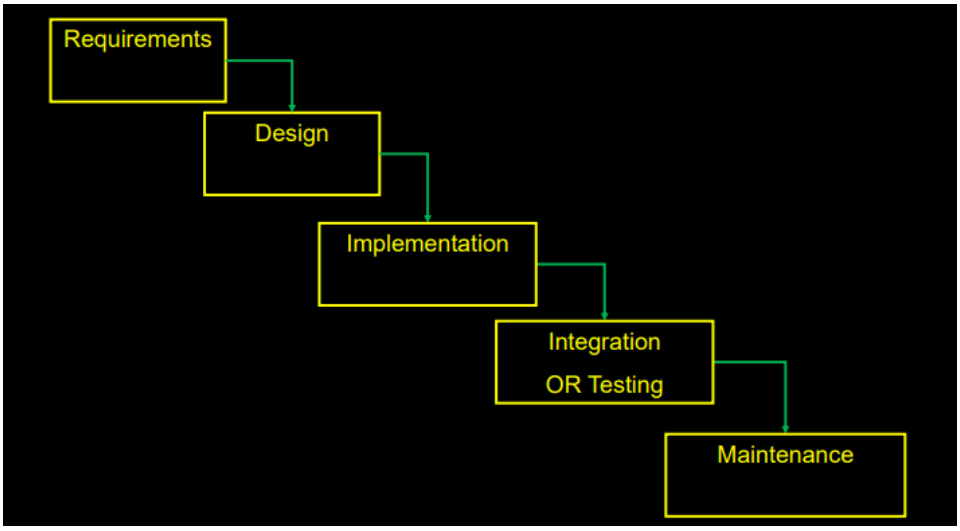
Lecture3 Extreme Programming

1. 极限编程介绍 XP

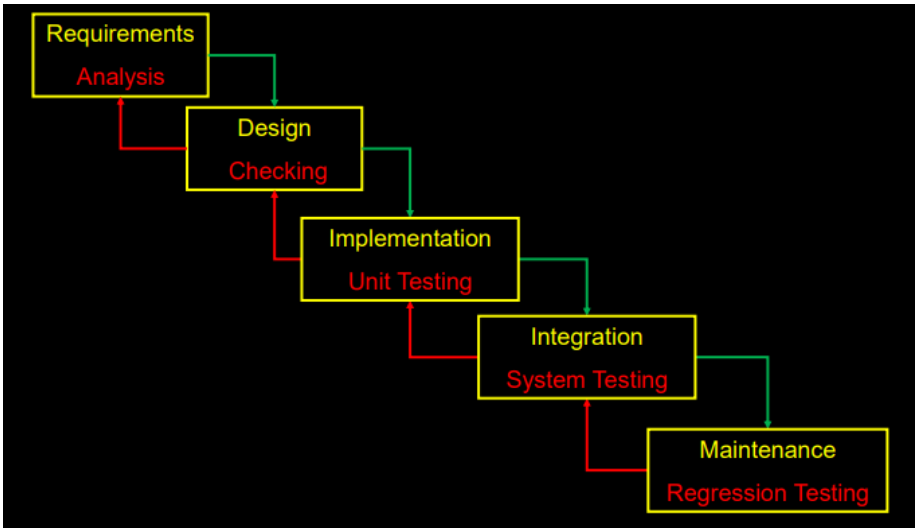
瀑布模型 Waterfall Process

模型架构



- **需求 Requirements**：软件应该做什么
- **设计 Design**：将代码构建成模块，架构
- **实现 Implementation**：编码
- **继承 Integration**：把各个模块放在一起
- **测试 Testing**：检查代码是否工作
    - 上面两个步骤经常被合并到一起，称为**验证 Verification**
- **维护 Maintenance**：持续做出改变

真实情况下的瀑布模型



极限编程 Extreme Programming

> XP 的创造者是谁?
>
> - KENT BACK

与严格的瀑布流程截然不同

- 用**协作 collaborative** 和**迭代 iterative** 设计过程代替它

核心思想

- 不写很多文档
  - 代码和测试是主要编写的产品
- 一个一个地实现需求
- 频繁的发布代码
- 与客户紧密合作
- 与团队伙伴进行大量交流

关键的实践

- Planning game 为需求规划一种游戏
- Test-driven development 设计和测试的测试驱动开发
- Refactoring 对设计进行重构
- Pair programming 结对编程
- Continuous integration 持续集成

XP 是一种迭代的过程

- Iteration = 以 1-3 周为一个循环
- 在迭代开始时，召开的**迭代会议**，**计划 plan**每一次迭代
- 迭代将实现一组**用户故事 user story**
- 把工作分成**小任务**，小到可以在一天内完成
- 每天，程序员们**结对 pair programming**完成任务

2. 结对编程 Pair Programming

结对编程是一个简单、直接的概念。两名程序员在**一台**计算机上**并肩工作**，在**相同**的设计、算法、代码和测试上持续协作，它允许两个人产生比他们单独努力总和产生的**更高质量**的代码

结对编程的两个人分成两个角色

- **驾驶员 Driver**：写代码
- **导航员 Navigator**：观察者（寻找战术和战略缺陷）
    - 定期切换驾驶员和导航员的角色

结对写代码、设计、Debug、测试等

研究成果

- 来自业界的有力证据
    - 我们可以用不到一半的时间生成近乎无缺陷的代码
- 经验研究
    - 结对编程产生**更高质量的代码**
        - 减少15%的缺陷
    - 结对编程用**大约一半的时间**完成了任务
        - 预估时间的 58%
    - 结对编程程序员是**快乐的**程序员
        - 成对的人更喜欢他们的工作（92%）
        - 成对的人对自己的工作成果更有信心（96%）

预期收益

- 更高的产出质量
- 改善了循环时间
- 增加了程序员的满意度
- 增强学习积极性
- 结对编程的人员切换
    - 简化员工培训和过渡
    - 知识管理/降低产品风险
    - 加强团队建设

好的结对编程示例

> https://www.newyorker.com/magazine/2018/12/10/the-friendship-that-made-google-huge

杰夫·迪恩(Jeff Dean)和桑杰·格玛沃特(Sanjay Ghemawat)，谷歌仅有的两位资深研究员

News Culture Books Business & Tech Humor Cartoons Magazine Video Podcasts Archive Going

THE FRIENDSHIP THAT MADE GOOGLE HUGE

Coding together at the same computer, Jeff Dean and Sanjay Ghemawat changed the course of the company—and the Internet.

By James Somers

结对编程的问题

同伴问题

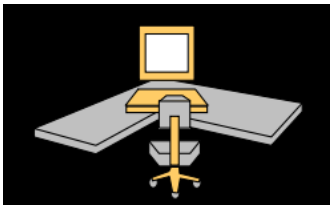
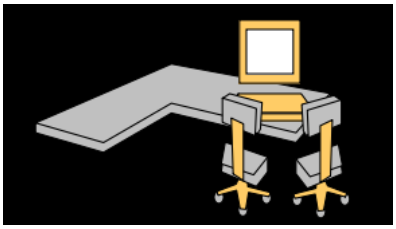
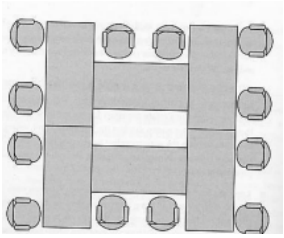
当两个专家结对编程，效果是好的

但是如果是下面的情况

情况	示例图
专家与新手配对	
新手与新手配对	
专业问题	
文化问题	

工作环境布局

工作环境布局也是问题

		
Bad	Better	Best

结对编程是如何起作用的

- **同伴压力 Pair Pressure**
  - 让彼此专注于工作
  - 不想让搭档失望
  - 不按规定程序来写会“不好意思”
- **同伴协商 Pair Negotiation**
  - 有共同的目标和计划
  - 将之前的不同经历带到任务中
  - 不同任务相关信息的获取方式
  - 必须协商采取共同行动
- **同伴鼓励 Pair Courage**
  - “如果我觉得对，你也觉得对——你猜怎么着？这可能是正确的!”
- **同伴复查 Pair Reviews**
  - 四个眼球比两个好
  - 持续的设计和代码审查
  - 消除程序员对审查的厌恶
    - 80%的(单人)程序员不会经常做这些事情，甚至根本不会做
- **同伴 Debug**
  - 否则一个人有问题的话，会另一个人解释问题 → “没关系;我知道是怎么回事了。很抱歉打扰你。
- **同伴学习 Pair-Learning**
  - 持续复习 → 向合作伙伴学习技巧、语言知识、领域知识等
  - 每一分钟都轮流做老师和学生

使用 Git 进行结对编程

如果你要为类项目成对编码，尝试通过确认你的成对程序员来配置 git

> https://github.com/findmypast-oss/git-mob
>
> https://github.com/augustohp/git-pair

3. 用户故事 User Story

什么是用户故事

- 用户故事表达了
  - 客户想要在软件里看到的一种功能
  - 用户故事是允许客户定义（并引导）贯穿系统的路径所需的最小信息量（一个步骤）
  - 由客户编写（通过与开发人员沟通），**而不是由开发人员编写**
  - 经常是写在索引卡片上

编写用户故事

- 材料
  - 空白的索引卡片
  - 笔
  - 橡皮筋
- 从系统的目标开始
  - eg：申请人提交一个贷款申请
- 考虑用户在执行活动时所采取的步骤
- **在每张卡片上不要写超过一个步骤**

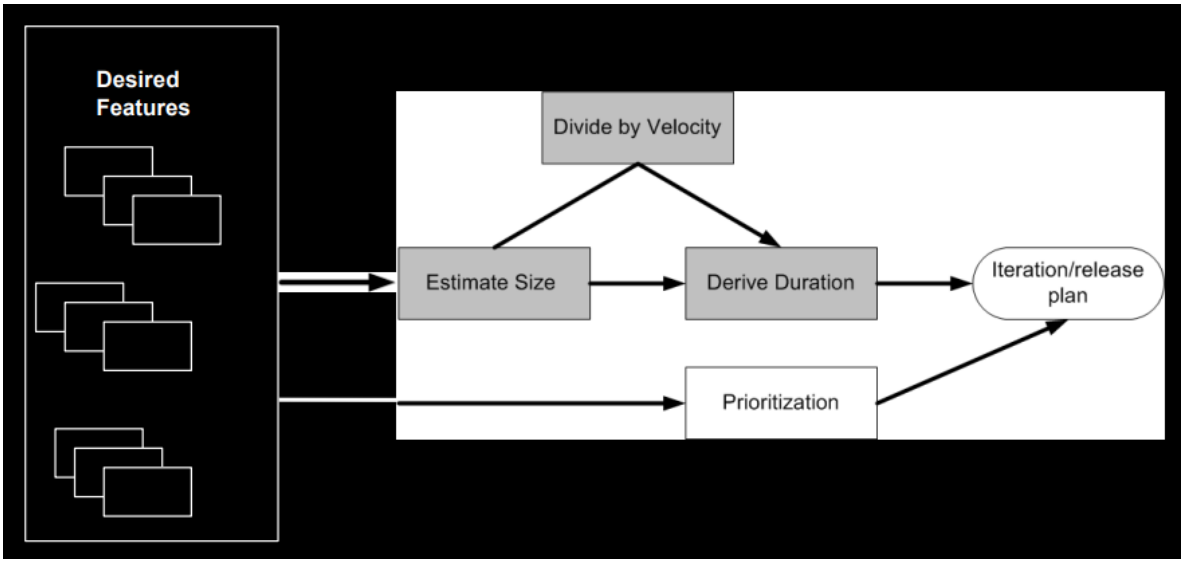
用户故事的格式

Title: Enter Player Info		
Acceptance Test: enterPlayerInfo1	Priority: 1	Story Points: 1
Right after the game starts, the Player Information dialog will prompt the players to enter the number of players (between 2 and 8). Each player will then be prompted for their name, which may not be an empty string. If Cancel is pressed the game exits gracefully.		

- **标题 Title**：2-3 个单词
- **验收测试（唯一标识符）Acceptance test**
- **优先级 Priority**：1-2-3（1 是最高的）
- **故事点 Story points**：可以意味着**理想**开发时间（天数），也就是说，没有干扰或致力于其他事情
- **描述 Description**：1-2 句话（实现目标的某个单一步骤）

4. 计划 Plan

计划即将到来的事情



概念

- **故事点 Story point**：表示用户故事、特性或其他工作的总体大小的度量单位，**故事点的原始价值并不重要**，重要的是**相对值**
  - 与它有多难和有多少有关
  - 与时间或人数**无关**
  - 无单位，但有数值意义
- **理想时间 Ideal time**：剥离所有外围活动“做某事”所花费的时间
  - eg：美式足球比赛 = 60分钟
- **运行时间 Elapsed time**：指在时钟上流逝去做“某事”的时间
  - eg：美式足球比赛 = 3小时
- **速度 Velocity**：衡量团队的进度

优先级

- High：给我们提供这些故事来提供一个最小的工作系统
- Medium：我们需要这些故事来完成这个系统
- Low：铃声和口哨声？哪些故事可以稍后再讲？

预估任务大小 Estimate Size

预估故事点

- 选择一个中等数量的用户故事，给它分配“5” 个故事点
- 估计与之相关的故事

预估理想时间

- 软件开发中的理想时间与实际运行时间
  - 支持当前版本
  - 生病的时间
  - 会议
  - 演示
  - 个人问题
  - 电话
- 当在预估理想时间时，假设

- 故事点是你唯一要做的事情
- 当你开始时，你需要的一切都会在你身边
- 不会有任何干扰

支持预估故事点

- 帮助推动跨职能行为
- 不衰减(基于经验的变化)
- 是纯粹的尺寸衡量标准(关注特征，而不是人)
- 从长远来看，评估通常更快
- 我的理想时间不是你的理想时间(关注人和他们的速度)

支持预估理想时间

- 在团队外更容易解释
- 一开始的估算时间会更短一些

对用户故事进行估算

- **专家意见 Expert opinion**
  - 依靠基于（丰富的）经验的直觉
  - 敏捷的缺点:需要考虑开发用户故事的各个方面，所以一个专家可能是不够的
- **类比 Analogy**
  - 相对于（几个）其他用户场景
- **分解 Disaggregation**
  - 将任务分成更小、更容易估计的部分/任务
  - 需要确保你没有错过任何任务
  - 完整性检查：所有部分的总和是否有意义?
- **规划扑克 Planning poker**
  - 结合专家意见，类比，分解

规划扑克 Planning poker

https://blog.csdn.net/weixin_42556618/article/details/85102975

除以速度并获得时间 Divide by Velocity -> Derive Duration

速度 Velocity

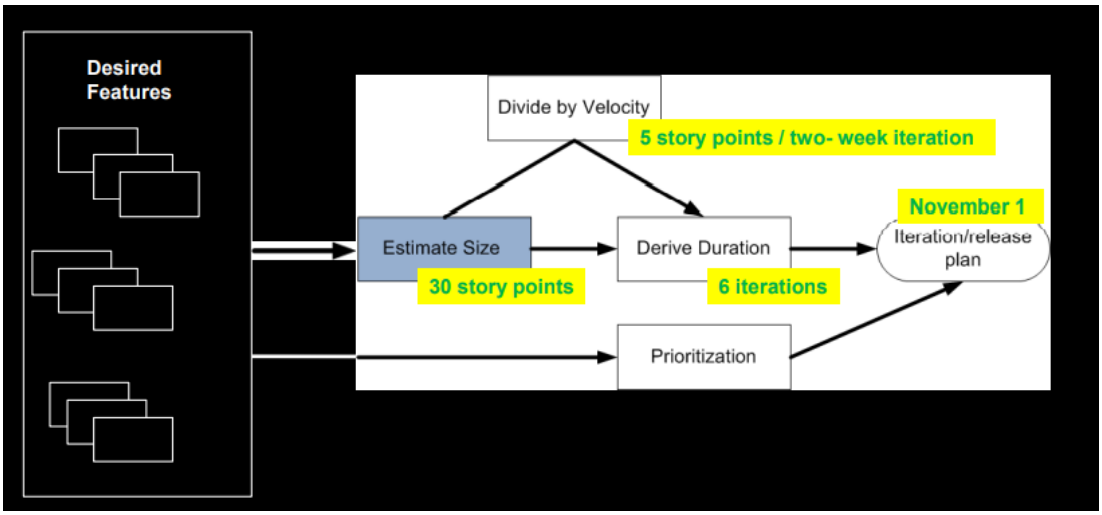
- 开发速度是对团队进度的度量
- 速度是通过将团队在操作期间**完成**的**分配给每个用户故事的故事点**的数量相加来计算的
- 我们假设团队将在未来的迭代中以过去的平均速度生成产品
  - 昨天的效果就是今天的天气预报

优先次序 Prioritization

- 客户驱动，与开发人员合作
- 选择填入本次迭代的特征，基于
  - 对广泛的客户/用户的需求
  - 功能对少数重要客户/用户的可取性
  - 故事与其他故事之间的衔接性

- 例如
  - “放大”是一个高优先级的功能
  - “缩小”不是高优先级功能
    - 但是它与放大有关

预期效果示例



5. XP 的流程

预估游戏 Planning Game

- 客户写下用户故事
- 程序员预估完成每个用户故事需要的时间
- 如果某个故事太大了，客户将它拆分
- 用户选择能够匹配项目**速度**的故事
- 项目速度是指在前一次迭代中完成的工作量

计划 Planning

- 程序员每次只关心一次迭代
- 客户可以根据需要计划尽可能多的迭代，但是可以更改未来的迭代

简化 Simplicity

- 一次添加一个特性（用户故事）
- 不要担心将来的用户故事
- 让程序越简单越好

单元测试和重构 Unit Tests and Refactoring

- 因为代码是尽可能简单的，添加一个新特性往往会使它变得不那么简单
- 要恢复简单性，必须**重构**代码
- 为了安全地重构，应该有一组严格的**单元测试**

有效的 XP

- 存在具有相关经验的客户
- 小型团队
- 人们比较擅长沟通
- 客户和开发人员都在一个房间里
- 需求有变更

有效的软件

- 所有软件都有自动化（单元）测试
- 所有测试都能通过
    - 永远不要签入损坏的代码
- 如何完成一项任务
    - 获得最近版本的代码，确保所有测试通过
    - 先写该任务的测试（当然它肯定是失败的）
    - 编写代码通过所有的测试（现在所有的测试都可以了）
    - 重构（使代码变得整洁）
    - 最后检查一次代码