Measures

软件生命周期模型

麦哲思科技（北京）有限公司

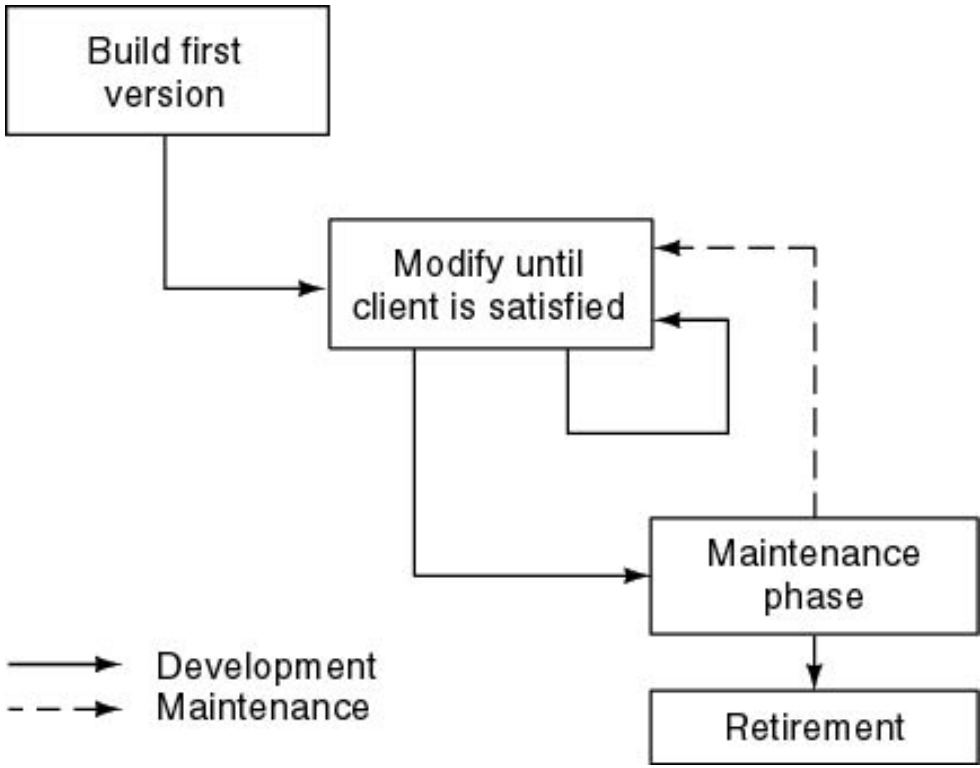
内容

Measures

- 边做边改模型
- 瀑布模型
- 快速原型法
- 增量模型
- 迭代模型
- 螺旋模型
- 统一软件过程
- 选择生命周期模型的原则
- 生命周期模型的描述

边做边改模型

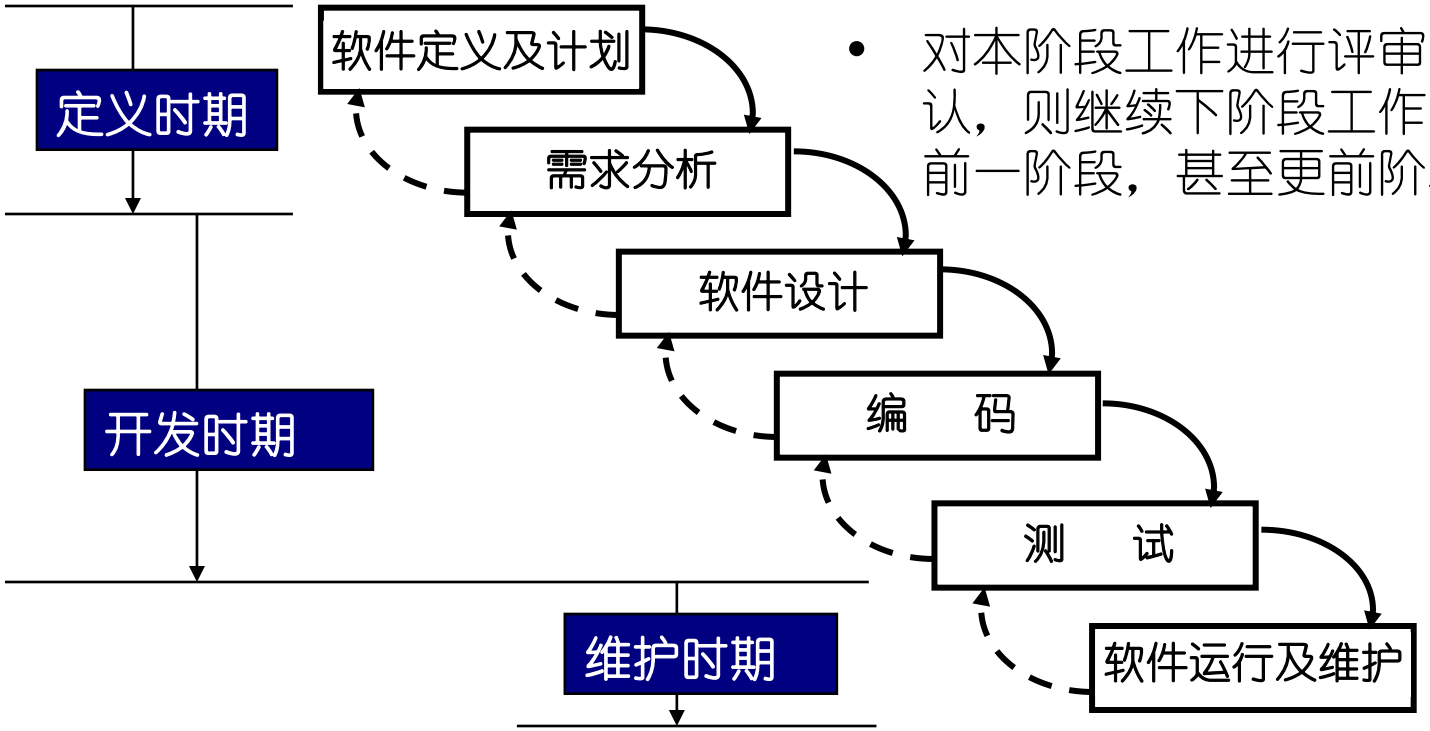
- 问题
  - 没有规格说明或设计
  - 不断修改直到客户满意
- 总体满意度很低
- 需要生命周期模型
  - 计划活动
  - 分阶段
  - 里程碑
- 最简单的开发方法也是最昂贵的开发方法



瀑布模型

每个阶段均具有以下特征：

- 从上一阶段接受本阶段工作的对象，作为输入；
- 对上述输入实施本阶段的活动；
- 给出本阶段的工作成果，作为输出传入下一阶段；
- 对本阶段工作进行评审，若得到确认，则继续下阶段工作，否则返回前一阶段，甚至更前阶段。



瀑布模型的优点

Measures

- 通过文档强制规范化的开发
  - 只有完成的了文档并且被检查通过后，一个阶段才可以结束，强调开发的阶段性；
  - 项目的进展具有有形的证据
  - 使项目阶段易于控制
- 强调早期计划及需求调查；
- 可以对人员进行明确分工，不同的人员在不同的阶段：
  - 系统分析员做分析，
  - 架构师做设计，
  - 程序员做编码，
  - 测试人员做确认等.
- 由于需求已明确，所以不需要代码重构等方面的开销，因此效率较高
- 最小的变化，最大的预见能力

瀑布模型的缺点

Measures

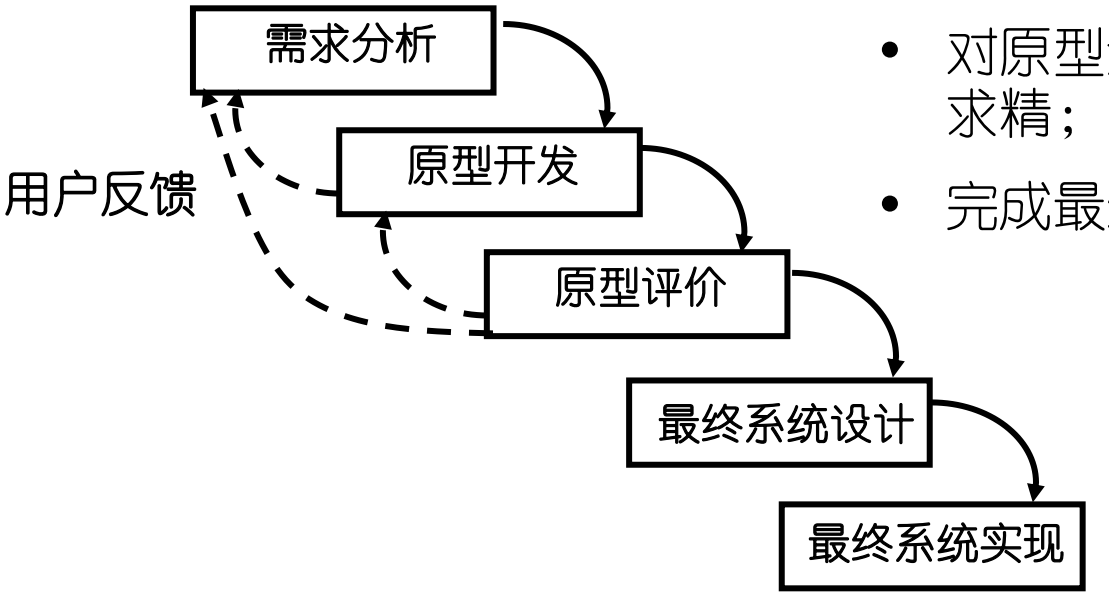
- 文档驱动的模型
  - 客户无法理解文档
  - 产生大量的文档
- 客户直到最后才能看到软件
- 依赖于早期进行的唯一一次需求调查，难以适应需求的变化
- 由于是单一流程，开发中的经验教训不能反馈应用于本产品的过程；
- 风险往往迟至后期的开发阶段才显露，因而失去及早纠正的机会。
- 不符合人们认识事物的客观规律
- 没有体现在开发过程中的并行活动

IBM公司的开发模型

Measures

- 概念阶段
- 计划阶段
- 开发阶段
- 质量阶段（测试）
- 发布阶段
- 支持阶段

原型模型



具有以下特征：

- 用户需求不完全或不确定；
- 针对总体的轮廓先建立一个用户需求原型，然后进行评价；
- 对原型进行扩充、改进和求精；
- 完成最终系统

软件产品的原型有多种形式

Measures

- 丢弃型：原型无需保留而废弃。
- 演示型：以演示为目的，往往用在软件产品的用户界面的开发上。
- 样品型：规模与最终产品相似，原型仅供研究用。
- 增长式演化型：原型作为最终软件产品的一部分，它可满足用户的部分需求，进一步在此基础上开发，则可增加需求，整个实现后交付使用。
- 粗陋型：就较短时间开发的简易原型。

原型的用途

Measures

- 阐明并提炼范围和需求
  - **需求阶段快速原型，其他阶段瀑布模型**
- 详细说明将要建立的可视化界面
- 研究已察觉的项目的高风险元素
- 证实物理设计
- 展示并告诉管理人员、潜在的用户和策划组
- 提供给用户一个可操作的原型以得到反馈

迭代模型

Measures

- 该模型主要针对事先不能完整定义需求的软件开发。
- 用户可以给出待开发系统的核心需求，并且当看到核心需求实现后，能够有效地提出反馈，以支持系统的最终设计和实现。
- 软件开发人员根据用户的需求，首先开发核心系统。当该核心系统投入运行后，用户试用之，完成他们的工作，并提出精化系统、增强系统能力的需求。
- 每一迭代过程都是一个瀑布模型

迭代模型的优点

Measures

- a. 任何功能一经开发就能进入测试以便验证是否符合产品需求。
- b. 帮助导引出高质量的产品要求。如果没有可能在一开始就弄清楚所有的产品需求，它们可以分批取得。而对于已提出的产品需求，则可根据对现阶段原型的试用而作出修改。
- c. 风险管理可以在早期就获得项目进程数据，可据此对后续的开发循环作出比较切实的估算。提供机会去采取早期预防措施，增加项目成功的机率。
- d. 大大有助于早期建立产品开发的配置管理，产品构建（build），自动化测试，缺陷跟踪，文档管理。均衡整个开发过程的负荷。
- e. 开发中的经验教训能反馈应用于本产品的下一个循环过程，大大提高质量与效率。
- f. 如果风险管理发现资金或时间已超出可承受的程度，则可以决定调整后续的开发，或在一个适当的时刻结束开发，但仍然有一个具有部分功能的，可工作的产品。
- g. 心理上，开发人员早日见到产品的雏型，是一种鼓舞。
- h. 使用户可以在新的一批功能开发测试后，立即参加验证，以便提供非常有价值的反馈。
- i. 可使销售工作有可能提前进行，因为可以在产品开发的中后期取得包含了主要功能的产品原型去向客户作展示和试用。

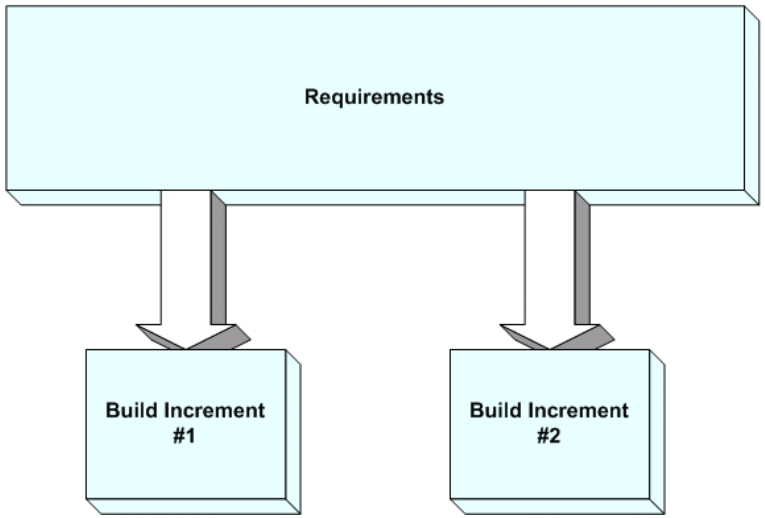
迭代模型的缺点

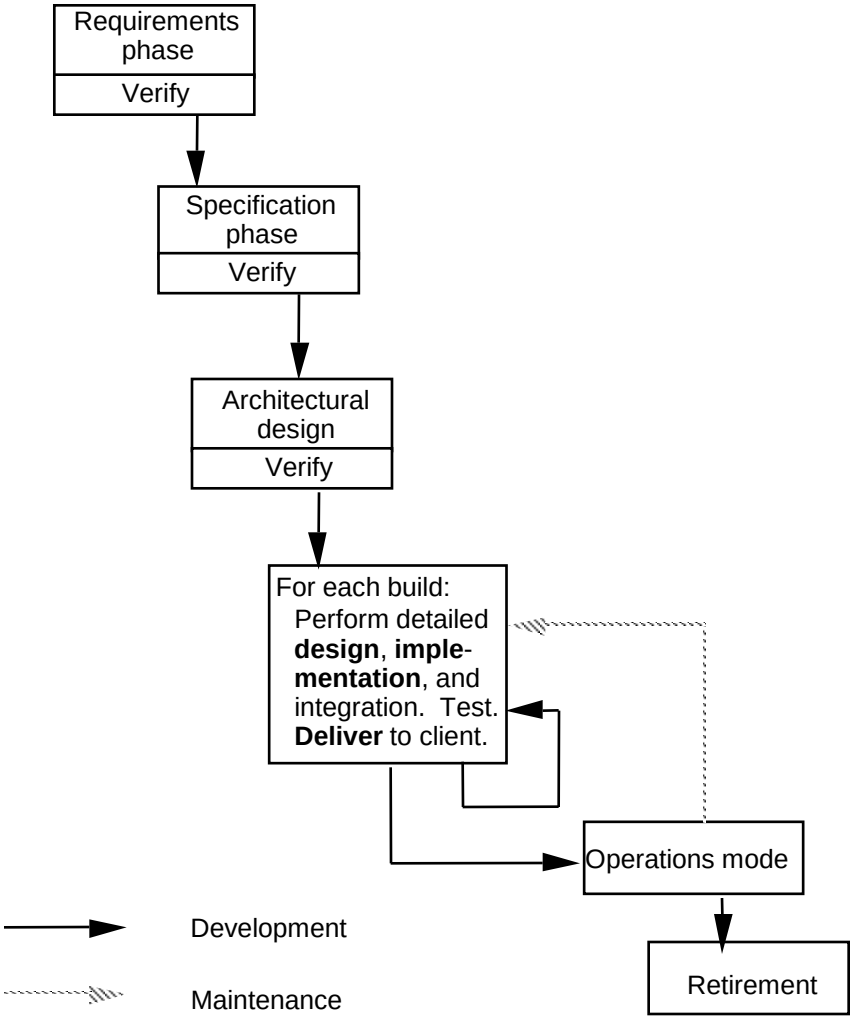
Measures

- a. 如果所有的产品需求在一开始并不完全弄清楚的话，会给总体设计带来困难及削弱产品设计的完整性，并因而影响产品性能的优化及产品的可维护性。
- b. 如果缺乏严格的过程管理的话，这个生命周期模型很可能退化为一种原始的无计划的“试－错－改”模式。
- c. 心理上，可能产生一种影响尽最大努力的想法，认为虽然不能完成全部功能，但还是造出了一个有部分功能的产品。
- d. 如果不加控制地让用户接触开发中尚未测试稳定的功能，可能对开发人员及用户都产生负面的影响。

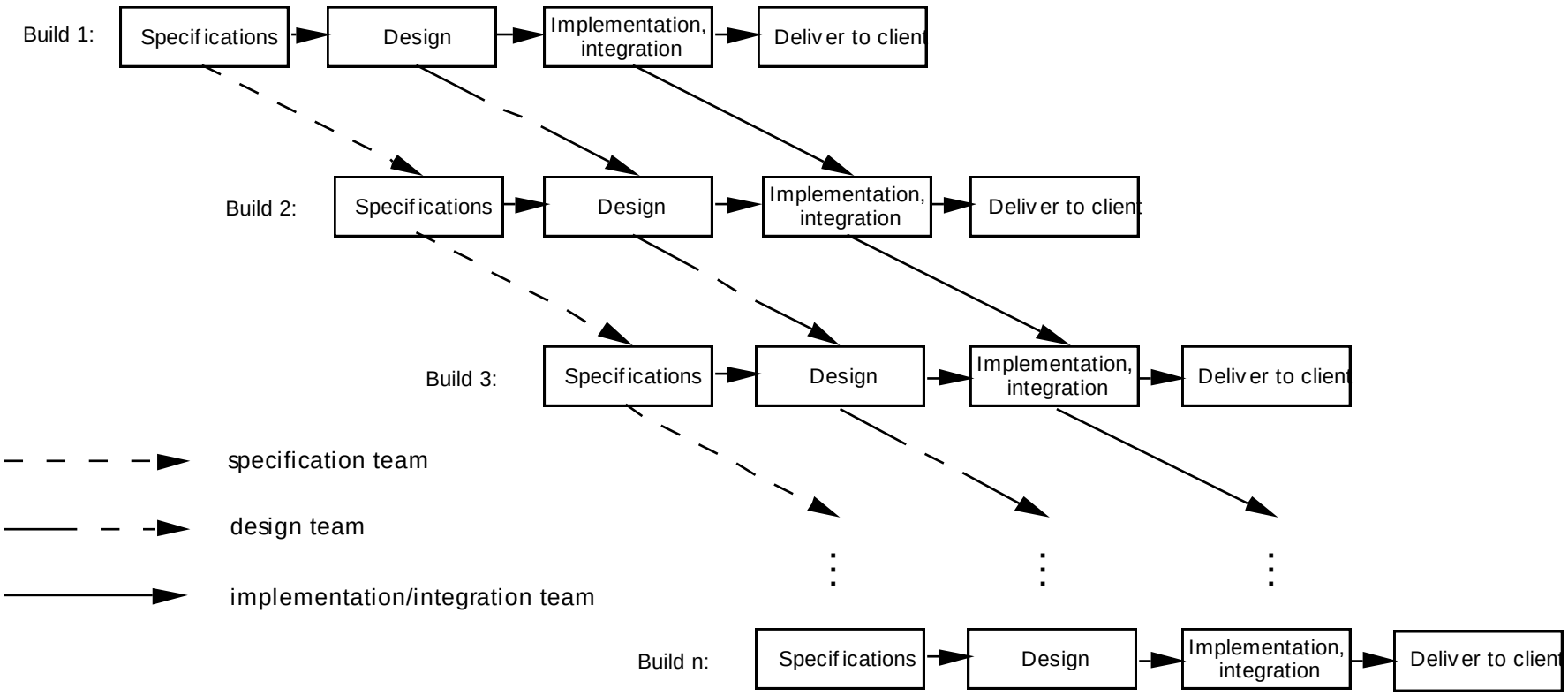
增量模型

- 将项目划分为多个*builds*
  - 每次增加新的功能
  - 每次build进行集成





其他增量模型


Build 1:
Specifications
Design
Implementation, integration
Deliver to client
Build 2:
Specifications
Design
Implementation, integration
Deliver to client
Build 3:
Specifications
Design
Implementation, integration
Deliver to client
specification team
design team
implementation/integration team
Build n:
Specifications
Design
Implementation, integration
Deliver to client

增量模型的优缺点

Measures

- 优点
  - 短期交付，增量交付
  - 现金流量比较低，快速的投资回报
  - 并行节省了时间
  - 关注核心价值
  - 在早期发现软件中的缺陷
  - 在早期检验结构的稳定性
- 缺点
  - 初期设计风险
  - 需要开放的体系结构
  - 边做边改的危险

增量模型与迭代模型的比较

Measures

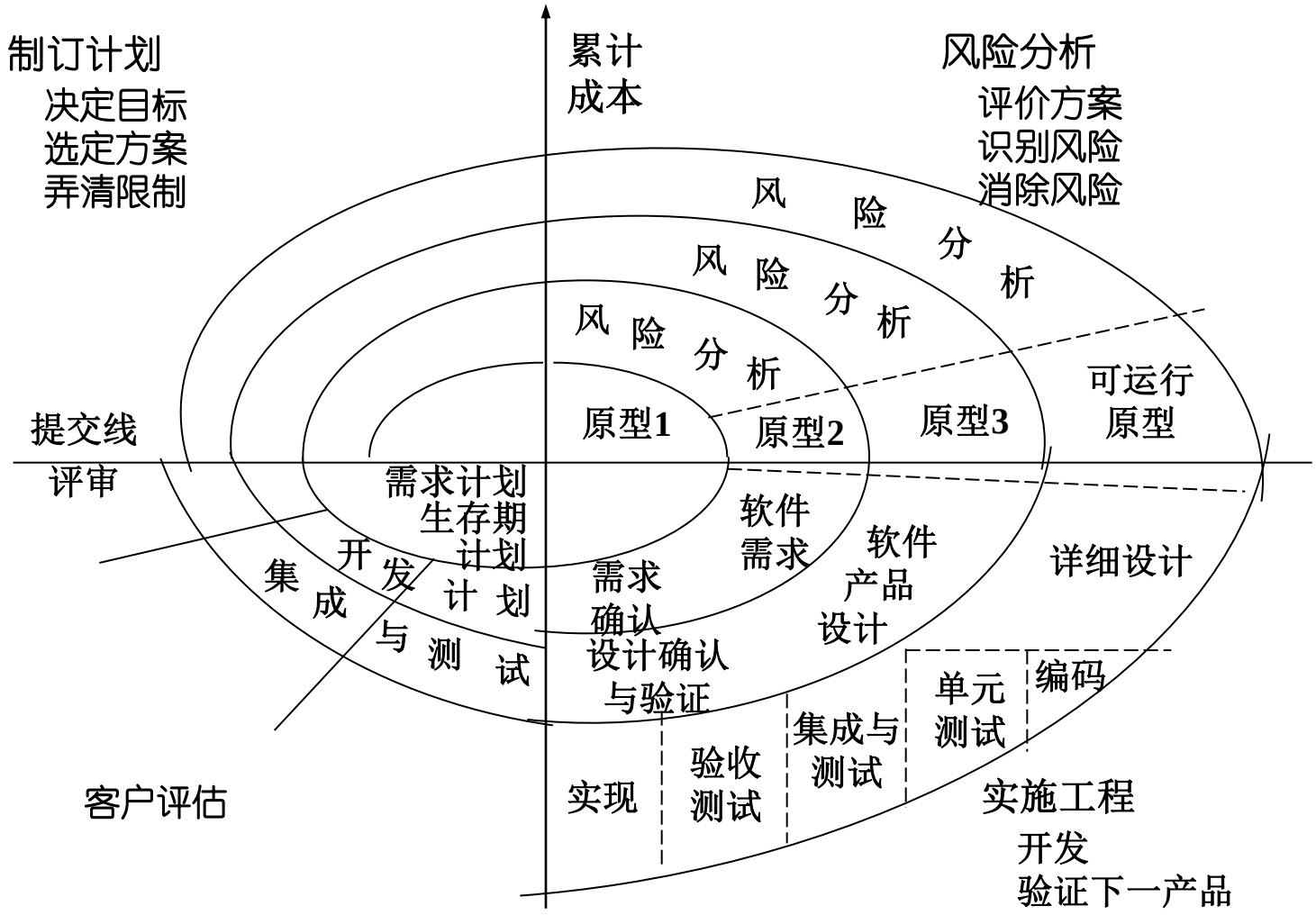
- 听起来类似,有时是相同的
- 区别:
  - 增量:每个阶段增加新的功能
  - 迭代:每个阶段修改功能
- 例子:盖房子
  - 增量: 从一个普通的房子开始,增加房间.
  - 迭代:每次迭代,房间被重新设计并重建
  - 区别:人们可以始终居住在一个增量型的房间中,而必须搬迁到一个新的迭代型的房间中

螺旋模型

- Barry Boehm 1988年在TRW公司提出的一种迭代模型
- 瀑布模型与迭代模型相结合，并加入两者所忽略的风险分析所建立的一种软件开发模型。
- 它将软件项目开发分别划分为四类活动，沿着螺线旋转，在笛卡儿坐标的四个象限上分别表达了四个方面的活动，即：
  - 制订计划：确定软件目标，选定实施方案，弄清项目开发的限制条件；
  - 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。
- 沿螺线自内向外旋转，每旋转一圈便开发出更为完善的一个新的软件版本

Prof. Barry Boehm

螺旋模型


制订计划
决定目标
选定方案
弄清限制
累计
成本
风险分析
评价方案
识别风险
消除风险
风险分析
风险分析
风险分析
原型1
原型2
原型3
可运行
原型
提交线
评审
需求计划
生存期
计划
开发
计划
集成
与
测试
需求
确认
软件
需求
软件
产品
设计
详细设计
设计确认
与验证
单元
测试
编码
集成与
测试
验收
测试
实现
客户评估
实施工程
开发
验证下一产品

螺旋模型

Measures

- 螺旋模型通常用以指导大型软件项目的开发，如果开发风险过大，开发者和客户无法承受，项目有可能因此终止。多数情况下会沿着螺线继续下去，自内向外逐步延伸，最终得到满意的软件。
- 如果对所开发项目的需求已有了较好的理解或较大的把握，无需开发原型，便可采用普通的瀑布模型。这在螺旋模型中可认为是单圈螺线。
- 相反，如果对所开发项目的需求理解较差，需要开发原型，甚至需要不止一个原型的帮助，那就要经历多圈螺线。在这种情况下，外圈的开发包含了更多的活动。也可能某些开发采用了不同的模型。

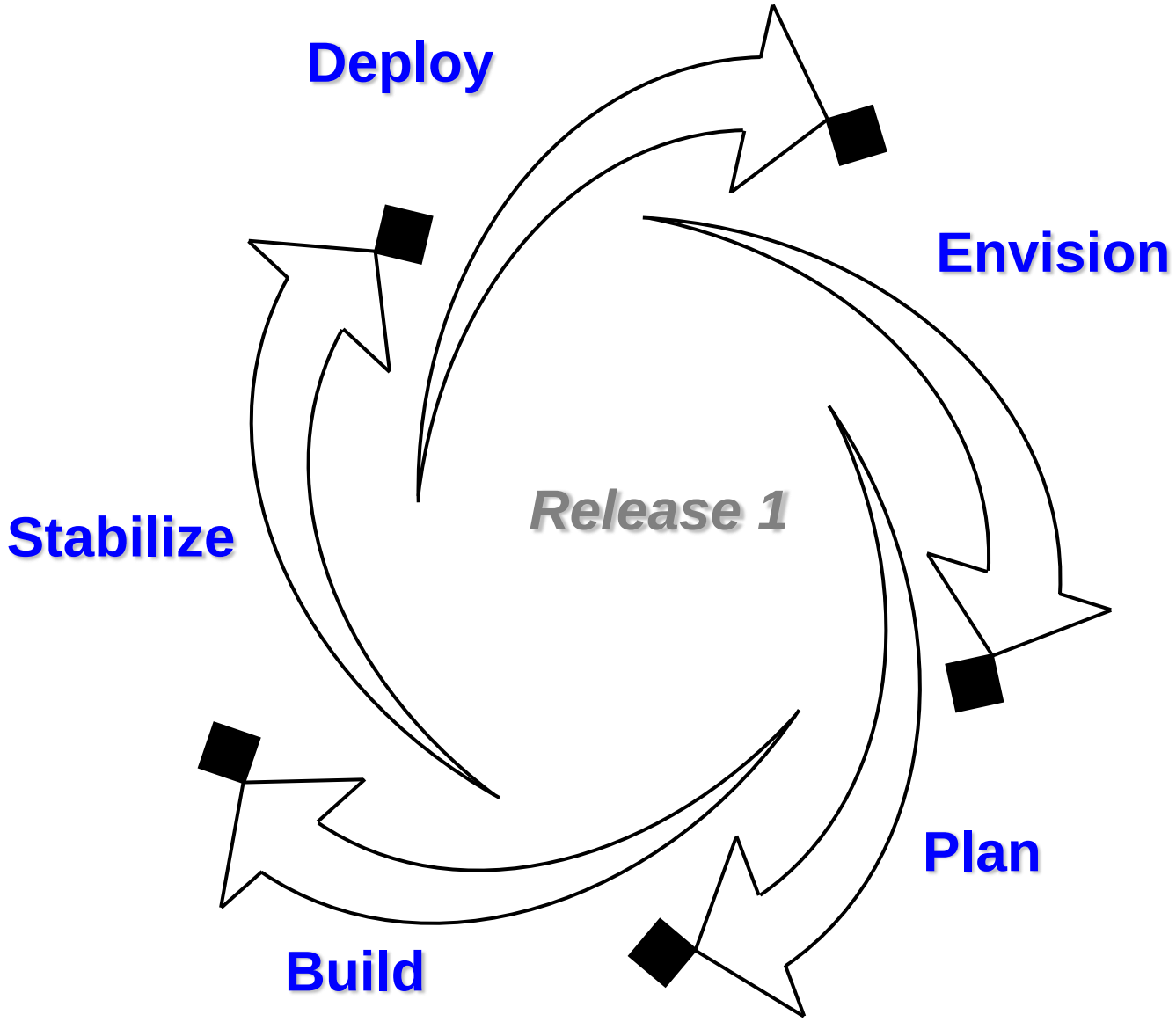
螺旋模型的优缺点

Measures

- 优点：
  - 强调严格的全过程风险管理。
  - 强调各开发阶段的质量。
  - 提供机会检讨项目是否有价值继续下去
  - 没有区分开发与维护
- 缺点：
  - 需要有经验的人员进行风险评估
  - 风险评估的成本比较高，而且需要花费时间
  - 适合于大项目
  - 适合于非订单项目

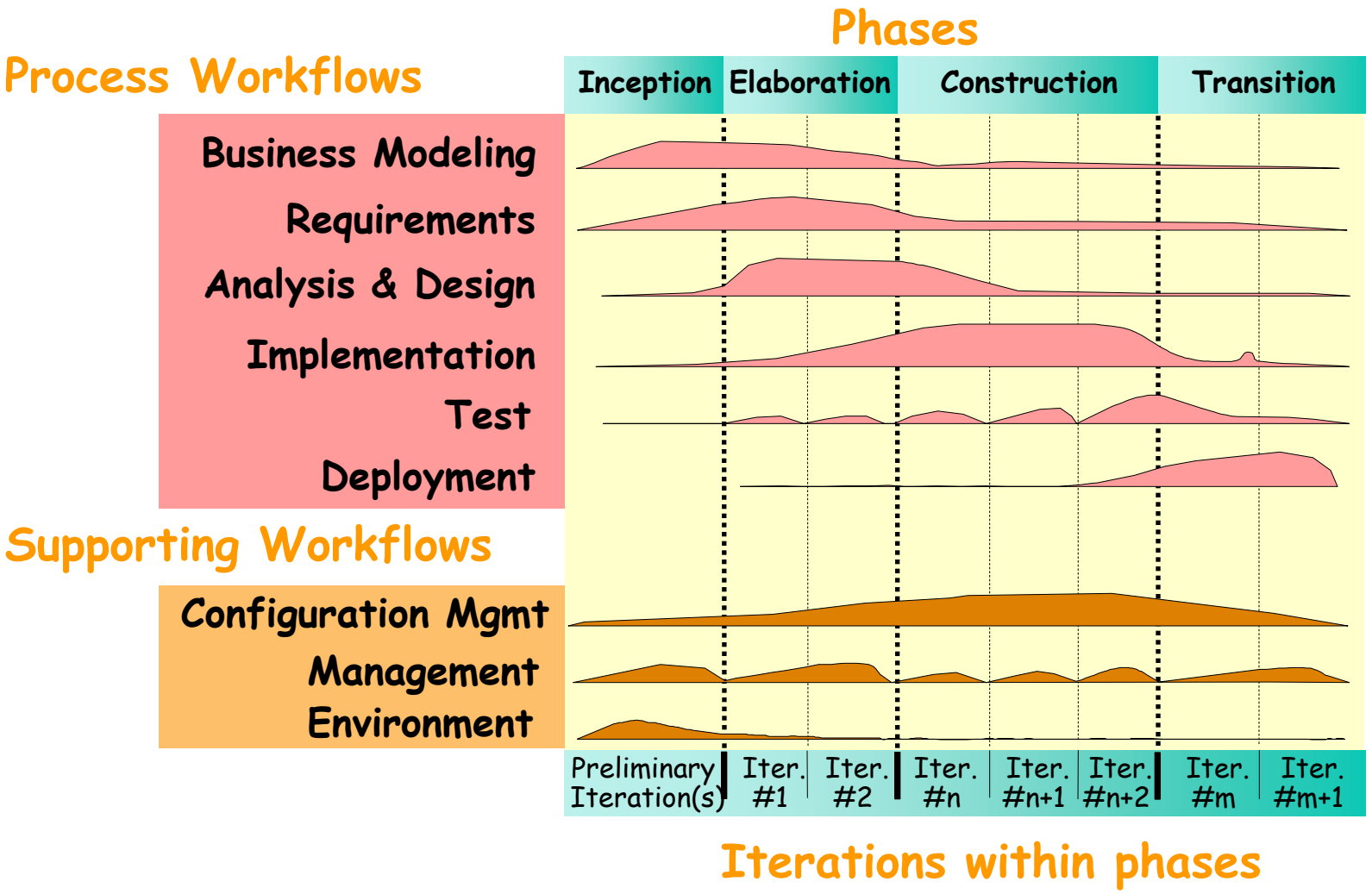
微软的过程模型

Measures

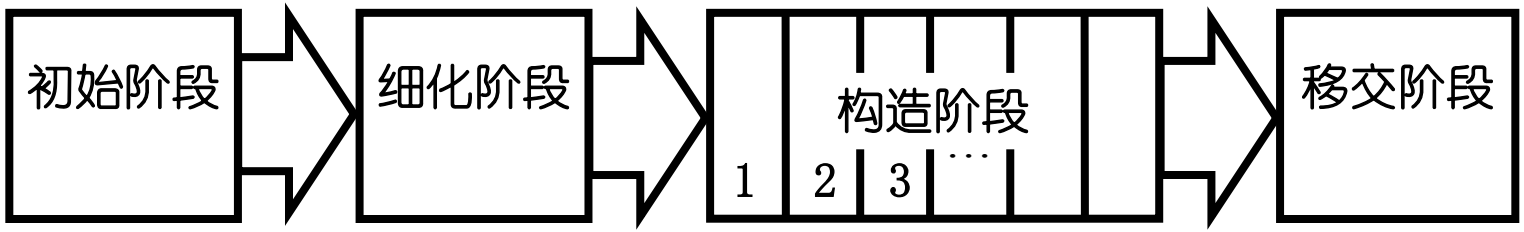


统一软件开发过程

Measures



USDP的高层视图 - 1



- USDP包含初始、细化、构造和移交等四个阶段
  - 其中构造阶段由多次迭代所组成，是一个迭代增量式的开发过程，即不是在项目结束时一次性提交软件，而是分块逐次开发和提交软件

USDP的高层视图 – 2

Measures

- 初始阶段，由1-2次迭代组成，核心问题是项目可行性研究，这一阶段需要与客户进行讨论
  - 需要考虑项目的效益
  - 确定项目的适用范围
- 细化阶段，由2-4次迭代组成，核心问题是建造一个高层构架
  - 收集详细的需求，进行高层分析和设计
  - 为构造阶段制定计划：选择一些功能点，完成这些功能；随后再选择别的功能点，完成这些功能；如此循环往复
- 构造阶段，由多次迭代组成，核心问题是建造一个实用的系统
  - 每一次迭代所得到的产品应满足项目需求的某一个子集
  - 提交给早期用户或是内部提交
- 移交阶段由一次或多次迭代组成

迭代式开发项目的典型时间线

Measures

- 不要把“各阶段”理解为每个阶段所经历的时间是等同的；阶段时间的长短需要根据具体情况具体分析。每个阶段的目标和里程碑才是最重要的
- 一个为期两年的项目的典型时间线：
  - 2.5个月的初始阶段（10%）
  - 7个月的细化阶段（30%）
  - 12个月的构造阶段（50%）
  - 2.5个月的移交阶段（10%）
- 在每个阶段中，都可以迭代地进行开发，包含一个或几个迭代过程

USDP的特征

- 基于UML、以构架为中心、用况驱动与风险驱动相结合的迭代增量式软件开发过程
  - 每一次迭代都包含整个软件生命周期，每次迭代中的软件开发工作都围绕需求捕获用况、分析、设计、实现和测试等9个核心工作流来组织
  - 职责明确：每个人都明白自己的职责；开发者能更好理解其他开发者的职责；管理人员能了解开发者的职责
  - 过程规范：单位对自身员工的培训标准化；开发人员和管理人员不需要学习新的过程就能在组间调动工作
  - 软件开发过程可复用：复用开发技术和管理技术；项目计划更准确，成本估算与实际更吻合，开发周期更短

USDP是用例驱动的

Measures

- 用例模型描述了系统的全部功能。说明对每个用户系统应该做什么？
- 用例不只是一种确定系统需求的工具，还能驱动系统设计、实现和测试的进行。基于用例模型，创建一系列实现这些用例的设计和实现模型。
- 开发人员可以审查每个后续建立的模型是否与用例模型一致。测试人员测试实现以确保实现模型的构件正确实现了用例

USDP是以构架为中心的

Measures

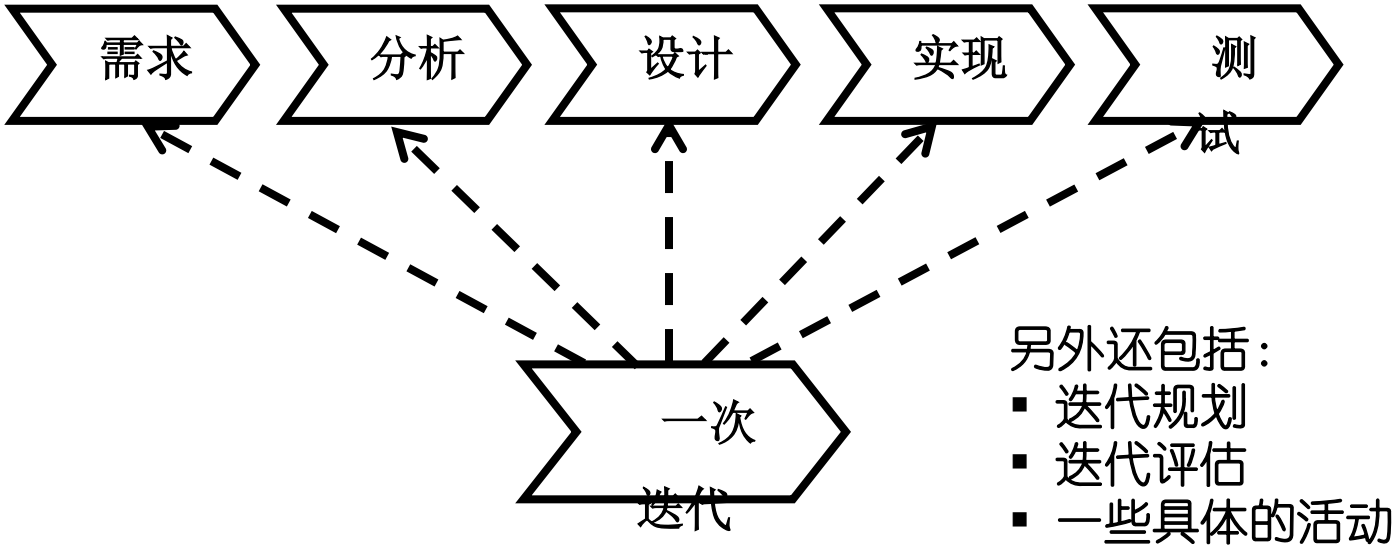
- 软件构架概念包含了系统中最重要的静态和动态特征。构架刻画了系统的整体设计，它去掉了细节部分，突出了系统的重要特征
- 系统的构架是所有工作人员必须达成或至少能够接受的共同目标。构架为我们提供了整个系统的清晰的视角
- 构架描述最为重要的模型元素，能够指导系统的开发，重要的模型元素包括子系统、相关性、接口、协作、节点和主动类。它们描述了系统中的基础部分，据此可以更为有效地理解、开发并改进这个系统

USDP是迭代和增量的

- 开发一个商用软件产品是一项艰巨的任务，将这项工作划分为较小的部分或袖珍项目是切实可行的
- 每个袖珍项目都是一次能够产生一个增量的迭代过程。迭代指工作流中的步骤，增量指产品中增加的部分。
- 迭代过程要处理一组用例，以扩展所开发产品的可用性。后续的迭代过程建立在前一次迭代过程末期所开发出的制品之上
- 一个增量不一定是对原有制品的增加，尤其在生命周期初期，可能是最初简单设计的细化和完善。在以后的阶段，增量通常是对原有制品的增加

USDP是迭代和增量的

- 一次迭代是一个能够产生内部版本的袖珍项目，或多或少要经历所有的核心工作流（需求、分析、设计、实现和测试）即迭代工作流。



USDP是迭代和增量的

- 增量是这次迭代的内部版本与上次迭代的内部版本之间的差别
- 在迭代过程的最后，代表系统的模型集处于一种特定的状态，这种状态称为基线
- 在迭代序列内的任一点，有些子系统已经完成了，有些子系统只是部分完成了，还有一些子系统仍然空着。增量是两次相继的基线之间的差别。

Measures

选择准则

选择生命周期模型的考虑因素

Measures

- 项目需求清晰性、完整性、稳定性
- 项目规模
- 项目类型
- 软件系统复杂度
- 项目用到的新技术
- 项目成员的技能
- 其它因素

选择瀑布模型的原则和软件特点

Measures

- 在开发时间内需求没有或很少变化；
- 分析设计人员对应用领域很熟悉；
- 合同对完成时间、进度有明确要求；
- 低风险项目（对目标、开发环境很熟悉）；
- 用户应用环境稳定；
- 用户除提出需求以外，很少参与开发工作；

选择迭代模型的原则和软件特点

Measures

- 分析设计人员对应用领域很熟悉；
- 高风险项目（对系统目标、开发环境较熟悉）；
- 用户可不同程度地参与整个项目的开发过程；
- 使用面向对象的语言或第四代语言；
- 具有高素质的项目管理者和软件研发团队。

选择增量模型的原则和软件特点

Measures

- 无严格合同进度约束，对软件面市时间有紧迫需求；
- 项目开发过程中，需求较易发生变化，但主要需求相对稳定；
- 项目组人员是经验丰富的，尤其是项目管理经验；
- 分析、设计人员对应用领域欠熟悉，或难以一步把握；
- 中等或高风险项目，项目组人员有足够的风险管理技能；
- 用户可以参与整个软件开发过程中；
- 对于项目组人员不充分的项目，可以考虑选择增量模型。

选择USDP模型的原则与软件特点

Measures

- 采用面向对象开发方法，开发人员具有足够的知识和技能；
- 项目组人员是经验丰富的，尤其是项目管理经验；
- 分析、设计人员对应用领域比较熟悉；
- 中等或高风险项目，项目组人员有足够的风险管理技能；
- 用户能够提供参与整个项目的开发过程的足够保证；
- 对于重视软件复用的项目，可以考虑采用USDP模型；
- 对于在稳定的产品基础上，增加定义好的功能特性的项目，可以考虑采用USDP模型；
- 对于新领域、需求不熟悉的情形，可以考虑采用USDP模型。

生命周期模型的选择策略

Measures

		纯瀑布	增量模型	原型	纯迭代	RUP	螺旋
需求稳定性	稳定	√	√			核心需求	
	变化			√	√		√
软件规模	大		√	√	√	√	√
	小	√		√			
项目类型	新产品			√	√	√	√
	移植	√	√	√	√	√	
	升级		√	√	√	√	
	维护	√	√		√		
复杂度	复杂			√	√	√	√
	简单	√	√	√			
技术新颖程度	新技术			√	√	√	√
	成熟技术	√	√				
人员技能	经验丰富	√	√			√	
	新手			√	√		√
用户的参与程度	很少参与	√					
	经常参与		√	√	√	√	√
项目风险	高			√	√	√	√
	低	√	√				
项目经理的管理能力	高				√	√	√
	低	√	√	√			

Measures

生命周期模型的描述

软件开发生命周期描述的主要内容

Measures

- 阶段划分
- 每个阶段的输入
- 每个阶段的输出
- 每个阶段的任务
- 每个阶段的工作产品
- 阶段之间的关系
- 任务之间的关系
- 每个阶段或每个任务对应的角色
- 每个阶段的开始条件，结束条件
- 侧重于描述工程活动，而不是管理活动与支持活动

过程定义的框架-1

Measures

- 过程角色（Process roles），哪些角色参与本过程的哪些活动，可以用角色-职责矩阵表示
- 适用的过程和产品标准（Applicable process and product standards），包括企业内的或者企业外的（标准没有，客户的标准、行业的标准等没有，放在项目定义过程中）
- 适用的规程、方法、工具和资源（Applicable procedures, methods, tools, and resources）。资源中包括了关键的设备（资源只列各种模板等，现在体现不出）
- 过程性能目的（Process performance objectives）。可用一些量化数据来表示，如周期、生产率和缺陷排除率等
- 入口准则（Entry criteria）
- 输入（Inputs）：哪些文档是该过程或活动的输入
- 活动，要注意这些活动是否覆盖了模型的要求

过程定义的框架-2

Measures

- 要收集和使用的产品和过程度量（Product and process measures to be collected and used）（在度量规范中说明，但说明属于哪个过程规范）
- 验证点（如同行评审）（Verification points (e.g., peer reviews)）（过程流图中）
- 输出（Outputs）：输出那些文档，要注意这些输出是否覆盖了模型的要求
- 接口（Interfaces）：与其他过程或规程的衔接关系（总体的过程体系结构，相互之间的关系）（体现在过程活动中）
- 出口准则（Exit criteria）：定义了过程或活动应达到什么要求才算结束了

小结

- 选择生命周期模型的考虑因素
  - 项目需求清晰性、完整性、稳定性
  - 项目规模
  - 项目类型
  - 软件系统复杂度
  - 项目用到的新技术
  - 项目成员的技能
  - 其它因素

参考资料

- Stephen R. Schach, Software Engineering with Java,inc.1999. 中译本:《软件工程:JAVA语言实现》,袁兆山等译. 机械工业出版社
- Kent Beck ,Extreme Programming explained: embrace change,inc.2000. 中译本:《解析极限编程—拥抱变化》,唐东铭译
- Ivar Jacobson, etc. The Unified Software Development Process. Addison Wesley Longman, Inc.1999. 中译本:《统一软件开发过程》，周伯生，冯学民，樊东平译。机械工业出版社，2002年1月。
- Philippe Kruchten. The Rational Unified Process: An Introduction (Second Edition). Addison Wesley Longman, Inc.2000.中译本:《Rational统一过程引论》，周伯生，吴超英，王佳丽译。机械工业出版社，2002年5月。

练习--场景描述

Measures

- 美国GiTech公司开发了一套企业管理系统ABC1.0，采用PB开发，使用SYBASE数据库，已经在美国企业中推广了很多套，1.0软件花费了10个人年开发出来。为了开拓中国市场，GiTech公司委托北京Measures公司开发一套适合中国国情的ABC2.0软件，GiTech公司委派了熟悉中文的TOM先生做为GiTech公司的全权代表，负责需求与技术框架的确认，并参与到项目中来，TOM先生参与了ABC1.0软件的开发，对需求与原来的软件设计思想很熟悉。TOM先生每周可以有2天的时间参与项目的活动。Measures公司成立了一支开发团队，项目经理是Dylan先生，技术经理是SUN博士，SUN博士负责技术路线的确定。还有其他8位开发人员全职参与进来，该团队是第一次合作开发软件，公司没有这方面的可复用的构件积累。
- GiTech公司要求该软件必须：
  - 1 采用.net 2.0软件开发。
  - 2 必须对1.0系统的架构与数据结构进行重新设计。
  - 3 必须满足一些新增的特性要求：
    - 所有的术语必须采用中国人熟悉的术语。
    - 能够支持多语种。
    - 采用中国人习惯的报表格式。
    - 对于权限管理模式要更新。
    - 能够支持多种数据库。
    - 提供稳定的API接口，适合将来大规模推广，可以让用户自己开发或修改软件。
  - 4 必须在6个月内提交贝塔版软件。
- 请针对该项目状况，请选择该项目的生命周期模型，定义出项目的主要阶段，每个阶段的主要任务，并说明你的理由。对于场景中不全的信息，你可以自己做出假设。

练习—指令

Measures

- 任务：
  - 根据下面给出的场景，参考关于生命周期模型的讲解，为下面的项目选择生命周期模型。
- 时间：
  - 5分钟准备
  - 30分钟小组讨论
  - 25分钟汇报
- 指令：
  - 1 分组，指定小组汇报人。每组的人员不要超过7个人。
  - 2 各小组讨论项目的生命周期模型。
  - 3 汇报，每小组的汇报时间不要超过5分钟


Measures
Q&A

Measures

谢　谢！