南京大学
NANJING UNIVERSITY

《软件工程与计算II》

软件开发过程模型

南京大学软件学院

主要内容

- 软件开发各典型阶段
- 软件生命周期模型
- 软件过程模型

第4～21章开发活动总结

阶段	目标	关注点	工作基础	方法	主要任务		执行者	制品
需求工程	建立解决方案	理解现实；制定解决方案（成本效益比有效）	客户、用户、环境等	结构化分析（DFD、ERD等）；面向对象分析（用例图、概念类图、顺序图、状态图等）	需求开发	系统需求开发	需求工程师	需求分析模型与需求文档
						软件需求开发		
			需求基线		需求管理		所有开发人员	
软件设计	建立由抽象软件实体组成的软件结构	整体功能组织与质量特征（各种质量属性）	系统需求	模块结构（UML: 包图、构件图、部署图）	软件体系结构设计		体系结构师	体系结构原型、体系结构模型和文档
		模块内部的结构及质量（易开发、可复用、可维护等）	软件体系结构 软件需求	结构化方法（结构图）；面向对象方法（包图、类图、顺序图等）	软件详细设计		设计师	软件详细设计模型和文档
		人机交互质量（易用性）	软件需求	人机交互设计方法	人机交互设计		人机交互设计	界面原型

第4～21章开发活动总结

软件构造	构建软件	程序的质量（可读、可靠、性能、可维护等）	软件设计方案	结构化程序设计；面向对象程序设计	编程、集成、测试与调试等		程序员	源代码可执行程序
软件测试	保障软件产品的质量	质量（程序正确性和对需求的符合度）保障	软件需求 软件设计 程序代码	黑盒、白盒等测试方法	测试执行	单元测试	程序员或测试人员	测试报告 通过测试的软件产品
						集成测试		
						系统测试	测试人员	
					测试计划			
					测试报告			

第4～21章开发活动总结

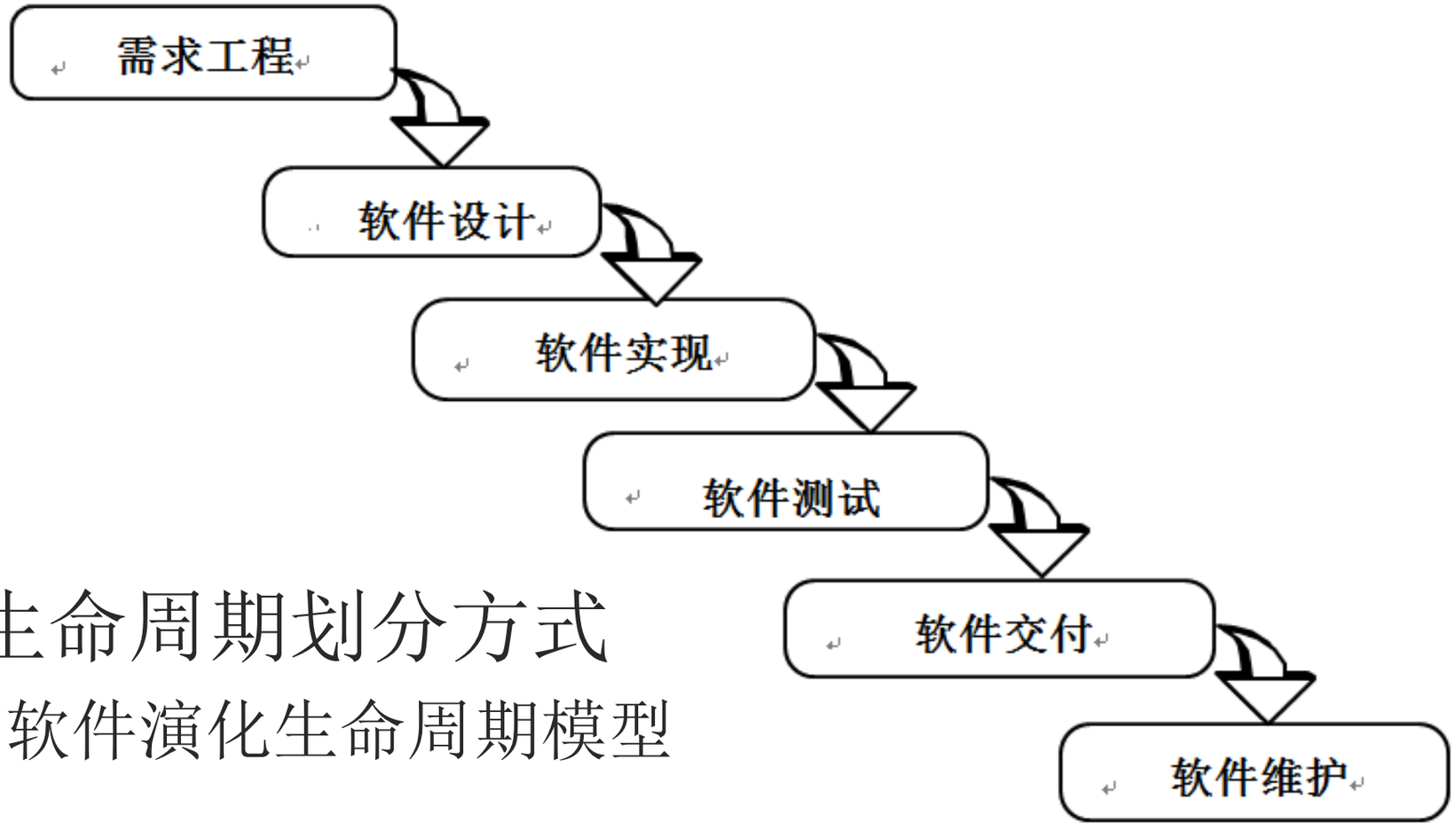
软件交付	将软件交付给用户	交付的有效性	可执行程序		安装与部署、用户培训、文档支持	专门人员	用户使用手册 已交付软件产品
软件维护	产品终结前的正常使用	变更时控制效益和质量的综合平衡	已交付软件产品	软件演化方法（重构、逆向工程、再工程）	重复上述各阶段任务	维护人员	

主要内容

- 软件开发各典型阶段
- 软件生命周期模型
- 软件过程模型

软件生命周期

- 人们将软件从生产到报废的生命周期分割为不同阶段，每段阶段有明确的典型输入/输出、主要活动和执行人，各个阶段形成明确、连续的顺次过程，这些阶段划分就被称为软件生命周期模型。
- 典型的软件生命周期



- 可以有不同的生命周期划分方式
  - 例如第21章的软件演化生命周期模型

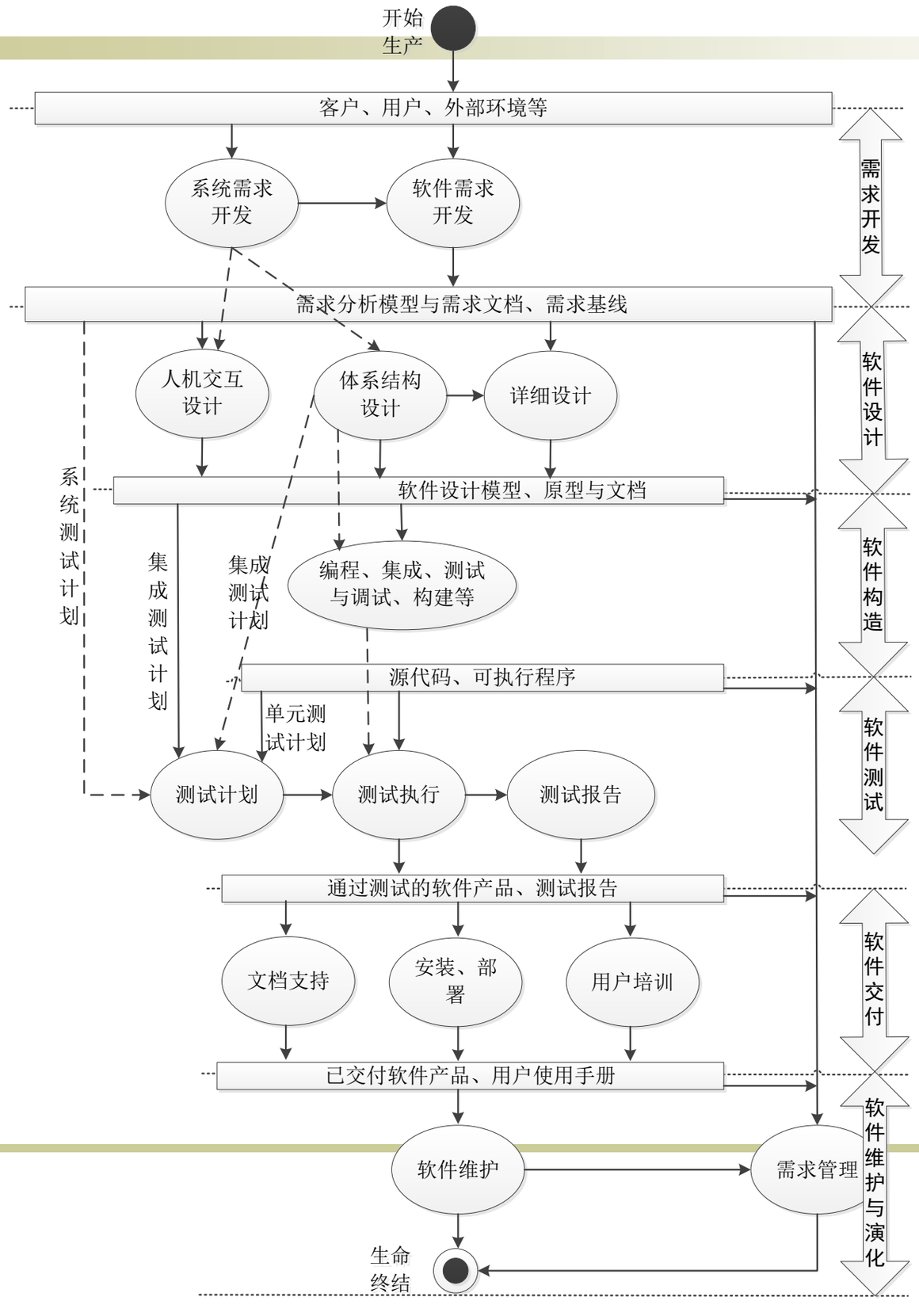
主要内容

- 软件开发各典型阶段
- 软件生命周期模型
- 软件过程模型
  - 构建-修复模型
  - 瀑布模型
  - 增量迭代模型
  - 演化模型
  - 原型模型
  - 螺旋模型
  - Rational 统一过程
  - 敏捷过程

软件过程模型

- 软件过程模型在生命周期模型的基础则进一步详细说明各个阶段的任务、活动、对象及其组织、控制过程。
- 与简略的软件生命周期模型不同，软件过程模型可以被看作是网络化的活动组织
- 不同的生命周期模型有不同的软件过程模型
  - 阶段划分不一样
- 同一个生命周期模型也会有多个不同的软件过程模型
  - 虽然阶段划分一样，但是各个阶段的时间安排、先后衔接等执行过程不一样

本书案例适用的软件过程模型

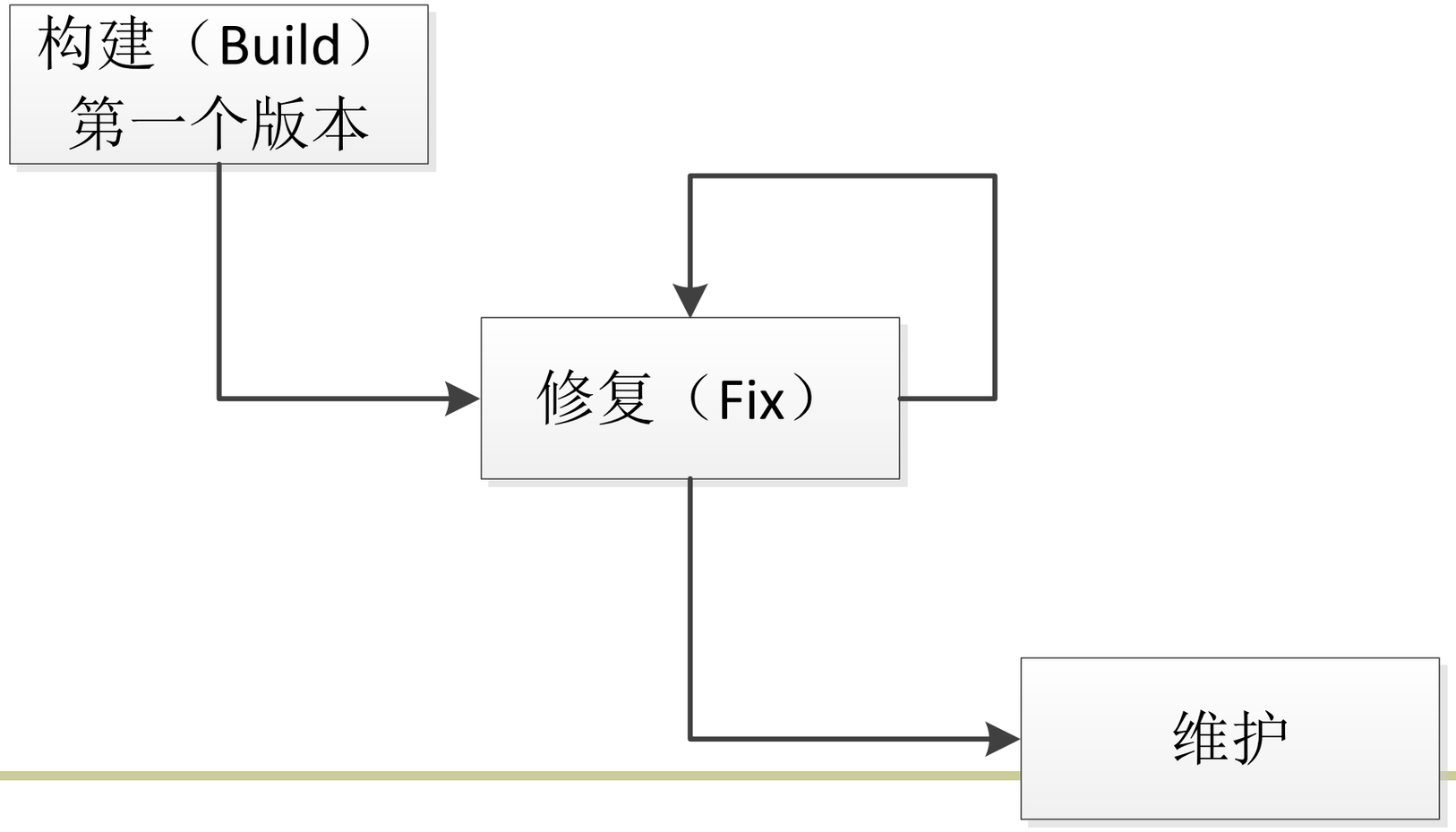


主要内容

- 软件开发各典型阶段
- 软件生命周期模型
- 软件过程模型
  - 构建-修复模型
  - 瀑布模型
  - 增量迭代模型
  - 演化模型
  - 原型模型
  - 螺旋模型
  - Rational 统一过程
  - 敏捷过程

构建-修复模型

- 最早也是最自然产生的软件开发模型
- 不能算是一个软件过程模型，因为它对软件开发活动没有任何规划和组织，是完全依靠开发人员个人能力进行软件开发的方式。



缺点

- 在这种模型中，没有对开发工作进行规范和组织，所以随着软件系统的复杂度提升，开发活动会超出个人的直接控制能力，构建-修复模型就会导致开发活动无法有效进行而失败；
- 没有分析需求的真实性，给软件开发带来很大的风险；
- 没有考虑软件结构的质量，使得软件结构在不断的修改中变得质量越来越糟，直至无法修改；
- 没有考虑测试和程序的可维护性，也没有任何文档，软件的维护十分困难。

适用性

- 软件规模很小，只需要几百行程序，其开发复杂度是个人能力能够胜任的；
- 软件对质量的要求不高，即使出错也无所谓；
- 只关注开发活动，对后期维护的要求不高，甚至不需要进行维护。

主要内容

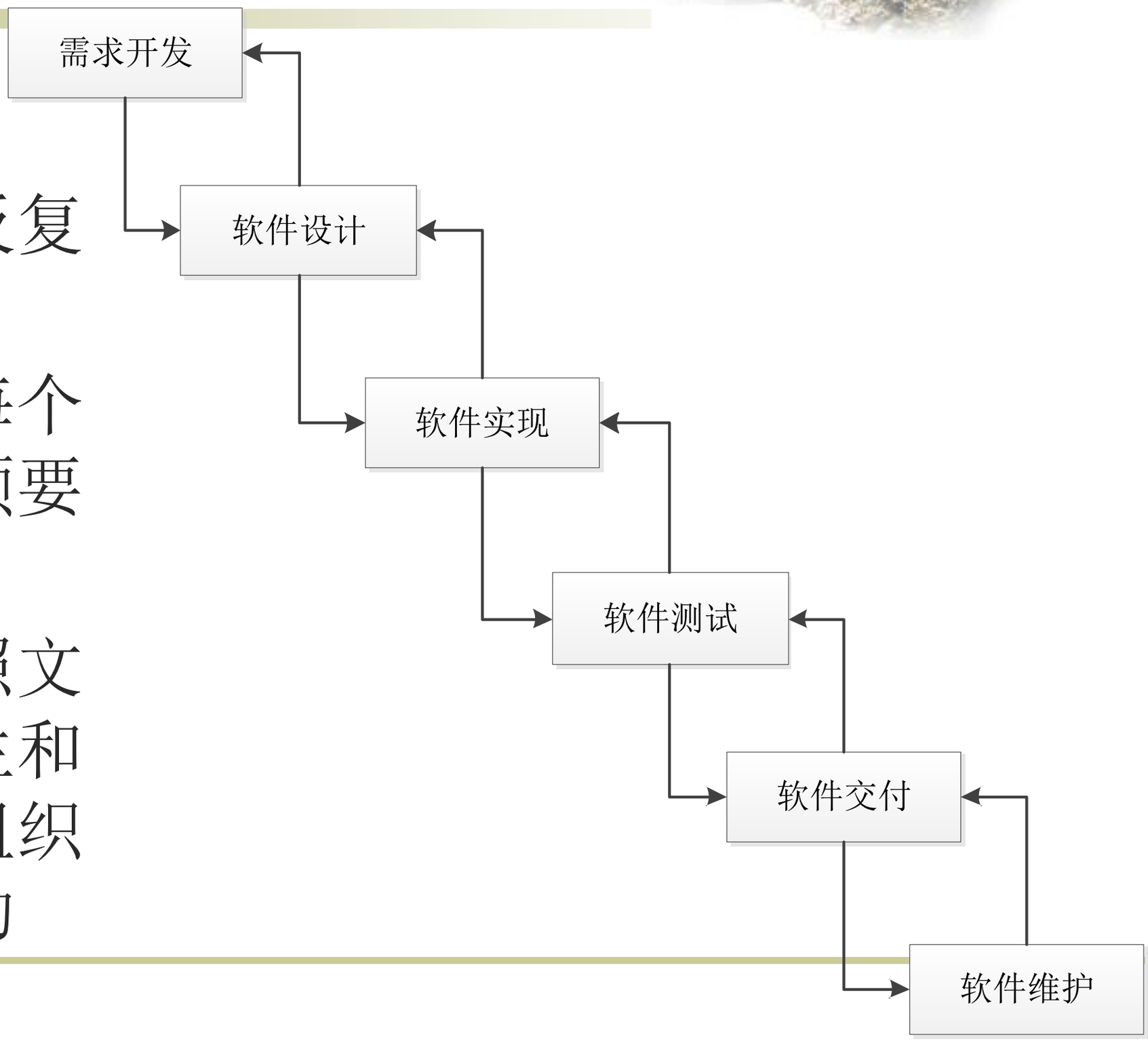
- 软件开发各典型阶段
- 软件生命周期模型
- 软件过程模型
  - 构建-修复模型
  - 瀑布模型
  - 增量迭代模型
  - 演化模型
  - 原型模型
  - 螺旋模型
  - Rational 统一过程
  - 敏捷过程

瀑布模型

- [Royce1970]通过分析早期（1960s）的软件开发活动后发现，如果将软件开发活动划分为不同的阶段，并且保障每一个阶段工作的正确性和有效性（尤其是重视分析和设计阶段），那么会取得比构建-修复方式好得多的表现，包括更高的质量、更低的成本和更小的风险。

瀑布模型

- 允许活动出现反复和迭代
- 重点在于要求每个活动的结果必须要进行验证
- 文档驱动：按照文档的划分、产生和验证来规划、组织和控制开发活动



优点

- 为软件开发活动定义了清晰的阶段划分（包括输入/输出、主要工作及其关注点），这让开发者能够以关注点分离的方式更好地进行那些复杂度超越个人能力的软件项目的开发活动。

缺点

- 对文档的过高期望具有局限性。
  - 一方面会耗费很大的工作量和成本
  - 另一方面很难为经常变化的需求建立完备可靠的文档。
- 对开发活动的线性顺序假设具有局限性。
  - 要求一个阶段的工作经过验证后才能进入后续阶段是不切实际的。在实际开发中，常常需要进行一定的后续工作才能验证当前的工作是否正确、可靠。
- 客户、用户参与具有局限性。
  - 成功的项目开发需要客户、用户从始至终的参与，而不仅仅是一个阶段。
- 里程碑粒度具有局限性。
  - 里程碑粒度过粗，基本丧失了“早发现缺陷早修复”这一思想。

适用性

- 需求非常成熟、稳定，没有不确定的内容，也不会发生变化；
- 所需的技术成熟、可靠，没有不确定的技术难点，也没有开发人员不熟悉的技术问题；
- 复杂度适中，不至于产生太大的文档负担和过粗的里程碑。

主要内容

- 软件开发各典型阶段
- 软件生命周期模型
- 软件过程模型
  - 构建-修复模型
  - 瀑布模型
  - 增量迭代模型
  - 演化模型
  - 原型模型
  - 螺旋模型
  - Rational 统一过程
  - 敏捷过程

增量迭代模型

- 开发复杂系统都需要迭代完成
- 软件规模日益增长带来了挑战及其对策
  - 周期过长-渐进交付
  - 时间压力-并行开发
- 迭代式、渐进交付和并行开发共同促使了增量迭代模型的产生和普及。

增量迭代模型

- 增量迭代模型需要在项目早期就确定项目的目标和范围，项目需求要比较成熟和稳定
- 少量的不确定性和影响不大的需求变更通过迭代的方式加以解决
- 每个迭代的增量需求相对独立，被开发为产品的独立部分交付给用户
- 第一个迭代完成的往往是产品的核心部门，满足基本的需求
- 需求驱动：增量需求是增量迭代模型进行迭代规划、开发活动组织和控制的主要依据

优点

- 迭代式开发更加符合软件开发的实践情况，具有更好的适用性；
- 并行开发可以帮助缩短软件产品的开发时间；
- 渐进交付可以加强用户反馈，降低开发风险。

缺点

- 由于各个构件是逐渐并入已有的软件体系结构中的，所以加入构件必须不破坏已构造好的系统部分，这需要软件具备开放式的体系结构。
- 增量交付模型需要一个完备、清晰的项目前景和范围以进行并发开发规划，但是在一些不稳定的领域，不确定性太多或者需求变化非常频繁，很难在项目开始就确定前景和范围。

适用性

- 因为能够很好地适用于大规模软件系统开发，所以增量迭代模型在实践中有着广泛的应用，尤其是比较成熟和稳定的领域。

主要内容

- 软件开发各典型阶段
- 软件生命周期模型
- 软件过程模型
  - 构建-修复模型
  - 瀑布模型
  - 增量迭代模型
  - 演化模型
  - 原型模型
  - 螺旋模型
  - Rational 统一过程
  - 敏捷过程

演化模型

- 演化模型与增量迭代模型相比
  - 都是迭代、并行开发和渐进交付，都适合大规模软件开发
- 演化模型能够更好地应对需求变更，更适用于需求变更比较频繁或不确定性较多的领域。
  - Ch21：演化模糊了维护与新开发的界限

优点

- 使用了迭代式开发，具有更好的适用性，尤其是其演化式迭代安排能够适用于那些需求变更比较频繁或不确定性较多的软件系统的开发；
- 并行开发可以帮助缩短软件产品的开发时间；
- 渐进交付可以加强用户反馈，降低开发风险。

缺点

- 无法在项目早期阶段建立项目范围，所以项目的整体计划、进度调度、尤其是商务协商事宜无法准确把握；
- 后续迭代的开发活动是在前导迭代基础上进行修改和扩展的，这容易让后续迭代忽略设分析与设计工作，蜕变为构建-修复方式。

适用性

- 在实践中，不稳定领域的大规模软件系统开发适合使用演化模型进行组织。

主要内容

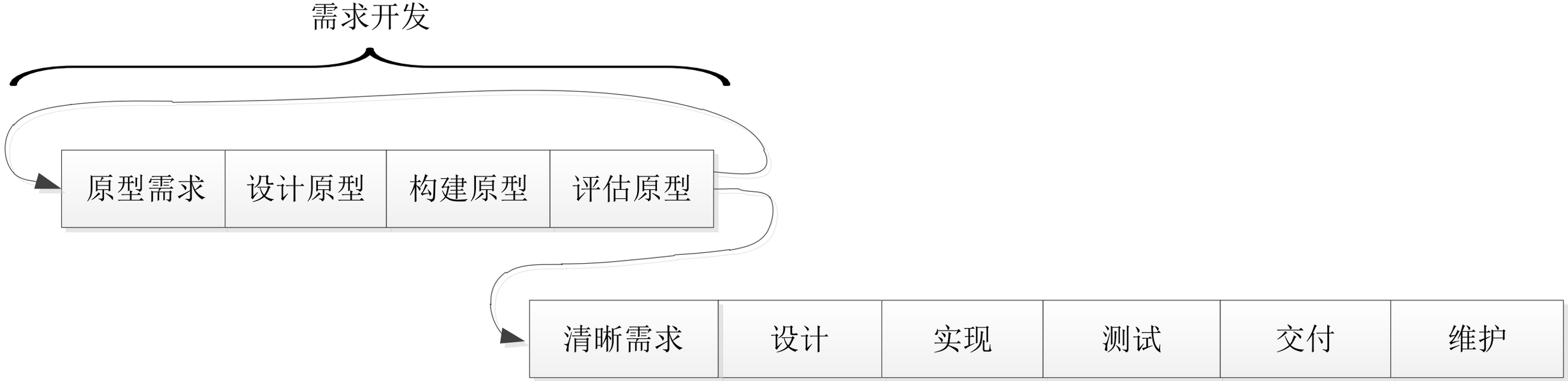
- 软件开发各典型阶段
- 软件生命周期模型
- 软件过程模型
  - 构建-修复模型
  - 瀑布模型
  - 增量迭代模型
  - 演化模型
  - 原型模型
  - 螺旋模型
  - Rational 统一过程
  - 敏捷过程

原型模型

- 抛弃式原型
  - 它通过模拟“未来”的产品，将“未来”的知识置于“现在”进行推敲，解决不确定性。
- 演化式原型
  - 在迭代中构建，是系统的核心，并不断扩充，最终成为真正的软件产品。

原型模型

- 大量使用抛弃式原型解决需求不确定性的过程模型
- 在抛弃式原型的帮助下，解决了不确定性之后，可以得到清晰的需求，此时再按照瀑布模型方式安排后续开发活动



- 原型模型的基本特征是注重使用抛弃式原型，适用于不确定性较多的软件开发

优点

- 对原型方法的使用加强了与客户、用户的交流，可以让最终产品取得更好的满意度；
- 适用于非常新颖的领域，这些领域因为新颖所以有着大量的不确定性。

缺点

- 原型方法能够解决风险，但是自身也能带来新的风险，例如原型开发的成本较高，可能会耗尽项目的费用和时间；
- 实践中，很多项目负责人不舍得抛弃“抛弃式原型”，使得质量较差的代码进入了最终产品，导致了最终产品的低质量。

适用性

- 实践中，原型模型主要用于在有着大量不确定性的新颖领域进行开发活动组织。

主要内容

- 软件开发各典型阶段
- 软件生命周期模型
- 软件过程模型
  - 构建-修复模型
  - 瀑布模型
  - 增量迭代模型
  - 演化模型
  - 原型模型
  - 螺旋模型
  - Rational 统一过程
  - 敏捷过程

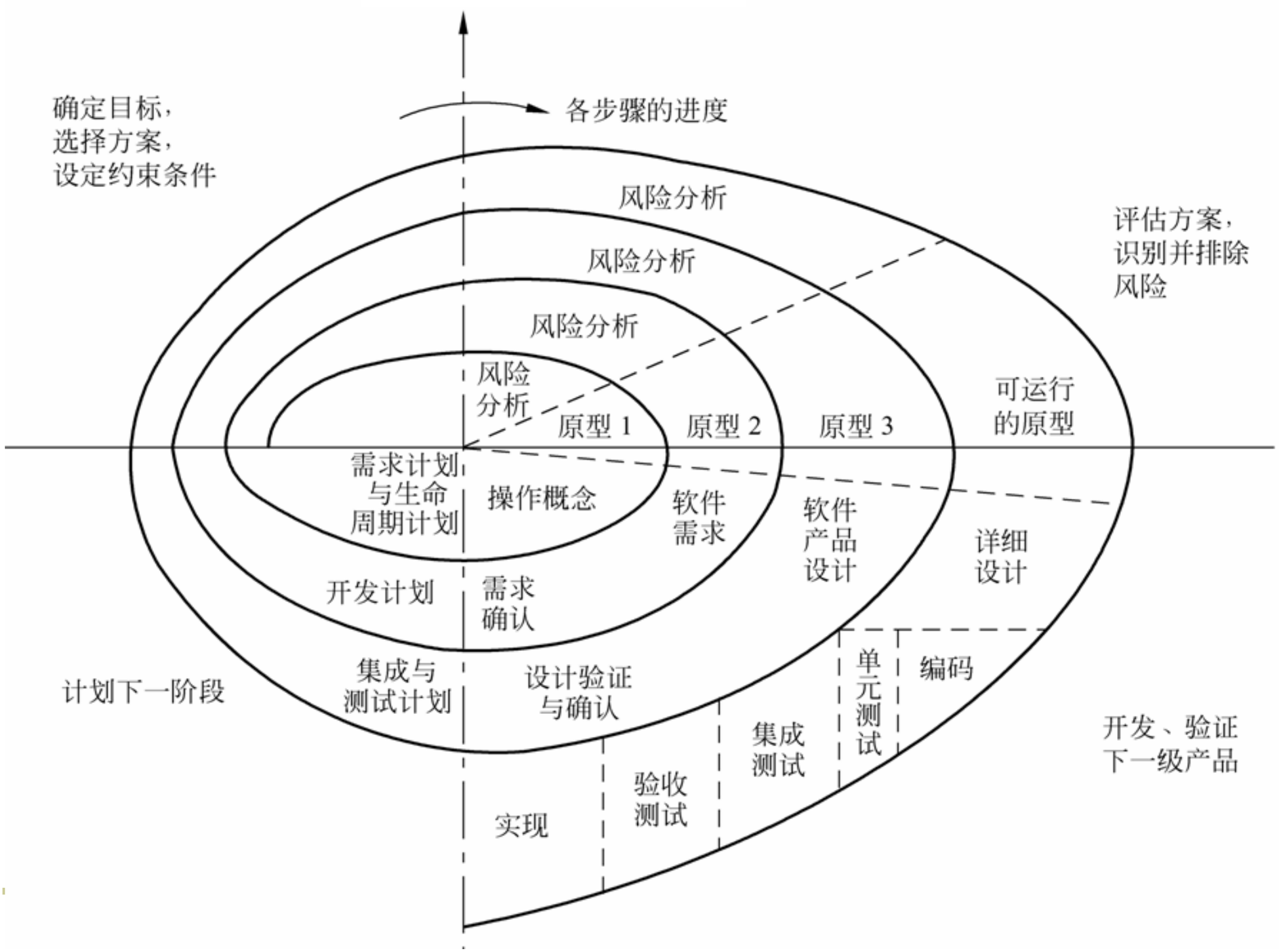
螺旋模型

- 风险是指因为不确定性（对未来知识了解有限）而可能给项目带来损失的情况
- 螺旋模型的基本思想是尽早解决比较高的风险，如果有些问题实在无法解决，那么早发现比项目结束时再发现要好，至少损失要小得多。
- 原型能够澄清不确定性，所以原型能够解决风险，螺旋模型是充分利用原型方法来解决风险的。

螺旋模型

- 风险驱动，完全按照风险解决的方式组织软件开发活动。
  - 确定目标、解决方案和约束
  - 评估方案，发现风险
  - 寻找风险解决方法
  - 落实风险解决方案
  - 计划下一个迭代

过程描述



迭代与瀑布的结合

- 开发阶段是瀑布式的
- 风险分析是迭代的

螺旋模型

- 原型模型 vs 螺旋模型
  - 原型模型：使用原型解决需求的不确定性
  - 螺旋模型：实用原型解决项目开发中常见的各种类型的技术风险，包括系统需求开发、软件需求开发、软件体系结构设计、详细设计等各个阶段

螺旋模型

- 优点
  - 可以降低风险，减少项目因风险造成的损失。
- 缺点
  - 风险解决需要使用原型手段，也就会存在原型自身带来的风险，这一点与原型模型相同；
  - 模型过于复杂，不利于管理者依据其组织软件开发活动；
- 适用性
  - 在实践中，螺旋模型在高风险的大规模软件系统开发中有着较多的应用。

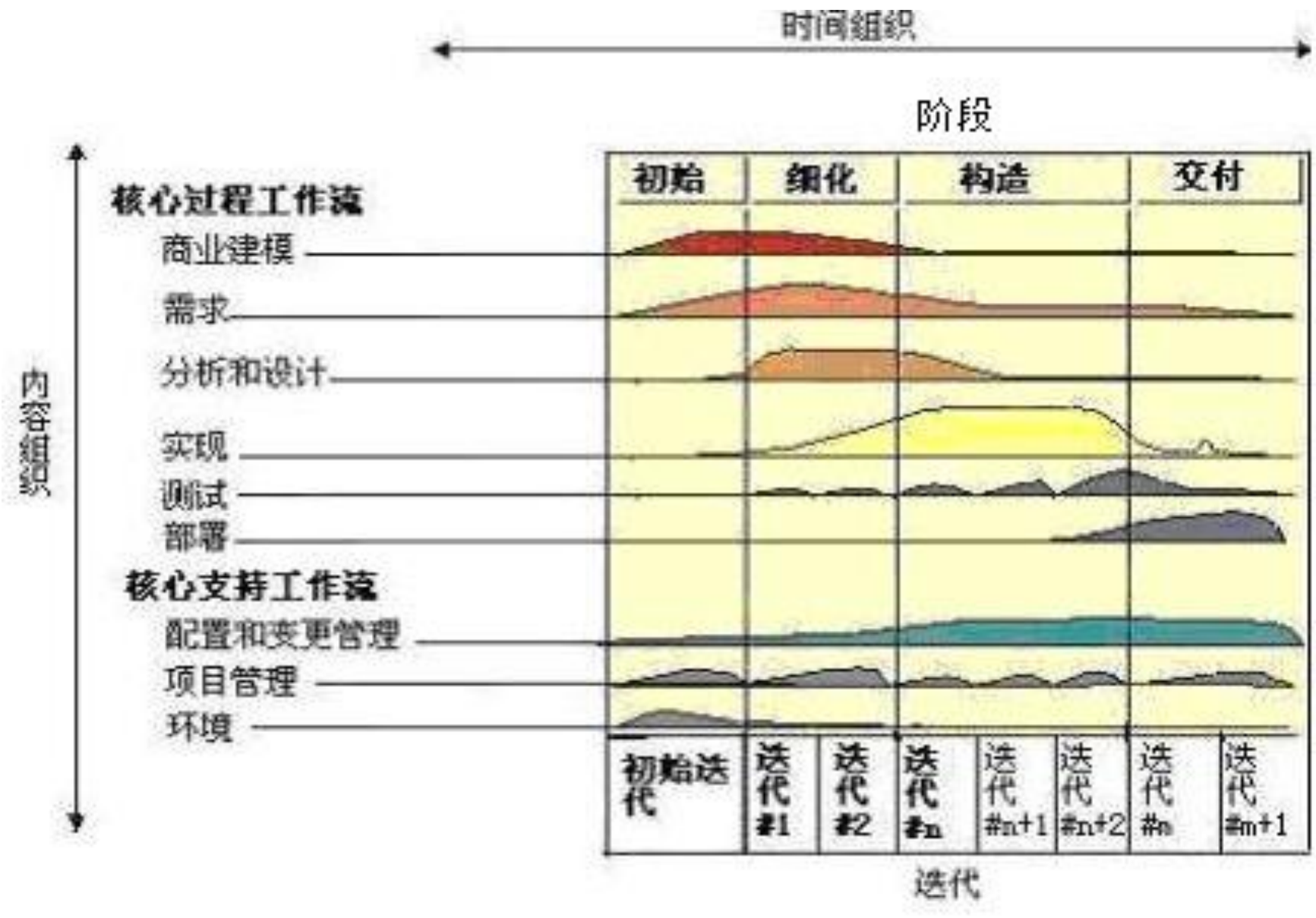
主要内容

- 软件开发各典型阶段
- 软件生命周期模型
- 软件过程模型
  - 构建-修复模型
  - 瀑布模型
  - 增量迭代模型
  - 演化模型
  - 原型模型
  - 螺旋模型
  - Rational 统一过程
  - 敏捷过程

Rational统一过程模型

- 统一过程（Rational Unified Process，RUP）总结和借鉴传统上的各种有效经验，建立最佳实践方法的集合，并提供有效的过程定制手段，允许开发者根据特定的需要定制一个有效的过程模型。

Rational统一过程模型



RUP核心实践方法

1、迭代式开发，这是过去被反复证明的最佳实践方法；

2、管理需求，重视需求工程中除了需求开发之外的需求管理活动；

3、使用基于组件的体系结构，它帮助建立一个可维护、易开发、易复用的软件体系结构；

4、可视化建模，利用UML进行建模；

5、验证软件质量，尽早和持续地开展验证，以尽早发现缺陷，降低风险和成本；

6、控制软件变更，适应1990s以后需求变更越来越重要的事实。

RUP裁剪

- RUP是一个通用的过程模板，在一个项目使用RUP指导开发活动组织时，需要对RUP进行裁剪和配置。

  1、确定本项目需要哪些工作流。RUP的9个核心工作流并不总是需要的，可以取舍。

  2、确定每个工作流需要哪些制品。

  3、确定4个阶段之间如何演进，决定每个阶段要哪些工作流，每个工作流执行到什么程度，制品有哪些。

  4、确定每个阶段内的迭代计划。

  5、规划工作流的组织，这涉及人员、任务及制品，通常用活动图的形式给出。

优点

- 吸收和借鉴了传统上的最佳实践方法，尤其是其核心的6个实践方法，能够保证软件开发过程的组织是基本有效和合理的。
- RUP依据其定制机制的不同，可以适用于小型项目，也可以适用于大型项目的开发，适用面广泛。
- RUP有一套软件工程工具的支持，这可以帮助RUP的有效实施。

缺点

- 没有考虑交付之后的软件维护问题；
- 裁剪和配置工作不是一个简单的任务，无法保证每个项目都能定制一个有效的RUP过程。

适用性

- RUP是重量级过程，能够胜任大型软件团队开发大型项目时的活动组织。但RUP经过裁剪和定制，也可以变为轻量级过程，也能够胜任小团队的开发活动组织。

主要内容

- 软件开发各典型阶段
- 软件生命周期模型
- 软件过程模型
  - 构建-修复模型
  - 瀑布模型
  - 增量迭代模型
  - 演化模型
  - 原型模型
  - 螺旋模型
  - Rational 统一过程
  - 敏捷过程

敏捷过程

- 传统的过程模型在应对需求变更和重视用户价值等发展时遇到了极大的挑战。
  - 传统的软件过程模型过度强调了“纪律”，忽视了个人的能力，尤其是过度强调了计划、文档和工具，导致项目开发失去了灵活性，不能快速地应对需求变更和用户反馈。
- 针对传统过程模型的缺陷和新的形势，人们总结实践中的经验和最佳实践方法，尝试建立轻量级过程方法。
- 2001年，企业界一些对轻量级过程方法有着共同思想的人士召开了一次会议，会议上宣布成立敏捷联盟，敏捷过程方法正式诞生。

敏捷思想

- 敏捷过程并不是要为软件开发活动组织提供一种特定的过程模型，而是倡导一些指导性的思想和原则
- 最为重要的敏捷思想是敏捷联盟宣言所声明的价值观：
  - 个体和互动 高于 流程和工具
  - 工作的软件 高于 详尽的文档
  - 客户合作 高于 合同谈判
  - 响应变化 高于 遵循计划
- 也就是说，尽管右项有其价值，敏捷方法更重视左项的价值。

敏捷原则

1. 我们最重要的目标，是通过持续不断地及早交付有价值的软件使客户满意。
2. 欣然面对需求变化，即使在开发后期也一样。为了客户的竞争优势，敏捷过程掌控变化。
3. 经常地交付可工作的软件，相隔几星期或一两个月，倾向于采取较短的周期。
4. 业务人员和开发人员必须相互合作，项目中的每一天都不例外。
5. 激发个体的斗志，以他们为核心搭建项目。提供所需的环境和支援，辅以信任，从而达成目标。
6. 不论团队内外，传递信息效果最好效率也最高的方式是面对面的交谈。

敏捷原则

7. 可工作的软件是进度的首要度量标准。

8. 敏捷过程倡导可持续开发。责任人、开发人员和用户要能够共同维持其步调稳定延续。

9. 坚持不懈地追求技术卓越和良好设计，敏捷能力由此增强。

10. 以简洁为本，它是极力减少不必要工作量的艺术。

11. 最好的架构、需求和设计出自自组织团队。

12. 团队定期地反思如何能提高成效，并依此调整自身的举止表现。

践行敏捷思想与原则的过程方法

- 极限编程XP（eXtreme Programming）
- Scrum
- 特性驱动开发（FDD/Feature Driven Development）
- 自适应软件开发（ASD/Adaptive Software Development）
- 动态系统开发方法（DSDM/Dynamic Systems Development Method）
- ……

极限编程

- 极限编程XP的一个重要思想是极限利用简单、有效的方法解决问题（这也是它被称为极限编程的原因），例如：
  - 如果单元测试好用，那么就让所有人一直做单元测试（测试驱动）；
  - 如果集成测试好用，那么就一直做集成测试（持续集成）；
  - 如果代码评审好用，那么就一直做评审（结对编程）；
  - 如果简洁性好用，那么就只做最简洁的事情（简单设计）；
  - 如果设计好用，那么就一直设计（重构）；
  - 如果短迭代好用，那么就把迭代做的足够小（小版本发布）；
  - 如果用户参与好用，那么就让用户始终参与（现场客户）。
  - ……

极限编程的实践方法

开发活动	实践方法	方法描述
迭代规划	规划游戏	计划是持续的、循序渐进的。每2周，开发人员就为下2周估算候选特性的成本，而客户则根据成本和商务价值来选择要实现的特性
需求开发	现场客户	用户代表作为开发团队的一份子，始终参与软件开发活动
软件设计	系统隐喻	将整个系统联系在一起的全局视图；它是系统的未来影像，是它使得所有单独模块的位置和外观变得明显直观。如果模块的外观与整个隐喻不符，那么你就知道该模块是错误的
	简单设计	保持设计简洁，满足需求，但不要包含为了未来预期而进行的设计
	重构	不改变系统外部行为的情况下，改进软件内部结构的质量

极限编程的实践方法

开发活动	实践方法	方法描述
软件实现	结对编程	两个人坐在一台电脑前一起编程，一个程序员控制电脑进行编程时，另外一个人进行代码评审。编程控制权可以互换
	编码规范	所有人都遵循一个统一的编程标准，因此，所有的代码看起来好像是一个人写的，每个程序员更容易读懂其他人写的代码
	代码集体所有权	每个人都对所有的程序负责，每个人都可以更改程序的任意部分
软件测试	测试驱动	在编程之前，先写好程序的设计用例和测试框架，然后再编写程序
	持续集成	频繁地进行系统集成，每次集成都要通过所有的单元测试；每个用户任务完成后都应该进行集成
软件交付	小版本发布	频繁地发布软件，如果有可能，应该每天都发布一个新版本；在完成任何一个改动、集成或者新需求后，就应该立即发布一个新版本
其他	每周 40 小时工作制	保持团队可持续开发能力，长时间加班工作会降低开发的质量和效率

特点

- 敏捷过程包含的方法众多，各有特点，除了共同的思想和原则之外，很难准确描述它们的共同点，所以也无法确切界定它们的优缺点。

适用性

- 从敏捷联盟声明的思想和原则来看，它们反映了1990s之后软件工程的发展趋势，所以得到了广泛的应用，尤其是能够适应于快速变化或者时间压力较大的项目。

总结

- 软件生命周期模型和过程模型都是人们关于如何组织软件开发活动的有效经验总结
- 不同的过程模型适用于不同情况软件项目的开发活动组织
  - 构建-修复模型
  - 瀑布模型
  - 增量迭代模型
  - 演化模型
  - 原型模型
  - 螺旋模型
  - Rational 统一过程
  - 敏捷过程