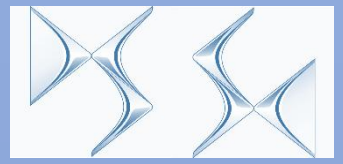
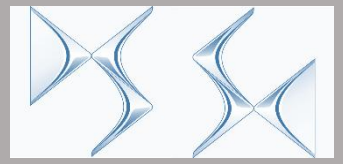


软件工程 模型与方法
Models & Methods of SE

软件生命周期模型

肖丁：dxiao@bupt.edu.cn　　QQ:14915703
田野：yetian@bupt.edu.cn

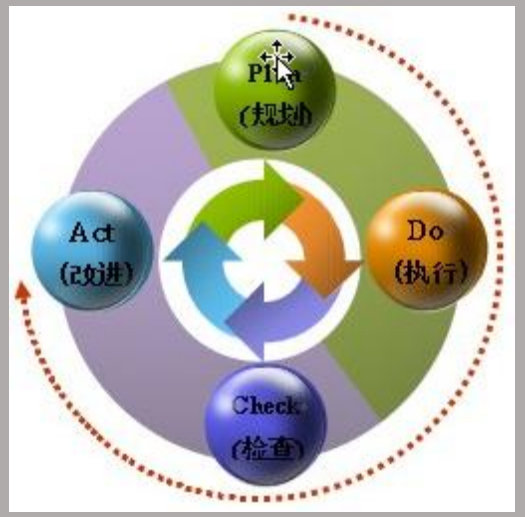


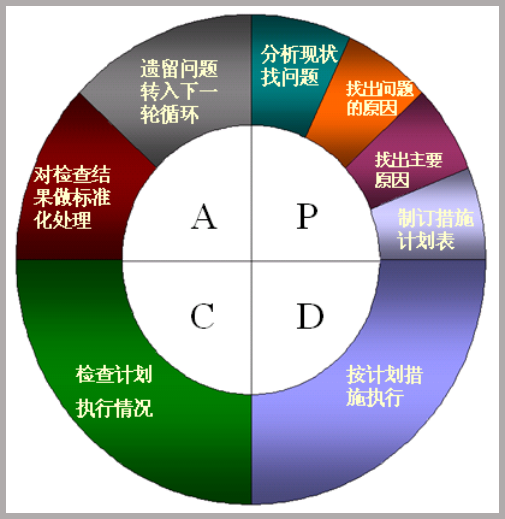
本章内容

- 软件工程过程
- 软件生命周期
- 传统软件生命周期模型
- 新型软件生命周期模型

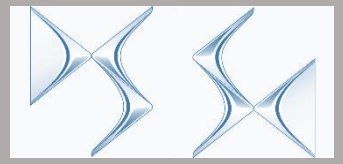
工程过程

- 工程项目有三个基本的目标：
  - 合理的进度；
  - 有限的经费；
  - 一定的质量；



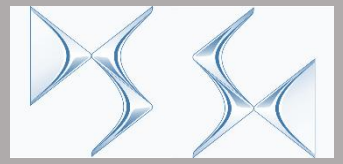


- 美国质量管理专家戴明博士针对工程项目的质量目标，提出了PDCA循环，称为戴明环
  - Plan, Do, Check, Action



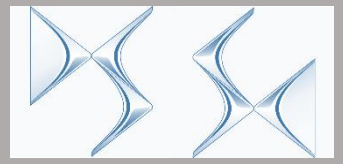
软件工程过程

- 软件工程过程是为了获得软件产品，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。主要活动有：
  - 编写软件规格说明：规定软件的功能及其使用限制；
  - 软件开发：产生满足规格说明的软件；
  - 软件确认：通过有效性验证以保证软件能够满足客户的要求；
  - 软件演进：为了满足客户的变更要求，软件必须在使用过程中进行不断地改进。



软件生命周期

- 软件生命周期：指软件产品从考虑其概念开始，直至废弃为止的整个时期，包括概念阶段、分析与设计阶段、构造阶段、移交和运行阶段等不同时期。
- 软件生命周期的六个基本步骤
  - 制定计划　　P
  - 需求分析　　D
  - 设计　　D
  - 程序编码　　D
  - 测试　　C
  - 运行维护　　A



制定计划

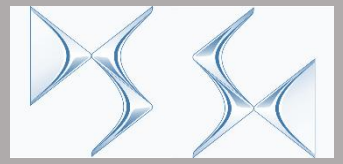
- 确定要开发软件系统的总目标；
- 给出功能、性能、可靠性以及接口等方面的要求；
- 完成该软件任务的可行性研究；
- 确定软件的过程模型及建模方法；
- 估计可利用的资源 (硬件，软件，人力等)、成本、效益、开发进度；
- 制定出完成开发任务的实施计划
  - 任务列表
  - 每个任务的起止时间
  - 每个任务的责任人
- 责任人：项目负责人

需求分析及软件设计

- 需求分析
  - 对用户提出的要求进行分析并给出详细的定义，确定软件的功能；
  - 编写软件需求规格说明书或系统功能说明书及初步的系统用户手册；
  - 提交管理机构评审；
  - 责任人：软件开发人员（泛指）/需求分析人员（具体）
- 软件设计
  - 概要设计：把各项需求转换成软件的功能结构。结构中每一组成部分都是意义明确的功能模块，每个功能都和某些需求相对应；
  - 详细设计：对每个模块要完成的工作进行具体的描述，为源程序编写打下基础；
  - 编写设计说明书，提交评审。
  - 责任人：软件开发人员（泛指）/软件架构师、软件设计人员、DBA等（具体）

程序编码及软件测试

- 软件实现
  - 把软件设计转换成计算机可以接受的程序代码，即写成以某一种特定程序设计语言表示的“源程序清单”；
  - 写出的程序应当是结构良好、清晰易读的，且与设计相一致的；
  - 责任人：软件开发人员（泛指）/编码工程师（具体）
- 软件测试
  - 单元测试，查找各模块在功能和结构上存在的问题并加以纠正；
  - 组装测试，将已测试过的模块按一定顺序组装起来；
  - 按规定的各项需求，逐项进行一系列有效性测试，决定已开发的软件是否合格，能否交付用户使用；
  - 责任人：软件测试人员、编码工程师、客户（具体）

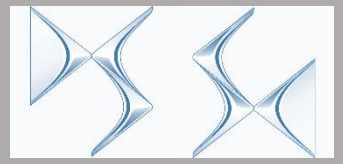


运行维护

- 软件交付给用户后的软件开发活动
  - 改正性维护：运行中发现了软件中的错误需要修正；
  - 适应性维护：为了适应变化了的软件工作环境，需做适当变更；
  - 完善性维护：为了增强软件的功能需做变更。
- 软件维护是更加复杂的软件开发活动
- 责任人：软件维护人员（泛指）、软件开发工程师（各种岗位）

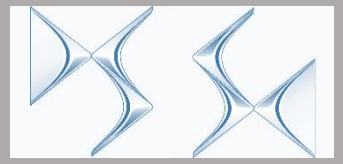
软件生命周期模型

- 软件过程模型：从一个特定角度提出的对软件过程的概括描述，是对软件开发实际过程的抽象，包括构成软件过程的:
  - 1、各种活动（Activities→How）；
  - 2、软件工件（Artifacts→What）；
  - 3、参与角色（Actors/Roles→Who）。
- 软件生命周期模型是一个框架，描述从软件需求定义直至软件经使用后废弃为止，跨越整个生存期的软件开发、运行和维护所实施的全部过程、活动和任务，同时描述生命周期不同阶段产生的软件工件，明确活动的执行角色等。



传统模型种类

- 瀑布模型（Waterfall Model）
- 演化模型（Evolutional Model）
- 增量模型（Incremental Model）
- 喷泉模型（Fountain Model）
- V模型和W模型（V & W Model）
- 螺旋模型（Spiral Model）
- 构件组装模型（Component Assembly Model）
- 快速应用开发模型（Rapid Application Development Model）
- 原型方法（Prototype Method）



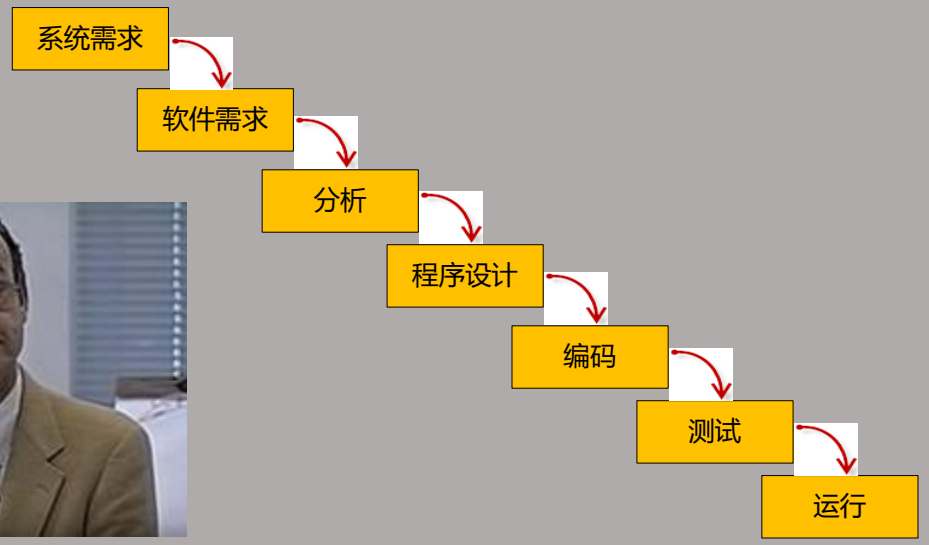
无规则的小作坊式软件开发

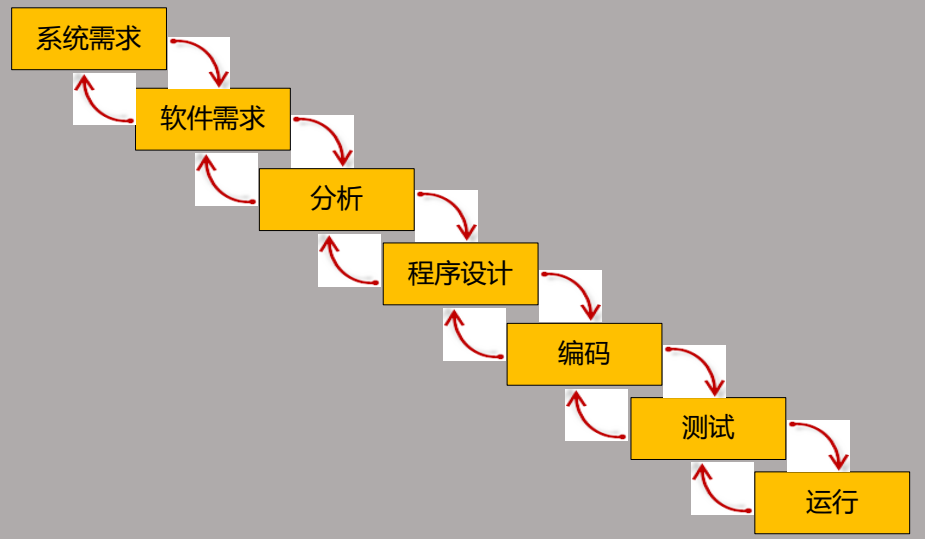
- 获取需求
  - 面谈或简单的书面说明
- 理解需求
  - 头脑风暴
- 编写代码
- 运行+调试
- 交付

- 缺乏统一的项目规划；
- 不重视需求的分析；
- 软件交付前缺少设计和测试；
- 软件质量无保障；
- 基本不考虑维护；
- 无法支持大规模的软件开发！

瀑布模型

- Winston W. Royce于1970年在其论文 “Managing the Development of Large Software Systems” 中第一次提出了 瀑布模型（Waterfall Model）



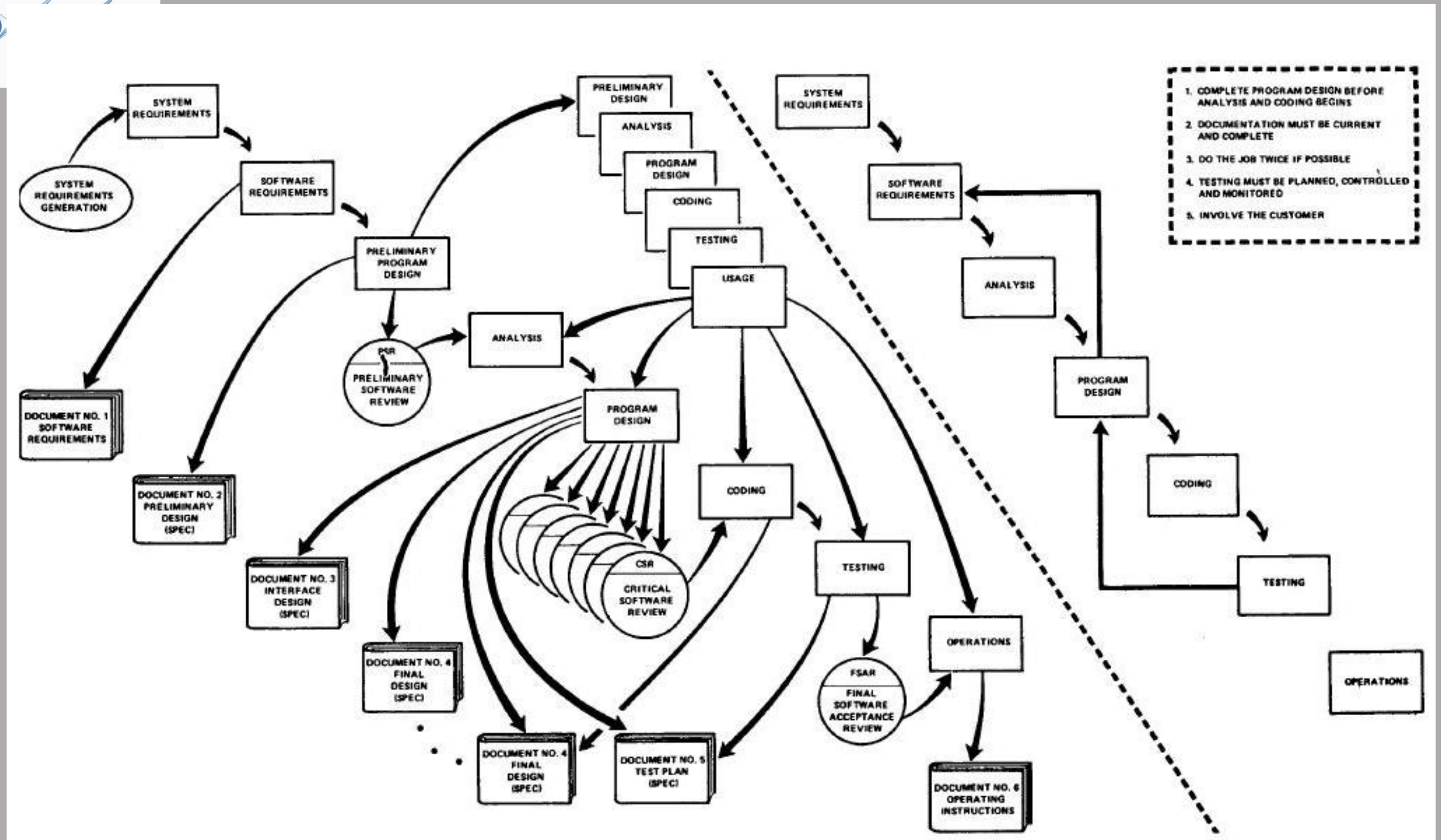


瀑布模型为软件开发和软件维护提供了一种有效的管理模式，它在软件开发早期为消除非结构化软件、降低软件复杂度、促进软件开发工程化方面起着显著的作用。

瀑布模型

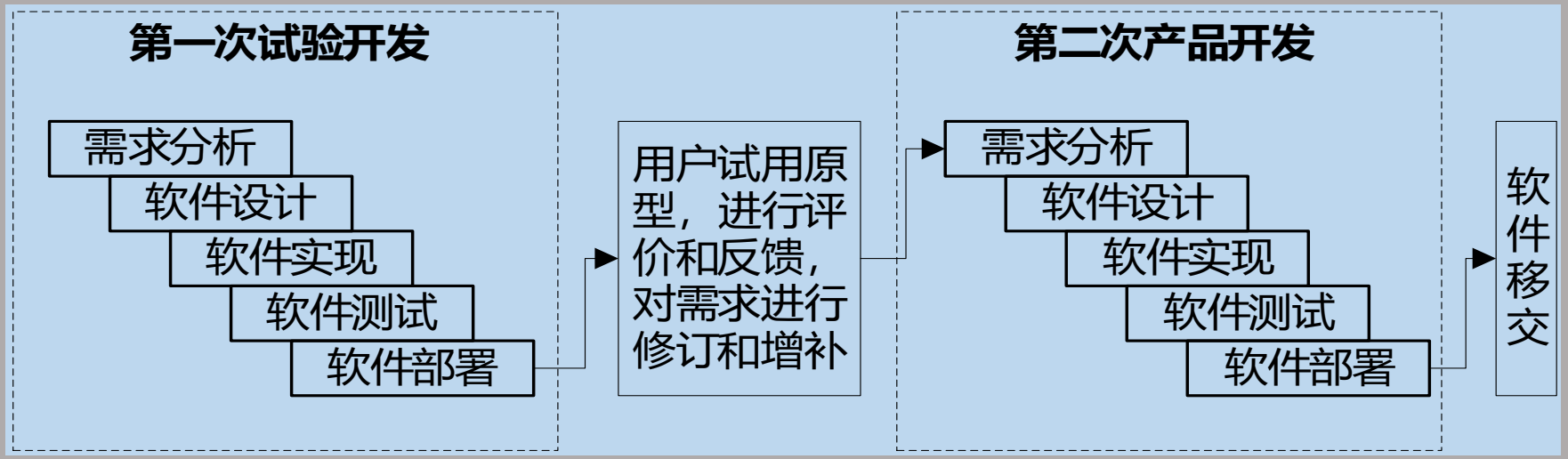
- 瀑布模型中的每一个阶段的开发活动具有下列特征：
  - 本阶段的工作对象来自于上一阶段活动的输出，这些输出一般是代表上一阶段活动结束的里程碑式的文档。
  - 根据本阶段的活动规程执行相应的任务。
  - 产生本阶段活动相关产出，软件工件，作为下一阶段活动的输入

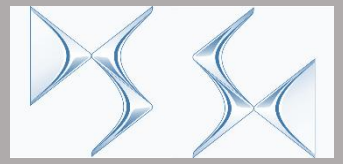
优点	缺点
降低了软件开发的复杂程度，提高了软件开发过程的透明性及软件开发过程的可管理性。	模型缺乏灵活性，特别是无法解决软件需求不明确或不准确的问题。
推迟了软件实现，强调在软件实现前必须进行分析和设计工作。	模型的风险控制能力较弱。
以项目的阶段评审和文档控制为手段有效地对整个开发过程进行指导，保证了阶段之间的正确衔接，能够及时发现并纠正开发过程中存在的缺陷，从而能够使产品达到预期的质量要求。	瀑布模型中的软件活动是文档驱动的，当阶段之间规定过多的文档时，会极大地增加系统的工作量；而且当管理人员以文档的完成情况来评估项目完成进度时，往往会产生错误的结论。


SYSTEM REQUIREMENTS GENERATION
SYSTEM REQUIREMENTS
SOFTWARE REQUIREMENTS
PRELIMINARY PROGRAM DESIGN
PRELIMINARY DESIGN
ANALYSIS
PROGRAM DESIGN
CODING
TESTING
USAGE
PSR
PRELIMINARY SOFTWARE REVIEW
ANALYSIS
PROGRAM DESIGN
CODING
TESTING
CSR
CRITICAL SOFTWARE REVIEW
DOCUMENT NO. 1 SOFTWARE REQUIREMENTS
DOCUMENT NO. 2 PRELIMINARY DESIGN (SPEC)
DOCUMENT NO. 3 INTERFACE DESIGN (SPEC)
DOCUMENT NO. 4 FINAL DESIGN (SPEC)
DOCUMENT NO. 4 FINAL DESIGN (SPEC)
DOCUMENT NO. 5 TEST PLAN (SPEC)
FSAR
FINAL SOFTWARE ACCEPTANCE REVIEW
OPERATIONS
DOCUMENT NO. 6 OPERATING INSTRUCTIONS
SYSTEM REQUIREMENTS
SOFTWARE REQUIREMENTS
ANALYSIS
PROGRAM DESIGN
CODING
TESTING
OPERATIONS
1. COMPLETE PROGRAM DESIGN BEFORE ANALYSIS AND CODING BEGINS
2. DOCUMENTATION MUST BE CURRENT AND COMPLETE
3. DO THE JOB TWICE IF POSSIBLE
4. TESTING MUST BE PLANNED, CONTROLLED AND MONITORED
5. INVOLVE THE CUSTOMER

演化模型

- 使用瀑布模型人们认识到，由于需求很难调研充分，所以很难一次性开发成功。
- 演化模型提倡两次开发：
  - 第一次是试验开发，得到试验性的原型产品，其目标只是在于探索可行性，弄清软件需求；
  - 第二次在此基础上获得较为满意的软件产品。



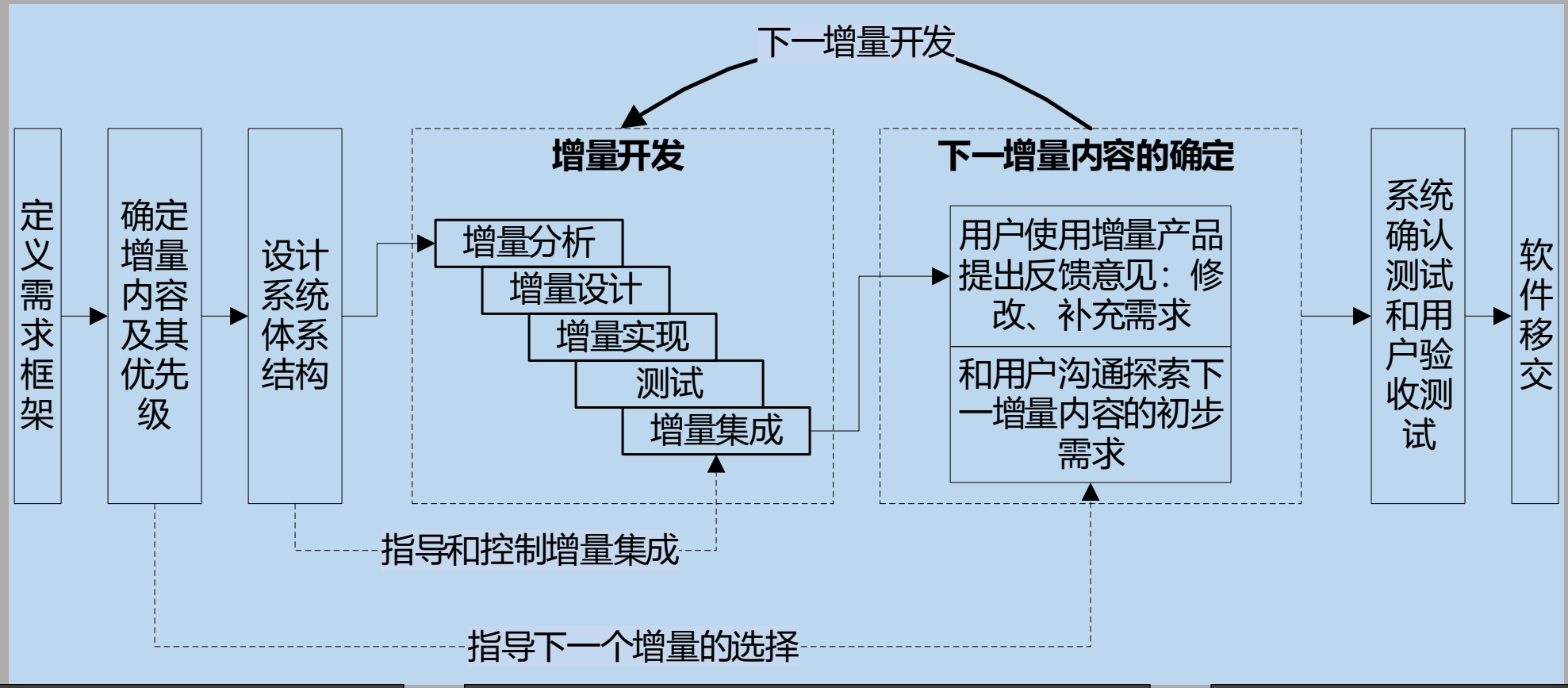


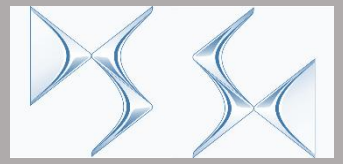
演化模型

- 演化模型的特点：
  - 优点：明确用户需求、提高系统质量、降低开发风险；
  - 缺点：
    - 难于管理、结构较差、技术不成熟；
    - 可能会抛弃瀑布模型的文档控制优点；
    - 可能会导致最后的软件系统的系统结构较差；
- 演化模型适用范围：
  - 需求不清楚；
  - 小型或中小型系统；
  - 开发周期短

增量模型

- Mills等人于1980年提出，指首先对系统最核心或最清晰的需求进行分析、设计、实现、测试并集成到系统中。再按优先级逐步对后续的需求进行上述工作，逐步建设成一个完整系统的开发方法。结合了瀑布模型和演化模型的优点。



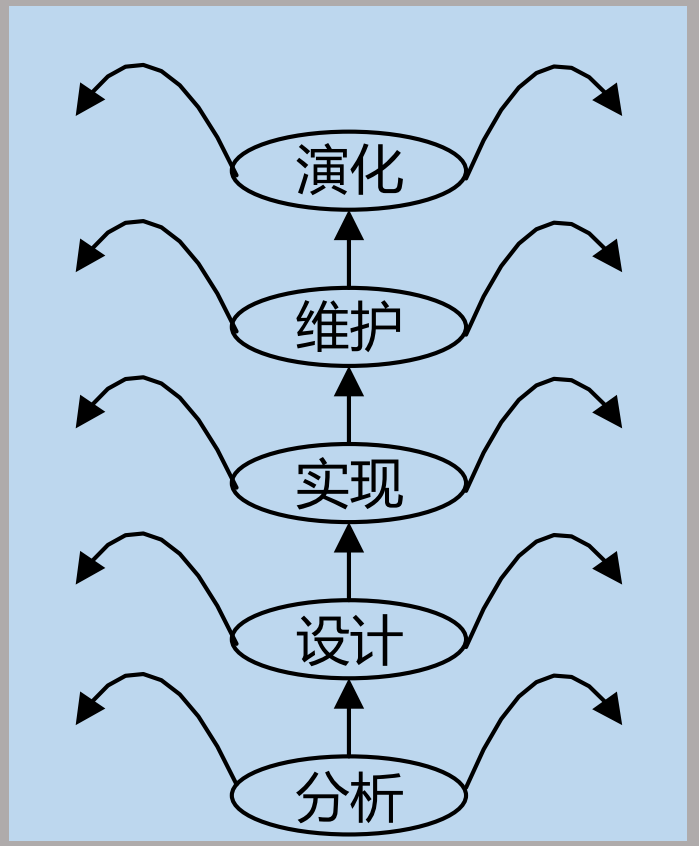


增量模型

- 增量模型的优点：
  - 客户可以在第一次增量后就使用到系统的核心功能，增强了客户使用系统的信心；
  - 项目总体失败的风险较低，因为核心功能先开发出来，即使某一次增量失败，核心功能的产品客户仍然可以使用。
  - 由于增量是按照从高到低的优先级确定的，最高优先级的功能得到最多次的测试，保障了系统重要功能部分的可靠性。
  - 所有增量都是在同一个体系结构指导下进行集成的，提高了系统的稳定性和可维护性。
- 增量模型的缺点：
  - 增量粒度难以选择；
  - 确定所有的需求比较困难；

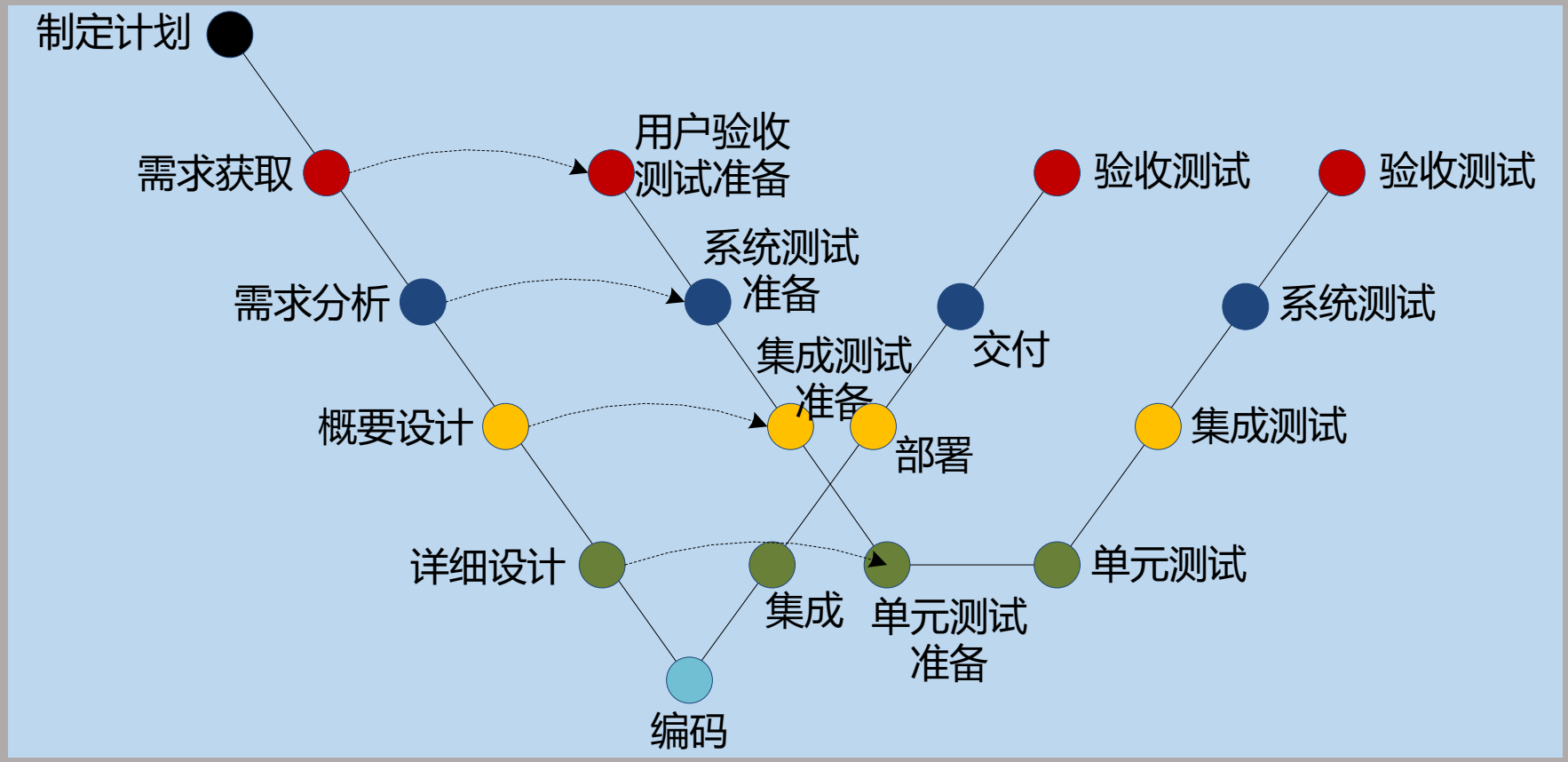
喷泉模型

- 喷泉模型也称迭代模型，认为软件开发过程的各个阶段是相互重叠和多次反复的，就象喷泉一样，水喷上去又可以落下来，既可以落在中间，又可以落到底部。
- 各个开发阶段没有特定的次序要求，完全可以并行进行，可以在某个开发阶段中随时补充其他任何开发阶段中遗漏的需求。
- 优点：
  - 提高开发效率
  - 缩短开发周期
- 缺点：难于管理，工作计划要随时更新



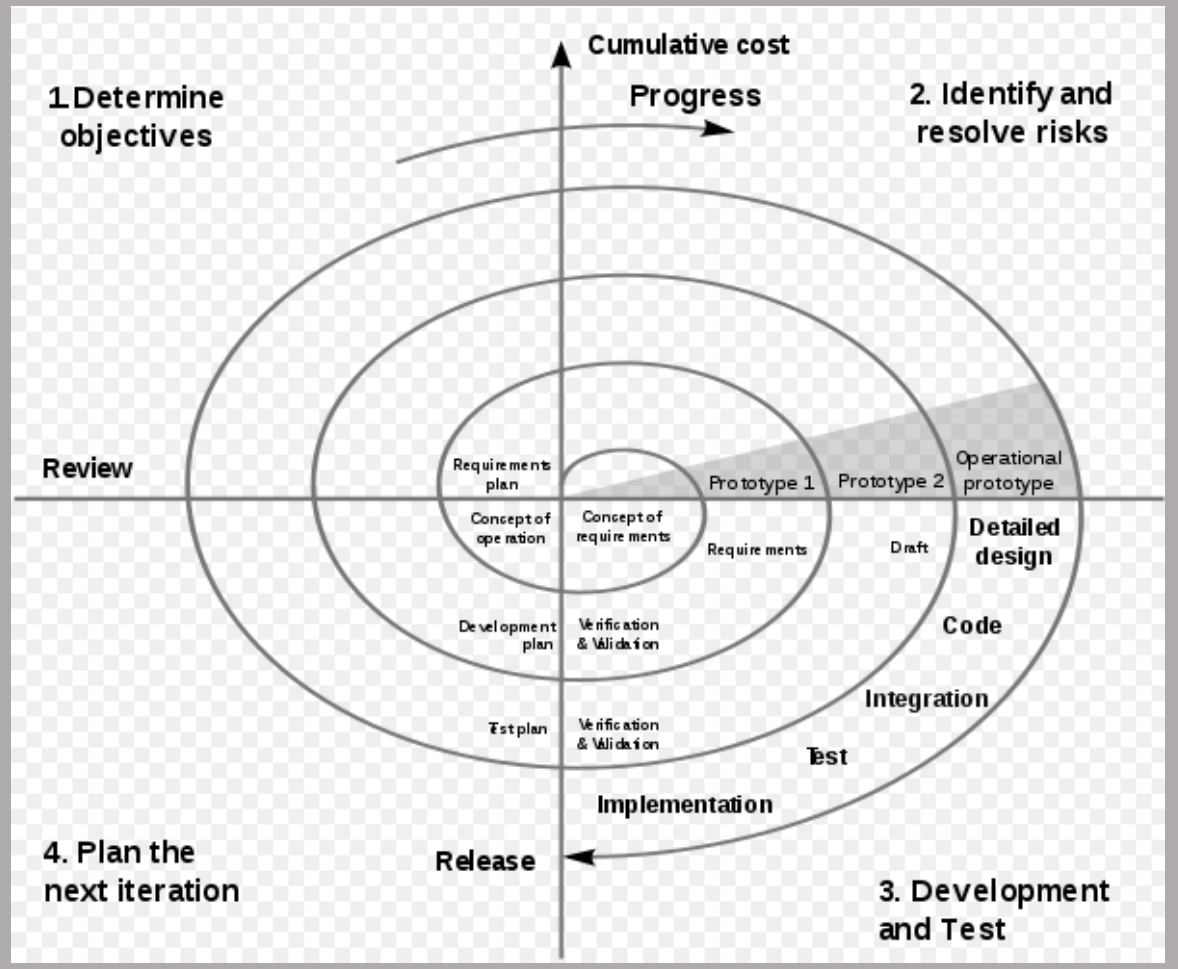
V模型和W模型

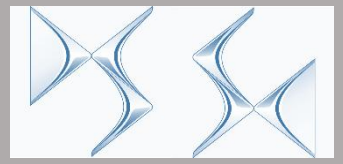
- 1980年代后期Paul Rook提出了V模型，将测试活动提前，使得瀑布模型能够驾驭风险。Evolutif公司在V模型的基础上提出了W模型。



螺旋模型

- Boehm于1988年提出，主要针对大型软件项目的开发周期长，风险高的特点。
- 四个象限
  - 制定计划
  - 风险分析
  - 实施工程
  - 客户评价





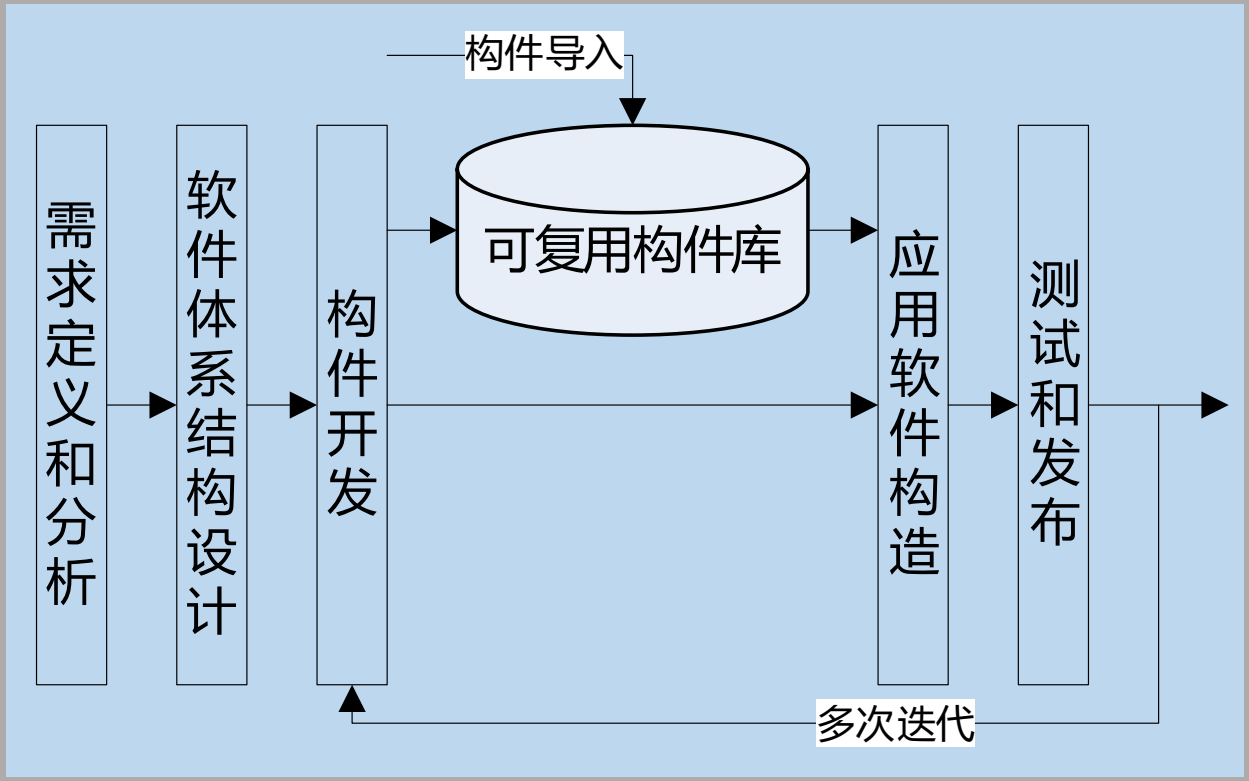
螺旋模型

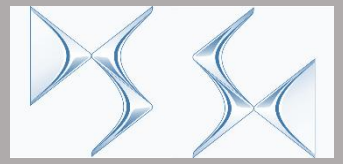
- 制定计划：确定软件项目目标；明确对软件开发过程和软件产品的约束；制定详细的项目管理计划；根据当前的需求和风险因素，制定实施方案，并进行可行性分析，选定一个实施方案，并对其进行规划。
- 风险分析：明确每一个项目风险，估计风险发生的可能性、频率、损害程度，并制定风险管理措施规避这些风险。
- 实施工程：针对每一个开发阶段的任务要求参照某一种生命周期模型执行本开发阶段的活动。
- 客户评估：客户使用原型，反馈修改意见；根据客户的反馈，对产品及其开发过程进行评审，决定是否具备进入到下一个迭代中。

构件组装模型

- 利用模块化思想将整个系统模块化，并在一定构件模型的支持下复用构件库中软件构件，通过组装高效率、高质量地构造软件系统。构件组装模型本质上是演化的，开发过程是迭代的 。

- 构件组装模型的开发过程就是构件组装的过程，维护的过程就是构件升级、替换和扩充的过程。



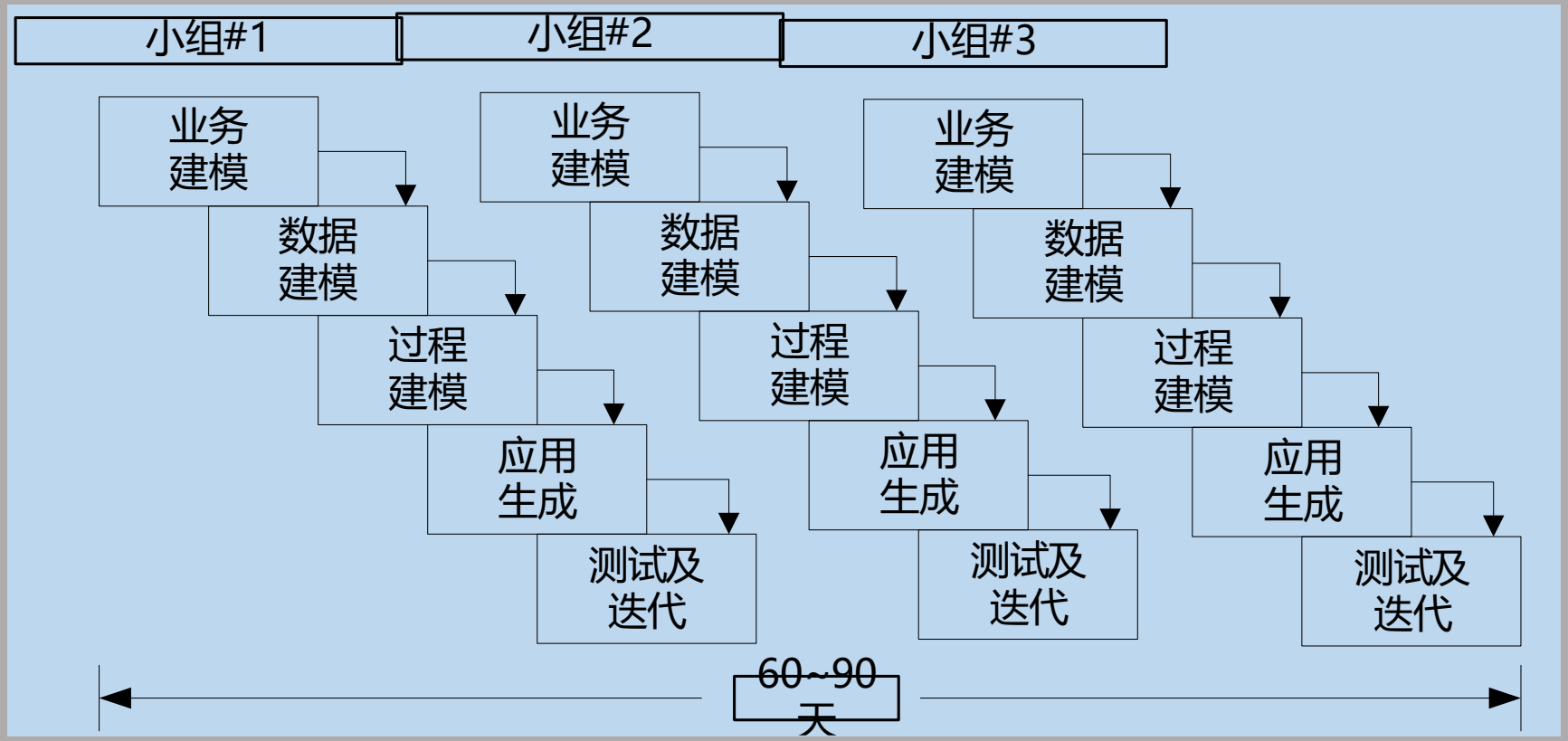


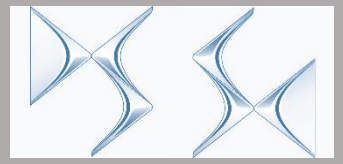
构件组装模型

- 优点：
  - 充分利用软件复用，提高了软件开发的效率。
  - 允许多个项目同时开发，降低了费用，提高了可维护性，可实现分步提交软件产品。
- 缺点：
  - 缺乏通用的构件组装结构标准，风险较大；
  - 构件可重用性和系统高效性之间不易协调；
  - 由于过分依赖于构件，构件质量影响着最终产品的质量。

快速应用开发模型

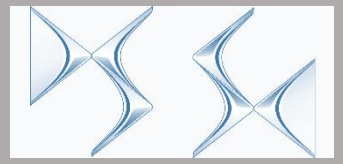
- 快速应用开发（Rapid Application Development，RAD）是一个增量型的软件开发过程模型，强调极短的开发周期 。





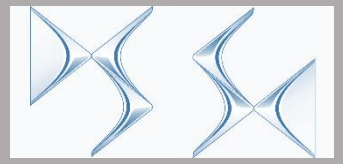
快速应用开发模型

- RAD模型的缺点：
  - 并非所有应用都适合采用RAD
  - 由于时间约束，开发人员和客户必须在较短的时间内完成一系列的需求分析，沟通配合不当都会导致应用RAD模型的失败
  - RAD适合于管理信息系统的开发，对于其他类型的应用系统，如技术风险较高、与外围系统的互操作性较高等，不太合适



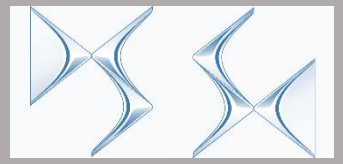
原型方法

- 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。
- 然而完整而准确的需求规格说明是很难一次性得到，因为：
  - 在开发早期用户往往对系统只有一个模糊的想法，很难完全准确地表达对系统的全面要求
  - 随着开发工作的推进，用户可能会产生新的要求
  - 开发者又可能在设计与实现的过程中遇到一些没有预料到的实际困难，需要以改变需求来解脱困境



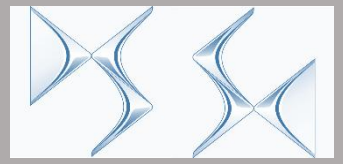
原型方法的定义

- 原型指模拟某种最终产品的原始模型；
- 原型方法指在获得一组基本需求后，通过快速分析构造出一个小型的软件系统原型，满足用户的基本要求。
- 用户通过使用原型系统，提出修改意见，从而减少用户与开发人员对系统需求的误解，使需求尽可能准确。
- 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。



原型方法的种类

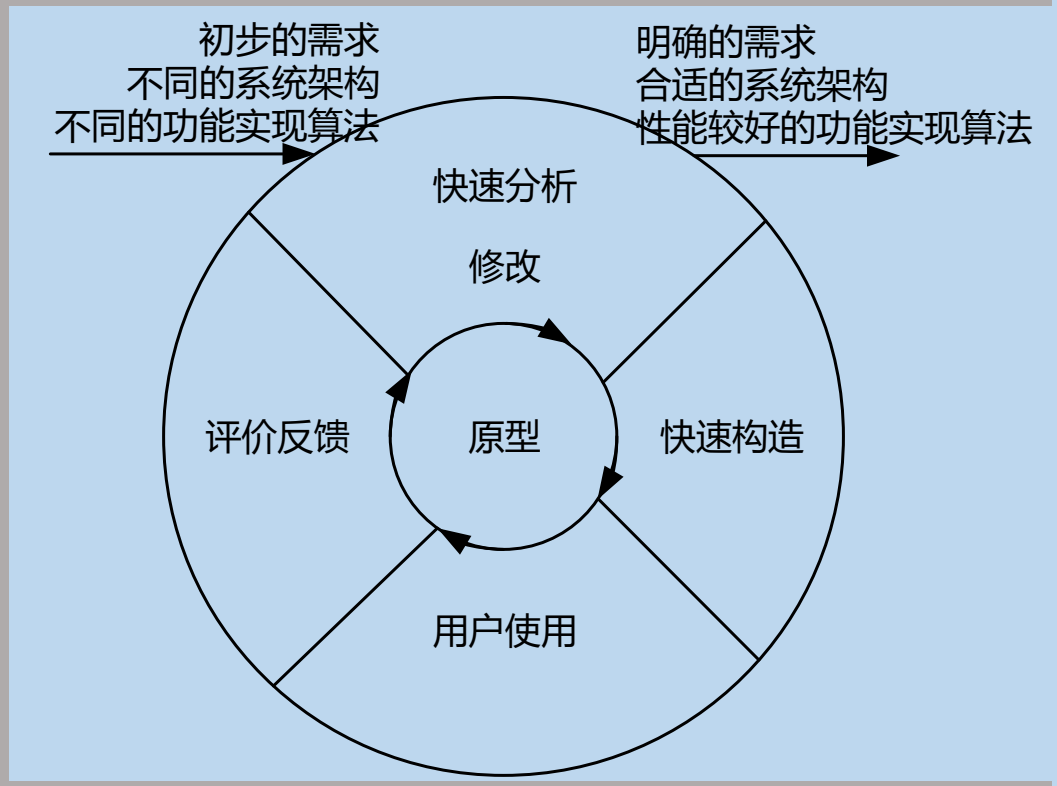
- 废弃策略
  - 探索型：弄清用户对目标系统的要求，确定所期望的特性；探讨多种实现方案的可行性。主要针对需求模糊、用户和开发者对项目开发都缺乏经验的情况。
  - 实验型：用于大规模开发和实现之前，考核技术实现方案是否合适、分析和设计的规格说明是否可靠。
- 追加策略
  - 进化型：在构造系统的过程中能够适应需求的变化，通过不断地改进原型，逐步将原型进化成最终的系统。它将原型方法的思想扩展到软件开发的全过程，适用于需求经常变动的软件项目。

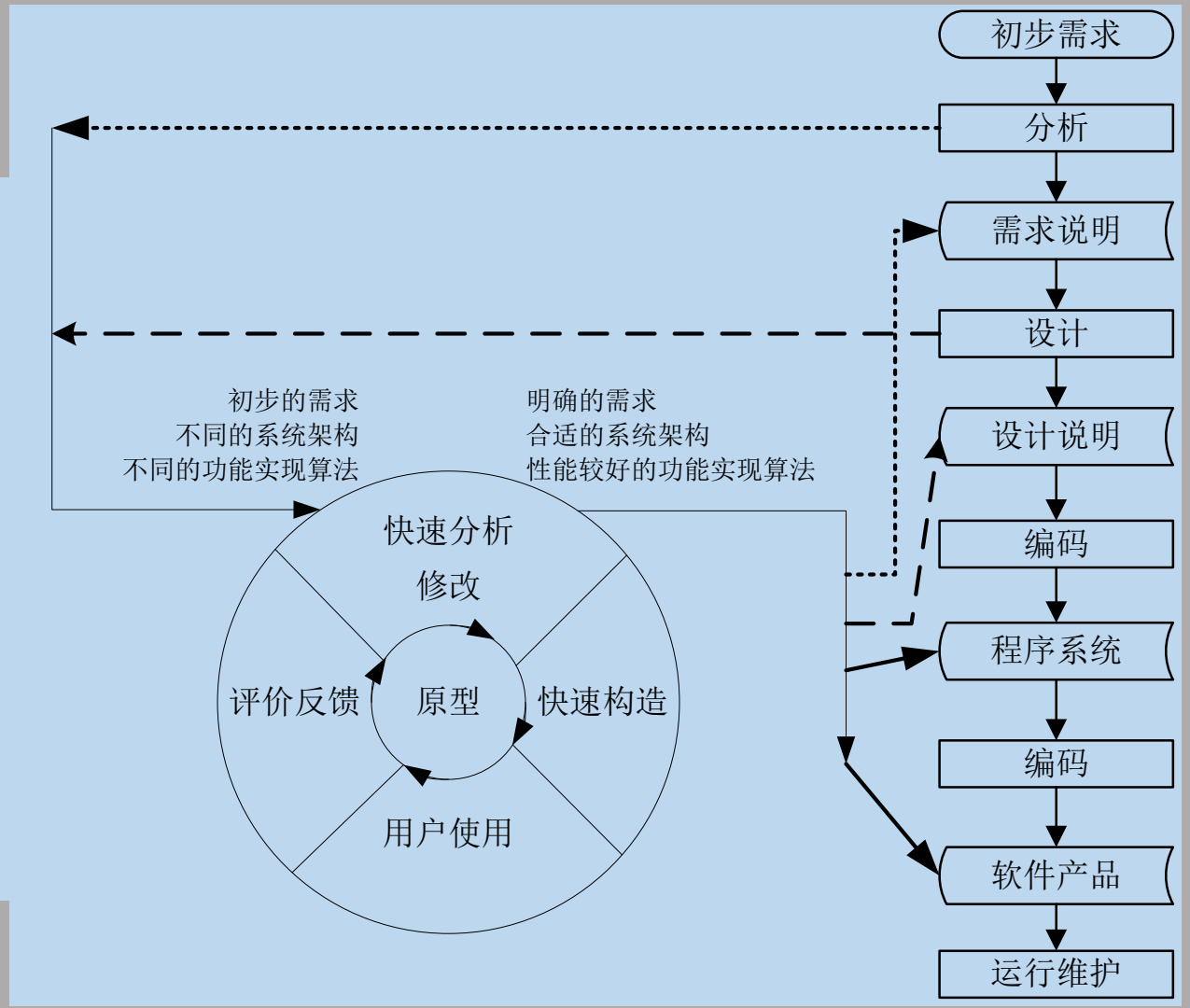


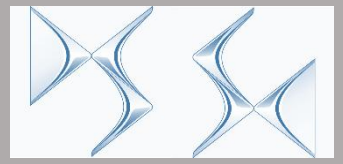
原型方法的优点

- 原型提供了用户与开发人员良好的沟通手段，易于被人们接受：
  - 原型方法有助于快速理解用户对于需求的真实想法；
  - 可以容易地确定系统的性能，确认各项主要系统服务的可应用性，确认系统设计的可行性，确认系统作为产品的结果；
  - 软件原型的最终版本，有的可以原封不动地成为产品，有的略加修改就可以成为最终系统的一个组成部分，这样有利于建成最终系统。
- 原型法的适用范围和局限性：
  - 大型系统如不经过系统分析得到系统的整体划分，而直接用原型来模拟是很困难的。
  - 对于大量运算的、逻辑性较强的程序模块，原型方法很难构造出该模块的原型来供人评价。
  - 对于原有应用的业务流程、信息流程混乱的情况，原型构造与使用有一定的困难。
- 原型方法存在的问题：
  - 文档容易被忽略。
  - 建立原型的许多工作会被浪费掉 。
  - 项目难以规划和管理。

原型方法的应用



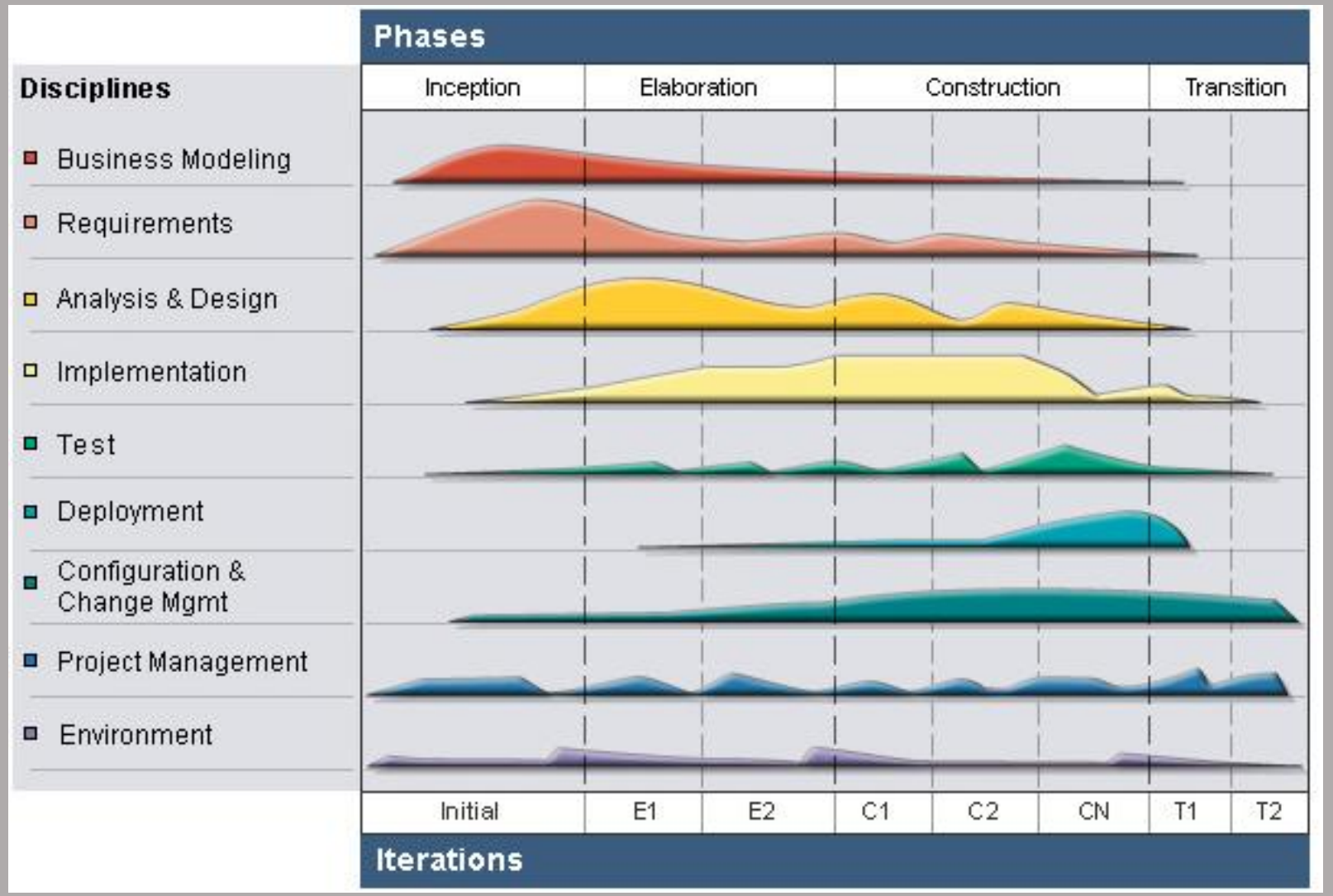




新型软件生命周期模型

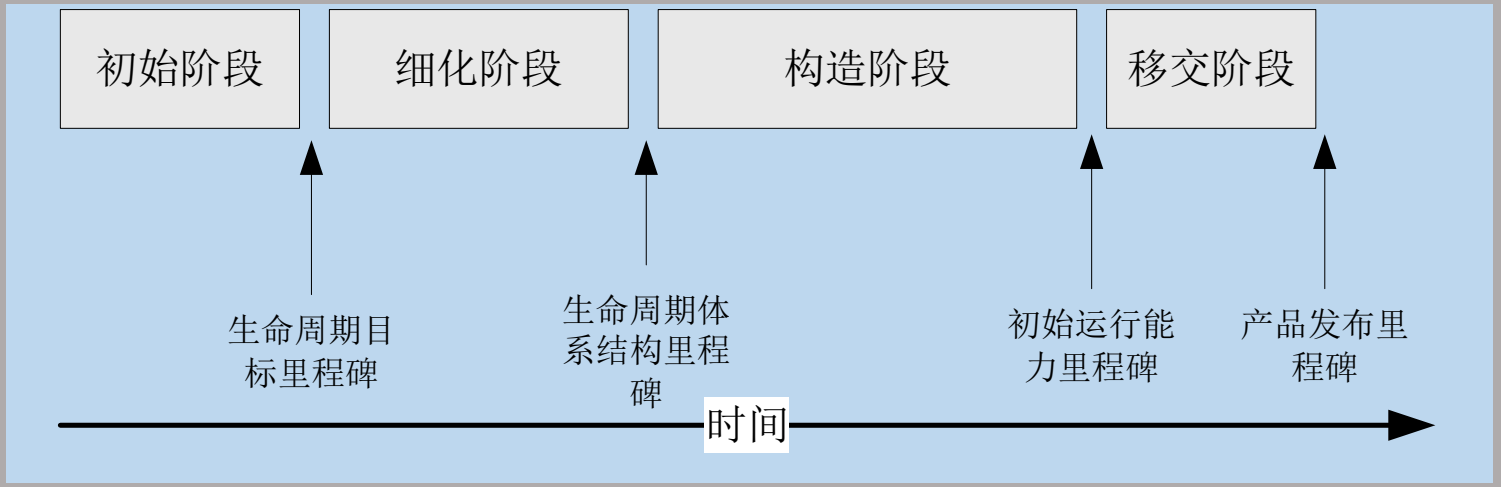
- RUP
  - RUP（Rational Unified Process）是由Rational公司（现被IBM公司收购）开发的一种软件工程过程框架，是一个基于面向对象的程序开发方法论 。
  - RUP既是一种软件生命周期模型，又是一种支持面向对象软件开发的工具，它将软件开发过程要素和软件工件要素整合在统一的框架中 。
- 敏捷及极限编程
  - 敏捷建模（Agile Modeling，AM）是由Scott W. Ambler从许多的软件开发过程实践中归纳总结出来的一些敏捷建模价值观、原则和实践等组成的，它是快速软件开发的一种思想代表，具体的应用有极限编程、SCRUM、水晶、净室开发等。
  - 2001年敏捷联盟成立，其主要特点就是具有快速及灵活的响应变更的能力。

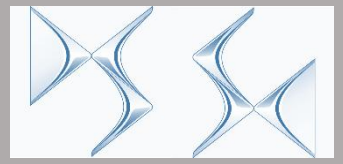
RUP的基本结构



RUP的四个主要阶段

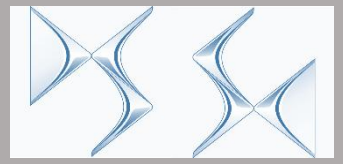
- RUP中的软件生命周期在时间上被分解为四个顺序的阶段：初始阶段（Inception）、细化阶段（Elaboration）、构造阶段（Construction）和交付阶段（Transition）。
- 每个阶段结束于一个主要的里程碑(Major Milestones)，并在阶段结尾执行一次评估以确定这个阶段的目标是否已经满足。如果评估结果令人满意的话，可以允许项目进入下一个阶段。





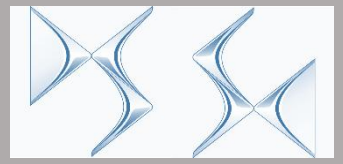
RUP 初始阶段

- 阶段目标是通过业务场景（business case）了解业务并确定项目的边界。包括项目的验收规范、风险评估、所需资源估计、阶段计划等。
  - 要确定项目边界，需识别所有与系统交互的外部实体，主要包括识别外部角色（actor）、识别所有用例并详细描述一些重要的用例。
  - Milestone：软件目标里程碑。包括一些重要的文档，如：项目愿景（vision）、原始用例模型、原始业务风险评估、一个或者多个原型、原始业务场景等。
  - 需要对这些文档进行评审，以确定正确理解用例需求、项目风险评估合理、阶段计划可行等。



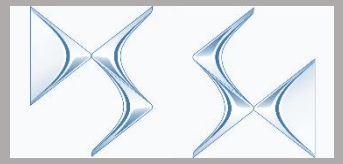
RUP 细化阶段

- 阶段目标是分析问题领域，建立适合需求的软件体系结构基础，编制项目计划，完成项目中技术要求高、风险大的关键需求的开发。
- Milestone：体系结构里程碑。
  - 包括风险分析文档、软件体系结构基线、项目计划、可执行的进化原型、初始版本的用户手册等。
- 通过评审确定软件体系结构的稳定性、确认高风险的业务需求和技术机制已经解决、修订的项目计划可行等。



RUP 构造阶段

- 阶段目标是将所有剩余的技术构件和稳定业务需求功能开发出来，并集成为产品，所有功能被详细测试。
- 从某种意义上说，构造阶段只是一个制造过程，其重点放在管理资源及控制开发过程以优化成本、进度和质量。
- Milestone：运行能力里程碑。
  - 包括可以运行的软件产品、用户手册等，它决定了产品是否可以在测试环境中进行部署。
- 此刻，要确定软件、环境、用户是否可以开始系统的运行。

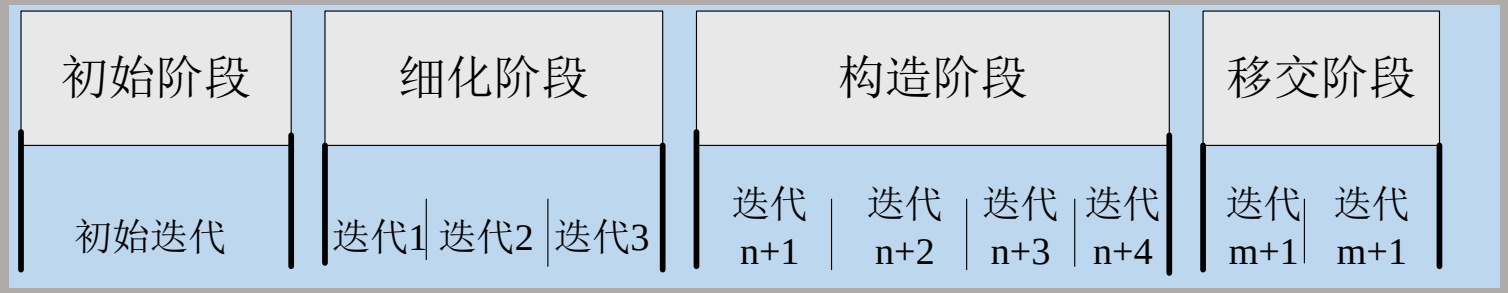


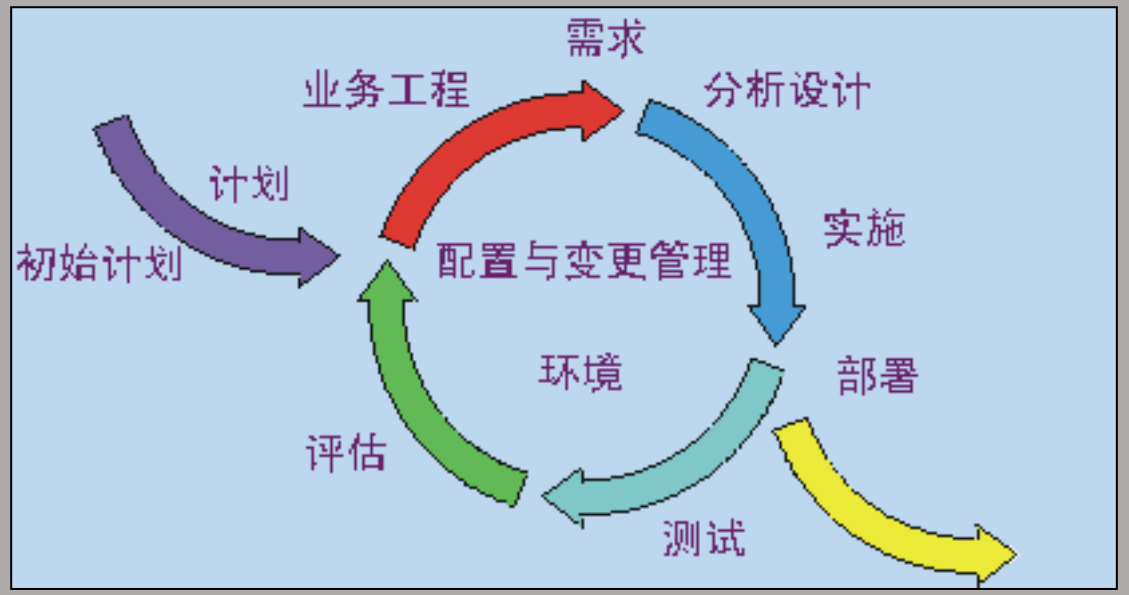
RUP 移交阶段

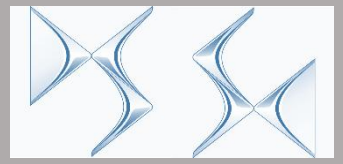
- 移交阶段的重点是确保软件对最终用户是可用的。交付阶段可以跨越几次迭代，包括为发布做准备的产品测试，基于用户反馈的少量调整。
- Milestone：产品发布里程碑。
- 此时，要确定最终目标是否实现，是否应该开始产品下一个版本的另一个开发周期。

RUP的特点

- 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。

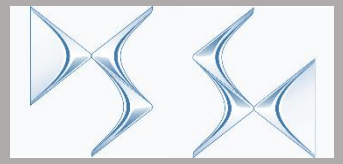






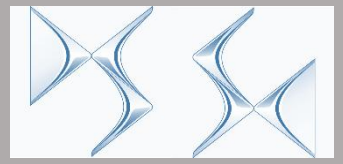
RUP的核心活动

- UP由9个核心工作流构成每一个迭代的主要活动
- 6个核心过程工作流
  - 业务建模（Business Modeling）
  - 需求（Requirements）
  - 分析和设计（Analysis & Design）
  - 实现（Implementation）
  - 测试（Test）
  - 部署（Deployment）
- 3个核心支持工作流：
  - 配置和变更管理（Configuration & Change Management）
  - 项目管理（Project Management）
  - 环境（Environment）



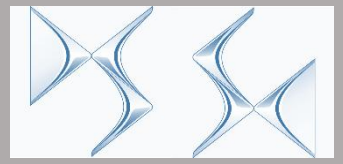
2.5.2 敏捷建模

- 敏捷建模（Agile Modeling，AM）是由Scott W. Ambler从许多的软件开发过程实践中归纳总结出来的一些敏捷建模价值观、原则和实践等组成的，它是快速软件开发的一种思想代表，具体的应用有极限编程、SCRUM、水晶（Crystal）、净室（Clean Room）开发等。
- 2001年敏捷联盟成立，其主要特点就是具有快速及灵活的响应变更的能力。http://agilemanifesto.org/



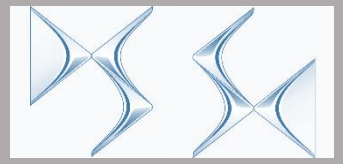
敏捷宣言

- We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:
  - *Individuals and interactions over processes and tools*
  - *Working software over comprehensive documentation*
  - *Customer collaboration over contract negotiation*
  - *Responding to change over following a plan*
- That is, while there is value in the items on the right, we value the items on the left more.



敏捷思想的核心原则

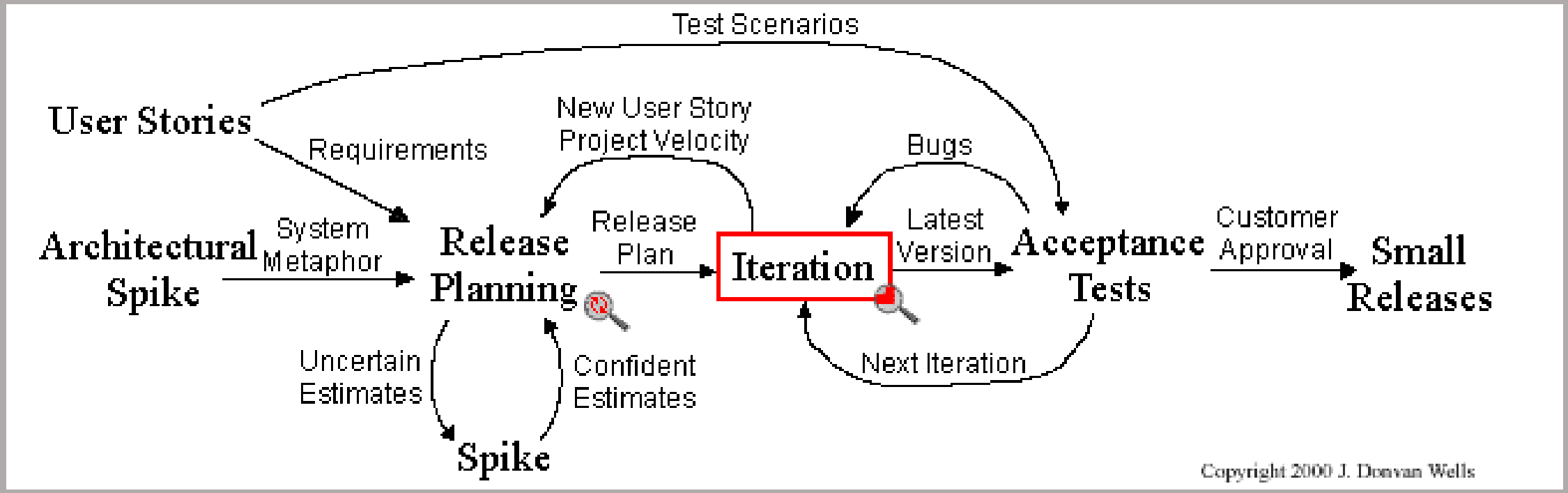
- Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
- Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
- Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
- Business people and developers must work together daily throughout the project.
- Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
- The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.



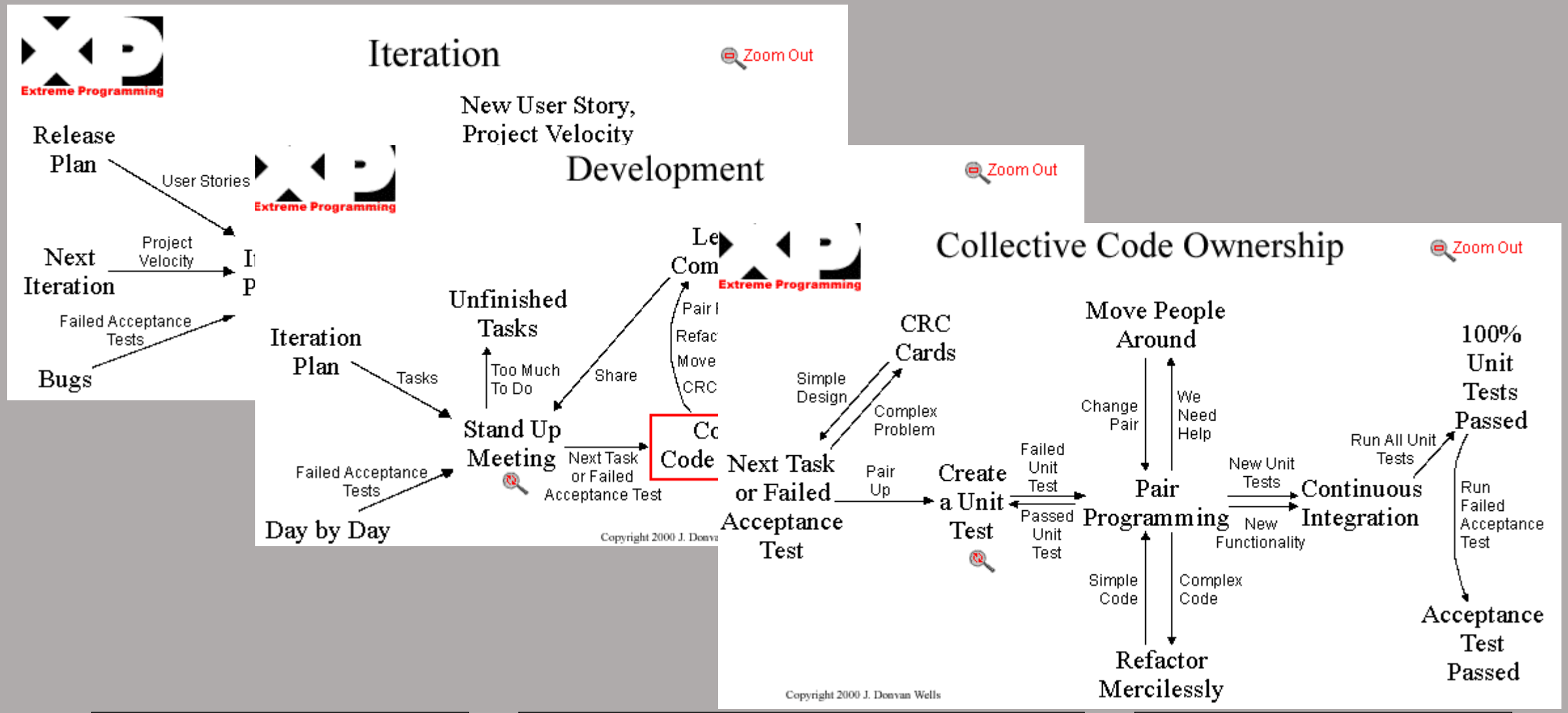
eXtreme Programming 极限编程

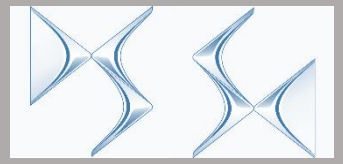
- 1996年三月，Kent Beck在为Daimler Chrysler所做的一个项目中引入了新的软件开发观念：XP。
- XP是一种轻量级的软件开发方法，是一种以实践为基础的软件工程过程和思想。
- 它使用快速的反馈，大量而迅速的交流，经过保证的测试来最大限度的满足用户的需求。
- XP强调用户满意，开发人员可以对需求的变化作出快速的反应。

极限编程的处理流程



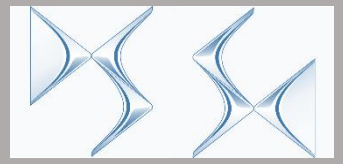
XP 的迭代开发





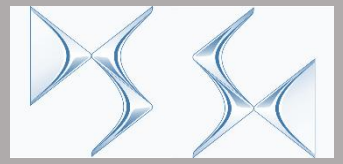
极限的工作环境

- 为了在软件开发过程中最大程度地实现和满足客户和开发人员的基本权利和义务，XP要求把工作环境也做得最好。
- 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。
- 所有人都在同一个开放的开发环境中工作
  - 最好是所有人在同一个大房子中工作，还有茶点供应；
  - 每周40小时，不提倡加班；
  - 每天早晨，所有人一起站着开个短会；
  - 墙上有一些大白板，所有的Story卡、CRC卡等都贴在上面，讨论问题的时候可以在上面写写画画；
  - 下班后大家可以一起玩电脑游戏……。



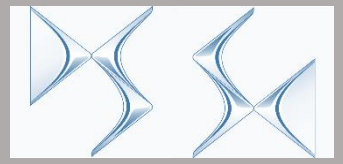
极限的需求

- 开发人员和客户一起，把各种需求变成一个个小的需求模块（User Story）；
- 这些模块又会根据实际情况被组合在一起或者被分解成更小的模块，且它们都被记录在一些小卡片（Story Card）上；
- 客户根据每个模块的商业价值来指定它们的优先级；
- 然后，开发人员确定每个需求模块的开发风险；
- 经过开发人员和客户的评估后，它们被安排在不同的开发周期里，客户将得到一个尽可能准确的开发计划；
- 客户为每个需求模块指定验收测试（功能测试）。



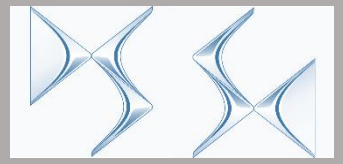
极限的设计

- 从开发的角度来看，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。
- 随着这些测试的进行，通过的单元测试也越来越多。通过这种方式，客户和开发人员都很容易检验，是否履行了对客户的承诺。
- 同时，XP还大力提倡设计复核（Review）、代码复核以及重整和优化（Refectory），所有的这些过程其实也是优化设计的过程；



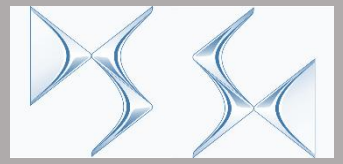
极限的编程

- XP提倡配对编程（Pair Programming），而且代码所有权是归于整个开发队伍（Collective Code Ownership）。
- 程序员在写程序和重整优化程序的时候，都要严格遵守编程规范。
- 任何人都可以修改其他人写的程序，修改后要确定新程序能通过单元测试。



极限的测试

- XP提倡在开始写程序之前先写单元测试。
- 开发人员应该经常把开发好的模块整合到一起（Continuous Integration，持续集成），每次整合后都要运行单元测试；
- 做任何的代码复核和修改，都要运行单元测试；
- 发现了BUG，就要增加相应的测试。
- 除了单元测试之外，还有整合测试，功能测试、负荷测试和系统测试等。
- 所有这些测试，是XP开发过程中最重要的文档之一，也是最终交付给用户的内容之一。



课后思考题

- 结合本章节的主要内容，各小组思考本次作业哪种模型比较合适?
- 可否采用其他模型?
- 根据已决定的模型，小组内的人员如何分配?
  - 课程作业模式
  - 实际的工程项目
- 各小组针对已了解的需求内容，请考虑技术解决方案。