第二章 软件过程

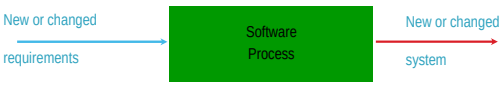
Software Process

软件过程也称为软件生存周期过程，是指软件生存周期中的一系列相关过程。为了获得高质量软件所需要完成的一系列任务的框架，它规定了完成各项任务的工作步骤。

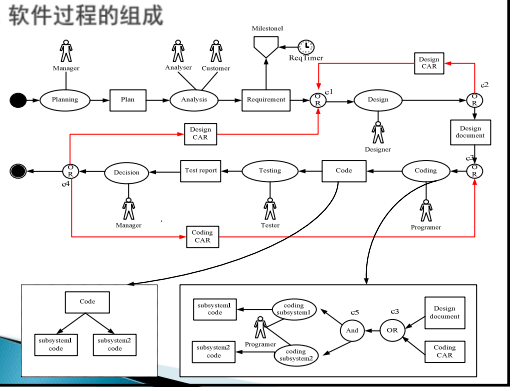
2.1 软件过程基本概念

1. 什么是软件过程

- Defines Who is doing What, When to do it, and How to reach a certain goal.



2. 软件过程的组成

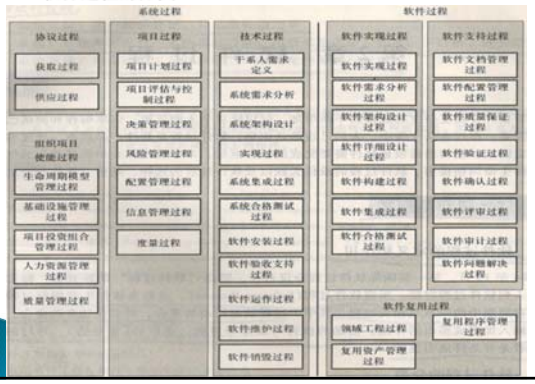


软件过程七大元素：

- 活动
  - 开发、维护、管理等；
- 任务
  - 活动的细分，确定、安排任务等；
- 工件
  - 软件过程的工作产品，分输入与输出工件；
- 角色
  - 定义了软件过程中的个人或小组的行为与职责；
- 资源
  - 最佳实践、工具、技术、机器、场地等；
- 目标
  - 每个过程有明确的目标；
- 度量指标
  - 目标的具体度量与分析，如进度、成本、质量、返工率。

3. 软件过程的分类

ISO/IEC 12207 系统和软件的生命周期过程

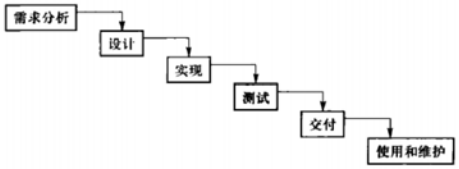


2.2 软件生存周期模型

- 又称软件开发模型，是软件生命周期的一个框架，规定了软件开发、运作和维护等所需的过程、活动和任务。
- 软件生存周期模型分类：
  - 线性顺序模型 Waterfall Model
  - 增量式模型 Incremental Model
  - 演化模型 Evolutionary Model

2.2.1 线性顺序模型
瀑布模型 Waterfall Mode

- 最早的软件开发模型
- 1970年W. Royce提出
- 又称为线性顺序模型



瀑布型特点

- 特点
  - 强调阶段的划分顺序与依赖；
  - 强调各阶段工作文档的完备性，即文档驱动静态描述；
  - 每个阶段从技术和管理进行严格的审查，即质量保证的观点；
  - 是一种线性的、顺序的、逐步细化的开发模式；
  - 推迟实现的观点；
- 适用时机
  - 所有功能、性能等要求能一次理解和描述时
  - 所有的系统功能一次交付时
  - 必须同时淘汰全部老系统时

特点:
具有反馈环

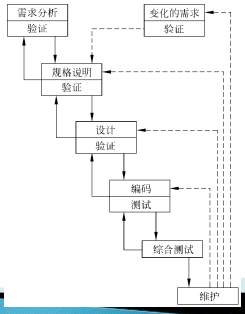


图1.3 实际的瀑布模型

瀑布型的价值

- 结构简单明了；历史较长、应用面广泛、为广大软件工作者所熟悉；已有与之配套的一组十分成熟的开发方法和丰富的支撑工具。
- 一种较为有效的管理模式：订计划、成本预算、组织开发人员,阶段评审,文档管理,从而对软件质量有一定的保证。

瀑布型的风险和缺点

- 获得完善的需求规约是非常困难的；
- 难以适应快速变化需求；
- 系统太大时,难以一次做完；
- 反馈信息慢；
- 极可能引起开发后期的大量返工，如返工到需求、设计等早期活动；
- …

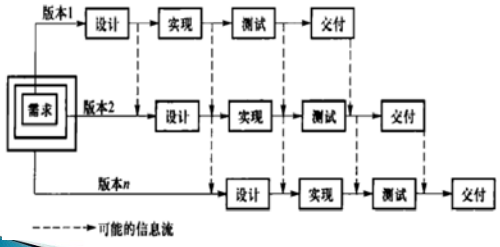
建议不要使用该模型的几种情况：

（1）需求未被充分理解。
（2）系统太大而不能一次开发完成。
（3）事先打算采用的技术迅速发生变化。
（4）需求迅速发生变化。
（5）资源有限，如现有的工作人员/资金不足。
（6）无法利用某一中间产品。

2.2.2 增量模型Incremental Model

软件被分解成许多增量构件，逐个提交。

- 构造一系列可执行的中间版本(Version by Version)



增量型适用时机

- 需要早期获得功能；
- 中间产品可以提供使用；
- 系统被自然地分割成增量；
- 工作人员/资金可以逐步增加。

增量型需考虑的风险

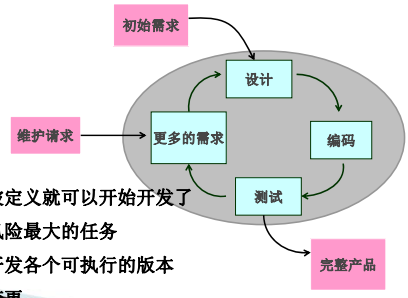
- 需求未被很好地理解
- 一次要求所有功能
- 需求迅速发生变化
- 事先打算采用的技术迅速发生变化
- 长时期内仅有有限的资源（人员/资金）

2.2.3 演化模型(Evolutionary)

- 现状：
  - 软件需求在软件开发过程中常常发生改变，想要一次迭代就开发出最终产品是不可能的
  - 紧迫的市场期限使得难以一下子完成一个完善的软件产品
- 解决方案：演化模型
  - 只要核心需求能够被很好地理解，就可以进行渐进式开发，其余需求可以在后续的迭代中进一步定义和实现。这种过程模型称为演化模型，它能很好地适应随时间演化的产品的开发。
- 特点：
  - 迭代的开发方法，渐进地开发各个可执行版本，逐步完善软件产品。每个版本在开发时，开发过程中的活动和任务顺序地或部分重叠平行地被采用。
  - 与增量模型的区别是：需求在开发早期不能被完全了解和确定，在一部分被定义后开发就开始了，然后在每个相继的版本中逐步完善。

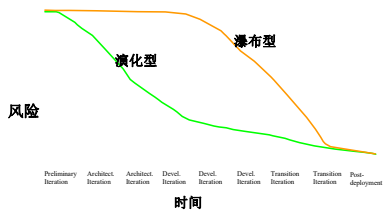
演化模型

理解了核心需求就可以进行渐进开发，其余需求在后续迭代中实现。



- **部分需求被定义就可以开始开发了**
- **首先执行风险最大的任务**
- **渐进迭代开发各个可执行的版本**
- **允许需求变更**

演化型价值：降低风险



演化模型是目前采用最广泛的模型

演化模型已成为主流

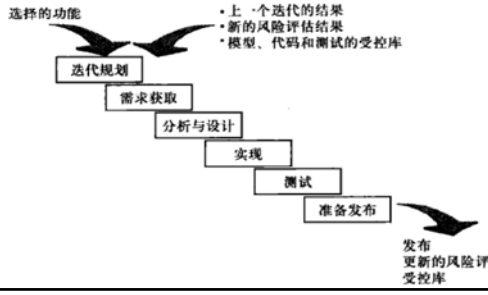
- 现代软件过程都采用演化模型
  - 统一软件过程RUP
  - 敏捷过程 （SCRUM、XP等）
  - 净室（Cleanroom）软件过程
- 演化模型的“子类”
  - 原型 Prototyping
  - 螺旋模型 Spiral Model
  - 并发开发模型 Concurrent Development Model

(1) 迭代化开发

特点： 尽可能降低风险，适用处理不确定的复杂系统。

原则：1、每次迭代产生一个可执行的版本；

2、要求有计划地迭代。



• 快速原型模型

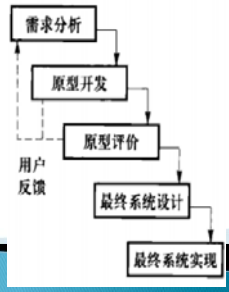
Rapid Prototyping

特点

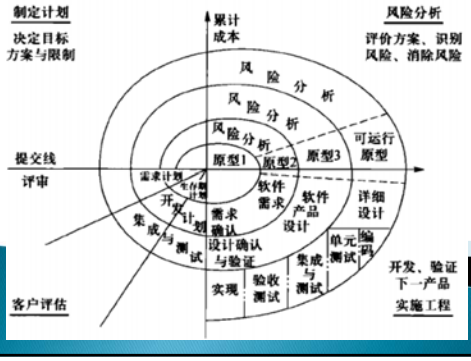
定义出总体目标或初步需求就开发原型，通过原型与用户交互识别进一步的需求.

（1）抛弃式原型

（2）演化式原型



• 螺旋模型



汽车4S店业务管理系统案例

（1）整车销售：对整车销售过程中的客户登记、报价、订购、签订销售合同、交车结算等进行管理。（Sales）

（2）配件销售：对汽车配件销售过程中的报价、领料、销售等进行管理。（Sparepart）

（3）售后服务：对汽修过程中的客户登记、维修项目派工、领料、维修完工、检验、交车结算等进行管理。（Service）

（4）信息反馈：客户投诉登记，并记录处理结果。（Survey）

（5）采购：对整车和配件的采购进行管理，包括采购申请、核准、下订单、进货结算等。

（6）系统管理：对企业员工及其权限、车型及配件等进行管理。

汽车 4S 店业务管理系统的迭代安排

首先分析汽车 4S 店业务管理系统的风险，最大的 3 个风险如下。

（1）第一大风险：需求风险。FD 公司提不出具体需求，同时由于激烈的市场竞争，在开发过程中需求将会发生变化。

（2）第二大风险：技术风险。应采用什么架构，是否要采用 Web service 技术，SJTU 项目组没有相关经验。

（3）第三大风险：进度风险。整个项目的开发进度非常紧。按当前掌握的需求，需要 9 个月才能完成。但 FD 公司为了配合一款新车上市，要求 7 个月后系统就要上线。

根据上述风险分析可以进行如下的迭代安排。

（1）第一次迭代，解决需求风险，开发需求界面原型，并和 FD 公司召开需求联合评审，获得用户反馈。

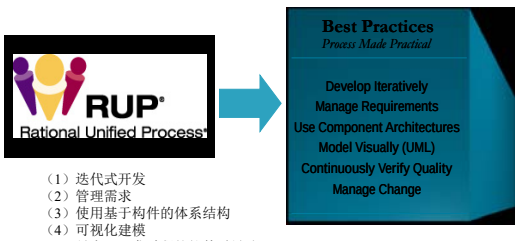
（2）第二次迭代，解决技术风险和架构风险，开发架构原型，并通过测试确保所选的技术和架构是合理的。

（3）第三次迭代～第六次迭代，解决进度风险。将用户需求排出优先级，将优先级高的需求放在第三次迭代完成，构建系统第一个版本，并在第四次迭代进行第一次移交，包括移交时的系统打包、用户手册准备、验收测试和部署上线。在第五次迭代中完成优先级中的需求，构建系统第二个版本，并在第六次迭代进行第二次移交。每次迭代用时一个月。

在新车上线时，第四次迭代已经完成，虽然系统的全部功能没有完成，但系统最重要的功能已经移交和上线，而且质量很好，用户可以开始使用。余下的次要功能在第 9 个月底全部完成，不影响 FD 公司的业务。

(2) RUP统一软件过程

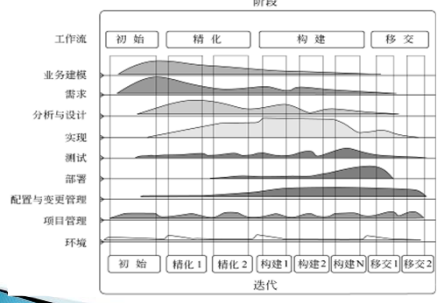
RUP蕴涵了最佳实践准则



（1）迭代式开发
（2）管理需求
（3）使用基于构件的体系结构
（4）可视化建模
（5）贯穿于开发过程的软件质量验证
（6）控制软件变更

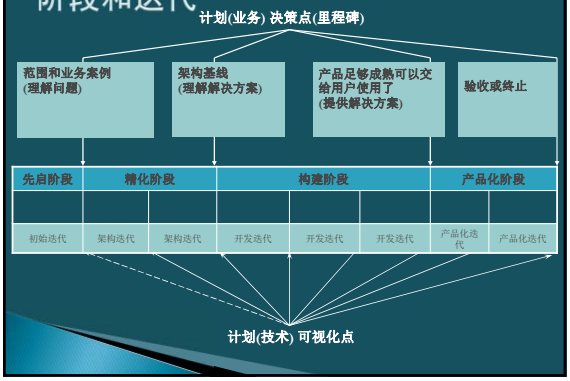
Rational 统一过程

1. RUP (Rational Unified Process) 软件开发生命周期



RUP是一个风险驱动的、基于UML和构件式架构的迭代、递增型开发过程 。

阶段和迭代



Conditions that Increase the Number of Iterations

Inception(初始) 目的:在所有项目干系人之间就项目目标达成共识	Elaboration(精化) 目的:建立架构基线,解决技术风险,为设计与实现奠定基础
▪ Working with new functionality ▪ Unknown business environment ▪ Highly volatile scope ▪ Make-buy decisions	▪ Working with new system environment (new architectural features) ▪ Untested architectural elements ▪ Need for system prototypes
Construction(构建) **目的:完成系统开发**	**Transition(产品化)** **目的:确保最终用户可以使用**
▪ Lots of code to write and verify ▪ New technology or development tools	▪ Need for alphas and betas ▪ Conversions of customer base ▪ Incremental delivery to customers

UP把软件开发生命周期分成了6个核心规范和3个支持规范

核心规范:
- 业务建模(系统目标达成共识)
- 需求(系统范围达成共识)
- 设计
- 实现
- 测试
- 部署

支持规范:
- 配置与变更管理
- 项目管理(风险,计划,进度等)
- 环境

(3) 敏捷过程与极限编程

1.敏捷过程

具有高效、快速响应变化的开发过程。

*1994*年由*Ken Schwaber*和*Jeff Sutherland*提出*Scrum*一词来源于橄榄球运动，意为两队并列争球，*Scrum*过程的核心：

一个体育队加小队长,全体团队负责拿球向前冲

团队成员能够独立地、集中地在创造性的环境下工作

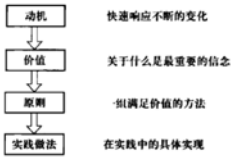


图 2－10 敏捷过程的 4 个层次

动机

快速的市场进入时间、快速变化的需求、快速发展的技术。

价值--敏捷宣言

（1）个体和交互胜过过程和工具；
（2）可以工作的软件胜过面面俱到的文档；
（3）客户合作胜过合同谈判；
（4）响应变化胜过遵循计划。

敏捷过程的原则

- 优先目标是尽早持续交付高价值的软件来满足客户需求；
- 通过驾驭变化帮助客户赢得竞争；
- 经常交付可用软件；
- 业务员和开发人员必须每天一起工作；
- 以积极主动地人为核心建立项目团队；
- 可用软件是最主要的项目进展目标；
- 团队内外最有效的交流是面对面交流；
- 提倡可持续开发，保持稳定的工作步调；
- 用精益求精和优良设计增强敏捷性；
- 简约—工作最小化；
- 最优的架构、需求和设计来自自组织的团队；
- 团队不断开展工作反思，校正自身行为。

--Martin Fowler "New Methodology"

• 实践做法

- **敏捷过程很容易适应变化并迅速做出自我调整，在保证质量的前提下，实现企业效益的最大化。**
- **敏捷过程在保证软件开发有成功产出的前提下，尽量减少开发过程中的活动和制品，Just enough**
- **2001年2月，新方法的一些创始人在美国犹他州成立Agile 联盟（www.agilealliance.org）**

XP　SCRUM　Crystal　ASD　dx

MSF　FDD　DSDM　Lean Development

敏捷过程的适用范围

Martin Fowler认为：新方法不是到处可适用的

适合采用敏捷过程的情况：

- 需求不确定、易挥发
- 有责任感和积极向上的开发人员
- 用户容易沟通并能参与
- 小于10个人的项目团队

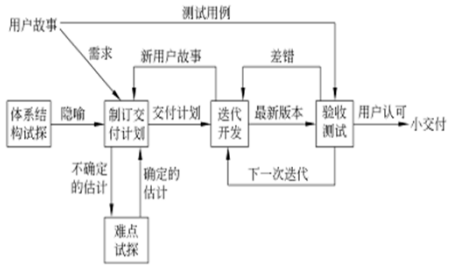
2.极限编程（XP）—敏捷的软件工程实践 (eXtreme Programming)

极限编程是敏捷过程中最著名的一种，指把好的开发实践运用到极致，多应用于软件需求模糊的场合。

- 由Kent Beck、Ward Cunningham、Ron Jeffries等人提出反响最大、最为完善的敏捷过程方法。
- 价值观：
  沟通、反馈、简化、勇气
- 特点：
  测试成为开发的核心；
  纪律性与灵活性巧妙结合.

www.extremeprogramming.org

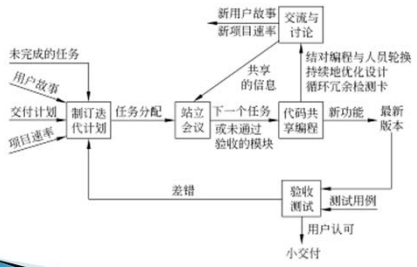
XP项目的整体开发过程

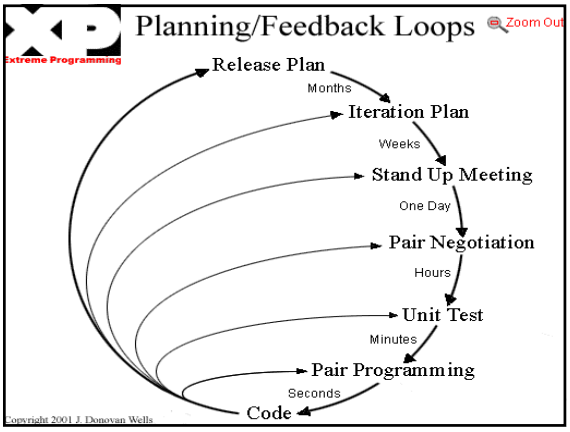


隐喻是将整个系统联系在一起的全局视图；它是系统的未来景像，是它使得所有单独模块的位置和外观（shape）变得明显直观。如果模块的外观与整个系统的隐喻不符，那么你就知道该模块是错误的

- 比如，研发规范中要求对于每个对象要有一个属性类，对于这个对象的业务有一个业务类，对于这个对象的处理接口会有一个接口类，加入这个对象的数据库表名为TEST的话，那么它的属性类就是test，业务类是test_business，接口类就是test_instance。于是整个系统有多少数据表，基本上就会有多少套类，从目录结构上很容易看出来实现的进度和类之间的关系。
- 这样的规范在大家接受之后，所有人都会按照这样的思路去做，假如在对代码进行重构的时候发现需要重新生成一个接口类，那么也会按照类似test_instance_xxxx这样的方式给类起名。

XP迭代开发过程





XP关键做法

- 现场客户（On-site Customer）
- 计划博弈（Planning Game）
- 系统隐喻（System Metaphor）
- 简化设计（Simple Design）
- 集体拥有代码（Collective Code Ownership）
- 结对编程（Pair Programming）
- 测试驱动（Test-driven）
- 小型发布（Small Releases）
- 重构（Refactoring）
- 持续集成（Continuous integration）
- 每周40小时工作制（40-hour Weeks）
- 代码规范（Coding Standards）

RUP与XP的共性

- 基础都是面向对象方法（取代传统的结构化方法）
- 都重视代码、文档的最小化和设计的简化
- 采用动态适应变化的演进式迭代周期（取代传统的瀑布型生命周期）
- 需求和测试驱动
- 鼓励用户积极参与

RUP与XP的区别

- XP以代码为中心，编码和设计活动融为一体，弱化了架构的概念。
- RUP过程通常以架构为中心，细化阶段的主要目的就是构造出一个可运行的架构原型，作为将来添加需求功能的稳固基础。
- XP不包含业务建模、部署、过程管理等概念。
- RUP适合各种规模的项目，XP只适用于小团队。

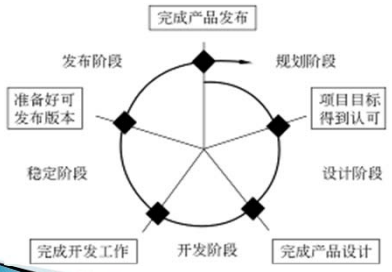
(4) 微软过程MSF(Microsoft Solutions Framework)

1.微软过程准则

- 项目计划应该兼顾未来的不确定因素;
- 用有效的风险管理来减少不确定因素的影响;
- 经常生成并快速的地测试软件的过渡版本，提高稳定性和可预测性;
- 采用快速循环，递进的开发过程;
- 用创造性的工作来平衡产品特性和产品成本;
- 项目进度表应该具有较高稳定性和权威性;
- 使用小型项目组并发的完成开发工作;
- 在项目早期把软件配置项基线化，项目后期则冻结产品;
- 使用原型验证概念，对项目进行早期论证;
- 把零缺陷作为追求的目标;
- 里程碑评审会的目的是改进工作，切忌相互指责.

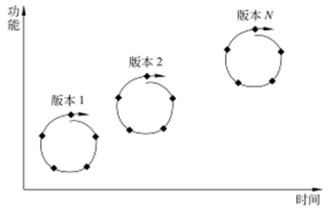
2、微软软件生命周期

阶段划分和主要里程碑



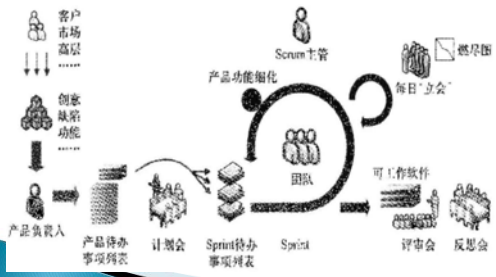
3.微软过程模型

微软过程的生命周期模型



(5) Scrum过程

- 强调经验性过程而不是确定性过程
- 演化型的迭代开发过程



2.3 软件过程的选择与裁剪

每种过程都有其价值，分别具有一些最佳实践，适合于某类软件的开发。

软件过程的选择

（1）产品/项目自身的特点

（2）团队的实际情况和企业文化

（3）客户的影响

软件过程进行裁剪

（1）流程归并与裁剪

（2）角色的筛选与定制

（3）工件的裁剪和定制

2.4 软件过程的评估与改进

参考模型

（1）CMM/CMMI

过程能力成熟度模型

（Capability Maturity Model，CMM）

CMMI 是一个标准簇

（Capability Maturity Model Integration，CMMI）

- CMMI for Development（CMMI－DEV）：开发模型
- CMMI for Service（CMMI－SVC）：服务模型
- CMMI for Acquisition（CMMI－ACQ）：采购模型

CMMI 模型不同的改进方法：

组织成熟度方法（阶梯式模型）

过程能力方法（连续式模型）

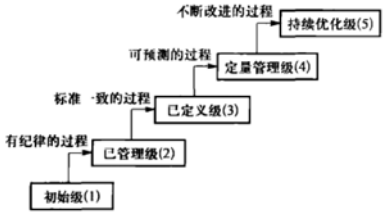


图 2－15　CMMI 阶梯式模型

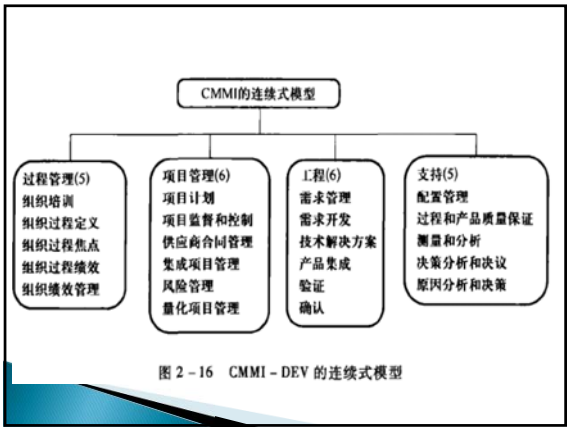


图 2－16　CMMI－DEV 的连续式模型

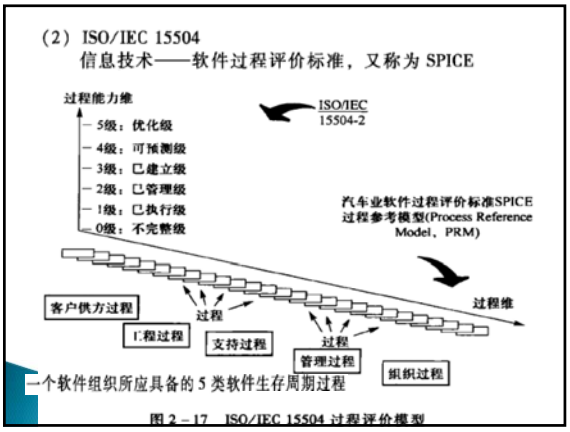


图 2－17　ISO/IEC 15504 过程评价模型

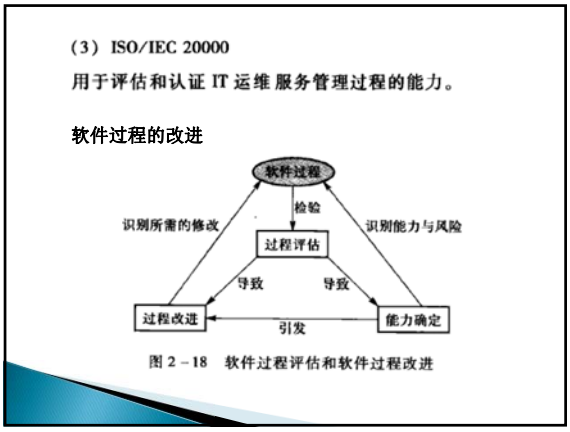


图 2－18　软件过程评估和软件过程改进