

## 《软件开发过程与项目管理》

### Software Development Process and Project Management

任课教师： 范 国 祥

电 话： 0451-86418876-811(O)

13199561265(Mobile)

邮 箱： fgx@hit.edu.cn

哈工大计算学部

国家示范性软件学院

软件工程教研室

2023. 09

# 软件过程模型

## 1 软件过程

## 2 典型软件过程模型

### — 预测型过程模型

- 瀑布模型、V模型、形式化过程

### — 增量过程模型

- 增量模型、快速应用开发(RAD)

### — 迭代过程模型

- 螺旋模型、原型模型

### — 敏捷开发模型

- XP、Scrum

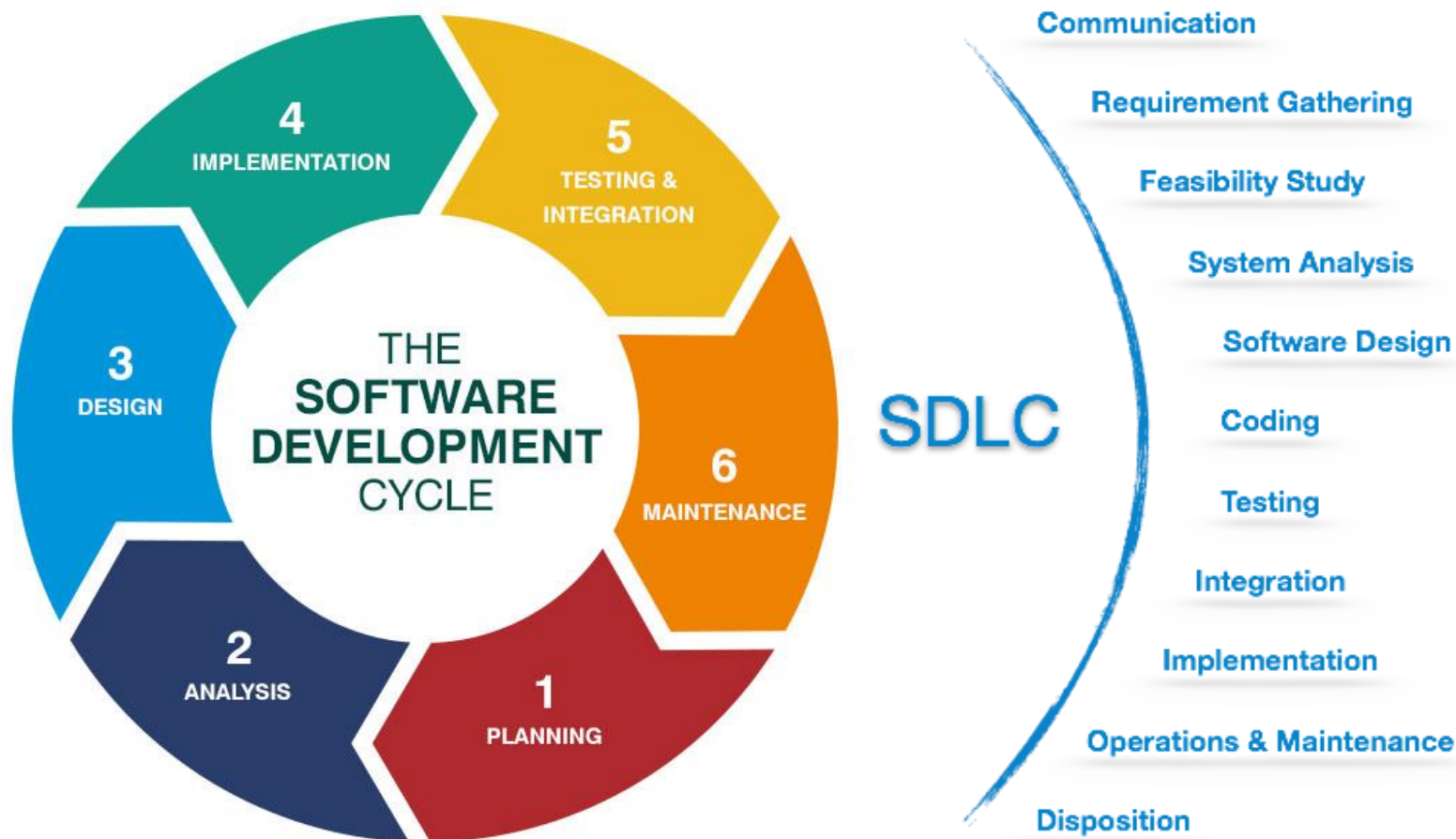
### — 其他过程模型

- 基于复用的过程模型

## 3 案例分析

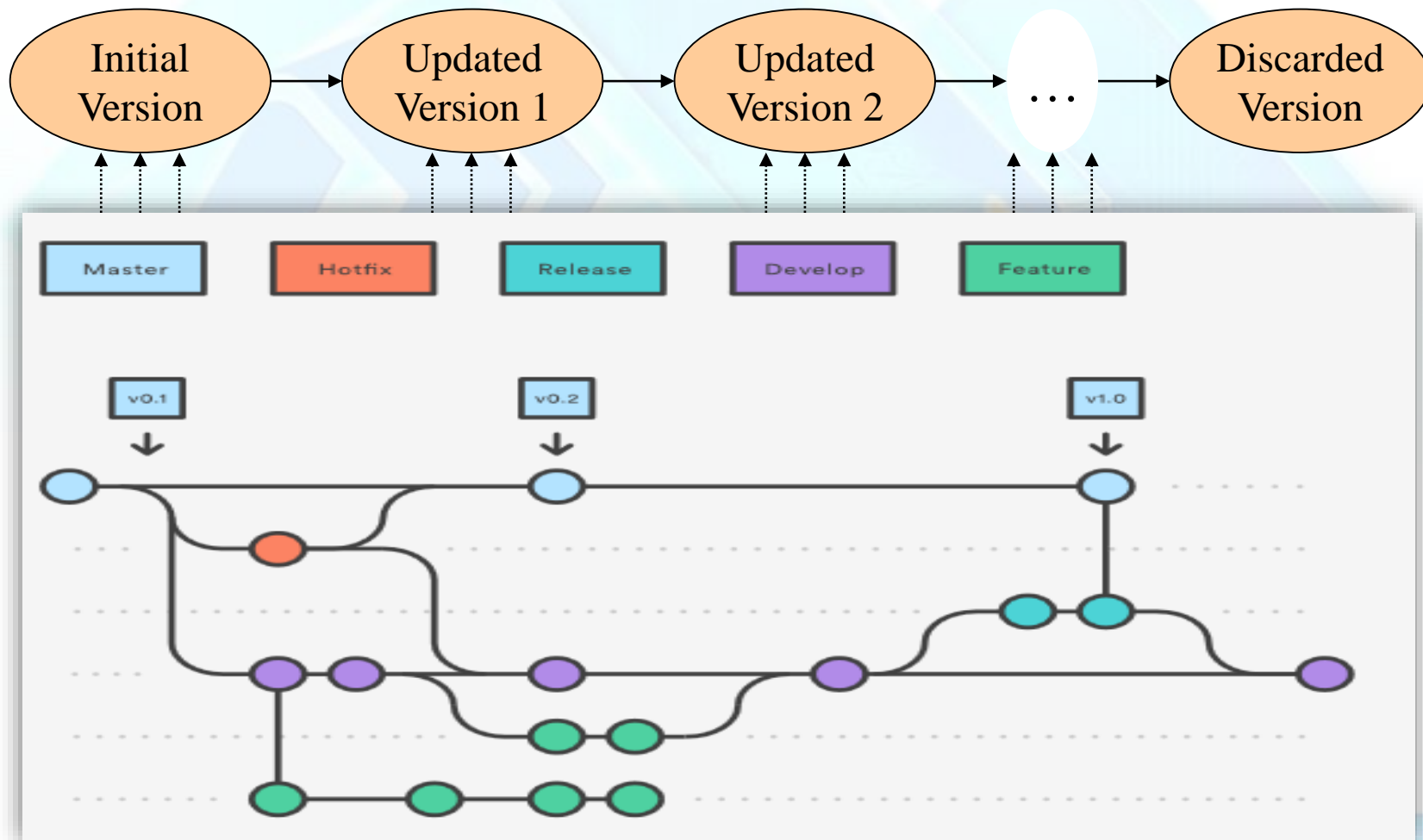
## 软件生命周期

- 软件开发生命周期（Software Development Life Cycle – SDLC）：  
**From 0 to 1**



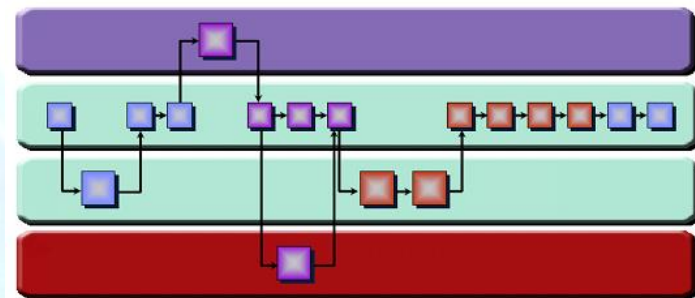
## 软件生命周期

- 软件开发生命周期（Software Development Life Cycle – SDLC）：  
Multiple versions in the life of a software: **From 1 to  $n$**



## 软件过程

- 软件过程定义以下内容：
  - 人员与分工
  - 所执行的活动
  - 活动的细节和步骤
- 软件过程通过以下方式组织和管理软件生命周期：
  - 定义软件生产过程中的活动
  - 定义这些活动的顺序及其关系
- 软件过程的目的：
  - 标准化(可模仿)、可预见性(降低风险)、提高开发效率、获得高质量产品
  - 提升制定时间和预算计划的能力

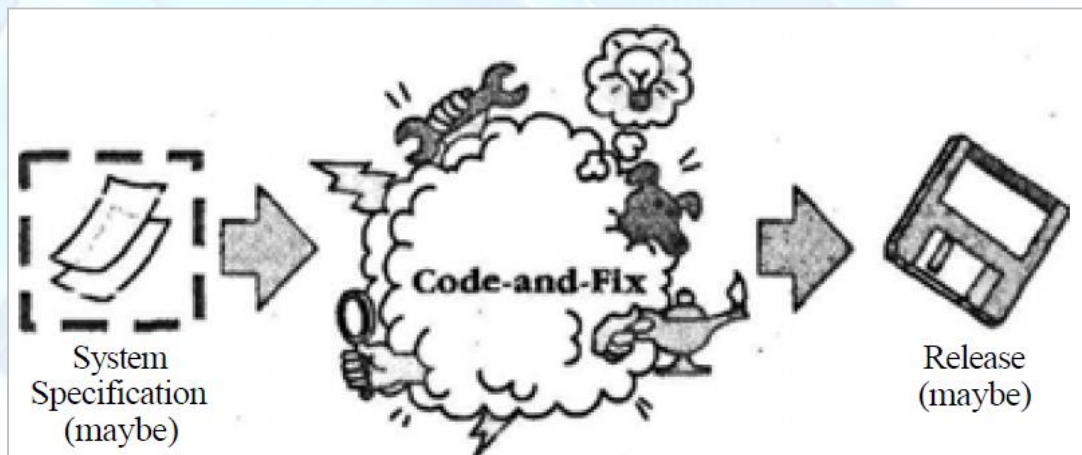




## 软件开发需要过程吗？

- 写了再改(Code-and-Fix), 不挺好吗?
  - 不需要太多其他准备或相关知识, 无需文档, 无需规划, 无需质量保障, 上来就写代码
  - 也许就能写出来, 写不出来就改, 也许能改好

- 适用场合:
  - “只用一次”的程序
  - “看过了就扔”的原型
  - 一些不实用的演示程序



- 但是要开发一个复杂的软件, 这个方法的缺点就太大了, 现实中基本上毫无用处!

From 《现代软件工程讲义》

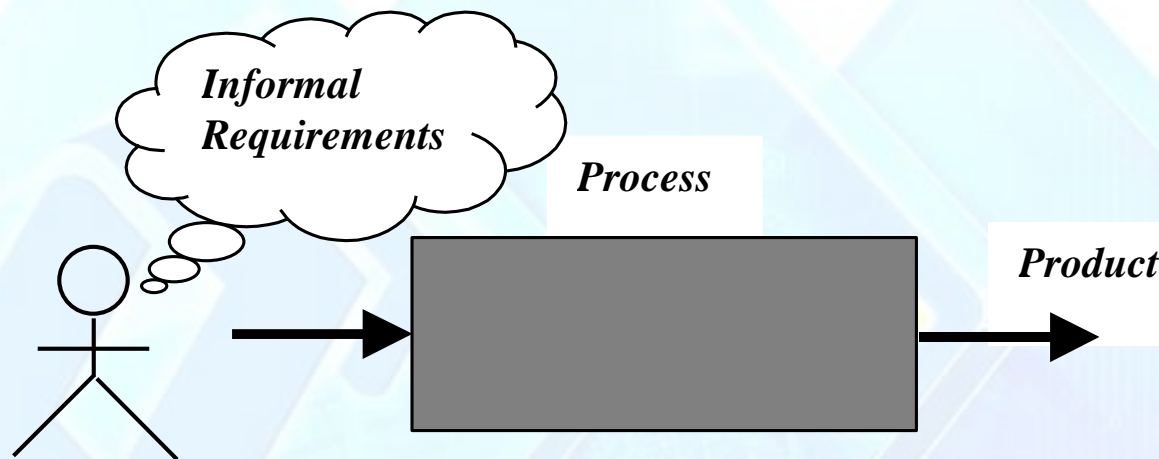
…构造一所房子…



相同的目标

不同的活动与过程

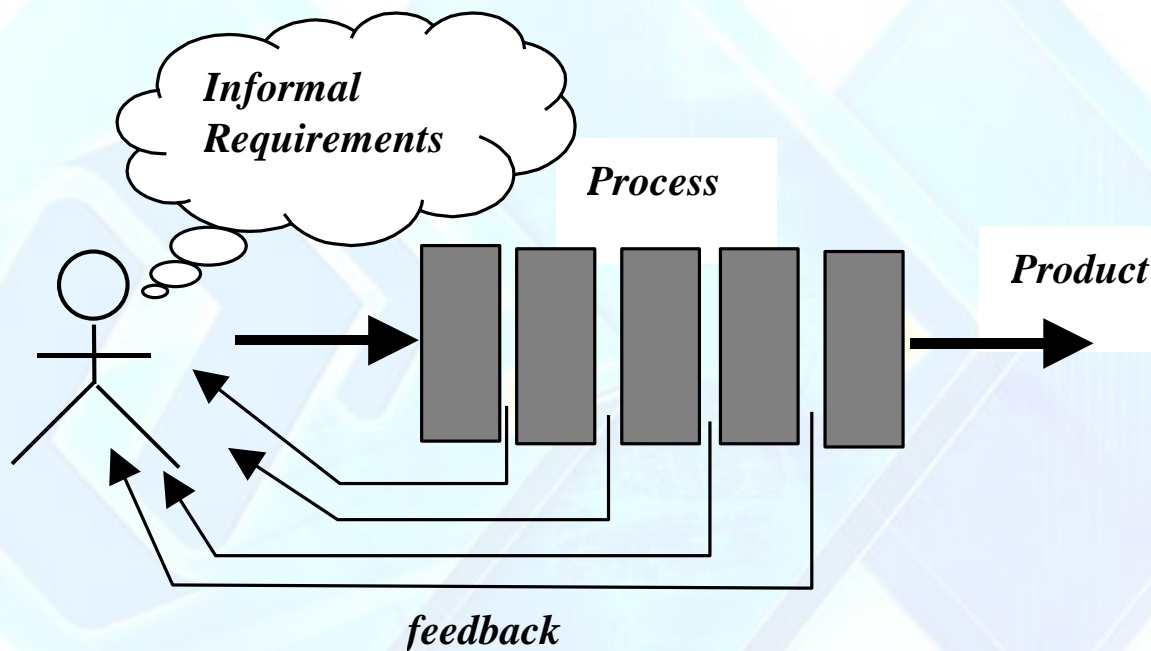
## 黑盒过程与白盒过程



- 存在的问题：
  - 要求开发之前需求被充分理解
  - 与客户的交互只在开始(需求)和最后(发布)——类似于产品制造过程
  - 而实际情况完全不是这样



## 黑盒过程与白盒过程

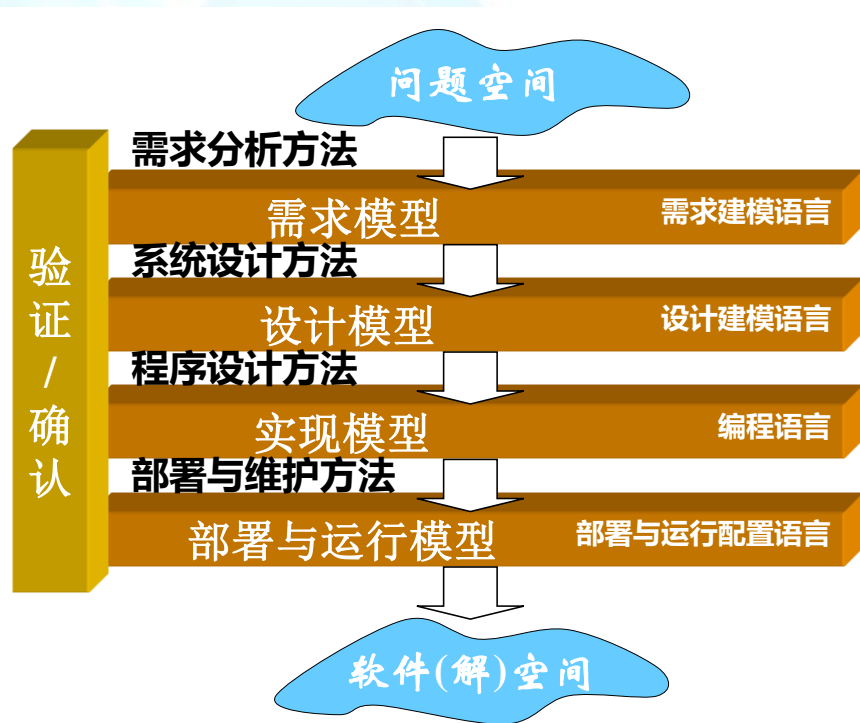


### ■ 优点:

- 通过改进可见性来减少风险
- 在开发过程中，通过不断地获得顾客的回馈允许变更——类似于服务过程

## 软件过程的典型阶段

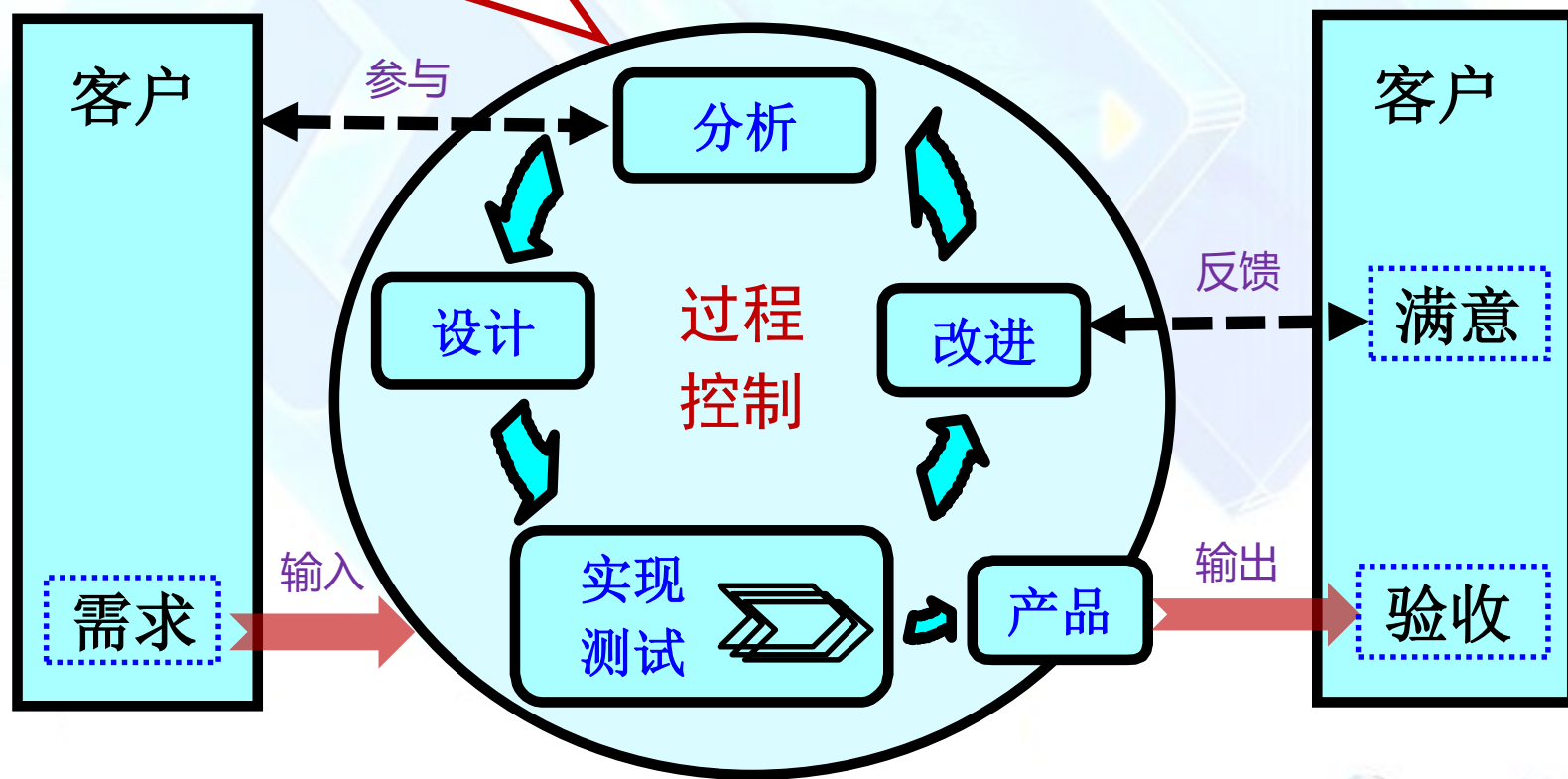
- Dream(提出设想)
- Investigation(深入调研)
- Software Specification(需求规格说明)
- Software Design(软件设计)
- Software Implementation(软件实现)
- Software Deployment (软件部署)
- Software Validation(软件验证)
- Software Evolution(软件演化)



## 软件过程的实质

在软件开发生命周期内采取特定的方式和策略进行过程管理控制

软件过程模型就是一种开发策略，这种策略针对软件工程的各个阶段提供了一套范形，使工程的进展达到预期的目的



# 软件过程模型

## 1 软件过程

## 2 典型软件过程模型

### — 预测型过程模型

- 瀑布模型、V模型、形式化过程

### — 增量过程模型

- 增量模型、快速应用开发(RAD)

### — 迭代过程模型

- 螺旋模型、原型模型

### — 敏捷开发模型

- XP、Scrum

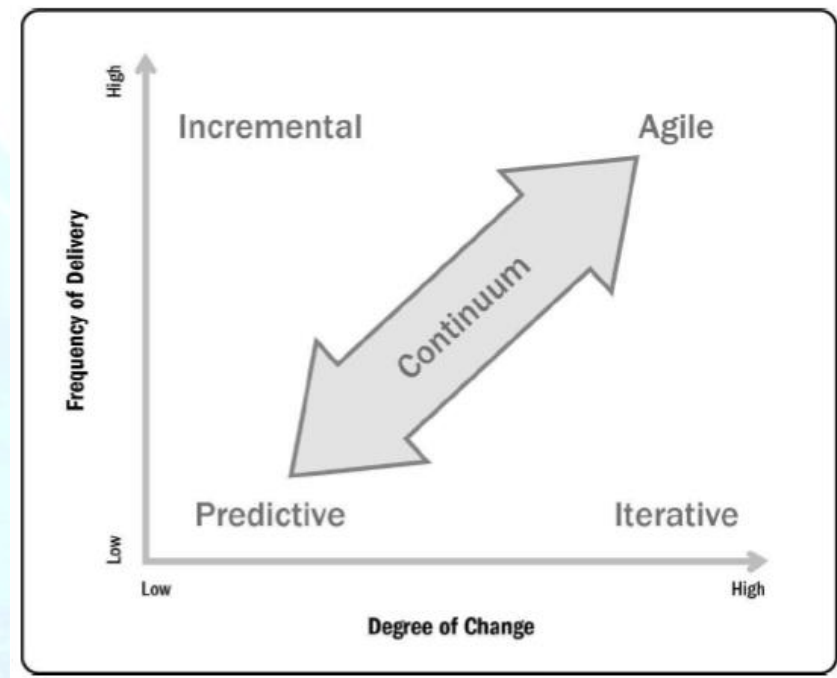
### — 其他过程模型

- 基于复用的过程模型

## 3 案例分析

## 软件过程模型分类

- 预测型(Predictive): 提前进行大量的计划工作, 然后一次性执行; 执行是一个连续的过程。
- 迭代型(Iterative): 允许对未完成的工作进行反馈, 从而改进和修改该工作。
- 增量型(Incremental): 分次向客户提供已完成的, 可以立即使用的可交付成果。
- 敏捷型(Agile): 既有迭代, 也有增量, 便于完善工作, 频繁交付。



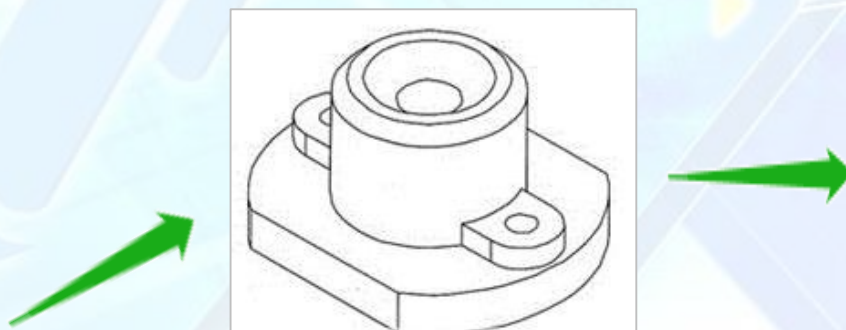
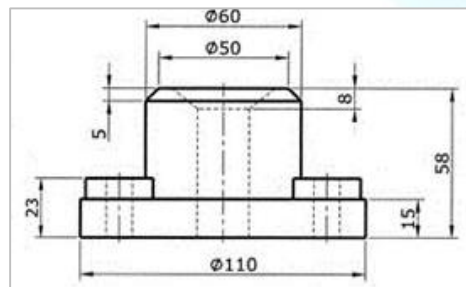
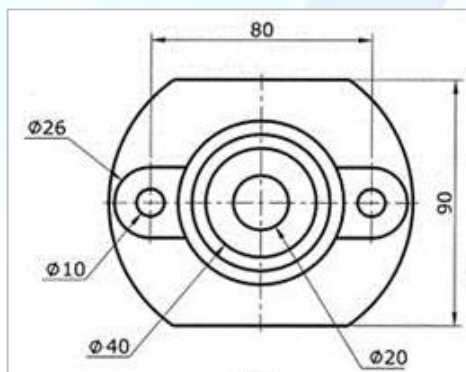
Characteristics				
Approach	Requirements	Activities	Delivery	Goal
Predictive	Fixed	Performed once for the entire project	Single delivery	Manage cost
Iterative	Dynamic	Repeated until correct	Single delivery	Correctness of solution
Incremental	Dynamic	Performed once for a given increment	Frequent smaller deliveries	Speed
Agile	Dynamic	Repeated until correct	Frequent small deliveries	Customer value via frequent deliveries and feedback



## 典型的软件过程模型

### ■ 预测型过程模型(Predictive model)

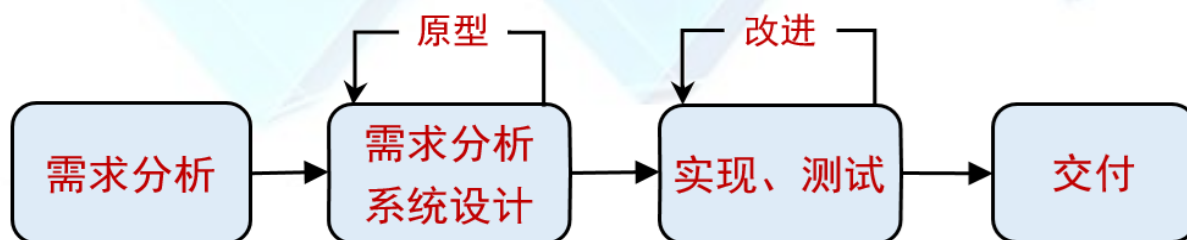
- 项目开发活动通常按照固定顺序执行，前一步活动结果是后一步活动的“蓝图”，前一步对后一步结果的预期约束十分精准
- 需要提前进行大量的计划工作，然后一次性执行，执行过程是一个连续的过程



## 典型的软件过程模型

### ■ 迭代过程模型 (Iterative model)

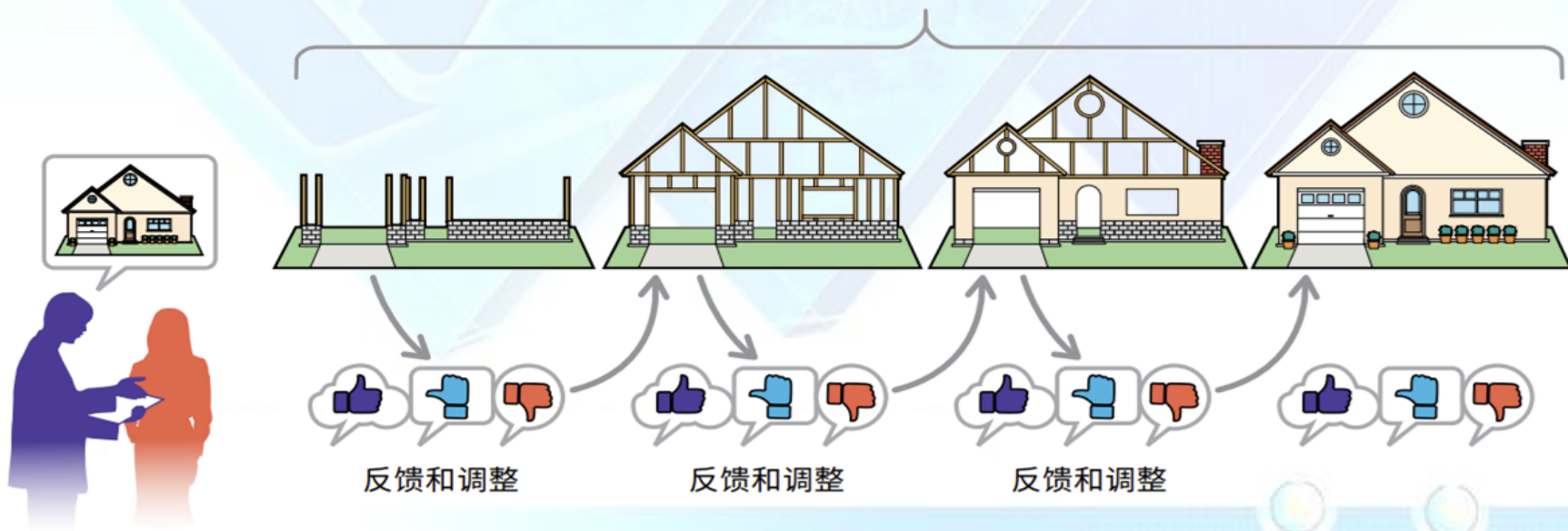
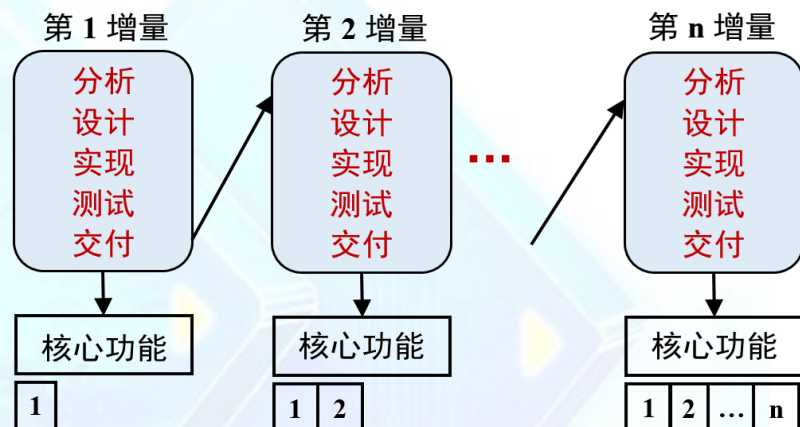
- 允许对未完成的工作进行反馈，可以进行修改和改进
- 允许对部分已经完成的工作进行反馈，从而进行修改和改进
- 逐步反馈、改进完善，直至开发完成



## 典型的软件过程模型

### ■ 增量过程模型 (Incremental process model)

- 将未来系统分阶段完成，每个阶段都会产生一个可交付成果
- 每个阶段成果都是一个增量
- 每个增量都是一个独立的开发过程，包括分析、设计、实现、测试、交付

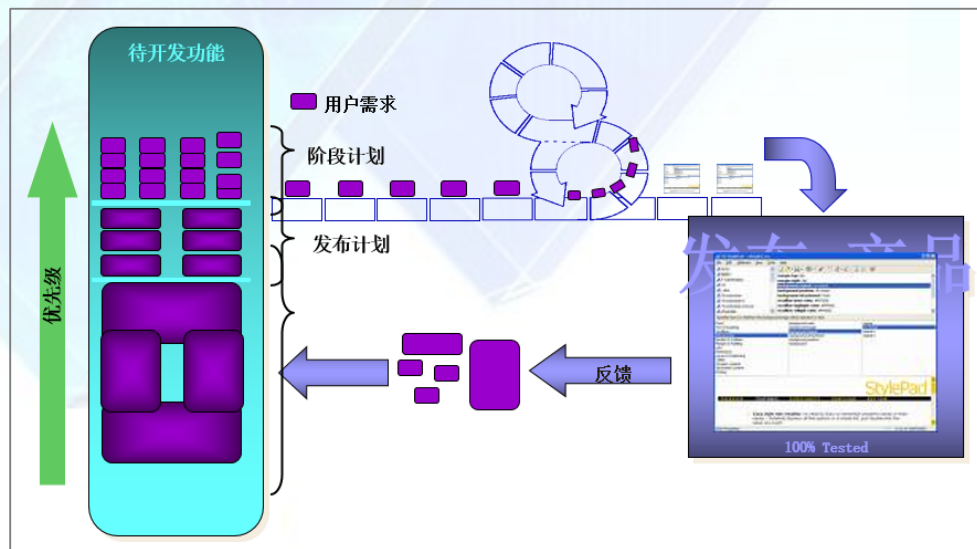




## 典型的软件过程模型

### ■ 敏捷过程模型(Agile model)

- 应对变化、增量开发、快速反馈、快速迭代、快速交付
- 基于迭代的敏捷过程：每次迭代，都是针对最重要的系统功能，团队合作开发
- 基于流程的敏捷过程：不是按照迭代计划，而是根据团队能力，领取任务，恒速开发



## 典型的软件过程模型

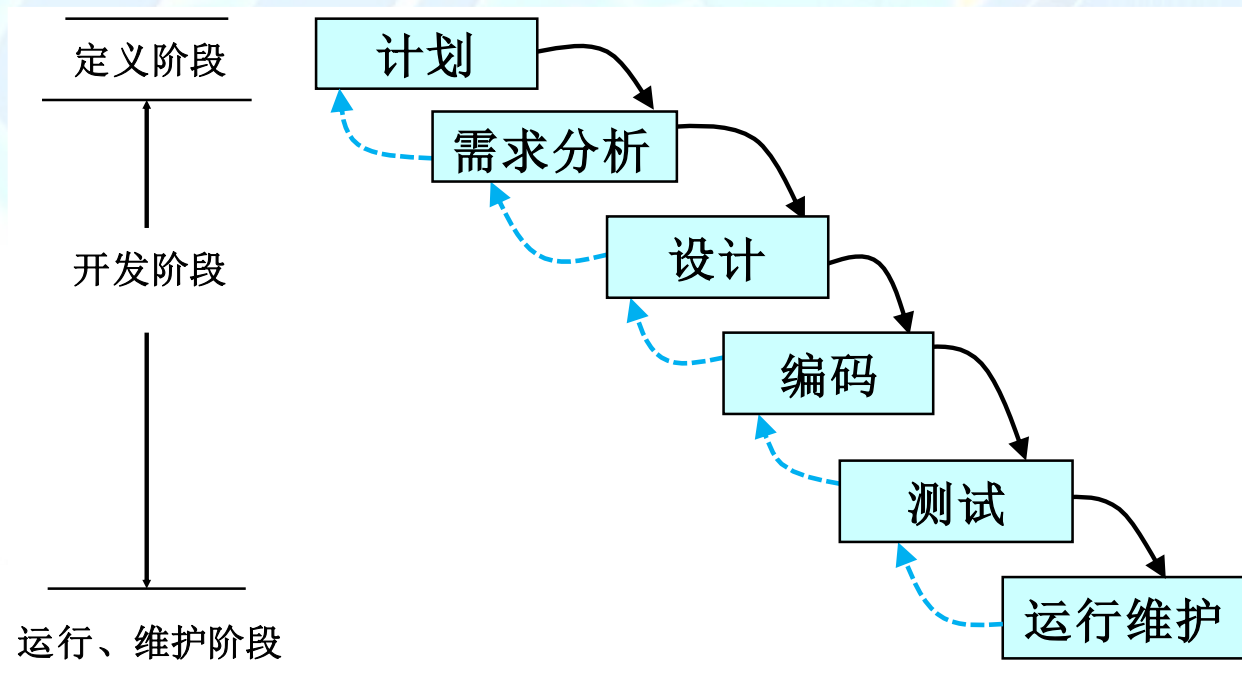
- 预测型过程模型(Predictive model)
  - 瀑布模型 (Waterfall)、V模型(V Model)、W模型 (W Model) 形式化过程 (Formal model)
- 迭代过程模型 (Iterative model)
  - 螺旋模型 (Spiral)、原型模型 (Prototype)
- 增量过程模型 (Incremental process model)
  - 增量模型 (Incremental)、快速应用开发(RAD)
- 敏捷过程模型(Agile model)
  - XP、Scrum、Kanban、DevOps等



## 瀑布模型

- 1970年, Winston Royce提出
- 将SDLC各项活动规定为按**固定顺序而连接**的若干阶段工作, 形如瀑布流水, 最终得到软件产品
- 自诞生之日起至80年代初期, 瀑布模型是唯一被广泛采用的软件开发模型

- 上一个阶段结束, 下一个阶段才能开始
- 每个阶段均有里程碑和提交物
- 上一阶段输出是下一阶段输入
- 每个阶段均需要进行V&V
- 侧重于文档与产出物



## 瀑布模型

- 也叫做鲑鱼模型(Salmon model): 向前一阶段回溯, 很难!



- i 你(用户) 提出要发动机, 车身, 车窗, 方向盘, 加速踏板, 刹车, 手刹, 座位, 车灯...
- i 生产商按照瀑布模型流程给你设计, 生产, 六个月后交付。
- i 看到样车后...
- i 你提出 - 我当初忘了一件小事, 要有倒车灯
  - 当倒车的时候, 倒车灯会亮
- i 生产商说:
  - 我要重新设计车尾部, 加上倒车灯, 把车底拆开, 安装线路, 修改传动装置把倒车档和倒车灯联系起来。。。我得重新开始
- i 你说: 这不是很小的一件事么?

这是小事还是大事?

## 瀑布模型

### ■ 优点——追求效率

- 简单、易懂、易用、快速
- 为项目提供了按阶段划分的检查点，项目管理比较容易
- 每个阶段必须提供文档，而且要求每个阶段的所有产品必须进行正式、严格的技术审查

### ■ 缺点——过于理想化

- 在开发早期，用户难以清楚地确定所有需求，需求的错误很难在开发后期纠正，因此难以快速响应用户需求变更
- 开发人员与用户之间缺乏有效的沟通，开发人员的工作几乎完全依赖规格说明文档，容易导致不能满足客户需求
- 客户必须在项目接近尾声的时候才能得到可执行的程序，对系统中存在的重大缺陷，如果在评审之前没有被发现，将可能会造成重大损失

## 瀑布模型

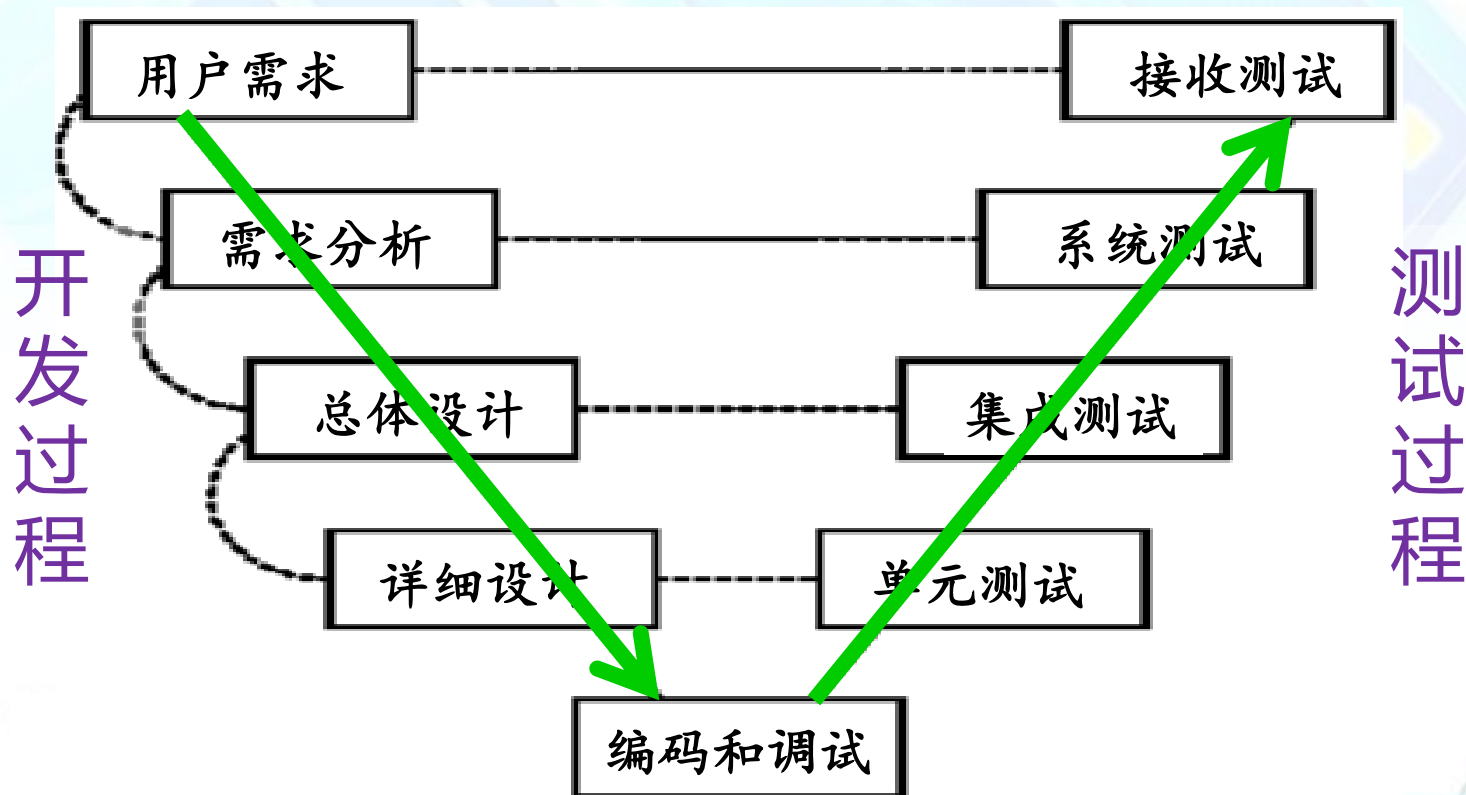
- 瀑布模型太理想化，太单纯，已不再适合现代的软件开发模式，在大型系统开发中已经很少使用
- 适用场合：
  - 软件项目较小，各模块间接口定义非常清晰
  - 需求在项目开始之前已经被全面的了解，产品的定义非常稳定
  - 需求在开发中不太可能发生重大改变
  - 使用的技术非常成熟，团队成员都很熟悉这些技术
  - 负责各个步骤的子团队分属不同的机构或不同的地理位置，不可能做到频繁的交流
  - 外部环境的不可控因素很少





## V模型

- V模型是由Paul Rook在1980年提出的
- V模型是瀑布模型的一种变形
- V模型**强调测试**的重要性，将开发活动与测试活动紧密联系在一起





## W模型

- W模型，由Evolutif公司提出，相对于V模型，W模型增加了软件开发各阶段中同步进行的验证和确认活动
- 由两个V字型模型组成，分别代表测试与开发过程，明确表示出了测试与开发的并行关系
- W模型同样**强调测试**的重要性，将开发活动与测试活动紧密联系在一起



## V模型与W模型的优缺点

### ■ 优点——开发过程重视测试/验证

- 简单易用，只要按照规定的步骤执行即可
- 强调测试或验证与开发过程的对应性和并行性；测试方案在编码之前就已经制定了
- 与瀑布模型相比，项目开发成功的机会更高
- 避免缺陷向下游流动

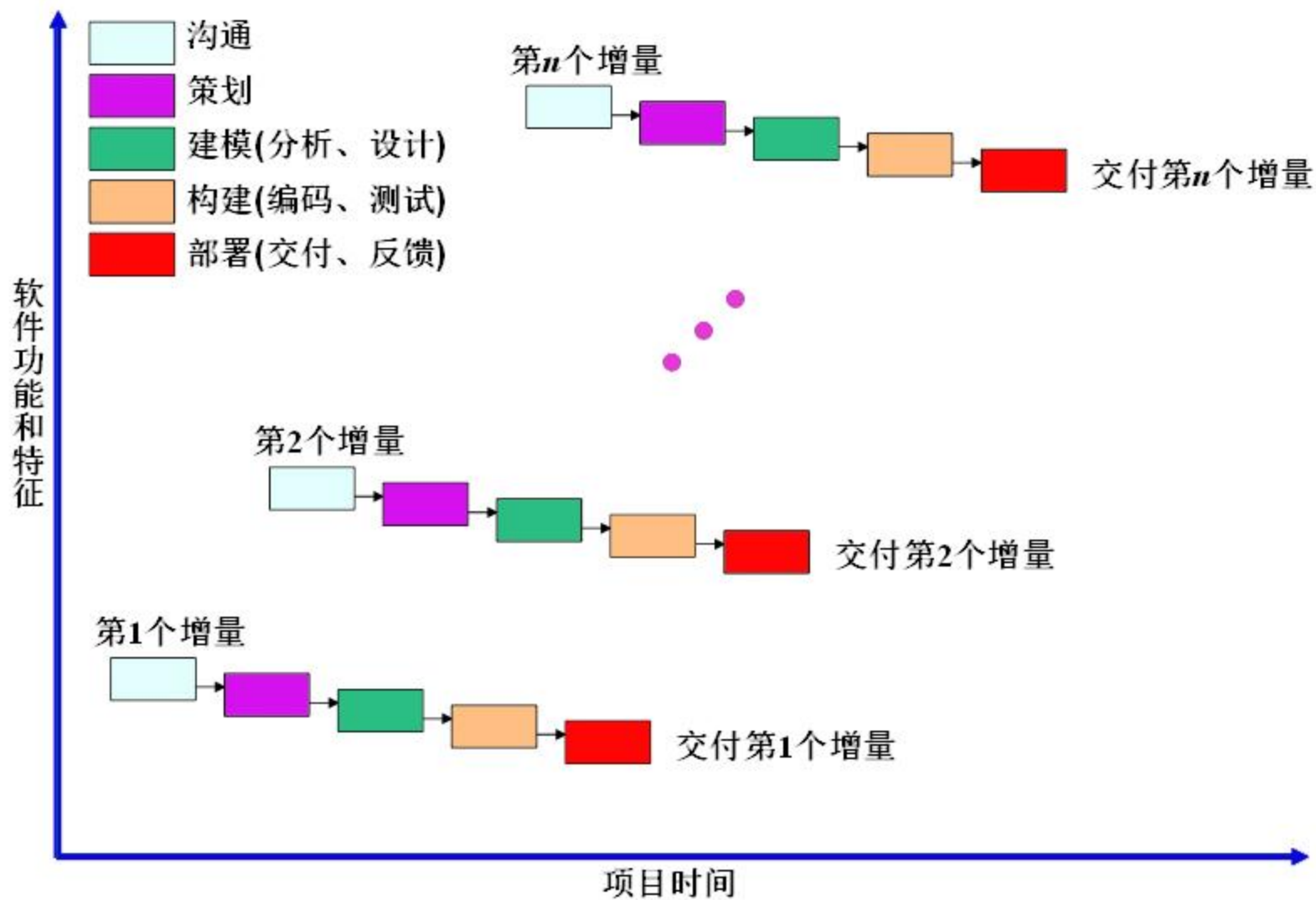
### ■ 缺点

- 比瀑布模型繁琐

## 增量过程模型

- 在很多情况下，由于初始需求的不明确，开发过程不宜采用瀑布模型等预测性模型
- 因此，无须等到所有需求都出来才进行开发，只要某个需求的核心部分确定下来，即可进行开发
- 另外，可能迫切需要为用户迅速提供一套功能有限的软件产品，然后在后续版本中再细化和扩展功能
- 在这种情况下，需要选用**增量方式的软件过程模型**
  - 增量模型
  - RAD模型

## 增量模型



## 增量模型

- 软件被作为一系列的增量来设计、实现、集成和测试，每一个增量是由多个相互作用的模块所形成的特定功能模块或功能模块组
- 本质：以迭代的方式运用瀑布模型
  - 第一个增量往往是核心产品：满足了基本的需求，但是缺少附加的特性
  - 客户使用上一个增量的提交物并进行自我评价，制定下一个增量计划，说明需要增加的特性和功能
  - 重复上述过程，直到最终产品开发完成为止



## 增量模型

- 举例1：开发一个类似于Word的字处理软件
  - 增量1：提供基本的文件管理、编辑和文档生成功能
  - 增量2：提供高级的文档编辑功能
  - 增量3：实现拼写和语法检查功能
  - 增量4：完成高级的页面排版功能
- 举例2：开发一个教务管理系统
  - 增量1：提供基本的学籍管理和成绩管理功能
  - 增量2：提供选课功能
  - 增量3：提供查询教室使用情况的功能
  - 增量4：提供课表生成、上课名单生成、成绩录入等功能

## 增量模型

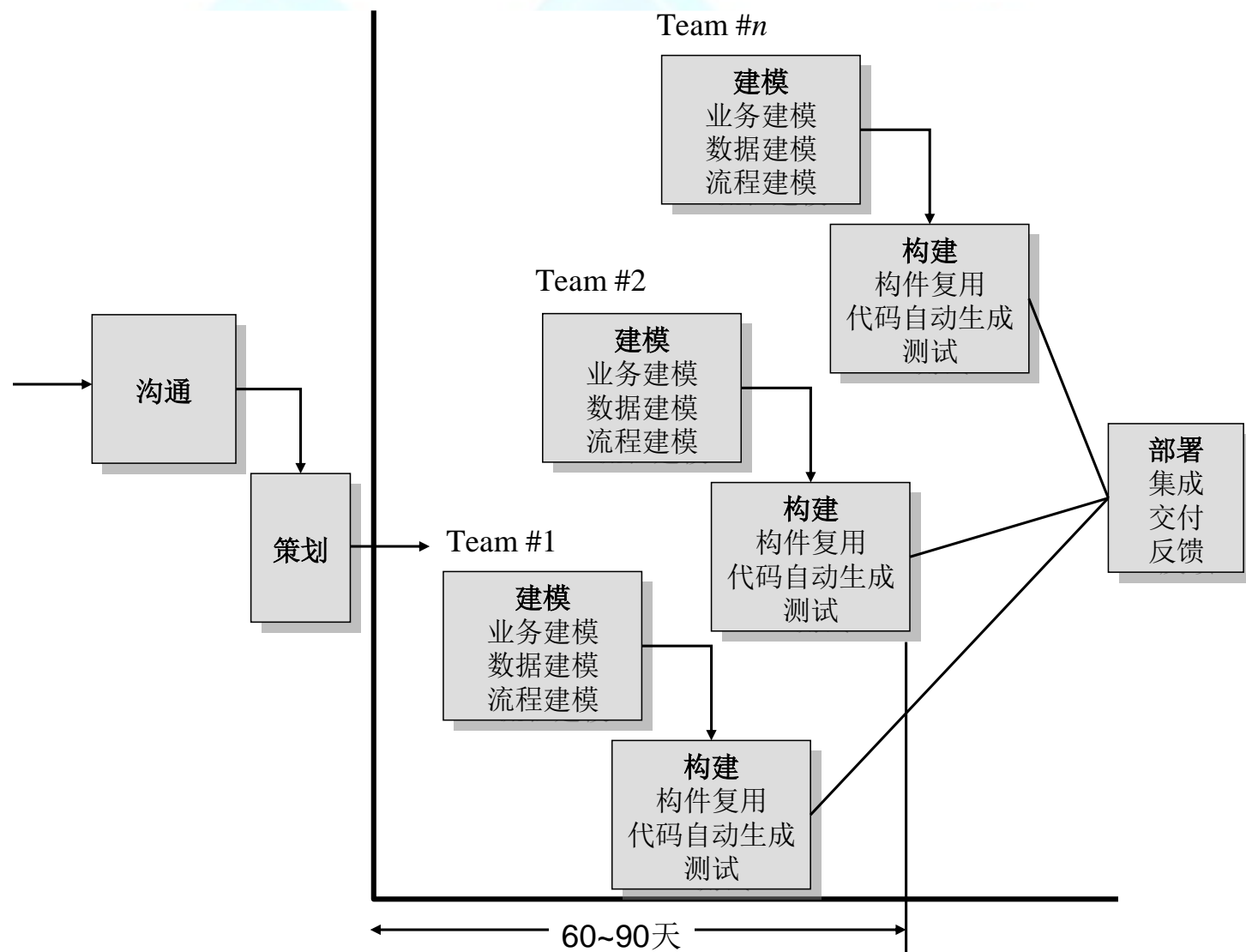
### ■ 优点：

- 在时间要求较高的情况下交付产品：在各个阶段并不交付一个可运行的完整产品，而是交付满足客户需求的一个子集的可运行产品，对客户起到“镇静剂”的作用
- 人员分配灵活：如果找不到足够的开发人员，可采用增量模型：早期的增量由少量人员实现，如果客户反馈较好，则在下一个增量中投入更多的人力
- 逐步增加产品功能可以使用户有较充裕的时间来学习和适应新产品，避免全新软件可能带来的冲击
- 因为具有较高优先权的模块被首先交付，而后面的增量也不断被集成进来，这使得最重要的功能肯定接受了最多的测试，从而使得项目总体性失败的风险比较低

## 增量模型

- 困难：
  - 每个附加的增量并入现有软件时，**必须不破坏原来已构造好的部分**
  - 同时，**加入新增量时应简单、方便** —— 该类软件的体系结构应当是开放的
  - 仍然**无法处理需求发生变更的情况**
  - 管理人员必须有足够的技术能力来**协调好各增量之间的关系**

## RAD模型 (Rapid Application Development)



## RAD 模型

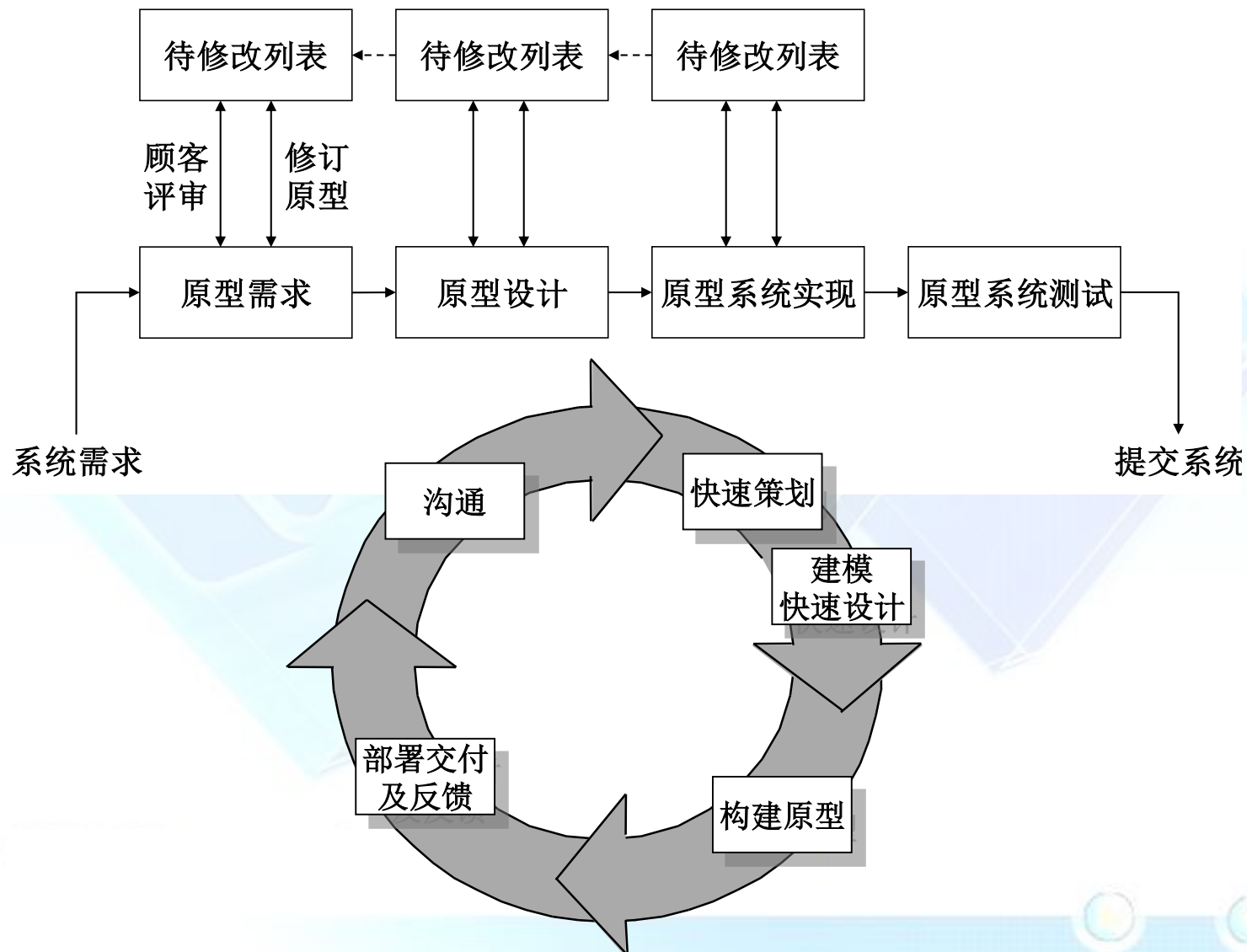
- 快速应用开发RAD (Rapid Application Development)
  - 侧重于短开发周期（一般为60~90天）的增量过程模型，是瀑布模型的高速变体，通过基于构件的构建方法实现快速开发
  - 多个团队并行进行开发，但启动时间有先后，先启动团队的提交物将作为后启动团队的输入
- 缺点：
  - 需要大量的人力资源来创建多个相对独立的RAD团队
  - 如果没有在短时间内为急速完成整个系统做好准备，RAD项目将会失败
  - 如果系统不能被合理的模块化，RAD将会带来很多问题
  - 技术风险很高的情况下（采用很多新技术、软件需与其他已有软件建立集成等等），不宜采用RAD



## 迭代过程模型

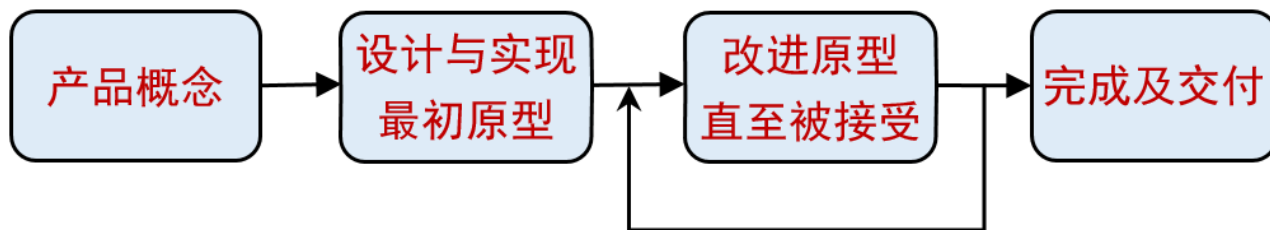
- 软件开发过程面临的客观情况：
  - 软件系统会随着时间的推移而发生变化，在开发过程中，需求经常发生变化，直接导致产品难以实现
  - 严格的交付时间使得开发团队不可能圆满完成软件产品，但是必须交付功能有限的版本以应对竞争或压力
  - 很好的理解核心产品与系统需求，但对其他扩展的细节问题却没有定义
- 在上述情况下，需要一种专门应对不断演变的软件过程模型，即“迭代过程模型”
- 本质：循环、反复、不断调整当前系统以适应需求变化
- 主要包括两种形态：
  - 快速原型法
  - 螺旋模型

## 快速原型法



## 快速原型法的步骤

- Step 1:** 双方通过沟通，明确已知的需求，并大致勾画出以后再进一步定义的东西
- Step 2:** 迅速策划一个原型开发迭代并进行建模，主要集中于那些**最终用户所能够看到的内容**，如人机界面布局或者输出显示格式等
- Step 3:** 快速设计产生原型，对原型进行部署，由客户和用户进行评价
- Step 4:** 根据反馈，进一步细化需求并调整原型
- Step 5:** 原型系统不断调整以逼近用户需求



## “原型”的类型

- **Throwaway prototyping(抛弃式原型)** 【例子】
  - 最初的原型在完成并得到用户认可之后，将不会作为交付给用户的最终系统的一部分，而是被抛弃，其目的只是为了收集与验证需求
  - 该类原型可能是不可执行的(例如，只包含用户界面)
- **Evolutionary prototyping(演化式原型)** 【例子1】 【例子2】 【例子3】
  - 最初构造的原型将具备较高的质量，包含了系统的核心功能，然后通过收集需求对其进行不断的改善和精化
  - 该类原型是可执行的，将成为最终系统的一部分

## 快速原型法的优缺点

### ■ 优点：

- 提高和改善客户/用户的参与程度，最大程度的响应用户需求的变化
- 克服预测型模型的缺点，减少由于需求不够明确带来的开发风险

### ■ 缺点：

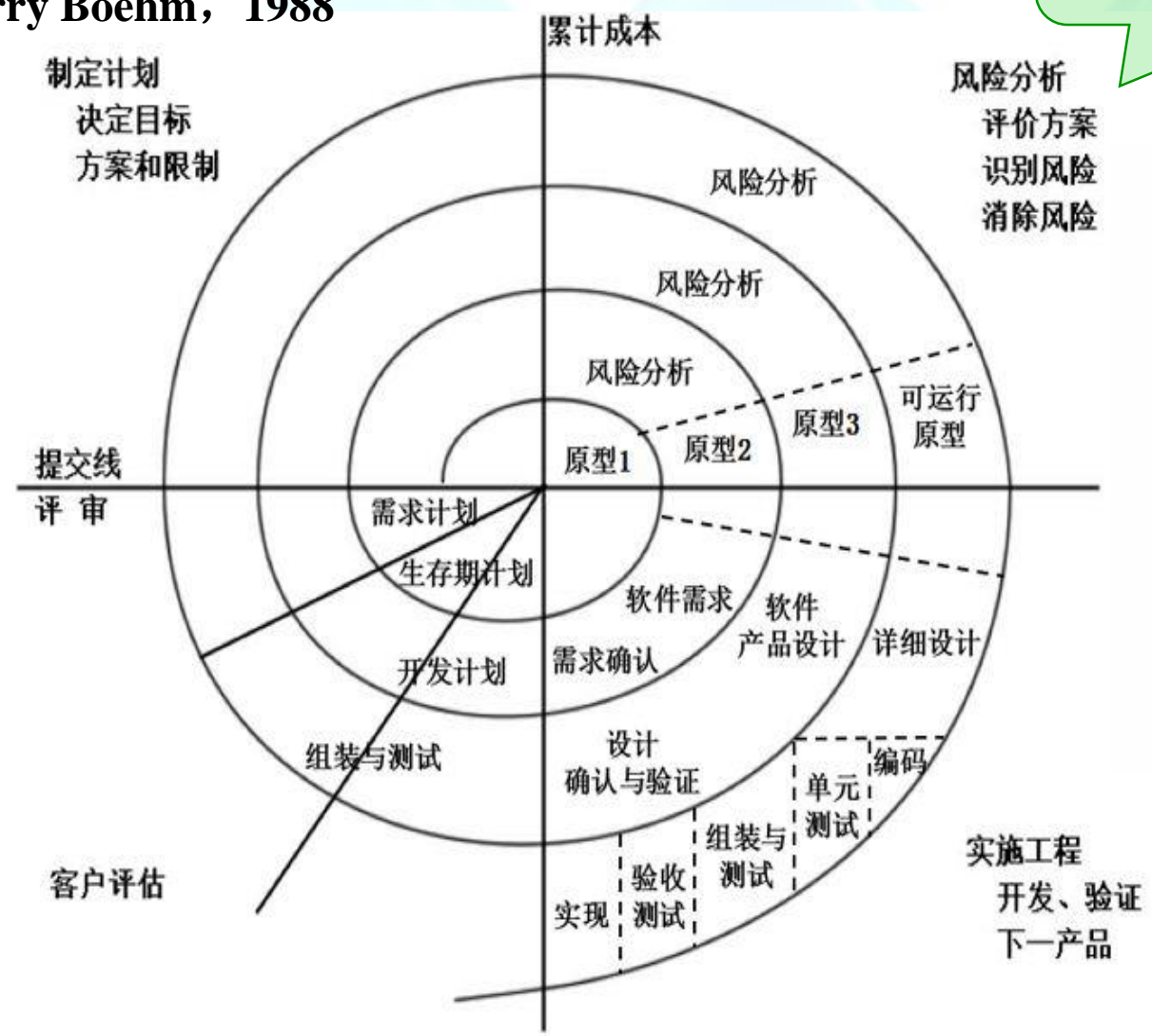
- 为了尽快完成原型，开发者没有考虑整体软件的质量和长期的可维护性，系统结构通常较差
- 可能混淆原型系统与最终系统，原型系统在完全满足用户需求之后可能会被直接交付给客户使用
- 额外的开发费用



# 螺旋式过程模型

Barry Boehm, 1988

与增量、RAD等的最大区别在于重视风险评估



## 螺旋式过程模型

- 螺旋模型沿着螺旋线旋转，在四个象限内表达四个方面的活动：
  - **制定计划**：确定软件目标，选定实施方案，弄清项目开发的限制
  - **风险分析**：分析所选方案，考虑如何识别和消除风险
  - **实施工程**：实施软件开发
  - **客户评估**：评价开发工作，提出修正建议
- 举例：
  - **第1圈**：开发出产品的规格说明
  - **第2圈**：开发产品的原型系统
  - **第3~n圈**：不断的迭代，开发不同的软件版本
  - 根据每圈交付后用户的反馈来调整预算、进度、需要迭代的次数

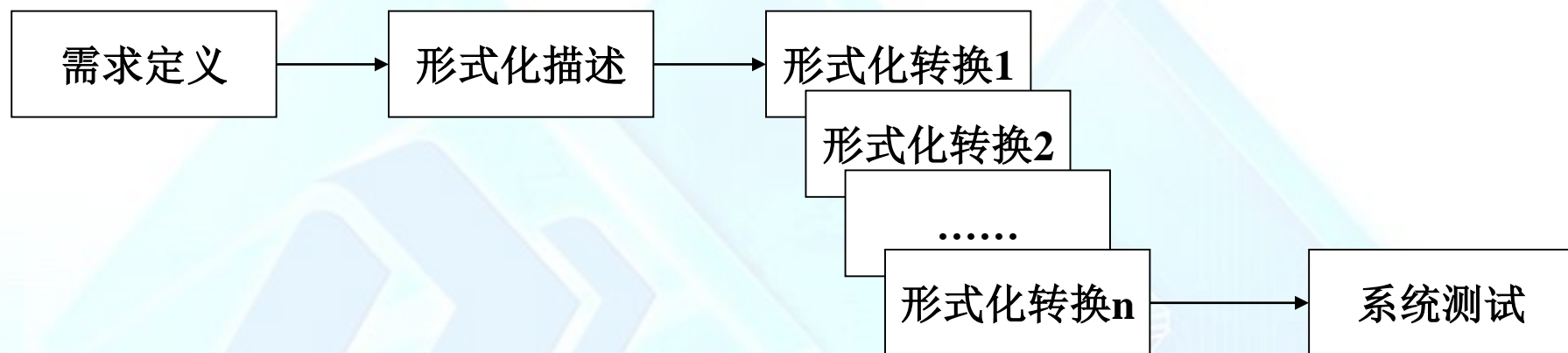
## 螺旋式过程模型

- 出发点：开发过程中及时识别和分析风险，并采取适当措施以消除或减少风险带来的危害
- 优点：结合了原型的迭代性质与瀑布模型的系统性和可控性，是一种风险驱动型的过程模型：
  - 采用循环的方式逐步加深系统定义和实现的深度，同时更好的理解、应对和降低风险
  - 确定一系列里程碑，确保各方都得到可行的系统解决方案
  - 始终保持可操作性，直到软件生命周期的结束
  - 由风险驱动，支持现有软件的复用
- 缺陷：
  - 适用于大规模软件项目，特别是内部项目，周期长、成本高
  - 软件开发人员应该擅长寻找可能的风险，准确的分析风险，否则将会带来更大的风险

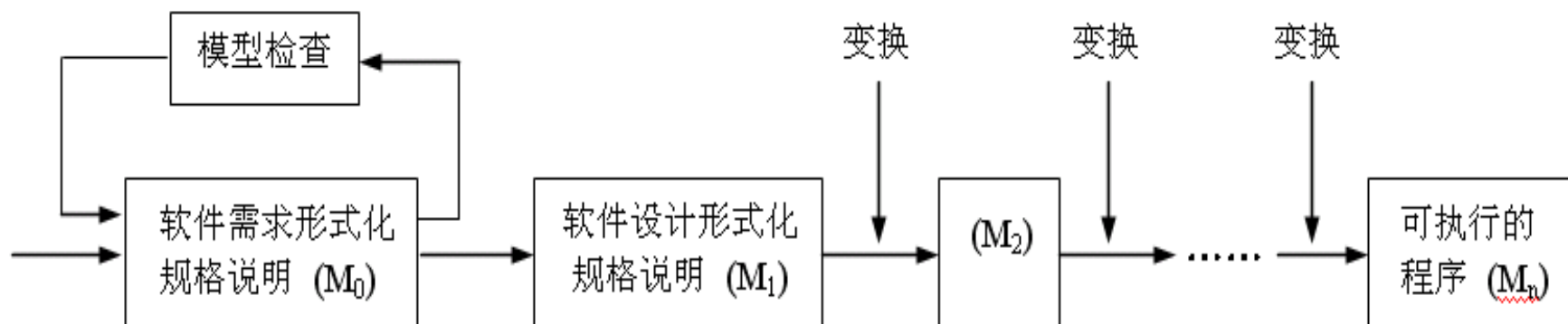
## 总结：迭代过程模型的缺点

- 迭代过程模型的目的：
  - 需求的变更频繁，要求在非常短的期限内实现，以充分满足客户/用户要求、及时投入市场
- 存在的问题：
  - 由于构建产品所需的周期数不确定，给项目管理带来困难
  - 迭代速度太快，项目陷入混乱；迭代速度太慢，影响生产率
  - 为追求软件的高质量而牺牲了开发速度、灵活性和可扩展性

## 形式化过程模型



- 使用严格的数学形式来刻画每一阶段的产物(需求、设计、程序、测试)
- 应用一系列形式化方法在各阶段的产物之间进行自动/半自动的转换





## 形式化过程模型

### ■ 优点：

- 应用数学分析方法，歧义性、不完整性、不一致性等问题更容易被发现和改正，目的是“提供无缺陷的软件”

### ■ 缺陷：

- 形式化数学方法难以理解，可视性太差，对开发人员技能要求较高
- 构造形式化模型是一件非常耗时的工作，成本也很高
- 软件系统中的某些方面难以用形式化模型表达出来（如用户界面）

### ■ 应用场合：

- 对可靠性和安全性要求较高的一些关键系统，在真正被投入使用之前，需要严格保证100%的正确；传统的方法靠人去验证，难以奏效
- 太过于理想化，因此仅停留在理论研究中，实践中很少使用

## 面向复用的软件过程

- 该过程模型的主要思想是**复用(Reuse)**
- 针对一个新的软件系统，不是完全从一无所有开始入手，而是通过使用已有的软件单元（称为“软构件”）来构造系统
- 主要过程：
  - 需求分析
  - 体系结构设计
  - 构件获取(购买 / 重新开发)
  - 构件修改与测试
  - 构件组装
  - 集成测试

## 软件过程模型

### 1 软件过程

### 2 典型软件过程模型

#### — 预测型过程模型

- 瀑布模型、V模型、形式化过程

#### — 增量过程模型

- 增量模型、快速应用开发(RAD)

#### — 迭代过程模型

- 螺旋模型、原型模型

#### — 敏捷开发模型

- XP、Scrum

#### — 其他过程模型

- 基于复用的过程模型

### 3 案例分析

## 一个案例

- 某个老师T想要考察一个同学S的学习情况和技术水平，于是交给该学生一个任务...
  - **T:** 我有一个朋友想要一个图象浏览软件，能够查看多种格式的图像，包括BMP、TIFF、JPG、PNG，并且能够支持一般的放大、缩小、漫游。你能做这样一个软件吗？
  - **S:** 就是类似ACDSEE这样的软件吗？
  - **T:** 差不多，不过不需要那么强大的功能，我这个朋友计算机是外行，最好能做的比较方便，傻瓜型的，例如象ACDSEE自动翻页这种功能还是要的。
  - **S:** 我以前学过BMP和JPG的图象格式解析，我想没有问题。
  - **T:** 好的，给你30天时间，下周你再来一趟，跟我讲一下你的工作进度。
- 这位同学非常明白老师的意图，回去后想了一下，并列出了一个清单

## 工作清单

### 一 功能:

1. 读取、显示、另存四种格式图片( BMP、TIFF、JPG、PNG )
2. 放大、缩小、漫游
3. 列出当前目录下所有四种格式图片文件名
4. PageUp(PageDown)自动调出当前目录上一张(下一张)图片

### 二 其它说明:

1. 界面尽量简介, 容易操作
2. 不要图片预览和打印

### 三 开发工具: VC6

### 四 开发环境: PC, Windows

### 五 工作量:

1. 研究一下4种图片的格式
2. 设计一个解析器类, 解析这4种格式
3. 设计一个文档类, 实现读取、另存和目录浏览功能
4. 设计一个视图类, 实现显示、缩放、漫游功能



## 理想情况1

- 一切顺利，学生S按期交付了软件，经过一两周的试用、修改、完善后，三方都比较满意，该软件在老师的朋友那里成为一个得心应手的工具



**Waterfall Model**

**Classic Life Cycle Model**

**Linear Sequential Model**

## 实际情况2

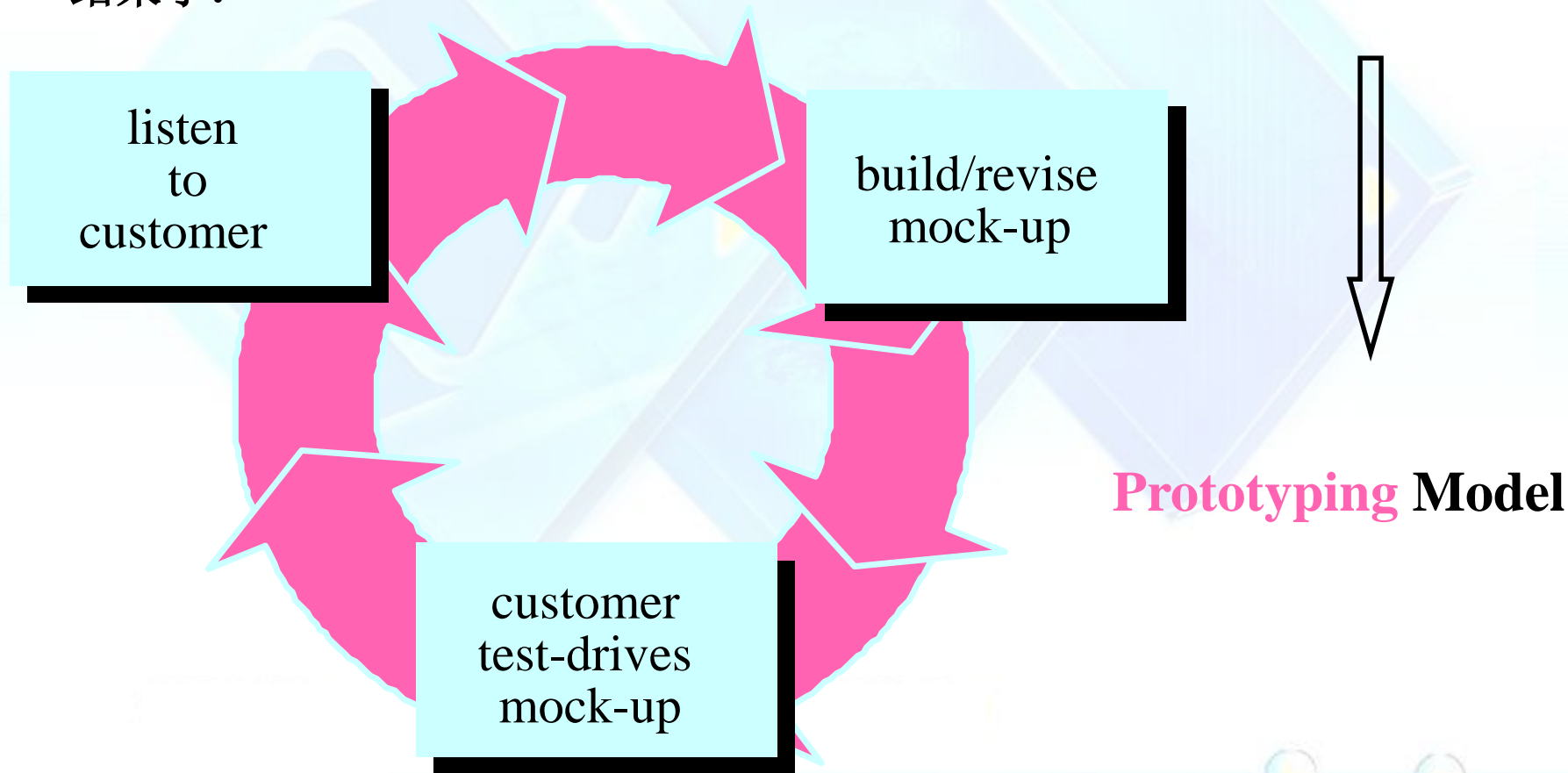
- 一周后，学生去见老师，并提交了工作清单，他发现老师的这位朋友C和老师在一起
  - S: 这是工作清单，我已经研究清楚4种文件的格式，可以写代码了。
  - T: 很好，不过我这位朋友有一些新想法，你不妨听听。
  - C: 你好！我新买了一个扫描仪，你的程序可不可以直接扫描图片进来。
  - S: 你可以自己扫描呀，买扫描仪的时候一般都会送正版软件的。
  - C: 是的，可是我一直不太会用，你知道我计算机水平不高，学一些新东西很累，也没有时间，如果你能直接链接扫描仪，我只要学会你的软件就行了，我愿意多支付一些费用……。还有，我想建一个图片库，你知道，我工作时需要上百个图片，经常找不到，最好还带模糊查询。
  - S: .....!!!!!!

## 实际情况2(续)

- C: 还有一些, 现在一时想不起来, 我想起来的话会再跟你联系, 时间上可以长一些。
  - S: .....!!!!!!
  - T: 要不这样吧, 你先做一个样子出来给C看看, 一边做, 一边改。
  - C: 这样最好, 看见一个基本样子我就知道我想要什么了。
- 
- 事情就这样定下来了, S愤怒的撕掉了自己的工作清单, 回去后花1天时间用Delphi做了个样子, 只能读BMP和JPG文件, 做了些菜单和工具栏, 用Access建了一个图片库
  - 就这个“假”的程序, S和C讨论了一天, S又修改了几次, 又讨论了几次, 一周后, 这个“假”的程序表面看起来和真的一模一样

## 实际情况2(续)

- 于是S打算用VC重写这个程序，但是他很快发现继续用Delphi写更方便，因为至少界面不用重做了，于是……，两个月后，这个事情终于结束了！



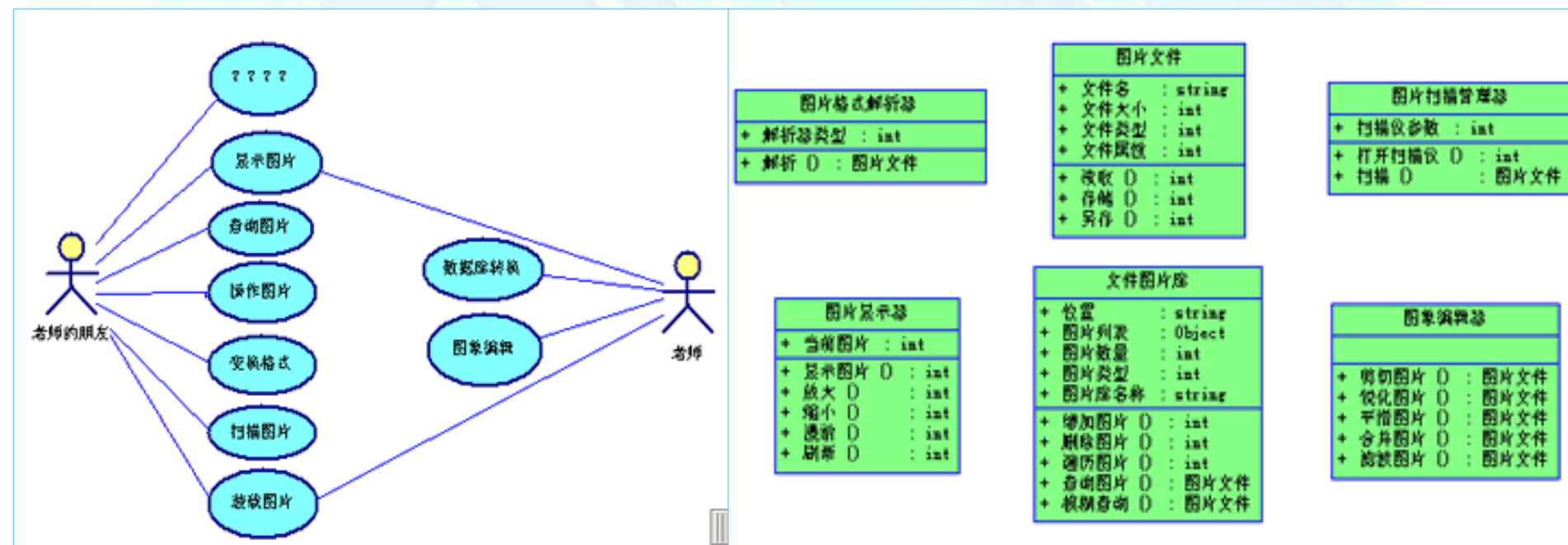
## 实际情况3

- 正如上一种情况一样，用户提出了很多新要求，但是麻烦还不止这些！
- 一天，老师T匆匆忙忙的找到S：
  - **T:** 我的研究生正在做的“海量多媒体数据库管理技术”项目需要一个对图象管理的模块，主要是数据库对象和图象文件之间的转换、显示和一些编辑操作，时间很紧，你目前正在做的代码可否直接利用一下？
  - **S:** 恐怕有难度，我不清楚……
  - **T:** 最好能够模块化强一些，你做的东西两边都能用，我这边比较急，一周后就要，我可以给你增加1个人一起做。
  - **S:** 可是……
  - **T:** 没有关系，就这样决定了，这是一次锻炼机会。我再帮你找一个这方面的专家，你可以请教他。下周这个时间我会再来。
- S感觉头脑里面“海量”、“JPG”、“编辑”、“图片库”、“一周时间”等等交织在一起，剪不清，理还乱。于是他准备去请教一下专家E。



## 实际情况3(续)

- E听了S说的情况，帮他画了两个图。
  - 用例模型：用于说清两个用户到底要什么
  - 领域模型：分析业务模型图中的名次和动词

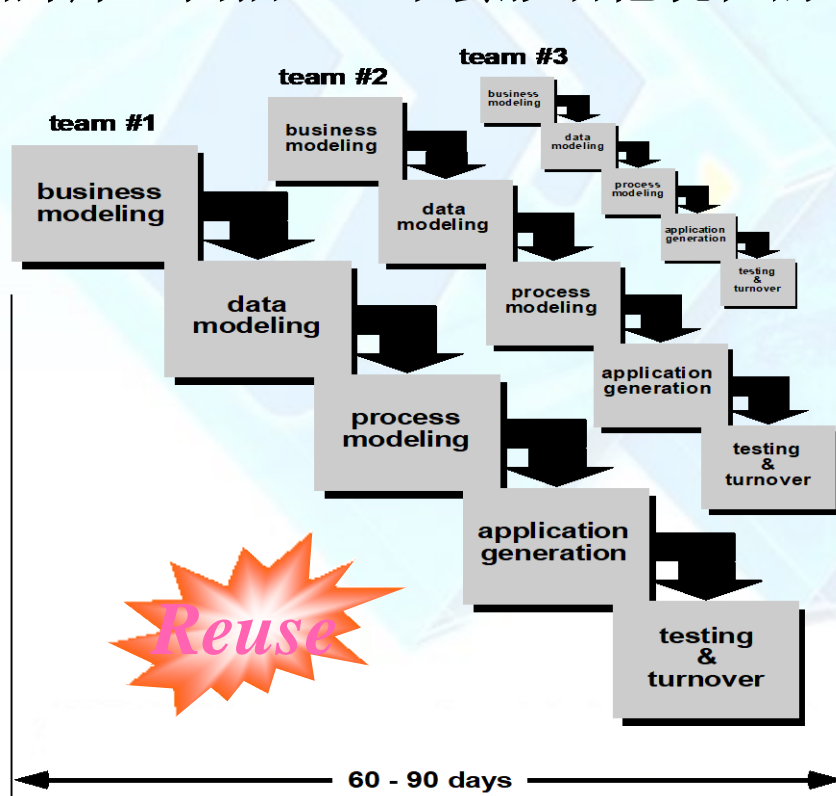


## 实际情况3(续)

- E要求S自己再画这样几张图：对于用例模型中的每一个用例，使用类图中的类说明业务中数据对象(类对象)之间的关联关系
- S试着这样做了，很快根据自己画的8张图进行了模块设计：
  - 1. 图片文件类模块和图片库类模块
  - 2. 图片格式解析器父类模块；5个图片解析子类模块(4个文件格式和一个数据库格式)
  - 3. 图片扫描管理器模块
  - 4. 图片编辑器模块
  - 5. 图片显示器模块
- S发现在网上有很多现成的图片扫描管理控件和图片编辑控件，完全满足要求，他自己花了一天一夜的时间编写了图片文件类模块和图片格式解析器父类，以及数据库解析子类，剩下的几天，他和老师新来的同学一起完成了剩余的模块。

## 实际情况3(续)

- 一周过去了，他将图片文件类模块、图片格式解析器父类模块、数据库解析子类，以及自己封装的图片编辑器交给了自己的老师，而由于每一个模块都是相对独立的，即使开始的用户要求他修改图片显示、图片库、扫描，也不会影响他现在的工作代码。



**RAD Process  
&  
Reuse-based Process**