

软件工程

大连理工大学软件学院



第2章 软件开发过程

软件开发过程（**software development process**）又叫做软件开发生命周期（**software development life cycle, SDLC**），是软件产品开发的任务框架和规范，又可以简单的称为软件生命周期及软件过程。

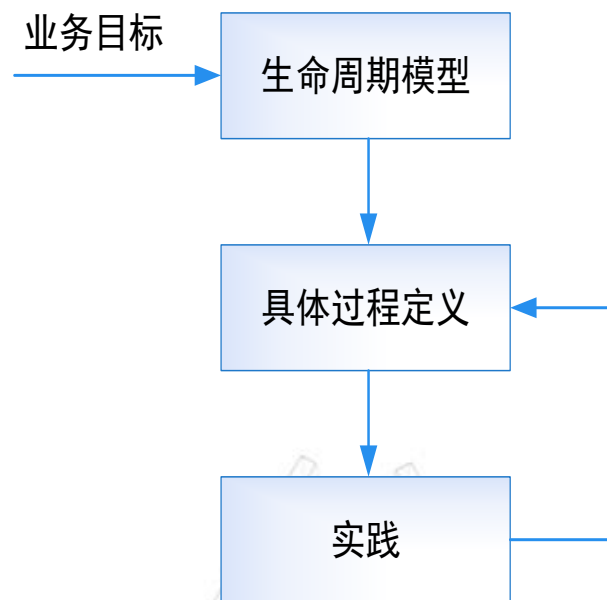
- 软件过程和生命周期
- 软件过程模型：传统过程和敏捷（**Agile**）过程
- 过程建模
- 风险管理过程

软件生命周期与开发过程

- 国际标准ISO/IEC 12207是软件生命周期过程的国际标准，旨在提供一套软件开发与维护过程中涉及的各种任务定义的标准，如软件生命周期的选择、实现与监控等。
- 可重复的、可预测的过程能够提升软件生产的效率和质量。
- 软件工程过程小组(**Software Engineering Process Group, SEPG**)提供给软件开发人员统一的标准的开发原则，充分协调各开发人员、开发小组，通过过程控制的方法，保证软件产品的质量。

软件过程与生命周期

- 通常生命周期模型要更一般化，而软件开发过程则更为具体化。
- 可重复的、可预测的过程能够提升软件生产的效率和质量。（过程改进）
- 生命周期是软件开发的宏观上的框架，软件过程则涉及到软件开发的流程等管理细节，在框架稳定的前提下允许对软件过程进行裁剪。
- 4种不同类型的生命周期：顺序式、迭代式、增量式以及敏捷式



软件生命周期

- 过程管理主要采用的是一种“分而治之”的思想，即将整个软件的生命周期划分成**软件定义**、**软件开发**和**运行维护**三个主要的时期，每个时期再细分为具体的阶段，分别对应明确的任务。



可行性分析与开发计划

1. 目的：用最小的代价在尽可能短的时间内确定该软件项目是否能够开发，是否值得开发，最后给决策者提供做与不做的依据。（较高层次的需求分析和设计）
2. 任务：确定项目的规模和目标，确定项目的约束和限制；进行简要的需求分析，抽象出逻辑结构，建立逻辑模型；从逻辑模型出发，经过压缩的设计，探索出若干种可供选择的主要解决办法。
3. 每种解决方法都要从三方面研究它的可行性：技术可行性、经济可行性和社会可行性。
4. 描述所提出的解决方案和方案的可行性，并拟定开发计划，包括对费用、时间、进度、人员组织、硬件配置、软件开发环境和运行环境配置等进行说明和规划。

需求分析

1. 在确定软件开发可行的情况下，对目标软件未来需要完成的功能进行的详细分析。
2. 需求分析是软件开发后续阶段的基础，直接关系到整个系统开发的成功与否。由于用户的需求随着项目的进展和理解处在不断的变化之中，应对需求进行**变更管理**。
3. 应充分理解和掌握用户对目标软件的期望，除**功能需求**外还要对系统设计有影响的**非功能性需求**加以识别和分析。
4. 需求分析阶段的输出是一份“**需求规格（Specification）说明书**”的文档。

软件设计

1. 软件设计是在需求分析的基础上寻求系统求解的框架，如系统的架构设计、数据设计等。
2. 软件设计可分为**概要设计**和**详细设计**，此阶段的输出分别为“概要设计说明书”和“详细设计说明书”。
3. 设计方案是软件实现的蓝图，应综合考虑软件的性能、扩展、安全等因素，合理规划系统模块的结构，充分考虑未来变化的可能性并预留空间，尽可能保证系统设计结构在整体上的稳定性。

程序编码

- 此阶段是将软件设计的结果翻译成某种计算机语言实现的程序代码。
- 在程序编码中必须要制定统一的如何编码的标准规范，尽量提高程序的可读性、易维护性，提高程序的运行效率。

软件测试

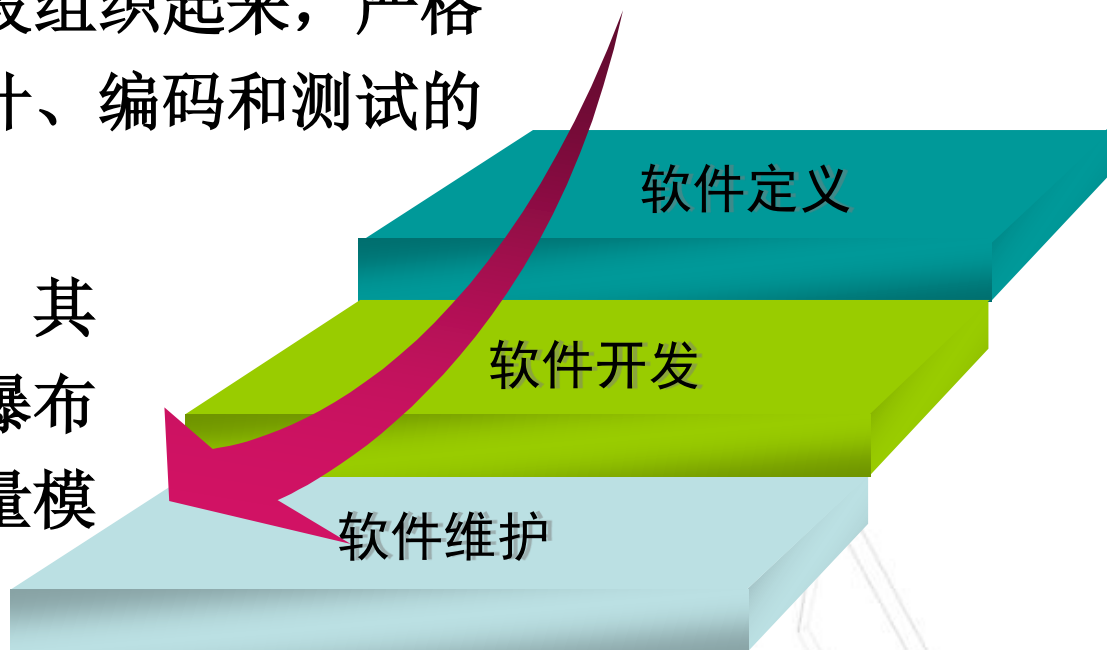
- 程序编码后需要对代码进行严密的测试，以发现软件在整个设计过程中存在的问题并加以纠正。
- 测试的环节可分为**单元测试、集成测试及系统测试**三个阶段进行。
- 测试方法主要包括**黑盒和白盒方法**。
- 测试过程需要建立详细的测试计划、编写测试用例、记录并分析测试结果，以保证测试过程实施的有效性，避免测试的随意性。

软件维护

- 软件维护是软件生命周期中持续时间最长的阶段，是为软件能够持续适应用户的要求延续软件使用寿命的活动。
 - **改正性维护**：诊断和改正在使用过程中发现的软件错误；
 - **适应性维护**：修改软件以适应环境的变化；
 - **完善性维护**：根据用户的要求改进或扩充软件使它更完善；
 - **预防性维护**：修改软件为将来的维护活动预先做准备。

传统生命周期模型

- 顺序的将生命周期阶段组织起来，严格按照需求、分析、设计、编码和测试的阶段进行。
- 传统的生命周期模型，其中具有代表性的包括瀑布模型、原型模型、增量模型等。
- 最基本和有效的可供选择的软件开发模型。






瀑布模型

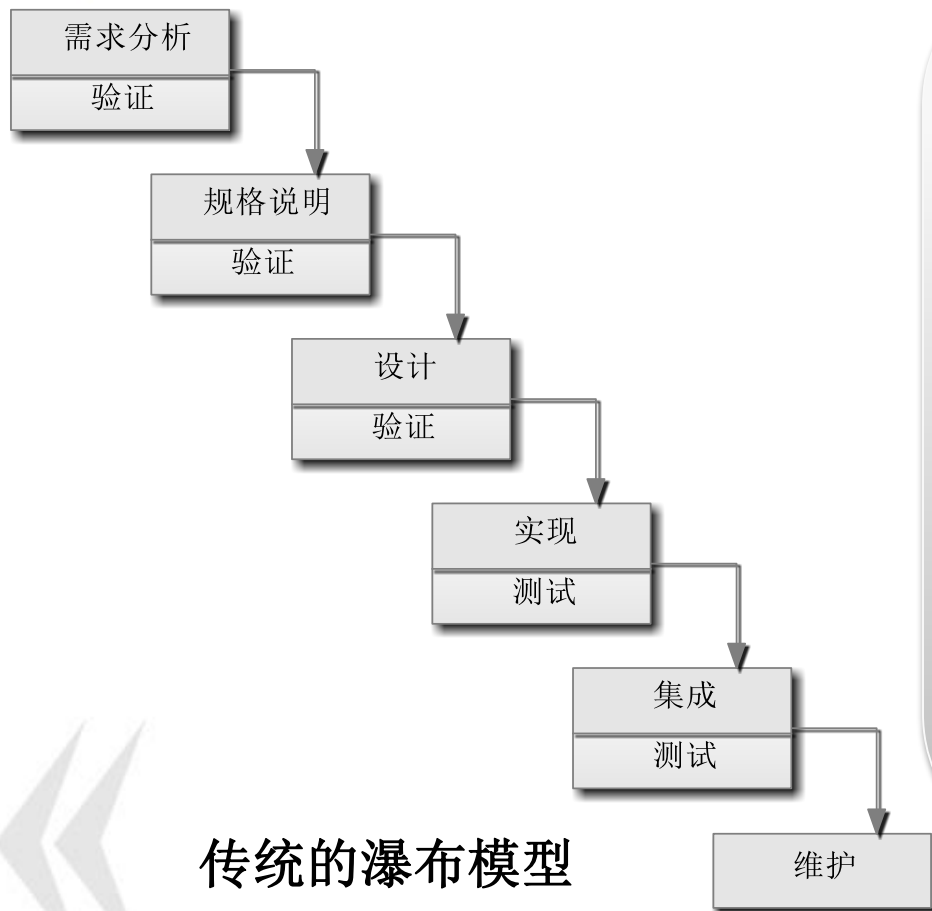
- 在20世纪80年代之前，瀑布模型一直是唯一被广泛采用的生命周期模型，现在仍然是应用得最广泛的过程模型。
- 按照传统的瀑布模型来开发软件，有如下特点。
 - 阶段间具有顺序性和依赖性，文档驱动
 - 推迟实现，不急于编写代码
- 尽可能的理解和掌握系统需求，
- 清楚地区分逻辑设计与物理设计，尽可能推迟程序的物理实现。



— 质量保证的观点

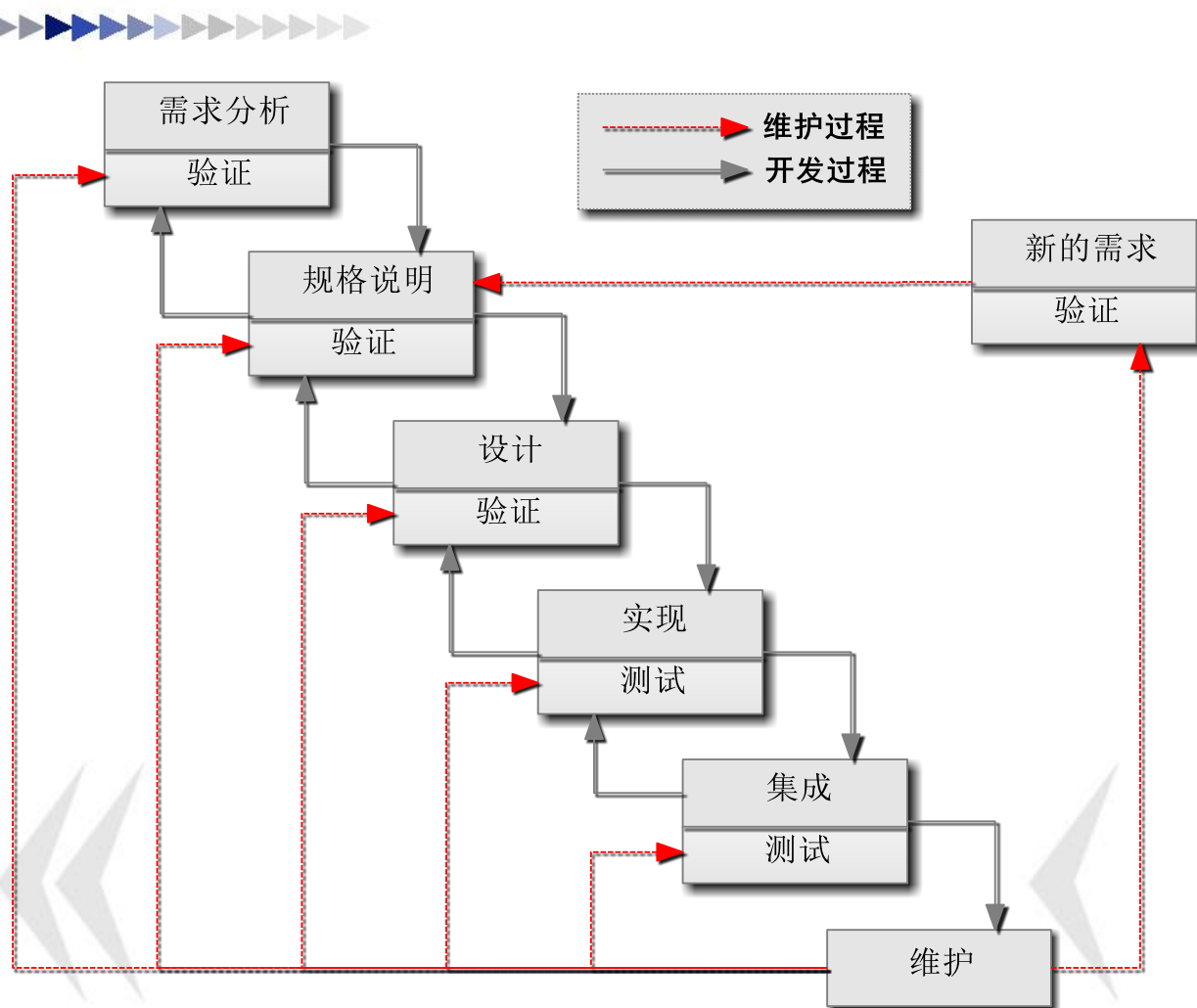
- 每个阶段都必须完成规定的文档，没有交出合格的文档就是没有完成该阶段的任务。
 - 每个阶段结束前都要对所完成的文档进行评审，以便尽早发现问题，改正错误。
- 
- 
- 

瀑布模型的问题



- 不希望有“变化”
- 变化来的越晚，付出的代价越高。
- 设计阶段过多的假设，导致理想化、一厢情愿的东西过多。(用户只参与需求)
- “文档驱动”，静态

实际瀑布模型



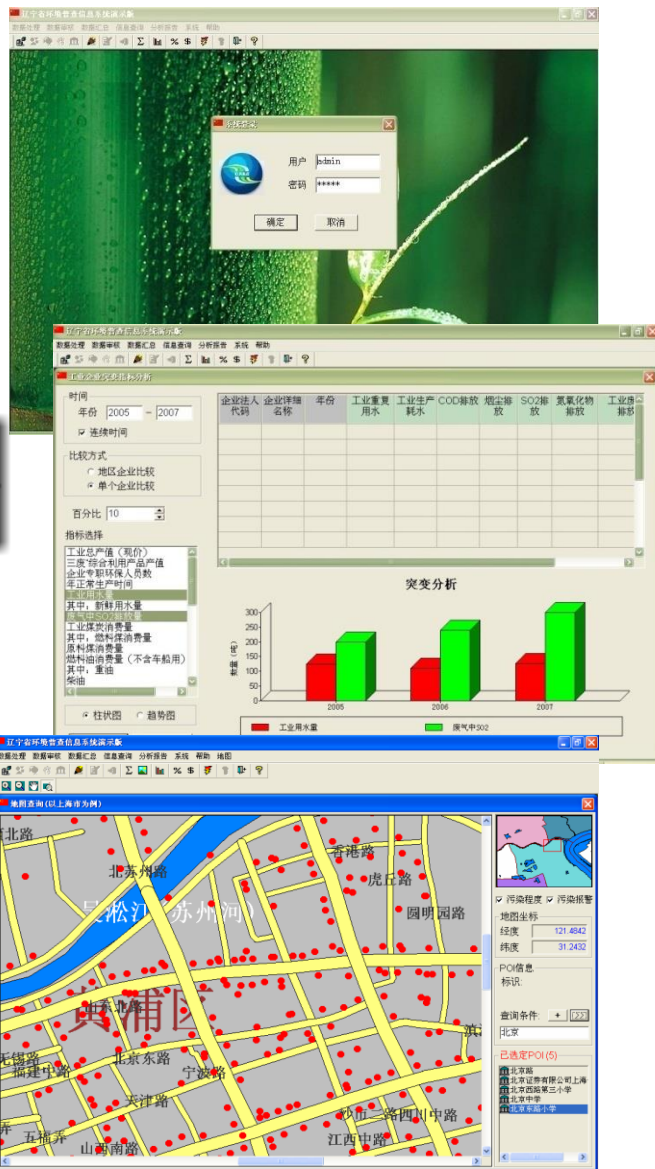
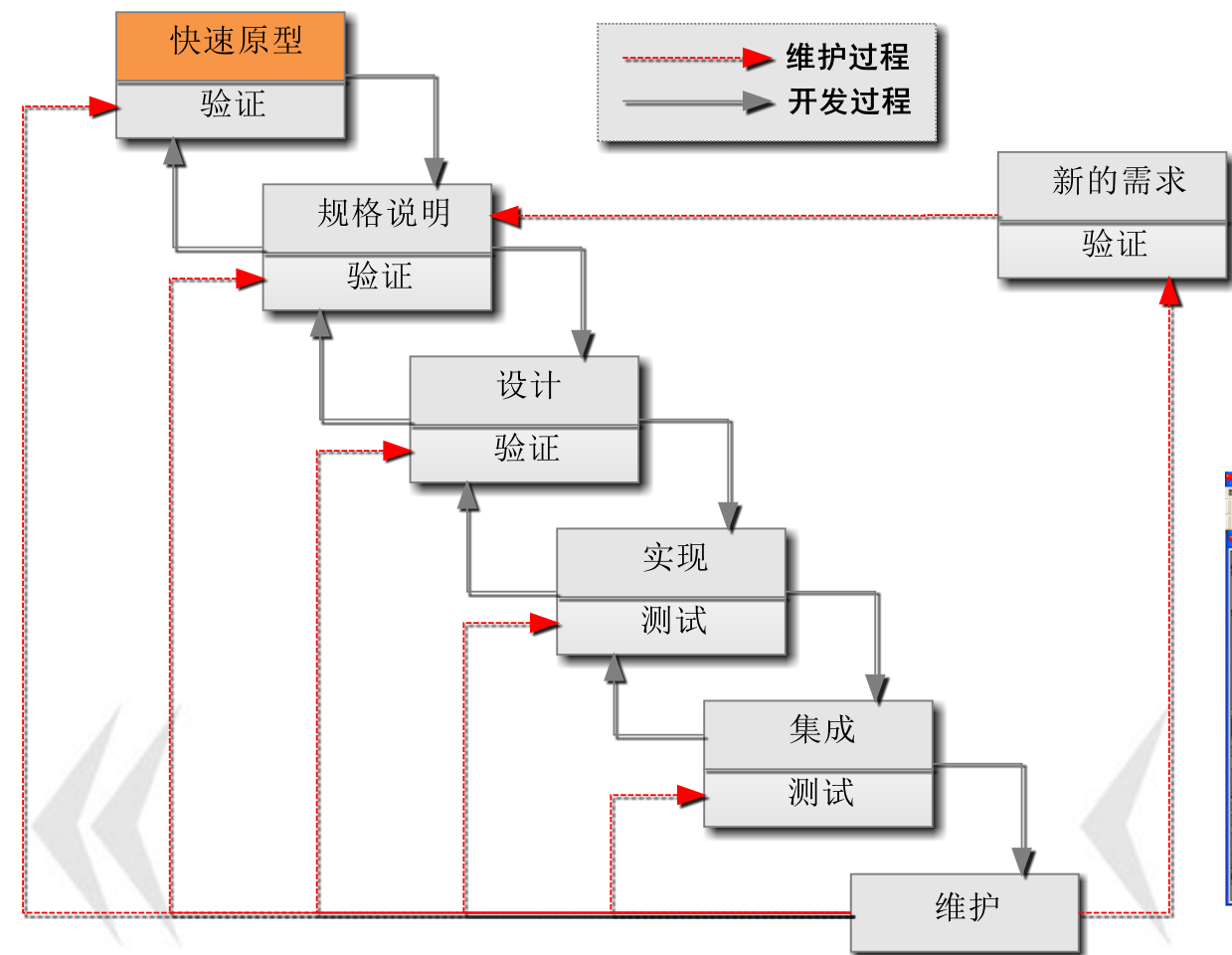
- 可以在一定程度解决“变化”的问题
- 如何“拥抱”变化？
- 计划驱动，在对系统整体的把控和协调上，具有优势，因此适合规模较大的系统或分布式开发模式。

快速原型模型

- 快速原型模型（**Rapid Prototype**）的主要作用是在用户和开发者之间起到“桥梁”的作用。
- 开发者和用户之间经常面临的一个状况是：用户熟悉的是业务但不懂得开发的技术，而开发者正好相反，其更熟悉具体的开发方法、工具等技术内容而不明白相关的业务流程。
- 这也是为什么需求分析较难开展的原因之一，也是无法固化用户需求的客观原因之一。



快速原型模型



快速原型模型的特点

- 快速原型模型要求对系统进行简单和快速的分析，快速构造一个软件原型。
- 用户和开发者在试用或演示原型过程中加强沟通和反馈，通过反复评价和改进原型，减少双方的误解，降低缺陷引入的几率，降低由于需求不明确带来的开发风险和提高软件质量，获取到用户真正的需求。
- 比较适合一个全新的系统开发，用户借助原型了解开发方向的正确性，如用户界面的构建，使用户建立起对未来系统的认识 and 了解。
- 快速原型中可以尝试运用未来系统中需要的新技术，提前测试一些性能上的要求是否能够达到预期。

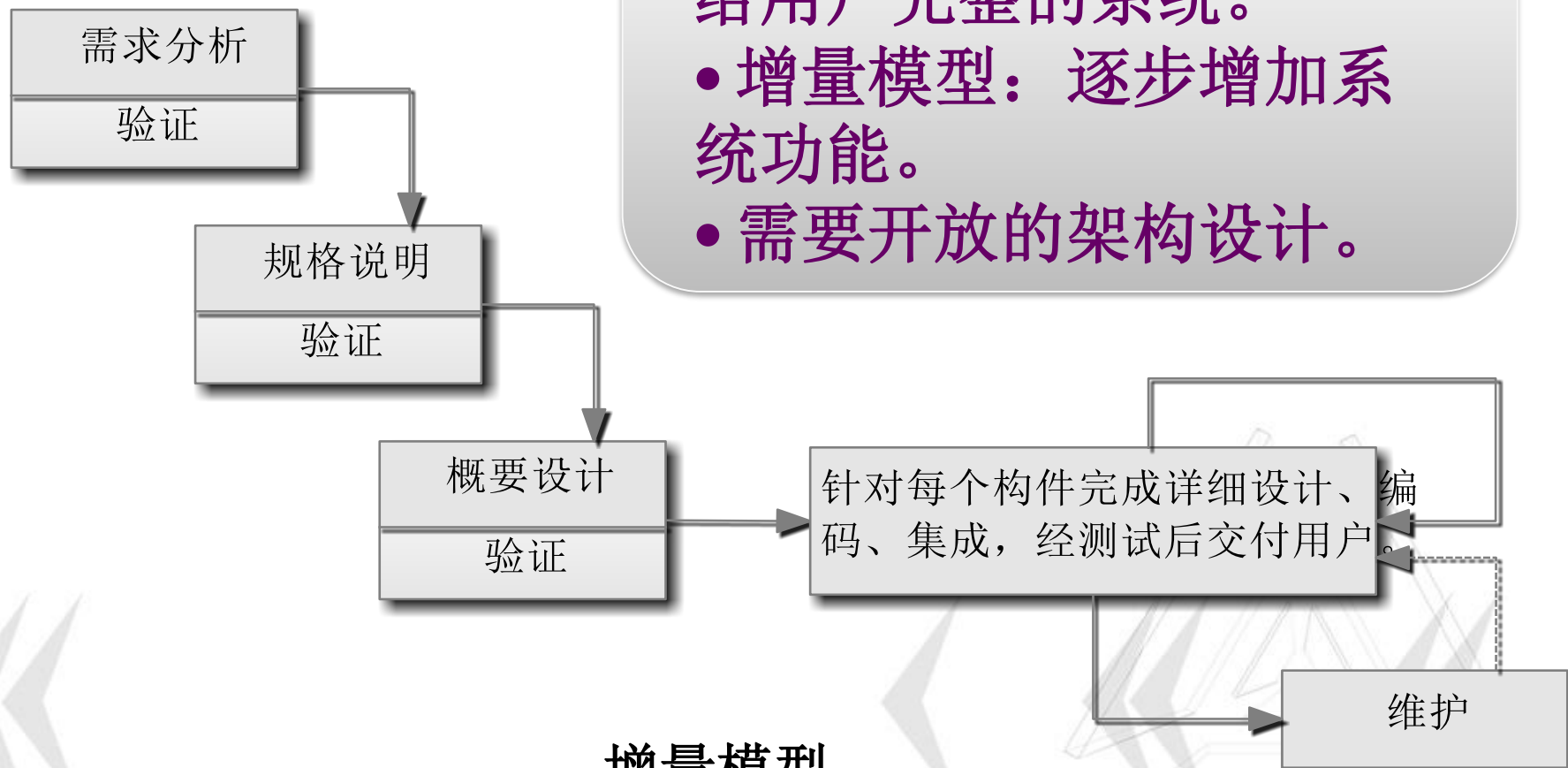
快速原型的问题

- 所选用的开发技术和工具不一定是实际项目的需要。
- 快速建立起来的模型可能由于不符合各种开发规范，再加上不断的修改，质量一般都比较差，通常在实际开发过程中会完全抛弃之前建立起来的原型系统。

增量模型

- 也称为渐增模型。把软件产品作为一系列增量构件来设计、编码、集成和测试。
- 每个构件由多个相互作用的模块构成，并且能够完成特定的功能。
- 使用增量模型时，第一个增量构件往往实现软件的基本需求，提供最核心的功能。(滚雪球方式)

- 瀑布模型：力求一次性给用户完整的系统。
- 增量模型：逐步增加系统功能。
- 需要开放的架构设计。

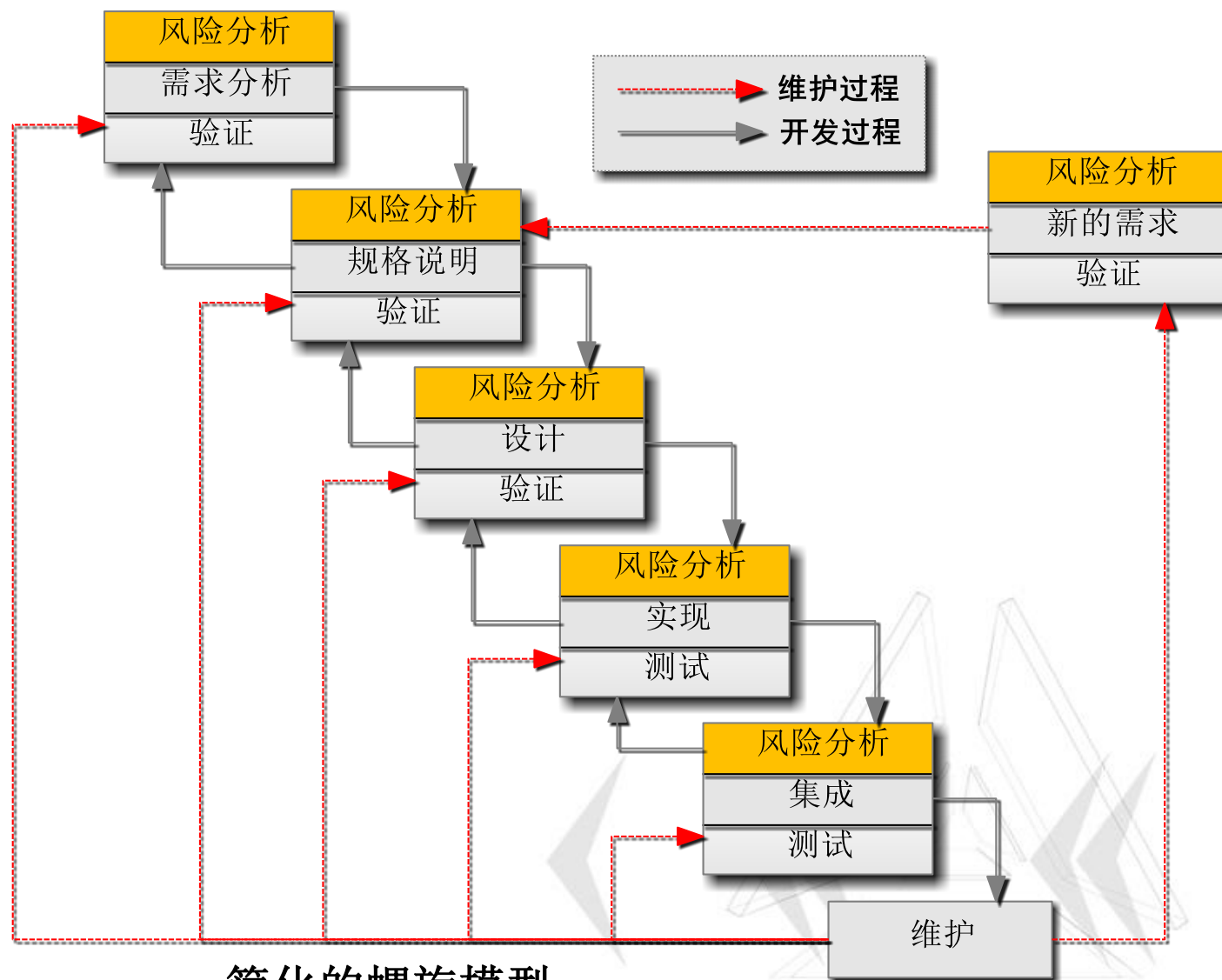


增量模型

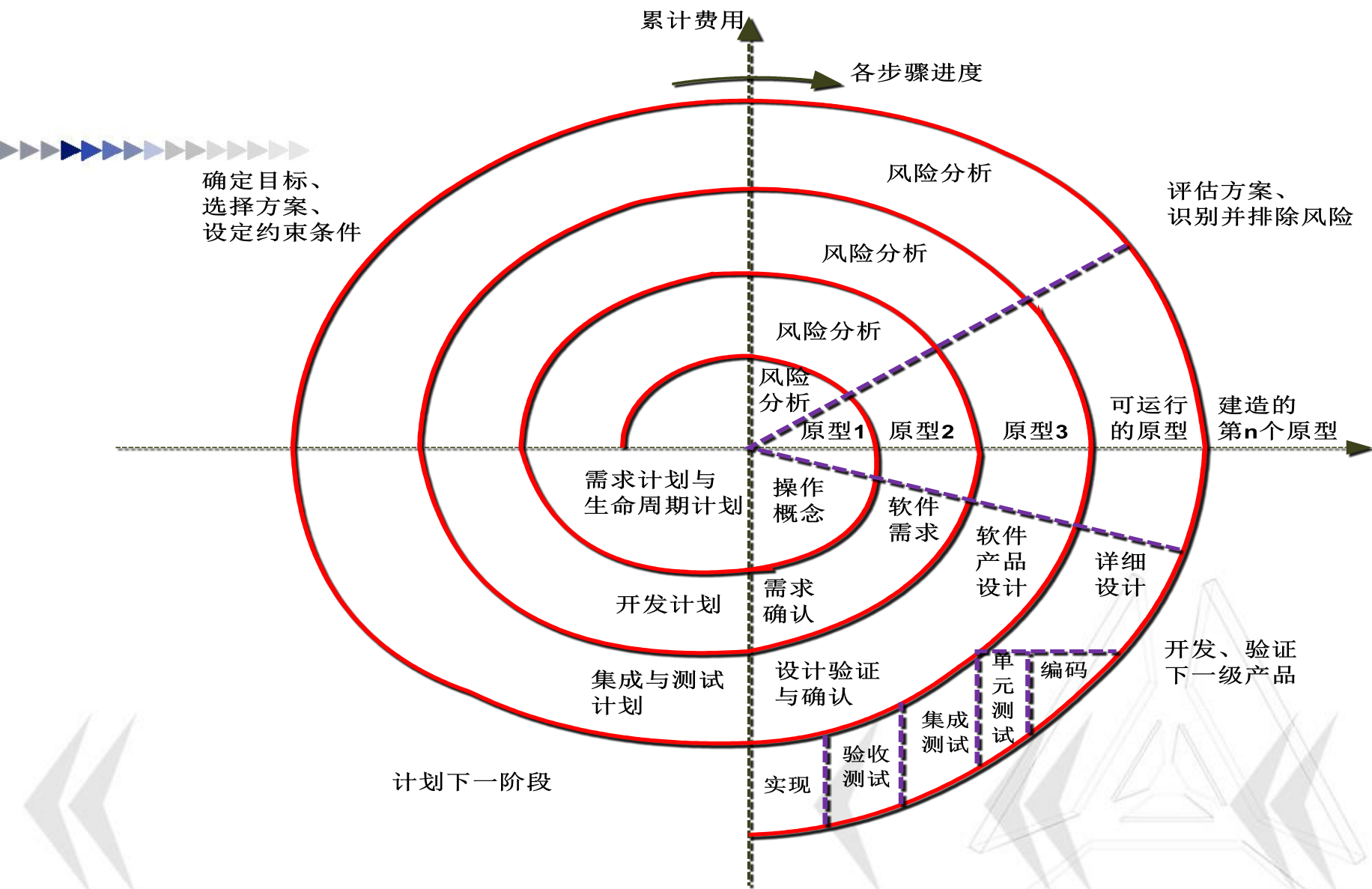
螺旋模型

- 螺旋模型的基本思想是使用原型及其它方法尽量降低风险。
 - 在每个阶段之前都增加了风险分析过程的快速原型模型。
- 特别适合于大型复杂的系统。螺旋模型沿着螺线进行若干次迭代，四个象限代表了不同的活动。

- 原型模型可以在一定程度上降低风险，但对有些风险也无能为力。
- 需要专业的风险评估人员。



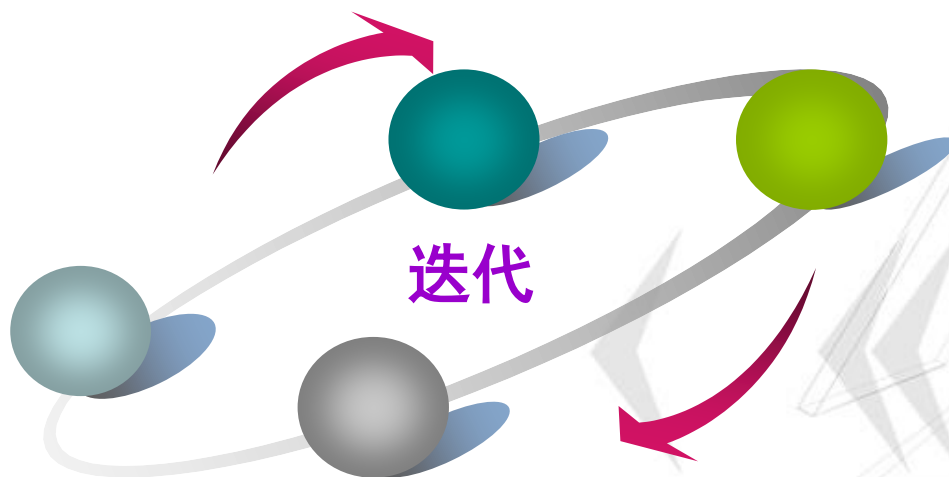
简化的螺旋模型

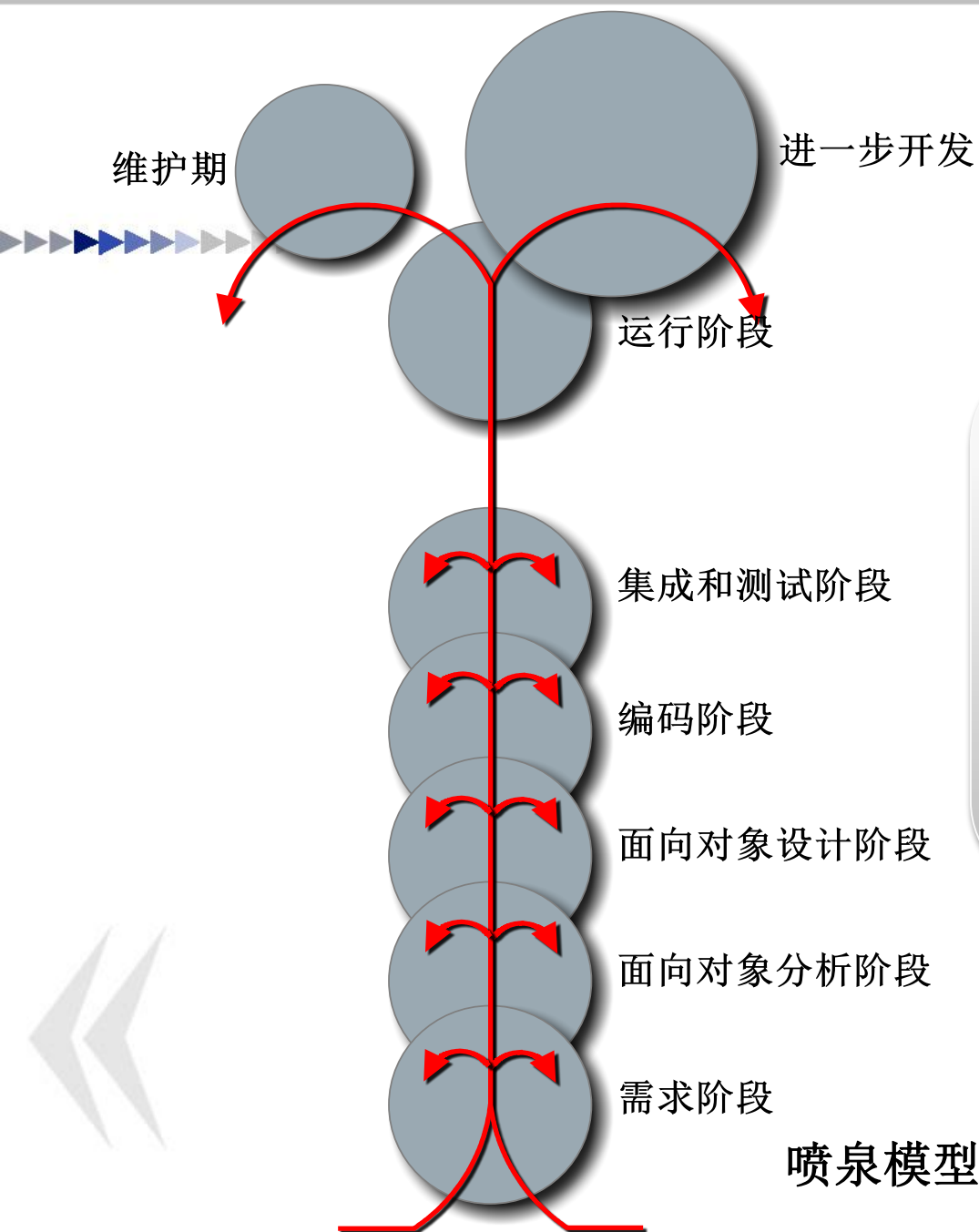


完整的螺旋模型

喷泉模型

- **迭代**是OO开发过程的主要特性。
- 喷泉模型是典型的面向对象生命周期模型。
- “喷泉”体现了面向对象软件开发过程**迭代和无缝**的特性。
- 为避免喷泉模型的过分无序，把一个**线性过程**作为总目标。





- 迭代：逐步求精
- 阶段间没有明显的界限—面向对象的思想保证了各个阶段开发的一致性。

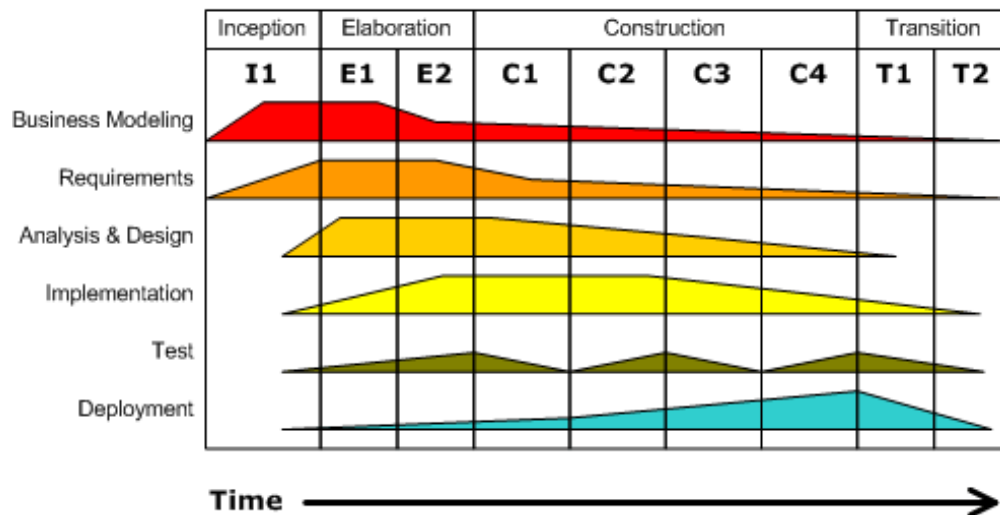
喷泉模型

敏捷软件开发

- 迭代式开发
- 增量交付
- 开发团队和用户反馈推动产品开发
- 持续集成
- 开发团队自我管理

Iterative Development

Business value is delivered incrementally in time-boxed cross-discipline iterations.



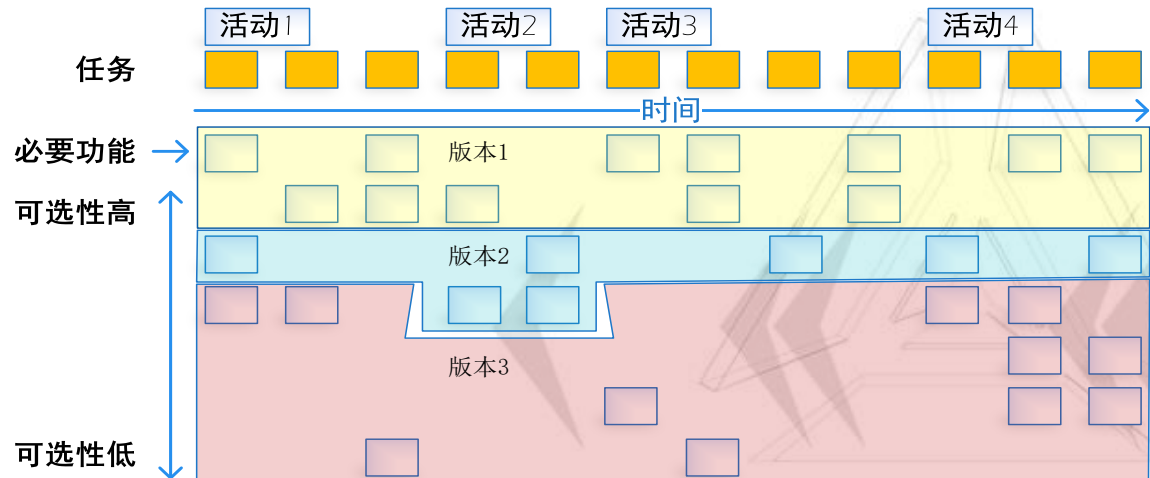
敏捷宣言：

- 个体和互动胜过流程和工具；
- 工作的软件胜过详尽的文档；
- 客户合作胜过合同谈判；
- 响应变化胜过遵循计划。

增量的开发方式

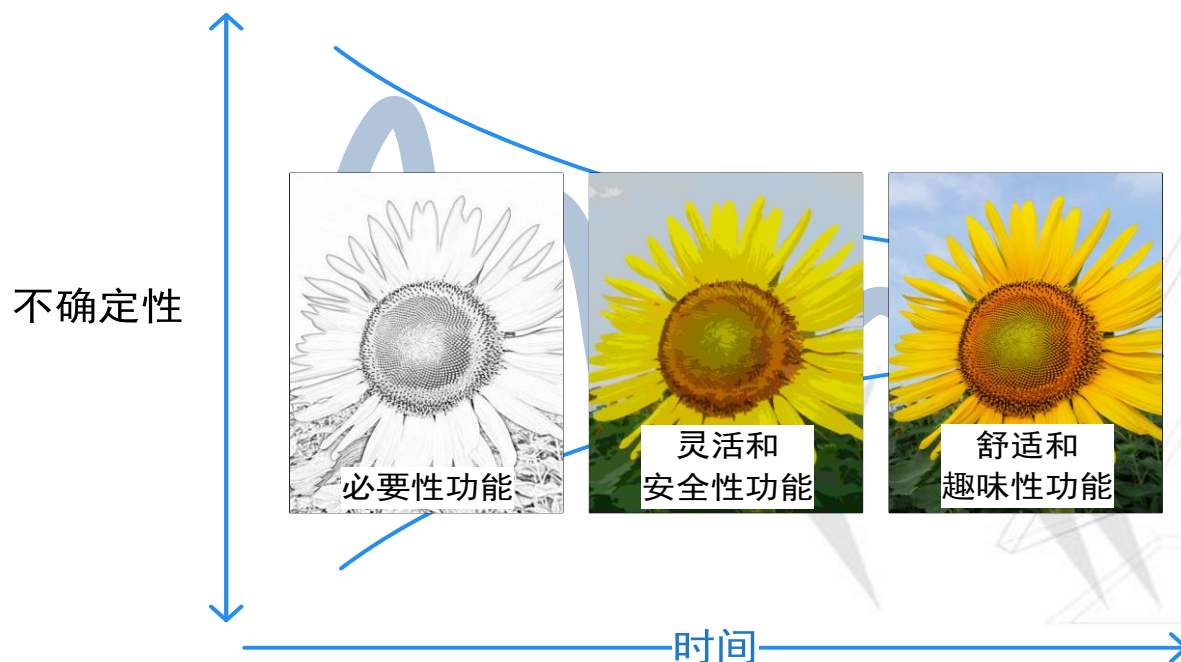
- 增量开发的方式即分批分期的交付用户产品，通过增量开发来应对软件产品之外的不确定因素（风险）。
- 敏捷方法建议先实现必要性的用户案（用）例（用户故事），体现出软件的价值，然后在后续版本中对功能进行细化，使得我们的软件产品的所有功能都能够达到相同的用户体验水平。

“用例”是指一件用户通过系统完成的有价值的目标，它不是一个具体的功能。



迭代的开发方式

- 但用户的需求往往无法立刻稳定。
- 迭代的思想就是当对需求还没有信心的时候，不指望构建的软件正是客户所想要的，但可以先构建后修改，通过多次反复找到真正客户需要的软件。（逐步求精）



敏捷的优势

- 精确
- 质量
- 速度
- 丰厚的投资回报率
- 高效的自我管理团队



- 敏捷开发更适合规模中小、需求变化频繁的系统开发，并且强调团队的作用，所以更适合集中式的开发模式。

极限编程 (XP)

- e**X**treme **P**rogramming
- 主要目的是降低需求变化的成本
- 崇尚的开发方法：
 - 客户代表与开发团队紧密融合
 - 结对编程 (pair-programming)
 - 等等
- 开发流程：编写用户故事、架构规范、实施规划、迭代计划、代码开发、单元测试、验收测试
- 积极接受变化



原则与做法

价值观与原则

- 互动与交流
- 反馈
- 简单
- 勇气
- 团队

核心做法

- 小规模，频繁的版本发布，短迭代周期
- 测试驱动开发（Test-driven development）
- 结对编程（Pair programming）
- 持续集成（Continuous integration）
- 每日站立会议（Daily stand-up meeting）
- 共同拥有代码（Collative code ownership）
- 系统隐喻（System metaphor）

每日站立会议



Scrum



Scrum是一个包括了一系列的实践和预定义角色的过程骨架（是一种流程、计划、模式，用于有效率地开发软件）。



SCRUM过程

Jeff Sutherland
Ken Schwaber

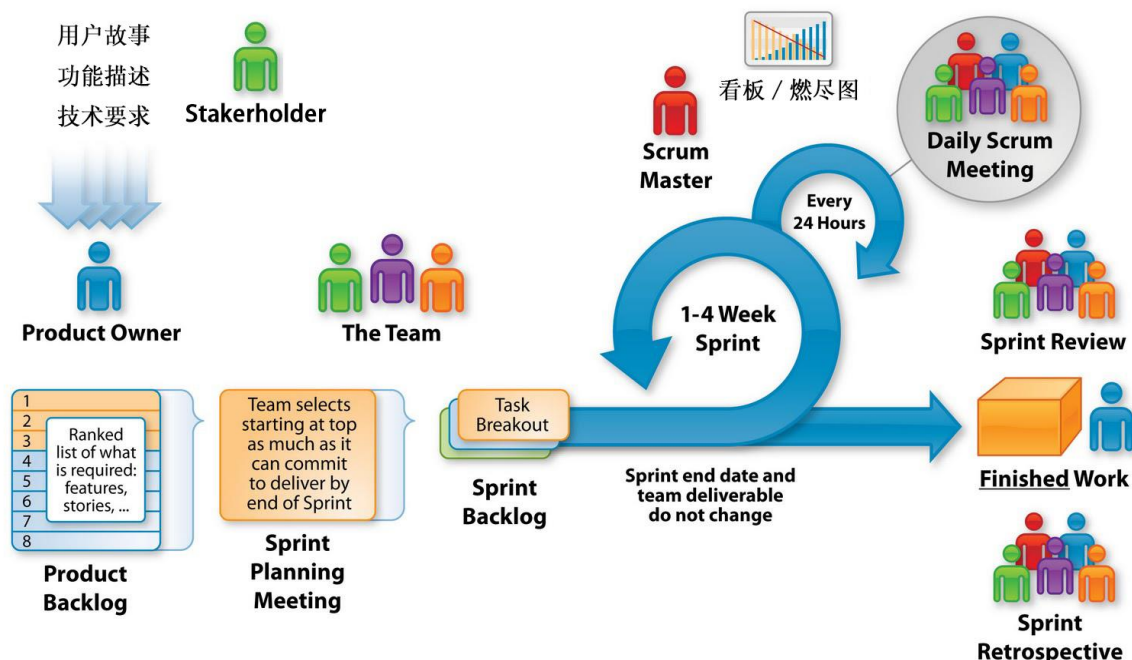
- Scrum注重过程,
XP注重实践

- 需求被定义为产品需求积压
(product backlogs)





- 开发过程分为多个冲刺 (Sprint) 周期

- 燃尽图 (burn down) 是一个公开展示的图表, 显示当前冲刺中未完成的数目

- 产品增量 Increment: 最终交付给客户的内容



SCRUM冲刺

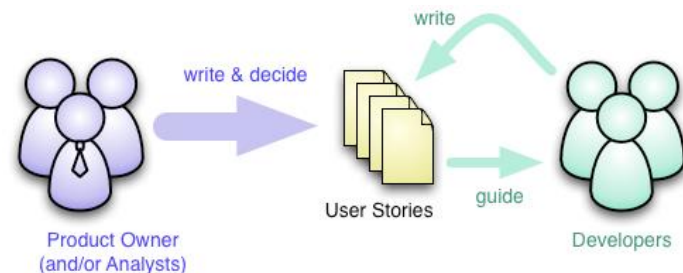
- 
- **冲刺 Sprint:** 一个时间周期（通常在2周到1个月之间），开发团队会在此期间内完成所承诺的一组订单项的开发。
- 
- 
- 

SCRUM核心工件

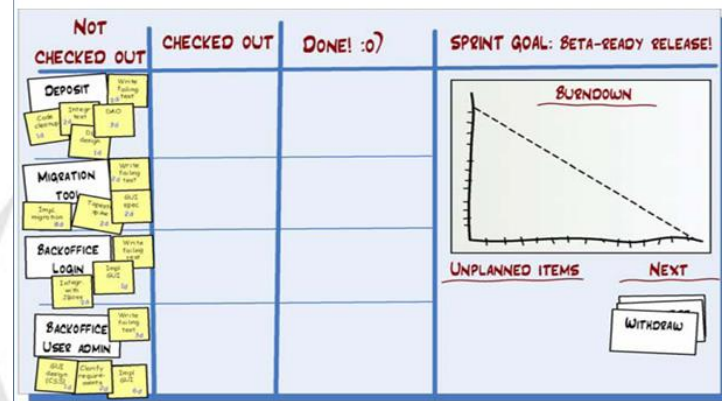
- **产品订单（product backlog）是整个项目的概要文档。**产品订单包括所有所需特性的粗略的描述。产品订单是关于将要创建的什么产品。产品订单是开放的，每个人都可以编辑。产品订单包括粗略的估算，通常**以天为单位**。估算将帮助产品负责人衡量时间表和优先级冲刺订单
- **冲刺订单（sprint backlog）是大大细化了的文档，包含团队如何实现下一个冲刺的需求的信息。**任务被分解为以小时为单位，没有任务可以超过**16个小时**。如果一个任务超过16个小时，那么它就应该被进一步分解。冲刺订单上的任务不会被分派，而是由团队成员签名认领他们喜爱的任务。
- **燃尽图（burn down chart）是一个公开展示的图表，显示当前冲刺中未完成任务数目，或在冲刺订单上未完成的订单项的数目。**不要把燃尽图与挣值图相混淆。燃尽图可以使'冲刺(sprint)'平稳的覆盖大部分的迭代周期，且使项目仍然在计划周期内。

SCRUM角色

- **产品拥有者 (Product Owner)** : 产品远景规划, 平衡利益相关者利益, 确定产品积压的优先级等。它是开发团队和客户间的联络点。
- **利益相关者 (Stakeholder)** : 客户或最终用户代表, 收集编写产品需求, 审查项目成果等。
- **专家 (Scrum Master)** : 指导进行Scrum开发与实践, 是开发团队与产品拥有者间的联络点。
- **团队成员 (Team Member)** : 项目开发人员。



任务看板



SCRUM活动

- **计划会 Sprint Planning Meeting**: 在每个冲刺之初，由产品负责人讲解需求，并由开发团队进行估算的计划会议。
- **每日立会 Daily Standup Meeting**: 团队每天进行沟通的内部短会，因一般只有15分钟且站立进行而得名。
- **评审会 Review Meeting**: 在冲刺结束前给产品负责人演示并接受评价的会议。
- **反思会/回顾会 Retrospective Meeting**: 在冲刺结束后召开的关于自我持续改进的会议。

敏捷方法之极限编程(XP)和 Scrum区别

• 区别之一：迭代长度的不同

- XP的一个Sprint的迭代长度大致为1~2周, 而Scrum的迭代长度一般为 2~ 4周。

• 区别之二: 在迭代中, 是否允许修改需求

- XP在一个迭代中, 如果一个User Story(用户素材, 也就是一个需求)还没有实现, 则可以考虑用另外的需求将其替换, 替换的原则是需求实现的时间量是相等的。而Scrum是不允许这样做的, 一旦迭代开工会完毕, 任何需求都不允许添加进来, 并有Scrum Master严格把关, 不允许开发团队受到干扰。

• 区别之三: 在迭代中, User Story是否严格按照优先级别来实现

- XP是务必要遵守优先级别的。但Scrum在这点做得很灵活, 可以不按照优先级别来做, Scrum这样处理的理由是: 如果优先问题的解决者, 由于其它事情耽搁, 不能认领任务, 那么整个进度就耽误了。另外一个原因是, 如果按优先级排序的User Story #6和#10, 虽然#6优先级高, 但是如果#6的实现要依赖于#10, 则不得不优先做#10。

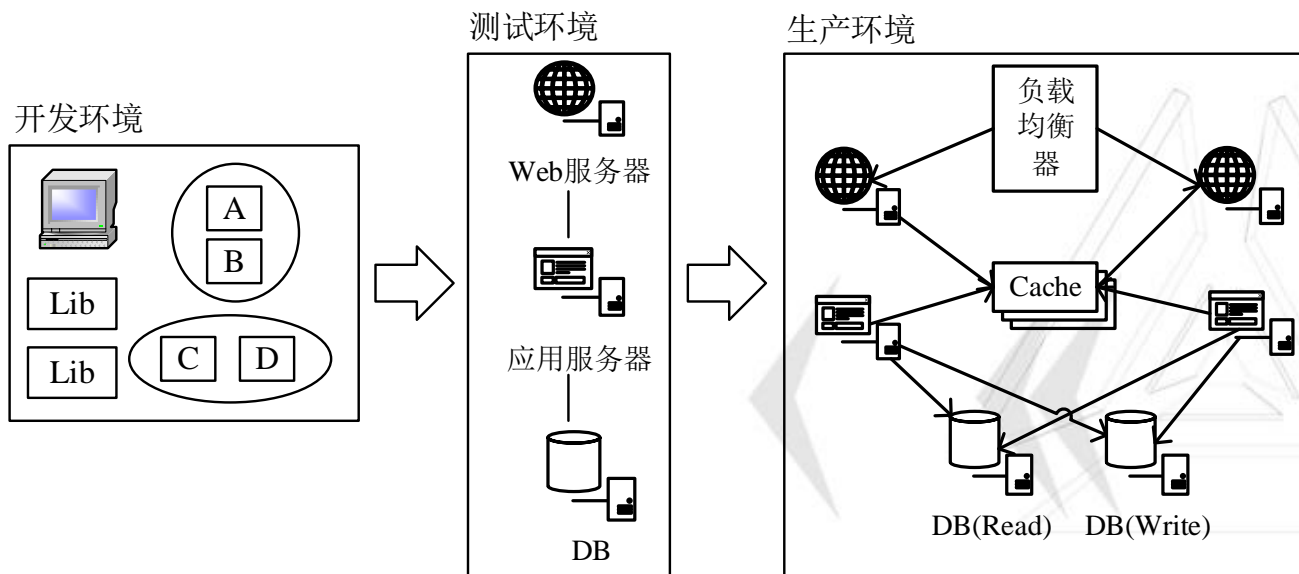
• 区别之四: 软件的实施过程中, 是否采用严格的工程方法, 保证进度或者质量

- Scrum没有对软件的整个实施过程开出工程实践的处方, 要求开发者自觉保证。但XP对整个流程方法定义非常严格, 规定需要采用TDD、自动测试、结对编程、简单设计、重构等约束团队的行为。

DevOps过程

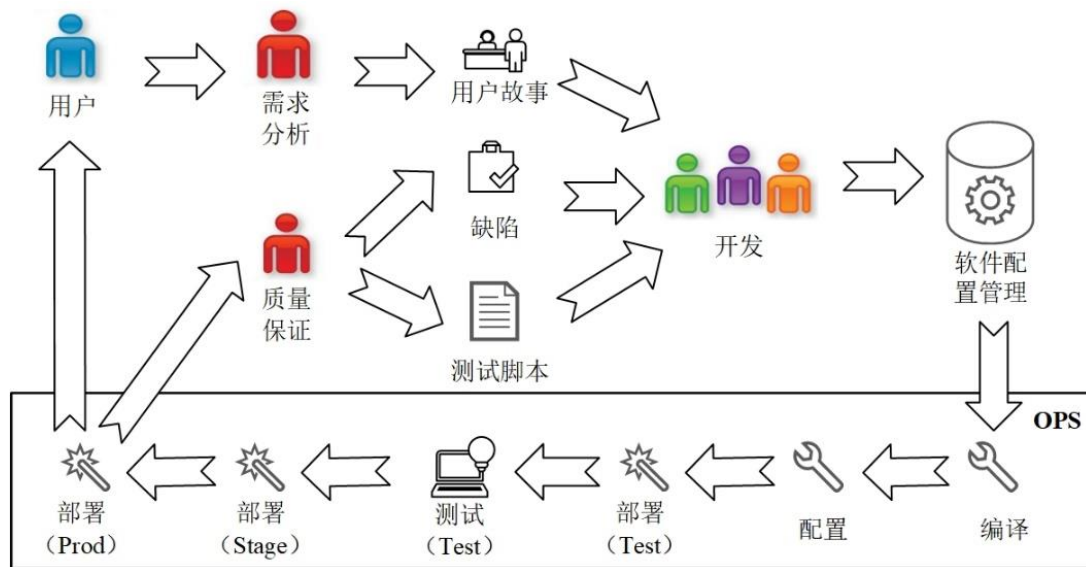


- **DevOps:** 开发(Development)和运维(Operations), 源于敏捷开发过程, 遵从基本的敏捷宣言, 强调了“个体和交互胜过流程和工具”的作用, 但却不局限于某一种软件过程, 甚至是在瀑布模型中也会发挥作用。
- **DevOps**是一组过程、方法与系统的统称, 用于促进开发、技术运营和质量保证部门之间的沟通、协作与整合。
- 强调开发和运维必须紧密合作。

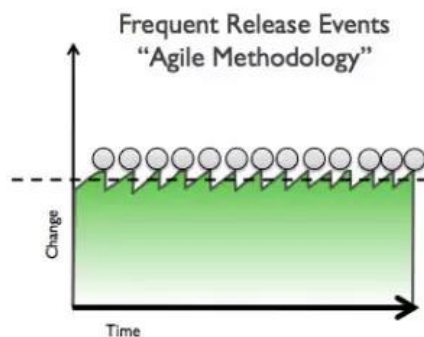


DevOps过程

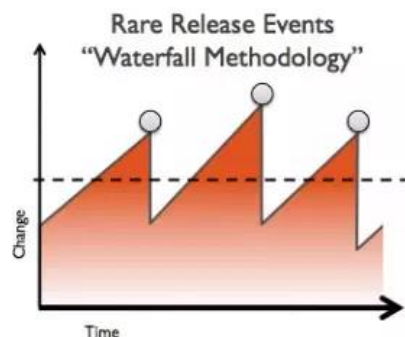
- DevOps的核心目标是自动化和可持续交付。
- 实现全生命周期的工具全链路打通与自动化、跨团队的线上协作能力，从而全面提高生产环境的可靠性、稳定性、弹性和安全性。



- 纵向集成打通了应用全生命周期
- 横向集成打通了架构、开发、管理、运维等部门墙

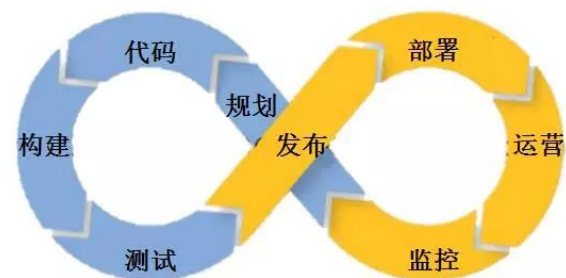


Smoother Effort
Less Risk



Effort Peaks
High Risk

DevOps能力环



无尽头的可能性：DevOps涵盖了代码、部署目标的发布和反馈等环节，闭合成一个无限大符号形状的DevOps能力闭环。

高度依赖工具

- 无论是纵向集成还是横向集成，DevOps都需要通过**工具链**与持续集成、交付、反馈与优化进行端到端整合。
- 华为基于二十几年的研发实践，并融合DevOps等理念方法，打造了软件开发云服务，希望为企业提供一站式的云上开发工具平台。华为软件开发云提供了**项目管理、配置管理、代码检查、编译构建、测试、部署、发布**等端到端地覆盖软件生命周期的相关服务
- **FlowDock或HipChat**

DevOps发展

- 在云计算、大数据等技术颠覆性趋势继续在应用经济下发挥作用的同时，**DevOps**也已经稳健地在业务思维方式中占有一席之地。
- 在应用驱动、云连接、移动化的大环境下，**DevOps**战略将助力业务增值

作业

- 习题1~3

