第一章软件工程概述

软件是计算机系统中与硬件相互依存的另一部分。它包括程序、数据及其相关文档的完整集合。

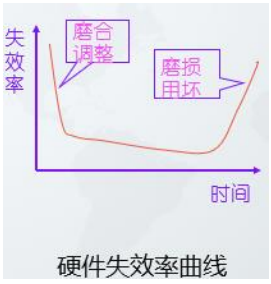
（1）能够完成预定功能和性能的可执行指令（program）

（2）使得程序能够适当地操作信息的数据结构（data）

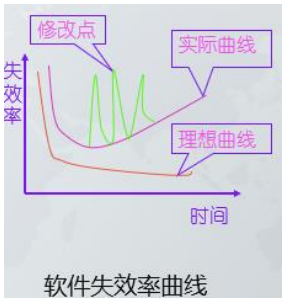
（3）描述程序的操作和使用的文档（document）

软件特点：

- 软件是一种逻辑实体，而不是具体的物理实体。
- 软件的生产与硬件不同。
- 在软件的运行和使用期间，没有硬件那样的机械磨损，老化问题，但它存在退化问题，开发人员必须维护软件。
- 大多数软件是自定的，而不是通过已有构件组装而成的。
- 软件成本相当昂贵。
- 软件本身是复杂的。



硬件失效率曲线



软件失效率曲线

软件危机：软件在开发和维护过程中遇到的一系列严重问题。

软件危机包含两层含义：

1.如何开发软件。　　2.如何维护数量不断膨胀的已有软件。

软件危机的表现：

（1）软件开发的**进度难以控制**，经常出现经费超预算、完成期限拖延的现象。

（2）软件**需求在开发初期不明确**，导致矛盾在后期集中暴露，从而对整个开发过程带来灾难性的后果。

（3）软件**文档资料不完整、不合格**。由于缺乏完整规范的资料，加之软件测试不充分，从而造成软件质量低下。

（4）软件的**可维护性差**，程序错误难以改正，程序不能适应硬件环境的改变。

（5）软件**价格昂贵**，软件成本在计算机系统总成本中所占的比例逐年上升。

软件危机的原因：

（1）**客户**对软件需求的描述不精确，可能有遗漏、有二义性、有错误，在软件开发过程中，用户提出修改软件功能、界面、支撑环境等方面的要求。

（2）**软件开发人员**对用户需求的理解与用户的本来愿望有差异。不能有效地、独立自主地处理大型软件的全部关系和各个分支，因此容易产生疏漏和错误。

（3）**管理人员**、软件开发人员等各类人员的信息交流不及时、不准确、有时还会产生误解。

（4）缺乏有力的**方法和工具**方面的支持，过分地依靠程序人员在软件开发过程中的技

巧和创造性，加剧软件产品的个性化。

软件工程学的存在价值：促进软件项目成功

软件工程：是研究和应用如何以系统化的、规范的、可度量的方法去开发、运行和维护软件，即把工程化应用到软件上。

- 1968 年，“软件工程”这个术语第一次使用，作为一个会议标题，该项目由北约（NATO）赞助；该会议确认了要用定义最佳实践的方式帮助改善软件开发；

软件生存周期：是指软件产品从考虑其概念开始到该软件产品交付使用，直至最终退役为止的整个过程。一般包括计划、分析、设计、实现、测试、集成、交付、维护等阶段。

1.计划阶段

确定待开发系统的**总体目标**和**范围**。

研究系统的**可行性**和可能的解决方案，对资源、成本及进度进行合理的估算。

2.分析阶段

分析、整理和提炼所收集到的**用户需求**，建立完整的分析模型，将其编写成**软件需求规格说明**和初步的用户手册。

3.设计阶段（总体设计和详细设计）

设计阶段的目标是**决定软件怎么做**。

软件设计主要集中于**软件体系结构、数据结构、用户界面和算法**等方面。

4.实现阶段（编码）

实现阶段是将所设计的各个模块编写成计算机可接受的程序代码。

5.测试阶段

设计测试用例，对软件进行测试，发现错误，进行改正。

6.运行和维护阶段

应当在软件的设计和实现阶段充分考虑软件的**可维护性**。

维护阶段需要测试是否正确地实现了所要求的修改，并保证在产品的修改过程中，没有做其他无关的改动。

维护常常是软件生命周期中最具挑战性的一个阶段，其费用是相当昂贵的。

软件工程三要素：工具（系统），方法（技能），开发过程（框架）

瀑布模型：

问题定义	可行性研究	需求分析	概要设计	详细设计	编码	测试	运行与维护
计划时期		开发时期					运行时期

特点：

1. 自上而下、相互衔接的固定次序，如同瀑布流水，逐级下落。
2. 上一阶段的变换结果是下一阶段变换的输入，相邻两个阶段具有因果关系。

问题：

1. 各个阶段的划分完全固定，阶段之间产生大量的文档，极大地增加了工作量。
2. 开发模型是线性的，用户只有等整个过程的末期才能见到开发成果，增加了风险。
3. 早期的错误可能要等到开发后期的测试阶段才能发现，进而带来严重的后果。

RUP 统一软件过程：RUP 的中心思想是：用例驱动、架构为中心、迭代和增量。

统一建模语言 UML：语言是用来表达和沟通思想的，UML 承载着 OOAD 的思想

UML静态图	类图（Class Diagram）：模型化系统的结构
	对象图（Object Diagram）：对象及对象间的相互关系
	组件图（Component Diagram）：模型化组件的组织和依赖
	部署图（Deployment Diagram）：模型化系统的硬件分布
UML动态图	时序图（Sequence Diagram）：模型化系统的行为
	协作图（Collaboration Diagram）：模型化系统的行为
	状态图（Statechart Diagram）：模型化状态相关的方面
	活动图（Activity Diagram）：模型化系统内的事件流
	用例图（Use Case Diagram）：模型化系统与外界的交互

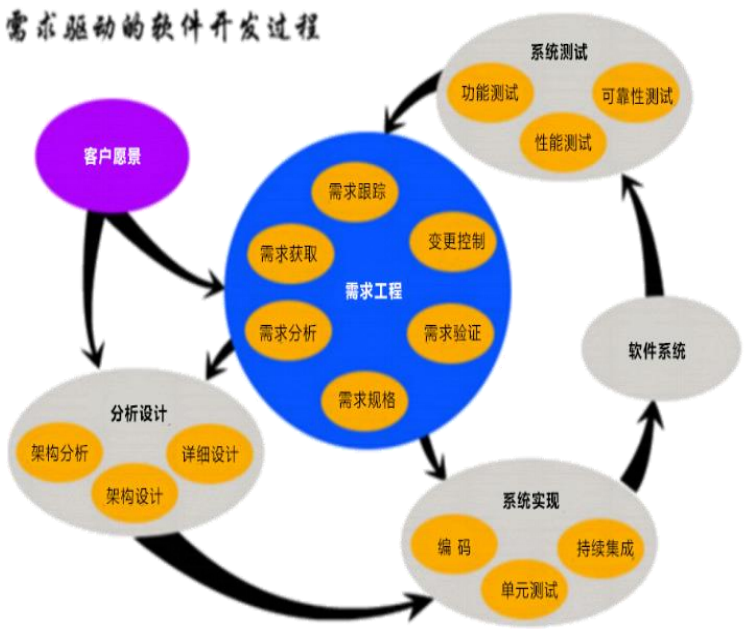
第二章　踏上 ICONIX 软件过程之路

企业：如果能帮助客户“开源、节流”客户就愿意购买我们的软件

需求是软件成功的基础，需求十分重要，并且贯穿整个软件开发的整个过程，需要引起足够的重视

ICONIX 过程总览

扩展 ICONIX 过程可分为：愿景、业务建模、需求分析、健壮性分析、关键设计、最终设计和实现这几步。它又分为两个大的部分，分别是需求阶段和系统的设计和实现阶段，又可分为四个阶段：需求分析阶段、初步设计阶段、详细设计阶段和部署阶段。它基于极限编程和敏捷软件开发的思想，提倡在项目开始阶段构建域模型和用例模型，其中用例模型驱动整个动态模型，而域模型驱动整个静态模型。ICONIX 过程是一种以最小步骤实现用例到代码的方法学，覆盖了软件过程中所有关键的环节。



需求开发：

需求调查->需求分析->需求定义

不同的软件过程，需求阶段都很重要，但方法是不一样的

需求调查->需求分析：

老板（企业战略、开源节流）->**定义愿景**

中层经理(简化管理、优化流程)->**业务建模**

一线员工(工作简单)->**用例分析**

获取愿景的三步曲：
第一步：找到软件项目的“老大”；
第二步：得到“老大”对项目的期望（愿景）；
第三步：描述出愿景的度量指标；

1. 找“老大”

要点：系统要改善哪个组织的流程？**老大就是要改善的组织中最有权力的干系人**

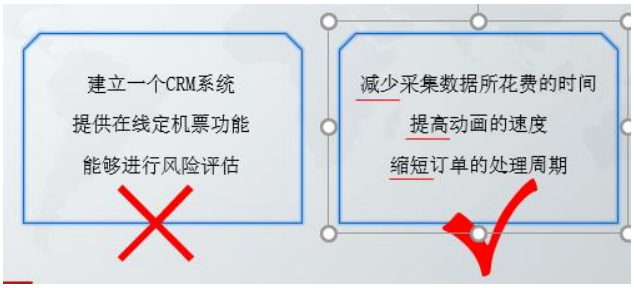
2. 得到“老大”对项目的期望（愿景）

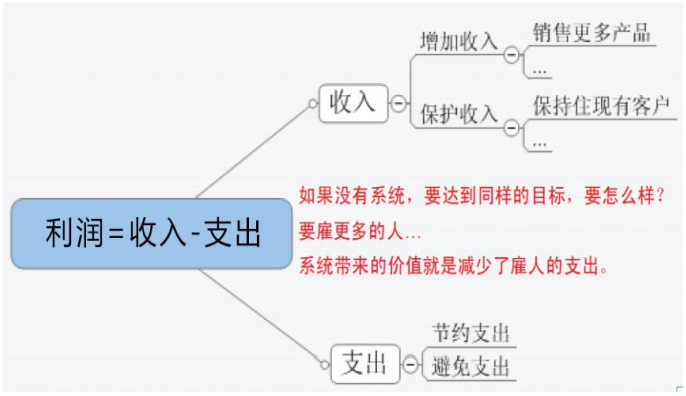
软件项目的愿景是“老大”愿意掏钱开发这个系统的目的

注意：愿景不是功能

3. 描述出愿景的度量指标

愿景必须指出度量指标。不是做具体的事，是改善组织的指标。





要点：买了这个系统对组织有什么好的用处

练习：PPT2/28 开始

第三章　业务建模，精准了解客户

业务建模的意义：

- 业务建模要求我们**把视角从软件系统转向客户组织，站在客户角度看问题**，以达到清晰准确地“诊断”，对症“开方”。
  1. 明确为谁服务--找准**客户**及其**愿景**，切记不是在为自己做系统；
  2. 要改进的组织是什么现状--有什么**痛处**和**不足**；
  3. 如何**改进**--新系统的价值就是解决客户痛处、改良客户不足，这才是客户愿意掏腰包的动力；
  4. 在业务建模和需求分析阶段，忘掉自己技术专家的身份；

步骤：

1. **明确我们为谁服务（选定愿景要改进的组织）。**
2. 要改进的组织是什么现状（业务用例图、现状业务序列图）。
3. 我们如何改进（改进业务序列图）。

第一步**选定组织：**

选定的业务组织跟老大的职权范围有关，选定业务组织要结合系统愿景

第二步**了解组织现状：**

从外部看：组织是**价值**的集合，用业务用例图来建模。

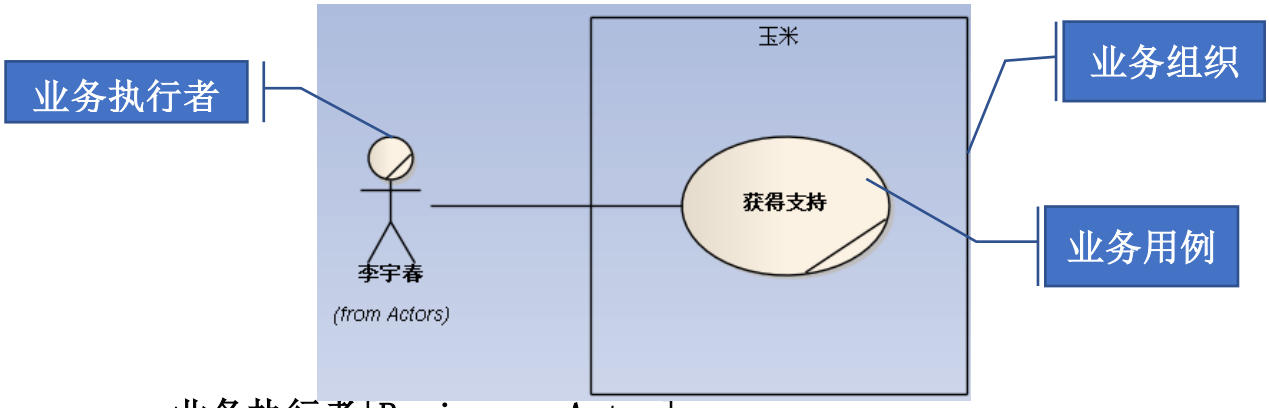
从内部看：组织是**系统**的集合(人是一种智能系统)，用业务序列图来建模

***业务用例图*:**

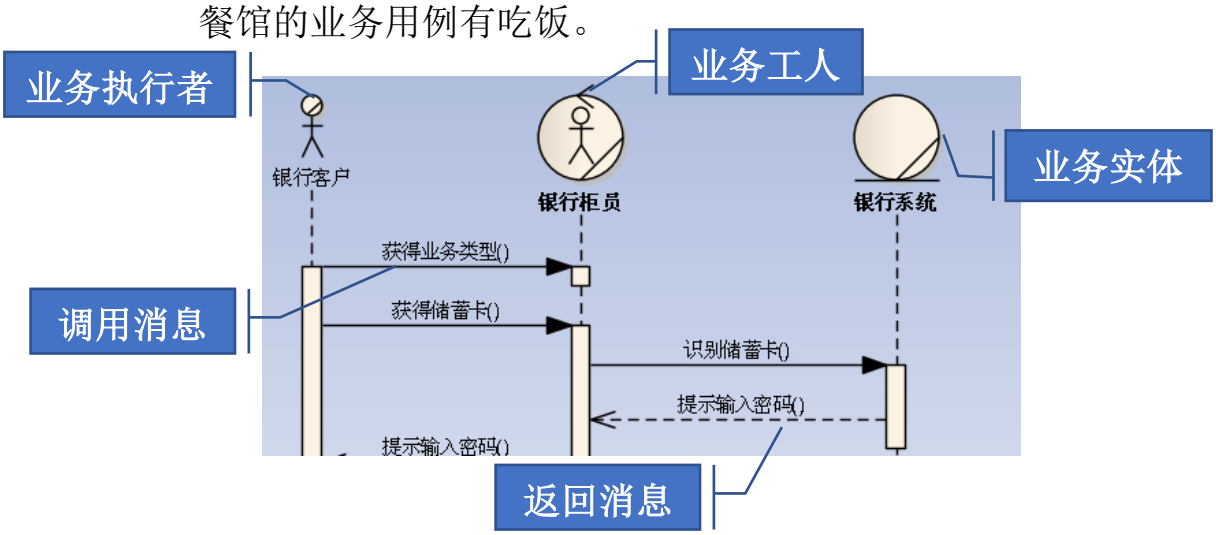
业务用例图帮助我们从**高层次**了解组织的**业务构成**

现状业务序列图（序列图以**面向对象**的思想来看业务流程）

业务序列图帮助我们从细节上了解组织的业务流程。每个业务用例都代表一条业务流程，一般用文字、活动图或序列图来描述这个流程。



- **业务执行者**[Business Actor]
  - 在业务组织之外，与其交互，享受其价值的人或组织。
  - 例如储户是银行的业务执行者，食客是餐馆的业务执行者。
- **业务用例**[Business Use Case]
  - 业务组织为业务执行者提供的价值。例如银行的业务用例有存款、贷款等；餐馆的业务用例有吃饭。



- **业务工人**[Business worker]
  - 位于业务组织内部，负责业务流程中某些工作的人员。例如银行工作人员，医院医生。
- **业务实体**[Business Entity]
  - 在业务用例的实现流程中，业务工人所使用的“**系统**”。例如银行的数钞机，学校的校园卡系统。
  - 可以和业务工人相互取代各自的职责。

第三步**改进业务组织的业务流程：**

了解业务组织现状的目的——发现流程中的改进点

接下来具体分析流程中的改进点

体会业务序列图和改进业务序列图的价值

- 业务序列图展现企业内现有业务流程，暴露问题，为优化提供直观依据
- 通过改进业务序列，可以提前模拟出新系统的出现，将对组织现行的业务流程造成哪些影响，可以提前评估新系统的可行性或提前进行相应的准备工作，实现安

全平稳的组织改进。

- 不要小看这一点，在真实世界里，这一点很关键，很多优秀的系统就因为不能适应组织的业务流程而被遗弃。

第四章　需求分析，用例分析法

域建模的意义

- 域建模[Domain Modeling]
  - 为项目创建一个术语表。确保项目中的每个人都能以清晰一致的术语来理解和交流问题领域。
  - 域建模比普通的项目术语表优良的地方体现在：以图的方式清晰地显示出不同术语间的关系（减少理解偏差）。
  - 域模型图将通过不断修正完善逐步演化为最终的静态类图。

域建模的步骤

Step 1 仔细阅读**需求文档**，提取出名词和名词短语
Step 2 排除列表中重复、相似的术语
Step 3 排除超出系统范围的术语
Step 4 画出第一版域模型图
Step 5 整理第一版域模型

域模型之间的关系

- 泛化[Generalization]，一般元素和特殊元素的关系。
- 关联[Association]，两个类之间存在着某种语义上的联系。

域模型≠数据模型（域模型设计期间不用考虑数据的存放问题，只考虑业务描述中涉及的实体以及实体之间的关系）

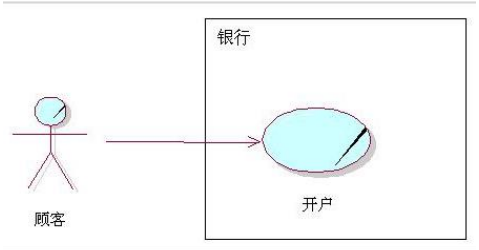
系统用例建模

- 系统需求分析的目的是把视角从业务组织转向新系统，站在最终用户及其它干系人的角度看问题。
- 系统用例是对（新）系统为系统执行者提供的价值的建模。

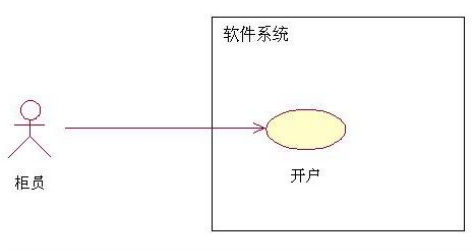
业务用例 VS 系统用例

- 对银行进行业务建模，研究对象是银行。
- 用于分析现有业务的利与弊。

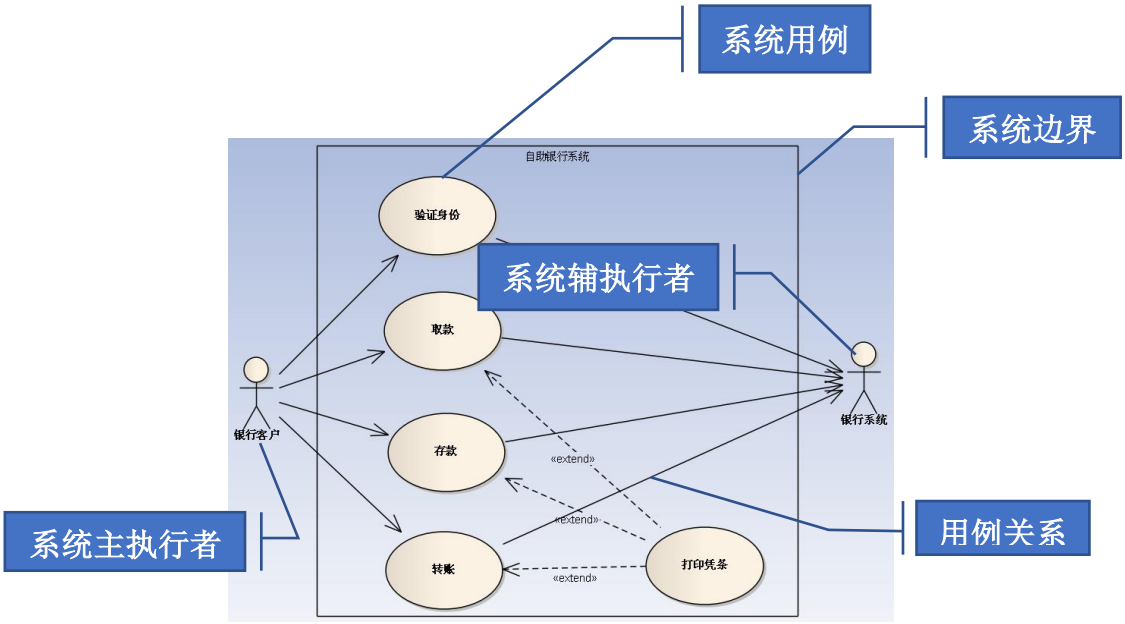
业务用例:顾客与企业组织的交互.



- 对银行的软件系统进行系统建模，研究对象是银行的软件系统。
- 用于分析新系统所带来的价值。



系统用例:用户与信息系统的交互
系统用例图包括：



系统用例建模步骤：

1. **绘制系统用例图**

   确定系统边界

   识别系统执行者(系统外)

   主执行者:用例发起者，用例为其实现有价值的目标

   辅执行者:用例支持者，用例的完成需要与其交互，得到其支持

   识别系统用例

   系统用例是系统执行的一系列动作，这些动作可以生成“执行者”可观测的**有价值**的结果

   系统用例是执行者通过系统可以**达到的某个目标**

   确定用例间的关系（泛化，包含，扩展）

2. 编写系统用例描述

   系统用例图描述总体，系统用例文档描述细节。

   每个系统用例必须对应有系统用例描述。

   用例描述的基本组成：干系人利益，基本路径，扩展路径，业务规则

   基本路径：客户最想看到的、最关心的路径

3. 更新域模型

先发现执行者还是先发现用例

- 执行者比用例明显。
- 执行者的个数远比用例的个数少。
- 找到一个执行者，就可以找到一堆用例。
- 执行者是系统外部人物的代表，所以当然是要先找到执行者，才能够从执行者的角度去寻找用例。

执行者与重要性无关

用例≠功能，用例≠步骤

用例与愿景目标

- 所有用例应该都能追溯到愿景目标。
- 所有的愿景目标都应有对应的用例实现。

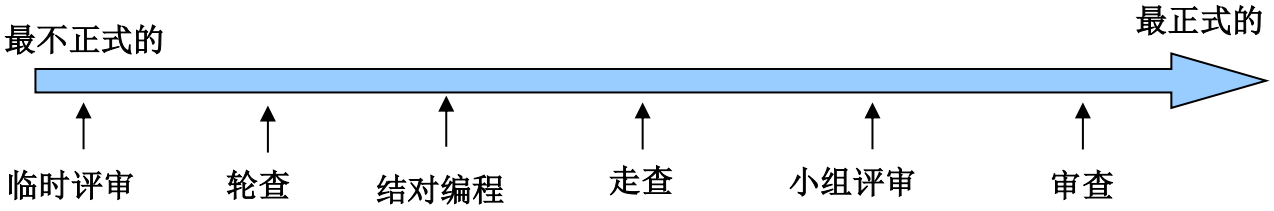
功能性需求 & 非功能性需求

- 功能性需求，通俗讲就是系统可以做什么。**人无我有**
- 非功能性需求，通俗讲就是系统可以把某项功能做到什么程度。**人有我优**

软件产品的典型非功能性需求(RUPS)：可靠性，可用性，性能，可支持性

三种相对正式的需求评审：审查，小组审评，走查

三种相对不正式的评审：结对编程，轮差，临时评审



审查过程：

1. 谁参加，评审什么
2. 总体会议（会前会）：召集参加评审会的所有成员开一个简短的会议，讨论、明确要评审的内容、评审的要点、评审时所需的资料、缺陷检查表
3. 准备：评审人员提前阅读和准备，才能提出有价值的建议
4. 审查会议：暴露问题、讨论问题，待审查文档应该符合：
   a) 遵循标准模板，统一模板易于阅读和评审；
   b) 已进行了拼写检查，减少文字错误带来的问题
   c) 已检查了排版错误
   d) 所有未解决问题提前标记好，避免大家浪费精力于还存在问题的地方
5. 返工：没有返工的评审必然沦为“形式主义”。评审中发现的错误必须得到重视和回应。
6. 跟踪
   a) 解决问题：对评审提出的问题进行解决
   b) 避免问题再次出现：对问题进行分类、因果分析，找到问题的深层次原因

第五章　需求与设计的桥梁：健壮性分析

健壮性分析帮助完善和确认需求分析的成果。

- 健壮性分析中的三种元素：
  - **边界类**[Boundary objects]与用户交互的对象，系统和外部世界的界面，如窗口，对话框等等。
  - **实体类**[Entity objects]是现实世界存在的实体对象，域模型中的类，它常对应于数据库表和文件。有些实体对象是“临时”对象（如搜索结果），当用例结束后将消失。
  - **控制器类**[Controller objects]边界和实体间的“粘合剂”,将边界对象和实体对象关联起来，它包含了大部分应用逻辑，它们在用户和对象之间架起一座桥梁。控制对象中包含经常修改的业务规则和策略。
- 健壮性分析中三种元素的交互规则：
  - 执行者只可以和边界对象通话；
  - 边界对象和控制器可以互相通话（**名词<->动词**）；
  - 控制器可以和另一个控制器通话（**动词<->动词**）；

- 控制器和实体对象可以互相通话（**动词<->名词**）；

健壮性分析的步骤
第一步：创建一个空的健壮性图。
第二步：直接将用例文本粘贴到图上（基本路径和扩展路径）。
第三步：从基本路径的第一句话开始画健壮性图。
第四步：贯串整个用例基本路径，一次一个句子，画执行者、适当的边界对象和实体对象以及控制器，和各元素之间的连线。
第五步：将每一个扩展路径画在健壮性图上，并以红色标示出。

第六章　关键设计

关键设计意义:就是要通过寻找对象之间的交互关系,进而进行方法（操作或行为）分配基于用例图、用例描述和健壮性图，采用序列图来描述参与者、边界、实体之间的交互.
关键设计的步骤：
- 第一步：将现有的域模型直接作为第一版静态类模型；
- 第二步：基于用例描述和健壮性分析结果，画出每个用例的序列图；
    - 健壮性图中的控制类会转化为方法；
    - 如果也转化为控制类，那么就添加到类图中（**注意：边界类不添加到类图中**）；
- 第三步：整理静态类图和序列图；
- 第四步：关键设计复核，迭代更新用例图、类图和序列图；

关键设计复核方法：
- 形式：面对面会议。**（可能多次、每次会很久 ☺）**
- 参会人：分析设计师、专家（分析、设计、开发）。
- 被审材料：用例图、用例描述、类图、序列图（为什么没有健壮性图？）；
- 过程：需求分析师主持，介绍需求分析成果，所有参与者交流讨论，达成统一理解和确认。
- 结论：所有参与者签字确认。　（当然，也有可能是未达成共识，需要返工。）
- **注意：已不需要甲方人员参与。这是一个技术性的复核对话，因此，需要的是拥有技术思想的人员。**

第七章　详细设计

详细设计的技术架构以及相关考虑：
- 选择开发语言
- 网络拓扑及安全
- 体系结构
- 硬件支持环境
- 软件支持环境
- ...

体系结构——三层：用户界面层，业务逻辑层，数据访问层

SCRUM 敏捷过程概述

敏捷宣言：Scrum 是当前最流行的敏捷过程

我们正在通过亲身实践以及帮助他人实践，揭示更好的软件开发方法。通过这项工作，我们认为：

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

虽然右项也具有价值，
但我们认为左项具有更大的价值。

- **敏捷宣言（ 2001 年)是敏捷起源的基础，由上述 4 个简单的价值观组成，敏捷宣言的签署推动了敏捷运动**
- **敏捷宣言本质是揭示一种更好的软件开发方式，启迪人们重新思考软件开发中的价值和如何更好的工作**

统一认识：敏捷=理念（核心思想）+优秀实践（经验积累）+具体应用（能够结合自身灵活应用才是真正敏捷）

理念：

1. 聚焦客户价值，消除浪费（产品商业成功为目标，聚焦客户价值、围绕价值流消除浪费）
2. 激发团队潜能，加强协作（人是软件开发的决定因素）
   团队是价值的真正创造者，应加强团队协作、激发团队潜能
   软件开发是一种团队活动，首先应做到提升沟通效率降低交流成本
3. 不断调整以适应变化（不断的根据经验调整，最终交付达到业务目标的产品）

① PO 和开发团队对产品业务目标形成共识
② PO 建立和维护产品需求列表（需求会不断新增和改变），并进行优先级排序
③ PO 每轮迭代前，Review 需求列表，并筛选高优先级需求进入本轮迭代开发
④ 开发团队细化本轮迭代需求，并按照需求的优先级，依次在本轮迭代完成
⑤ 开发团队每日站立会议、特性开发、持续集成，使开发进度真正透明
⑥ PO 对每轮迭代（2－4 周）交付的可工作软件进行现场验收和反馈
⑦ 团队内部进行本轮冲刺的过程回顾，发现可改进的方面，指导下一轮迭代

敏捷软件开发核心——迭代增量开发

什么是迭代式开发

- 迭代开发将整个软件生命周期分成多个小的迭代（一般 2-4 周）

- 每一次迭代都由需求分析、设计、实现和测试在内的多个活动组成
- 每一次迭代都可以生成一个稳定和被验证过的软件版本。

迭代式开发的好处

- 通过将高技术风险的需求在早期迭代里实现，有助于尽早暴露问题和及时消除风险
- 通过提供功能渐增的产品，持续从客户获得反馈，根据反馈及时调整，使最终产品更加符合客户的需要
- 通过小批量减少排队，提供更灵活、快速的交付能力
- 平滑人力资源的使用，避免出现瓶颈

迭代开发是有节奏地小步快跑，但建立在坚实的质量基础上

敏捷团队包括 3 个核心角色: PO(Product Owner)、Scrum Master(Scrum 教练)和 Team(开发产品)

敏捷团队实践：完整团队（特种兵小组）

什么是完整团队

- 敏捷开发中，以 Story 为单位的持续交付要求系统组、开发和测试等跨功能团队进行密切协同，相互独立的功能团队难以应对
- 完整团队是跨功能领域（需求分析师、设计师、开发人员、测试人员、资料人员等）的人员组成一个团队，坐在一起工作，团队成员遵循同一份计划，服从于同一个项目经理。

完整团队的好处

- 有助于团队成员形成共同目标和全局意识，促进各功能领域的拉通和融合；
- 通过面对面沟通提升沟通效率。
- 实现团队成员的高度协同，支撑高密度地、持续地、短周期的交付。

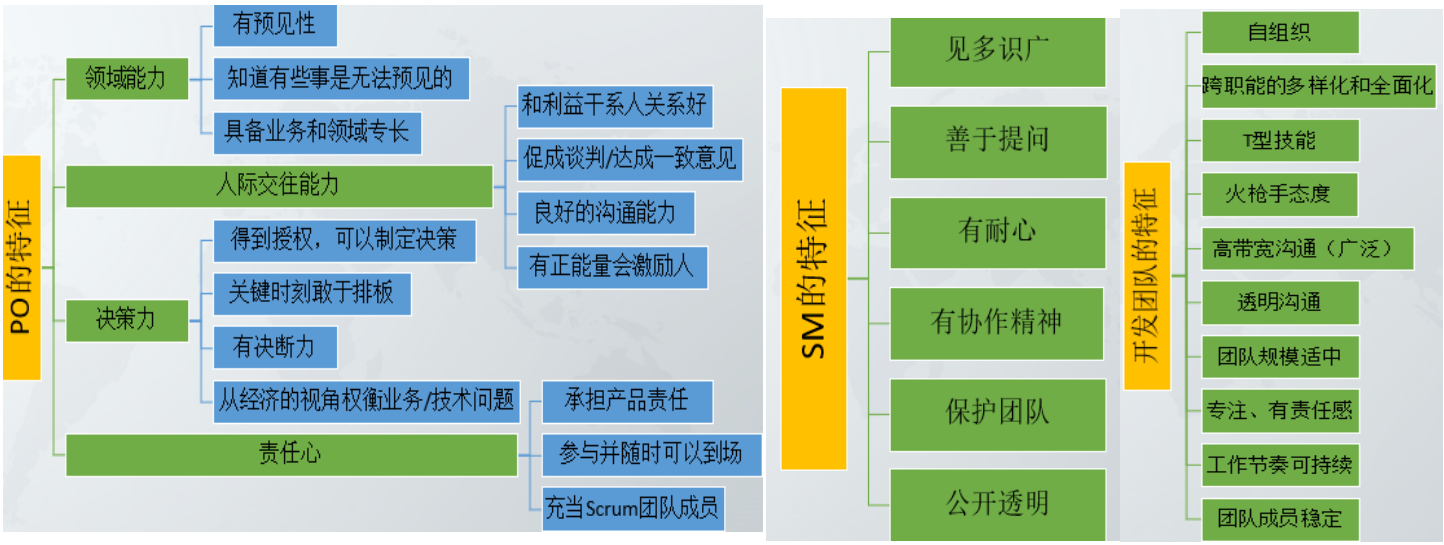
完整团队的关键要点

- 成员来自多功能领域：团队拥有完成目标所需的各职能成员；
- 坐在一起办公：团队成员无障碍地沟通；
- 团队保持相对稳定：临时组建的团队生产效率较低，团队稳定非常关键。
- T 型人才，背靠背精神：每个成员都一专多能，工作上互助互补。

完整团队聚焦客户需求交付，提高协作效率

激发团队,敏捷方式下管理者的转变:敏捷方式下对管理者最大挑战是学会放松"控制"

敏捷方式下团队成员的转变:从被动到主动心态转变是团队成员适应敏捷开发的关键



1.谁来担任 PO?

内部开发：内部业务方代表，例如为市场营销团队开发系统，那么就应该由市场营销团队中得到授权的人当 PO；

商业开发：组织内部员工，充当实际客户的代言人，通常是产品管理或营销部门成员；

外包开发：甲方安排 PO，乙方安排相应的人对接；

2.谁来担任 SM?

产品经理、项目经理、开发、测试、职业经理人、人力经理……,必须具备前面的 6 大特征，并愿意掌握 SM 的技能；

3.SM 必须全职吗？

对于成熟的 Scrum 团队，SM 可以兼任其它团队的 SM 工作；

4.Scrum 团队是否需要保持稳定？

尽最大可能稳定，成熟的 Scrum 团队非常难形成，一旦形成战斗力极强；

敏捷工作件：产品 Backlog

什么是产品 Backlog

- 经过优先级排序的动态刷新的产品需求清单，用来制定发布计划和迭代计划。

产品 Backlog 的好处

- 通过需求的动态管理应对变化，避免浪费；
- 易于优先交付对用户价值高的需求。

产品 Backlog 关键要点

- 清楚表述列表中每个需求任务对用户带来的价值，做为优先级排序的重要参考；
- 动态的需求管理而非“冻结”方式，PO 持续地管理和及时刷新需求清单，在每轮迭代前，都要重新筛选出高优先级需求进入本轮迭代；
- 迭代的需求分析过程，而非一次性分析清楚所有需求（只对近期迭代要做的需求进行详细分析，其它需求停留在粗粒度）。

ID（统一标识符）	Name（标题）–简短的、描述性的故事名	Story（故事）–故事内容描述	Priority（重要性）–产品负责人评出一个数值，指示这个故事有多重要	Initial estimate（初始估计）–团队的初步估算，表示和其他故事相比，完成该故事所需的工作量	How to demo（如何做演示）–它大略描述了这个故事应该如何在sprint 演示上进行示范	Notes（注解）–相关信息、解释说明和对其它资料的引用等等

产品 Backlog 是需求动态管理的载体

好的产品功能列表具备 DEEP 特征

敏捷工作件：迭代 Backlog

什么是迭代 Backlog

- 迭代 Backlog 是团队在一轮迭代中的“**任务**”（Task）清单，是团队的详细迭代开发计划；
- 当团队接收从产品 Backlog 挑选出要在本轮迭代实现的需求时，召开团队迭代计划会议，将需求转化为具体的“任务”；
- 每项任务信息包括当前剩余工作量和责任人。

迭代 Backlog 的好处

- 将需求分解成更细小的任务，利于对迭代内进度进行精确控制；
- 剩余工作量可用来实时跟踪团队当前进展。

迭代 Backlog 关键要点

- **“任务”**由团队成员自己分解和定义，而不是上级指派，支撑需求完成的所有工作都可以列为任务；
- 任务要落实到具体的责任人；
- 任务粒度要小，工作量大于两天的任务要进一步分解；
- 用小时做为任务剩余工作量的估计单位，并每日重估计和刷新。

敏捷工作件：完成标准

什么是完成标准

- 基于“随时可向用户发布”的目标制定衡量团队工作是否已完成的标准，由团队和 PO 形成共识；

完成标准的好处

- 共同协商的完成标准是团队的自我承诺，团队会更认真；
- 用于准确评估团队工作进展；
- 清晰和明确的完成标准保证了每次迭代是高质量的。

完成标准的关键要点

- 团队自协商：团队根据项目实际情况来定义完成标准，并严格遵守；
- 有层次：一般分为三个层次：Story 级别，迭代级和发布级，每个级别都有各自的完成标准。
- **完成标准确保团队每一步前进都奠定在坚实的质量基础之上**

敏捷管理实践：迭代计划会议

什么是迭代计划会议

- 每轮迭代启动前，团队共同讨论本轮迭代详细开发计划的过程，输入是产品 Backlog，输出是团队迭代 Backlog
- 多团队迭代计划会议要分层召开
  - 版本迭代计划会议：将产品 Backlog（需求）分配给团队
  - 团队迭代计划会议：将选取的产品 Backlog 需求转换成迭代 Backlog（任务），分配给团队成员；
- 迭代计划会议内容：
  - 澄清需求、对“完成标准”达成一致
  - 工作量估计、根据团队能力确定本轮迭代交付内容
  - 细化、分配迭代任务和初始工作计划。

迭代计划会议由团队共同确定迭代交付内容和完成标准

迭代计划会议的好处

- 通过充分讨论，使团队成员对任务和完成标准理解一致
- 团队共同参与，促进团队成员更认真对待自己的承偌。

迭代计划会议的关键要点

- 充分参与：Scrum Master 确保 PO 和 Team 充分参与讨论，达成理解一致；
- 相互承诺：Team 承诺完成迭代 Backlog 中的需求并达到”完成标准“，PO 承诺在短迭代周期不增加需求（2-4 周）；
- 确定内部任务：Team 和 PO 协商把一些内部任务放入迭代中（例如重构、持续集成环境搭建等），由 PO 考虑并与其他外部需求一起排序 。

迭代 Backlog 规划

每个迭代的周期通常为 2-4 周，对应的规划时间为 4-8 小时

参与者：

- PO：分享本迭代的目标，展示排好序的 PBI,回答团队提出的问题
- 团队：确定可交付哪些特性，在冲刺规划结束时做出承诺
- SM：观察规划活动，提出深入细节的问题，引导并帮助团队确保有成果

迭代 Backlog 规划步骤

某团队开展两周（10 工作日）的迭代，全部时间都用于执行工作吗？
1.SCRUM 活动；
2.支持、维护；
3.日常事务：会议、邮件…
4.休假!!
剩下的才是真正的生产能力

1.有迭代目标的按目标选择
2.无目标的按排好序的优先级选择
3.绝不选择一个迭代做不完的 PBI

选取 PBI 的总工作量小于团队的迭代速率，同时考虑当前迭代团队的生产能力

团队给出承诺，在本迭代周期结束时完成所选的 PBI

迭代执行包含规划、管理、执行和沟通工作，以确保创建可工作的、经过测试的特性

敏捷管理实践：每日站立会议

什么是每日站立会议

- 每日工作前，团队成员的例行沟通机制，由 Scrum Master 组织，Team 成员全体站立参加
- 聚焦在下面的三个主题：
  - 我昨天为本项目做了什么？
  - 我计划今天为本项目做什么？
  - 我需要什么帮助以更高效的工作？

每日站立会议的好处

- 增加团队凝聚力，产生积极的工作氛围
- 及时暴露风险和问题；
- 促进团队内成员的沟通和协调。

每日站立会议的关键要点

- 准时开始：按计划会议制定的时间地点开会，形成团队成员的自然习惯；
- 高效会议：会议限时 15 分钟，每个人都保持站立，依次发言，不讨论与会议三个主题无关的事情（如技术解决方案等）；
- 问题跟踪：Scrum Master 应该记录下所有的问题并跟踪解决；

每日站立会议促进团队沟通协调，及时暴露问题

敏捷管理实践：迭代评审会

什么是迭代验收

- 每次迭代开发结束时举行，通过演示可工作的软件检查需求是否满足客户要求；
- 由 Scrum Master 组织， PO 和用户代表（外部或内部利益相关人）负责验收、

Team 负责演示可工作软件。

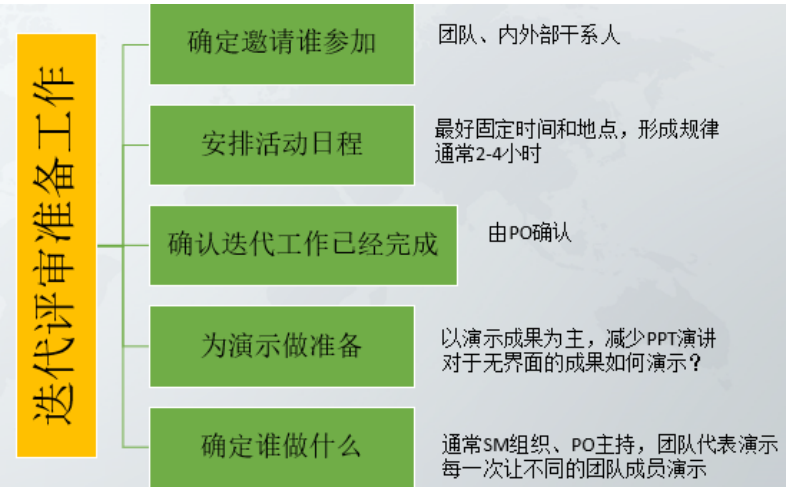
迭代验收的好处

- 通过演示可工作的软件来确认项目的进度，具有真实性；
- 能尽早的获得用户对产品的反馈，使产品更加贴近客户需求

迭代验收的关键要点

- 展示“真实”的产品：Team 应在真实环境中展示可运行的软件，判断是否达到“完成”标准；
- 收集反馈：PO 根据验收情况及客户反馈意见，及时调整产品 Backlog。

迭代验收尽早演示可工作的软件，收集反馈意见



迭代评审会的典型内容

- PO: 概要说明迭代目标中哪些完成，哪些没完成
- 成员：演示完成的增量
- 参会人：讨论产品当前状态
- 团队：调整产品未来方向

1. 如果完成的成果与目标不符，团队要给出解释。但评审的目的是通过对当前目标的探讨，确定最佳前进路线。所以不要成为互相指责的场合。
2. 如果评审时，一位级别高的干系人不同意接收产品，他认为 PBI 没有完成，怎么办？

敏捷管理实践：迭代回顾会议

什么是迭代回顾会议

- 在每轮迭代结束后举行的会议，目的是分享好的经验和发现改进点，促进团队不断进步；
- 围绕如下三个问题：
  - 本次迭代有哪些做得好
  - 本次迭代我们在哪些方面还能做得更好
  - 我们在下次迭代准备在哪些方面改进？

迭代回顾会议的好处

- 激励团队成员；
- 帮助团队挖掘优秀经验并继承；
- 避免团队犯重复的错误；
- 营造团队自主改进的氛围。

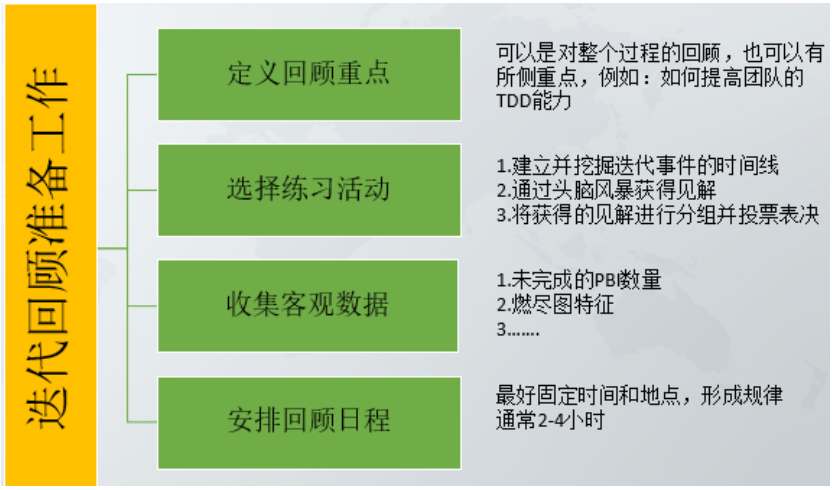
迭代回顾会议的关键要点

- 会议气氛：Team 全员参加，气氛宽松自由，畅所欲言，头脑风暴发现问题，共同

分析根因；

- 关注重点：Team 共同讨论优先级，将精力放在最需要的地方（关注几个改进就够了）；
- 会议结论要跟踪闭环：可以放入迭代 backlog 中。

迭代回顾会议是促进团队持续改进的最有效手段



迭代回顾会典型内容

敏捷工程实践：用户故事

什么是用户故事

- 用户故事是站在用户角度描述需求的一种方式；
- 每个用户故事须有对应的验收测试用例；
- 用户故事是分层分级的，在使用过程中逐步分解细化；
- 典型的描述句式为：作为一个 XXX 客户角色，我需要 XXX 功能，带来 XXX 好处。

用户故事的关键要点

- I - Independent，可独立交付给客户
- N - Negotiable，便于与客户交流
- V - Valuable ，对客户有价值
- E - Estimable ，能估计出工作量
- S - Small ，分解到最底层的用户故事粒度尽量小，至少在一个迭代中能完成
- T - Testable，可测试

用户故事的好处

- 用户故事站在用户视角便于和客户交流，准确描述客户需求；
- 用户故事可独立交付单元、规模小，适于迭代开发，以获得用户快速反馈；

- 用户故事强调编写验收测试用例作为验收标准，能促使需求分析人员准确把握需求，牵引开发人员避免过度设计。

用户故事便于团队站在用户角度分解细化需求并制定验收标准

故事样例

> **初始需求：**1.作为网络规划人员，我想要配置一个媒体网关，因为想要增加网络容量和服务
>
> **初次分解：**1.1 作为网络规划人员，我想把媒体网关参数上传到管理系统
>
> 1.2 作为网络规划人员，我想从管理系统下载媒体网关参数
>
> **再次分解：**1.2.1 作为网络规划人员，我想用文件方式从管理系统下载媒体网关参数
> 用例：用户在管理系统上选择以文件方式下载媒体网关参数，执行成功后，检查文件是否正确下载到本地且内容正确
>
> 1.2.2 作为网络规划人员，我想用 MML 结构方式从管理系统下载媒体网关的参数
> 用例:……

用户故事特征

- 体现客户（用户）价值，轻量级的点位符
- 3C 特质
  - 卡片（Card）：作为 XX，我希望 XXX，这样可以 XXX
  - 对话（Conversation）：不描述到细节，由团队通过持续对话细化，激发大家的深度理解
  - 确认（Confirmation）：有明确的验收标准

用户故事大小级别

- 史诗故事（1-2 个月）
- 特性故事（1-2 周）
- 冲刺故事（1-2 天）
- 任务（几小时）：可分工执行

用户故事 INVEST 标准

- 独立：故事之间松耦合，依赖小，否则无法进行优先级排序
- 可协商：开始仅用于做占位符，逐步细化
- 有价值：客户付费、用户使用（支付时间）
- 可估算：设计、开发、测试团队可估算工作量和成本
  - 不可估算的原因：太大需要分解，或信息不全需要进一步探索
- 大小合适：冲刺中的故事应足够小（例如两周冲刺，故事就应该是 2 天以内的）
- 可测试：有相应测试验收标准（史诗级的可以没有）

敏捷工程实践：结对编程

什么是结对编程

- 两位程序员在一台电脑前工作，一个负责敲入代码，而另外一个实时检视每一行敲入的代码；
- 操作键盘和鼠标的程序员被称为“驾驶员”，负责实时评审和协助的程序员被称为“领航员”；
- 领航员检视的同时还必须负责考虑下一步的工作方向 ，比如可能出现的问题以及改进等。

结对编程的关键要点

- 程序员应经常性地在“驾驶员”和“领航员”间切换，保持成员间平等协商和相互理解，避免出现一个角色支配另一个角色的现象；
- 开始一个新 Story 开发的时候即可变换搭档，以增进知识传播；
- 培养团队成员积极、主动、开放、协作的心态能够增进结对编程效果；
- 实施初期需要精心辅导，帮助团队成员克服个性冲突和习惯差异。

结对编程的好处

- 有助于提升代码设计质量；
- 研究表明结对生产率比两个单人总和低 15%，但缺陷数少 15%，考虑修改缺陷工作量和时间都比初始编程大几倍，所以结对编程总体效率更高（*source: The Economist*）；
- 结对编程能够大幅促进团队能力提升和知识传播。

结对编程提高代码质量和工作效率

敏捷工程实践：测试驱动开发（TDD）

什么是测试驱动开发

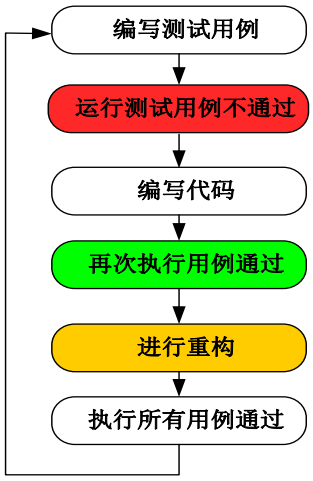
- TDD 以测试作为编程的中心，它要求在编写任何代码之前，首先编写定义代码功能的测试用例，编写的代码要通过用例，并不断进行重构优化；
- TDD 要求测试可以完全自动化运行。

测试驱动开发的关键要点

- 测试代码和源代码一样都需要简洁，可读性好；
- 测试用例的设计要保证完备，覆盖被测单元的所有功能；
- 每个测试用例尽量保持独立，减少依赖，提高用例的可维护性；
- 当功能单元较大时，为降低难度，可分解为多个更小的功能单元，并逐一用 TDD 实现。

测试驱动开发的好处

- 和代码同步增长的自动化测试用例，能为代码构筑安全网，保证代码重构的质量；
- TDD 有助于开发人员优化代码设计，提高代码可测试性



敏捷工程实践：持续集成(CI)

什么是持续集成

- 持续集成（CI）是一项软件开发实践，其中团队的成员经常集成他们的工作，通常每人每天至少集成一次，每次集成通过自动化构建完成。

持续集成的好处

- 大幅缩短反馈周期，实时反映产品真实质量状态；
- 缺陷在引入的当天就被发现并解决,降低缺陷修改成本；
- 将集成工作分散在平时，通过每天生成可部署的软件；避免产品最终集成时爆发大量问题。

持续集成的关键要点

- 持续集成强调 “快速”和“反馈”，要求完成一次系统集成的时间尽量短，并提供完备且有效的反馈信息；
- 自动化测试用例的完备性和有效性是持续集成质量保障；
- 修复失败的构建是团队最高优先级的任务；
- 开发人员须先在本地构建成功，才可提交代码到配置库 ；
- 持续集成的状态必须实时可视化显示给所有人；
- 大系统持续集成需分层分级，建立各层次统一的测试策略。

敏捷工程实践：Code Review

目的：持续的提高开发团队的工作质量

好处：

- 代码复查者(reviewer)能从他们的角度来发现问题并且提出更好的解决方案。
- 确保至少你团队的另一个其他成员熟悉你的代码，通过给新员工看有经验的开发者的代码能够某种程度上提高他们的水平。
- 公开 reviewer 和被复查者的想法和经验能够促进团队间的知识的分享。
- 能够鼓励开发者将他们的工作进行的更彻底，因为他们知道他们的代码将被其他的人阅读。

在项目开发的过程中，25%的时间应该花费在 code review 上