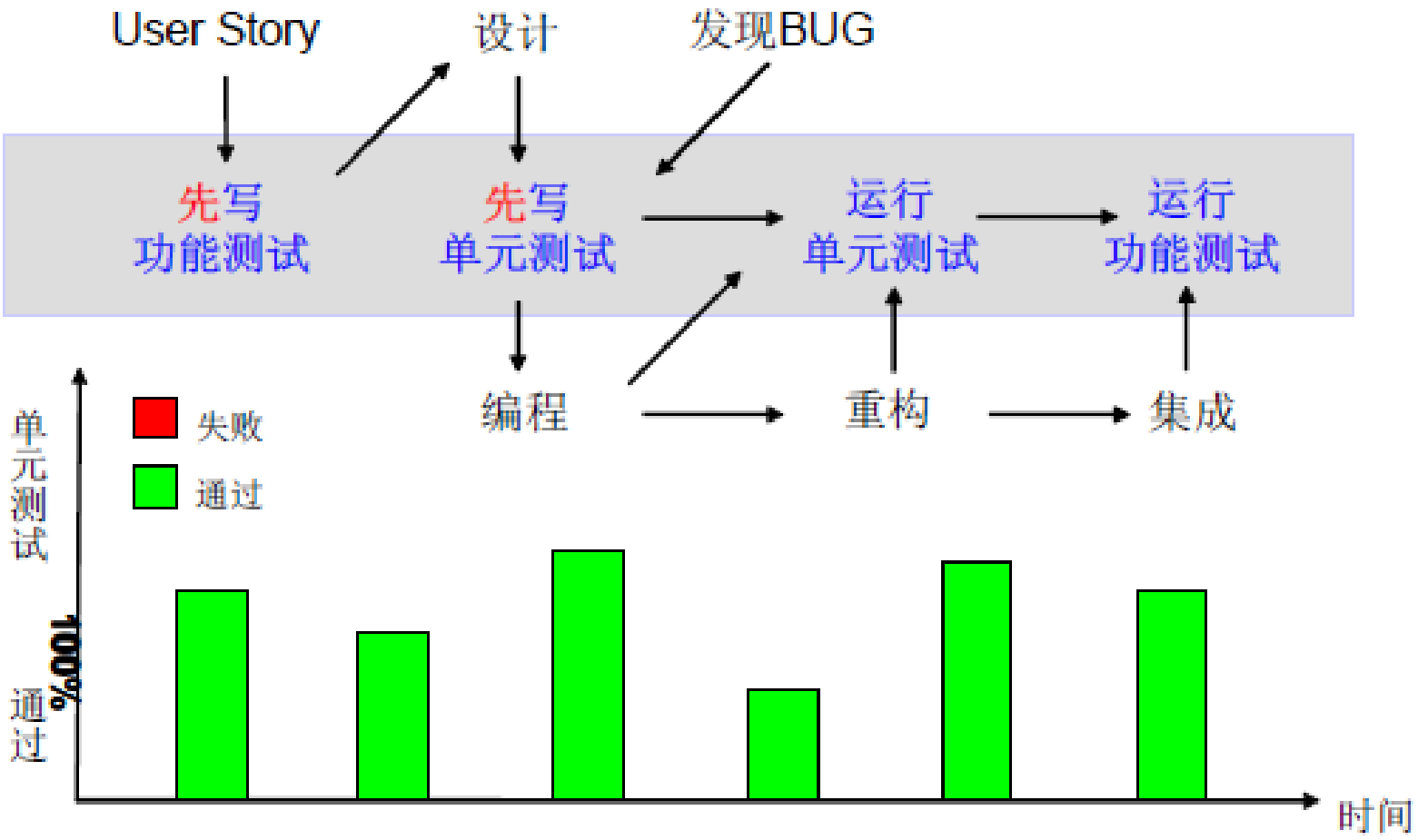


开发和测试理念的转变

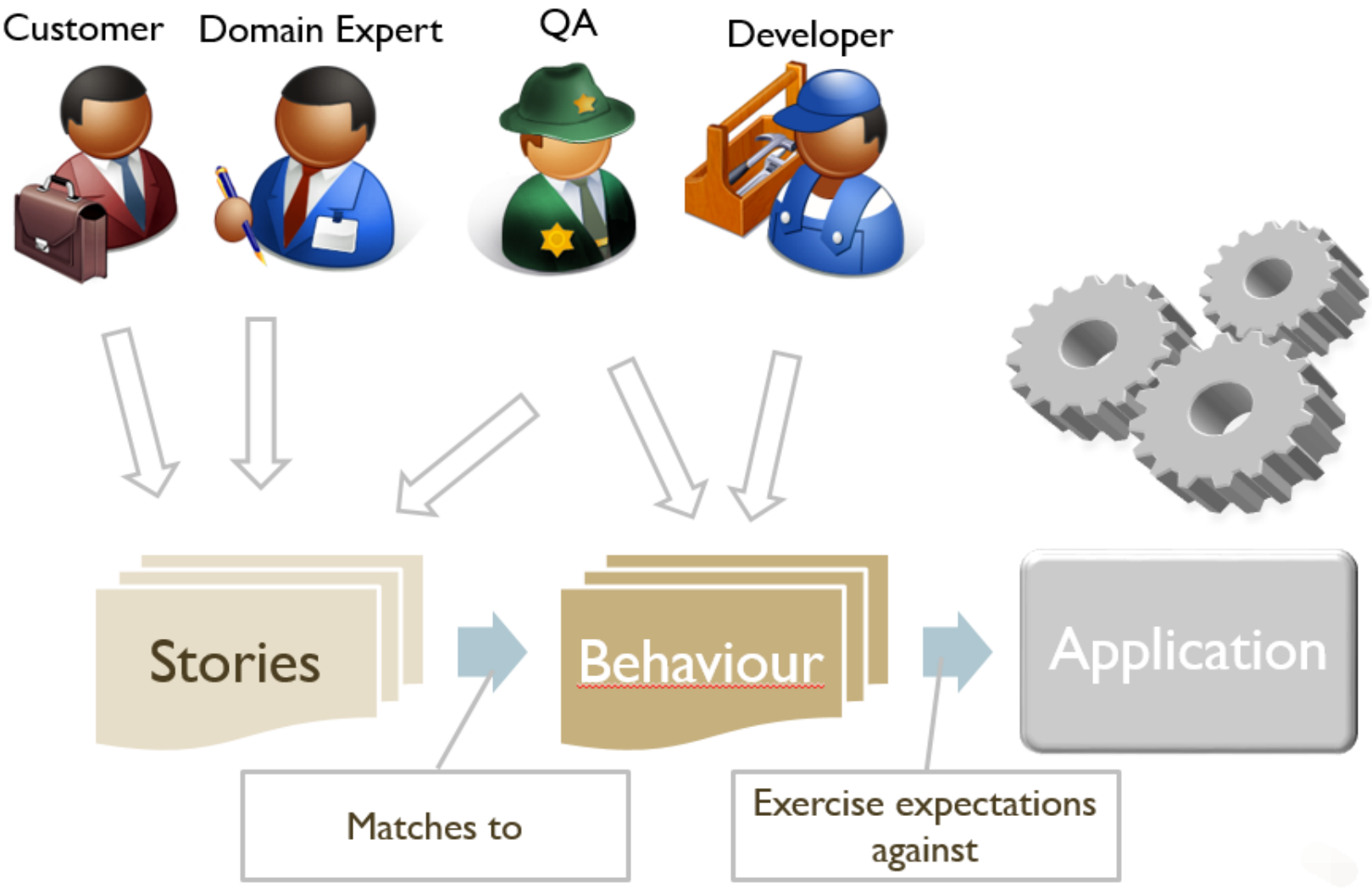
如何“快”成为所有互联网公司的关注点！

技术	测试前置/Unit Test/Code Review/Static Analysis 成为重点技术
文化	Who write code, who own quality 成为开发理念，代码质量成为面子问题，即互联网研发文化
方法	持续集成是标配，持续发布，敏捷迭代

测试驱动开发—TDD

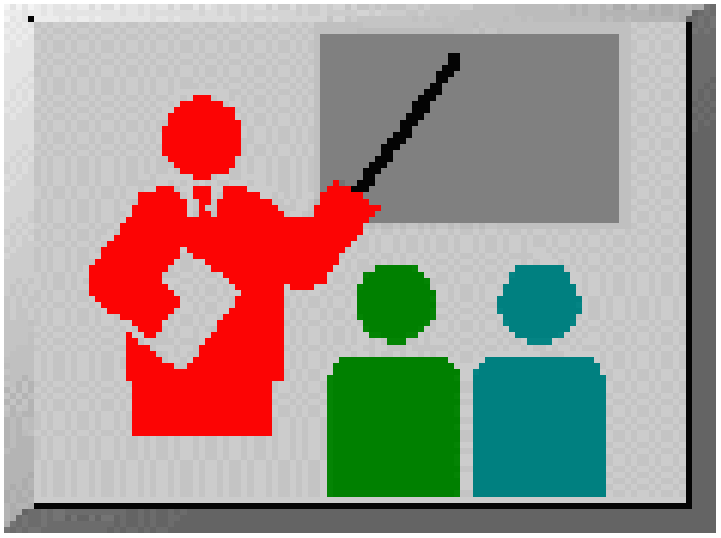

User Story
设计
发现BUG
先写
功能测试
先写
单元测试
运行
单元测试
运行
功能测试
编程
重构
集成
失败
通过
单元测试
通过
100%
时间

行为驱动开发—BDD

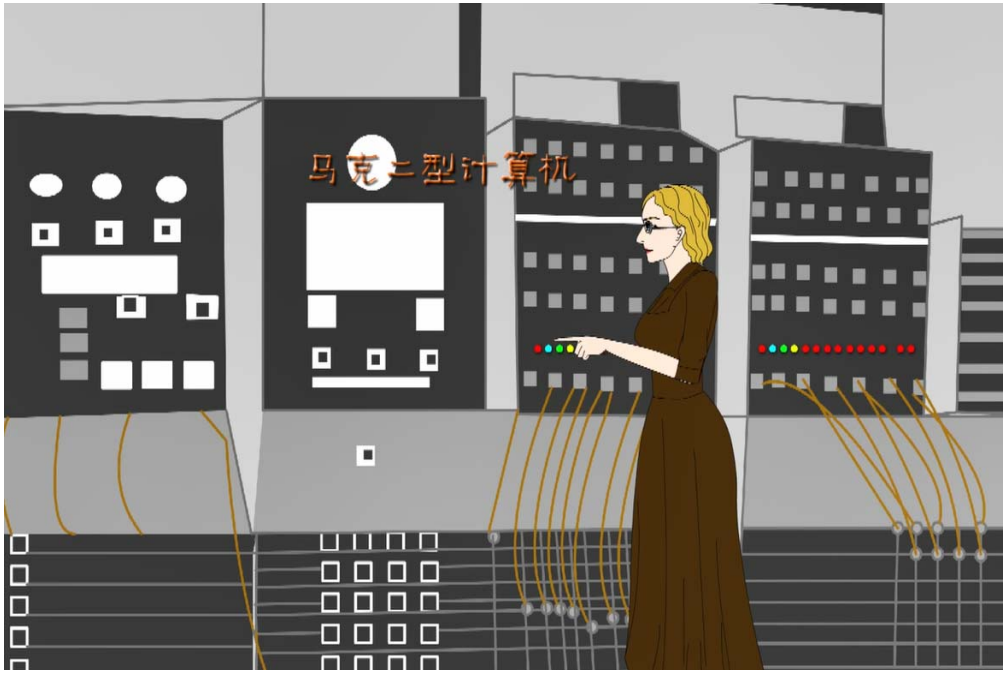

Customer
Domain Expert
QA
Developer
Stories
Behaviour
Application
Matches to
Exercise expectations against

内容提要

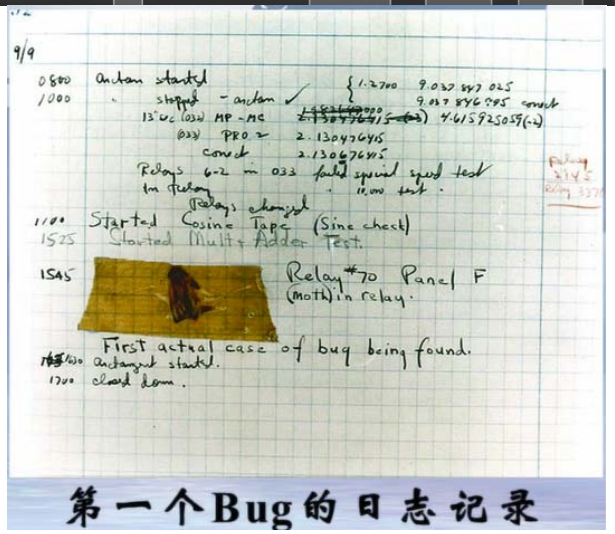
- 软件测试的目的
- 软件测试的定义
- 软件测试与软件开发
- 软件测试发展



Bug的由来



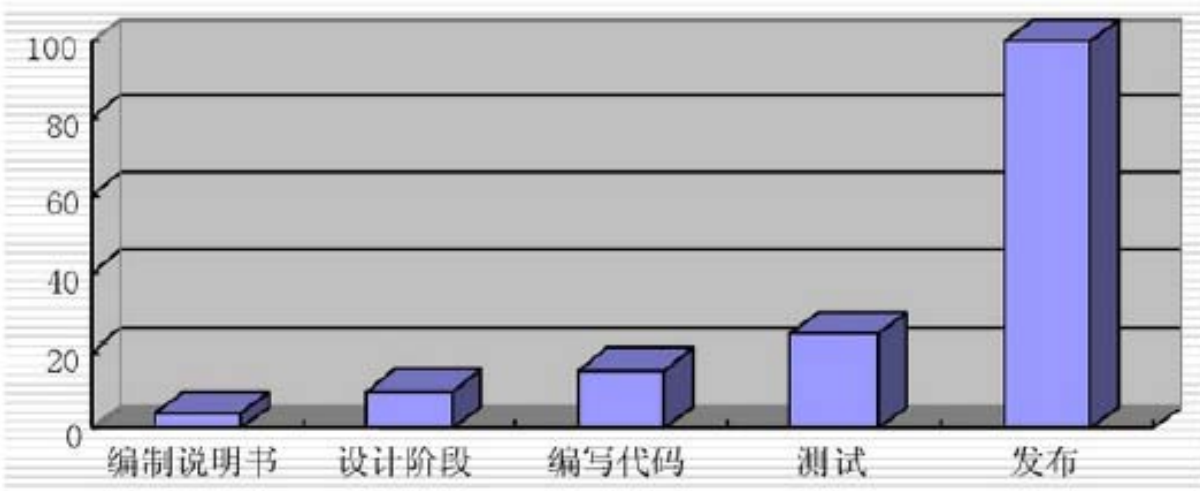
- 格蕾丝·霍珀（Grace Hopper）



第一个Bug的日志记录

一、软件测试的目的

- 基于不同的立场，存在着两种完全不同的测试目的
  - 从用户的角度出发，希望通过软件测试暴露软件中隐藏的错误和缺陷，以考虑是否可接受该产品。
  - 从软件开发者的角度出发，希望测试成为表明软件产品中不存在错误的过程，验证该软件已正确地实现了用户的要求，确立人们对软件质量的信心。



软件测试不充分的后果

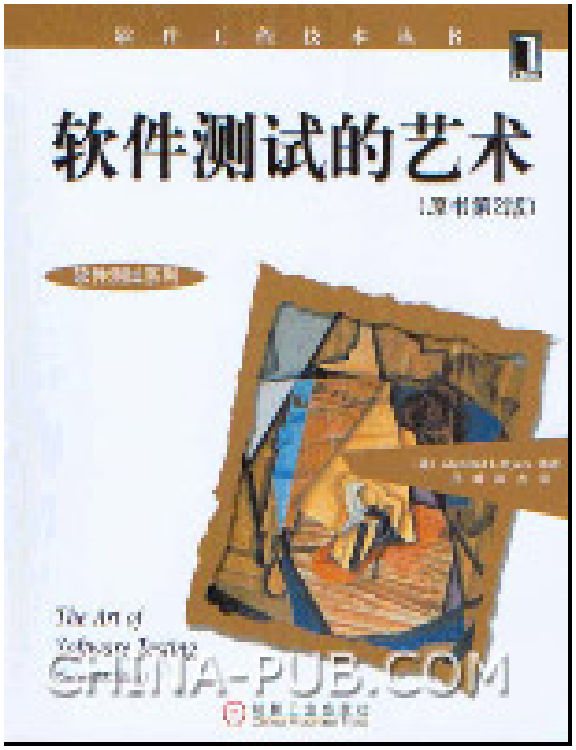
- 2020年2月28日，波音公司承认，该公司测试载人飞船星际客机软件系统的程序存在严重缺陷，现在计划对测试程序进行修改；
- “我们不知道我们有多少软件故障，我们到底只有两个故障还是几百个。”
- 星际客机的计时问题
- 服务舱和乘员舱分离过程
  - 使用“模拟器”有错误的推进器配置
- 测试的问题“绝对不是成本问题”
  - 未来，波音公司将继续以更小的模块测试系统

软件测试不充分的后果（续）

- 2012年11月，美国福特公司召回8.9万辆汽车，原因是汽车冷却系统的监控软件存在故障，会导致发动机在运行时产生过热现象，并已至少产生了9起由此引发的汽车失火事故；
- 2011年7月23日，“甬温线”动车追尾事故造成40人死亡、172人受伤。事故的主要原因之一是列控中心设备中的自检模块软件存在严重设计缺陷，未将系统导向安全侧；
- 2007年10月16日，德国弗鲁蒂根（Frutigen）地区的勒奇山基底隧道（Lötschberg-Basistunnel）附近列车脱轨。该事故是由于ETCS-2级列控系统无线闭塞中心（RBC）接入时与一个列车移动授权延伸的相关软件错误引起的。
- 2004年9月，由于空管软件中的时钟管理缺陷，美国洛杉矶机场400余架飞机与机场指挥一度失去联系，给几万名旅客的生命安全造成严重威胁；

Myers软件测试目的

- **Grenford J. Myers在《The Art of Software Testing》一书中的观点:**
  - (1)软件测试是为了发现错误而执行程序的过程;
  - (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。



- 换言之，测试的目的是
  - 想以最少的时间和人力，系统地找出软件中潜在的各种错误和缺陷。如果我们成功地实施了测试，我们就能够发现软件中的错误。
  - 测试的附带收获是，它能够证明软件的功能和性能与需求说明相符合。
  - 实施测试收集到的测试结果数据为可靠性分析提供了依据。
  - 测试不能表明软件中不存在错误，它只能说明软件中存在错误。

二、软件测试的定义

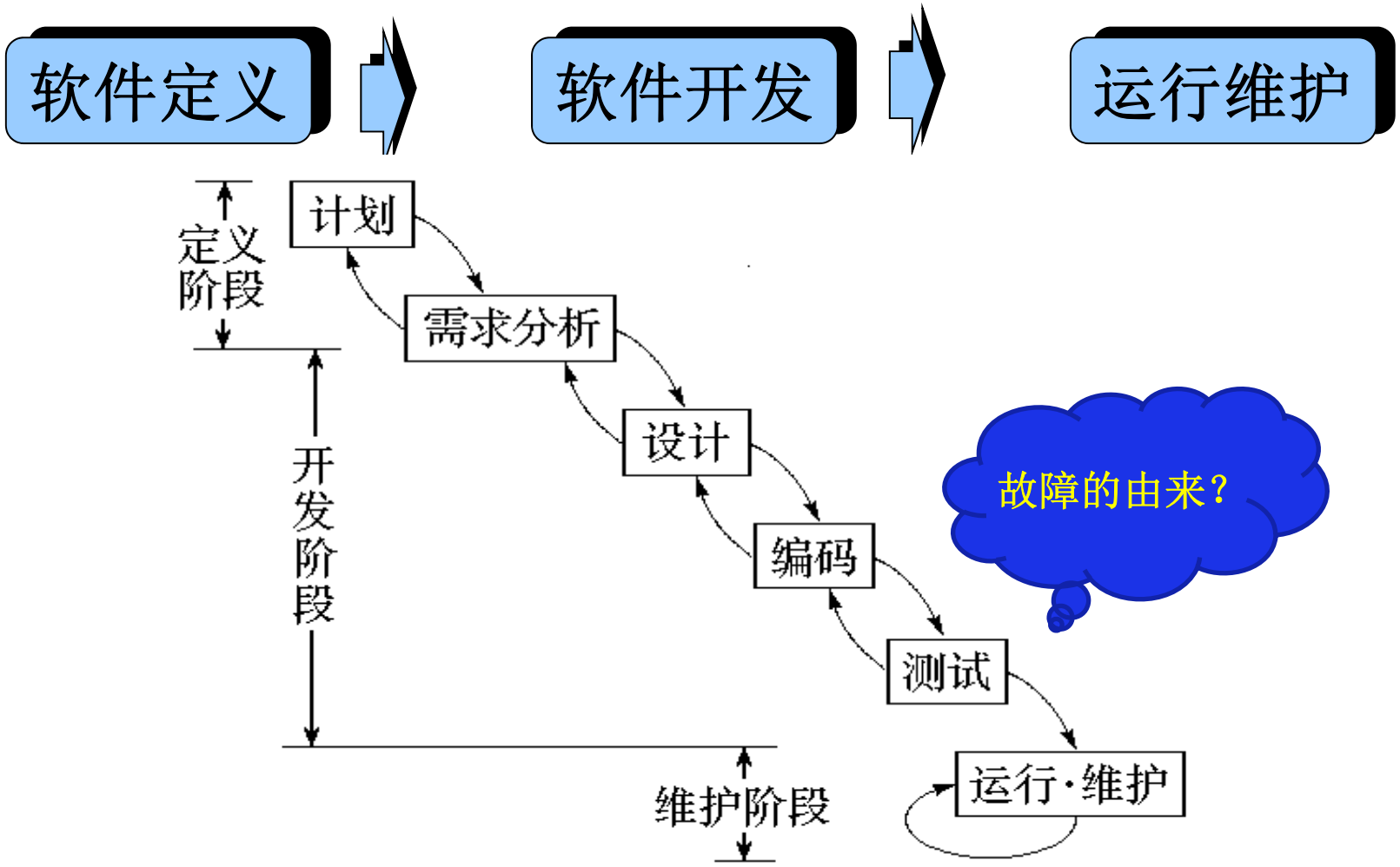
- IEEE（1983）：使用人工或自动手段运行或测定某个系统的过程，其目的在于检验它是否满足规定的需求或是弄清预期结果与实际结果之间的差别。
- Myers：是为了发现错误而执行程序的过程。
- 通过经济、高效的方法，捕捉软件中的错误，从而达到保证软件内在质量的目的。

软件测试的作用

- 测试是执行一个系统或者程序的操作；
- 测试是带着发现问题和错误的意图来分析和执行程序；
- 测试结果可以检验程序的功能和质量；
- 测试可以评估软件项目产品是否获得预期目标以及是否能被客户接受；
- 测试不仅包括执行代码，还包括对需求等编码以外东西的测试。

三、软件测试与软件开发的关系

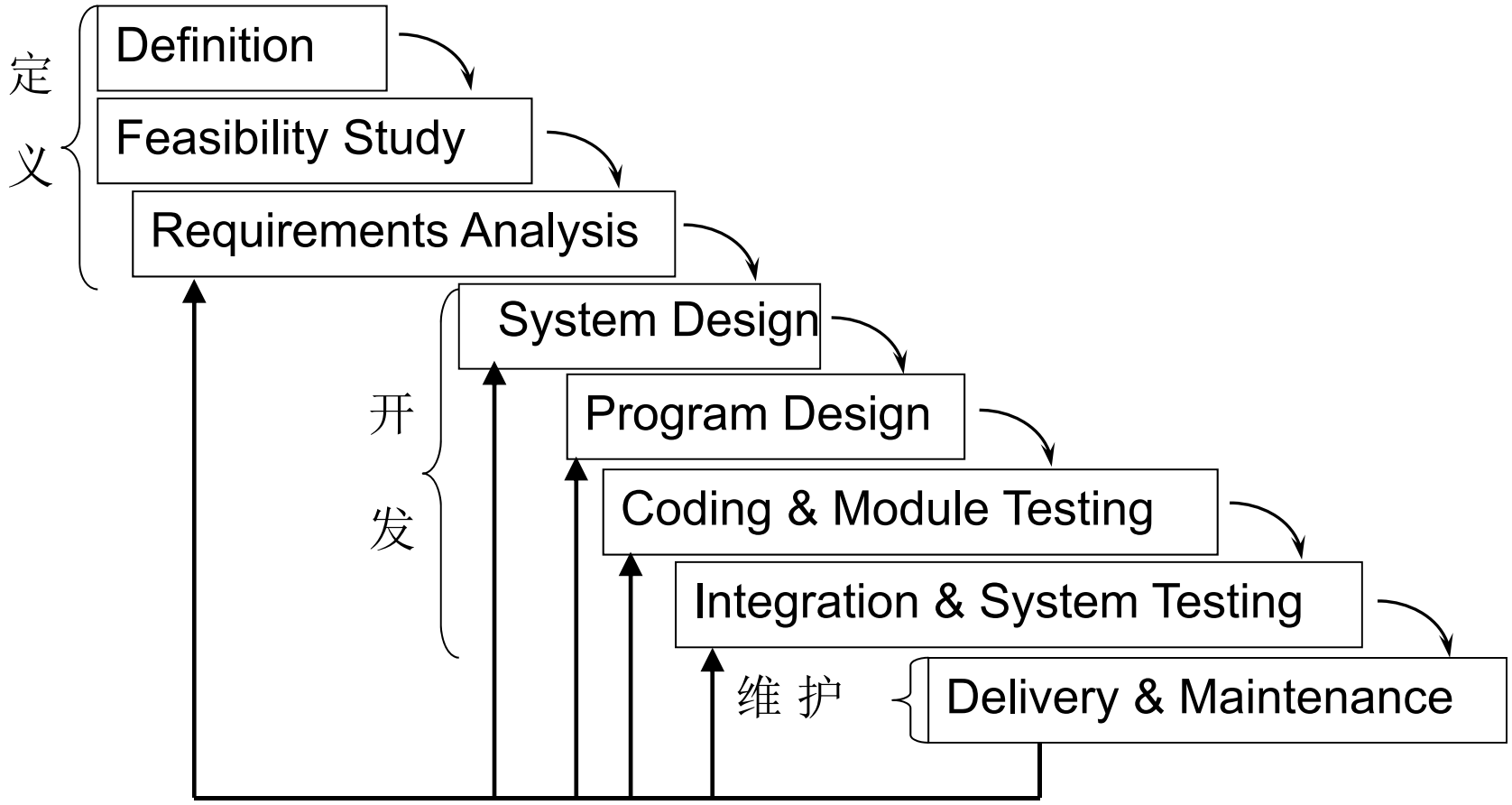
- 软件生命周期

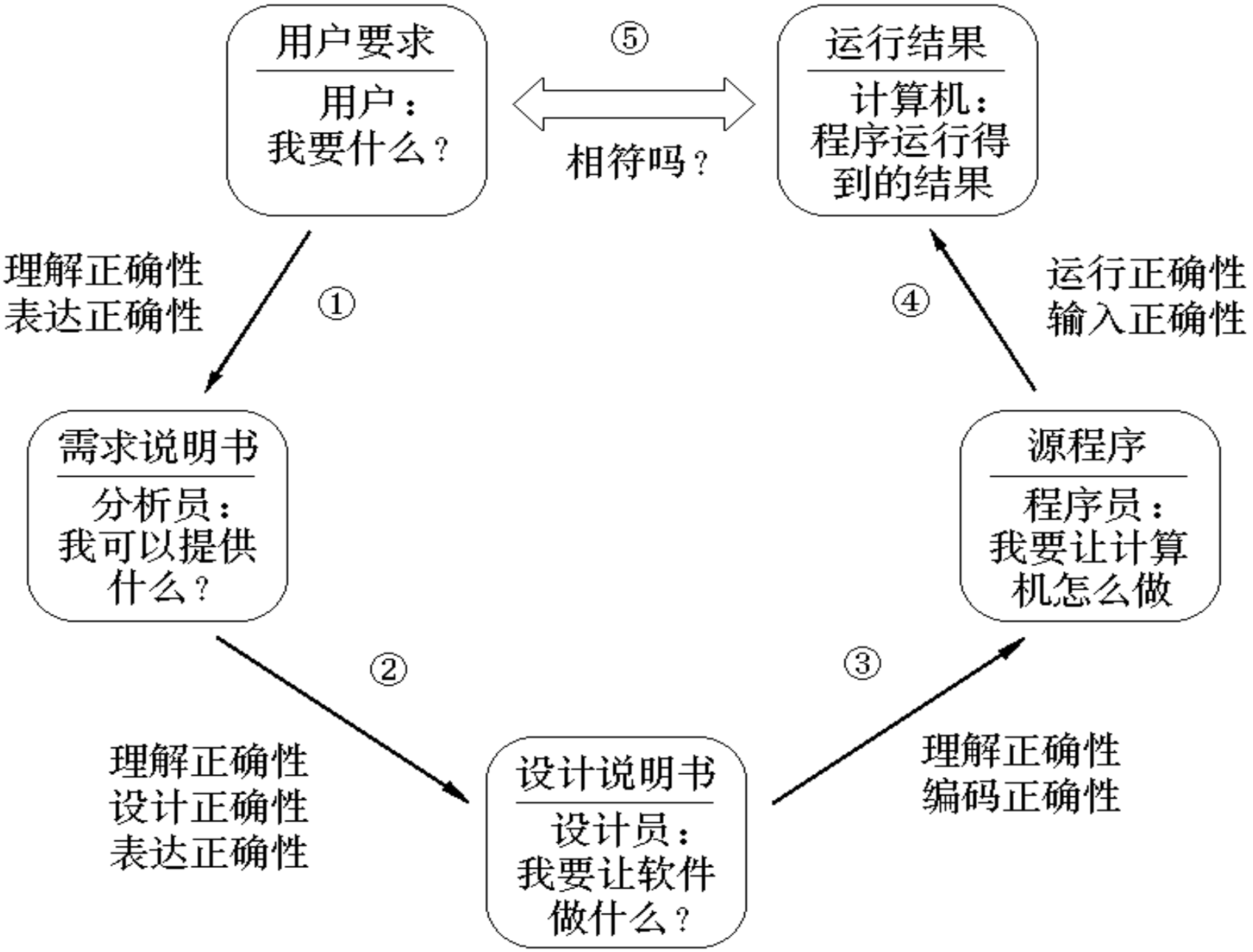


软件测试的时间

- 表现在程序中的故障并不一定是编码所引起的，产生故障的根源可能在开发前期的各个阶段（需求分析阶段、概要设计阶段、详细设计阶段），因此，解决问题、排除故障也必须追溯到前期的工作。
- 因此，软件测试应贯穿于软件定义与开发的整个期间。

以 瀑布模型 为例

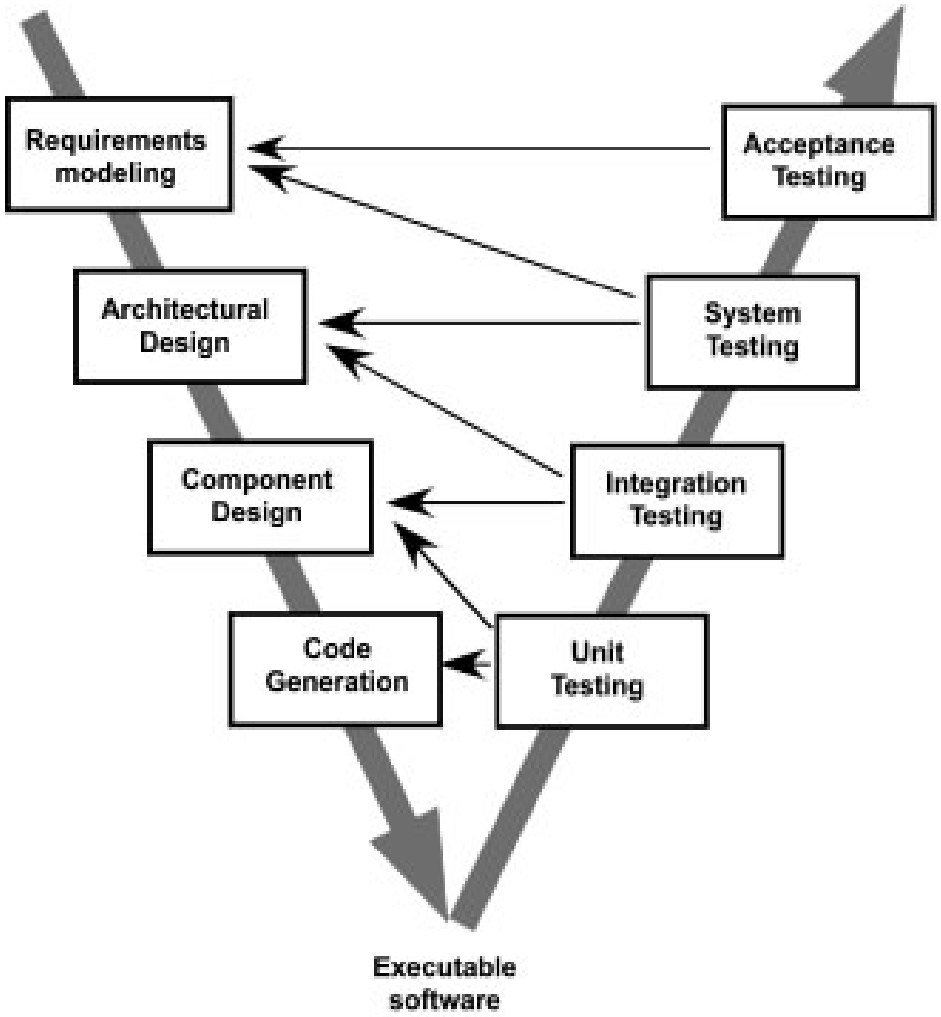

定义
Definition
Feasibility Study
Requirements Analysis
开发
System Design
Program Design
Coding & Module Testing
Integration & System Testing
维护
Delivery & Maintenance


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

软件测试在各阶段的工作

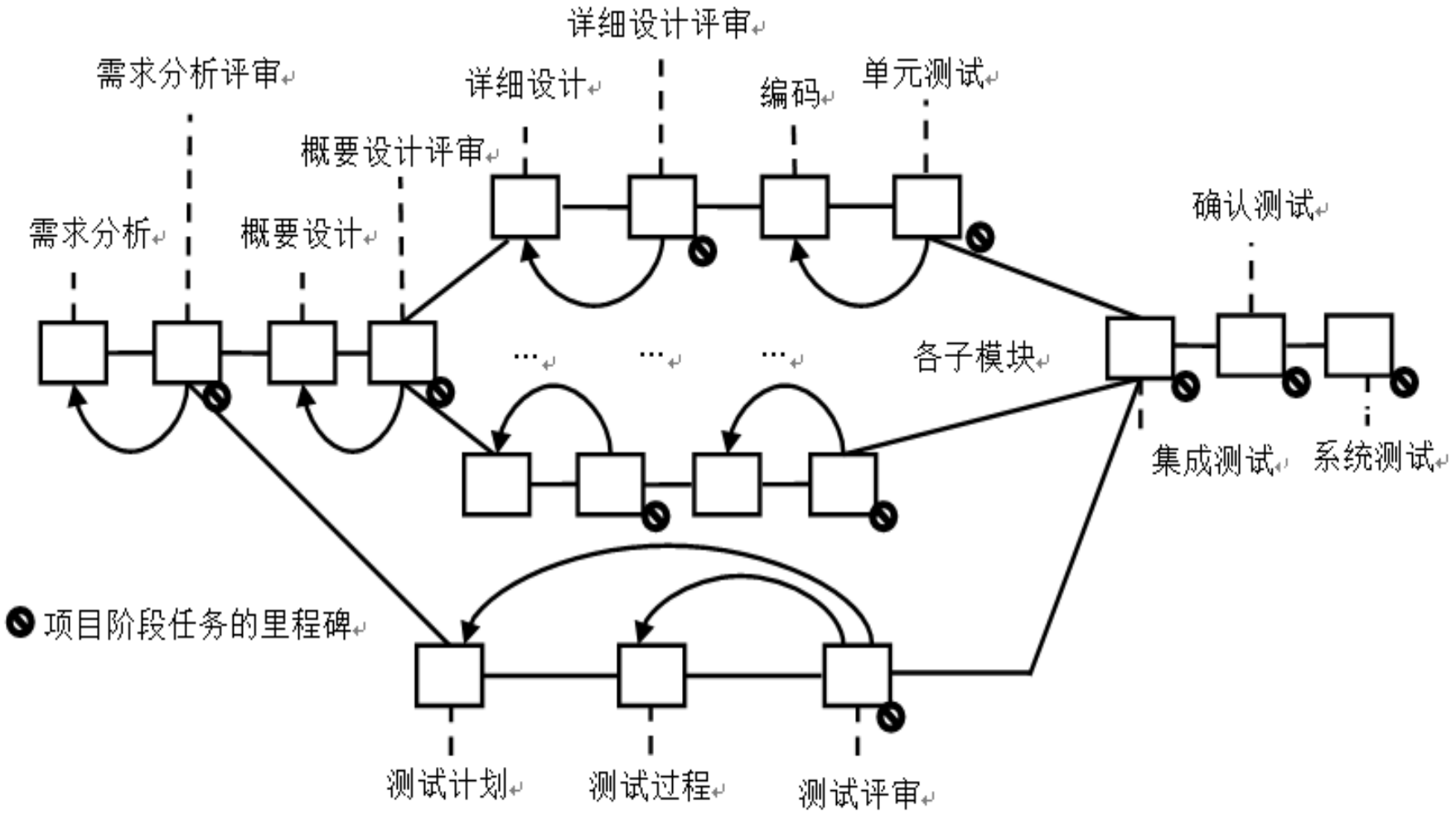
- 项目规划阶段：负责整个测试阶段的监控。
- 需求分析阶段：确定测试需求分析，制定系统测试计划。测试需求分析是指分析产品生存周期中测试所需的资源、配置、各阶段评审通过的标准等。
- 概要设计和详细设计阶段：制定集成测试计划和单元测试计划。
- 程序编写阶段：开发相应的测试代码或测试脚本。
- 测试阶段：实施测试，并提交相应的测试报告。

对瀑布模型的改进：V模型

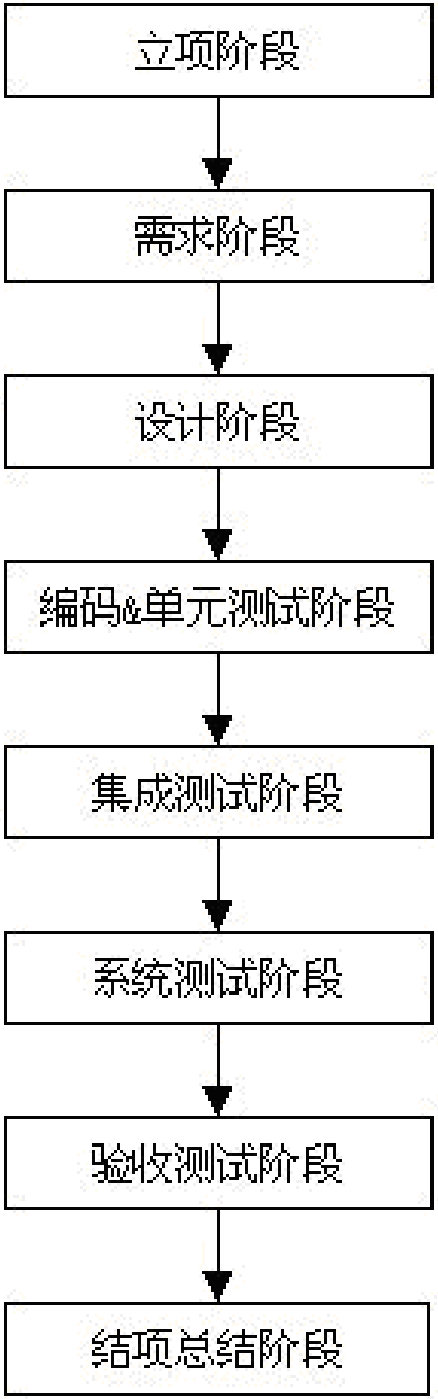


- 瀑布模型的变体
- 特点：将验证确认动作用于早期的工作中
- 沿左侧向下推进
  - 基本问题逐步细化
  - 形成解决方案的技术描述
- 沿右侧向上推进
  - 一系列测试（质量保证）

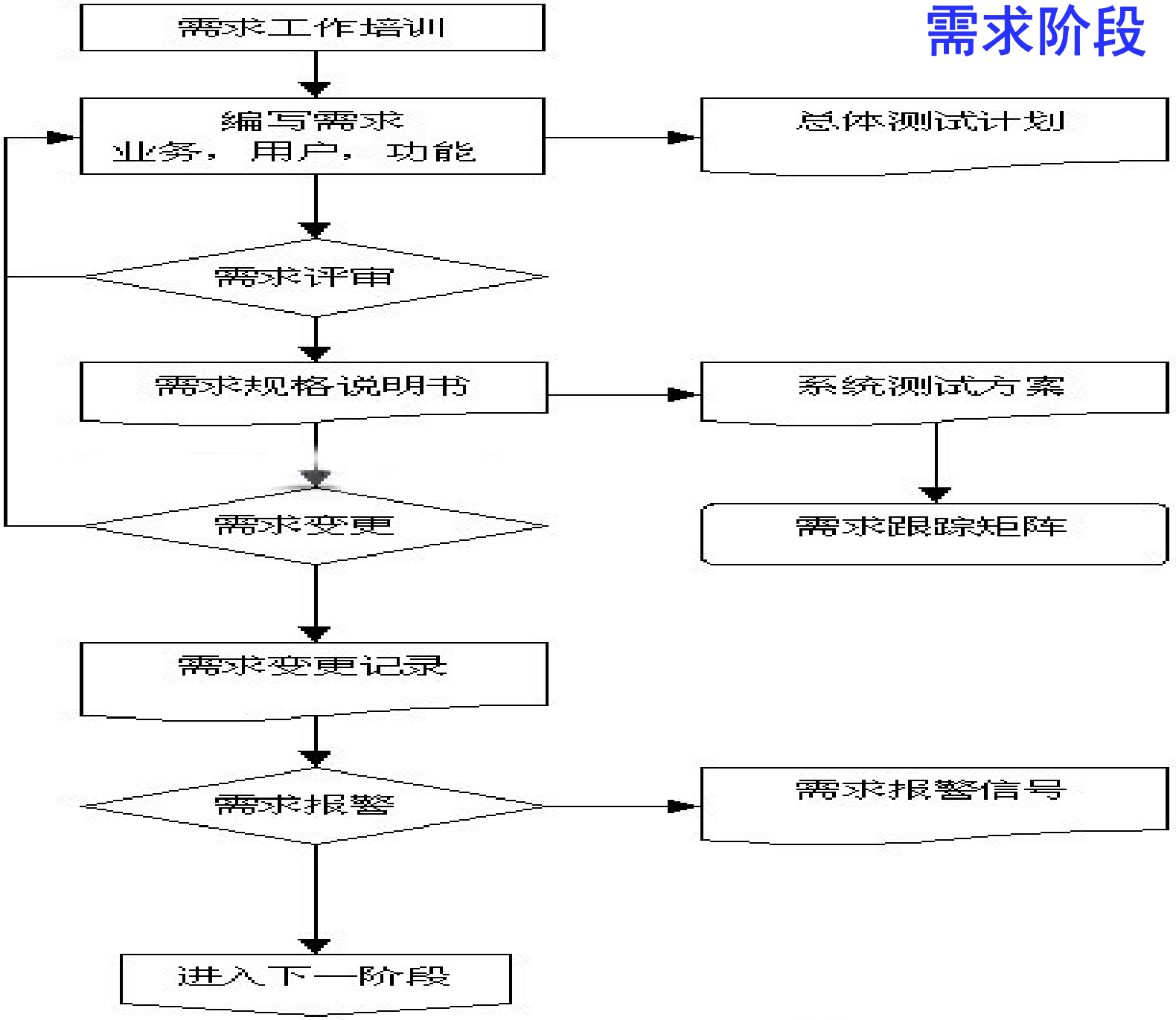
1. 软件测试与软件开发


详细设计评审
需求分析评审
详细设计
编码
单元测试
概要设计评审
确认测试
需求分析
概要设计
各子模块
集成测试
系统测试
项目阶段任务的里程碑
测试计划
测试过程
测试评审

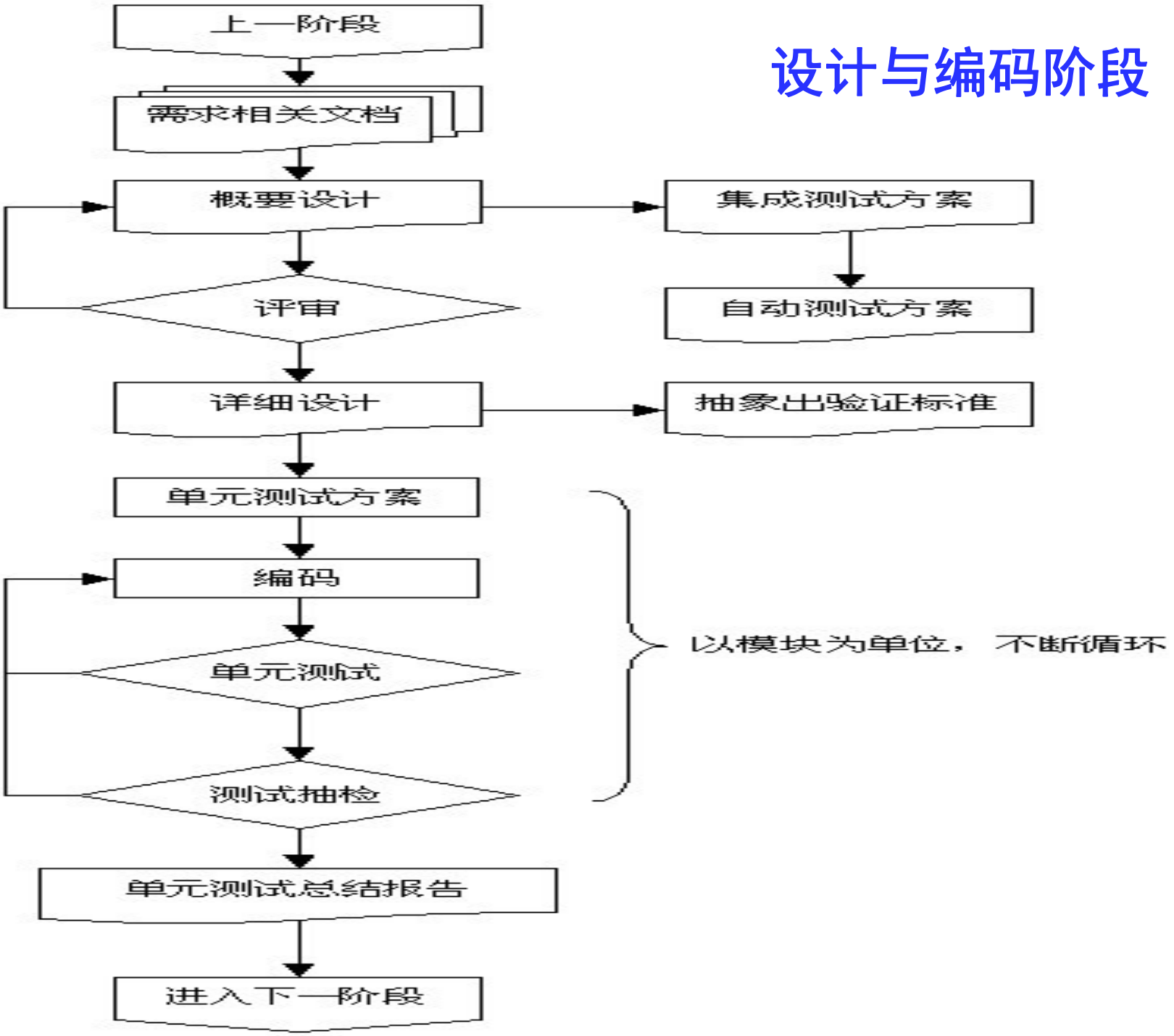
软件测试工作总体流程



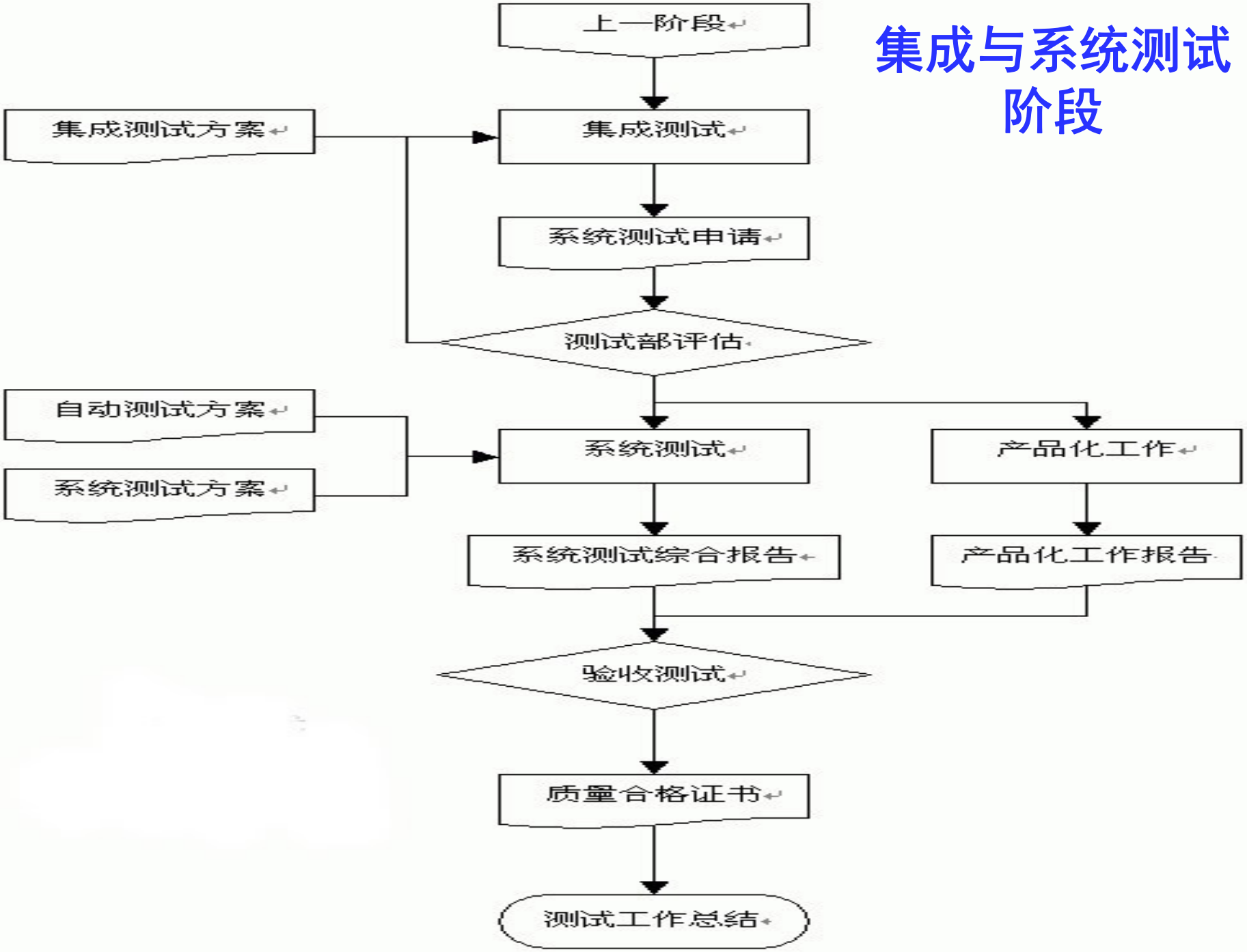
需求阶段


需求工作培训
编写需求
业务，用户，功能
总体测试计划
需求评审
需求规格说明书
系统测试方案
需求变更
需求跟踪矩阵
需求变更记录
需求报警
需求报警信号
进入下一阶段

设计与编码阶段


上一阶段
需求相关文档
概要设计
集成测试方案
评审
自动测试方案
详细设计
抽象出验证标准
单元测试方案
编码
单元测试
测试抽检
以模块为单位，不断循环
单元测试总结报告
进入下一阶段

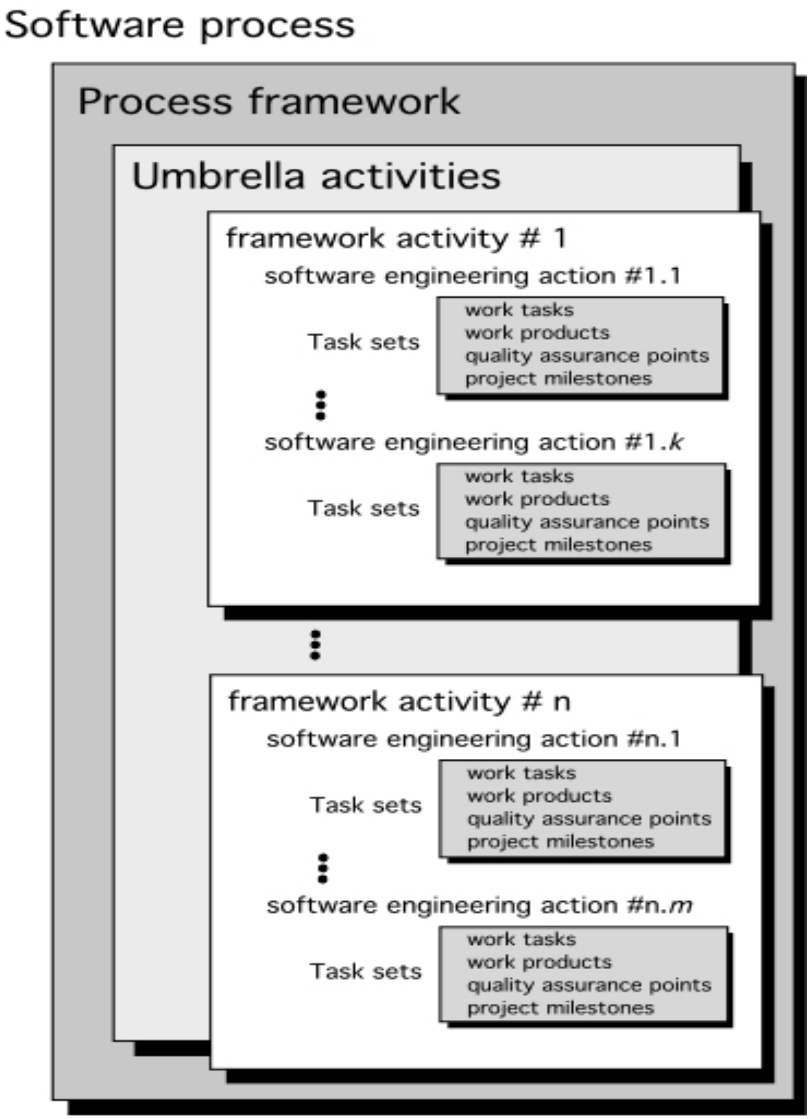
集成与系统测试阶段


上一阶段
集成测试方案
集成测试
系统测试申请
测试部评估
自动测试方案
系统测试方案
系统测试
产品化工作
系统测试综合报告
产品化工作报告
验收测试
质量合格证书
测试工作总结

2. 软件开发过程模型

- 通用过程模型
- 惯用过程模型
- 专用过程模型

（1）通用过程模型



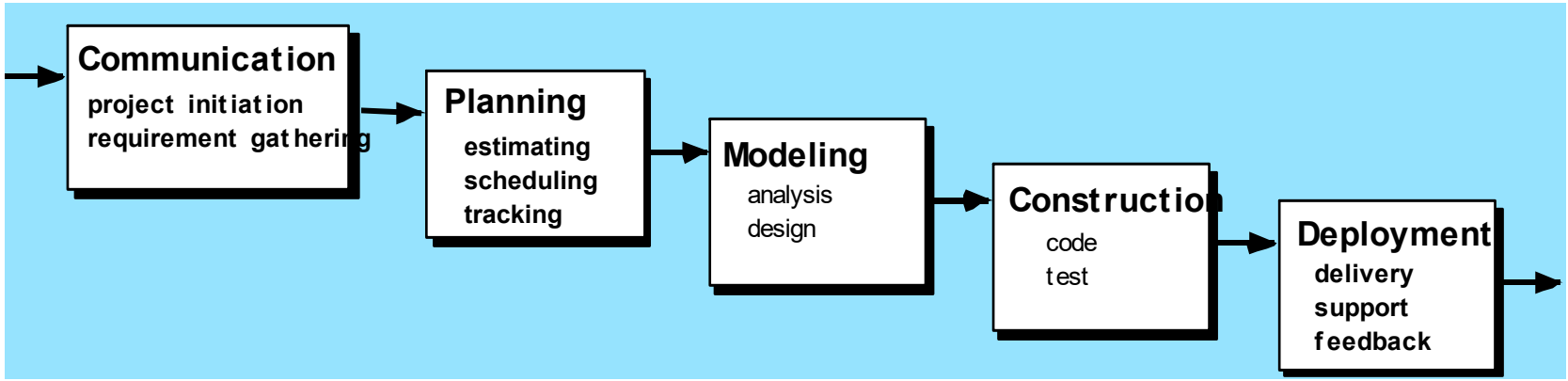
- 框架活动：沟通、策划、建模、构建和部署
  - 软件工程动作
    - 任务集合
      - ✓ 工作任务
      - ✓ 工作产品
      - ✓ 质量保证点
      - ✓ 项目里程碑

（2）惯用过程模型

- 传统过程模型：引入结构化设计，使软件开发有序
- 传统：规定了一套过程元素，包括框架活动、软件过程动作、任务、工作产品、质量保证及每个项目的变更控制机制；还定义了过程流（工作流）
- 代表过程模型
  - 瀑布模型
  - 增量过程模型
  - 演化过程模型
  - 协同模型

瀑布模型

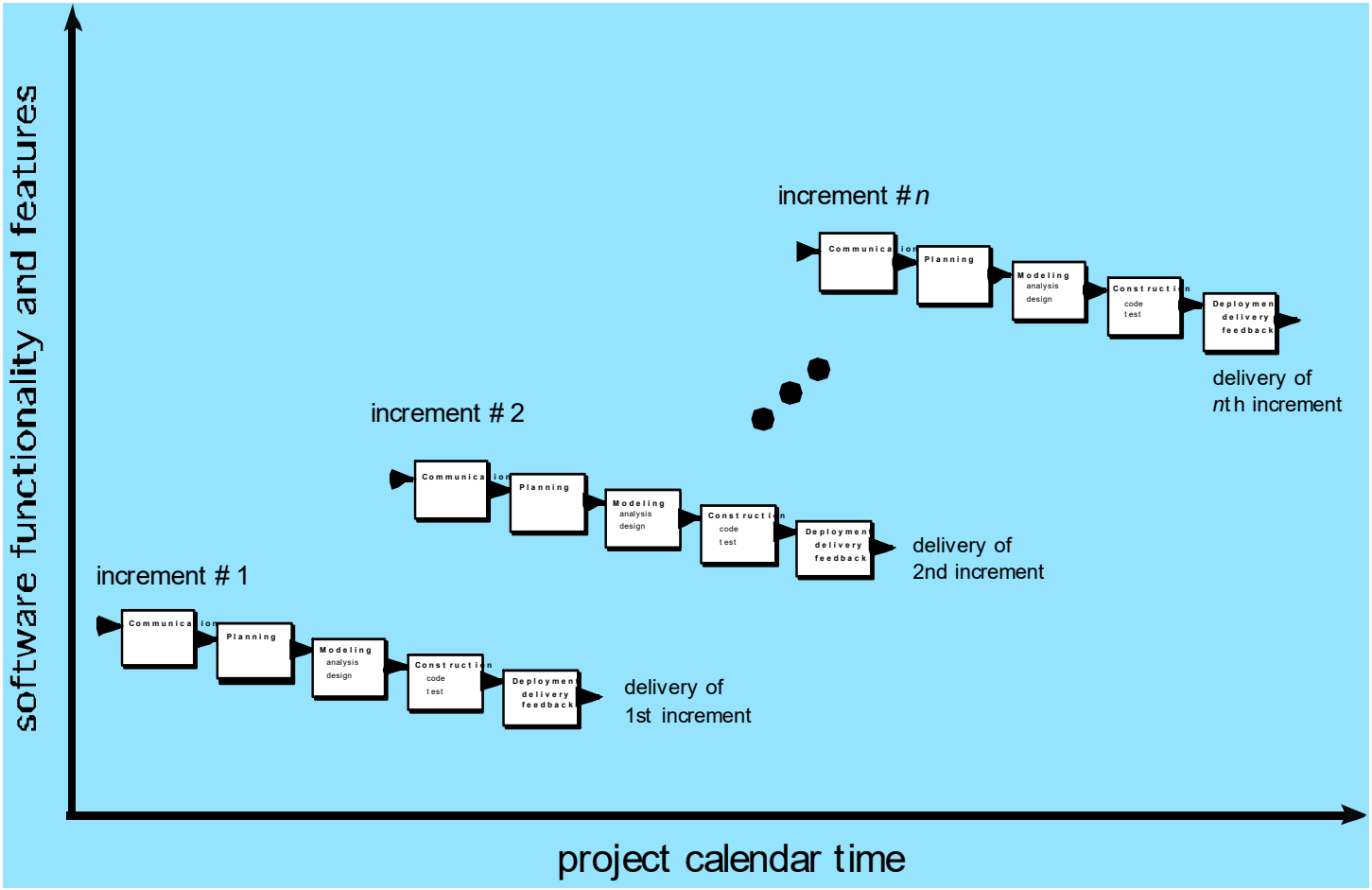
- 线性工作流：从沟通到部署



- 适用情况：
  - 对已有系统进行明确定义的适应性调整或增加功能
  - 对新系统，需求准确定义和相对稳定

增量过程模型

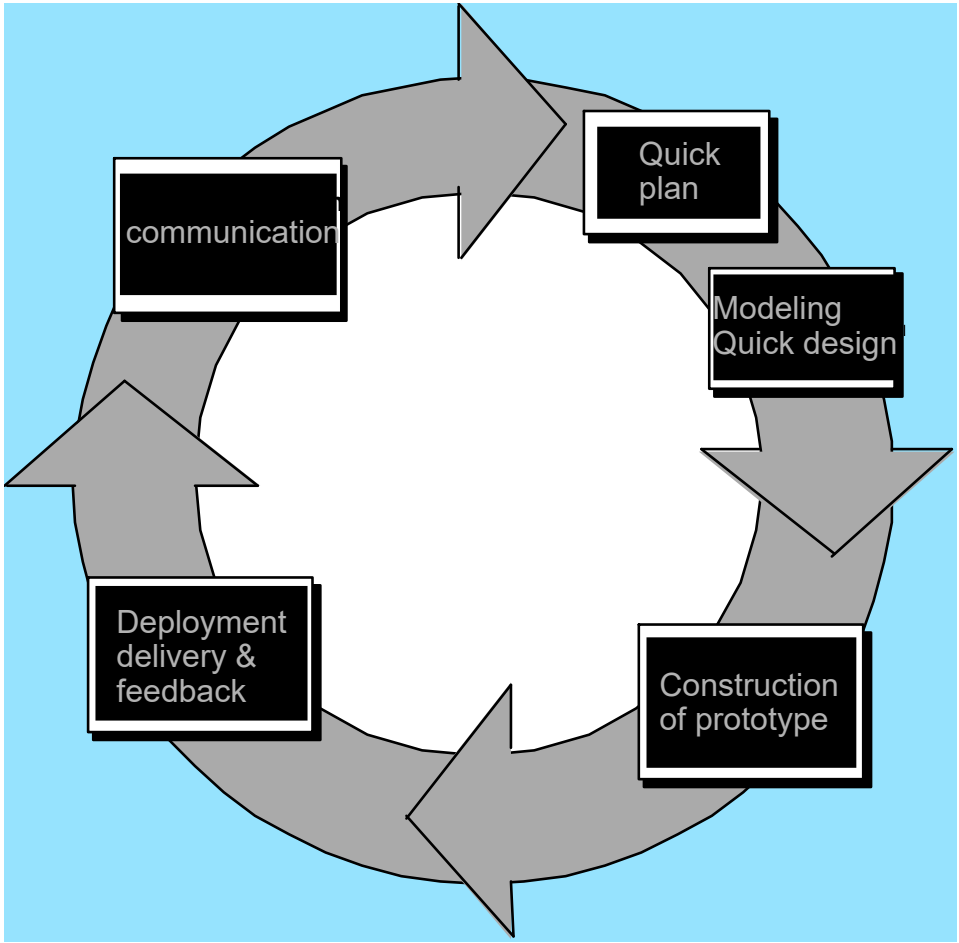
- 综合线性过程流和并行过程流



演化过程模型

- 需求：专门针对不断演变的软件产品的过程模型
  - 产品需求经常发生变化
  - 交付时间严格，功能有限的版本
  - 理解产品核心需求，未定义扩展细节
- 演化过程模型：本质是迭代
  - 原型开发
  - 螺旋模型

原型开发（Prototyping）



1）沟通

2）快速策划

3）建模快速设计

4）构建原型

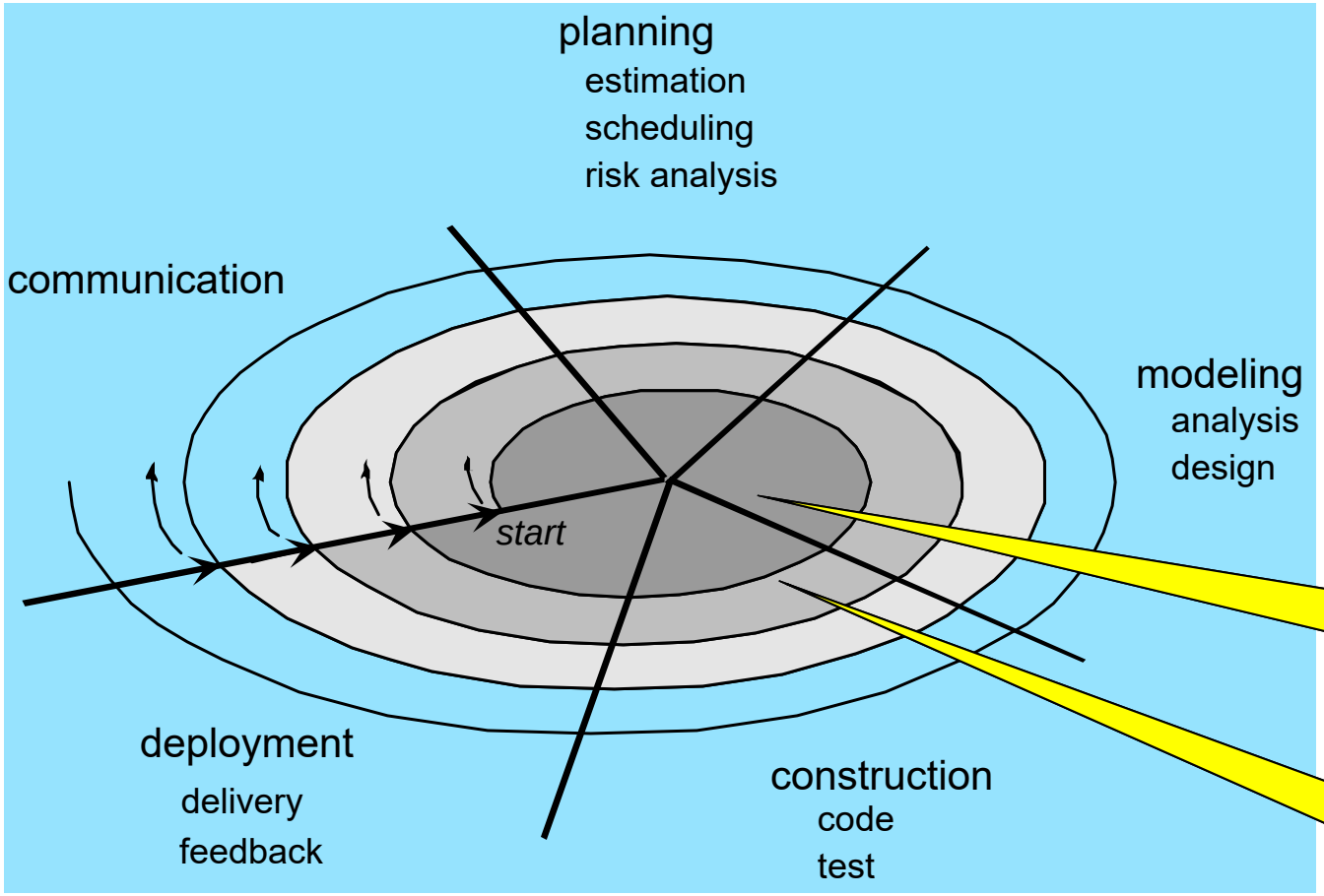
5）部署交付及反馈

原型系统提供了定义软件需求的一种机制

螺旋模型

- 螺旋模型：一种演进式软件过程模型
  - 原型开发的迭代性质
  - 瀑布模型的系统性和可控性
  - 快速开发越来越完善的软件版本
- 特点：
  - 风险驱动型，以降低风险为目标
  - 采用循环方式逐步加深系统定义和实现深度
  - 确定一系列的里程碑

螺旋模型



1）沟通

2）策划

3）建模

4）构建

5）部署

产品的规格说明

迭代完善的产品的原型系统

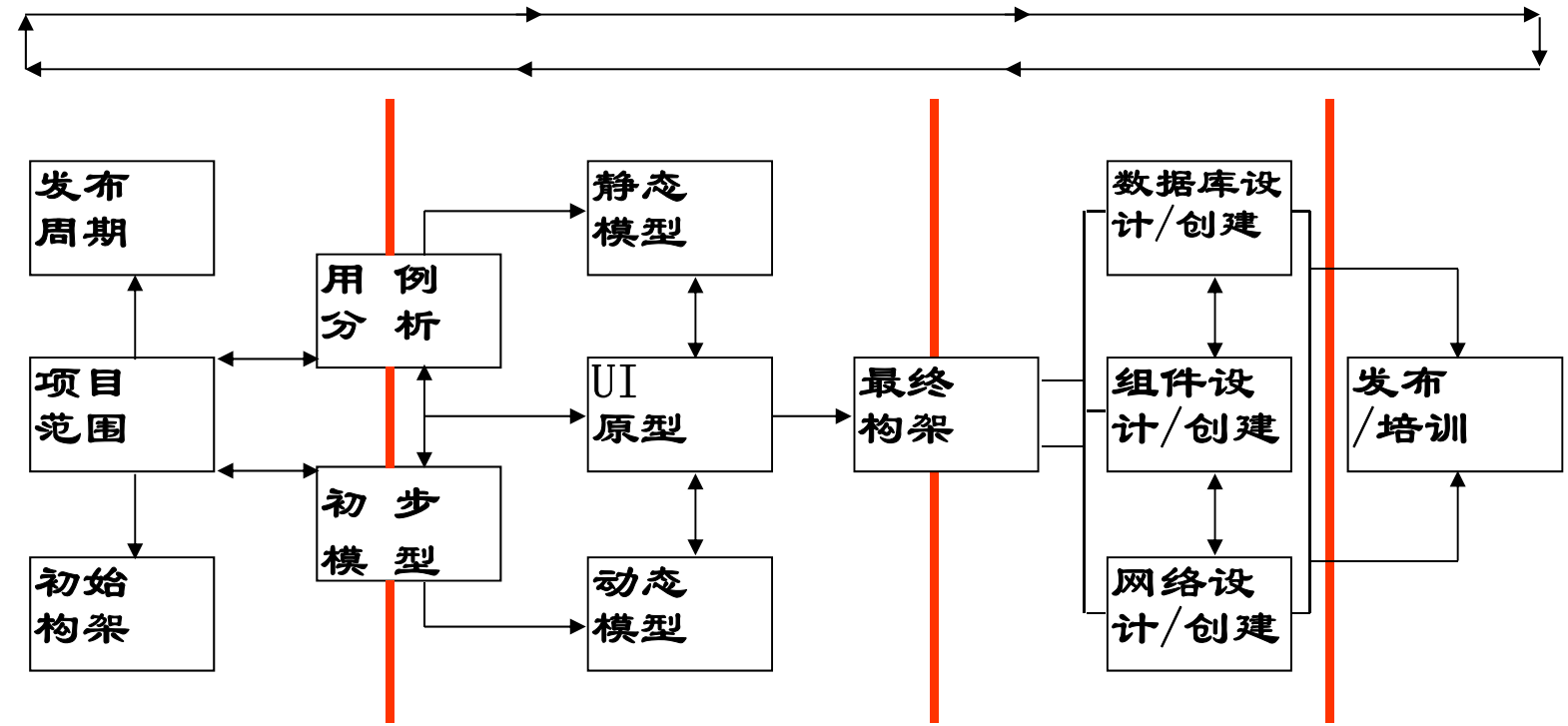
协同开发模型

- 适合于不同的工程团队共同开发的系统工程项目
- 所有的软件工程活动同时存在并处于不同的状态
  - 非活动状态
  - 正在开发状态
  - 等待变更请求
  - 正在评审
  - 正在修改
  - 建立基线
  - 完成

1）定义了一个过程网络，所有活动、动作和任务同时存在

2）定义了一系列事件，触发活动、动作和任务的状态转换

协同过程模型


发布
周期
项目
范围
初始
构架
用 例
分 析
初 步
模 型
静态
模型
UI
原型
动态
模型
最终
构架
数据库设
计/创建
组件设
计/创建
网络设
计/创建
发布
/培训
初始
细化
构造
移交

（3）专用过程模型

- 专用过程模型：应用面较窄而专一
- 基于构件的开发
- 形式化方法模型
- 面向对象的开发：UP模型
- 面向方面的开发
- 敏捷过程模型

基于构件的开发模型

- Component-based development model
- 采用预先打包的软件构件开发应用系统
- 软件复用：reuse
- 构件：具有良好定义的接口，提供特定的功能
  - 通用的软件模块
  - 面向对象的类或者软件包
- 特点：本质是演化模型，以迭代方式构建软件

构件开发步骤

- 对问题领域，研究和评估可用的构件产品
- 考虑构件集成问题
- 设计软件架构，容纳选择的构件集合
- 将构件集成到架构中
- 进行充分地测试，保证功能正确

形式化方法模型

- Formal methods model 采用数学方法进行开发和验证
- 生成计算机软件形式化的数学规格说明
  - 使用形式化方法，解决歧义性、不完整和不一致问题
  - 应用数学分析的方法，不是依靠特定地评审
  - 设计阶段是程序验证的基础
- 应用领域
  - 高度关注安全性的软件（如飞行器、医疗类）
  - 开发出错将导致重大经济损失的软件
- 应用现状
  - 难于在商业环境中应用（对开发人员和客户要求高）

面向方面的开发

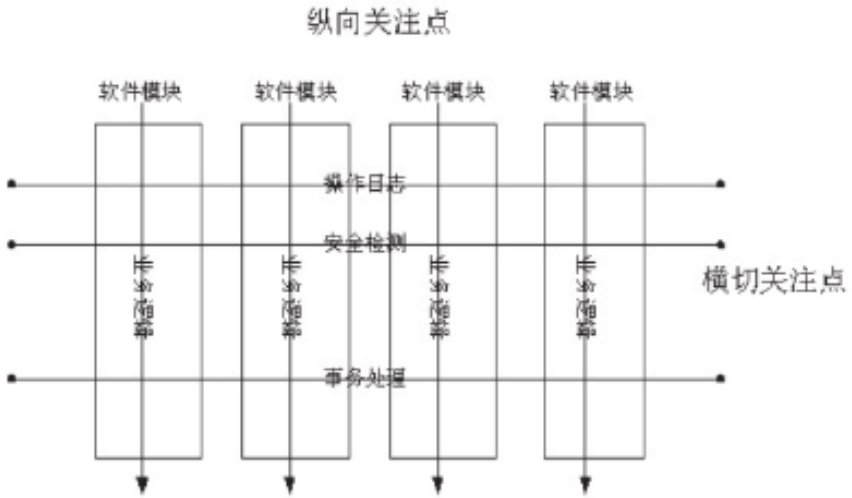
- Aspect-Oriented Software Development [较新]
- 关注点：客户需要的属性或者技术兴趣点
  - 系统高层属性（如：安全性、容错能力）
  - 系统性（如：任务同步或内存管理）
  - 系统功能（如：商业规则的应用）
- 方面：对纵向分解的软件进行横向切片，以表示软件构件的功能及非功能的横切属性。
- 为定义、说明、设计和构建Aspect提供过程和方法
- AOSD—AOP—AOCE

面向方面编程

- 面向方面编程（aspect-oriented programming，AOP）是一种编程范式，旨在提高模块化允许横向关注点的分离。该范式以一种成为方面（Aspect）的语言构造为基础，切面是一种新的模块化机制，用来描述分散在对象、类或函数中的横切关注点（Crosscutting Concern）。
- 核心思想：从主关注点中分离出横切关注点。
- 分离关注点使得解决特定领域问题的代码从业务逻辑中独立出来，业务逻辑的代码中不再含有针对特定领域问题代码的调用。

示例说明

- 日志处理
  - 分布与各个模块中（横向）
  - 不是业务逻辑部分，但确是构成系统完整性的重要部分
  - OOP解决：纵向调用，引入模块高耦合

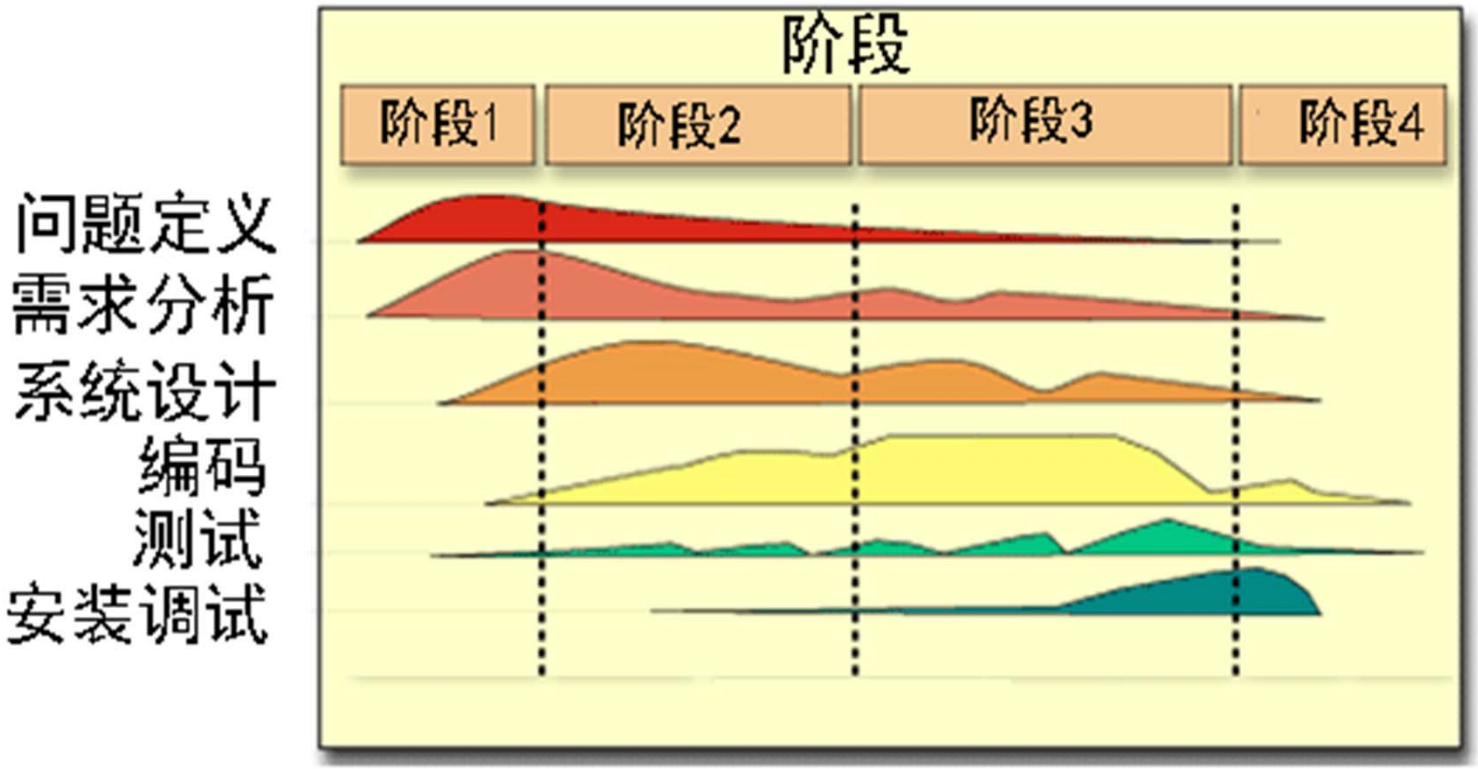


横向切片

日志记录、异常处理、事务处横切拦截

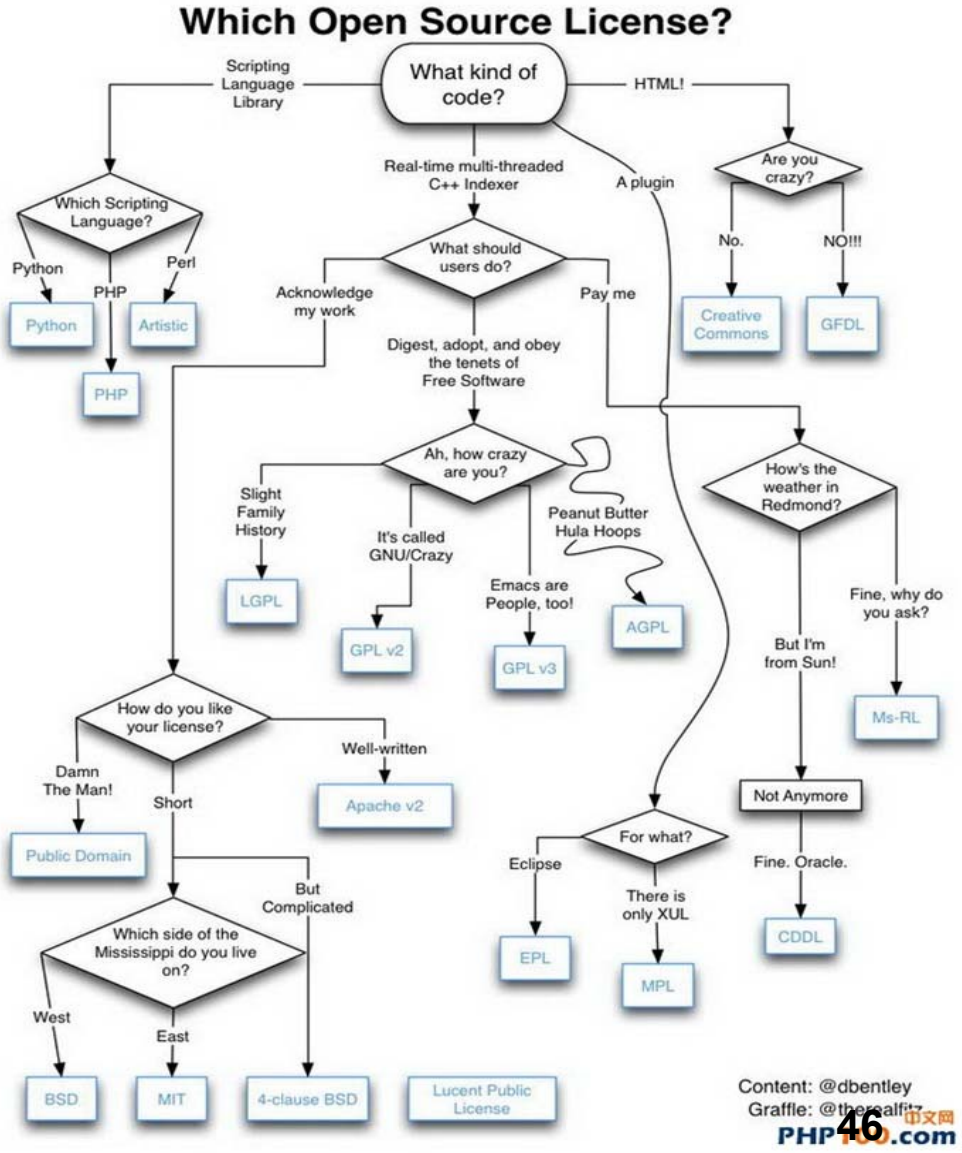
动态代理模式，装修者模式

面向对象的软件生命周期模型



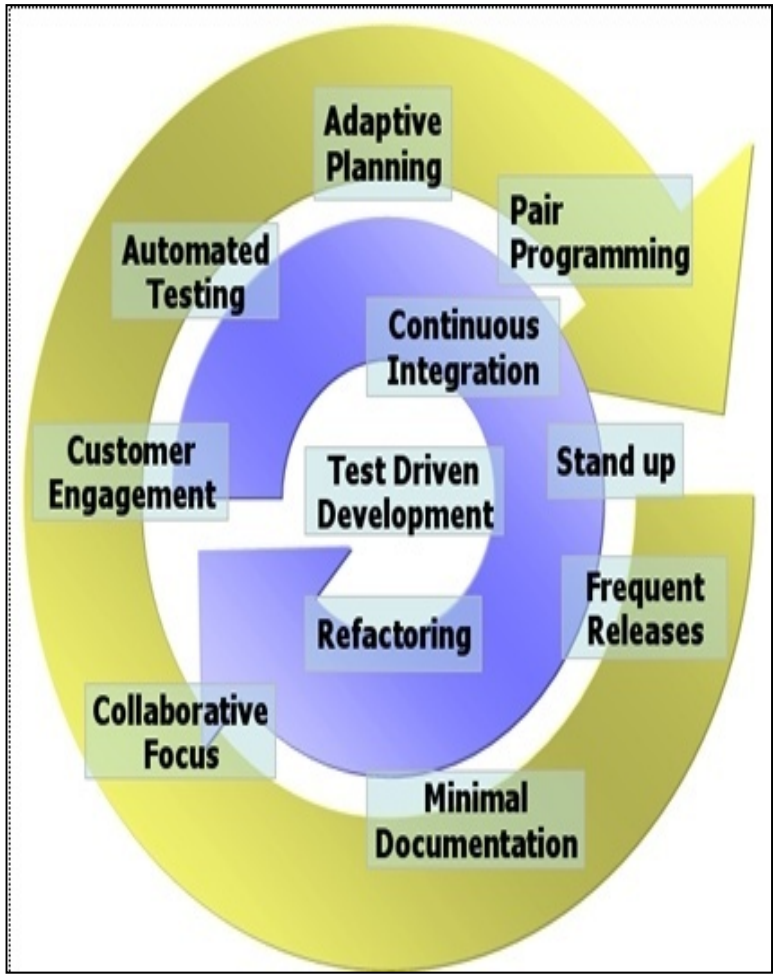
开源生命周期模型

- 开源生命周期模型
- 一个有编程构想的人建立初始版本
- 通过互联网免费下载
  sourceForge.net
  FreshMeat.net
- 一些用户报告缺点，另一些用户修改；一些用户提出扩充想法，另一些用户实现这些想法



敏捷过程模型

- 敏捷开发是一种以人为核心、迭代、循序渐进的开发方法。
- 在敏捷开发中，软件项目的构建被切分成多个子项目，各个子项目的成果都经过测试，具备集成和可运行的特征。
- 就是把一个大项目分为多个相互联系，但也可独立运行的小项目，并分别完成，在此过程中软件一直处于可使用状态。



主要流派

- 极限编程： Extreme Programming
  - Communication、Simplicity、Feedback
  - Courage、Modesty
  - 13个核心实践
  - 有限资源 & 有限时间 & 小项目
- 动态系统开发
  - Dynamic System Development Method
  - 9条基本原则
  - 倡导以业务为核心，快速而有效的进行系统开发
  - 增量原型，80%的应用系统可以用20%的时间交付

主要流派（续）

- 特征驱动开发
  - Feature-Driven Development，FDD
  - 以实现功能为目标
  - 域建模：四色原型，Color UML
  - 每个功能开发时间不超过两周
  - Domain expert、Chief Architect、Chief Programmers
- SCRUM
  - 没有一种固定的流程可以保证项目成功
  - 高度自主权、紧密沟通合作
  - Pigs： Product Owner、ScrumMaster、Team
  - Chickens：Users、Stakeholders、Managers

Scrum 框架

角色

- Product owner
- ScrumMaster
- Team

活动

- Sprint planning
- Sprint review
- Sprint retrospective
- Daily scrum meeting

工作产品

- Product backlog
- Sprint backlog
- Burndown charts

敏捷过程模型（续1）

- 结队编程（Pair Programming）
- 15到20分钟轮换
- 不进行键盘操作的人检查代码

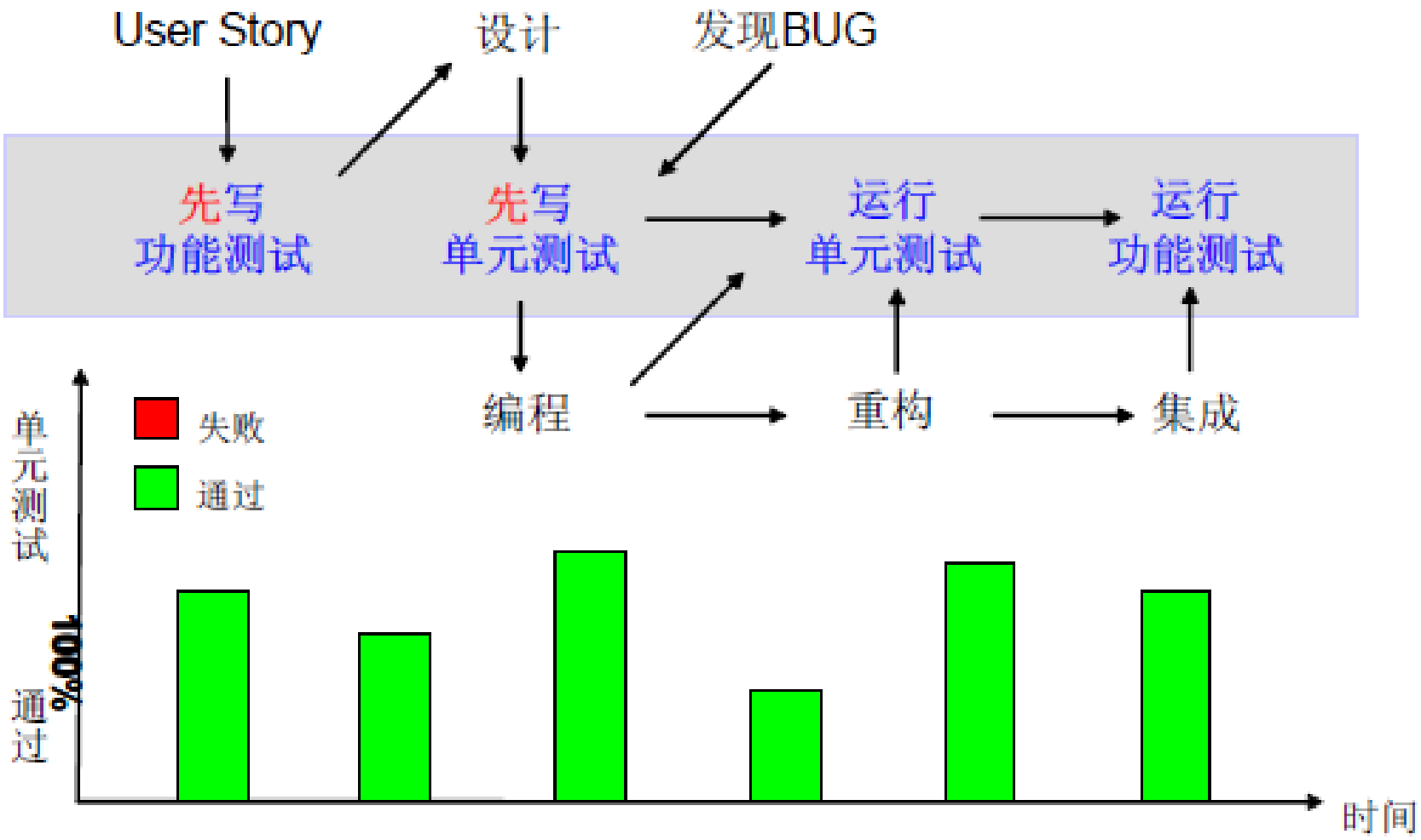
敏捷过程模型（续2）

- 时间盒（Time Boxing）
- 2到3周，频繁交付运行软件
- 为一个任务设定一段时间，小组成员在这段时间尽可能地做好工作
- 如果在时间盒内不能完成任务，这个迭代工作将简化

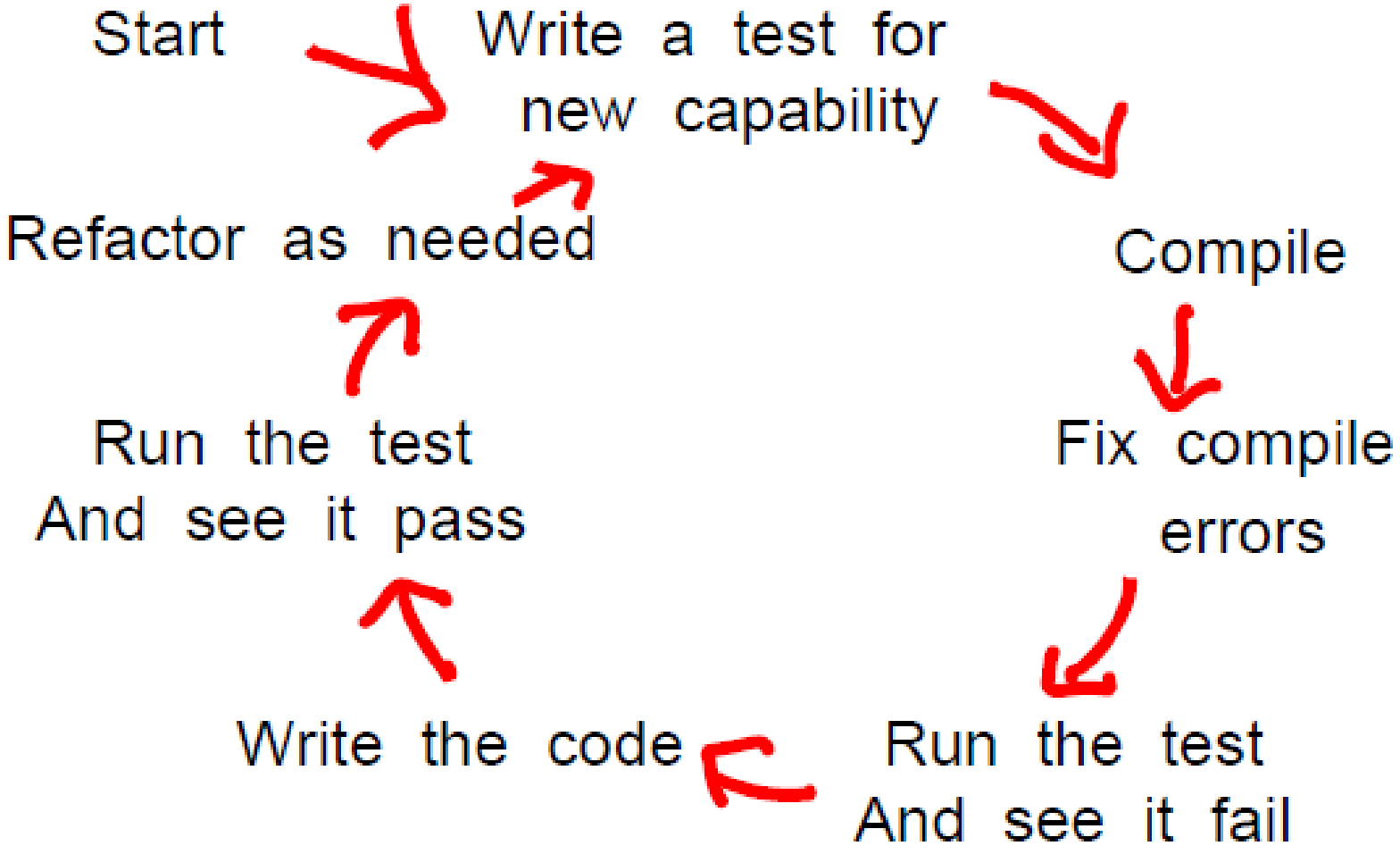
敏捷过程模型（续3）

- 站立会议
- 每天、所有成员、站成一个圈
- 昨天做了什么
- 今天的计划
- 将遇到什么问题
- 我们忘记了什么
- 可分享的经验

测试驱动的开发（TDD）

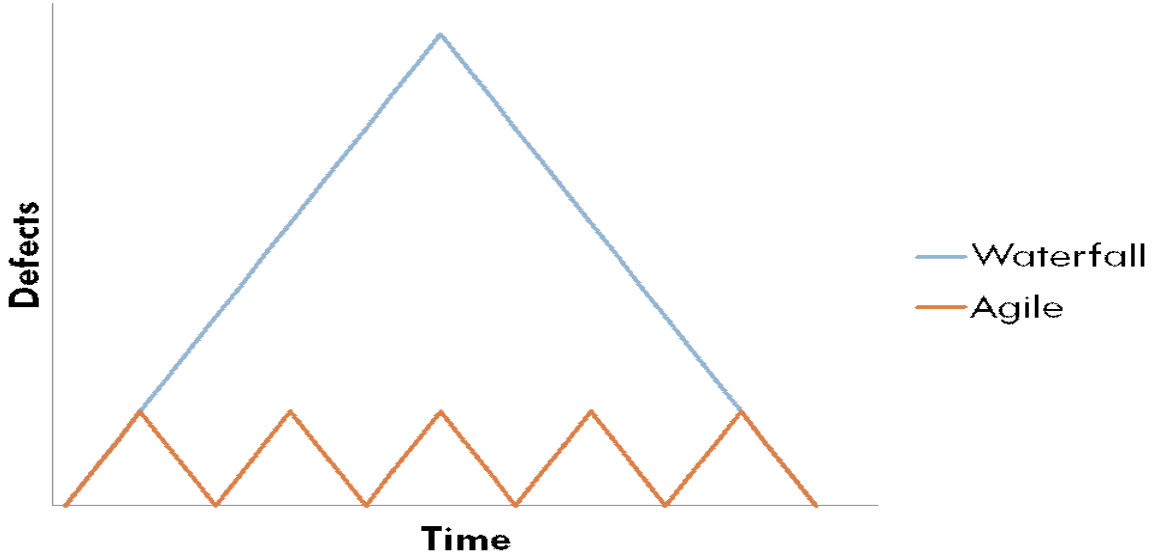

User Story
设计
发现BUG
先写
功能测试
先写
单元测试
运行
单元测试
运行
功能测试
编程
重构
集成
单元测试
失败
通过
100%
通过
时间

测试驱动的开发（续1）


Start
Write a test for new capability
Compile
Fix compile errors
Run the test And see it fail
Write the code
Run the test And see it pass
Refactor as needed

测试驱动的开发（续2）

- 尽早测试 (Unit testing)
- 经常测试 (Continuous regression testing)
- 自动测试 (Testing a part of build process)
- 设计可测试的软件
- 单元测试与功能测试

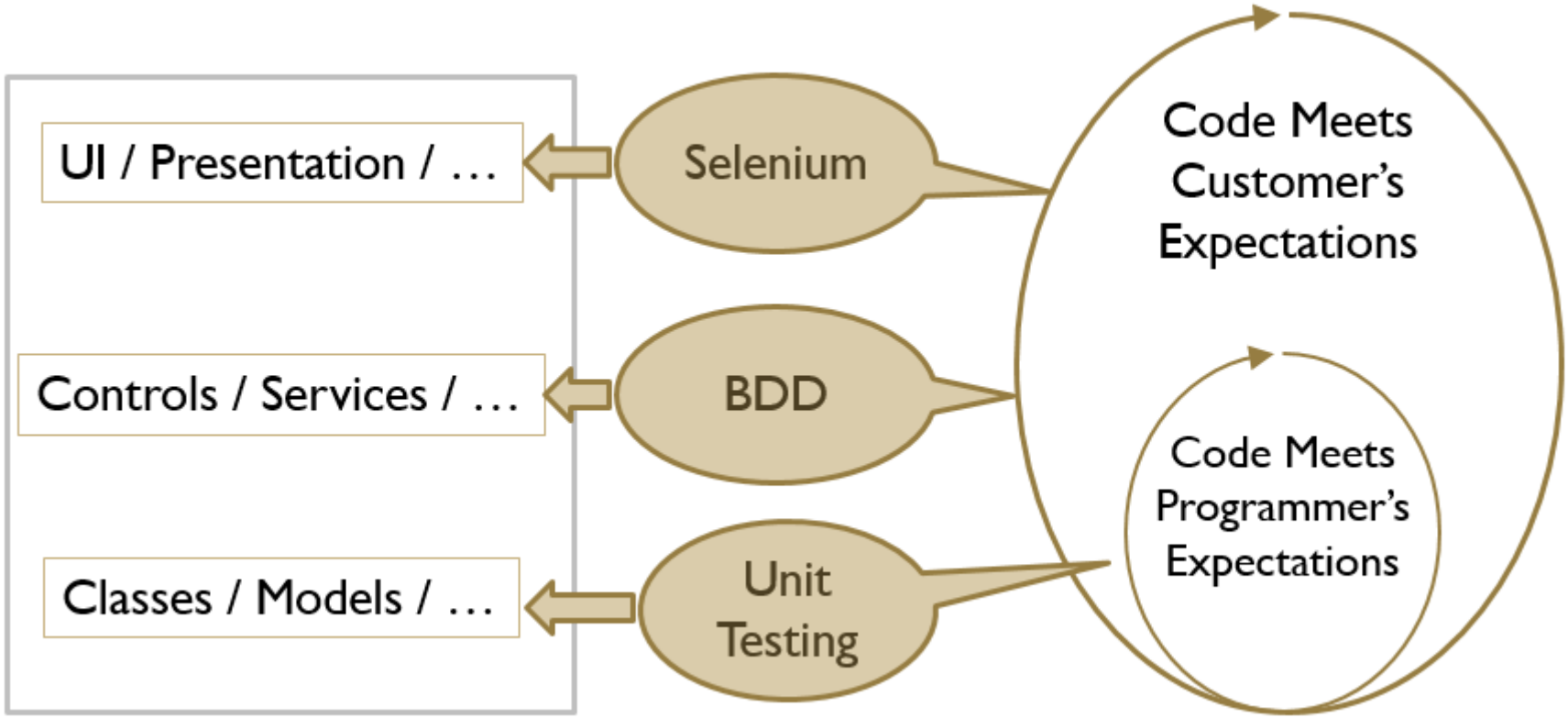


行为驱动的开发(BDD)

- 行为驱动开发： Behavior Driven Development
- 是一种敏捷软件开发的技术，它鼓励软件项目中的开发者、QA和非技术人员或商业参与者之间的协作
- 2009， Dan North给出BDD的定义：
  - 第二代的、由外及内的、基于拉(pull)的、多方利益相关者的(stakeholder)、多种可扩展的、高自动化的敏捷方法
  - 一个交互循环，可以具有带有良好定义的输出，即：已测试过的软件
  - 通过与利益相关者的讨论，取得对预期的软件行为的清醒认识
  - 通过用自然语言书写非程序员可读的测试用例

BDD的由来

- 传统的BDD：更侧重代码的功能逻辑



BDD VS. TDD

- TDD中侧重：开发
  - ✓ 通过测试用例来规范约束开发者编写出质量更高、bug更少的代码
- BDD中侧重：设计
  - ✓ 要求在设计测试用例的时候对系统进行定义，倡导使用通用的语言将系统的行为描述出来，将系统设计和测试用例结合起来，从而以此为驱动进行开发工作
- 举例：创建一个Stack并初始化为空
- TDD：test create stack，期望输出用assert验证
- BDD：it should be empty when created

BDD

- BDD的通用语言是一种近乎自然语言的描述软件的形式
- **Story**：系统业务专家、开发者、测试人员一起合作，分析软件的需求，然后将这些需求写成一个个的故事。开发者负责填充这些故事的内容，测试者负责检验这些故事的结果。

Story:

```
1  As a 角色
2  I want 特征
3  so that 利益
```

As a标识出这个系统行为是为哪一个角色而定义的。

I want和so that则指明了该角色想做的事，以及想达到的目的。

这三个断句定义了这个系统行为的参与者、范围。

BDD（续）

- **Scenario**：同一个Story可能会有不同的场景描述

Story:

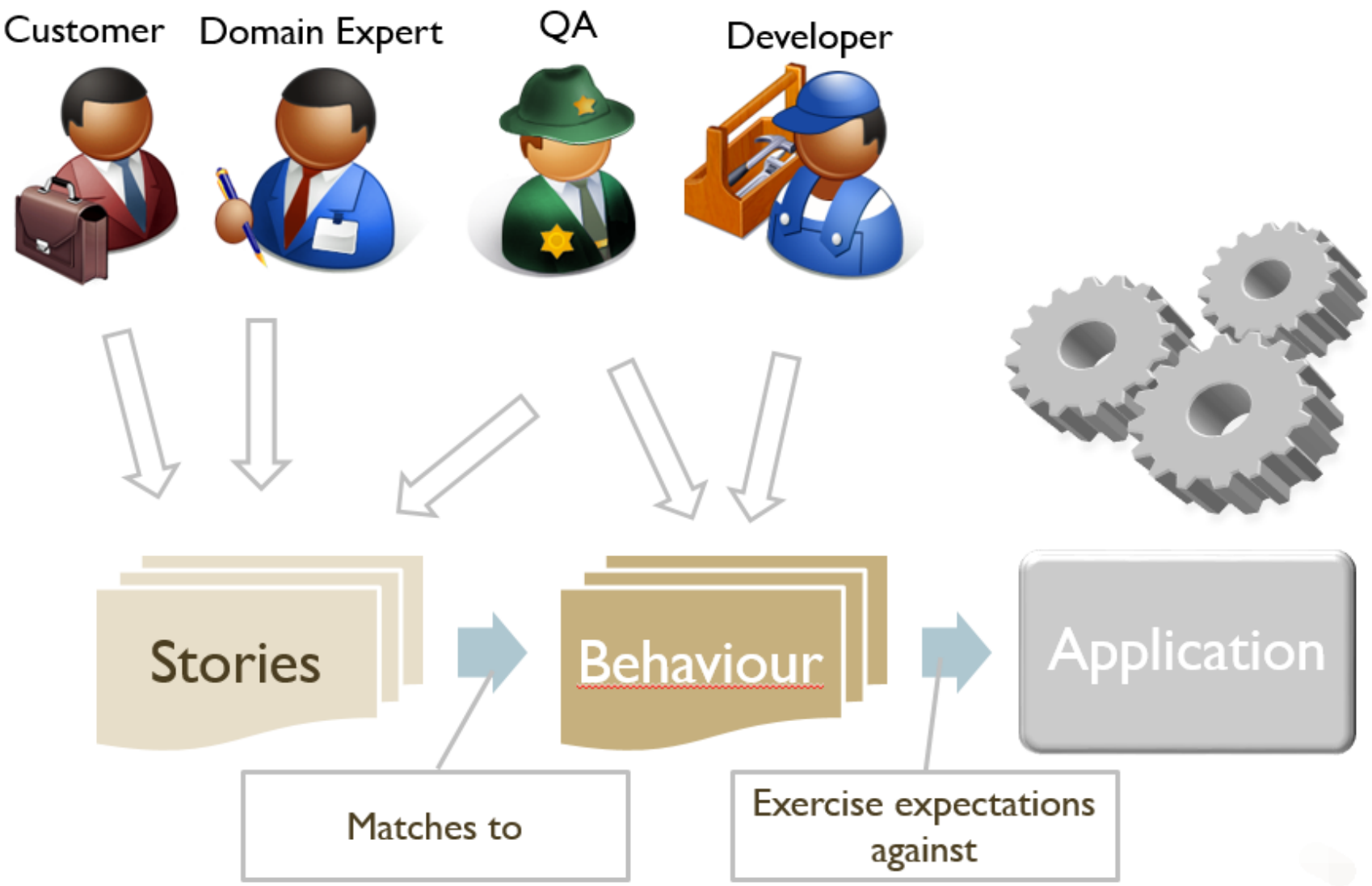
```
1  As a 角色
2  I want 特征
3  so that 利益
```

Scenario:

```
1  Given [上下文]
2      And [更多的上下文]
3  When [事件]
4  Then [结果]
5      And [其他结果]
```

- 场景中的Given…When…Then…实际上就是设定该场景的状态、适用的事件，以及场景的执行结果。
- 通过这样的Story描述和场景设置，基本就完成了一个完整测试的定义。

BDD 测试流程

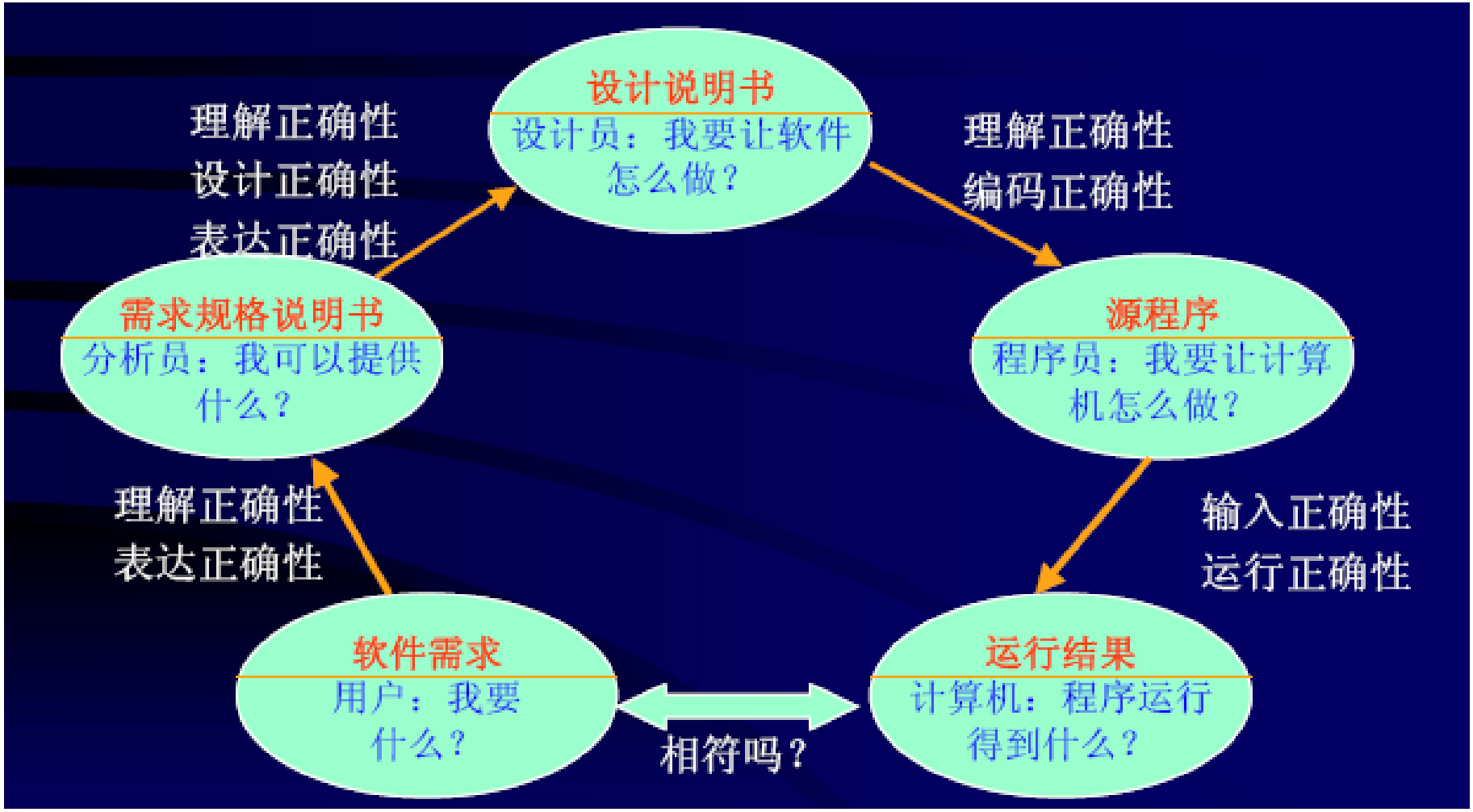

Customer
Domain Expert
QA
Developer
Stories
Behaviour
Application
Matches to
Exercise expectations against

BDD 框架及测试工具

常见的**BDD**框架：
C – Cspec
C++ – CppSpec, Spec-CPP
.Net – NBehave, NSpecify, SpecFlow
Groovy – GSpec, easyb, Cuke4Duke
PHP – PHPSpec
Python – Specipy
Ruby – RSpec, Shoulda, Cucumber

与**Java**相关的**BDD**测试工具：
JBehave – Java annotations based, Test frameworks agnostic
Cuke4duke – Cucumber support for JVM
JDave – RSpec (Ruby) inspired, Mojo 2 & Hamcrest based
beanSpec – Java based
easyb – Java based, Specifications written in Groovy
instinct – BDD framework for Java, providing annotations for contexts. Inspired by Rspec
BDoc - Extracts behaviour from unit tests

3、软件开发模型中的测试



软件测试过程

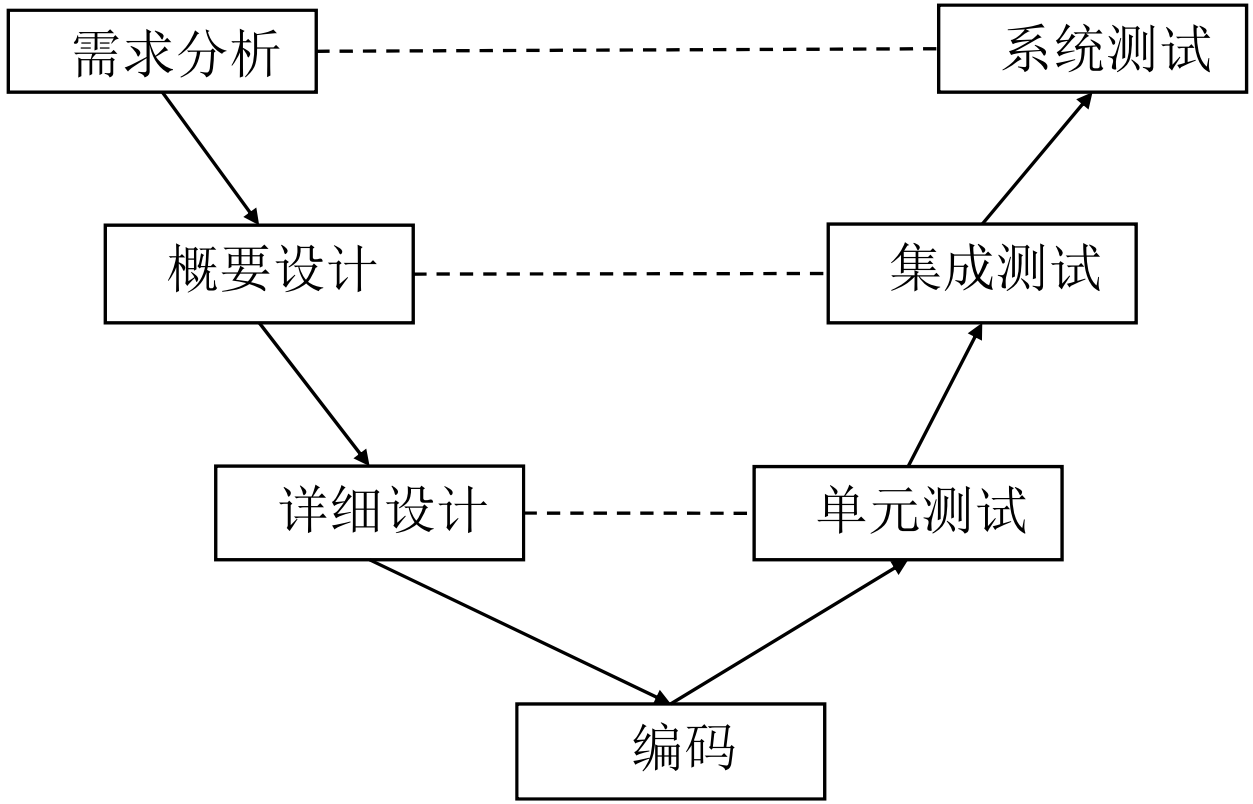
- 测试什么？（条件、范围）
- 如何测试？（ 测试用例的制作）
- 建立测试环境（软件、硬件、网络等）

测试环境＝硬件＋软件＋网络＋数据准备＋测试工具

- 执行测试
- 评估测试结果

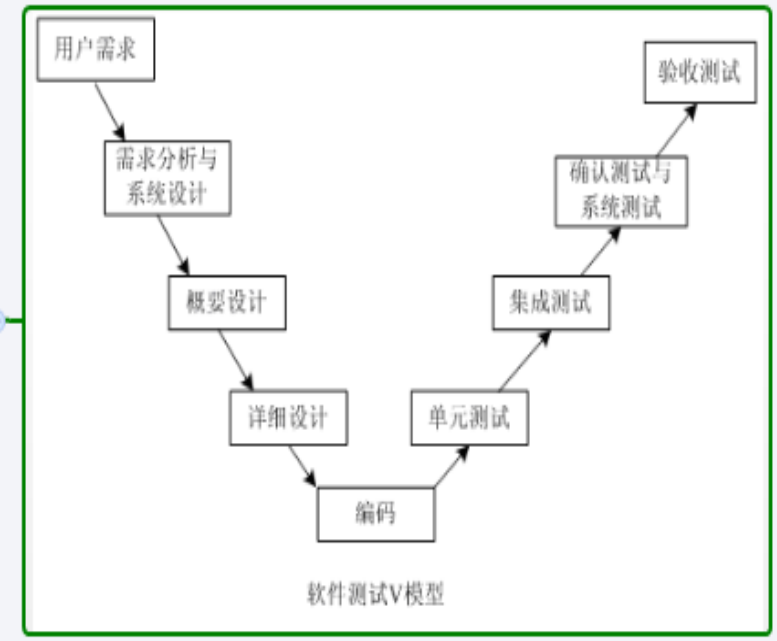
常用的软件测试过程模型

V模型：每一测试阶段的前提和基础是对应开发阶段的文档。



V模型

模型图



描述

单元和集成测试： 验证程序设计，开发人员和测试组应检测程序的执行是否满足软件设计的要求

系统测试：验证系统设计，检测系统功能、性能的质量特性是否达到系统设计的指标

确认测试和验收测试： 测试人员和用户执行测试，并追溯软件需求说明书进行测试，以确定软件的实现是否满足用户需求或合同的要求

优缺点

优点

1．改进了瀑布模型

2．反映了测试活动与分析和设计的关系

3．制定的测试策略既包括低层测试又包括了高层测试

- 低层测试：验证源代码的正确性
- 高层测试： 整个系统满足用户的需求

缺点

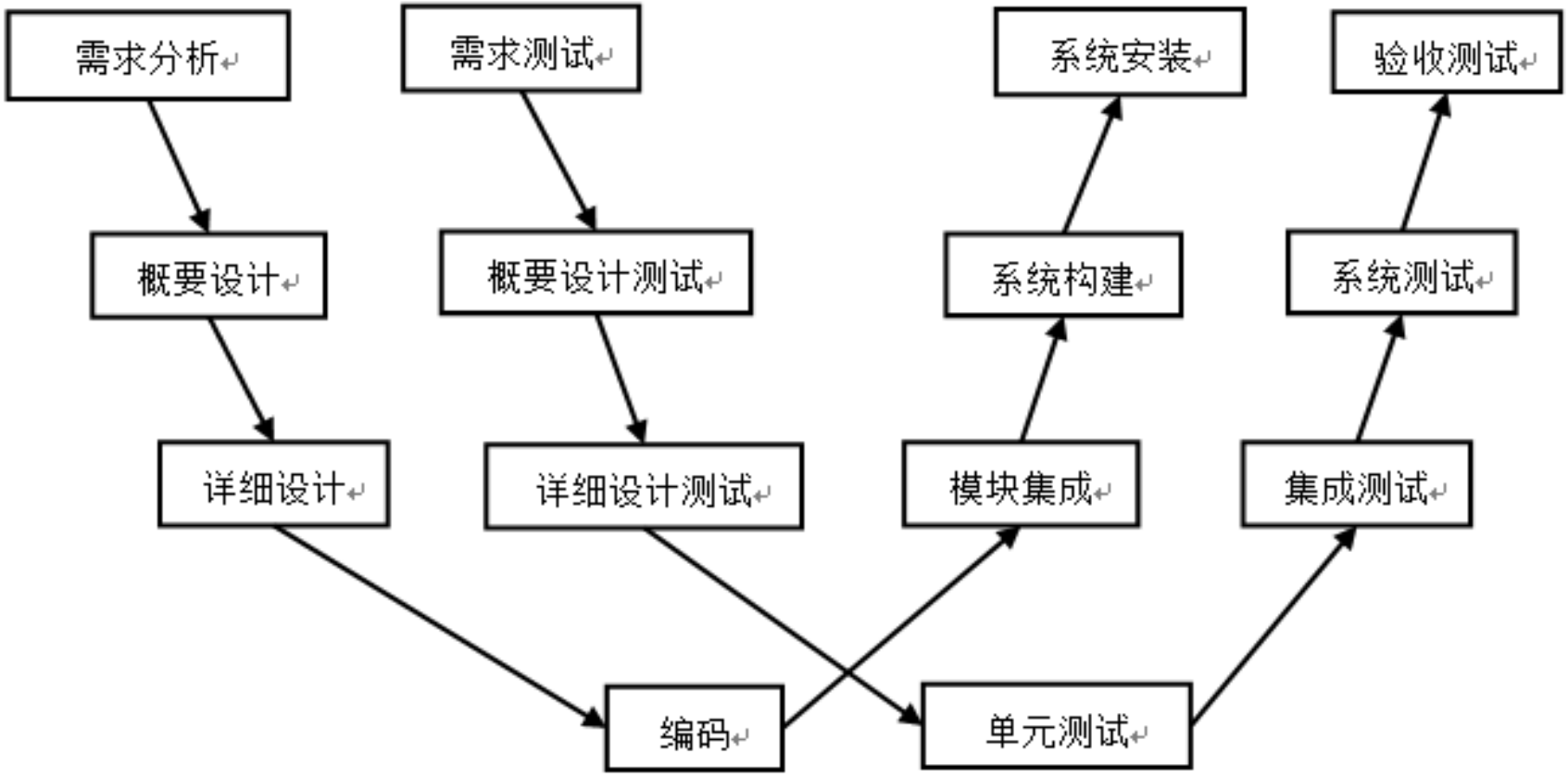
1．测试过程作为在需求分析、概要设计、详细设计及编码之后的一个阶段

2．该模型容易使人理解主要是针对程序进行测试寻找错误

3．需求阶段分析错误，直到后期才被发现。

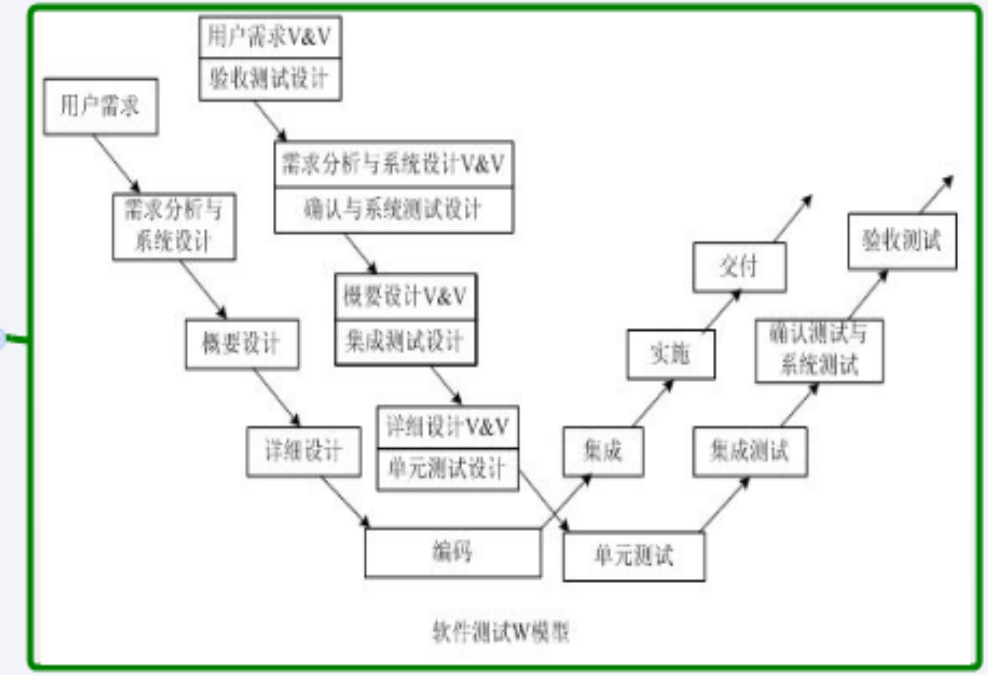
常用的软件测试过程模型

W模型：由两个V型结构组成，分别代表测试与开发过程，每一个开发过程对应一个测试过程。



W模型

模型图



描述

W模型可以说是V模型自然而言的发展

W模型强调：测试伴随着整个软件开发周期，而且测试的对象不仅仅是程序，需求、功能和设计同样要测试

一旦有文档提供，就要及时确定测试条件，以及编写测试用例

优缺点

优点

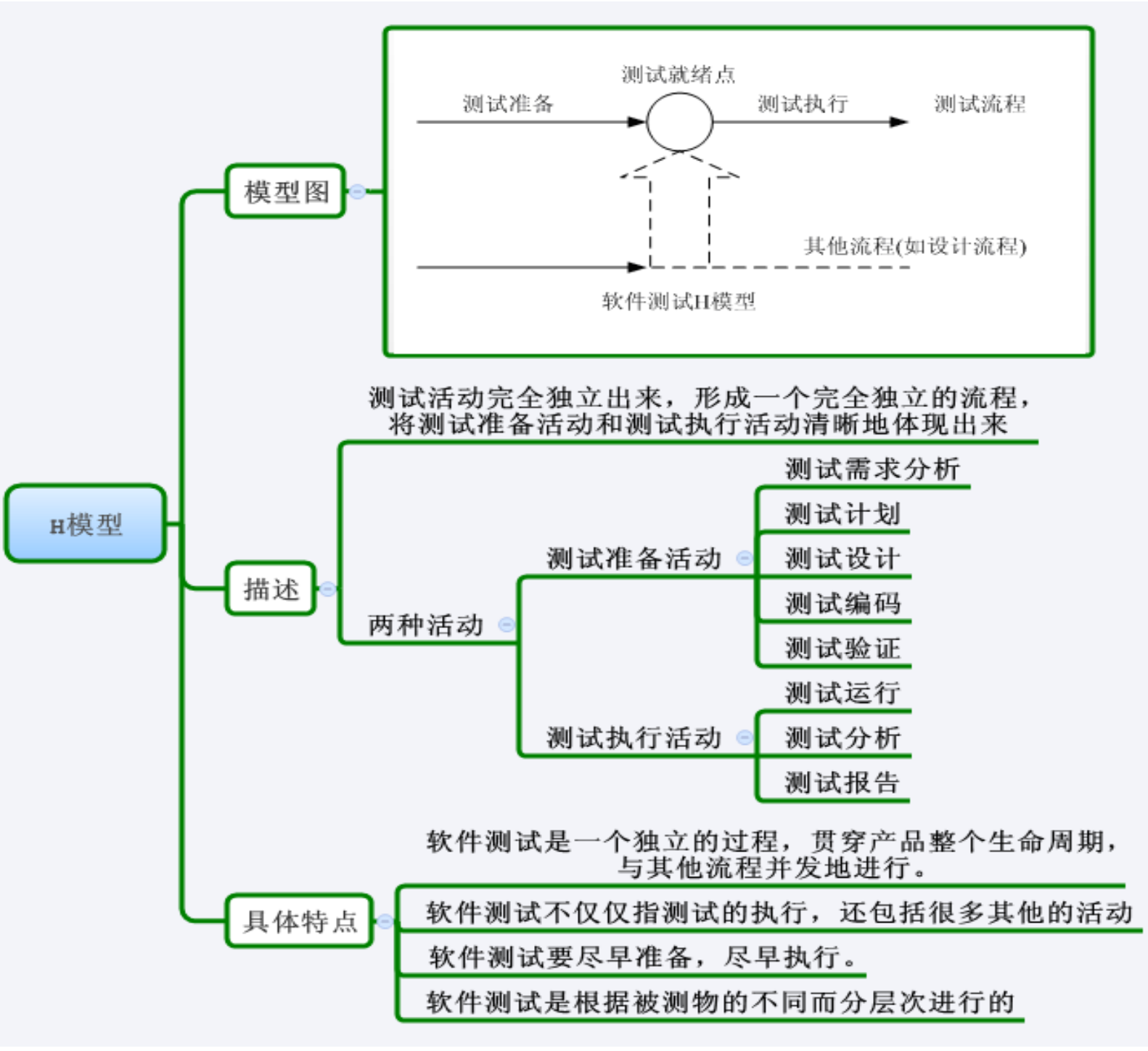
尽可能早地找出各种规格说明的缺陷

测试与开发是同步进行的，从而有利于尽早地发现问题

缺点

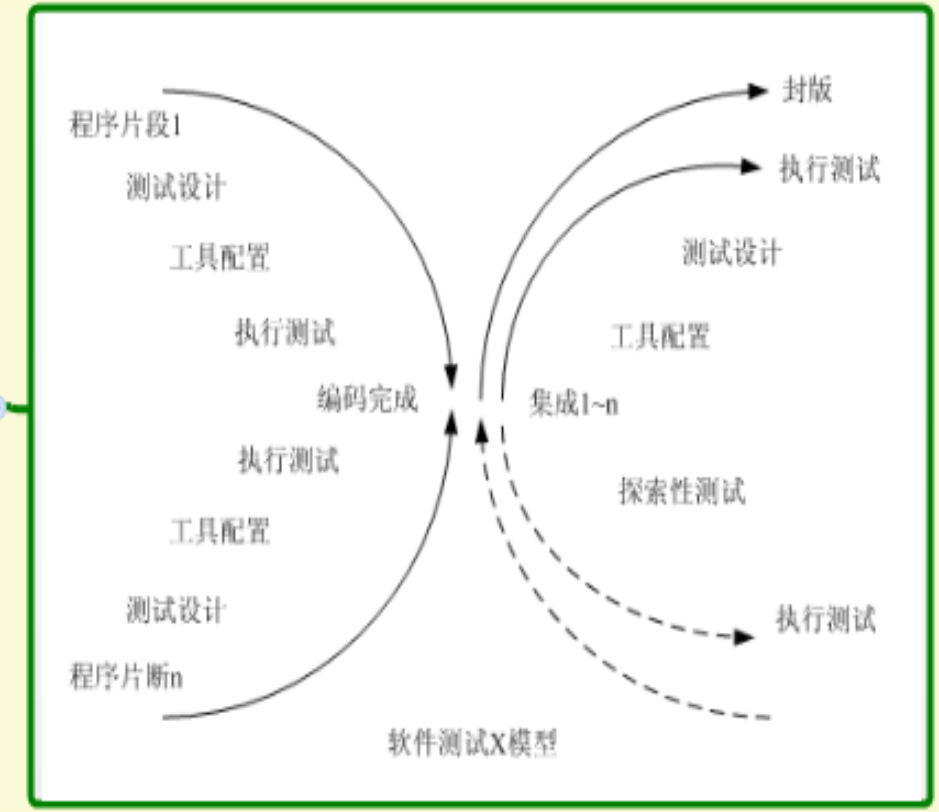
把软件的开发视为需求、设计、编码等一系列串行的活动，从而使程序开发无法迭代，自更新。

当前很多文档需要事后补充，或者根本没有文档的做法下(这已成为一种开发的文化)，开发人员和测试人员都面临同样的困惑


H模型
模型图
测试就绪点
测试准备
测试执行
测试流程
其他流程(如设计流程)
软件测试H模型
描述
测试活动完全独立出来，形成一个完全独立的流程，
将测试准备活动和测试执行活动清晰地体现出来
两种活动
测试准备活动
测试需求分析
测试计划
测试设计
测试编码
测试验证
测试执行活动
测试运行
测试分析
测试报告
具体特点
软件测试是一个独立的过程，贯穿产品整个生命周期，
与其他流程并发地进行。
软件测试不仅仅指测试的执行，还包括很多其他的活动
软件测试要尽早准备，尽早执行。
软件测试是根据被测物的不同而分层次进行的

X模型

模型图



描述

- 出现原因
  - 批评V模型而提出的
    - V模型太死板，太学院派
    - 单元测试和集成测试在实际工作中没有那么完全的区分。
- 模型概述
  - 左边：针对单独程序片段所进行的相互分离的编码和测试。
  - 右边： 进行频繁的交接与集成，形成可执行程序，进行对EXE的测试
  - 右下方： 描述了探索型测试

4、软件测试发展

- Debugging oriented（1950年左右）
- Demonstration oriented（1960年左右）
- Destruction oriented（1970年左右）
- Evaluation oriented（1980年左右）
- Prevention oriented（1990年以后）
- Professional、education and research（2000年以后）

软件测试发展（续）

- 软件测试理论的发展
- 20世纪50年代，图灵给出了软件测试的原始定义：测试是程序正确性证明的一种极端实验形式。
- 20世纪70年代以后：
  - Goodenough首次提出了软件测试理论
  - Program Test Methods ，Hetzel
  - 美国北卡来纳大学召开了首次软件测试技术会议

软件测试技术的发展

- “程序插装”测试方法
- 符号测试系统，Howden和Clarke
- 错误驱动测试，Demmo
- 数据流测试方法,Osterweit和Fosdick
- 路径测试覆盖准则
- 面向对象的软件测试，1994年9月，Communication of ACM
- 自动化测试：应用越来越普遍！

自动化测试

- 针对软件测试的重复特点：
  - 同样的一个功能点或是业务流程需要借助于不同类型的数据驱动而运行很多遍
  - 由于某一个功能模块的修改而有可能影响其它模块而需要进行回归测试，包括以前用过的测试用例
- 目前常见的自动化测试技术应用：
  - 功能测试
  - 性能测试：负载压力测试

目前测试技术的趋势

- WEB应用测试：Cloud
- 手机软件测试
- 嵌入式软件测试
- 游戏软件测试
- 安全测试
- 可靠性测试
- Application of Big data?

本节小结

- 软件测试的目的
- 软件测试的定义
- 软件测试与软件开发
- 软件测试发展