软件质量概述

参考：

- 《Software Engineering - A Practitioner's Approach》（软件工程——实践者的研究方法）
- ISO/IEC 9000:2008
- 《敏捷开发知识体系》 (ISBN：978730231573 5)
- 《Microsoft Solutions Framework Essentials 》(ISBN 9780735623538)。 Microsoft Press。
- 《软件工程教程：IBM RUP方法实践》(ISBN：9787111406815)

软件质量概述

- 1.质量和软件的概念
- 2.软件生命周期

质量和软件的概念

- 1.1 什么叫质量
- 1.2 影响质量的因素
- 1.3 质量目标
- 1.4 质量成本
- 1.5 质量管理
- 1.6 软件和软件产品

质量初步认识

- 1.1 什么叫质量(quality)
  - 《辞海》和《辞源》中，把质量解释为“产品或工作的优劣程度”。
  - ISO9001:2008把质量定义为：一组固有特性满足要求的程度。
    - 术语“质量”可使用形容词如差、好或优秀来修饰。
    - “固有的”就是指存在于某事或某物中的，尤其是那种永久的特性。

质量初步认识

- 特性：可区分的特征。
  - 注1：特性可以是固有的或赋予的。
    - 固有特性就是指某事或某物中本来就有的，尤其是那种永久的特性，如：螺栓的直径、机器的生产率或接通电话的时间等技术特性。
    - 赋予特性不是固有的，不是某事物本来就有的，而是完成产品后因不同的要求而对产品所增加的特性，如产品的价格、硬件产品的供货时间和运输要求（如：运输方式）、售后服务要求（如：保修时间）等特性。
    - 固有与赋予特性的相对性：不同产品的固有特性和赋予特性不同，某种产品赋予特性可能是另一种产品的固有特性（转换）。

质量初步认识

- 特性：可区分的特征。
  - 注2：特性可以是定性的或定量的。
  - 注3：有各种类别的特性，如：

    --物理的(如：机械的、电的、化学的或生物学的特性)；

    --感官的(如：嗅觉、触觉、味觉、视觉、听觉)；

    --行为的(如：礼貌、诚实、正直)；

    --时间的(如：准时性、可靠性、可用性)；

    --人体工效的(如：生理的特性或有关人身安全的特性)；

    --功能的(如：飞机的 最高速度)。
- 要求：明示的、通常隐含的或必须履行的需求或期望

质量初步认识

- 质量具有经济性、广义性、时效性、相对性

- 人们是如何认识质量？
  - 狭义的质量概念就是产品质量。
  - 广义的质量概念包括产品质量和工作质量两个组成部分，即全面质量。

质量初步认识

- 对产品的解释

  过程的结果。(ISO9000:2008)

  - 产品可分为4种类别，即硬件、流程性材料、软件、服务或它们的组合。（ISO 8402）
    - 硬件是不连续的具有特定形状的产品，如空调、洗衣机、电视机等；
    - 流程性材料是将原料转化为某一预定状态的有形产品，如流体、气体、粒状、块状、线状或板状，其典型的交付方式有桶装、袋装、罐装、瓶装或通过管道等；

质量初步认识

- 软件是通过支持媒体表达的信息所构成的一种智力创作，软件的形式如概念、信息、程序、规划、记录、计算机程序等；
- 服务是为满足客户的需要，供方和顾客之间在接触时的活动，以及供方内部活动所产生的结果。

- 产品可以是有形的(如流程性材料)，也可以是无形的(如知识或概念)或是它们的组合。
- 产品可以是预期的(如提供给顾客)或非预期的(如污染或不愿有的后果)。

质量初步认识

- 1.2 影响质量的因素

  包括5大因素：人、机(设备)、物（材料）、方法、环境。

- 1.3 质量目标

  在质量方面所追求的目的。

  质量目标是产品和工程质量在一定时间内可达到的水平。

- 1.4 质量成本

  将产品质量保持在规定的质量水平上所需的有关费用。

  质量成本由两部分构成，即运行质量成本和外部质量保证成本。

（continue）

质量初步认识

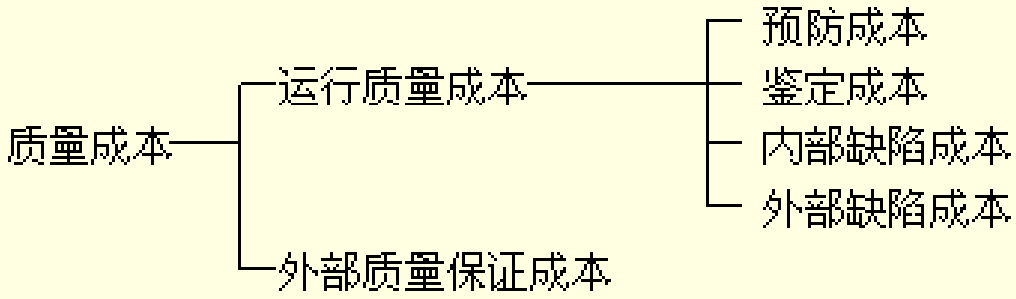


图 1　　质量成本的构成

质量初步认识

1.4 质量成本(续)

- 运行质量成本

  指企业为保证和提高产品质量而支付的一切费用以及因质量故障所造成的损失费用之和。它又分为四类，即企业内部损失成本、鉴定成本、预防成本和外部损失成本等。

- 外部质量保证成本

  指为用户提供所要求的客观证据所支付的费用。主要包括：

  1）为提供特殊附加的质量保证措施、程序、数据所支付的费用。

  2）产品的验证试验和评定的费用。

  3）满足用户要求，进行质量体系认证所发生的费用。

质量初步认识

- 1.5 质量管理

  不仅从技术层面上去思考产品质量，也从质量管理的角度去思考。

质量和软件的概念

- 1.6 软件和软件产品
  - 1.6.1 软件的含义
  - 1.6.2 软件产品的组成
  - 1.6.3 软件产品和其他产品的差异

什么是软件

- 1.6.1 软件的含义

  软件是（摘自《软件工程——实践者的研究方法》）

  (1)能够完成预定功能和性能的可执行的指令(计算机程序)；

  (2)使得程序能够适当地操作信息的数据结构；

  (3)描述程序的操作和使用的文档。

  软件 = 程序(数据) + 文档 + 服务

软件产品的组成

- 1.6.2 软件产品组成部分

（1）程序代码　（2）帮助文件　（3）用户手册

（4）样本和示例　（5）标签　（6）产品支持信息

（7）图表和标志　（8）错误信息　（9）广告与宣传材料

（10）软件的安装　（11）软件说明文件

（12）测试错误提示信息

软件产品和其他产品的差异

- 1.6.3 软件产品和其他产品的不同
  - 软件是逻辑产品而不是实物产品。
  - 软件的功能只能依赖于硬件和运行环境，以及人们对它的操作，才能得以体现。
  - 对软件产品的要求比一般有形产品要复杂。

软件产品和其他产品的差异

- 1.6.3 软件产品和其他产品的不同（续）
  - 软件设计时发生的复杂性（引起软件的4个特征）：
    - 功能的多样性；
    - 实现的多样性 ；
    - 能见度低；
    - 软件结构的合理性差
  - 软件是智力密集型产品
    - 知识产权保护极为重要。

软件质量概述

- 1.质量和软件的概念
- 2.软件生命周期

软件生命周期

- 2.1 软件开发项目组
- 2.2 软件生命周期质量管理
- 2.3 软件开发模型
- 2.4 软件开发与软件测试的关系
- 2.5 软件神话

2.1 软件开发项目组

- 项目管理经理：全程负责整个软件项目的开发。
- 系统设计师：设计整个系统构架或软件构思。
- 程序员：负责设计、编写程序，并修改软件中的缺陷。
- 软件测试员/测试师或质量保证员（QA）：负责找出并报告软件产品的问题，与开发组密切合作，进行测试并报告发现的问题。
- 技术制作、用户助手、用户培训员、手册编写和文件档案专员：负责编写软件产品附带的文件和联机文档 。
- 结构管理和制作人员：负责将程序员编写的全部文档资料合并成一个软件包 。

2.2 软件生命周期质量管理

- 质量形成于过程

  产品质量是生产出来的，不是检验出来的。（威廉·戴明）

  上医医未病之病,中医医欲病之病,下医医已病之病

2.2 软件生命周期质量管理

- 2.2.1 概述

要建造一个软件，相关工作可分为三个一般阶段。每一阶段，又分几个小阶段（共6个阶段，传统生命周期，即瀑布模型）

- 定义阶段

（1）计划（Planning）

（2）需求分析（Requirement Analysis）

- 开发阶段

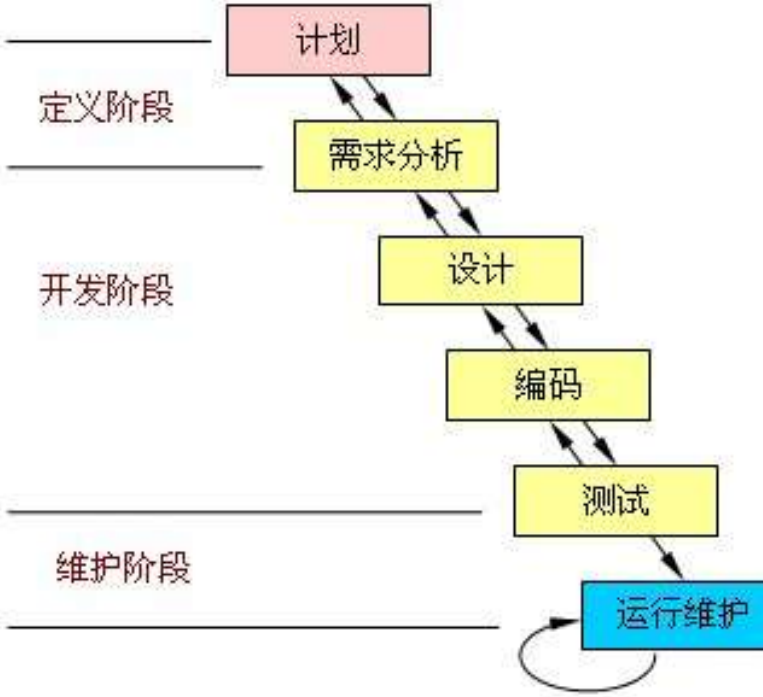
（3）设计（Design）

（4）编码（Coding）

（5）测试（Testing）

- 维护阶段

（6）运行与维护（Run and Maintenance）



2.2 软件生命周期质量管理

2.2.2 需求分析

了解、分析客户需求，确定软件产品所能达到的目标。

- 主要工作
  - 调研用户需求；
  - 论证项目可行性；
  - 确定系统运行环境；
  - 建立系统逻辑模式；
  - 确定系统功能及性能要求；
  - 编写需求规格说明、用户手册概要、测试计划；
  - 确认项目开发计划

需求分析阶段

- 应完成的文档
  - 可行性报告
  - 项目初步开发计划
  - 需求规格说明
  - 用户手册概要
  - 测试计划
- 其他（软件生命周期每个阶段必有）
  - 配置管理
  - 评审

设计阶段

2.2.3 设计

- 根据需求分析的结果，考虑如何在逻辑、程序上去实现所定义的产品功能、特性等。
- 可分为概要设计和详细设计；也可以分为数据结构设计、软件体系结构设计、应用接口设计、模块设计、算法设计、界面设计等。
- 设计过程将需求转换成软件表示，设计的结果将作为编码的框架和依据。

设计阶段

- 概要设计主要工作
  - 建立系统总体结构，划分功能模块；
  - 定义各功能模块接口；
  - 数据库设计（如需要）
  - 指定组装测试计划
- 详细设计主要工作
  - 设计各模块具体实现算法；
  - 确定模块间的详细接口
  - 指定模块测试方案

设计阶段

- 概要设计完成的文档
  - 概要设计说明书；
  - 数据库设计说明书（如有）；
  - 组装测试计划
- 详细设计完成的文档
  - 详细设计说明书；
  - 模块测试计划

编程阶段

2.2.4 编程

需求分析、设计之后，用一种或多种具体的编程工具进行编码，即将设计转换成计算机可读的形式。

- 主要工作
  - 编程；
  - 进行模块调试和测试
  - 编写用户手册
- 完成文档
  - 程序调试报告
  - 用户手册

测试阶段

2.2.5. 测试

任何编程，免不了存在这样或那样的错误，所以有必要进行软件测试。

测试过程集中于软件的内部逻辑，以及外部功能。

测试按不同的过程阶段分为单元测试、集成测试、功能测试、系统测试、验证测试等。

维护阶段

2.2.6 维护

- 软件交付之后不可避免地要进行修改、升级等。

  从理论上，软件测试不可能挑出所有错，所以软件在交付给用户之后有可能存在某些问题；

  用户的需求会发生变化，特别是开始使用产品之后，对计算机系统有了真正的认识和了解，会提出适用性更好的、功能增强的要求。

- 维护阶段重复定义和开发阶段的步骤，但却是在已有软件的基础上发生的。在维护阶段可能遇到四类修改要完成：

  1）纠错　2）适应　3）增强　4）预防

2.3 软件开发模型

- 2.3 软件开发模型

  软件开发模型是指软件开发全部过程、活动和任务的结构框架。

  - 2.3.1 瀑布模式、大棒模式、边写边改模式
  - 2.3.2 原型模式、快速应用(RAD)模式、螺旋模式、增量模式和跌代模式、混合模型
  - 2.3.3 敏捷开发
  - 2.3.4 MSF
  - 2.3.5 RUP

  UML代表着软件建模的发展趋势

2.3.1 瀑布模式、大棒模式、边写边改模式

- 瀑布模型
  - 优点

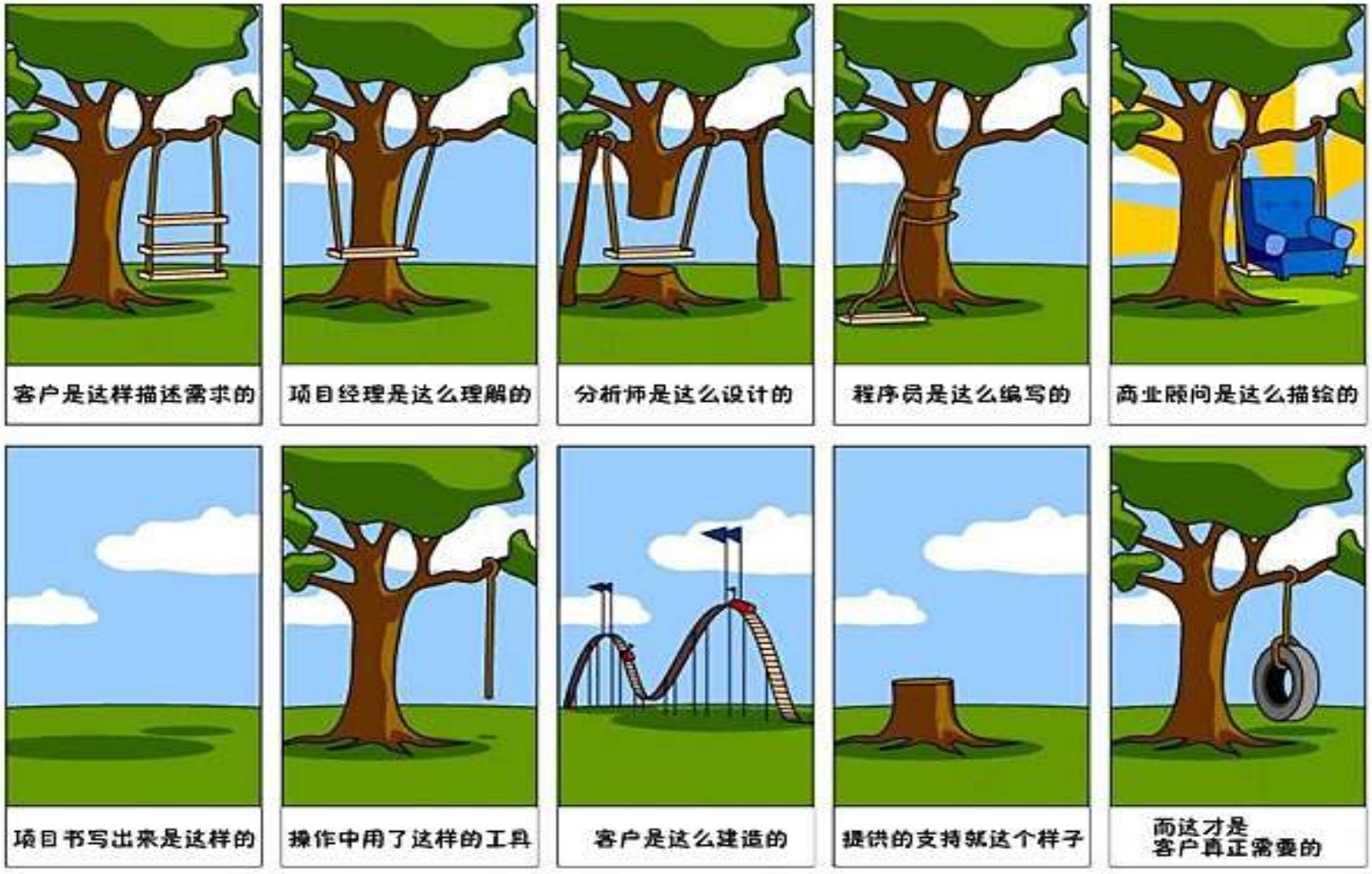
瀑布模型的优点是可以保证整个软件产品较高的质量, 可以保证系统在整体上的充分把握,使系统具备良好的扩展性和可维护性.

- 易于理解;
- 调研开发的阶段性;
- 强调早期计划及需求调查;
- 确定了何时产生可交付的产品及何时进行评审和审查;
- 强调产品测试。

2.3.1 瀑布模式、大棒模式、边写边改模式

- 缺点
  - 依赖早期进行的需求调查，不能适应需求变化；
  - 流程单一，开发中的经验得不到反馈和应用于本产品的开发中；
  - 没有反映出开发的迭代本质；
  - 不包含风险评估
  - 风险往往在后期才显露，失去及早纠正机会。

需求，设计阶段的问题


客户是这样描述需求的
项目经理是这么理解的
分析师是这么设计的
程序员是这么编写的
商业顾问是这么描绘的
项目书写出来是这样的
操作中用了这样的工具
客户是这么建造的
提供的支持就这个样子
而这才是
客户真正需要的

开发，维护阶段的问题

2.3.1 瀑布模式、大棒模式、边写边改模式

- 大棒模式

在这种模型中，既没有规格说明，也没有经过设计，软件随着客户的需要一次又一次地不断被修改。

- 优点是简单。几乎无计划。通常可能是开发者的“突发奇想”。项目成员精力都花在开发软件和编写代码上。
- 缺点：开发过程非工程化，随意性大。最终的软件产品是什么样不可知。
- 关于测试：有的较简单，有的则非常困难。

2.3.1 瀑布模式、大棒模式、边写边改模式

- 边写边改模式
  - 在大棒模式的基础上考虑了产品的要求。

    项目成员通常只有粗略的想法就进行简单的设计，然后开始漫长的编码、测试、修复这样一个循环的过程。

    在认为无法更精细的描述软件产品要求时，就发布产品。
  - 优点：能够较为迅速的展现成果，适合需要快速制作而且用完就扔的小项目，如示范程序、演示程序等。
  - 缺点：其编码和测试可能将是长期的循环往复的过程。

边写边改模式（续）

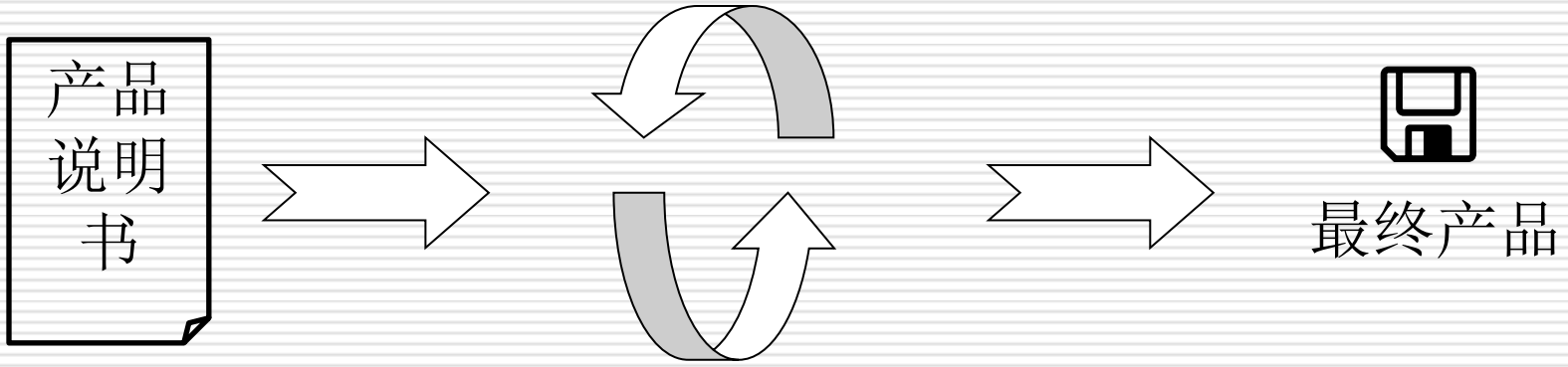


图 边写边改开发模式

大棒模式或边写边改模式

- 探索测试

  如采用大棒模式或者边写边改模式，就不会有作为测试依据的各类文档。尽管这对于软件测试员不是理想的状况，但是此时可以采用称为探索测试的解决方案。

  这需要把软件当产品说明书来对待。分步骤地逐项探索软件特性。记录软件执行情况，详细描述功能。在这种情况下，无法像有产品说明书那样完整测试软件--比如无法判定是否遗漏功能，但是可以进行系统测试。找到软件缺陷。

  另外，与同类型软件进行比较也是一个有效的方法。

2.3.2 原型模式、快速应用(RAD)模式、螺旋模式、增量模式和跌代模式、混合模型

- 原型模型

在基本需求分析之后，快速开发出产品原型，然后基于这个原型，同客户沟通、交流，更好地了解客户需求，不断修改这个原型，到双方认可的程度，再做详细分析、设计和编程，最终开发出令客户满意的产品。

- 一般步骤如下：

(1) 先定义软件的总体目标，根据已知的需求规划出可实现的区域。

(2) 然后是“快速设计”，集中于系统的总体框架、基本功能和直观的输入方式和输出格式等。

(3) 有了原型，使客户对系统实现哪些具体功能、功能实现到什么程度有更好的理解。开发者边开发边评估，不断细化软件的需求，逐步调整原型使其满足客户的要求。这形成一个迭代的过程。

软件开发模式 - 原型模型

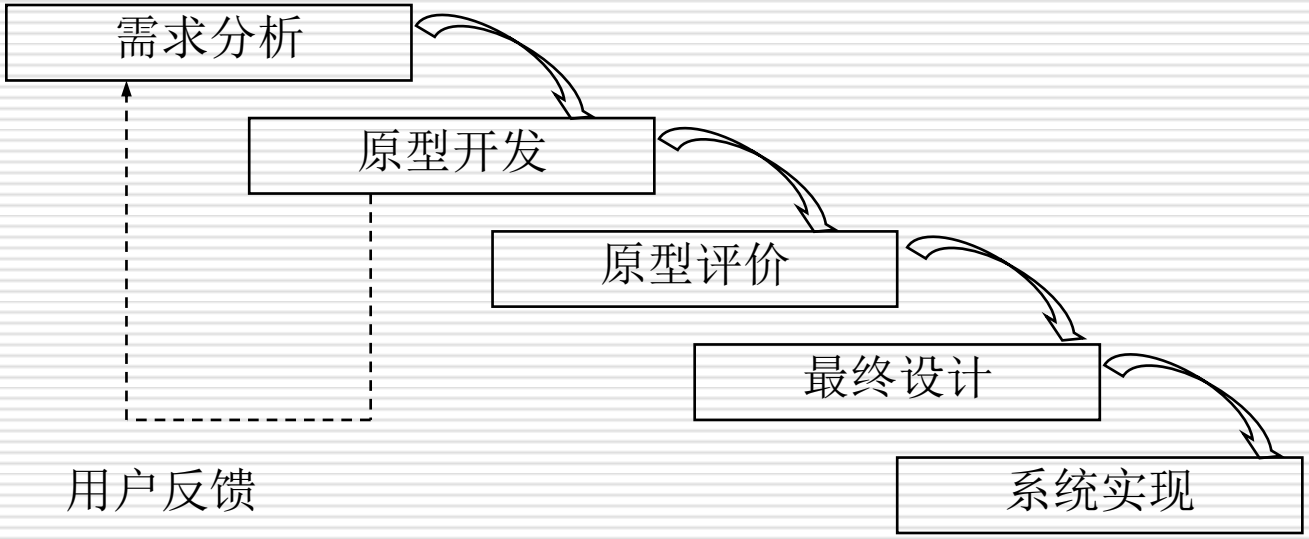


图　原型开发模式

原型模型

- 特点
  - 可克服瀑布模型的一些缺点，使用户能够感受到实际的系统，减少由于软件需求不明确带来的开发风险，具有显著的效果。
  - 即使开始建立的原型过于简单或性能很差，难以使用，但为下一次建立适用的模型积累了经验，而浪费的成本、时间有限。
  - 原型系统的内部结构并不重要，重要的是必须迅速建立原型，随之迅速修改原型，以反映客户的需求。
  - 产品的先天性不足，因为开发者常常需要做实现上的折中，可能采用不合适的操作系统或程序设计语言，以使原型能够尽快工作。

2.3.2 原型模式、快速应用(RAD)模式、螺旋模式、增量模式和跌代模式、混合模型

- RAD模型

  RAD（rap application development）模型，即快速应用开发模型。它通过使用基于构件的开发方法来缩短产品开发的周期，提高开发的速度。RAD模型实现的前提是能做好需求分析，并且项目范围明确，这一点正好和原型模型相反。

2.3.2 原型模式、快速应用(RAD)模式、螺旋模式、增量模式和跌代模式、混合模型

- 增量模型和迭代模型。
  - 增量模型　描述软件产品的不同阶段是按产品所具有的功能进行划分，先开发主要功能或用户最需要的功能，然后，随着时间推进，不断增加新的辅助功能或次要功能，最终开发出一个强大的、功能完善的、高质量的、稳定的产品。
  - 迭代模型　描述软件产品的不同阶段是按产品深度或细化的程度来划分。先将产品的整个框架都建立起来，在系统的初期，已经具有用户所需求的全部功能。然后，随着时间推进，不断细化已有的功能或完善已有功能，这个过程好像是一个迭代的过程。最终的目标是一致的，也是为了实现一个强大的、功能完善的、高质量的、稳定的产品。

演化模型(evolutionary model)

- 要针对事先不能完整定义需求的软件开发。
- 用户可给出待开发系统的核心需求，当看到核心需求实现后，提出反馈，以支持系统的最终设计和实现。
- 实际上，这个模型可看作是重复执行的多个“瀑布模型”。
- “演化模型”要求开发人员有能力根据功能的重要性及对总体设计的基础结构的影响而作出判断，把项目的产品需求分解为不同组，以便分批循环开发。

螺旋模型

- 螺旋模型是在瀑布模型和演化模型的基础上，加入两者所忽略的风险分析所建立的一种软件开发模型。
- 主要思想是在开始时不必详细定义所有细节，而是从小开始，定义重要功能，尽量实现，接受客户反馈，进入下一阶段，并重复上述过程，直到获得最终产品。
- 每一螺旋（开发阶段）包括5个步骤：①确定目标，选择方案和限制条件。 ②对方案风险进行评估，并能解决风险。 ③进行本阶段的开发和测试。 ④计划下一阶段。 ⑤确定进入下阶段的方法。
- 优点：严格的全过程风险管理；强调各开发阶段的质量；提供机会评估项目是否有价值继续下去。
- 缺点：可能难以使用户（尤其在有合同约束的情况下）相信演化方法是可控的；项目的成功依赖于风险评估专门技术，如果一个大的风险未被发现和管理，就会出现问题。

过程开发模式

- 过程开发模式：

  它又叫混合模式或元模式，是指把几种不同模式组合成一种混合模式，它允许一个项目能沿着最有效的路径发展。

  因为上述的模式中都有自己独特的思想，现在的软件开发团队中很少说标准的采用那一种模式的，因为模式和实际应用还是有很大的区别的。实际上，许多软件开发团队都是在使用几种不同的开发方法组成他们自己的混合模式。

2.3.3 敏捷开发

- 2001年二月敏捷开发宣言后形成敏捷联盟

  一组由17位在DSDM（动态系统开发方法），XP（极限编程），Scrum（混乱方法），FSD（功能设计文档），Feature Driver Development（特征驱动开发），Cristal Clear（水晶开发）等领域的专家组成的代表团齐聚美国犹他州，寻找这些方法的共同点。最终，这些专家制定并宣布了敏捷开发宣言。

  由此形成了现在我们所认识的敏捷开发和后来的敏捷联盟。

http://www.softed.com/agileacademy/

敏捷开发

- 敏捷宣言
  - 个体和交互　　胜过　　过程和工具
  - 可以工作的软件　　胜过　　面面俱到的文档
  - 客户合作　　胜过　　合同谈判
  - 响应变化　　胜过　　遵循计划

虽然右边的也有价值，但左边的具有更大的价值

敏捷开发

- 敏捷方法论采用迭代/增量开发的过程模型
  - 是一种以人为核心、迭代、循序渐进的开发方法。
  - 组织上，软件项目的构建被切分成多个子项目，各个子项目的成果都经过测试，具备集成和可运行的特征。
  - 时间上，相对于传统的瀑布式开发，迭代开发把软件生命周期分成很多个小周期（一般不大于2个月，建议2周），每一次迭代都可以生成一个可运行、可验证的版本，并确保软件不断的增加新的价值。
- 敏捷开发由几种轻量级的软件开发方法组成。

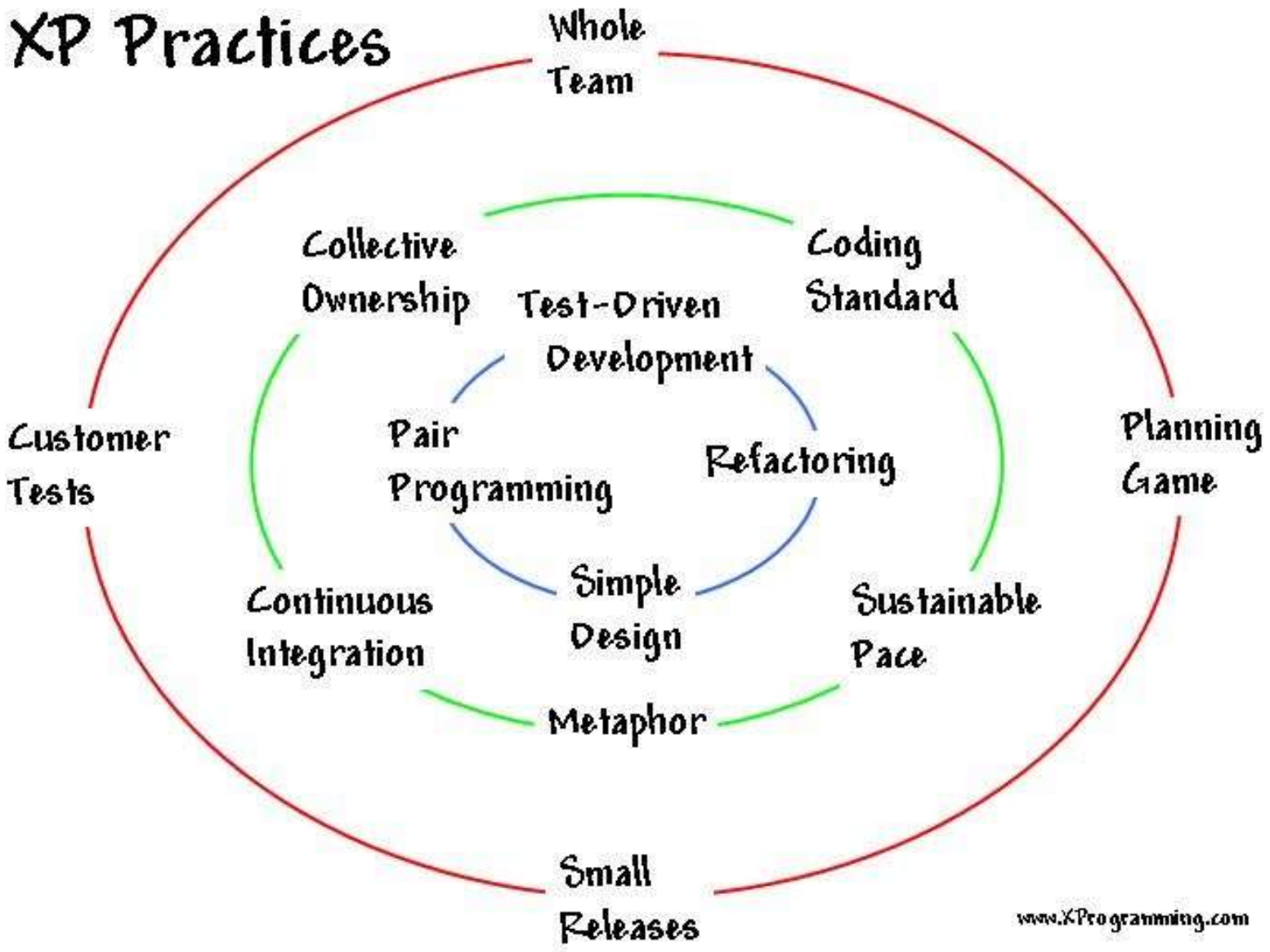
  它们包括：极限编程（XP），Scrum，精益开发（Lean Development）等等

敏捷开发介绍-极限编程XP

- 主要目的是降低需求变化的成本
- 定义了一套简单的开发流程

  包括：编写用户案例，架构规范，实施规划，迭代计划，代码开发，单元测试，验收测试等等
- 提倡互动交流、反馈、简单、勇气、团队
- 12个最佳实践

  核心做法：小规模，频繁的版本发布，短迭代周期，风险评估。


XP Practices
Whole Team
Collective Ownership
Coding Standard
Test-Driven Development
Customer Tests
Pair Programming
Refactoring
Planning Game
Simple Design
Continuous Integration
Sustainable Pace
Metaphor
Small Releases
www.XProgramming.com

敏捷开发介绍-精益

- 精益开发起源

  从丰田公司的产品开发方法中演化而来。它主要包括两个部分：一部分是核心思想及原则，另外一部分由一些相应的工具构成。

- 核心思想

  查明和消除浪费。在软件开发过程中，错误（bugs），没用的功能，等待以及其他任何对实现结果没有益处的东西都是浪费。浪费及其源头必须被分析查明，然后设法消除。

- 精益开发的原则包括:

  强调学习。不断改进所开发的产品和开发效率。

  （continue）

敏捷开发介绍-精益

- 精益开发的原则包括:

在最后时刻做决定。避免在可能改变的事情上做无谓的努力，避免浪费。

用最快的速度交付用户。缩短迭代周期加速开发及交付，加快交流，提高生产力

给团队自主权。激励团队并让团队成员自我管理-敏捷方法成功的基本因素之一。

诚信。确保系统正常工作，客户需求是团队努力坚持的诚信和对用户的承诺。

全局观。精益开发强调整体优化的系统。无论开发的组织还是被开发的产品，从整体上考虑优化比从各个局部去优化更高效。

- 精益软件更重要的是不断完善开发过程的一种思维方式。

敏捷开发介绍-scrum

- SCRUM是一个敏捷开发框架

  它由一个开发过程，几种角色以及一套规范的实施方法组成。它可以被运用于软件开发，项目维护，也可以被用来作为一种管理敏捷项目的框架。

- Scrum定义了4种主要的角色：

  1、产品拥有者（Product Owner）：负责产品的远景规划，平衡所有利益相关者（stakeholder）的利益，确定不同的产品需求积压的优先级等。它是开发团队和客户或最终用户之间的联络点。

  2、利益相关者（Stakeholder）：该角色与产品之间有直接或间接的利益关系，通常是客户或最终用户代表。他们负责收集编写产品需求，审查项目成果等。

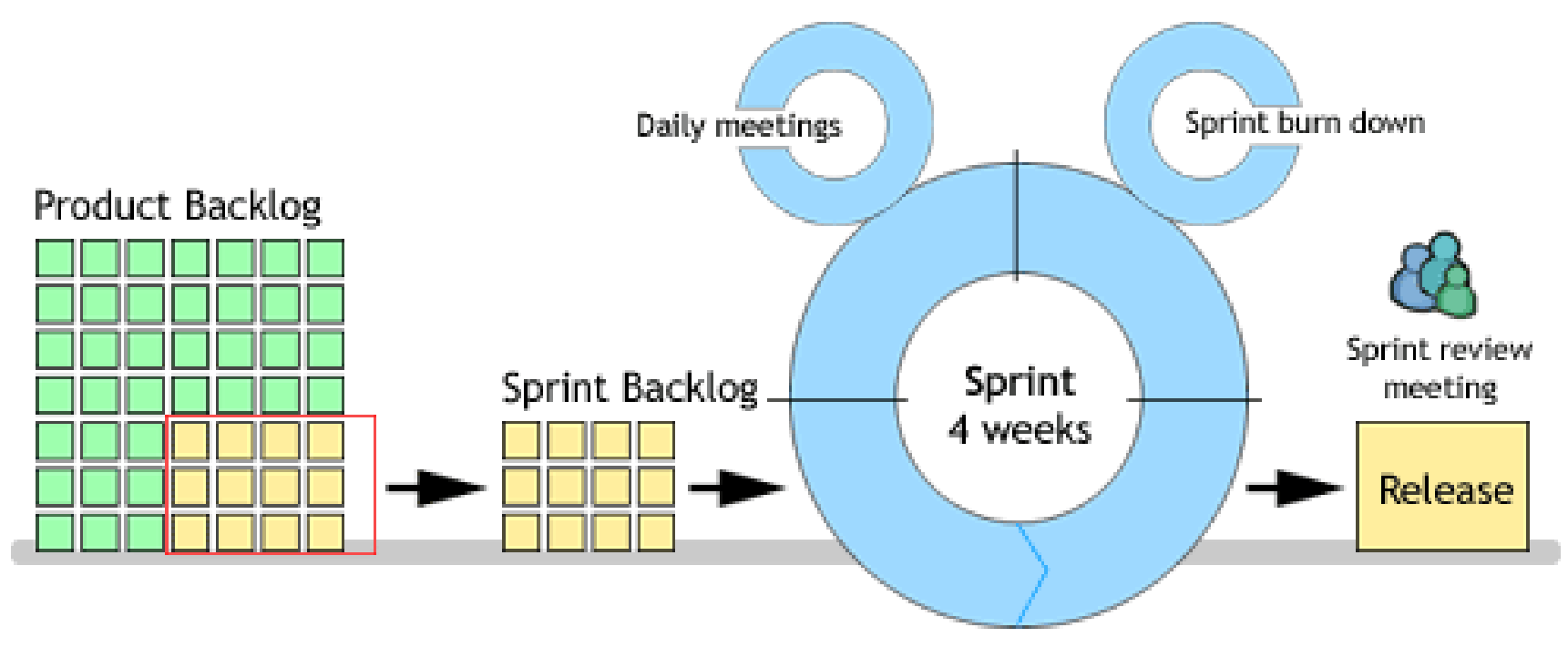
敏捷开发介绍-scrum

- Scrum定义了4种主要的角色：

  3、Scrum专家（Scrum Master）：Scrum专家负责指导开发团队进行Scrum开发与实践。它也是开发团队与产品拥有者之间交流的联络点。

  4、团队成员（Team Member）：即项目开发人员。

Scrum开发模型


Daily meetings
Sprint burn down
Product Backlog
Sprint Backlog
Sprint
4 weeks
Sprint review
meeting
Release

敏捷开发-SCRUM名词解释

- backlog：可以预知的所有任务，包括功能性的和非功能性的所有任务。
- sprint：一次跌代开发的时间周期，一般最多以30天为一个周期。在这段时间内，开发团队需要完成一个制定的backlog，并且最终成果是一个增量的，可以交付的产品。
- sprint backlog：一个sprint周期内要完成的任务。
- scrumMaster：负责监督整个Scrum进程，修订计划的一个团队成员。
- time-box：一个用于开会的时间段。比如每个daily scrum meeting的time-box为15分钟。

敏捷开发-SCRUM名词解释

- sprint planning meeting：

  在启动每个sprint前召开。一般为一天时间（8小时）。

  会议任务：产品Owner和团队成员将backlog分解成小的功能模块，决定在即将进行的sprint里需要完成多少小功能模块，确定好这个Product Backlog的任务优先级。另外，该会议还需详细地讨论如何能够按照需求完成这些小功能模块。制定的这些模块的工作量以小时计算。

- Daily Scrum meeting：

  开发团队成员召开，一般为15分钟。每个开发成员需要向ScrumMaster汇报三个项目：今天完成了什么？遇到了障碍无法继续下去？明天要做什么？通过该会议，团队成员可以相互了解项目进度。

敏捷开发-SCRUM名词解释

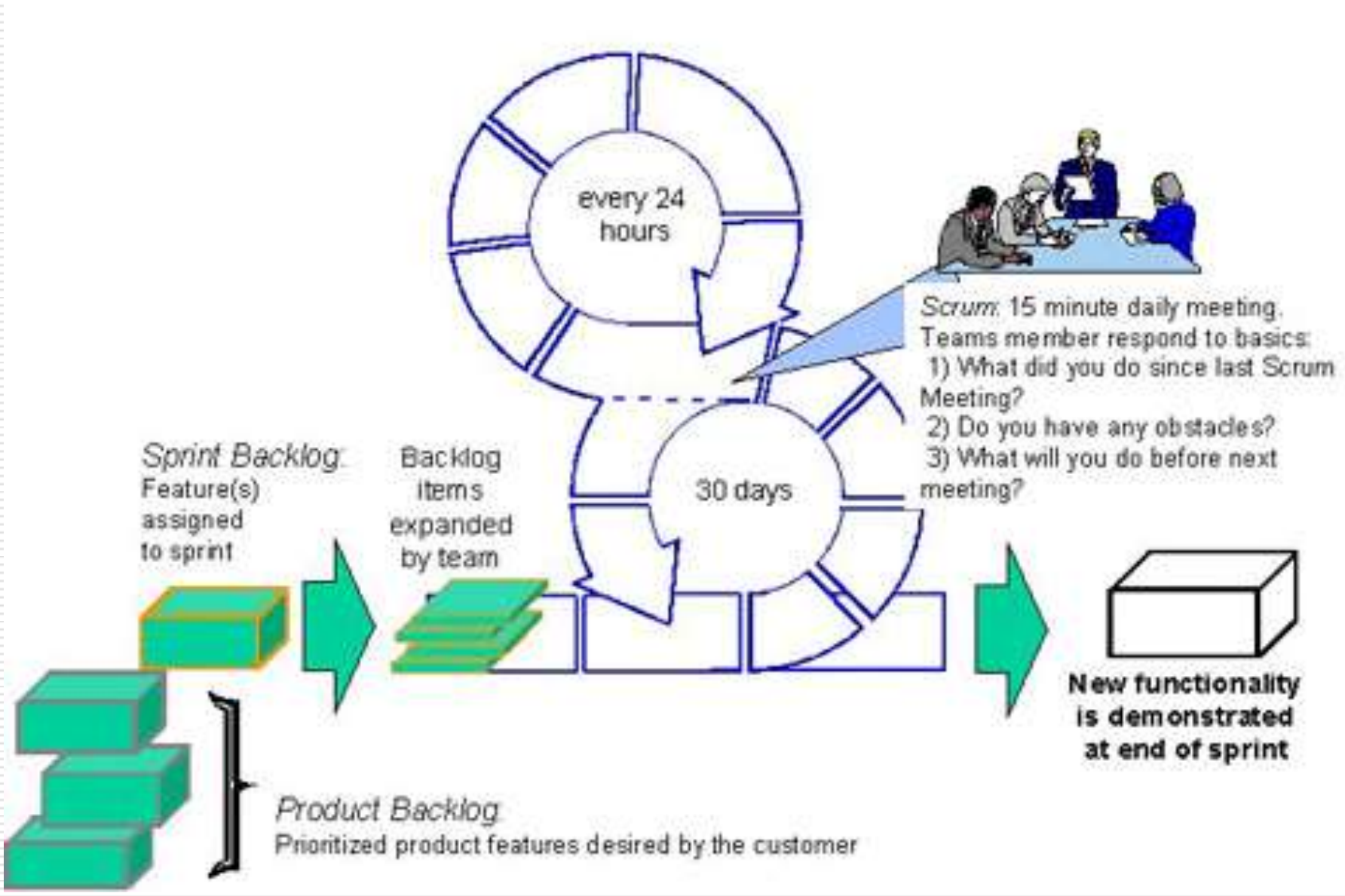
- Sprint review meeting：

  在每个Sprint结束后，这个Team将这个Sprint的工作成果演示给Product Owner和其他相关的人员。一般该会议为4小时。

- Sprint retrospective meeting：

  对刚结束的Sprint进行总结。会议的参与人员为团队开发的内部人员。一般该会议为3小时。

敏捷开发-实施Scrum的过程介绍



敏捷开发-实施Scrum的过程介绍

- 确定Sprint Backlog

  将整个产品的backlog分解成Sprint Backlog，这个Sprint Backlog是按照目前的人力物力条件可以完成的。

- 召开sprint planning meeting

  划分，确定这个Sprint内需要完成的任务，标注任务的优先级并分配给每个成员。注意这里的任务是以小时计算的，并不是按人天计算。

- sprint开发周期

  进入sprint开发周期，在这个周期内，每天需要召开Daily Scrum meeting。

敏捷开发-实施Scrum的过程介绍

- 成果演示

  整个sprint周期结束，召开Sprint review meeting，将成果演示给Product Owner。

- 回顾

  团队成员最后召开Sprint retrospective meeting，总结问题和经验。

- 下一次Sprint。

敏捷开发

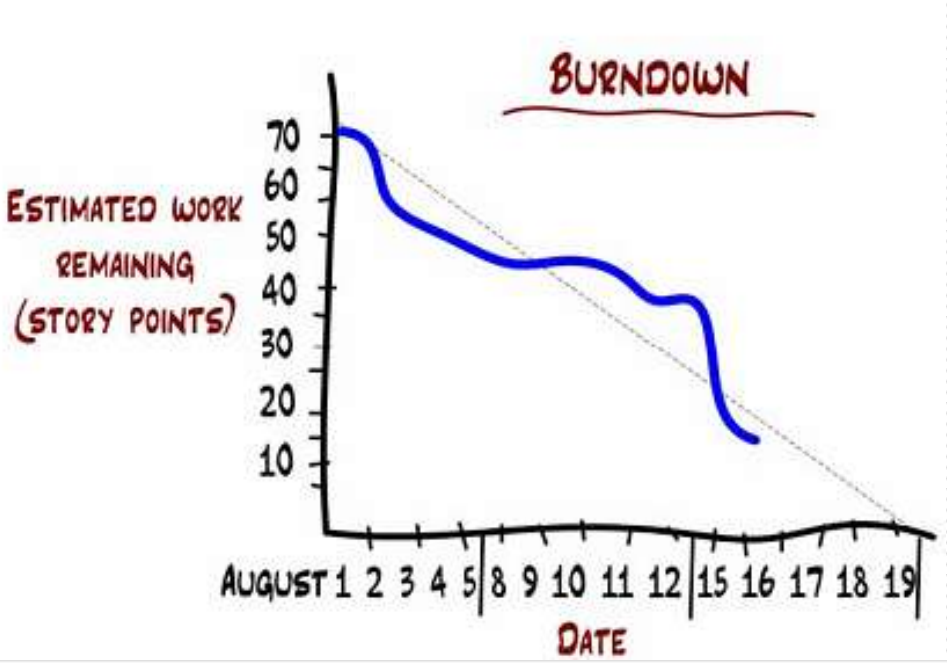
- 每日会议：

  目的：信息同步平台，非交流问题、讨论问题渠道。

  形式：固定地点、时间的站立会议。

- 生产率估算
- 燃尽线：

  Sprint剩余的工作量



敏捷开发原则和方法

- 迭代式开发。

  即整个开发过程被分为几个迭代周期，每个迭代周期是一个定长或不定长的时间块每个迭代周期持续的时间一般较短，通常为一到六周。

- 增量交付。

  产品是在每个迭代周期结束时被逐步交付使用，而不是在整个开发过程结束的时候一次性交付使用。每次交付的都是可以被部署到用户应用环境中被用户使用的、能给用户带来即时效益和价值的产品。

敏捷开发原则和方法

- 开发团队和用户反馈推动产品开发。

  主张用户能够全程参与到整个开发过程中。这使需求变化和用户反馈能被动态管理并及时集成到产品中。同时，团队对于用户的需求也能及时提供反馈意见。

- 持续集成。

  新的功能或需求变化总是尽可能频繁地被整合到产品中。一些项目是在每个迭代周期结束的时候集成，有些项目则每天都在这么做。

- 开发团队自我管理。

  拥有一个积极的、自我管理的、具备自由交流风格的开发团队，是每个敏捷项目必不可少的条件。

人是敏捷开发的核心。敏捷开发总是以人为中心建立开发的过程和机制，而非把过程和机制强加给人。

敏捷开发的常见问题

- 敏捷思想的"坏话"

  敏捷过程、极限编程适合Web的开发、适合免费的Web服务、适合永远的Beta版本。但也存在系列问题，只适合小型团队、适合开源社区等，而不适合大型软件企业；在软件开发过程的全局上，更适合采用统一过程（RUP）、微软软件开发框架（MSF），而在局部、细节，吸收敏捷思想。

  文档

- 敏捷不能解决问题，只能让问题暴露的更早

  敏捷方法是项目的救星吗？不，我认为它依然会失败。敏捷开发可以带给你的一件事情是：让这些项目失败的更快、损失 的更少，因为你可以将时间和精力用于开始做下一件事情。

  Kent Beck(XP法创始人)

敏捷开发的常见问题

Sean Landis总结出的十一个问题

1. 技术负债在敏捷团队中会快速的膨胀。

   Uncle Bob认为应该在最初的敏捷宣言中加入第五条原则“Craftsmanship over Crap”（精益求精胜过敷衍了事），来强调技术的对成功的敏捷项目的重要性。

2. 敏捷软件开发团队会想当然地认为每个团队成员都专业，称职并富有责任心。如果事实不是如此，项目开发很快会变得举步维艰。

   很多方法论认为只能通过审查监控的手段来确保项目的顺利运行，而敏捷团队更多的是依靠个人的责任心。

   在优秀的敏捷团队中，能力较弱的的团队成员会感受到来自其他成员的压力，要不然尽力做好，要不然只有走人。

敏捷开发的常见问题

3. 由于对敏捷开发实践的错误理解，导致团队不合理地频繁交付，疲于奔命。
4. 实施敏捷的门槛太高，敏捷开发需要更强的团队和个人的纪律性，勇于承诺和高度的公开性，但对一个不成熟的组织来说这个门槛太高。

   但这并不意味着不能在不成熟的组织中导入敏捷实践。这类组织可以逐步地导入敏捷实践。
5. 绩效差的团队成员很难在高度公开的敏捷团队中掩饰自己能力的不足。好的团队往往能够采取一定的措施来帮助这类成员。但如果没有采取措施，这些成员往往会想方设法通过消极怠工来掩饰自己能力的不足。

敏捷开发的常见问题

6. 敏捷团队容易过份关注眼前的短期目标，而忽视长期的战略目标。尽管在短期内能够取得成功，长期可能还是会失败。
7. Product Owner承担了太多的责任，不堪重负，从而成为团队的瓶颈。
8. 敏捷的效用被过度夸大，大家的期望值太高，很多人认为导入敏捷能以最小的投入解决实际开发中的所有问题。

   它毕竟不是万灵药。不要使他人有过高的期望。

敏捷开发的常见问题

9. 可能出现另一种形式的“相互诟病”。成功的敏捷开发团队一般不会成为产品开发的瓶颈，因此其他部门不能以这个为借口来指责开发团队，但是这有可能进一步演变成为政治游戏。

   建议尽早与其他部门沟通，大家的最终目标是一致的，各个部门应当一起寻找生产系统的瓶颈，然后努力突破瓶颈。基于这个共同目标，各个部门一起对流程进行修改，就会减少相互诟病。

敏捷开发的常见问题

10. 当Product Owner开始决定开发的方向，他就会被过度授权。敏捷开发中缺乏足够的审查和平衡机制。

    这可能不是一个问题！ Product Owner熟悉业务，明确他最终想要什么。尽管开发团队要利用技术手段，提供解决方案，满足业务需求。但作为开发团队不应该对业务方面干涉太多。

11. 敏捷实践大多是针对程序员的，很难在组织内平衡工作量。缺乏对团队中的非程序员提供更好的文档以及培训支持。

- Chris Taylor总结

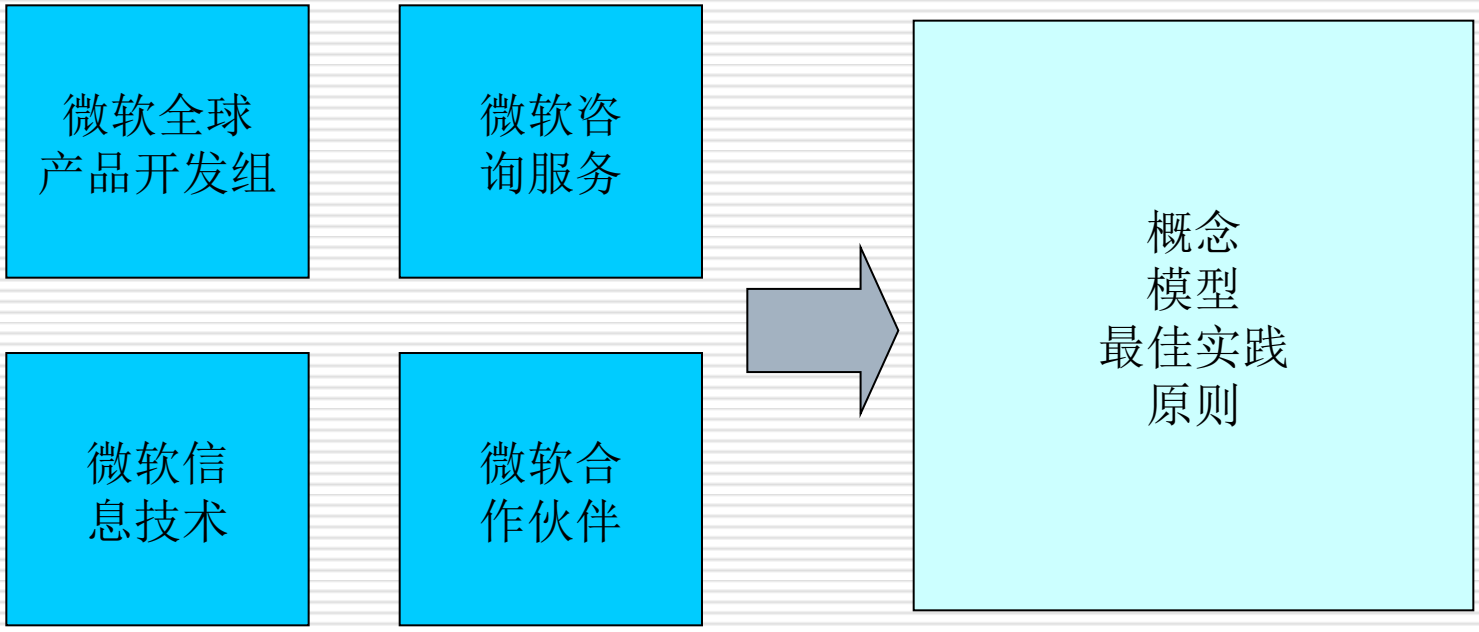
敏捷理论很美好，但是实践起来还是会有各种各样的问题，也有可能失败。

其实理论描述的是理想情况，实际情况往往不尽相同。但是我们不能因为这个就放弃向理想努力。尽管过去有很多团队导入敏捷失败，我们还是不能全面否定敏捷，毕竟也有很多成功的敏捷团队。正如敏捷项目团队在开发中不断进行反省修正一样，我们也要通过反省来加深对敏捷的理解和认识。

2.3.4 MSF模型

- Microsoft Solutions Framework (MSF)

MSF来源



2.3.4 MSF模型

参考:

- 《Microsoft Solutions Framework Essentials》(ISBN 9780735623538)。 Microsoft Press。

- http://msdn.microsoft.com/zh-cn/library/jj161047.aspx

2.3.4 MSF模型

- MSF是一个经验知识库

  它包括以下方面的内容：

  - 企业结构设计方案

    采用交互的方式，侧重于制定长期规划，同时也能完成短期目标。

  - 项目开发准则

    包含组队模型和过程模型，用于建立高效的项目组，管理项目的生命周期。

  - 项目设计过程和多层结构的应用程序模型

    用于支持设计复杂的分布式企业应用。

  - 企业信息基础设施的实施方法

    使用组队模型和过程模型支持实现、操作和技术上的方案。

2.3.4 MSF模型

- MSF 关注于：
  - 使业务和技术目标相匹配
  - 建立明确的项目目标、角色和责任
  - 实现迭代、里程碑/检查点驱动过程
  - 主动管理风险
  - 有效地响应变化

2.3.4 MSF模型

- MSF的基本观点
  - （1）用户的需求是变更的
  - （2）需求是未来的，而不是当前的
  - （3）资源永远匮乏
  - （4）风险普遍存在
  - （5）开发小组是协作的平等关系
  - （6）认识是渐进的，过程是迭代的
  - (7) 技术模型也可以影响业务模型

MSF概念

- 是一种框架结构

  框架结构不是一种预先决定工作结构、工作任务和发布产品具体方法的方法论，而是提供了灵活的方式、应用有创造力的方法去解决实际存在问题的思想。

- 一个资源的集合

  MSF收集了一组集成的资源和准则来指导项目组走向成功。它包括明确的概念、详细的工作指南和微软最好的实践经验，保证您能立即开始工作。

MSF概念

- 是一种框架结构

  若干原则或准则：
  - 基础原则
  - 观念
  - 项目管理准则
  - 风险管理准则
  - 就绪管理准则

  二种模型：
  - 小组模型
  - 监管模型（以前称为过程模型）

MSF 基础原则

原则是对如何为团队定向以获得最大限度地成功进行指导。MSF 的核心是 9 项基本原则：

- 打造开放的通信。

  共享适当级别的信息。 团队要了解需要执行的操作的性质，以及团队成员和外部联系人如何沟通。

- 为了实现共同的愿景努力工作。
- 授权团队成员。

  团队成员不仅可以生存，而且还可学会创造性地寻找成功的方式，同时相互帮助。创建表现良好的团队。

- 建立明确的责任和共同分担的责任。

MSF 基础原则

MSF 的核心是 9 项基本原则：(续)

- 交付增量价值。
- 保持灵活、期望并适应变化。
- 进行投入来提高质量。

  很多组织信奉质量至上（通常是一个松散定义的术语），但缺少对如何量化质量的了解。 质量必须主动地合并到解决方案交付生命周期中，而不只是发生即可。

MSF 基础原则

MSF 的核心是 9 项基本原则：(续)

- 从所有经验中学习。

  如果不总结经验，那么如何期望他们下次可以改进？

  团队成员必须理解并珍惜在所有级别中发生的学习：例如，

  - 在项目级别学习优化整个项目范围的过程
  - 在个人级别，学习如何更好地与其他团队成员交互
  - 在组织级别，学习对每个项目调整收集哪些质量指标
- 与内部和外部客户合作。

MSF观念

将团队成员作为个人进行定向以获得最大限度地成功则称为观念。以下是每个团队成员应该内在化的一些观念：

- 培养对等方的团队。

  如果每个人都了解使命及其目标，以便他们在交付解决方案时具有共同的愿景以及角色和责任，那么每个人都可作为对等方并可以被同等对待。 这是让每个人共同承担成功交付解决方案的责任。 每个人即对整个项目负责，又单独对项目相应的方面负责。

  仍需有计划管理员角色，但此角色侧重于在项目约束内交付项目，而不在于管理团队成员。

MSF观念

- 侧重于业务价值。

  不仅交付客户所需要的内容，而且还交付客户想要的内容以及价值。 若要交付价值，团队中的每个人都需要了解客户认为什么有价值。

  无法向客户提供业务价值使项目面临风险：偏离方向的风险;花费很多误导的时间、工作量和金钱的风险;甚至被取消的风险。

- 保持解决方案的视角。

  不要陷入细节之中而无法看到整体解决方案。

MSF观念

- 工艺出色。

  质量是整个解决方案交付生命周期中每个人的责任。团队成员可以帮助最终产品的持续改进。

- 持续学习。
- 将服务质量内在化。

  服务质量 (QoS) 定义解决方案的预期操作特征(质量特性)。

  不仅是架构师，利益相关者和团队成员也必须了解 QoS 。 否则，可能导致QoS 要求不一致。

MSF观念

- 做个好公民。

  从软件开发的角度来看，好公民意味着在你工作的所有方面都值得信赖、自豪，并且负责任和受到尊重。 这包括但不限于你如何：

  - 与你的资深团队成员、组织和利益相关者交互
  - 参与项目并帮助交付解决方案，包括成为企业、项目和计算资源的受信任的守护者。 这包括开放和自愿共享资源、信息和知识。好公民会奉行和关注更大的利益。
- 履行你的承诺。

MSF组队模型

- MSF组队模型展示了如何组织项目队伍，在时间控制和连续不断发展计划的要求下，有效的交付系统的解决方案。
- MSF 团队模型将典型的解决方案交付活动和责任划分为七个支持组。 这些组相互依赖并处于多领域。

  每一个角色都应对以下方面具有独特的视角：什么是需要的、什么是必须支持的，以及与交付解决方案相关联的目标应该是什么。

  这些角色可以在小团队的情况下进行合并，并在大团队的情况下进行扩展。

MSF组队模型

角色	目标	职能领域
产品管理	• 确保解决方案交付业务价值 • 定义项目约束内的解决方案 • 确保满足客户的需求和期望	• 市场/企业通信 • 业务分析 • 产品计划
计划管理	• 交付项目约束内的解决方案 • 设置满足支持方的需求和期望的方式	• 项目管理 • 计划管理 • 资源管理 • 过程保证 • 项目质量管理 • 项目操作
体系结构	• 设计解决方案以满足项目约束内的业务目标	解决方案体系结构 技术体系结构
开发	生成规范的解决方案	• 解决方案开发 • 技术咨询

MSF组队模型

角色	目标	职能领域
用户体验	• 最大程度地提高解决方案可用性 • 增强用户准备情况和有效性 • 确保满足用户的需求和期望	• 可访问性 • 国际化 • 技术支持通信 • 培训 • 可用性 • 用户界面设计
测试	只有在确保该解决方案的所有方面满足或超过它们各自定义的质量级别之后，批准发布解决方案	• 回归测试 • 功能测试 • 可用性测试 • 系统测试
版本/操作	平滑部署及转换到运营 确保满足 IT/业务运营需求和期望	• 版本管理 • 交付基础结构 • 操作 • 生成管理 • 工具管理

MSF组队模型

- 建立MSF小组的关键原则
  - 平等小组
  - 以客户为中心意识
  - 产品意识（注意：面向产品，而不是原型）
  - 零缺陷意识
  - 乐于学习
  - 有激情的小组是有效的

MSF组队模型

- 小组成功的六个目标：
  - 客户满意
  - 在项目的约束下交付解决方案
  - 按规格说明构造
  - 只能在标识和解决所有产品质量问题后批准发布
  - 提高用户工作效率
  - 平滑部署和连续运行

MSF监管模型

- 监管模型可在正确的时间提供正确的指导给正确的人。
- 模型旨在帮助团队对如何履行解决方案的各个方面达成共识。
- 监管模型最小化风险、改进解决方案的质量以及加快开发速度的灵活 MSF 组件。
- MSF 监管模型将项目监管与过程制定相关联。

  项目监管侧重于优化解决方案交付过程，并有效和高效地使用项目资源。

  过程制定侧重于定义、生成和部署满足利益相关者的需要和期望的解决方案。
- MSF 监管模型的关键方面包括：活动的重叠轨迹、同步检查点以及递增的方法，以便为客户交付价值。

MSF监管模型

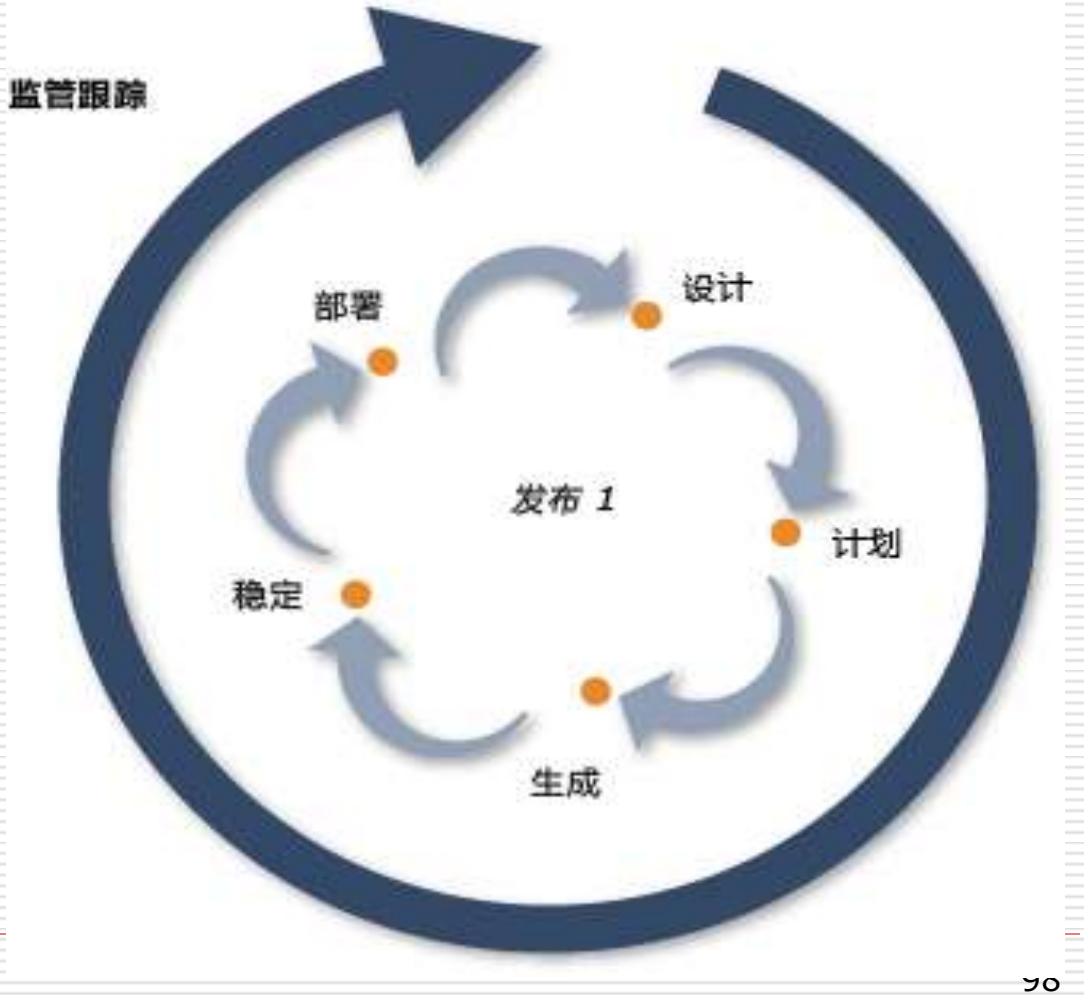
- 轨迹

  通过使用活动的重叠轨迹，MSF 监管模型可促进项目监管和过程制定。

  一般，轨迹是某些活动的重叠且协调的组，这些活动针对生成每次跟踪的相关可交付结果。 但是，MSF 轨迹比这更多；每一条都有独特的使命，并表示项目节奏和焦点中的变化。 轨迹在每次跟踪中使用评审和称为检查点的同步点，以帮助确定是否实现跟踪的目标。 此外，主要检查点还用来结束每次跟踪，这可实现对指导许多活动的责任的转移，并鼓励团队为下个跟踪的目标更适当地采取新的视角。

MSF监管模型

- MSF 监管模型由五个重叠的制定轨迹和贯穿整个制定轨迹的持久监管跟踪组成。



MSF监管模型

- 制定轨迹

  是详细的步骤顺序，通过这些步骤定义、生成和部署解决方案。

  制定轨迹可帮助团队达成预想的高级别协议，并创建方法选项以实现该愿景（预想跟踪）；评估这些选项并计划选定选项（计划跟踪）；生成解决方案（生成跟踪）；确保按预期交付解决方案（稳定化跟踪）；以及最终部署该解决方案（部署跟踪）。

MSF监管模型

每个制定跟踪的目标如下：

- 预想
  - 清楚地了解项目约束的上下文中所需的内容。
  - 借助最好地满足这些需要的选项和方法，组建必需的团队以设想解决方案，同时最好地满足这些约束。
- 计划

  将概念解决方案发展为有形的设计和计划，这样它可以在生成跟踪中进行生成。
- 生成

  依照计划跟踪可交付结果（如设计、计划、时间表和要求）生成解决方案的各个方面。

MSF监管模型

每个制定跟踪的目标如下：

- 稳定化
  - 提高解决方案的质量，以满足部署到生产的版本标准。
  - 验证解决方案满足利益相关者的需要和期望。
  - 从用户的角度验证解决方案的可用性。
  - 将成功最大化，并尽量减少解决方案的目标环境中与解决方案部署和操作相关联的风险。
- 部署
  - 将解决方案成功地集成到指定环境内的生产。
  - 将剩余解决方案交付的责任尽可能平稳并尽快地从项目团队传输到运营和支持团队。

MSF监管模型

- 监管跟踪

  遵循一组可能不断变化的项目约束，监管跟踪侧重于平衡项目资源和解决方案交付的有效和高效使用。 此外，监管跟踪还支持持续过程改进。

- 监管跟踪的目标如下：
  - 指导制定活动以交付带有可重复和可靠结果的解决方案
  - 优化并持续改进团队的表现和吞吐量、解决方案的质量以及过程改进
  - 安全审批来自：
    - 解决方案满足他们的需要并充分可用的用户
    - 准备好解决方案进行部署的运营部门
    - 完成项目的客户

MSF监管模型

- 检查点

  用于计划和监视项目进度并促进可交付结果和活动的完成。

  检查点用来为团队和客户再次确认项目范围提供明确的机会，或调整项目范围以反映不断变化的客户或业务要求，或在项目的过程期间适应可能会具体化的风险和问题。

  检查点用于许多方面，例如：

  - 帮助同步工作元素。
  - 提供进度和质量的外部可见性。
  - 支持中途更正。
  - 将评审的重点放在目标和可交付结果上。
  - 继续进行下一步之前，先提供工作的审批点。

MSF监管模型

- 检查点

  MSF 区分两种类型的检查点：主要检查站和临时检查点。

  主要检查点标记主要活动和主要可交付结果的完成，包括给定跟踪的计划活动的结束。

  由团队定义临时检查点以指示跟踪内的进度并将较大的工作分为可处理的片段。

MSF监管模型

- 迭代方法

  解决方案不提供业务价值，直到将它部署到生产并有效地使用。 因此，MSF 监管模型的生命周期包括解决方案生产阶段的增量开发和部署，从而确保业务价值以及团队总体战略愿景和目标的实现。 对于将明确的重点放在对过程中业务影响的团队，强大的多维业务表示形式组合是 MSF 确保项目履行技术的承诺的方式。

  迭代开发的实践是 MSF 中重复的主题。 以迭代方式开发文档、设计、计划和其他可交付结果。 如你所料，MSF 监管模型是一种迭代方法。

MSF过程模型

- MSF过程模型

  将瀑布模型和螺旋模型的最优秀的原理结合起来组成一体。

- 瀑布模型：

  使用里程碑来界定一个阶段到另一阶段的转变

- 螺旋模型：

  用于快速开发和不断完善过程（没有里程碑）

MSF过程模型的特点

- 基于阶段和里程碑的方法

MSF中的里程碑分为“主里程碑”和“中间里程碑”。

主里程碑是项目阶段的转换点，是角色职责的转移点。MSF中主里程碑有“远景/范围”、“项目计划认可”、…“部署成功”。

中间里程碑是指两个主里程碑之间的小的工作目标指示物或工作成果。是里主程碑所代表的任务分解过程

MSF过程模型的特点

- 迭代的方法
  - “迭代开发”是MSF中一个重复发生的主题。代码、文档、设计、计划和其他的工作成果都是以迭代的形式出现的。
  - MSF建议一个解决方案可以先构建、测试、开发出一个核心的功能。然后，其他的功能特征可以被加入，这就是通常所说的发布策略。对于一些小的工程来说，它通常只需一个版本。然而，微软推荐把它们分成多个版本，从而可以找到改进的机会。
  - 版本发布没有必要按顺序进行，成熟的软件产品经常会有多个版本重叠的发布周期。版本发布之间的间隔时间，根据项目的规模、类型、用户要求和策略的不同而不同。

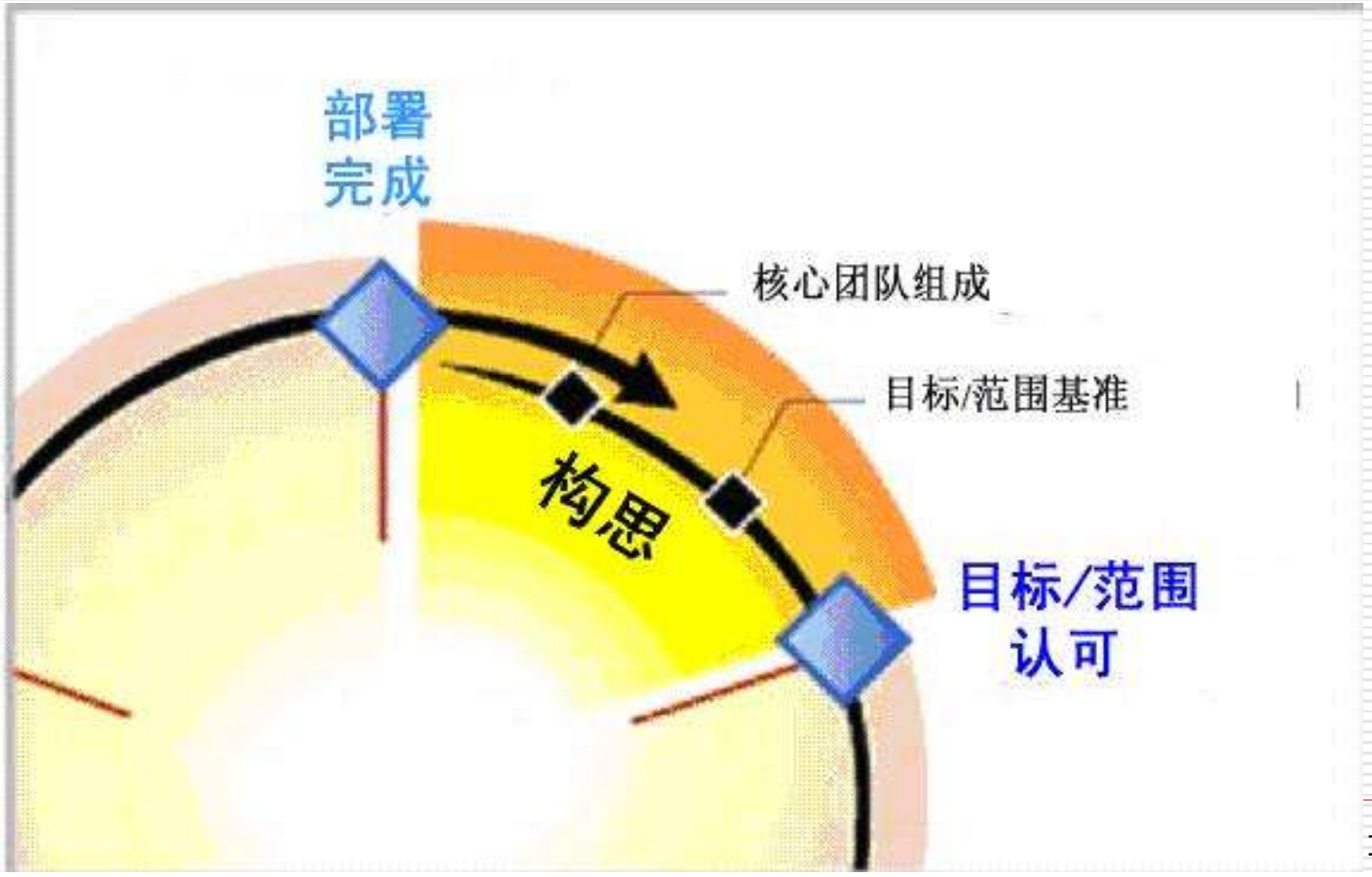
MSF过程模型的特点

- 整合了构建和部署的方法

  一个解决方案在没有完全被部署到生产环境中之前，是没有办法提供价值的。由于这个原因，MSF认为只有在开发完成后成功部署并交付价值以后，才开始下一个迭代过程。

MSF过程模型各阶段的工作成果

- 构思阶段



MSF过程模型各阶段的工作成果

- 构思阶段

1. 目标：创建一个关于项目的目标、限定条件和解决方案的架构
2. 团队的工作重点

   a) 确定业务问题和机会；b) 确定所需的团队技能；

   c) 收集初始需求；d) 创建解决问题的方法；

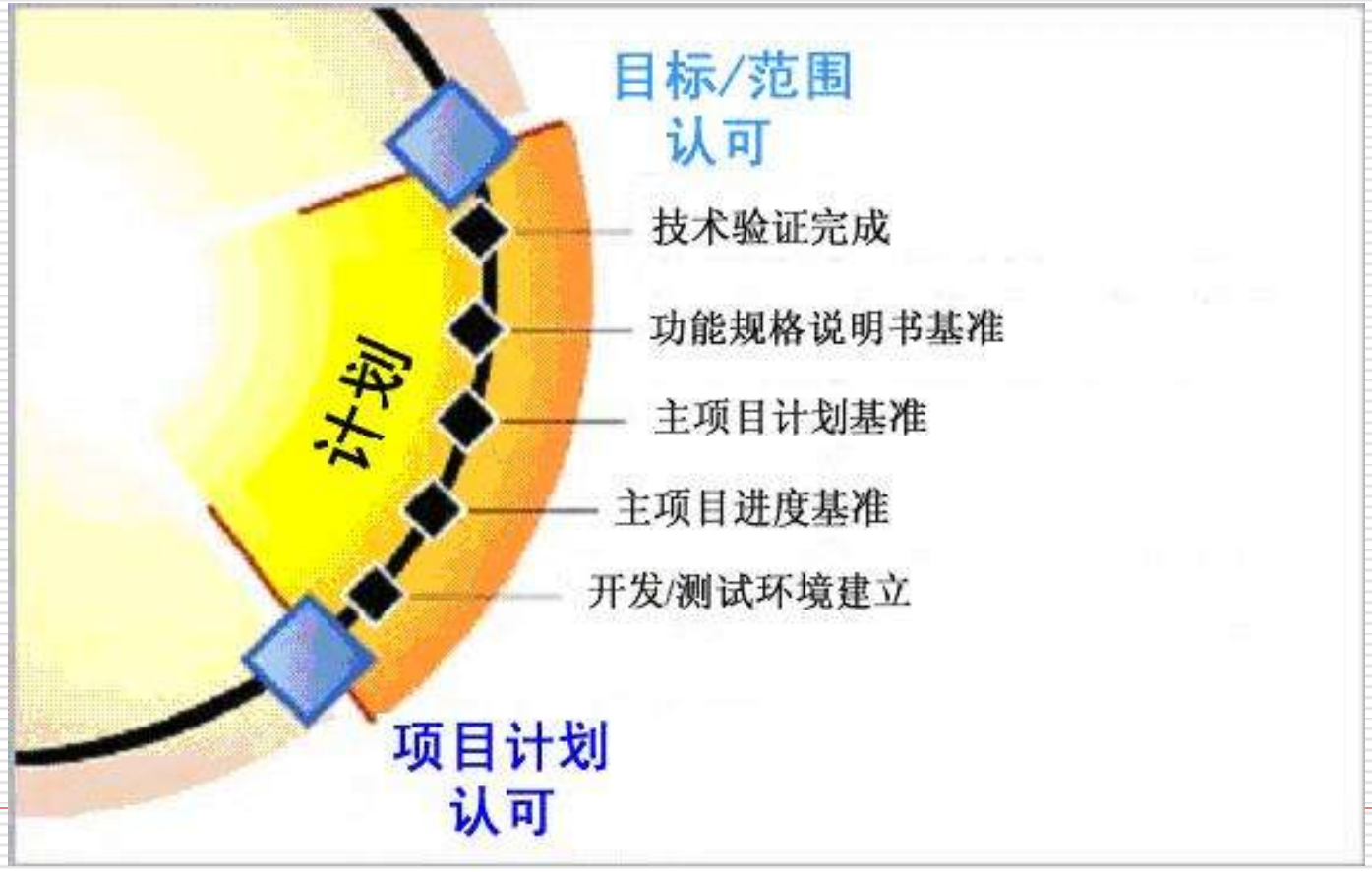
   e) 确定目标、假设和限定条件；f) 建立配置与变更管理
3. 交付成果

   a) 远景/范围文档；b) 项目结构文档；

   c) 初始风险评估文档

MSF过程模型各阶段的工作成果

- 计划阶段

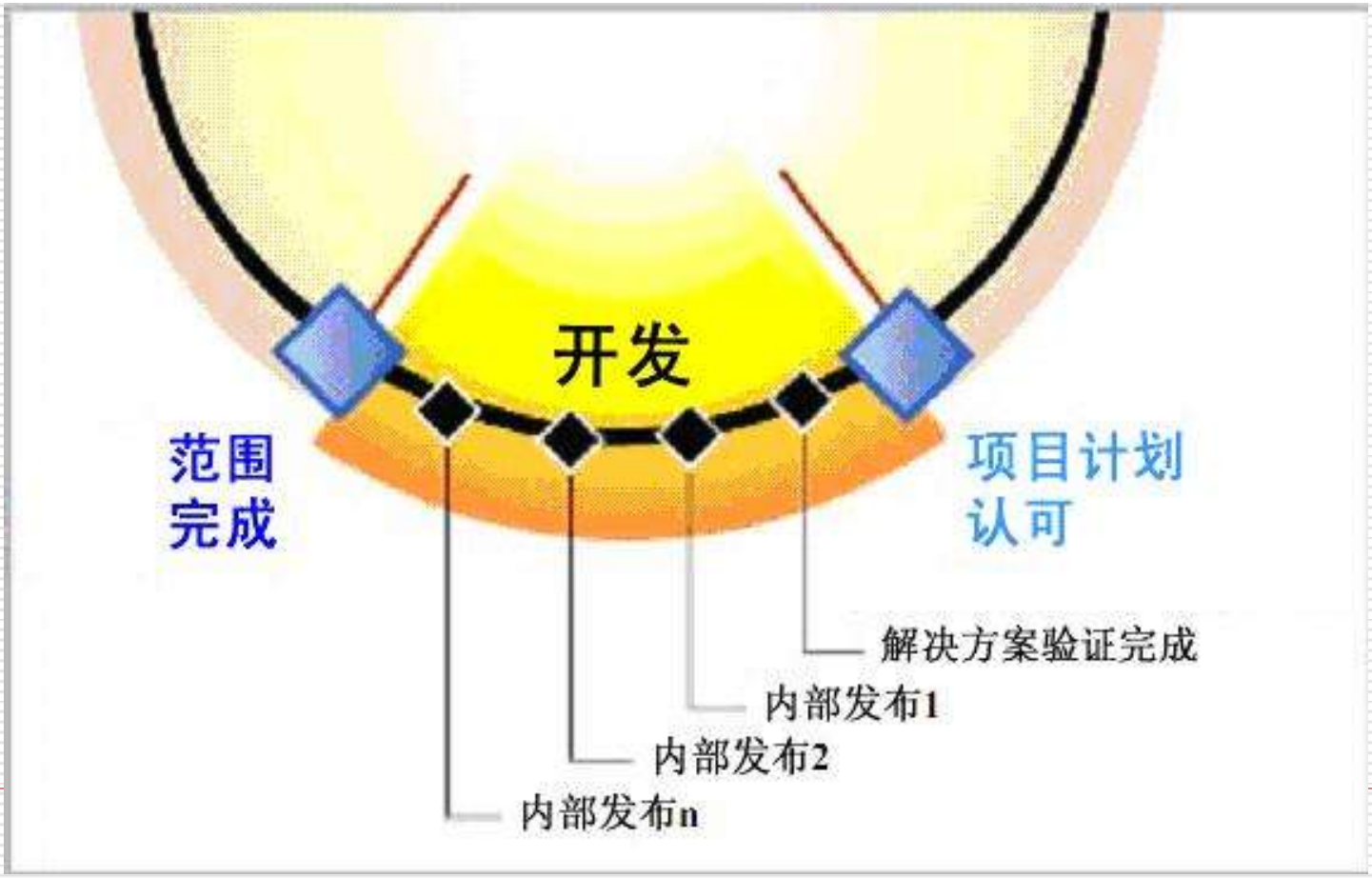


MSF过程模型各阶段的工作成果

- 计划阶段
  1. 目标：创建解决方案的体系结构和设计方案、项目计划和进度表
  2. 团队重点
     a) 尽可能早地发现尽可能多的问题
     b) 知道项目何时收集到足够的信息以向前推进
  3. 交付成果
     c) 功能规格说明书
     d) 主项目计划
     e) 主项目进度表

MSF过程模型各阶段的工作成果

- 开发阶段



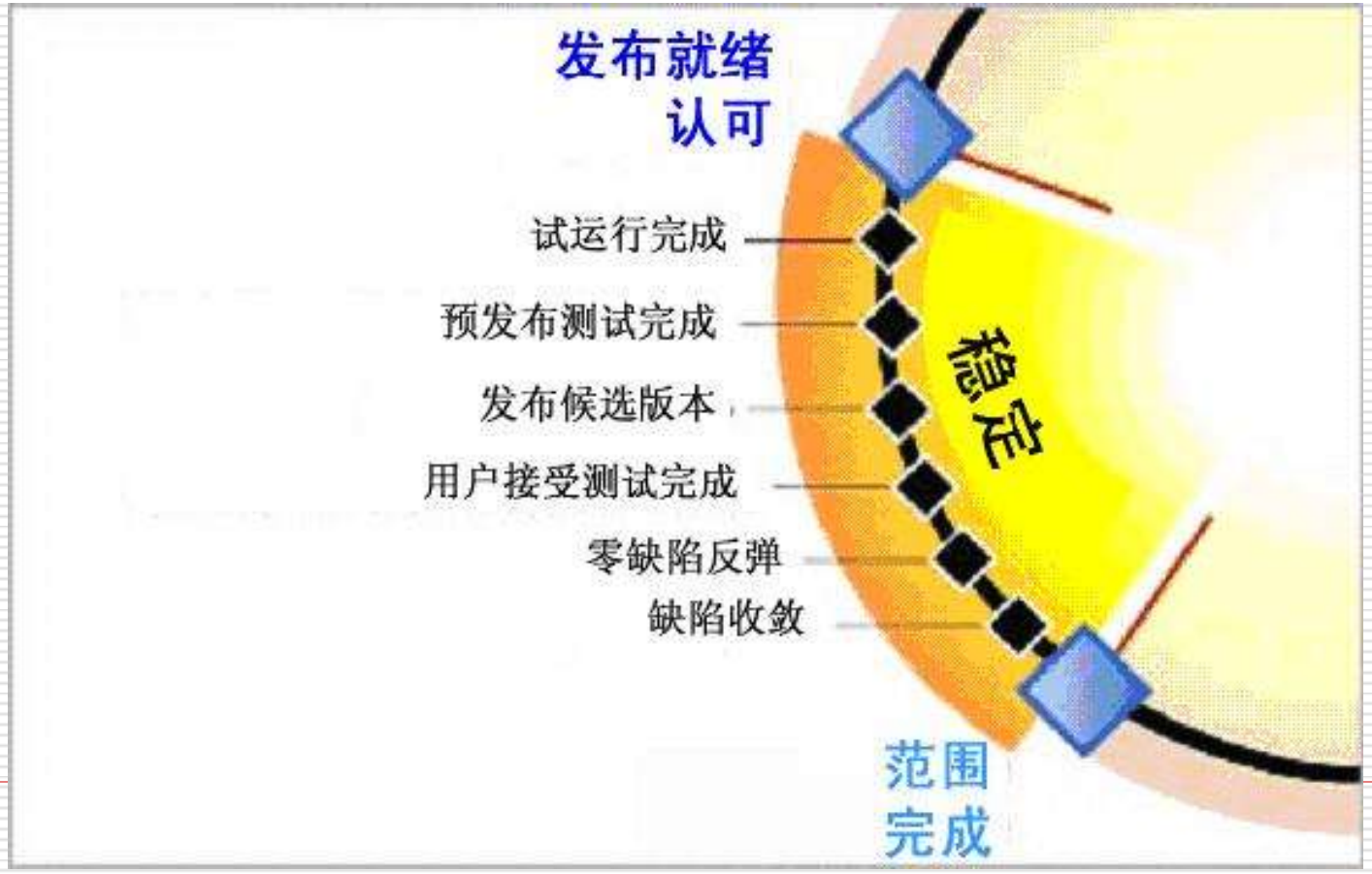
MSF过程模型各阶段的工作成果

- 开发阶段

1. 目标：完成功能规格说明书中所描述的功能、组件和其他要素
2. 团队主要工作
   a) 编写代码　　b) 开发基础架构
   c) 创建培训课程和文档　　d) 开发市场和销售渠道
3. 交付成果
   a) 解决方案代码　　b) 构造版本　　c) 培训材料
   d) 文档（包括部署过程、运营过程、技术支持、疑难解答等文档）
   e) 营销材料　　f) 更新的主项目计划、进度表和风险文档

MSF过程模型各阶段的工作成果

- 稳定阶段



MSF过程模型各阶段的工作成果

- 稳定阶段

1. 目标：提高解决方案的质量，满足发布到生产环境的质量标准
2. 团队的工作重点

a) 提高解决方案的质量　b) 解决准备发布时遇到的突出问题

c) 实现从构造功能到提高质量的转变

d) 使解决方案稳定运行　　e) 准备发布

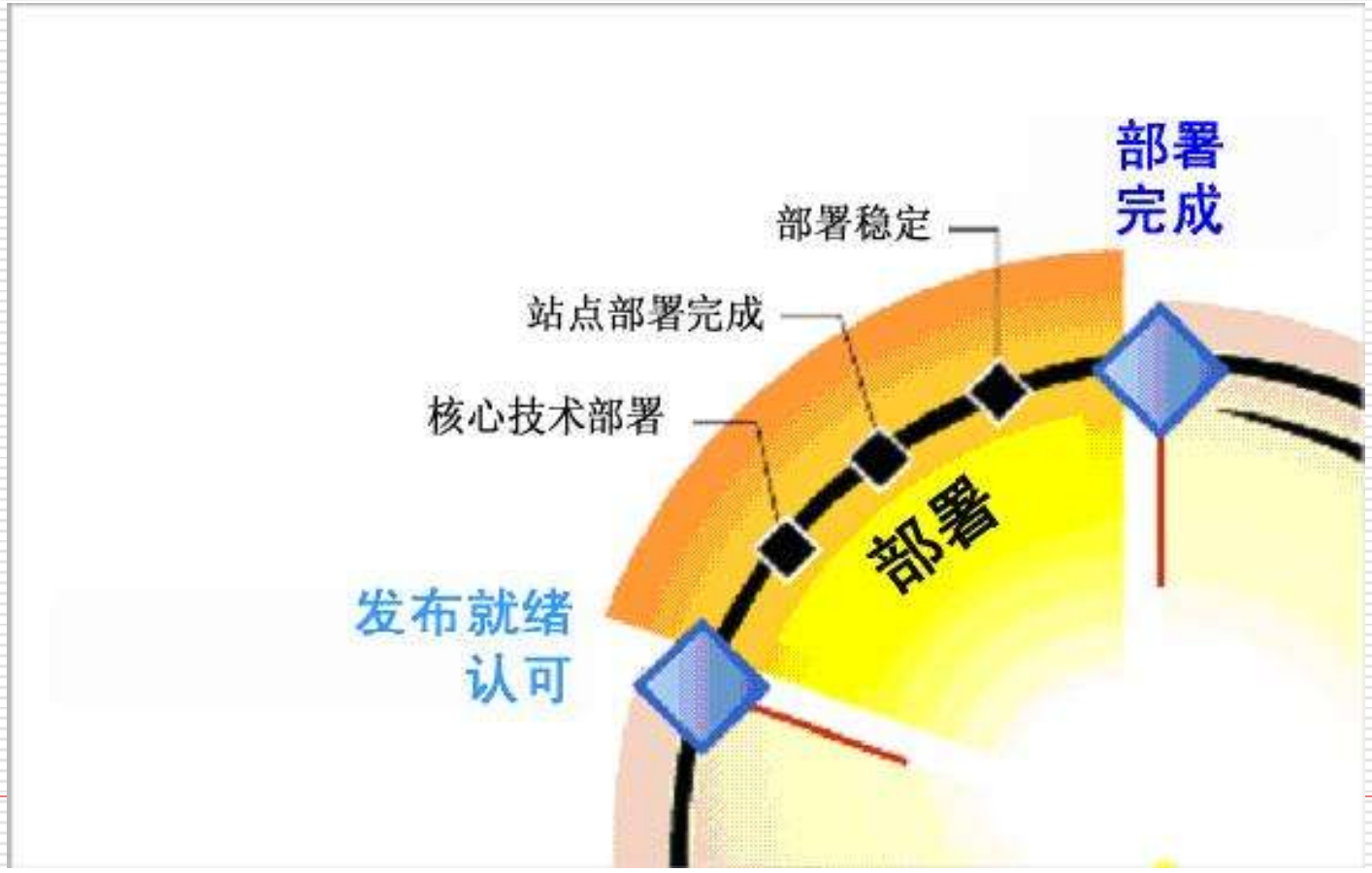
3. 交付成果

a) 试运行评审　　b) 可发布版本（包括源代码、可执行文件、脚本、安装文档、最终用户帮助、培训材料、运营文档、发布说明等）

c) 测试和缺陷报告　　d) 项目文档

MSF过程模型各阶段的工作成果

- 部署阶段



MSF过程模型各阶段的工作成果

- 部署阶段
  1. 目标：把解决方案实施到生产环境之中
  2. 团队的工作重点
     a) 促进解决方案从项目团队到运营团队的顺利过渡
     b) 确保客户认可项目完成
  3. 交付成果
     a) 运营及支持信息系统
     b) 所有版本的文档、装载设置、配置、脚本和代码
     c) 项目收尾报告

MSF准则

- MSF准则
  - 风险管理

    风险识别，风险分析，风险计划，风险跟踪，风险控制，风险学习
  - 就绪管理

    定义，评估，变更（培训、进度跟踪），评价
  - 项目管理

    项目范围 ，变更控制，预算、成本控制和时间表，沟通，供应商管理

2.3.5 RUP简介

- RUP（Rational Unified Process，统一过程)是一个面向对象且基于网络的程序开发方法论。
- 根据Rational的说法，好像一个在线的指导者，它可以为所有方面和层次的程序开发提供指导方针，模版以及事例支持。
- RUP是理解性的软件工程工具--把开发中面向过程的方面（例如定义的阶段，技术和实践）和其他开发的组件（例如文档，模型，手册以及代码等等）整合在一个统一的框架内。

2.3.5 RUP简介

二维的软件开发模型

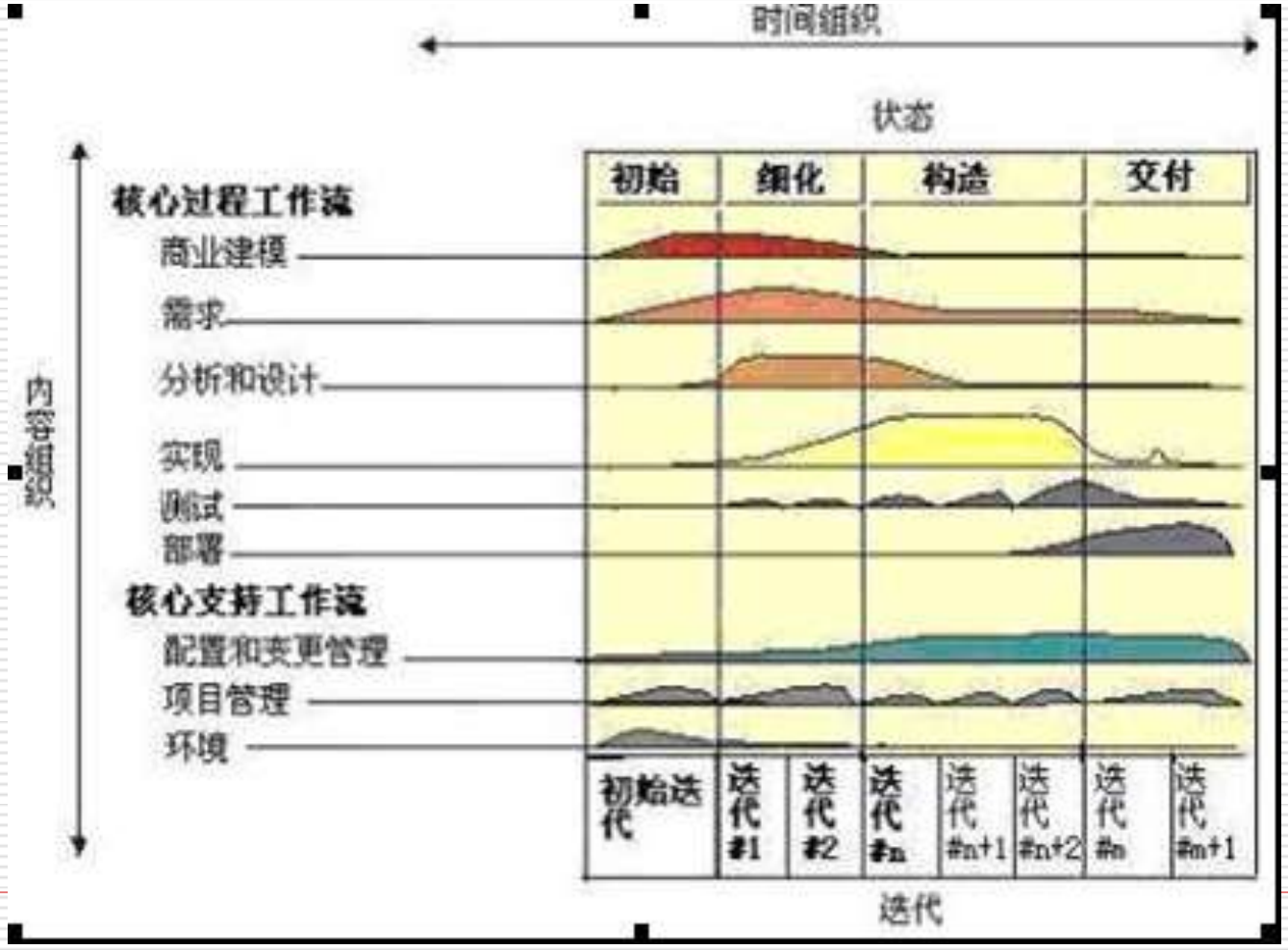
传统的软件开发模型是一个单维的模型，开发工作划分为多个连续的阶段。在一个时间段内，只能作某一阶段的工作。

RUP软件开发生命周期是一个二维的软件开发模型。

横轴通过时间组织，是过程展开的生命周期特征，体现开发过程的动态结构；

纵轴以内容来组织为自然的逻辑活动，体现开发过程的静态结构。

2.3.5 RUP简介


时间组织
状态
初始
细化
构造
交付
核心过程工作流
商业建模
需求
分析和设计
实现
测试
部署
核心支持工作流
配置和变更管理
项目管理
环境
内容组织
初始迭代
迭代#1
迭代#2
迭代#n
迭代#n+1
迭代#n+2
迭代#m
迭代#m+1
迭代

2.3.5 RUP简介

- 动态结构：迭代式开发

  从组织管理的角度描述整个软件开发生命周期。

  用来描述它的术语主要包括周期(Cycle)、阶段(Phase)、迭代(Iteration)和里程碑(Milestone)；

  在时间维上，RUP把整个软件开发周期划分为若干个周期，每个周期生成一个产品的新版本。周期依次由四个连续的阶段组成，每个阶段都应完成确定的任务：

2.3.5 RUP简介

- 动态结构：迭代式开发（续）

  周期包括的四个阶段：

  - 初始阶段：定义最终产品视图、商业模型并确定系统范围；
  - 细化阶段：设计并确定系统的体系结构，制定工作计划及资源要求；
  - 构造阶段：构造产品并继续演进需求、体系结构、计划直至产品提交；
  - 提交阶段：把产品提交给用户使用。

  每个阶段结束前都有一个里程碑（MileStone）评估该阶段的工作。如果未能通过该里程碑的评估，则决策者应该决定是取消该项目，还是继续进行该阶段的工作。

2.3.5 RUP简介

- 动态结构：迭代式开发（续）

  每个阶段可以进一步分解为迭代。

  一个迭代是一个完整的开发循环，产生一个可执行的产品版本，是最终产品的一个子集，它增量式地发展，从一个迭代过程到另一个迭代过程到成为最终的系统。

  与传统开发模型不同，RUP没有给出迭代流的具体实现步骤；

  RUP的迭代开发过程（迭代的个数、每个迭代的延续时间、目标）是受控的。

2.3.5 RUP简介

- 静态结构：方法描述

RUP采用以下四个基本模型元素，组织和构造系统开发过程：

- Workers:角色，它描述某个人或一个小组的行为与职责。
- Activities:是一个有明确目的的独立工作单元。
- Artifact：以多种形式存在，包括模型（Model）、原代码、可执行文件和文档等。 是Activity生成、创建或修改的一段信息。
- Workflows：描述了一个有意义的连续Activititys序列，每个Workflow产生一些有价值的Artifacts，并显示出Workers之间的关系。

RUP主要提供两种组织Workflow的方式：核心工作流（Coreworkflows）和迭代工作流（Iterationworkflows）。

2.3.5 RUP简介

- RUP的核心工作流

RUP中有9个核心工作流，分为6个核心过程工作流和3个核心支持工作流。

9个核心工作流在项目中轮流被使用，在每一次迭代中以不同的重点和强度重复。

- 商业建模：理解待开发系统的组织结构及其商业运作，确保所有参与人员对待开发系统有共同的认识；
- 需求分析：定义系统功能及用户界面，使客户了解系统的功能、开发人员了解系统的需求，为项目预算及计划提供基础；
- 分析与设计：把需求分析的结果转化为实现规格；

2.3.5 RUP简介

- RUP的核心工作流 （续）
  - 实现：定义代码的组织结构、实现代码、单元测试和系统集成；
  - 测试：校验各子系统的交互与集成，确保所有的需求被正确实现并在系统发布前发现错误；
  - 发布：打包、分发、安装软件，升级旧系统；培训用户及销售人员，并提供技术支持，制定并实施beta测试；
  - 配置管理：跟踪并维护系统所有Artifacts的完整性和一致性；
  - 项目管理：为计划、执行和监控软件开发项目提供可行性指导；为风险管理提供框架和环境；为组织提供过程管理和工具的支持。
  - 环境 ：向软件开发组织提供软件开发环境，包括过程和工具。

2.3.5 RUP简介

- 统一软件开发过程RUP裁剪

通过对RUP进行裁剪可以得到很多不同的开发过程，这些软件开发过程可以看作RUP的具体实例。RUP裁剪可以分为以下几步：

1) 确定本项目需要哪些工作流。

2) 确定每个工作流需要哪些制品。

3)以风险控制为原则确定4个阶段之间如何演进。

4) 确定每个阶段内的迭代计划。规划RUP的4个阶段中每次迭代开发的内容。

5) 规划工作流内部结构。通常用活动图的形式给出。

2.3.5 RUP简介

- 最佳软件工程实践
  - 迭代式开发。
  - 管理需求。
  - 基于组件的体系结构。
  - 可视化建模。
  - 验证软件质量。
  - 控制软件变更。

开发过程比较

- 不同的项目需要不同的方法论，一个项目的最佳过程是这个项目所能负担的最小过程。

项目	CMM/CMMI	RUP	MSF	XP
周期	螺旋模型。	演进式迭代周期，过程框架	瀑布模型和螺旋模型的结合	演进式迭代周期。软件开发方法学
核心	过程改进	架构、迭代	里程碑、迭代	以代码为中心。
范围	需求严格而极少变化的项目。	适合不同类型的项目	适合不同类型的项目	进度紧、需求不稳定的小项目、小型发布和小团队
组织	个人（PSP）、团队（TSP）和组织的3个层次，组间协作、培训	跨团队协作	强调产品的愿景，6种基本角色	以团队为基础，小团队、团队成员能力相当
技术	传统结构化方法	面向对象技术	综合技术	面向对象技术

项目	CMM/CMMI	RUP	MSF	XP
管理	侧重于过程的定义、度量和改进。一切用数字和文档说话。	从组织角度出发，侧重于过程建模、部署。	业务建模、部署、过程管理等概念。	侧重于具体的过程执行和开发技术，计划设计。
活动	通过过程域来定义活动	整个团队在整个过程中关注质量	项目管理、风险管理和就绪管理	以人为本，如每周40小时工作制、结对编程
实践	各类级别的关键实践。重视关键基础设施。	满足了CMM 2-3级KPA 的要求，而基本上没有涉及CMM 4-5 级的KPA	代码复审、版本管理方法、文档管理、人员招聘、重测试和重风险管理等。	编码和设计活动融为一体，弱化了架构。用例、单元测试、迭代开发和分层的架构。

项目	CMM/CMMI	RUP	MSF	XP
其它	通用性强，但复杂、高成本。	强调风险驱动，以保障可用产品的持续性交付为前提，尽量减少不必要的过程工件，使度量、文档最小化以获得弹性和应变能力。	提供了一系列指南，用于规划企业的基础技术设施，流程化商业的运作过程，并鼓励重用性。	拥抱变化，强调人性化、简单、沟通。尽量减少文档。 个体和交互胜过过程和工具。

经过实践检验和积累的、自我定义的软件过程才是最好的过程。

2.3.6 软件建模

- 模型是什么？
- 建模的使用是软件成功的一个基本因素。
- 模型的实质：对现实的简化。
- 建模的目标 ：
  - 便于展现系统。
  - 允许指定系统的结构或行为。
  - 提供构造系统的模板。
  - 记录决策。

面向对象建模

- 两种最常用的建模方法：基于算法的和面向对象的。
- 随着需求的变化和系统的增长，运用基于算法的建模方法建立起来的系统很难维护。
- 在面向对象的建模方法中，主要的模块是对象或者类。

UML代表着软件建模的发展趋势

- UML（unified modeling language，统一建模语言）共定义了三大类，若干种模型图。
  - 结构类模型图（structural diagrams）：用4种模型图描述系统应用的静态结构，包括类图、对象图、组件图和配置图。
  - 行为类模型图（behavior diagrams）：用5种模型图描述系统动态行为的各个方面，包括用例图、序列图、行为图、协作图和状态图。
  - 模型管理类模型图（model management diagrams）：用3种模型图来组织和管理各种应用模型，包括软件包、子系统和模型。

2.4 软件开发与软件测试的关系

1、测试与开发各阶段的关系

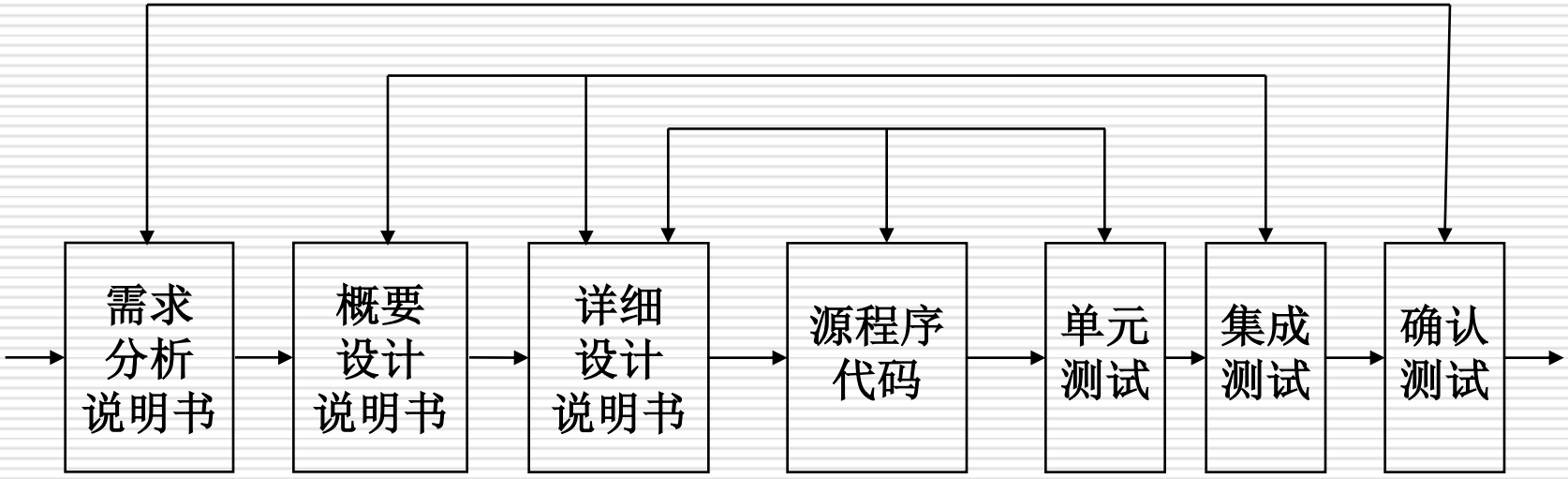


图　软件测试与软件开发过程的关系

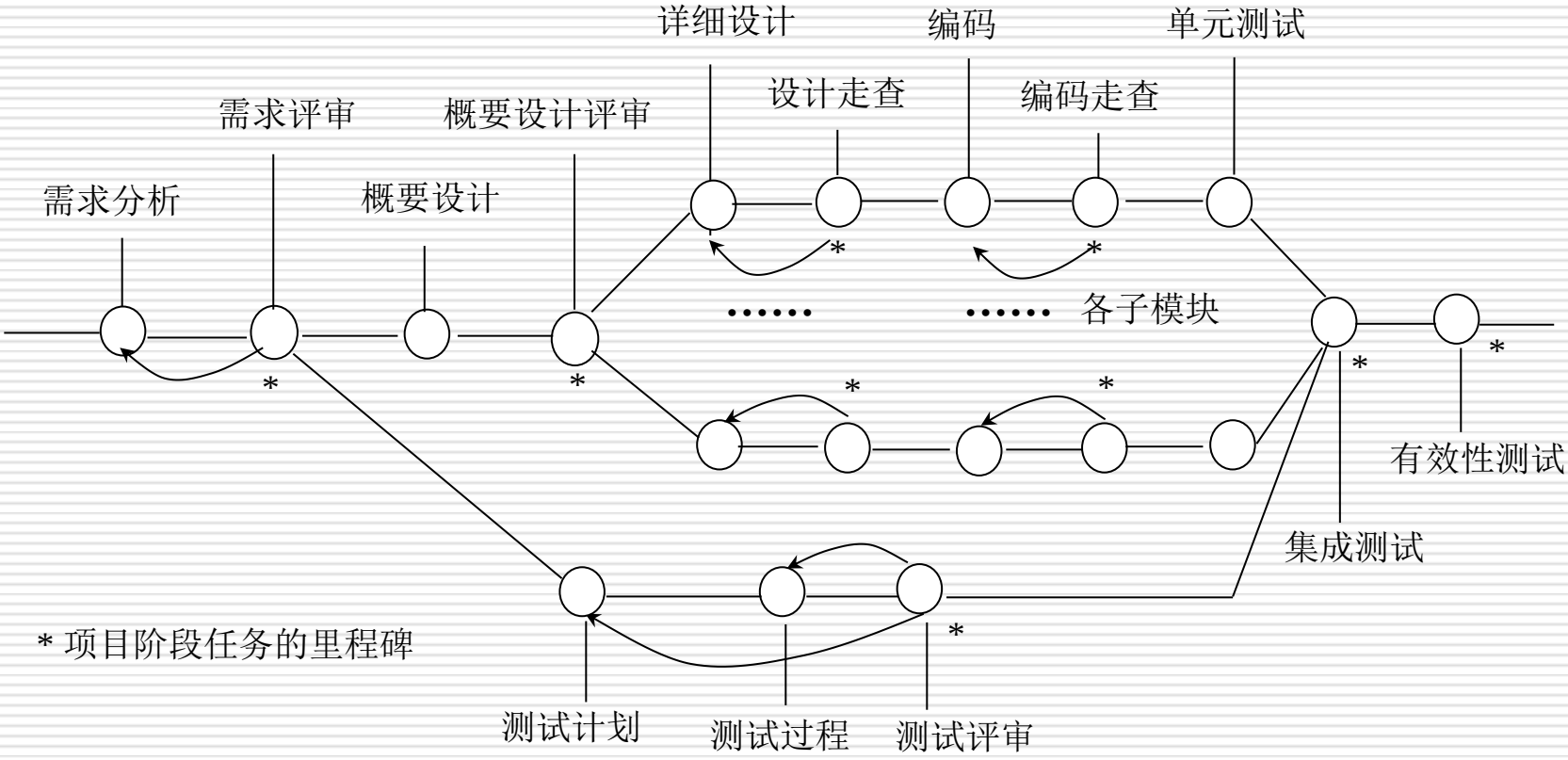
2.4 软件开发与软件测试的关系（续）

测试在开发阶段的作用：

- 项目规划阶段：负责从单元测试到系统测试的整个测试阶段的监控。
- 需求分析阶段：确定测试需求分析、系统测试计划的制定，评审后成为管理项目。
- 详细设计和概要设计阶段：确保集成测试计划和单元测试计划完成。
- 编码阶段：由开发人员进行自己负责部分的测试代码。在项目较大时，由专人进行编码阶段的测试任务。
- 测试阶段（单元、集成、系统测试）：依据测试代码进行测试，并提交相应的测试状态报告和测试结束报告。

2.4 软件开发与软件测试的关系（续）

2、测试与开发的并行性



软件开发过程3　软件测试与软件开发的并行性

2.4 软件开发与软件测试的关系（续）

3、完整的软件开发流程

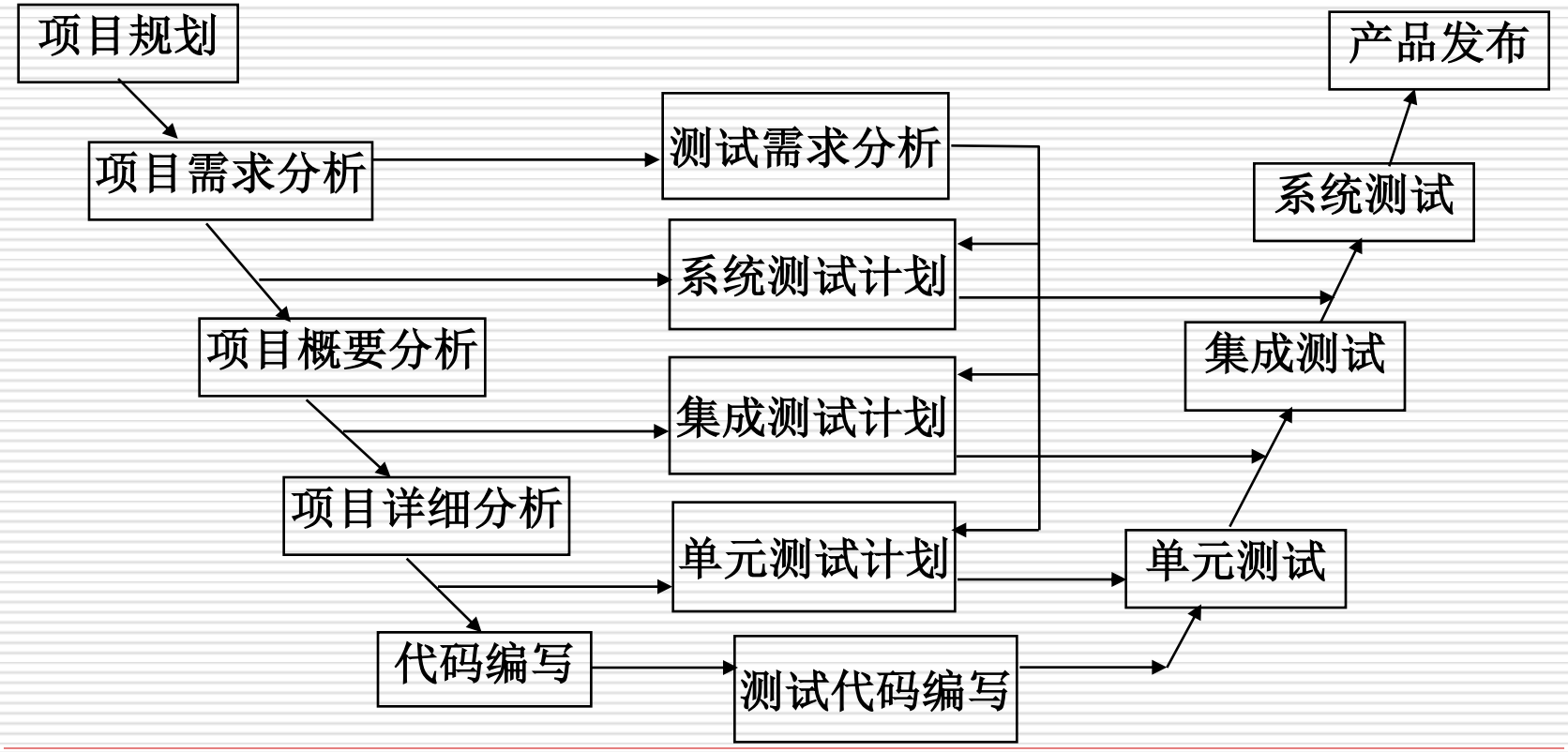


图　完整的开发流程

2.5 软件神话

软件神话具有一些特征使得它们很有欺骗性：如，它们表面上看很有道理(有时含有一定真实的成分)；它们符合人的直觉；它们常常是有经验的实践者发布出来的。

- **管理者的神话**

管理者有巨大的压力，要维持预算、保持进度及提高质量。就像溺水者抓住一根救命稻草，软件管理者常常抓住软件神话不放。

- **神话**：我们已经有了关于建造软件的标准和规程的书籍，难道它们不能给人们提供所有其需要知道的信息吗？
- **事实**：不错关于标准的书籍已经存在，但真正用到了它们吗？软件实践者知道它们的存在吗？它们是否反映了现代软件开发的过程？它们完整吗？很多情况下，对于这些问题的答案均是“不”。

（软件神话 摘自《软件工程——实践者的研究方法》）

2.5 软件神话

- **神话**：我们已经有了很多很好的软件开发工具，而且，我们为它们买了最新的计算机。
- **事实**：为了使用最新型号的主机、工作站和PC 机去开发高质量的软件，我们已经投入太多的费用。实际上，计算机辅助软件工程(CASE)工具相比起硬件而言对于获得高质量和高生产率更为重要，但大多数软件开发者并未使用它们。
- **神话**：如果我们已经落后于计划，可以增加更多的程序员来赶上进度(“有时称为蒙古大夫概念”)。
- **事实**：软件开发并非象制造一样是一个机械过程。用Brooks的话来说，“给一个已经延迟的软件项目增加人手只会使其更加延迟”。看起来，这句话与人的直觉正好相反。但实际上，增加新人，原来正在工作的开发者必须花时间来培训新人，这样就减少了他们花在项目开发上的时间。人手可以增加，但只能是在计划周密、协调良好的情况下。

2.5 软件神话

用户的神话

在许多情况下，用户相信关于软件的神话，因为负责软件的管理者和开发者很少去纠正用户的错误理解。神话导致了用户过高的期望值，并引起对开发者的极端不满意。

- **神话**：有了对目标的一般描述就足以开始写程序了——我们可以以后再补充细节。
- **事实**：不完善的系统定义是软件项目失败的主要原因。关于待开发项目的应用领域、功能、性能、接口、设计约束及确认标准的形式化的、详细的描述是必需要的。这些内容只有通过用户和开发者之间的通信交流才能确定。

2.5 软件神话

- **神话**：项目需求总是在不断变化，但这些变化能够很容易地满足，因为软件是灵活的。
- **事实**：软件需求确实是经常变化的，但这些变化产生的影响会随着其引入的时间不同而不同。如果我们很注重早期的系统定义，这时的需求变化就可被很容易地适应。用户能够复审需求，并提出修改的建议，这时对成本的影响会相对较小。当在软件设计过程中才要求修改时，对成本的影响就会提高得很快。资源已经消耗了，设计框架已经建立了，这时的变化可能会引起大的改动，需要额外的资源和大量的设计修改，例如，额外的花费。实现阶段(编码和测试阶段)功能、性能、接口及其他方面的改变对成本会产生更大的影响。当软件已经投入使用后再要求修改，这时所花的代价比起较早阶段做同样修改所花的代价可能是几何级数级的增长。

2.5 软件神话

开发者的神话

那些至今仍被软件开发者相信的神话是由几十年的程序设计文化培植起来的。如，在软件的早期阶段，程序设计被看作是一门艺术。这种旧的观念和方式是很难改变的。

- **神话**：一旦我们写出了程序并使其正常运行，我们的工作就结束了。
- **事实**：有人说过：“越早开始写程序，就要花越长时间才能完成它”，产业界的数据表明在一个程序上所投入的50%到70%的努力是花费在第一次将程序交给用户之后。

2.5 软件神话

- **神话**：在程序真正运行之前，没有办法评估其质量。
- **事实**：从项目一开始就可以应用的最有效的软件质量保证机制之一是正式的技术复审。软件复审是“质量的过滤器”，比起通过测试找到某类软件错误要有效得多。
- **神话**：一个成功项目唯一应该提交的就是运行程序。
- **事实**：运行程序仅是软件配置的一部分，软件配置包括：程序、文档和数据。文档是成功开发的基础，更重要的是，文档为软件维护提供了指导。

许多软件专业人士认识到上述这些神话是错误的。但令人遗憾的是旧的观念和方法培植了拙劣的管理和技术习惯，虽然现实情况已经需求有更好的方法。对软件现实的认识是形成软件开发的实际解决方案的第一步。