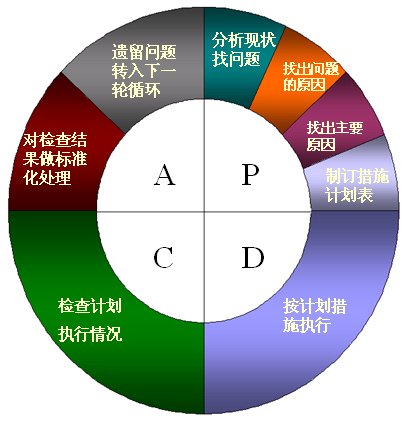
### 软件生命周期模型

# 一、工程过程

* 1. 工程项目目标：合理进度、有限经费、一定质量
     1. **PDCA循环**，针对**质量目标**提出的模型，称为戴明环（Plan, Do, Check, Action）
  2. **软件工程过程**
     1. 为了获得**软件产品**，在**软件工具**的支持下，由软件工程师完成的一系列软件工程活动。
     2. 具体活动包括:
        1. 软件规格说明
        2. 软件开发
        3. 软件确认
        4. 软件演进
  3. **软件生命周期**：
     1. 指软件产品从考虑其概念开始，到该软件产品不再使用为止的整个时期。
     2. 软件生命周期的六个基本步骤：
        1. 制定计划（P）
        2. 需求分析（D）
        3. 设计（D）
        4. 程序编码（D）
        5. 测试（C）
        6. 运行维护（A）
     3. 软件生命周期和戴明环的对应关系：



* 1. **软件过程模型**：
     1. 即软件生命周期模型，是对软件过程的概括描述和抽象。
     2. 包括：
        1. 各种活动（Activities）
        2. 软件工件（artifacts）
        3. 参与角色（Actors/Roles）
  2. 软件生命周期模型
     1. 需求定义
     2. 跨生存期的软件开发、运行、维护全过程，活动任务描述生命周期不同阶段的软件工件，明确活动和执行角色
     3. 指导软件开发人员按照确定的框架结构和活动进行软件开发的标准。

# 二、传统软件生命周期模型

### 瀑布模型：

* 1. 定义：规定软件生命周期模型提出的一些基本工程活动，并规定他们自上而下相互衔接的固定次序。
  2. **阶段划分**：系统需求--软件需求---分析---程序设计---编码---测试---运行
  3. 每个阶段都需要进行审核和文档输出
  4. 意义：为软件开发和维护提供了有效的管理模式。在软件开发早期，在消除非结构化软件、降低软件复杂度、促进软件开发工程化方面起着显著的作用。
  5. 每个开发活动的特征：
     1. 本活动依赖于上一项活动的输出作为工作对象，这些输出一般是代表某活动结束的里程碑式文档。
     2. 根据本阶段的活动规程执行相应的任务。
     3. 本阶段活动的最终产出——软件工件，将会作为下一活动的输入。
     4. 对本阶段活动执行情况进行评审。
  6. 优点
     1. 降低复杂度、提高透明性和可管理性、强调需求分析和设计、阶段审核和文档输出保证了阶段之间的正确衔接，能够及时发现问题
  7. 缺点
     1. 缺乏灵活性、不适用于需求不明的开发情况、风险控制能力较低、文档驱动增加了系统的工作量、只依赖于文档来评估进度，可能会得出错误结论、成功周期较长（do twice）
  8. 适用场景
     1. 需求明确、开发周期不紧张的较大型系统



### 演化模型

在瀑布模型的基础上，一次性开发难以成功，因此，演化模型提倡进行“两次开发”，分别是试验开发和产品开发。每个开发阶段按照瀑布模型进行具体开发活动。

优点：

明确用户需求、提高系统质量、降低开发风险

缺点：

管理难度提高、开发结构化较差、可能抛弃文档驱动、可能导致产品结构化较差

适用场景：

需求不清、开发周期短的中小型系统。



### 增量模型

结合瀑布模型和演化模型的优点，在需求不清时，对最核心或最清晰的需求，利用瀑布模型实现。再对后续需求进行重复开发（可能按照需求的优先级逐个进行），从而逐步建成一个完整的软件系统。

优点：

保障核心功能实现、开发风险较低、保障最高优先级的功能的可靠实现、提高系统稳定性和可维护性。

缺点：

增量粒度难以合理选择、确定所有的需求服务较困难

适用场景：

需求不清，了解核心需求的软件系统



### 喷泉模型/迭代模型

每个阶段是相互重叠、多次反复的。每个开发阶段没有次序要求，可以并发进行，在某个阶段随时补充其他阶段中遗漏的需求。

优点：

提高效率、缩短周期

缺点：

管理结构性较差



### V模型和W模型

#### V模型

是瀑布模型的变种。它将测试模块细化分解，把测试看作与开发同等重要的过程，每一测试阶段的前提和要求是对应开发阶段的文档。

测试模块：

·单元测试：根据详细设计说明书，检测每个单元模块是否符合预期要求。主要检查编码过程中可能存在的错误。

·集成测试：根据概要设计说明书，检测模块是否正确聚集。主要检查模块与接口之间可能存在的错误。

·系统测试：根据需求分析，检测系统作为一个整体在预定环境中能否正常的有效工作。

·验收测试： 是否满足客户的需求。

优点：

改进开发效率和开发效果

缺点：  
前期错误，后期难以挽救和弥补或者花费的代价巨大。



#### W模型

由两个V模型构成，分别代表测试与开发过程。每个开发过程对应一个测试，体现了开发和测试的并行关系，测试的不局限于程序，对于阶段文档也需要进行测试。

优点：

增加了软件各开发阶段中应同步进行的验证和确认活动。

缺点：

开发、测试活动保持着一种前后关系，无法支持迭代软件开发模型。



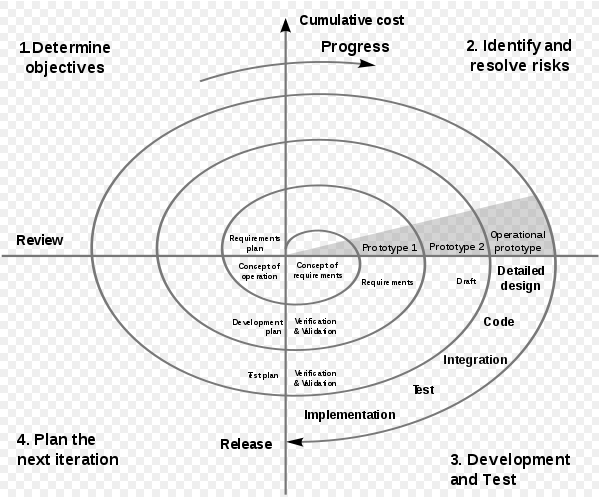
### 螺旋模型：

将开发过程分为四个类型：风险分析、制定计划、实施工程、客户评估。每次评估之后确定是否进行螺旋线的下一个回路。

适用对象：

风险较高、开发周期较长的大型软件项目

优点和缺点：降低风险，但是开发周期长。



### 构件组装模型

将整个系统模块化，在一定构件模型的支持下，复用构件库中软件构件，通过组装构件构造软件系统。开发过程就是构件组装的过程，是迭代的过程，维护过程就是构件升级、替换和扩充的过程。

优点：

软件复用充分，提高效率，允许多项目同时开发，降低开发费用，提高可维护性，可实现分步提交软件产品。

缺点：

构件组装结构没有通用标准，组装过程存在风险、构件可重用性和系统高效性不易协调；

构件质量直接影响产品质量。



### 快速应用开发模型（RAD）

增量型开发模型，

优点：

开发周期短

缺点：

需求分析的阶段时间较短、适用性一般、

适用范围：

适合于管理信息系统开发

不适合于技术风险较高、与外围系统的互动操作性较高的系统开发



**原型方法**

瀑布模型以及基于瀑布模型的软件生命周期模型，都需要精确的需求才能很好地执行后续的开发活动，但是准确的需求分析很难获取。

原型方法指在**获得基本需求后，快速分析，**实现一个小型的软件系统原型，满足用户的基本要求。用户可以提出修改意见，**校正需求**。

原型方法**主要用于明确需求，但也可以用于软件开发的其他阶段。**

原型算法的种类：

1、废弃策略

·探索型：弄清用户要求，确定期望特性；探讨多个方案的可行性。主要针对需求模糊、用户和开发者对项目都缺乏经验的情况。

·实验型：用于大规模开发和实现之前，考核技术方案是否合适、分析和设计的规格说明是否可靠。

2、追加策略

进化型：适应需求变化，不断改进原型，逐步将原型进化成最终系统。它将原型方法的思想扩展到软件开发的全过程，适用于需求经常变动的软件项目。

原型方法的优点：

有助于快速理解用户需求、容易确定系统性能和设计可行性、有利于建成最终系统。

原型方法的缺点：

文档容易被忽略、建立原型增加工作量、项目难以有效规划和管理。



# 三、新型软件生命周期模型

### RUP

面向对象的程序开发方法论，既是一个生命周期模型，也是一个支持面向对象软件开发的工具。

软件生命周期在时间上被分解为四个顺序的阶段：初始阶段、细化阶段、构造阶段、交付阶段。每个阶段在阶段结尾执行一次评估，确定阶段目标是否满足、是否可以进入下一个阶段。以用例为驱动，软件体系结构为核心，应用迭代及增量的新型软件生命周期模型。

各阶段工作说明

·**初始阶段**：了解业务，确定项目边界。包括验收规范、风险评估、资源估计、阶段计划制定等

·**细化阶段**：分析问题领域，建立软件体系结构基础，编制项目计划，完成技术要求高、风险大的关键需求开发。

·**构造阶段：**将所有剩余的技术构件和业务需求功能开发出来，集合成产品，所有功能被详细测试。

·**移交阶段**：重点是确保软件可被最终用户使用。交付阶段可以跨越几次迭代，包括为发布做准备的产品测试，基于用户反馈的少量调整。

每个迭代的主要活动：

核心过程工作流（6个）：业务建模、需求、分析和设计、实现、测试、部署

核心支持工作流（3个）：配置和变更管理、项目管理、环境

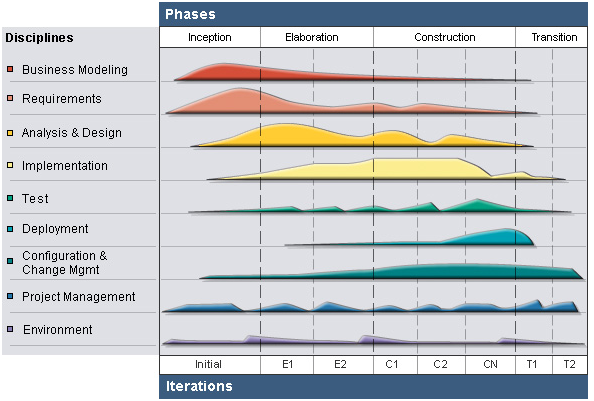
**模型特点**：

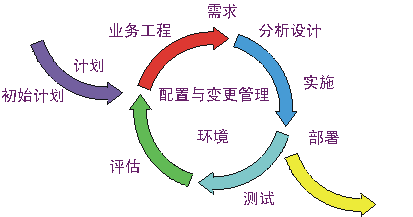
·适应性开发：小步骤、快速反馈和调整；

·使用用例驱动软件建模：用例是获取需求、制定计划、进行设计、测试、编写终端用户文档的驱动力量。

·可视化软件建模：使用UML进行软件建模。

·用例驱动，架构核心，面向对象





### 敏捷和极限编程（XP）

一种以实践为基础的软件工程过程和思想。使用**快速反馈**、**大量交流**，经过保证的测试来最大限度的满足用户的需求。主要强调用户满意，开发人员可以对需求的变化作出快速的反应。

每个参加项目开发的人都将担任一个角色（项目经理、项目监督人等等）并履行相应的权利和义务。

**特点：**

·测试驱动：XP内层的过程是一个基于Test Driven Development周期，每个开发周期都有很多相应的单元测试。

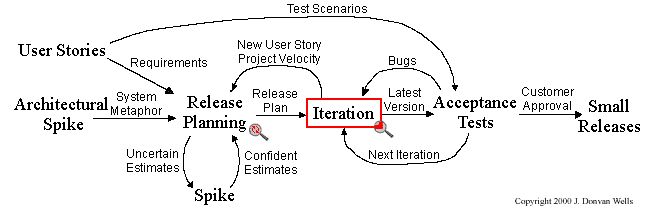
·大力提倡设计复核（Review）、代码复核以及重整和优化（Refectory），这些过程也是优化设计的过程；

·提倡配对编程（Pair Programming），而且代码所有权是归于整个开发队伍。

·程序员在写程序和重整优化程序时要严格遵守编程规范。

·任何人可以修改其他人写的程序，但修改后要确定新程序能通过单元测试。

·在开始写程序之前先写单元测试



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 模型 | 特点 | 优点 | 缺点 | 适用场景 |
| 瀑布模型 |  | 降低了软件开发过程的复杂性、提高开发过程的透明度和可管理性，强调需求分析和设计，阶段审核和文档输出保证了阶段之间的正确衔接，有利于及时发现问题 | 缺乏灵活性，风险控制能力低，文档驱动增加了系统的工作量，开发周期较长 | 需求明确、开发周期不紧张的较大型系统 |
| 演化模型 | 进行两次开发：试验开发+产品开发 | 明确用户需求，提高系统质量，降低风险 | 管理性差，开发结构性差， | 需求不清，开发周期短的中小型系统 |
| 增量模型  （体现了原型方法） | 优先开发核心需求 | 保障核心功能，开发风险低，提高系统稳定性和可维护性 | 增量粒度合理选择的难度大，确定所有需求难度大 | 需求不清，但是了解核心需求的系统 |
| 喷泉模型/迭代模型 | 阶段重复，可并行开发 | 提高效率，缩短开发周期 | 管理结构性差 |  |
| V模型和W模型 | 重视测试，将测试提升到和开发一样重要的地位，测试模块细分 | 提高开发效率，改善开发效果，W模型更能及时发现前期错误 | 不支持迭代开发 |  |
| 螺旋模型 | 开发过程分为四个阶段：风险评估、制定计划、实施工程、客户评估，每次评估之后确定是否进入下一回路 | 降低风险 | 开发周期长 | 风向高、开发周期较长的大型软件项目 |
| 构建组件模型 | 系统模块化，并进行组件的复用和组装 | 复用充分，提高效率，降低开发费用，提高可维护性 | 没有通用标准，可重用性和高效性存在不易协调的问题，构件质量直接影响产品质量 |  |
| 快速应用开发模型（RAD） | 增量开发。多组并行 | 开发周期短 | 需求分析时间段 | 管理信息的系统开发，风险低，和外围系统互动操作性低 |
| 原型方法 | 获得基本需求，快速分析，不断矫正 | 有利于快速理解用户需求，确定系统性能和设计的可行性 | 忽略文档，原型增加工作量，管理性降低 |  |
| RUP | 用例驱动，架构为核心，面向对象，可视化建模 |  |  |  |
| 敏捷和极限编程 | 快速反馈，大量交流，测试驱动，明确编程规范并且严格遵守 |  |  |  |

