COMP130015.02
软件工程

2. 软件过程

复旦大学计算机科学技术学院
沈立炜
shenliwei@fudan.edu.cn

软件开发流程

西乔：漫画《神秘的程序员们》

软件过程的含义

- 软件过程定义了软件组织和人员在软件产品的定义、开发和维护等阶段所实施的一系列活动（Activity）和任务（Task）
  - ✓ 例如，实现过程包括准备实现、实施实现以及管理实现的结果三个活动
  - ✓ 准备实现这个活动包括定义实现策略、识别实现约束、识别必要的和清晰的软件环境（比如开发或测试环境）并为之做计划、获得对软件环境或服务的访问权等具体任务
- 软件过程的三层含义
  - ✓ 个体：软件产品或系统在生存周期中的某一类活动的集合，如系统分析过程、实现过程
  - ✓ 整体：软件产品或系统在所有上述含义下的软件过程的总体
  - ✓ 工程：应用软件工程原则、方法来构造软件过程模型，并应用到软件产品的生产中，以此来提升软件生产率，降低生产成本

类比：饮料生产流程

由一系列步骤和顺序组成

工艺流程

火棘原汁的提取工艺流程

采果→挑选→清洗→软化打浆→酶解→分离、澄清

过滤→杀菌→冷却→原汁保存

火棘果汁饮料的加工工艺流程

类比：饮料生产流程

有一些关键质量控制点及相应的手段和措施

火棘饮料生产中 HACCP 体系的关键控制点及控制方法

Critical points and controlling methods of *Pyracantha fortuneana* beverage in HACCP system

工艺环节	关键控制点	控制范围	监测方法	控制手段	校正措施
原料	成熟度、新鲜度、虫果及腐烂率	色泽鲜艳，充分成熟，无病虫腐烂果，干缩果及枝叶杂质	感官检验	杜绝收购未成熟、质量差的原料	有选择地进行原料收购
软化打浆、酶解	料水比、酶用量和酶解时间	料水比 1∶2，酶用量 0.2%，6h	量具和时钟测量，酒精实验	严格按工艺参数进行操作	根据酒精实验结果对酶用量及酶解时间进行微调
分离、澄清、过滤	设备卫生状况、明胶用量、络合时间	设备干净卫生，明胶配成 3%～5% 的溶液，络合 20～24h	量具和时钟测量，感官检验	设备定期进行 CIP 清洗、严格按工艺参数进行操作	根据每批小试结果调整明胶用量及络合时间
杀菌、贮存	设备、贮桶卫生状况，灭菌温度及时间，原汁贮存温度及贮存时间	设备、贮桶干净卫生，115 ± 2℃下保持 3～5s，阴凉干燥处存放	感官检验，微生物分析检测	设备定期进行 CIP 清洗、严格按工艺参数进行操作	根据微生物分析检测结果调整工艺参数
配料	原辅料质量	符合相关卫生质量标准	原汁进行理化及卫生质量检测	使用符合工艺要求的原辅材料，严格控制辅助材料的进货渠道	有选择性地进行辅料购买
过滤、灭菌	设备卫生状况，灭菌温度及时间	设备干净卫生，120 ± 2℃下保持 3～5s	感官检验，微生物分析检测	设备定期进行 CIP 清洗，严格按工艺参数进行操作	根据分析检测结果调整工艺参数
罐装、封口	设备及包装物卫生状况	设备干净卫生，70～80℃热灌装		设备定期进行 CIP 清洗，严格按工艺参数进行操作	根据分析检测结果调整工艺参数

HACCP（hazard analysis and critical control point）
是国际上共同认可和接受的食品安全保证体系

课堂讨论：规范化过程

课堂讨论：为什么人们相信规范化的过程（例如符合国际标准）能够保证产品（例如饮料）的质量？

软件过程发展历史-1

- 20世纪50年代：包含在硬件工程中的编码、测试等软件开发相关活动
- 20世纪60年代：形成了“软件工艺”的概念，以一种黑盒的方式存在（Code-and-Fix）
  - ✓ 利用软件易于修改的特点，先编写代码然后修复错误
  - ✓ 整个过程对客户和开发人员自身都是不可见和不可控的
  - ✓ 难以评估当前进度，开发过程就像一个难以预测的黑盒
- 20世纪70年代：在IBM 360等大型项目经验基础上逐渐形成规范化软件过程的思想
  - ✓ 瀑布模型等过程模型出现，通过固化的流程定义开发过程
  - ✓ 通过反馈了解开发进度和效果，开发过程更易于管理控制

软件过程发展历史-2

- 20世纪90年代：“软件过程也是软件”以及开发优秀软件过程的重要性得到广泛认同
  - ✓ 美国卡耐基梅隆大学软件工程研究所（CMU SEI）提出软件开发能力成熟度模型CMM，用于评估和改进软件过程
  - ✓ 到2000年发展为CMMI（能力成熟度模型集成）
- 21世纪00年代：敏捷开发的思想成为主流
  - ✓ 人们在逐渐加快的开发和交付节奏中，发现传统的软件过程过于强调计划，往往难以很好地适应软件开发中的变化
  - ✓ 出现了多种具有快速迭代反馈、适应需求变化等“轻量级”特点的开发方法，被称为敏捷方法
- 21世纪10年代：开发运维一体化（DevOps）
  - ✓ 由于互联网的普及，大量的软件开发需求朝着快速、易变的方向发展，敏捷的思想和实践进入运维领域
  - ✓ 打破开发和运维的界限，实现从运维到开发的快速反馈

ISO/IEC 12207: 2017软件生存周期过程

软件过程模型（软件开发模型）

- 对开发人员所采用的软件开发方法与过程组织整体结构的抽象描述，表达了软件过程的结构框架
- 在一定抽象层次上刻画了一类软件过程的共性结构和属性
- 不同的软件过程模型阐述了不同的软件开发指导思想、方法步骤以及具体实践

经典软件过程模型

- 瀑布模型
- 增量模型
- 演化模型
- 统一过程模型

瀑布模型

- 核心思想是将软件开发的各个过程以线性的、顺序的方式进行
  - ✓ 首先，清晰地了解要解决问题的需求
  - ✓ 然后顺序地开展策划、建模、构建和部署等工作
  - ✓ 最终交付完整产品，并开展后续的技术支持和维护
- 反映了一种软件开发过程理想
  - ✓ 每个阶段的具体活动内容和目标都清晰定义
  - ✓ 阶段性的产出都能被明确签核（sign-off）
  - ✓ 前一个阶段完成后才会进入下一个阶段
- 相对于早期软件过程定义不清晰、缺乏工程化的开发方法而言是一大进步

像瀑布一样
一次性流过

瀑布模型的主要问题

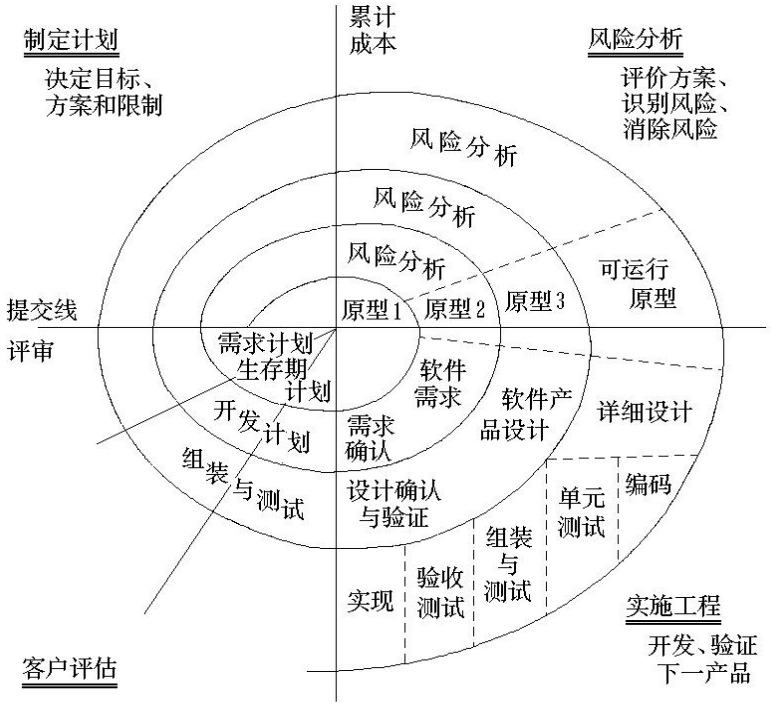
- 实际项目很少完全遵循瀑布模型提出的顺序
- 客户难以在一开始就完整、准确地描述需求
- 可执行的软件产品交付太晚，造成巨大的风险

增量模型

- 将开发过程分成若干增量，每次增量选择一部分需求（最关键、最核心的需求）作为交付目标，产出一个可执行的中间产品，例如
  - ✓ 增量1：支持文本文件保存、打开和编辑操作
  - ✓ 增量2：支持字体、颜色等文本格式设定
  - ✓ 增量3：支持自动拼写检查
- 每个增量产出的中间产品可以提供给客户或用户，用于发现问题或者潜在的新需求
- 客户反馈比瀑布模型更加及时，但缺少针对每次增量的时间和范围的指导建议

以迭代的方式使用瀑布模型

演化模型



- 通过“迭代”来应对不断演变的需求
- 典型代表是Barry Boehm提出的螺旋模型：通过周期性的迭代将各个阶段以螺旋扩增的方式进行编排
- 主要问题：难以判断螺旋演进何时结束

螺旋模型：风险驱动的周期性迭代

演化模型

原型（prototype）开发：
客户与开发团队之间有效沟通的重要技术手段

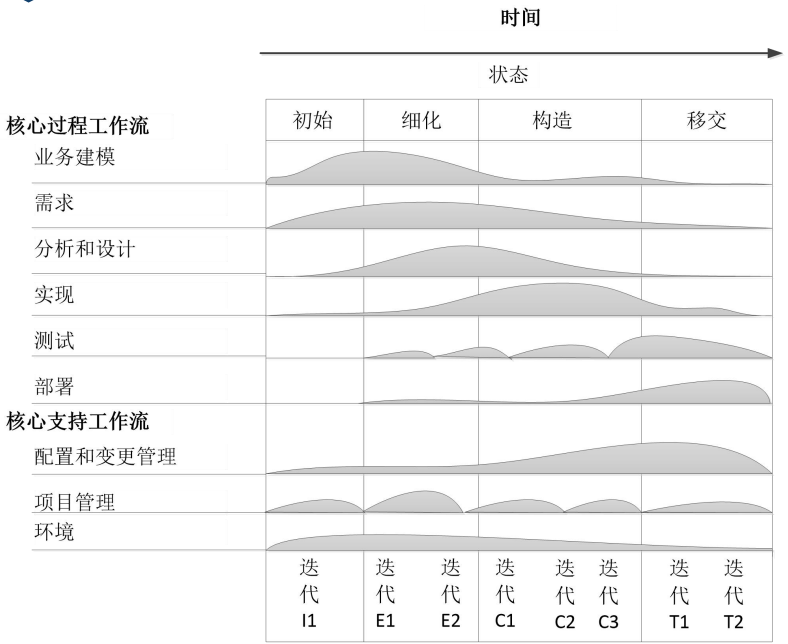
抛弃式原型

- ✓ 对某些特定问题的探索和实验
- ✓ 具有明确的目的性，不宜集成到正式产品中
- ✓ 在完成原型、达到目的后，应当被抛弃

增量式原型

- ✓ 目标系统的一个可运行的子集，关键设计决策应该在增量构建过程中逐步得到确定
- ✓ 确保后续的增量都建立在一个合适的基础上

统一过程模型（Unified Process）



- 增量、迭代的过程流，每个迭代涉及核心过程工作流和核心支持工作流中的多个过程
- 不同阶段迭代工作流中的过程工作量是不同的

增量迭代、用例驱动、以体系结构为中心

课堂讨论：软件开发项目的过程组织

课堂讨论：假设你们小组正在开发一款智能家居系统，那么你们组的开发过程大致将以什么样的方式组织？

软件过程改进

- 如同建筑企业的资质等级，软件项目发包方也希望软件企业有明确的能力等级
- 软件组织的“过程成熟度”：反映影响最终软件产品质量的因素
  - ✓ 软件过程本身的质量
  - ✓ 团队人员对过程理解和应用的程度
  - ✓ 软件工程实践的总体执行情况
- 通过评估寻找过程能力的改进空间，例如
  - ✓ 团队现状：完全不重视代码的版本管理，没有一个统一的配置管理工具
  - ✓ 改进措施：引入Git版本管理系统、制定代码版本管理规范等

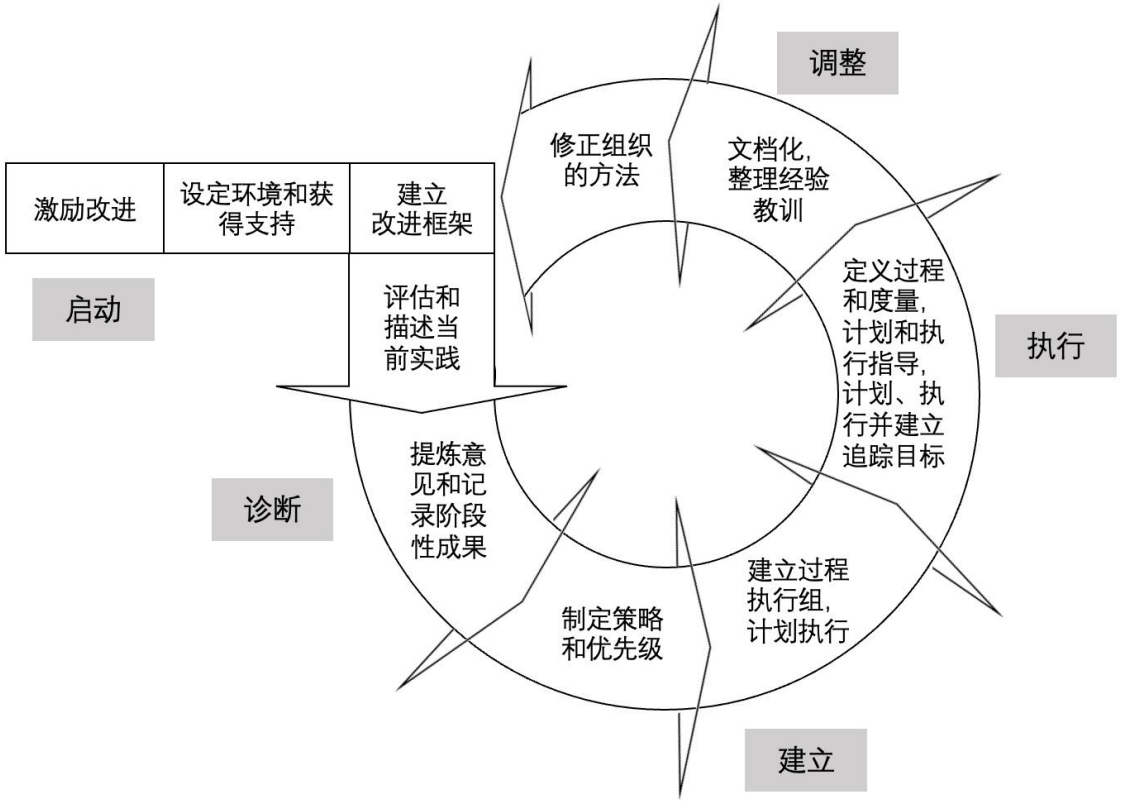
CMM与CMMI

- 上世纪80年代，美国卡内基·梅隆大学软件工程研究所（CMU/SEI）受国防部委托开始研制软件供应商能力评估模型
- 首先推出CMM（Capacity Maturity Model），在此基础上集成相关模型逐步形成CMMI
- 五个成熟度等级：初始级、已管理级、已定义级、量化管理级、优化级
- 定义了22个过程域，每个成熟度等级中包含一组相关的过程域（表示需要具备相应的能力）
- 每个过程域内部定义了不同的能力等级（0-3）

CMMI的成熟度等级和过程域

CMMI等级	过程域（中文）	过程域（英文）	过程类型
第2级 已管理级	需求管理	Requirements Management	工程
	项目规划	Project Planning	项目管理
	项目监控	Project Monitoring and Control	项目管理
	供应商协议管理	Supplier Agreement Management	项目管理
	度量分析	Measurement and Analysis	支持
	过程和产品质量保证	Process and Product Quality Assurance	支持
	配置管理	Configuration Management	支持
第3级 已定义级	需求开发	Requirements Development	工程
	技术方案	Technical Solution	工程
	产品集成	Product Integration	工程
	验证	Verification	工程
	确认	Validation	工程
	组织过程焦点	Organizational Process Focus	过程管理
	组织过程定义	Organizational Process Definition	过程管理
	组织培训	Organizational Training	过程管理
	集成化项目管理	Integrated Project Management	项目管理
	风险管理	Risk Management	项目管理
	决策分析与解决方案	Decision Analysis and Resolution	支持
第4级 量化管理级	组织过程绩效	Organizational Process Performance	过程管理
	定量项目管理	Quantitative Project Management	项目管理
第5级 优化级	组织革新与推广	Organizational Innovation and Deployment	过程管理
	原因分析与解决方案	Causal Analysis and Resolution	支持

IDEAL软件过程改进模型



用于启动、策划和实施改进活动

软件过程：计划驱动 vs. 敏捷

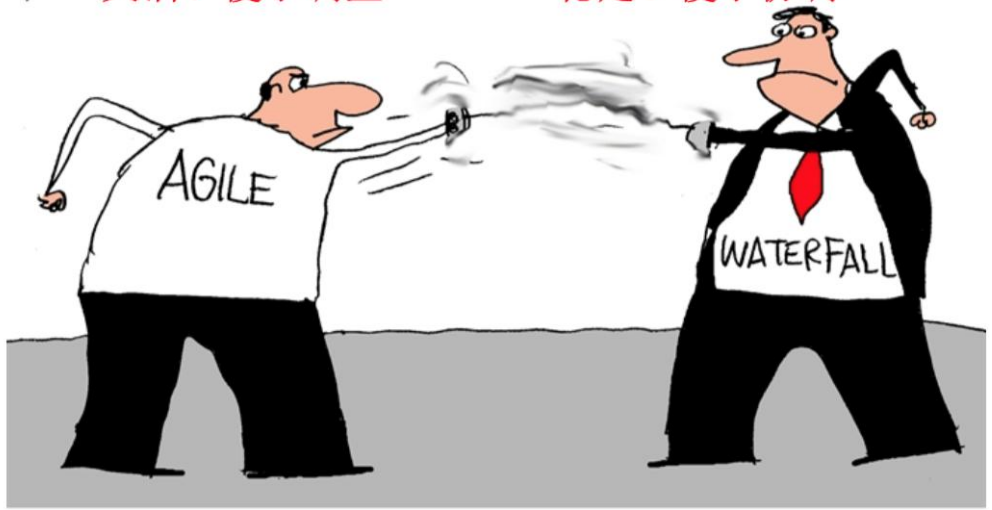
适应变化
快速迭代
个体与交互
可运行的代码

敏捷 ⟷ 计划

遵循计划
按部就班
过程与工具
详尽的文档

充分考虑软件的特点
灵活、便于调整

传统工程化思维
稳定、便于协调



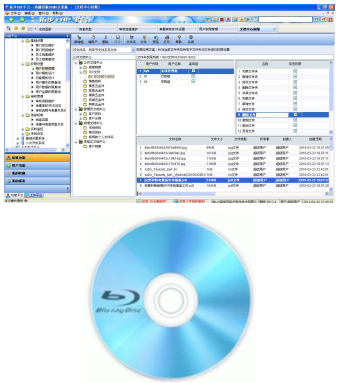
The ways of working out our differences took a sudden dangerous turn.

计划驱动与敏捷：适用条件与优缺点

- 计划驱动的过程：所有过程活动均事先计划，按照计划衡量进度
  - ✓ 需求可以事先清楚、完整定义，且不会有太大变化
  - ✓ 最终得到的软件系统有较高的可靠性、安全性要求
  - ✓ 涉及多个部门或团队，涉及复杂的协调和沟通
- 敏捷开发：只做增量的短期计划并根据变化和反馈不断进行调整
  - ✓ 需求不确定且经常变化（变化蕴涵着竞争优势）
  - ✓ 对于最终软件系统的可靠性和安全性要求没那么高
  - ✓ 开发团队规模较小且能够充分沟通

在实践中二者经常会有所结合

敏捷产生的背景



大型机时代
（科学计算、军工系统等）

应用面窄，需求确定，稳定性要求高，更新慢

桌面计算时代
（信息管理系统、办公软件等）

应用面有所拓展，需求阶段性发生变化，通过光盘等发布载体更新

互联网时代
（电子商务系统、社交与支付App等）

应用直面消费者，快速更新且竞争激烈，快速变化、随时更新

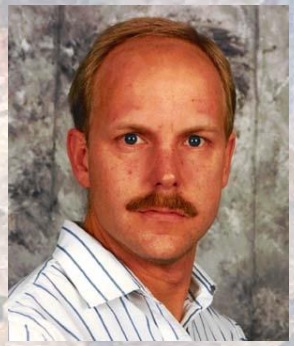
敏捷方法的探索

- 从20世纪90年代开始，业界开始探索一些不同于计划驱动的重量级方法的所谓敏捷开发方法
  - ✓ 极限编程（eXtreme Programming，XP）
  - ✓ Crystal
  - ✓ ASD（Adaptive Software Development）
  - ✓ DSDM（Dynamic System Development Method）
- 共同特点：拥抱变化
  - ✓ 认同软件开发中的变化
  - ✓ 采用各种技术手段来快速响应这些变化

敏捷宣言

“我们正通过亲身或者协助他人进行软件开发实践来探索更好的软件开发方法。基于此，我们建立了如下的价值观：

- 个体和交互 重于 过程和工具
- 工作的软件 重于 详尽的文档
- 客户合作 重于 合同谈判
- 响应变化 重于 遵循计划



也就是说，尽管右项有其价值，我们更重视左项的价值” *-- Kent Beck et al.*

2001年2月11日至13日，在犹他州瓦萨奇山雪鸟滑雪场的小屋里，17个人聚在一起聊天、滑雪、休闲，试图寻求共识，当然还在一起吃饭。产出的结果是《敏捷软件开发宣言》。来自Extreme Programming（XP）、SCRUM、DSDM、Adaptive Software Development（ASD）、Crystal、Feature-Driven Development（FDD）、Pragmatic Programming，以及其他主张替代文档驱动的重量级软件开发过程的人士聚集在一起。

敏捷过程的特点

- 客户需求（场景、故事）驱动
- 认识到有效的计划都是短期的
- 迭代地开发软件同时强调构造活动（设计与实现交织）
- 频繁交付可运行的“软件增量”
- 根据变化进行适应性调整

强调适应而非预测

敏捷开发的12条原则-1

- 我们的最高优先级是持续不断地、及早地交付有价值的软件来使客户满意
- 拥抱变化，即使是在项目开发的后期，愿意为了客户的竞争优势而采取变化
- 经常地交付可工作的软件，相隔几星期或一两个月，倾向于采取较短的周期
- 业务人员和开发人员在整个过程中紧密合作
- 围绕着被激励的个体构建项目，为个体提供所需的环境和支持，给予信任，从而达成目标
- 在团队内和团队间沟通信息的最有效和最高效的方式是面对面的交流

敏捷开发的12条原则-2

- 可工作的软件是衡量进度的首要标准
- 倡导可持续开发，项目发起者、开发人员和用户应该维持一个可持续的步调
- 持续地追求技术卓越和良好设计，可以提高敏捷性
- 以简洁为本，它是减少不必要工作的艺术
- 最好的体系结构、需求和设计是从自组织的团队中涌现出来的
- 团队定期地反思如何变得更加高效，并相应地调整自身的行为

课堂讨论：计划驱动的开发与敏捷开发

课堂讨论：什么样的项目适合于采用计划驱动的开发方法？什么样的项目适合于采用敏捷开发方法？

精益（Lean）思想

- 起源于汽车制造业
  - ✓ 当时传统生产方式遵循规模化的思路，增大产量才能分摊和降低成本
  - ✓ 但实际情况是，增加的产出除了消耗更多原材料、能源、人工以外，并没有产生实际的价值
  - ✓ 与规模化扩张相反，精益思想认为，应该从是否增加了价值的角度来评价一个活动是否是浪费
- 直观理解：消除浪费，做有价值的工作
- 核心思想：在于对系统和开发过程的“持续改善”以及“尊重人”

精益的改进原则

- 识别价值：过程中的一个活动是否有意义，取决于能不能清晰地描述出该活动对客户的价值
- 定义价值流：只有增值的活动才是必要的，否则就属于浪费，应该予以改进或消除
- 保持价值流的流动：等待、拥塞，都是价值流动的最大敌人，需要尽快发现这样的问题并加以干预消除
- 拉动系统：基于客户的需求逐级触发生产环节，即由需求拉动开发
- 持续改善：通过不断改善上述四个方面让价值的产出最大化

敏捷思想 vs. 精益思想

敏捷思想

可持续的开发步调

定期反思如何变得更加高效

产出更有针对性
价值交付更加密集

精益思想

避免超负荷工作

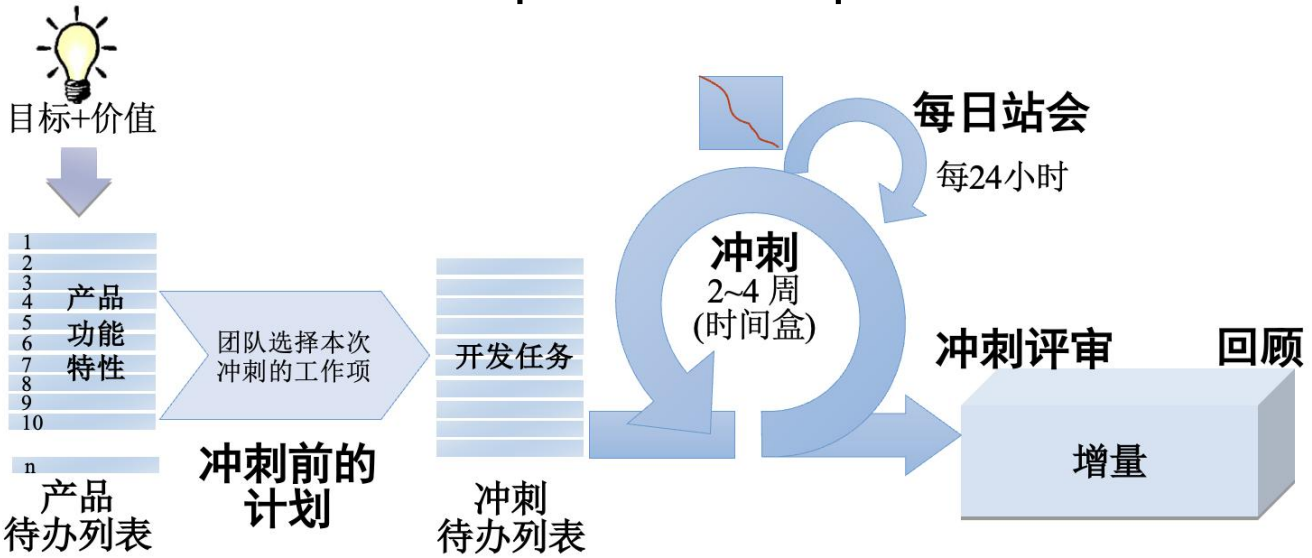
系统和过程的持续改善

同等价值观

精益思想的指引

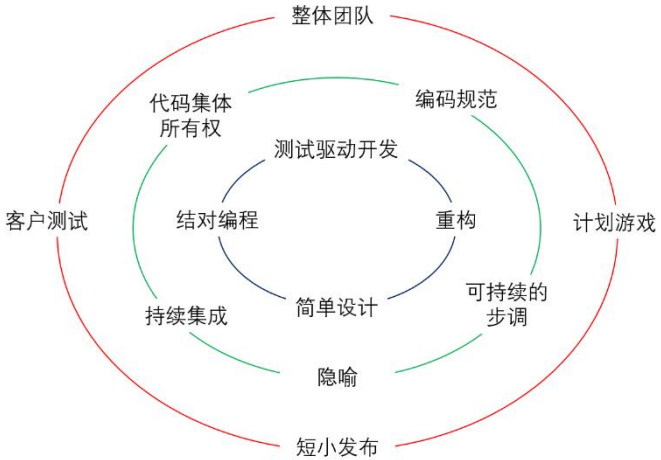
敏捷实践方法论：Scrum

- 价值观：承诺、聚焦、开放、尊重、勇气
- 主要特征
  - ✓ 基于时间盒（Time Box）的迭代
  - ✓ 增量以及演进式开发（Adaptive Development）



敏捷实践方法论：XP（极限编程）

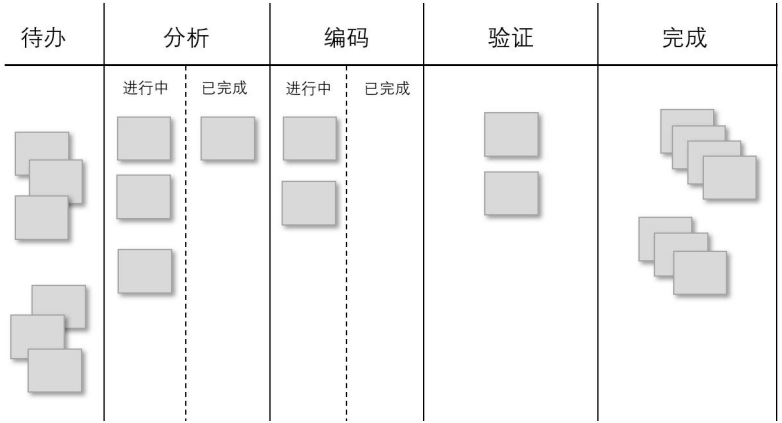
- 价值观：沟通、简单、反馈、勇气、尊重
- 原则：人性化、经济性、多赢、自相似、改善、多样性、反省、机遇、流动、失败、冗余、质量、小步骤和接受责任等



✓ 个人实践
✓ 团队实践
✓ 组织级实践

敏捷实践方法论：看板方法（Kanban）

- 精益思想的具体实践：将表示软件开发任务的卡片或贴纸放在“看板墙”上用于表示开发进度，让开发流程变得可见
- 解决软件开发与传统生产类似的过程不顺畅、工作拥塞、部门之间相互等待等问题
- 通过限制进行中的工作的数量，形成一个以拉动系统为核心的机制，暴露系统中的问题，激发协作来改善系统



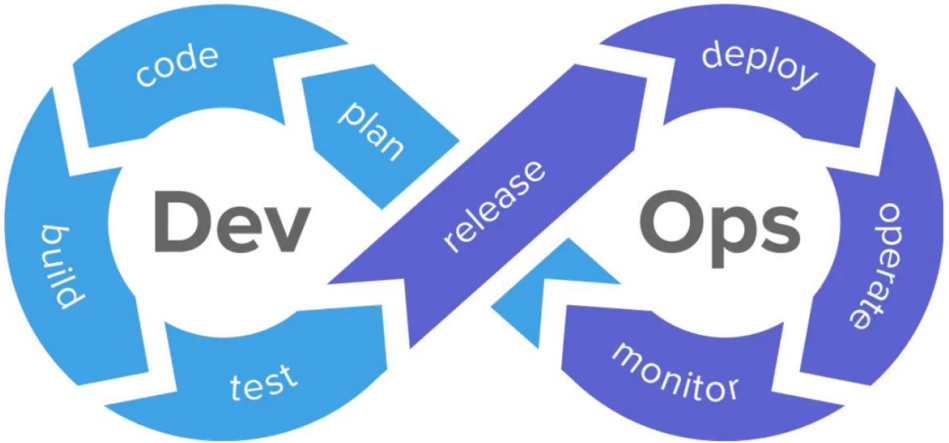
开发运维一体化（DevOps）

- 运维（Operation）工作：部署以及运行过程中的监控、异常处理、优化调整等
- 软件开发环节的快速迭代使得运维环节的效率和响应性成为主要瓶颈
- DevOps将敏捷的精神延伸到运维阶段，包含贯穿软件开发和运维的一系列实践集合
- 倡导以一种安全、快速、持续的方式将代码变更部署到生产环境中

DevOps实践

快速迭代的增量开发
持续的自动化测试
持续集成

频繁部署
持续的质量和性能监控
快速的反馈和改进机制



DevOps技术价值流与技术实践

- DevOps的技术价值流
  - ✓ 将业务构想转化为向客户交付价值的、由技术驱动的服务所需要的流程
  - ✓ 开发人员能快速地完成开发、集成、验证，并将代码更新部署到生产环境中，从而有效地将前置时间（从开发任务产生到交付相应的价值的这段时间）缩短到小时直至分钟级别
- DevOps的技术实践
  - ✓ 实现价值从开发到运维的快速**流动**
  - ✓ 建立从运维到开发的持续、快速的**反馈**机制
  - ✓ 建立**持续的实验和学习**的企业文化

持续集成

- Continuous Integration，简称CI
- 持续集成实践
  - ✓ 开发人员频繁地（一天多次）将代码变更提交合并到中央存储库，并自动运行构建和执行单元测试，从而确保新代码可以和原有代码正确地集成在一起
  - ✓ 如果持续集成失败，开发团队就要停下手中的工作，立即修复它，直到持续集成成功
- 主要目标
  - ✓ 使正在开发的软件始终处于可工作状态
  - ✓ 更快地发现、定位和解决错误
  - ✓ 提高软件质量，减少验证和发布新软件所需的时间

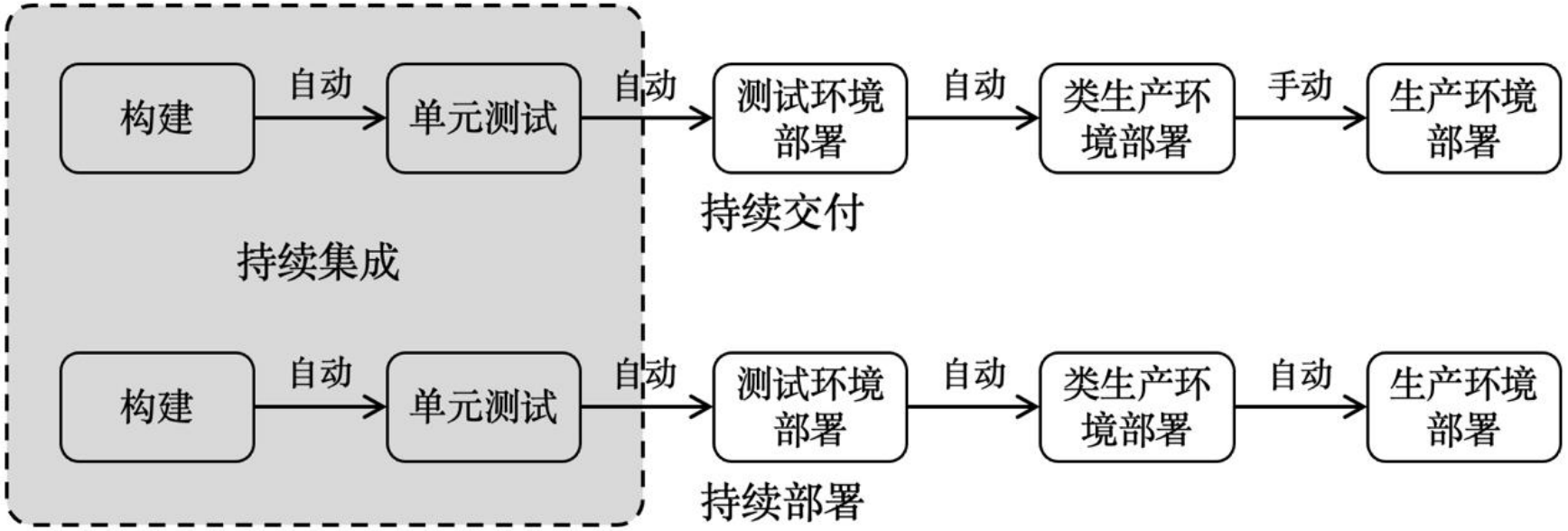
持续交付

- Continuous Delivery，简称CD
- 任何代码变更提交后都能够
  - ✓ 自动运行构建和执行单元测试
  - ✓ 自动将所有代码变更部署到测试环境和类生产环境
  - ✓ 确保当代码变更部署到生产环境后可以正常工作，从而以可持续的方式快速向客户交付新的代码变更
- 如果代码没有问题，可以继续手工部署到生产环境中
- 让正在开发的软件始终处于可部署状态同时实现快速交付，能够应对业务需求，并更快地实现软件价值

持续部署

- Continuous Deployment，简称CD
- 任何代码变更提交后都能够
  - ✓ 自动运行构建和执行单元测试
  - ✓ 自动将所有代码变更部署到测试环境、类生产环境以及生产环境
- 实现从代码变更提交到生产环境部署的全流程自动化而无需人工干预
- 加快代码提交到功能部署的速度，并能快速地收集真实用户的反馈

持续集成、持续交付与持续部署



部署流水线：将软件从开发完成到最终交付到用户手中的端到端的过程

课堂讨论：从敏捷到开发运维一体化

课堂讨论：从敏捷到开发运维一体化体现了现代软件开发什么样的特点？

本章小节

- 软件过程思想和实践的出现是软件开发进入工程化阶段的一个重要标志
- 软件过程规范和模型随着软件开发实践不断发展和演变
  - ✓ 瀑布模型传统的重量级过程
  - ✓ 新兴的轻量级过程模型
- 敏捷方法和精益思想给软件开发带来新的活力
- 敏捷进一步扩展到软件部署和运维，形成开发运维一体化（DevOps）的思想
- 持续集成、持续交付和持续部署支持了现代软件开发的高质量和高效交付

COMP130015.02
软件工程

End

2. 软件过程