中山大學
SUN YAT-SEN UNIVERSITY

Chapter 1

Overview of Software Engineering

Software Testing: Approaches & Technologies

School of Data & Computer Science, Sun Yat-sen University

Outline

- 1.1. 软件与软件危机
- 1.2. 软件开发
- 1.3. 软件生命周期
- 1.4. 软件质量模型
- 1.5. 敏捷开发

1.5 敏捷开发

1.5.1 敏捷开发

- 敏捷开发概述
  - 敏捷开发的起源
    - ✧ 敏捷建模 (Agile Modeling，AM) 源于 *Scott W. Ambler* 的 Extreme Modeling (XM，2000)。2001年以 *Kent Beck, Alistair Cockburn, Ward Cunningham, Martin Fowler* 等人为首在 Snowbird，Utah 发布《敏捷宣言》，决定将 Agile 作为新的轻量级软件开发过程的家族名称。
    - ✧ AM 是从软件开发过程实践中归纳总结出来的一些敏捷建模价值观、原则和实践。AM 是一种态度，而不是一个说明性过程。
    - ✧ AM 不是一个完整的方法论，而是对已有生命周期模型的补充，在应用传统的生命周期模型时可以借鉴 AM 的过程指导思想。
    - ✧ http://agilemanifesto.org/

1.5 敏捷开发

1.5.1 敏捷开发

- 敏捷开发概述
  - 敏捷宣言

1.5 敏捷开发

1.5.1 敏捷开发

- 敏捷开发概述 (续)
  - 敏捷宣言 (Manifesto for Agile Software Development)
    - We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value
      - ***Individuals and Interactions*** *over processes and tools*
      - ***Working Software*** *over comprehensive documentation*
      - ***Customer Collaboration*** *over contract negotiation*
      - ***Responding to Change*** *over following a plan*
    - That is, while there is value in the items on the right, they value the items on the left more.

1.5 敏捷开发

1.5.1 敏捷开发

- 敏捷开发概述 (续)
  - 敏捷宣言
    - ✧ 敏捷建模的价值观
      - ◌ **个人和交互** 重于 过程和工具
      - ◌ **可用的软件** 重于 面面俱到的文档
      - ◌ **客户合作** 重于 合同谈判
      - ◌ **响应变化** 重于 遵循计划

1.5 敏捷开发

1.5.1 敏捷开发

- 敏捷开发概述 (续)
  - 敏捷开发的目标
    - ✧ 敏捷开发的总体目标是通过“尽可能早地、持续地对有价值软件的交付”，使客户满意。
    - ✧ 敏捷开发强调软件开发应当能够对未来可能出现的变化和不确定性作出全面反应。
    - ✧ 敏捷开发主要用于在需求模糊或快速变化的前提下，支持小型开发团队的软件开发活动。

1.5 敏捷开发

1.5.1 敏捷开发

- 敏捷开发概述 (续)
  - 敏捷开发的管理原则
    - ✧ 敏捷开发是一种以人为核心、迭代、循序渐进的开发过程指导思想。
    - ✧ 在敏捷开发过程中，软件项目的构建被切分成多个子项目分别实现。各个子项目之间相互联系，独立运行，子项目的成果经过测试，具备集成和可运行的特征。

1.5 敏捷开发

1.5.1 敏捷开发

- 敏捷开发方法的12条原则

1. 最优先原则：通过尽早、持续交付有价值的软件来使客户满意。
2. 即使在开发的后期，用户的需求变更也是允许的。
3. 经常交付可工作软件。
4. 在整个项目开发期间，业务人员和开发人员最好在一起工作。
5. 强化激励机制，为受激励的个人单独构建项目。
6. 在团队内部，最富有效果和效率的信息传递方法是面对面交谈。
7. 可工作软件是进度的首要度量标准。
8. 敏捷过程提倡可持续的开发速度。
9. 不断地关注优秀的技能和好的设计，增强敏捷能力。
10. 简化原则：尽量简化所要做的工作。
11. 好的架构、需求和设计出自于组织团队自身。
12. 团队定期反省如何更有效地工作，并相应地调整自己的行为。

1.5 敏捷开发

1.5.1 敏捷开发

- 敏捷开发方法的核心实践
  - ✧ 项目关键利益方 (Project Stakeholder) 的积极参与
  - ✧ 正确使用工件
  - ✧ 集体所有制 (对代码、工件、模型的共有，包括使用和修改)
  - ✧ 测试性思维 (比如“测试优先”)
  - ✧ 并行创建模型 (为一个问题同时建立多种模型)
  - ✧ 创建简单的内容
  - ✧ 简单地建模
  - ✧ 公开展示模型 (使用 modeling wall 向项目参与各方展示建立的模型)
  - ✧ 切换到另外的工件 (遇到困难时的敏捷切换)
  - ✧ 小增量建模
  - ✧ 和他人一起建模
  - ✧ 用代码验证模型
  - ✧ 使用最简单的建模工具

1.5 敏捷开发

1.5.1 敏捷开发

- 敏捷开发方法的补充实践
  - ✧ 使用建模标准 (比如 UML)
  - ✧ 逐渐应用模式 (pattern)
  - ✧ 丢弃临时模型
  - ✧ 合同模型要正式
  - ✧ 为外部交流建模
  - ✧ 为帮助理解建模
  - ✧ 重用现有的资源
  - ✧ 不到万不得已不更新模型

1.5 敏捷开发

1.5.1 敏捷开发

- 敏捷开发方法分类
  - XP (极限编程)
    - ✧ XP 提倡测试先行，以将后面出现 bug 的概率降至最低。
  - SCRUM (迭代增量过程)
    - ✧ SCRUM 是一种迭代的增量化过程，用于产品开发或工作管理。
  - Crystal Methods (水晶方法系列)
    - ✧ 水晶系列与 XP 一样，都是以人为中心，但不同类型的项目需要不同的实践方法。
  - FDD (特性驱动开发)
    - ✧ FDD 是一套针对中小型软件开发项目的开发模式，采用模型驱动的快速迭代开发过程。

1.5 敏捷开发

1.5.1 敏捷开发

- 敏捷开发方法分类 (续)
  - ASD (自适应软件开发)
    - ASD 从复杂自适应系统理论派生出来，用于对需求多变、开发周期短项目的管理。
  - DSDM (动态系统开发方法)
    - DSDM 倡导以业务为核心，快速而有效地进行系统开发。
  - RUP
    - RUP 是一个过程框架，它可以包容许多不同类型的过程，但核心还是面向对象过程。

1.5 敏捷开发

1.5.2 极限编程 (XP) 模型

- 极限编程 (eXtreme Programming, XP)
  - 极限编程是敏捷模型的一种实现过程，由 *Kent Beck* 在1996年提出。极限编程适合：
    - ✧ 小团队 (2-10 programmers)
    - ✧ “高风险”
    - ✧ 快速变化或不稳定的需求
    - ✧ 强调可测试性
  - 格言
    - ✧ 沟通 Communication
    - ✧ 简化 Simplicity
    - ✧ 反馈 Feedback
    - ✧ 激励 Courage
    - ✧ *谦逊 Modesty

Kent Beck, 1996

最简单的可能就是最有效的

1.5 敏捷开发

1.5.2 极限编程 (XP) 模型

- 极限编程方法的13个核心实践
  - ✧ 团队协作 (Whole Team)
  - ✧ 规划策略 (The Planning Game)
  - ✧ 结对编程 (Pair programming)
  - ✧ 测试驱动开发 (Testing-Driven Development)
  - ✧ 重构 (Refactoring)
  - ✧ 简单设计 (Simple Design)
  - ✧ 代码集体所有 (Collective Code Ownership)
  - ✧ 持续集成 (Continuous Integration)
  - ✧ 客户测试 (Customer Tests)
  - ✧ 小规模发布 (Small Release)
  - ✧ 每周40小时工作制 (40-hour Week)
  - ✧ 编码规范 (Code Standards)
  - ✧ 系统隐喻 (System Metaphor)

1.5 敏捷开发

1.5.2 极限编程 (XP) 模型

- 极限编程的12个实践
  - 小版本
    - ✧ 小版本发布有利于高度迭代以及给客户展现开发的进展，客户可以针对性提出反馈。小版本也需要总体合理的规划，如果把模块缩得太小，会影响软件的整体思路。
  - 规划策略
    - ✧ 客户以故事的形式编写客户需求。极限编程不讲求统一的客户需求收集，客户需求不是由开发人员整理，而让客户编写，开发人员进行分析，设定优先级别，进行技术实现。规划策略可以进行多次，每次迭代完毕后再行修改。客户故事是开发人员与客户沟通的焦点，也是版本设计的依据，所以其管理必须是有效的、沟通顺畅的。

1.5 敏捷开发

1.5.2 极限编程 (XP) 模型

- 极限编程的12个实践 (续)
  - 现场客户
    - ✧ 极限编程要求客户参与开发工作，客户需求就是客户负责编写的，所以要求客户在开发现场一起工作，并为每次迭代提供反馈。
  - 隐喻
    - ✧ 隐喻是让项目参与人员都必须对一些抽象的概念 (行业术语) 理解一致，因为业务本身的术语开发人员不熟悉，而软件开发的术语客户不理解，因此开始要先明确双方使用的隐喻，避免歧异。

1.5 敏捷开发

1.5.2 极限编程 (XP) 模型

- 极限编程的12个实践 (续)
  - 简单设计
    - ✧ 极限编程体现跟踪客户的需求变化，既然需求是变化的，所以对于目前的需求不必过多考虑扩展性的开发，而讲求简单设计，实现目前需求即可。简单设计的本身也为短期迭代提供了方便，若开发者考虑“通用”因素较多，增加了软件的复杂度，将会加长开发的迭代周期。

1.5 敏捷开发

1.5.2 极限编程 (XP) 模型

- 极限编程的12个实践 (续)
  - 重构
    - ✧ 重构是极限编程先测试后编码的必然需求，为了整体软件可以先进行测试，对于一些软件要开发的模块先简单模拟，让编译通过，到达测试的目的。然后再对模块具体“优化”，所以重构包括模块代码的优化与具体代码的开发。重构是使用了“物理学”的一个概念，是在不影响物体外部特性的前提下，重新优化其内部的机构。这里的外部特性就是保证测试的通过。

1.5 敏捷开发

1.5.2 极限编程 (XP) 模型

- 极限编程的12个实践 (续)
  - 测试驱动开发
    - ✧ 极限编程是以测试开始的，为了可以展示客户需求的实现，测试程序优先设计，测试是从客户实用的角度出发，客户实际使用的软件界面着想，测试是客户需求的直接表现，是客户对软件过程的理解。测试驱动开发，也就是客户的需求驱动软件的开发。
  - 持续集成
    - ✧ 集成的理解就是提交软件的展现，由于采用测试驱动开发、小版本的方式，所以不断集成 (整体测试) 是与客户沟通的依据，也是让客户提出反馈意见的参照。持续集成也是完成阶段开发任务的标志。

1.5 敏捷开发

1.5.2 极限编程 (XP) 模型

- 极限编程的12个实践 (续)
  - 结对编程
    - ✧ 这是极限编程最有争议的实践。就是两个程序员合用一台计算机编程，一个编码，一个检查，增加专人审计是为了提供软件编码的质量。两个人的角色经常变换，保持开发者的工作热情。这种编程方式对培养新人或开发难度较大的软件都有非常好的效果。
  - 代码共有
    - ✧ 在极限编程里没有严格文档管理，代码为开发团队共有，这样有利于开发人员的流动管理，因为所有的人都熟悉所有的编码。

1.5 敏捷开发

1.5.2 极限编程 (XP) 模型

- 极限编程的12个实践 (续)
  - 编码规范
    - ✧ 编码是开发团队里每个人的工作，又没有详细的文档，代码的可读性很重要，所以规定统一的标准和习惯是必要的。
  - 每周40小时工作
    - ✧ 极限编程认为编程是愉快的工作，不要轻易加班，小版本的设计也是为了单位时间可以完成的工作安排。

1.5 敏捷开发

1.5.2 极限编程 (XP) 模型

- 极限编程模型 - 开发周期

Planning

- User stories are written.
- Release planning creates the schedule.
- Make frequent small releases.
- The Project Velocity is measured.
- The project is divided into iterations.
- Iteration planning starts each iteration.
- Move people around.
- A stand-up meeting starts each day.
- Fix XP when it breaks.

Coding

- The customer is always available.
- Code must be written to agreed standards.
- Code the unit test first.
- All code is pair programmed.
- Only one pair integrates code at a time.
- Integrate often.
- Use collective code ownership.
- Leave optimization till last.
- No overtime.

Designing

- Simplicity.
- Choose a system metaphor.
- Use CRC cards for design sessions.
- Create spike solutions to reduce risk.
- No functionality is added early.
- Refactor whenever and wherever possible.

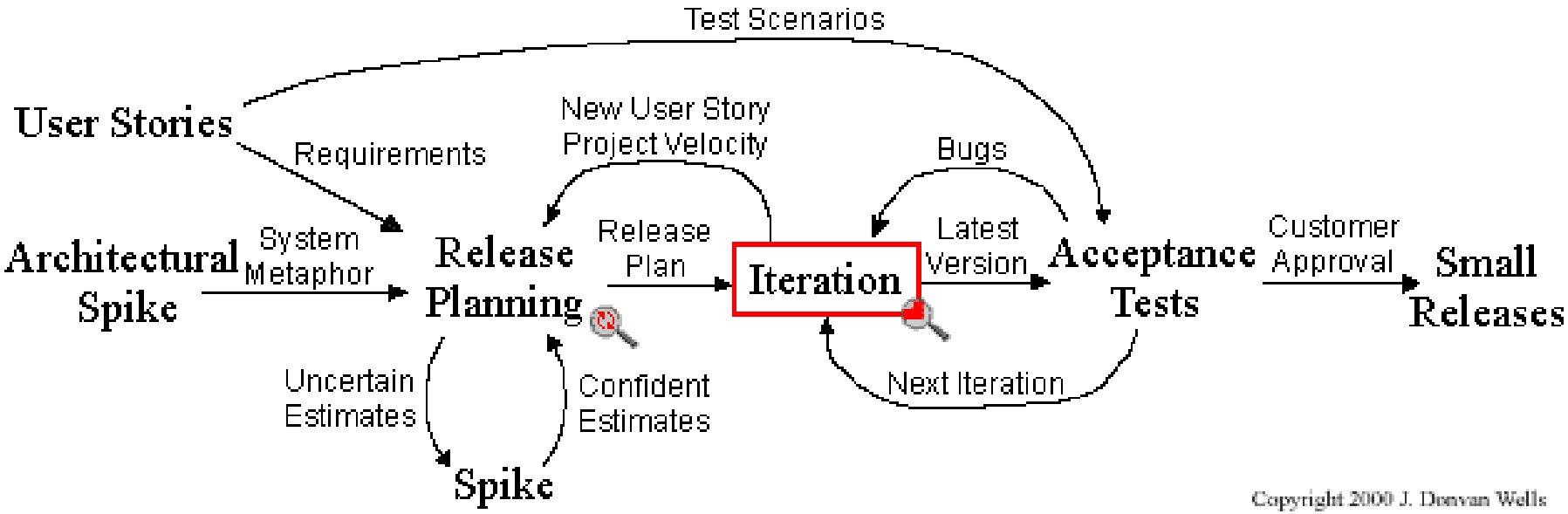
Testing

- All code must have unit tests.
- All code must pass all unit tests before it can be released.
- When a bug is found tests are created.
- Acceptance tests are run often and the score is published.

1.5 敏捷开发

1.5.2 极限编程 (XP) 模型

- 极限编程模型 - 开发周期



Thank you!