SOFTWARE TESTING

Chapter 1

Overview of Software Engineering

School of Data & Computer Science
Sun Yat-sen University

Approaches & Technologies

OUTLINE

软件生命周期的概念

- 计算机软件有一个孕育、诞生、成长、成熟、衰亡的生存过程，这样的过程称为软件的生命周期 (也称软件开发生命周期 SDLC)。软件生命周期将软件开发过程划分为若干阶段，每个阶段有明确的任务目标和运行机制，从而使复杂的软件开发过程能够得到适当的控制和管理。
- 软件生命周期一般包括可行性分析与计划、需求分析、设计 (概要设计和详细设计)、编码实现、测试、运行与维护等活动。这些活动应当以适当的方式分配到不同的阶段去完成。
  - GB/T 8567-2006：《计算机软件文档编制规范》

- 软件生命周期的6个阶段
  - 可行性分析与计划阶段
    - 确定软件开发的总体目标，给出功能、性能、可靠性以及接口等方面的要求，进行可行性分析。
    - 估计可利用的开发资源 (硬件、软件、人力等)、成本、效益、开发进度，进行投资-收益分析，制订开发计划。
    - 提交可行性分析报告、开发计划等文档。
  - 需求分析阶段
    - 分析用户提出的要求，给出用户需求详细定义，确定软件系统的各项功能、性能需求和设计约束，确定对文档编制的要求。
    - 提交软件需求说明、软件规格说明、数据要求说明等文档和初步的用户手册。

- 软件生命周期的6个阶段
  - 设计阶段
    - 概要设计/逻辑设计：把各项软件需求转换成软件的体系结构。结构中的每一个组成部分意义明确，并和某些需求相对应。
    - 详细设计/物理设计：对按照概要设计分解的每个模块所要完成的工作进行具体的描述，提供源程序代码编写的直接依据。
    - 提交概要结构设计说明书、详细设计说明书和测试计划初稿等文档。
  - 实现阶段
    - 完成源程序的编码、编译 (或汇编) 和运行调试，得到没有语法错误的程序清单。程序结构良好、清晰易读，且与设计相一致。
    - 编写进度日报、周报和月报 (取决于项目的重要性和规模)。
    - 编制测试计划。
    - 提交用户手册、操作手册等面向用户的文档。

- 软件生命周期的6个阶段
  - 测试阶段
    - 全面测试目标软件系统，并检查审阅已编制的文档，提交测试分析报告。逐项评价所实现的程序、文档以及开发工作本身，提交项目开发总结报告。
  - 运行与维护阶段
    - 软件提交给用户后，在运行使用中得到持续维护，根据用户新提出的需求进行必要而且可能的扩充、删改、更新和升级。
    - 软件维护包括改正性维护 (发现错误)、适应性维护 (适应运行环境变化) 和完善性维护 (增强功能)。
  - 软件生命周期的前五个阶段也合称开发阶段。在整个开发阶段，开发团队需要按月编写开发进度月报。

软件生命周期模型

软件生命周期模型

软件生命周期模型的概念

- 软件生命周期模型，也称软件开发过程模型 (SDM)，它从软件需求定义开始直至软件经使用后废弃为止，描述软件生命周期各个阶段的联系，是跨越整个软件生存期的软件开发、运行和维护所实施的全部过程、活动和任务的结构框架，也是对软件开发实际过程的抽象。SDM 包括构成软件过程的各种活动、生成的软件工件 (Artifact) 以及参与角色等，清晰、直观地表达了软件开发全过程。
- 软件开发过程一般应该采用某种类型的软件生命周期模型，按照一定的开发方法，使用相应的工具系统实现。

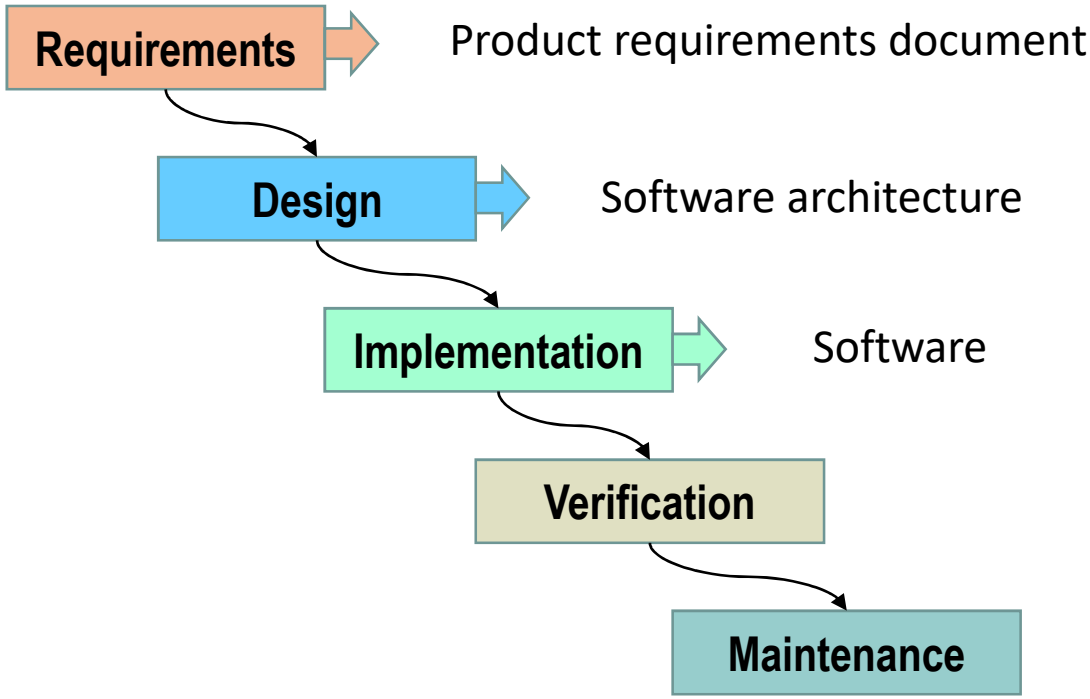
软件生命周期模型

- 软件生命周期常见模型包括：
  - 瀑布模型 (Waterfall Model)
  - V-W 模型 (V Model and W Model)
  - 快速应用开发模型 (RAD Model)
  - 原型模型 (Prototype Model)
  - 增量/演化/迭代模型 (Incremental Model)
  - 螺旋模型 (Spiral Model)
  - 喷泉模型 (Fountain Model, Object-Oriented Model)
  - 基于构件的开发模型 (CBSD Model)
  - Rational 统一过程模型 (RUP Model)
  - 敏捷开发模型与极限编程 (Agile Modeling and XP)

软件生命周期模型

瀑布模型

- Waterfall Model, *Winston Royce* 1970

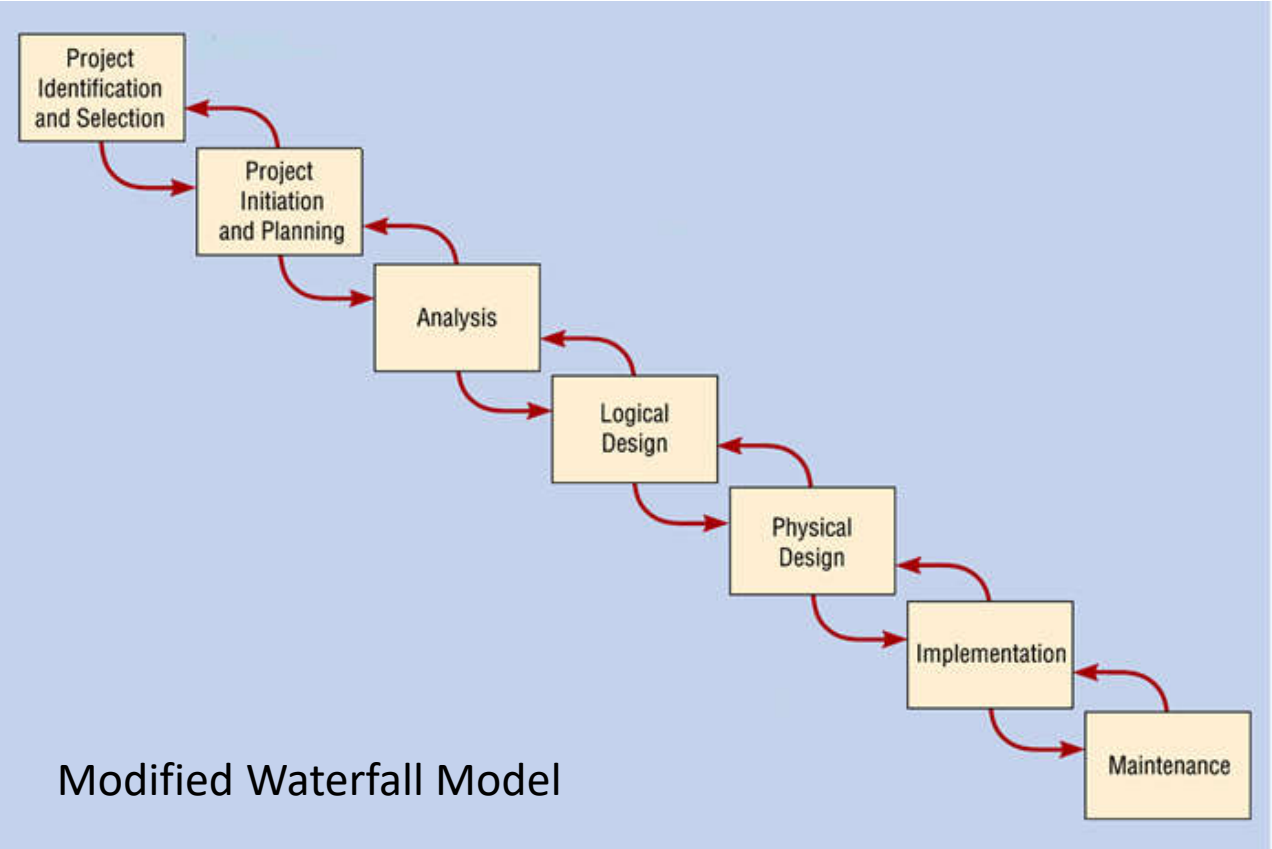


Unmodified Waterfall Model

软件生命周期模型

瀑布模型

- Waterfall Model, *Winston Royce* 1970



Modified Waterfall Model

瀑布模型

文档驱动

- 瀑布模型中的每一个开发活动具有下列特征：
  - 本阶段活动的工作对象来自于上一项活动的输出，这些输出一般是代表本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。
  - 本阶段活动产出相关的软件工件，作为下一阶段活动的输入。
  - 对本阶段活动执行情况进行评审。

瀑布模型

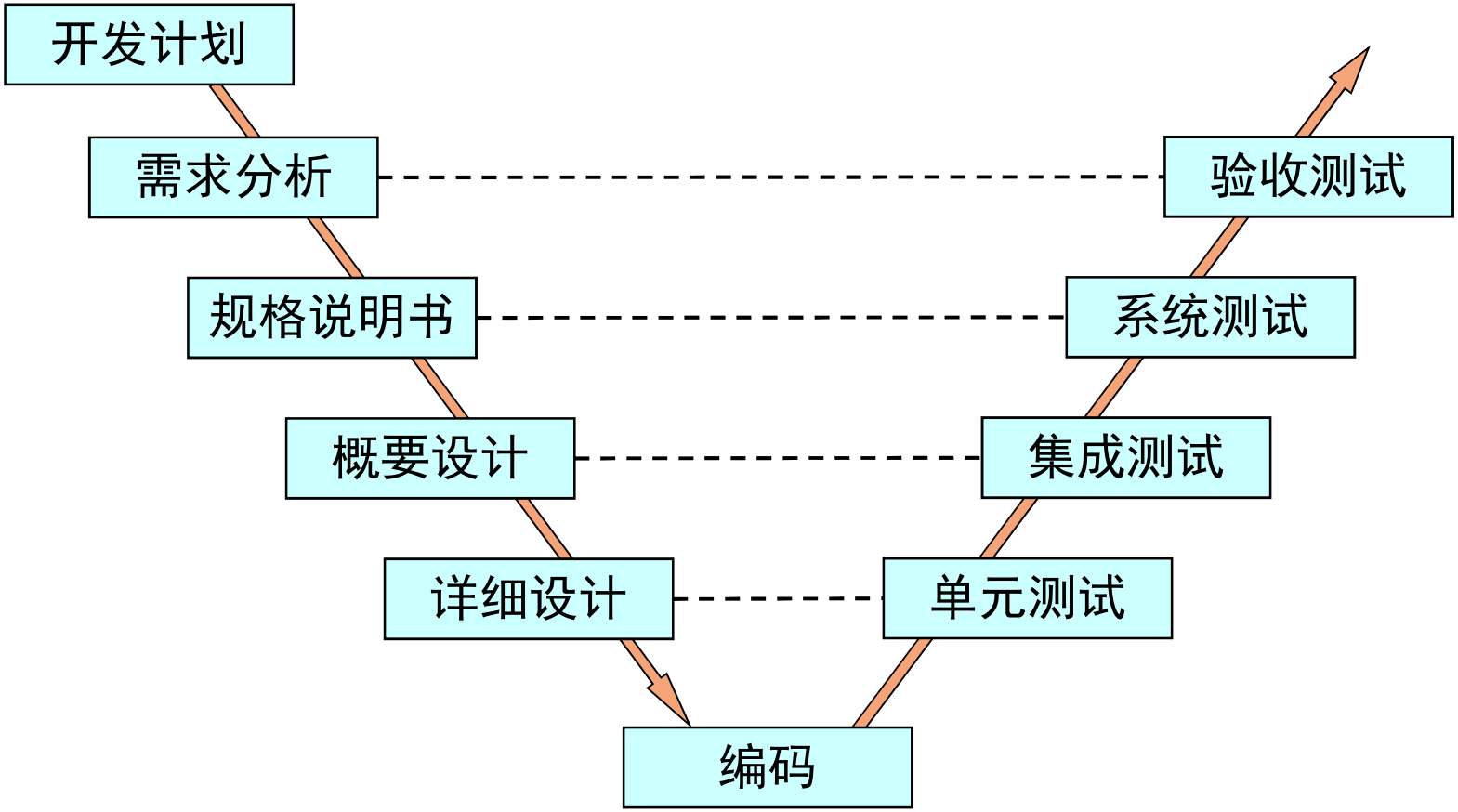
- 瀑布模型的优点：
  - 降低软件开发的复杂程度，提高软件开发过程的透明性，提高软件开发过程的可管理性。
  - 推迟软件实现，强调在软件实现前必须进行分析和设计工作。
  - 以项目的阶段评审和文档控制为手段有效地对整个开发过程进行指导，保证了阶段之间的正确衔接，能够及时发现并纠正开发过程中存在的缺陷，使产品达到预期的质量要求。

瀑布模型

- 瀑布模型的缺点：
  - 强调过程活动的线性顺序。
  - 缺乏灵活性，尤其无法解决软件需求不明确或不准确的问题。
  - 风险控制能力较弱。
  - 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。
  - 管理人员如果仅仅以文档的完成情况来评估项目完成进度，往往会产生错误的结论。

V 模型

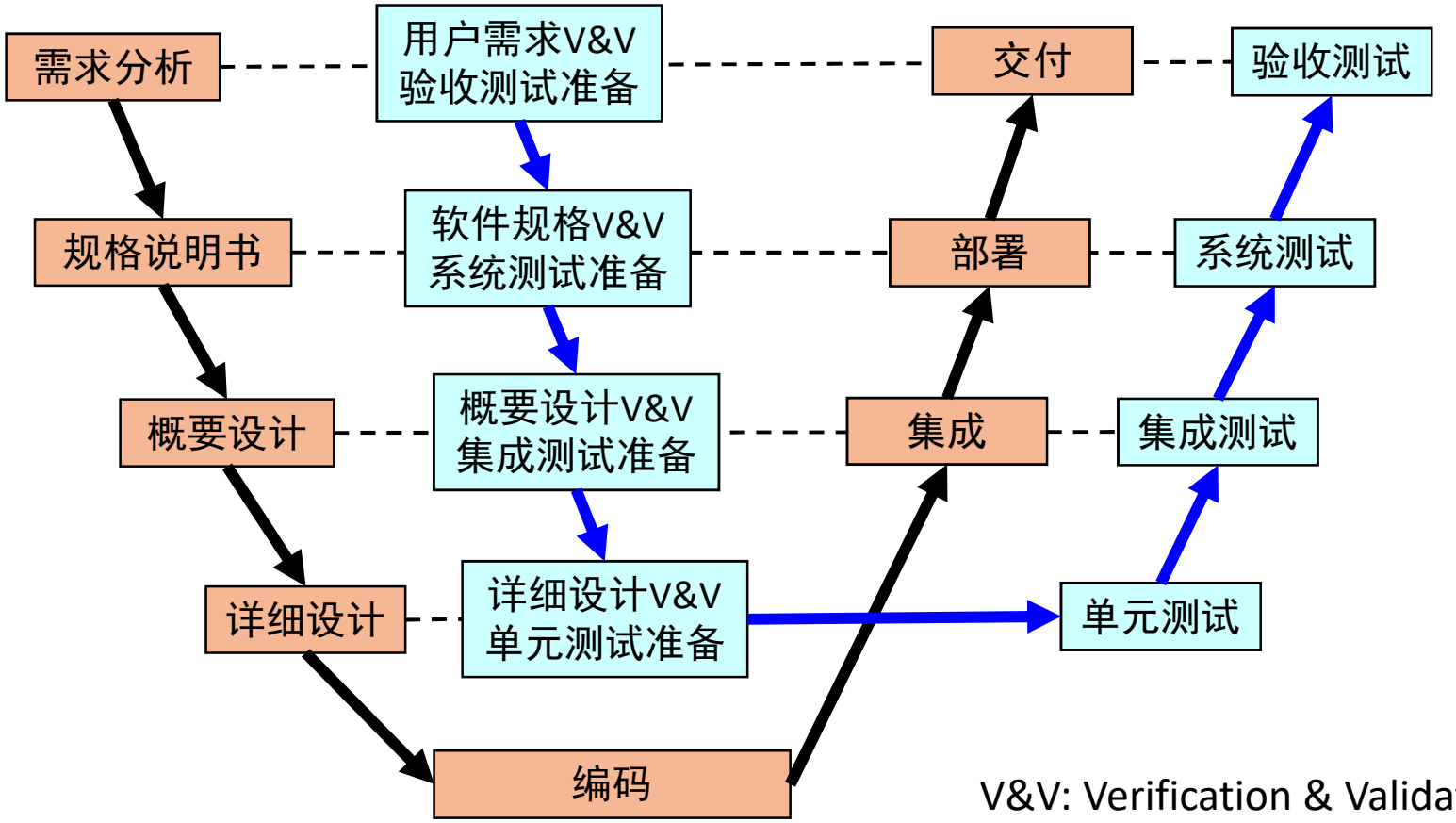
- V Model, *Paul Rook* 1980



- V 模型
  - V 模型的主要贡献
    - V 模型以测试为中心，明确标明了测试过程中存在的不同级别，并且清楚地描述了这些测试级别和软件生命周期各阶段的对应关系，为每一个阶段指定了相应的测试级别。
  - V 模型的局限性
    - 把测试作为编码之后的最后一个活动，需求分析等前期阶段产生的错误直到后期的验收测试才能被发现，无法体现“尽早地和不断地进行软件测试”的原则，大幅度提高了修补需求缺陷的成本。

W 模型

- W Model, Evolutif Co.



V&V: Verification & Validation

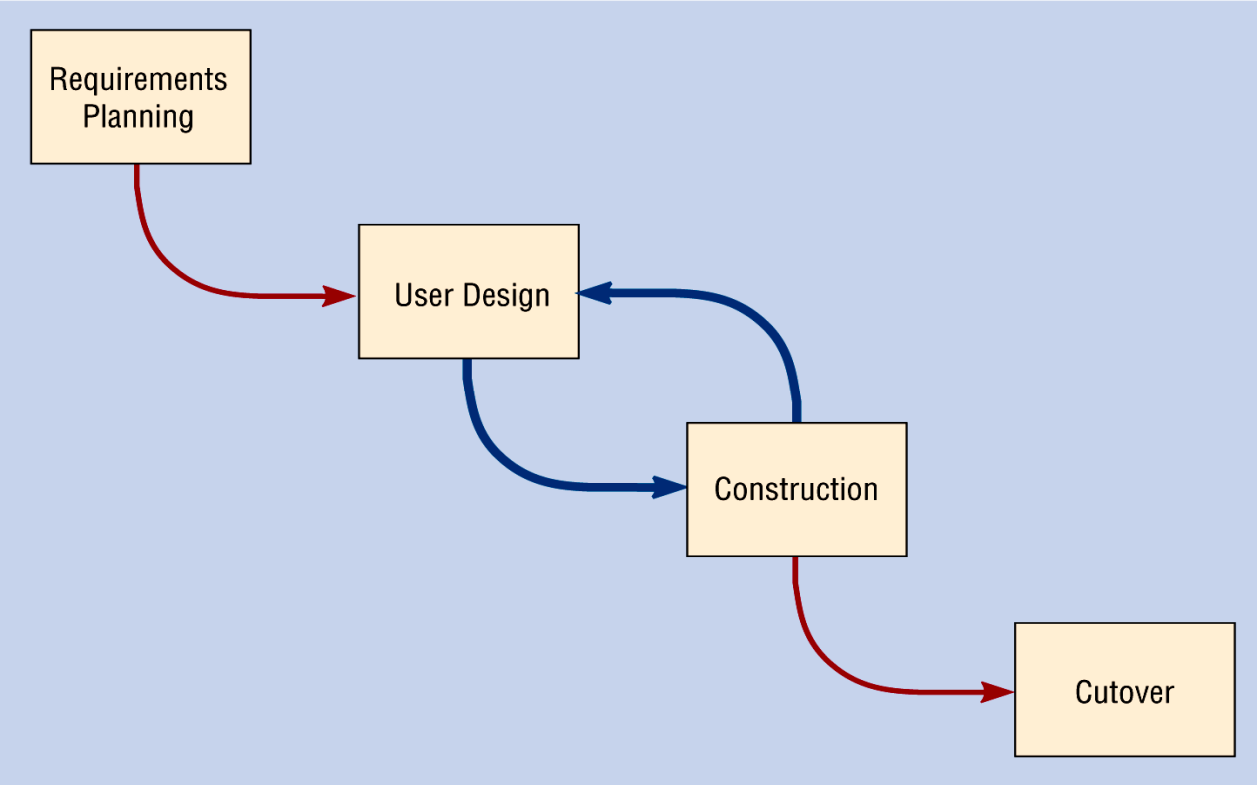
W 模型

- W 模型是 V 模型的演进
  - 在 V 模型中增加软件各开发阶段同步对应进行的测试。
  - W 模型中开发是一个“V”型，测试是与此并行的另一个“V”型。基于“尽早地和不断地进行软件测试”的原则，在软件的需求和设计阶段的测试活动遵循 IEEE1012-1998《软件验证与确认 (V&V)》。
  - 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，同样需要测试需求、功能和设计。测试与开发是同步进行的，从而有利于尽早地发现问题。

- W 模型
  - W 模型的局限性
    - W 模型和 V 模型都把软件的开发视为需求、设计、编码等一系列按时间顺序串行的活动，同时，测试和开发活动也保持着一种线性的前后关系，无法支持迭代、自发性以及需求功能的变更调整。

快速应用开发模型

- RAD Model, *James Martin* 1991



快速应用开发模型

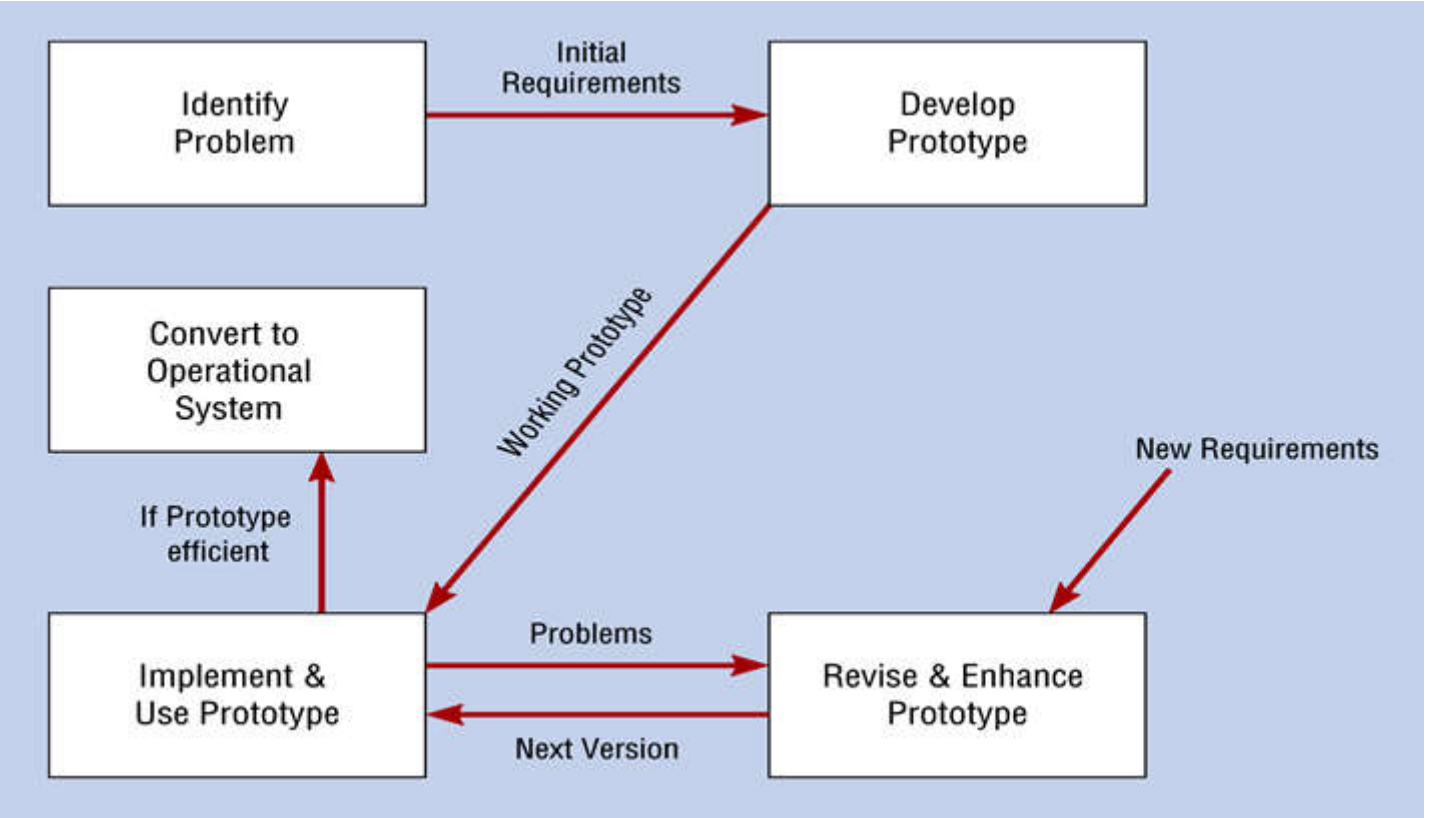
- 快速应用开发 (Rapid Application Development, RAD) 模型是一个增量型的软件开发过程模型，强调极短的开发周期。
  - RAD 过程使得一个开发小组能够在很短时间内 (比如60-90天) 创建出一个“功能完善的系统”。为了能够较快地开发以及便于系统维护，RAD 方法需要在功能和效率之间进行折衷。
- RAD 模型是线性顺序模型的一个“高速”变种，通过使用构件的建造方法赢得开发的速度。
  - RAD 过程强调复用，即复用已有的或开发可复用的构件。RAD 通常适合技术难度较低、可利用构件丰富、实时不敏感、低耦合的软件系统开发 (比如基于数据库的 MIS)，并不一定适用于其他类型的软件系统的开发流程。

快速应用开发模型

- RAD 模型的优点：
  - 开发速度快，同时质量有一定保证。
  - 对管理信息系统的开发特别有效。
- RAD 模型的缺点：
  - 应用局限性：主要用于管理信息系统开发。
  - 对于较大的项目需要足够的人力资源去建造足够的 RAD 组。
  - 开发者和客户必须在很短的时间内完成一系列的需求分析，任何一方配合不当都会导致 RAD 项目失败。
  - RAD 模型对模块化要求比较高，如果存在不能被模块化的功能需求，那么建造 RAD 所需要的构件就会有问题。
  - 技术风险很高的情况下不适合使用 RAD 模型。

原型模型

- Prototype Model



- 原型模型
  - 线性顺序方法的局限
    - 瀑布模型、V 模型和 W 模型都将软件生命周期划分成独立串行的几个阶段，如果前一个阶段的任务没有全部完成便无法开始下一阶段的工作。
    - 很多时候开发者很难得到完整而准确的系统需求规格说明，因为：
      - 在开发早期用户往往对目标软件系统的最终表现只有一个模糊的想法，不能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。
      - 开发者有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，需要通过改变需求来解脱困境。

- 原型模型
  - 原型与原型方法
    - 原型指模拟某种最终产品形态的原始模型。
    - 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。

- 原型模型
  - 原型方法的三种作用类型：
    - 探索型
      - 澄清用户对目标系统的要求，确定用户期望的特性；探讨多种实现方案的可行性。主要针对需求模糊、用户和开发者对目标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。
    - 进化型
      - 在构造系统的过程中适应需求的变化，不断改进原型，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。

原型模型

原型方法的策略

由于运用原型的目的和方式不同，在使用原型时可采取以下两种不同的策略：

(1) 废弃策略

- 构造的原型主要用于反馈和评价，据此设计并实现一个完整、准确、一致、可靠的目标系统。系统完成后，原型被废弃不用。探索型和实验型原型采用的就是废弃策略。

(2) 追加策略

- 构造的原型作为目标系统的核心，通过不断的扩充修改，逐步追加实现新的需求，最后形成目标系统。追加策略对应于进化型原型。

- 原型模型
  - 原型方法的特点
    - 从认知论的角度看，原型方法遵循了人们认识事物的规律，因而更容易为人们所普遍接受，这主要表现在：
      - 人们对任何事物的认知都有一个过程，认识和学习都是循序渐进的。
      - 人们对于事物 (需求) 的描述，往往都是在环境的启发下不断完善。
      - 人们评价一个已有的事物，要比空洞地描述自己的设想容易得多，改进一些事物要比创造一些事物容易得多。

- 原型模型
  - 原型方法的特点 (续)
    - 原型方法在分析的初期阶段引入模拟手段，提供了用户与开发人员良好的沟通手段，易于被人们接受。使用原型方法的好处有：
      - 有助于增进软件开发人员和用户对系统服务需求的理解，减少两者之间的误解。
      - 易于确定系统的性能，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目标系统的一个组成部分，有利于最终系统的建成。

- 原型模型
  - 原型方法的适用范围和局限性
    - 大型系统难以进行直接的原型模拟，只能经过系统分析得到系统的整体结构。
    - 原型方法难以构造处理运算量大、逻辑性较强的程序模块的原型。
    - 在用户需求的业务流程、信息流程混乱的情况下，原型的构造与评价有一定的困难。
    - 批处理系统的大部分活动是内部处理的，应用原型方法会有一定的困难。
    - 此外原型方法还存在容易忽略文档工作、建立原型带来资源浪费、项目规划和管理困难等问题。

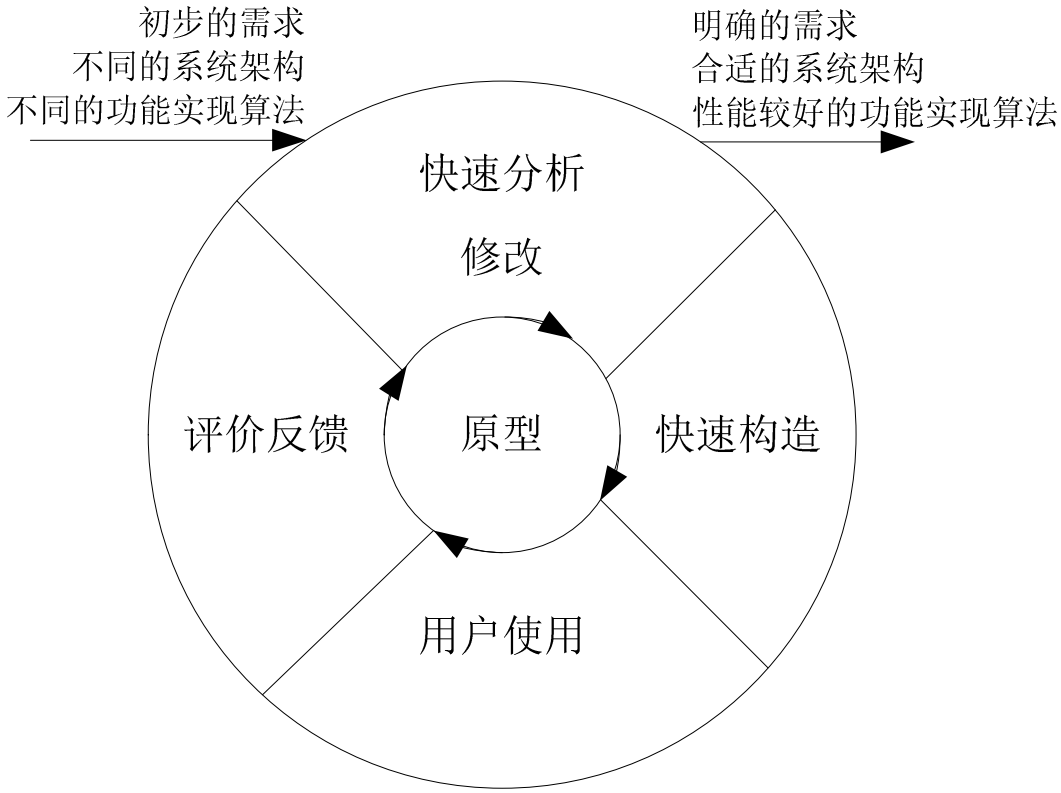
原型模型

- 原型方法的适用范围和局限性 (*Bernard Boar*, 1984)

	适用原型法	不适用原型法
系统结构	联机事务处理系统、相互关联的应用系统	批处理系统
逻辑结构	有结构系统，如运行支持系统、管理信息系统	基于大量算法和逻辑结构的系统
用户特征	积极参与、积极决策	
应用约束		在线运行系统的补充
项目管理	项目负责人愿意使用原型方法	项目负责人不愿意使用原型方法
项目环境	需求复杂易变、性能要求高	需求明确固定

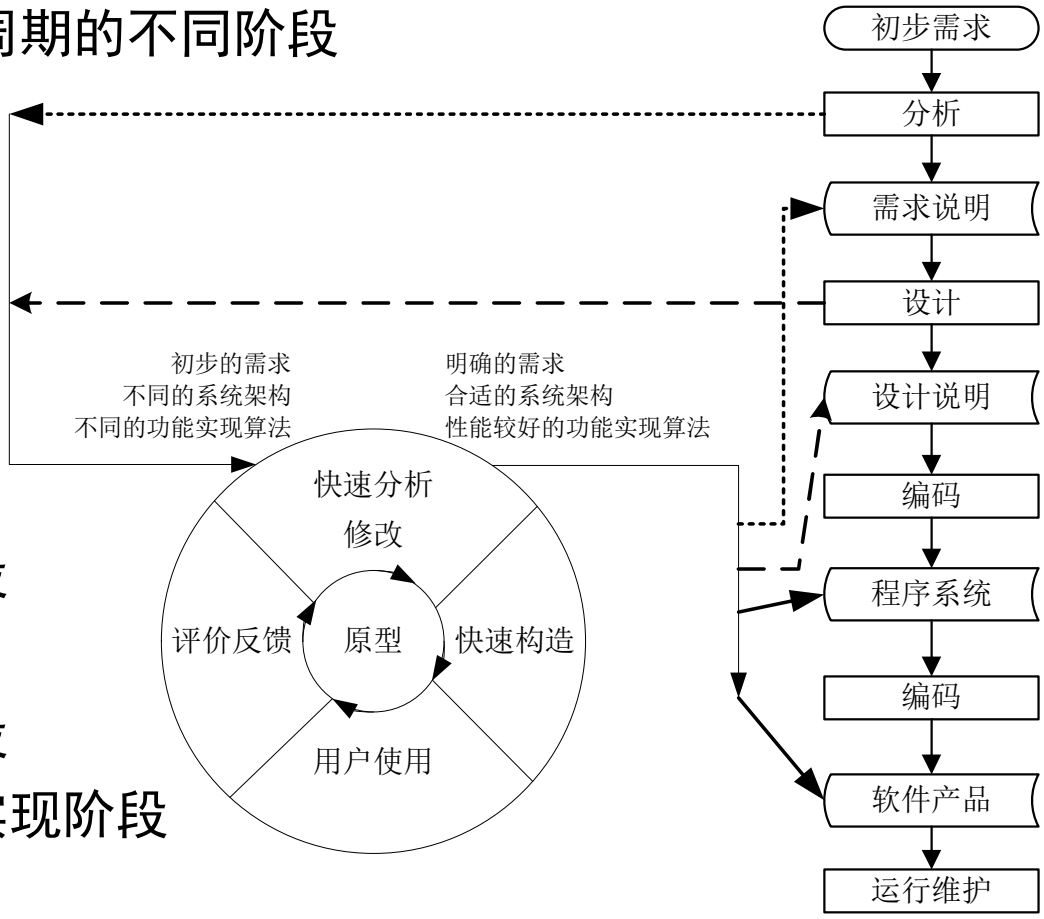
原型模型

原型方法的应用过程



原型模型

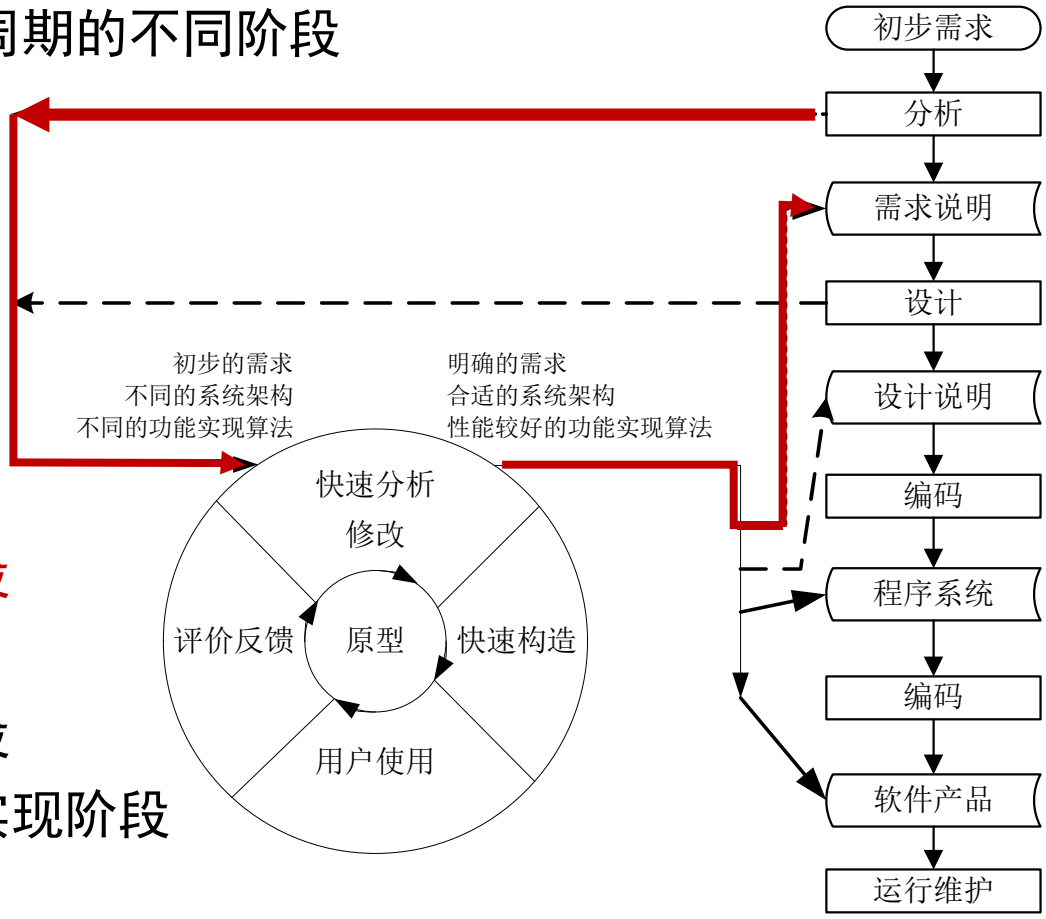
- 原型方法支持软件生命周期的不同阶段



- 辅助或代替分析阶段
- 辅助设计阶段
- 代替分析与设计阶段
- 代替分析、设计和实现阶段
- 代替全部开发阶段

原型模型

- 原型方法支持软件生命周期的不同阶段

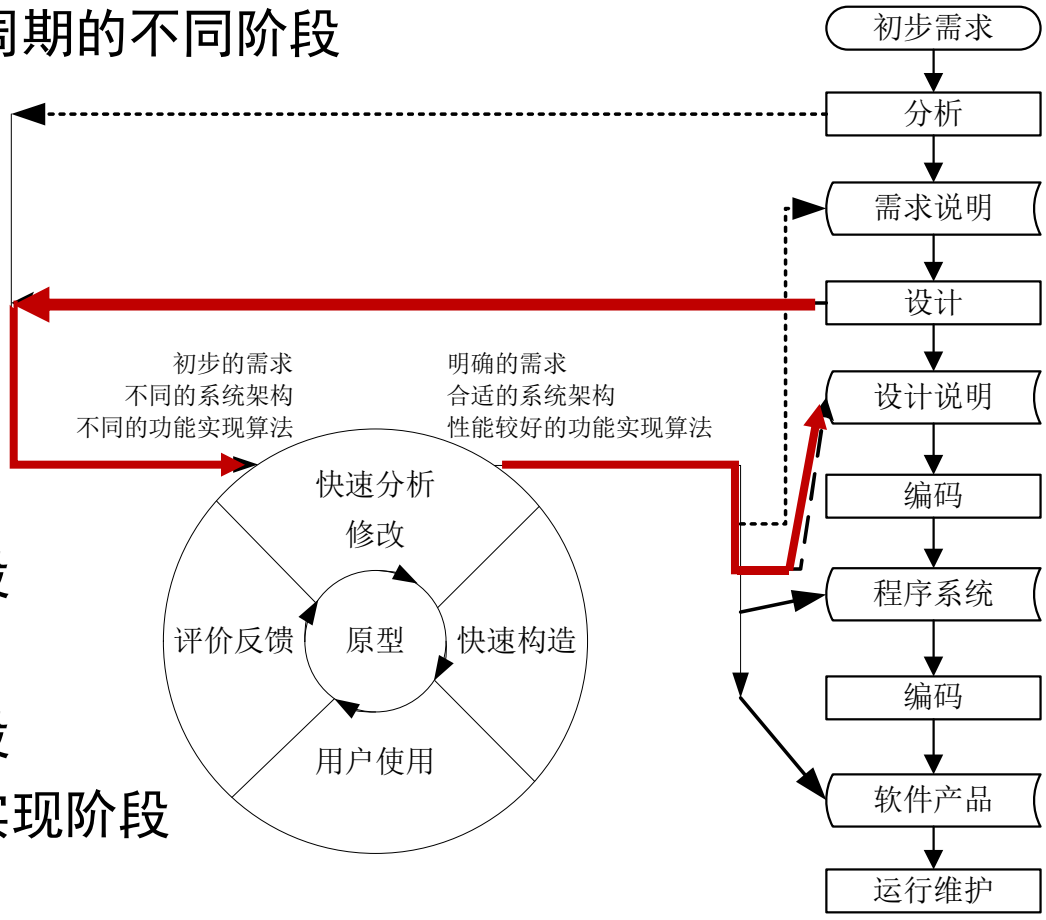


- 辅助或代替分析阶段
- 辅助设计阶段
- 代替分析与设计阶段
- 代替分析、设计和实现阶段
- 代替全部开发阶段

软件生命周期模型

原型模型

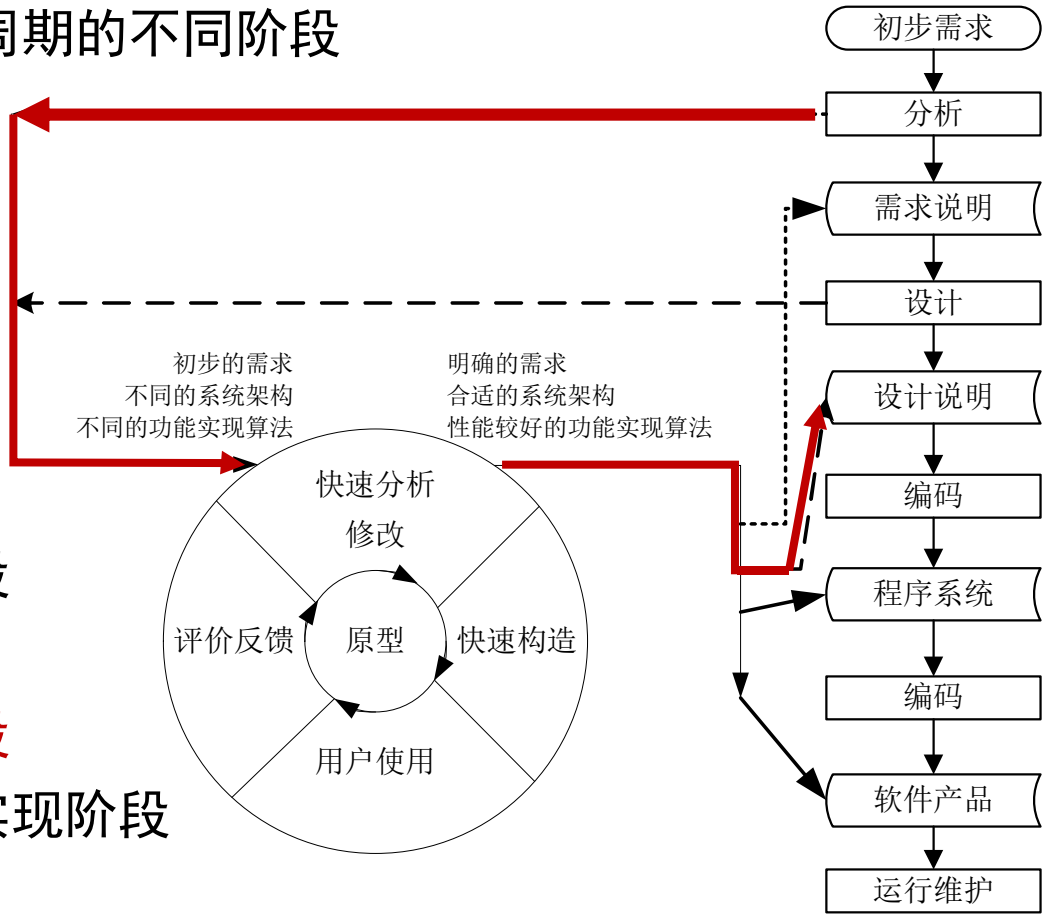
- 原型方法支持软件生命周期的不同阶段



- 辅助或代替分析阶段
- 辅助设计阶段
- 代替分析与设计阶段
- 代替分析、设计和实现阶段
- 代替全部开发阶段

原型模型

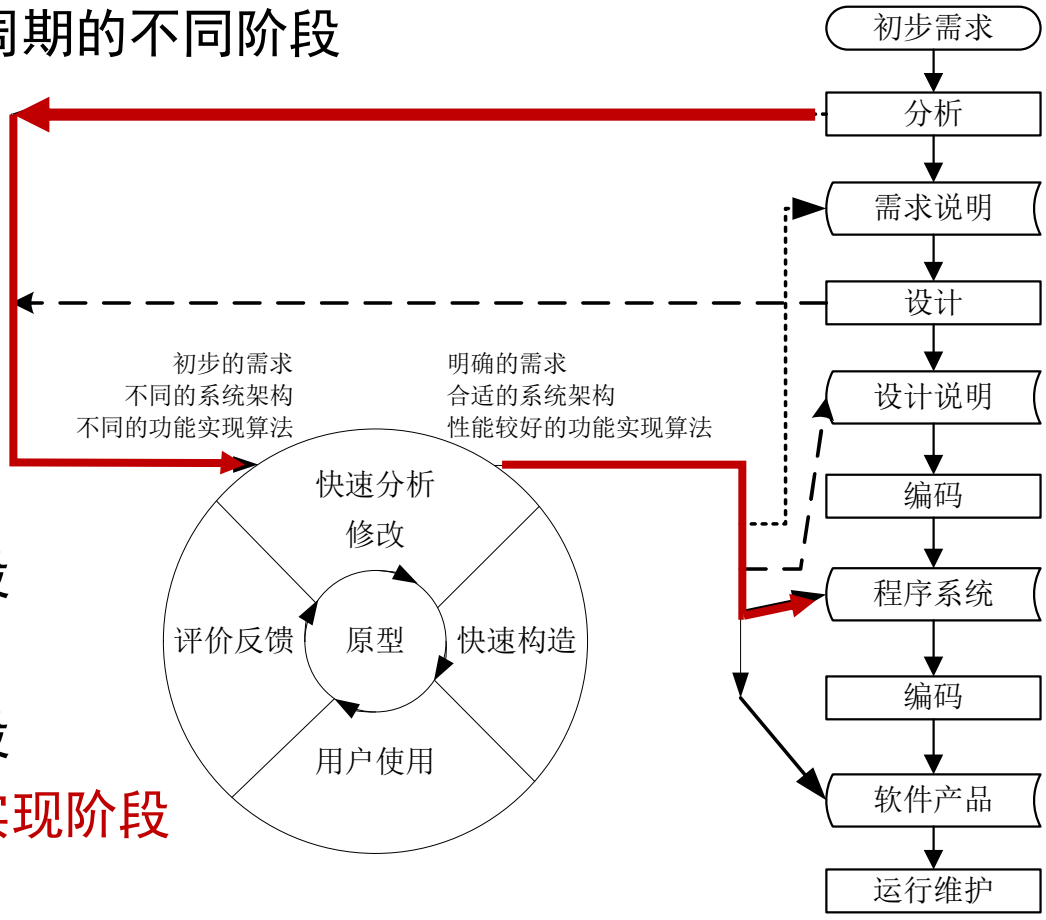
- 原型方法支持软件生命周期的不同阶段



- 辅助或代替分析阶段
- 辅助设计阶段
- 代替分析与设计阶段
- 代替分析、设计和实现阶段
- 代替全部开发阶段

原型模型

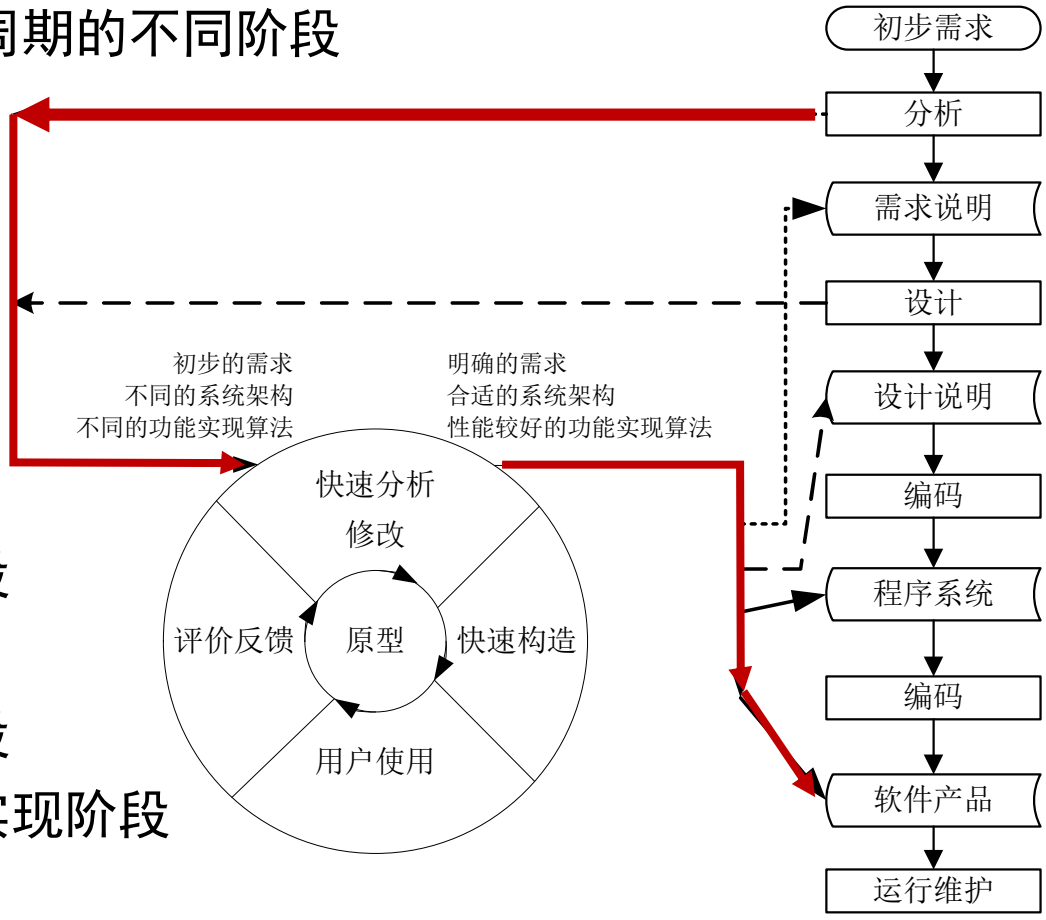
- 原型方法支持软件生命周期的不同阶段



- 辅助或代替分析阶段
- 辅助设计阶段
- 代替分析与设计阶段
- 代替分析、设计和实现阶段
- 代替全部开发阶段

原型模型

- 原型方法支持软件生命周期的不同阶段



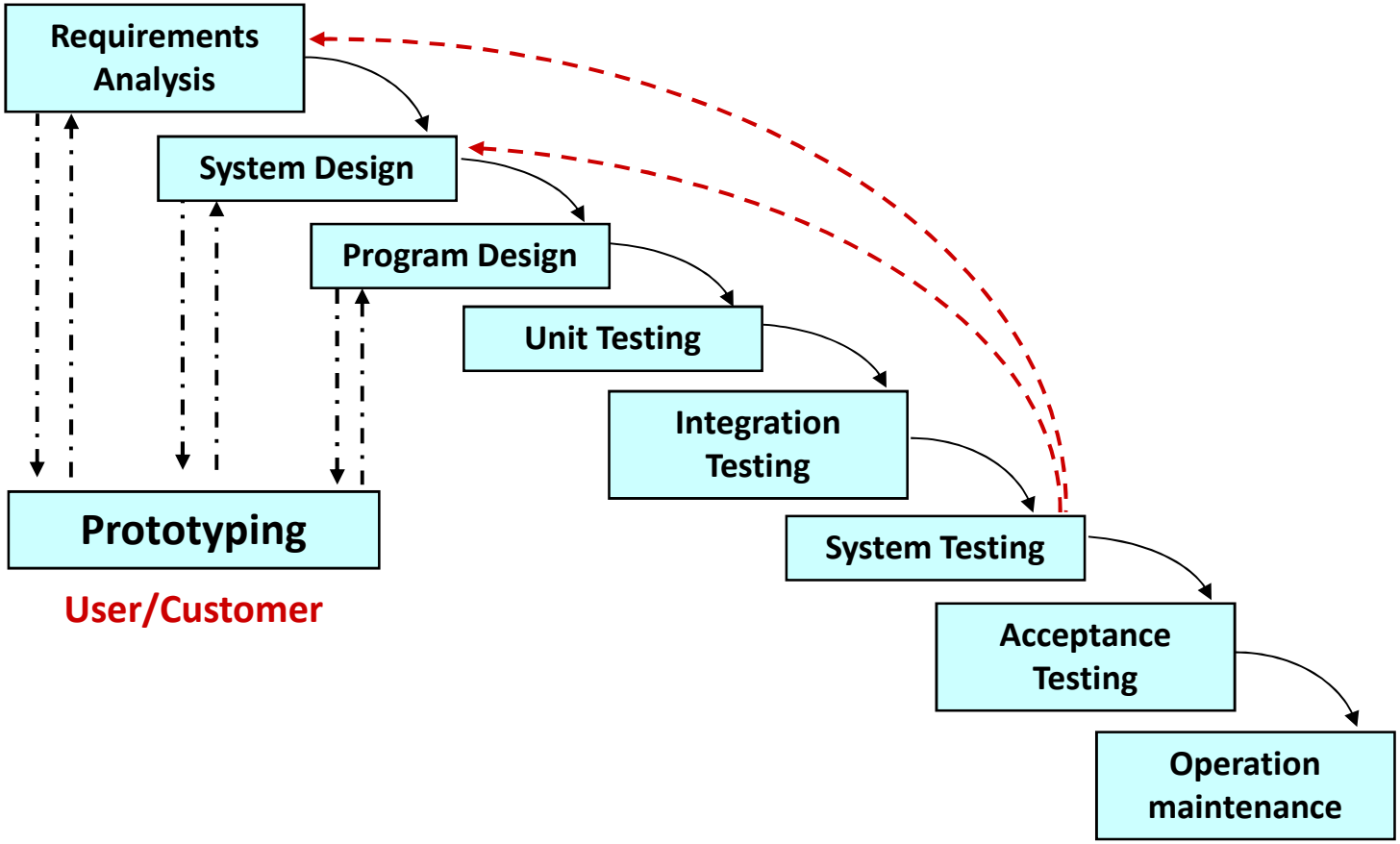
- 辅助或代替分析阶段
- 辅助设计阶段
- 代替分析与设计阶段
- 代替分析、设计和实现阶段
- 代替全部开发阶段

- 原型模型
  - 支持原型构造的软件复用技术
    - 软件复用：利用一些从早先软件开发过程中收集到的、对建立新的软件系统有用的信息来构建新系统。
    - 复用类型按内容划分：
      - 数据复用：实现不同数据环境的移植
      - 模块复用：COM/DCOM、JavaBean/EJB、CORBA
      - 结构复用：领域内通用业务逻辑；实现 MVC (Model-View-Control) 体系结构的 Struts 框架、实现数据库访问逻辑复用的 Hibernate 框架等
      - 设计复用：MDA (Model Driven Architecture，模型驱动体系结构)
      - 规格说明复用：规格说明可使用或者可参照使用

- 原型模型
  - 支持原型构造的软件复用技术 (续)
    - 软件复用的两种实现机制
      - 合成复用：
        - 构件是合成复用的基础。构件基于抽象数据类型，将功能实现细节与数据结构封装在构件内部，提供外部接口供使用者构造应用时调用。构件本身可以是对某一函数、过程、子程序、数据类型、算法等可复用软件成份的抽象，利用构件来构造软件系统，可以获得较高的生产率和较短的开发周期。
      - 生成复用：
        - 利用可复用的模式 (Patterns)，通过生成程序产生一个新的应用程序或程序段。

软件生命周期模型

原型+瀑布模型



Thank you!