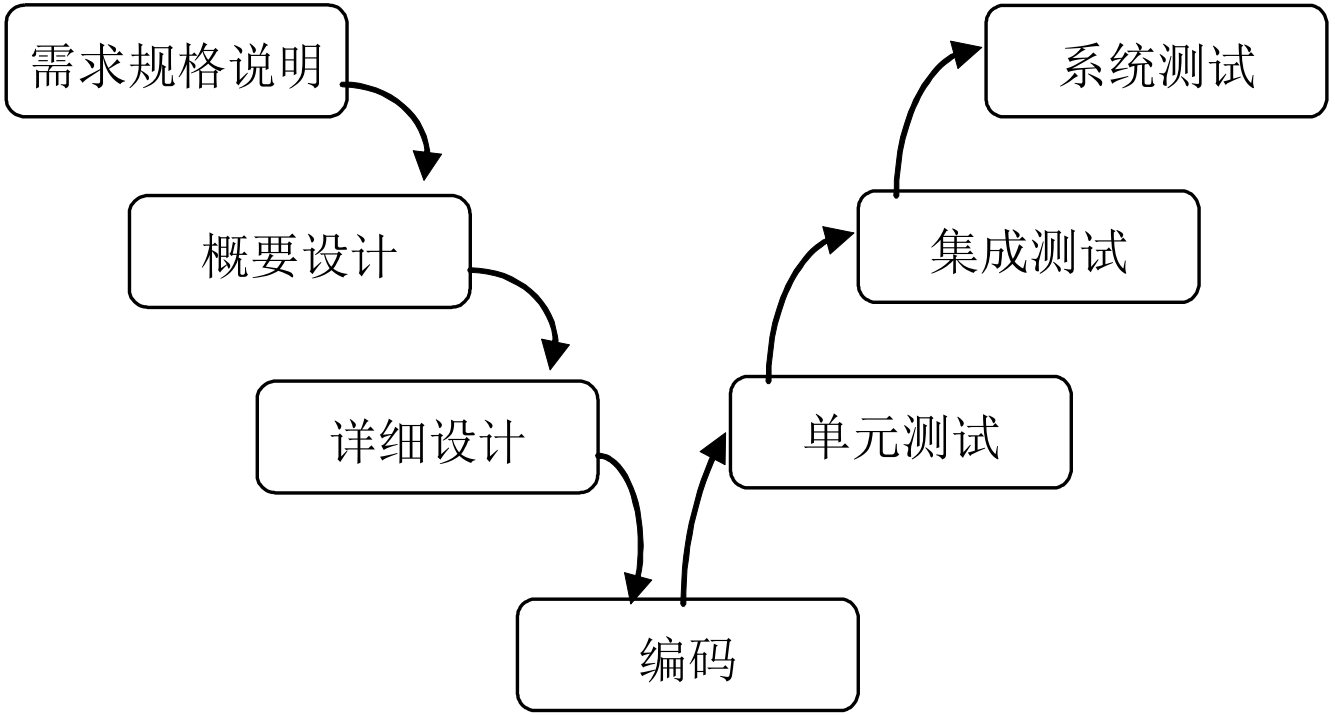
Chapter 8-1

基于生命周期的测试

8.1 测试层次

8.1.1 传统的观点



瀑布式生命周期

从传统瀑布模型的观点:

- 单元测试: 面向详细设计,完成对软件独立模块的测试。
- 集成测试：面向概要设计，完成软件模块之间的组合测试。
- 系统测试：面向需求分析，完成系统的功能测试。

瀑布模型的优缺点

- 优点：
  - 瀑布模型框架非常适合层次管理结构
  - 每个阶段都有明确的产物，便于进行项目管理
  - 详细设计中分工明确，可以并行工作，缩短周期

- 缺点？
  - 需求规格说明与系统测试之间周期很长，无法加入客户反馈
  - 模型几乎排除了综合可能，最先发生在集成测试
  - 单元级别大规模并行开发可能会受到人员数量限制
  - 开发人员需要对系统“完美”理解

瀑布模型的变体

- 增量开发模型：
  - 是将一个软件产品分成若干次产品进行提交，每一次新的软件产品的提交，都是在上次软件产品的基础上，增加新的软件功能，直到全部满足客户的需求为止。
- 演化开发模型：
  - 演化开发模型是在需求分析之前，首先提供给客户或用户一个最终产品的原型（部分主要功能的软件）
- 螺旋模型：
  - 它与瀑布模型和快速原型模型十分相似。但重要的是，它在每个阶段都增加了风险分析和验证这两个重要的步骤。

- 这三种派生模型的好处，是增加了迭代开发的方法，而不是将成功的风险全部放在了最后阶段！
- 在这样的开发模型中，回归测试就成为一种重要的测试方法。
- 回归测试（Regression test）：对已经完成测试的软件进行修改和增加之后，重新测试软件。
- 回归测试我们在后续章节进行专门的介绍，这里不做特别说明。

敏捷开发与测试

- 事实上，并不仅仅是软件维护阶段的工作。即使在软件开发阶段，软件的实现过程就是一个不断迭代和升级的过程！

  敏捷开发

- 是软件开发方法的一个很大的范畴，包括Scrum、极限编程（XP）、测试驱动（TDD）、适应性软件开发、结对编程等。

推荐书目：

1. 《敏捷软件开发：原则、模式与实践》，[美]马丁 著，邓辉 译，清华大学出版社。
2. 《Java敏捷开发——使用Spring、Hibernate和Eclipse》，（美）Anil Hemrajani;韩坤,徐琦，人民邮电出版社。

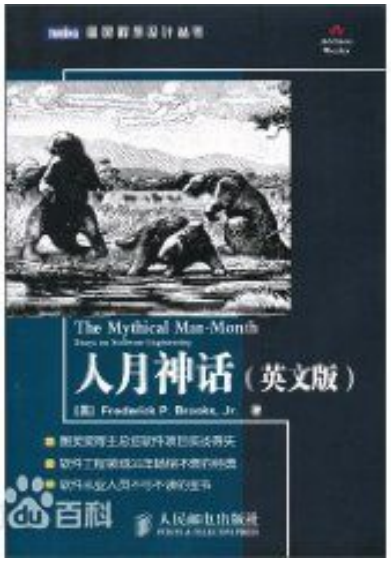
敏捷价值观之敏捷宣言

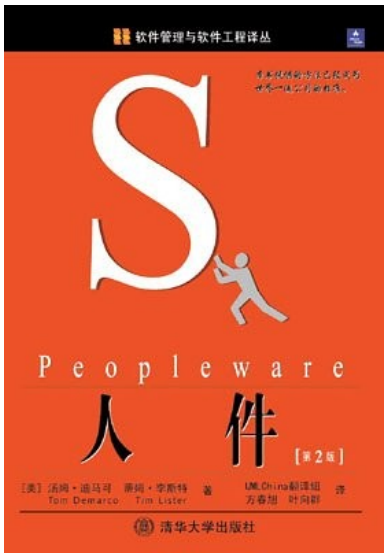
个体与交互	重于	过程和工具
可用的软件	重于	完备的文档
客户协作	重于	合同谈判
响应变化	重于	遵循计划

敏捷开发的核心思想是：以人为本，适应变化。

敏捷价值观之敏捷宣言-1

- 个体和交互 胜过 过程和工具
  - 人是软件项目获得成功最为重要的因素
  - 合作、沟通能力以及交互能力比单纯的软件编程能力和工具更为重要
  - 方法和工具是“死”的，人是“活”的，人要是协作不好，再强大的方法和工具都是“白扯”；





敏捷价值观之敏捷宣言-2

- 可用的软件 胜过 完备的文档
  - 过多的面面俱到的文档往往比过少的文档更糟
  - 软件开发的主要和中心活动是创建可以工作的软件
  - 直到迫切需要并且意义重大时，才进行文档编制
  - 编制的内部文档应尽量短小并且主题突出

敏捷价值观之敏捷宣言-3

- 客户协作 胜过 合同谈判
  - 客户不可能做到一次性地将他们的需求完整清晰地表述在合同中
  - 为开发团队和客户的协同工作方式提供指导的合同才是最好的合同

敏捷价值观之敏捷宣言-4

- 响应变化 胜过 遵循计划
  - 变化是软件开发中存在的现实
  - 计划必须有足够的灵活性与可塑性
  - 短期迭代的计划比中长期计划更有效

敏捷方法 VS. 瀑布模型

- 瀑布模型
  - 固定的、没有弹性的。
  - 很困难去达到互动。
  - 假如说需求没有完全的被了解，或是可能需要完全地改变项目的需求，瀑布式的模型是比较不适合的。
- 敏捷方法
  - 完整地开发，每少数几周或是少数几个月里可以测试功能。
  - 强调在获得最简短的可执行功能的部分，能够及早给予企业价值。
  - 在整个项目的生命周期里，可以持续的改善、增加未来的功能。

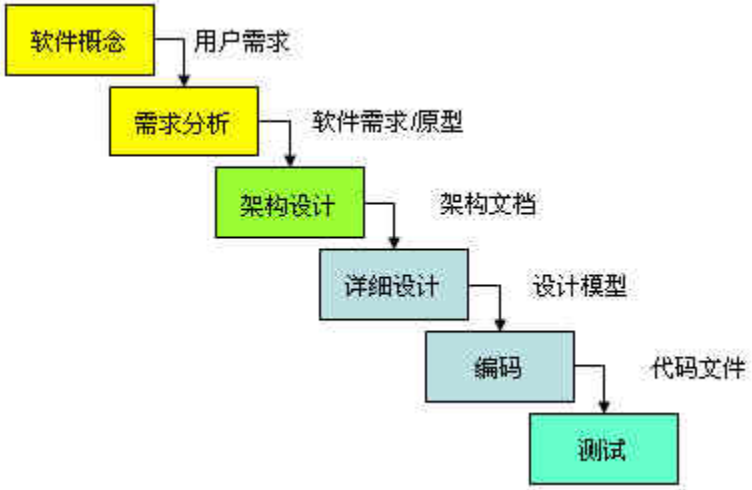
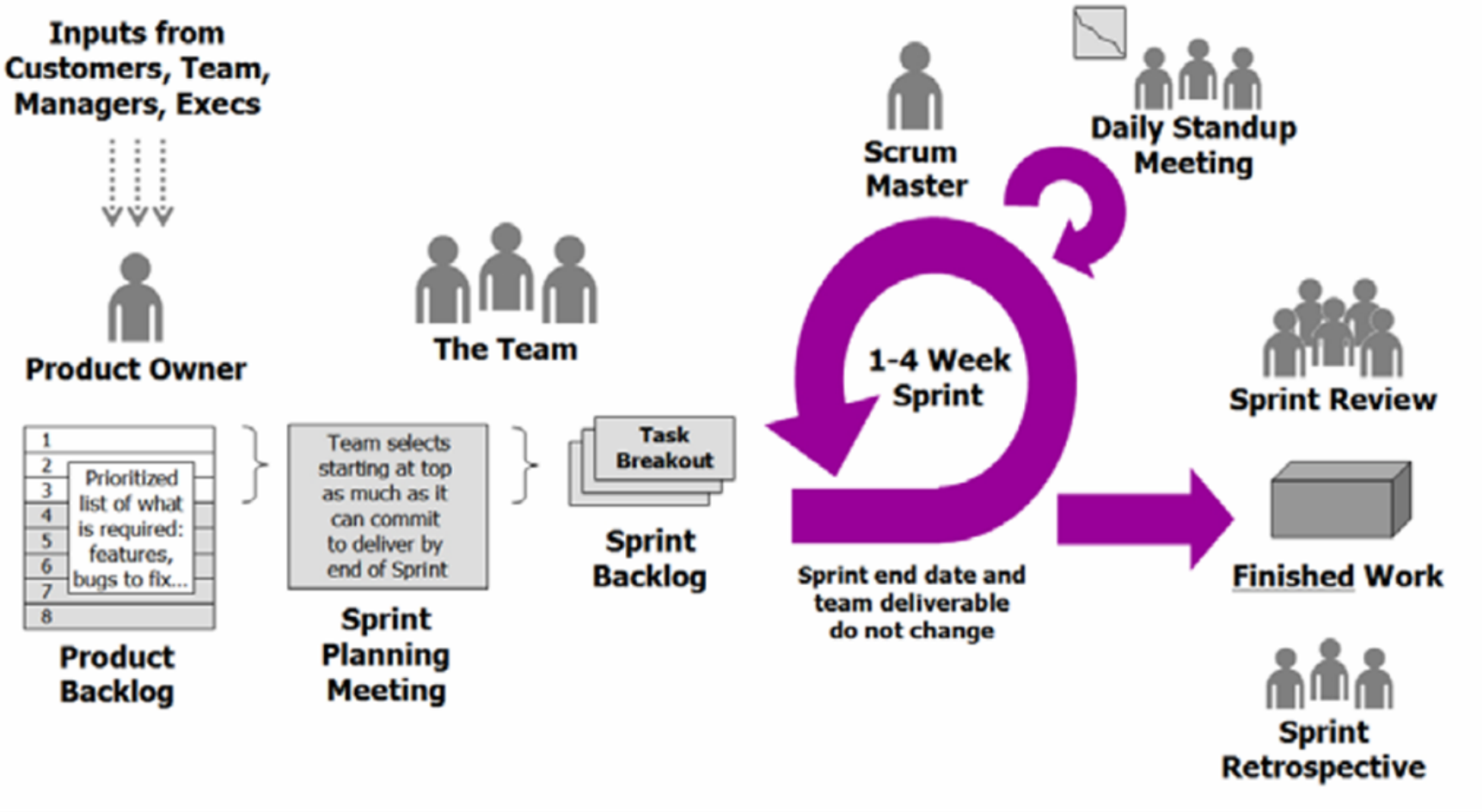


Figure 1. Scrum

图示 1 Scrum

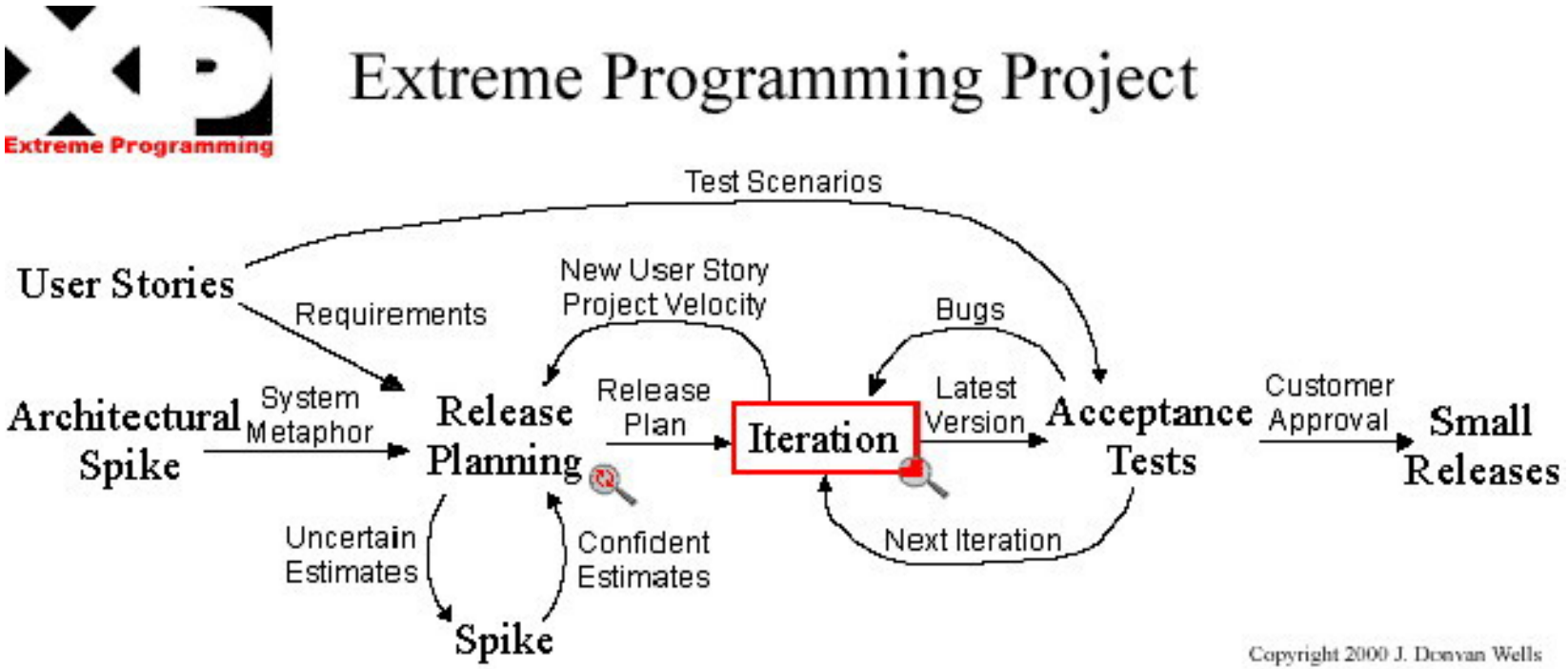
Scrum敏捷开发流程



相关网站：agilemodeling.com
extremeprogramming.org

极限编程流程

- XP活动：倾听→ 测试→ 编码→ 设计（重构）



XP的一些典型实践活动：共12项

（1）.现场客户（On-site Customer）

（2）.计划游戏（Planning Game）

（3）.系统隐喻（System Metaphor）

（4）.简单设计（Simple Design）

（5）.代码集体所有（Collective Code Ownership）

（6）.结对编程（Pair Programming）

（7）.测试驱动（Test-driven）

（8）.小型发布（Small Releases）

（9）.重构（Refactoring）

（10）.持续集成（Continuous integration）

（11）.每周40小时工作制（40-hour Weeks）

（12）.代码规范（Coding Standards）

（1） 现场客户

始终在开发团队中有一位客户。

现场客户的工作：

- 回答问题
- 编写验收测试
- 运行验收测试
- 指导迭代
- 接受版本

（2）计划游戏

- 计划是持续的、循序渐进的。每2周，开发人员就为下2周估算候选特性的成本。
- 以业务优先级和技术估计为基础，决定下一版本发布的范围。

（3）系统隐喻

在XP中，隐喻是一种概念框架并提供名称的描述系统，类似于其他方法中的体系结构（或体系结构基准）。

- 共识
- 共享的术语空间

例子：Windows风格的界面、网上购物站点的购物车

（4） 简单设计

系统应设计得尽可能简单。

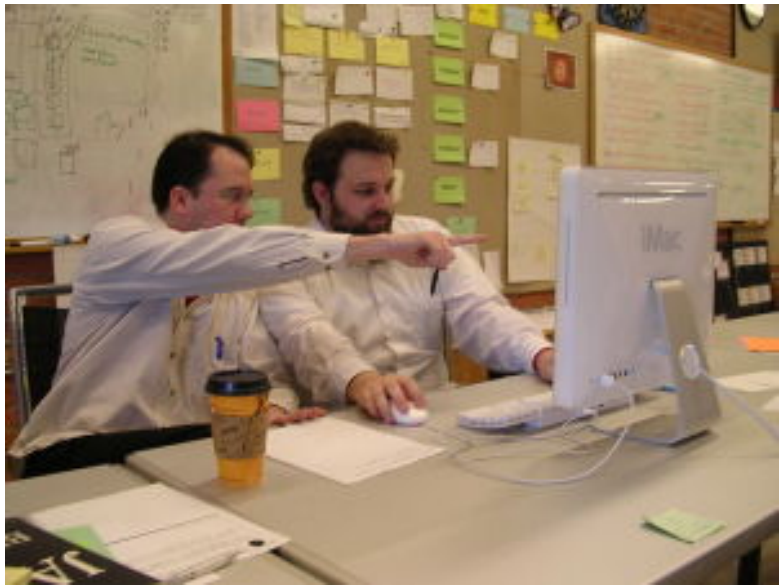
聚沙成塔，集腋成裘

（5）代码集体所有

整个团队拥有所有代码。任何人都可以更改他们需要更改的部分。没有唯一对代码有所有权的人。

（6）结对编程

- 结对编程是让两个人共同设计和开发代码的实践。结对者是全职合作者，轮流执行键入和监视；这提供了持续的设计和代码评审。
- 不是两个人做一个人的事情。



积极影响

- 经济性
- 满意度
- 提高设计质量：分享不同的先验知识、理解和角色
- 持续复查
- 问题解决更快：集思广益和配对接力
- 学习：耳濡目染
- 团队建设和沟通
- 有利于人员和项目管理

卡车问题

一个项目组集体外出，不幸被卡车撞上。有多少人受伤使项目不得不停止？

最坏的情况是一个！

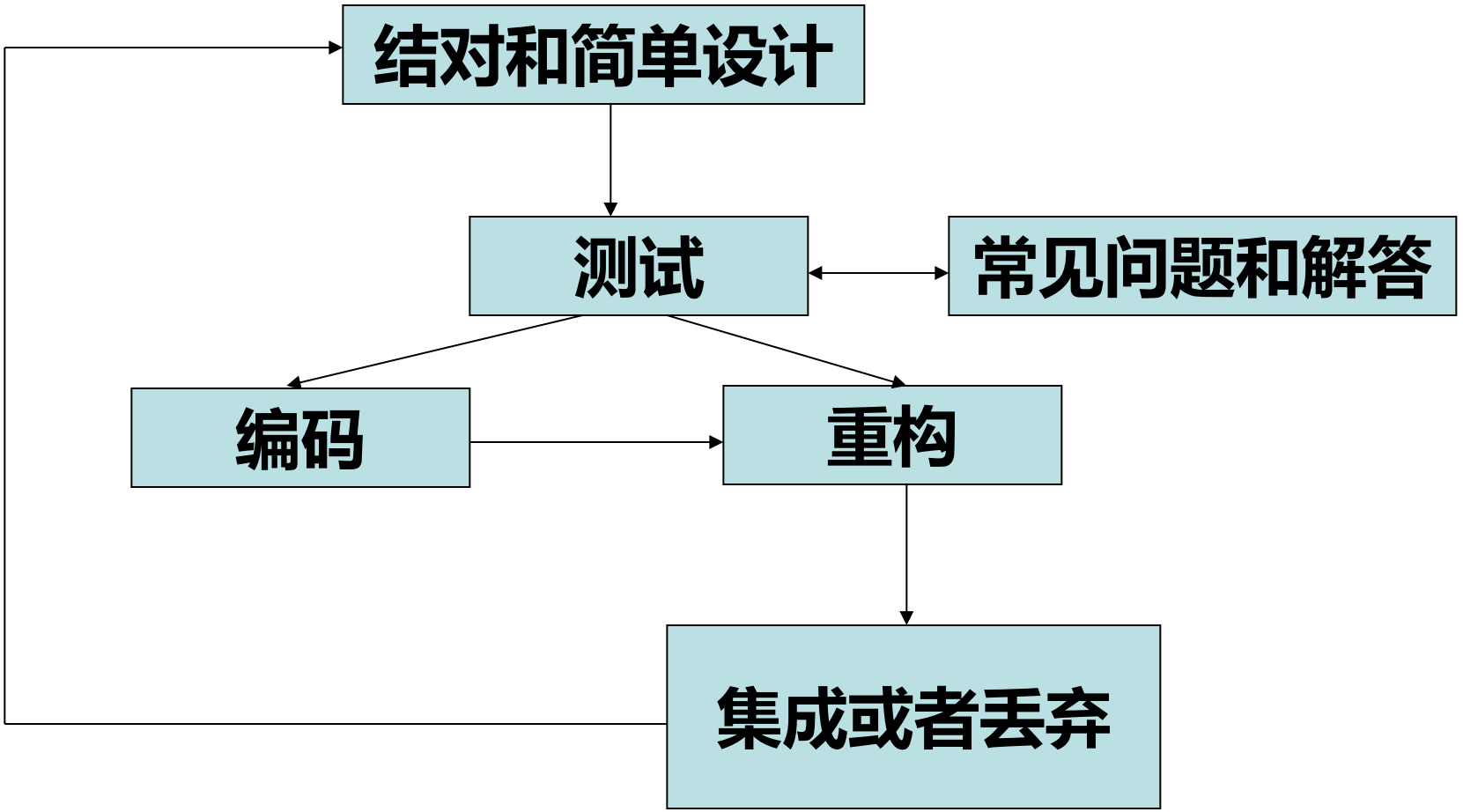
学到的经验

- 程序员和设计人员\协调人结对编程更有效。
- 键盘输入效率！
- 自愿结对编程。

我们行业的主要问题实质上更侧重于社会学而不是科学技术。

——《人件》

（7）测试驱动



（8）小型发布

XP推荐小而频繁的有意义发布。

（9）重构

- 重构是XP的一个重要组成部分。所谓重构是指在不改变代码外在行为的前提下对代码做出的修改，以改进代码的内部结构。
- 重构是一种有纪律的、经过训练的、有条不紊的代码整理方法，可以将整理过程中不小心引入错误的可能性降到最低。从本质上说，重构就是在代码写好之后改进它的设计。
- 重构的节奏：重新推理、小的更改、重新推理、小的更改、重新推理…

（10）持续集成

- 持续集成的思想是任何时候只有一项任务完成，就集成新代码，构造系统并测试。持续集成是每日构建\每晚构建的一种极限形式，是XP的重要基础。
- 每日构建\每晚构建是将一个软件项目的所有最新代码取出，从头开始编译、链接，用安装软件包将链接好的程序安装好，运行安装后的软件，使用测试工具对主要功能进行测试，发现错误并报告错误的完整过程。

- 让开发人员在第一时间了解到软件的错误，并迅速排除错误，是每日构建\每晚构建最重要的目标之一。
- 每日构建\每晚构建必须出日志和报告，并发布构建结果的有关信息，最好能够使用自动化工具发出电子邮件通知。

每日构建是项目的心跳。如果一个项目的心跳停止了，这个项目就死亡了。

Treat the daily build as the heartbeat of the project.

If there is no heartbeat, the project is dead.

作用

- 降低集成风险
- 加强错误诊断
- 降低不确定性
- 加快开发速度
- 增强团队合作
- 对项目参与者是重要激励

（11）每周四十小时工作制

在这里40是一个概数，不是确数。

- 如果能够努力地工作8小时，超过这个时间后就不适于有效地工作了——8小时燃烧
- 再学习
- 你无法改变时间，但是可以改变你的任务。

（12）代码规范

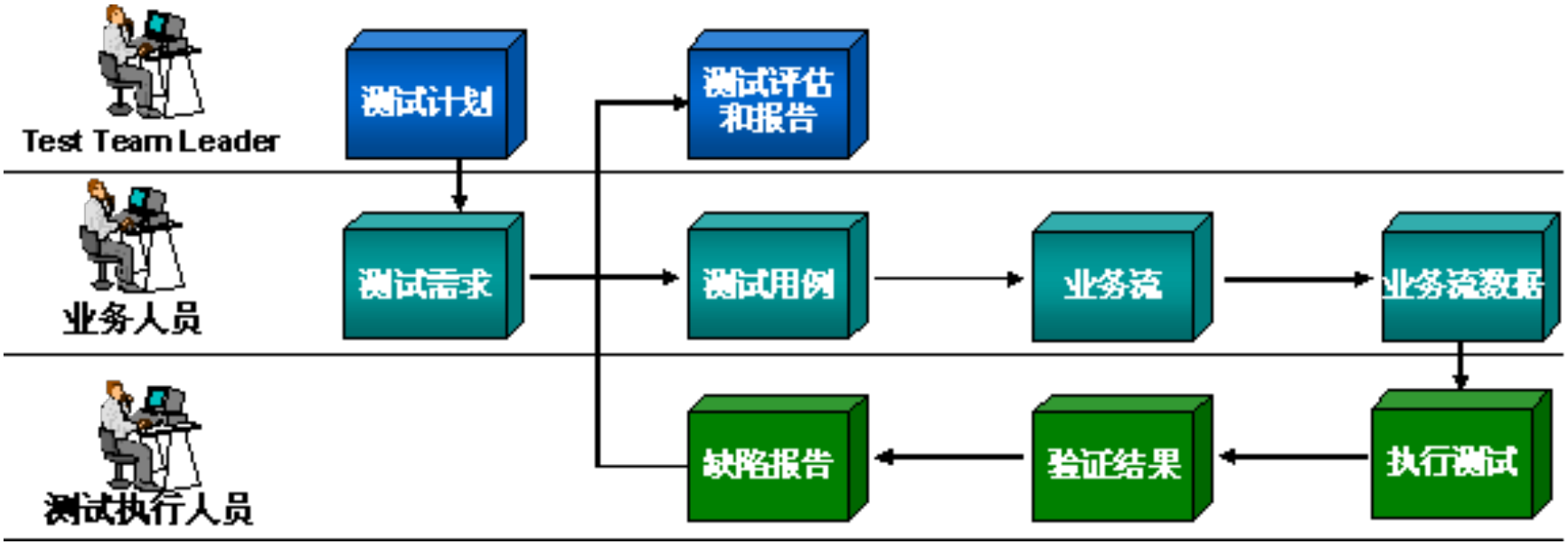
- 系统中所有的代码看起来就好像是被单独一人编写的。

XP的核心活动：

- 简单设计：系统应设计得尽可能简单，并易于重构；
- 测试先行：测试驱动，先有测试用例，后编码；
- 重构：在不改变代码外在行为的前提下对代码做出的修改，以改进代码的内部结构。
- 持续集成：持续集成的思想是任何时候只有一项任务完成，就集成新代码，构造系统并测试。持续集成是每日构建\每晚构建的一种极限形式，是XP的重要基础。

自动测试

软件是一个不断迭代的开发过程，回归测试是软件测试的重要内容。但是，实际执行情况却不尽如人意，其主要原因是目前的回归测试通常都是手工测试。

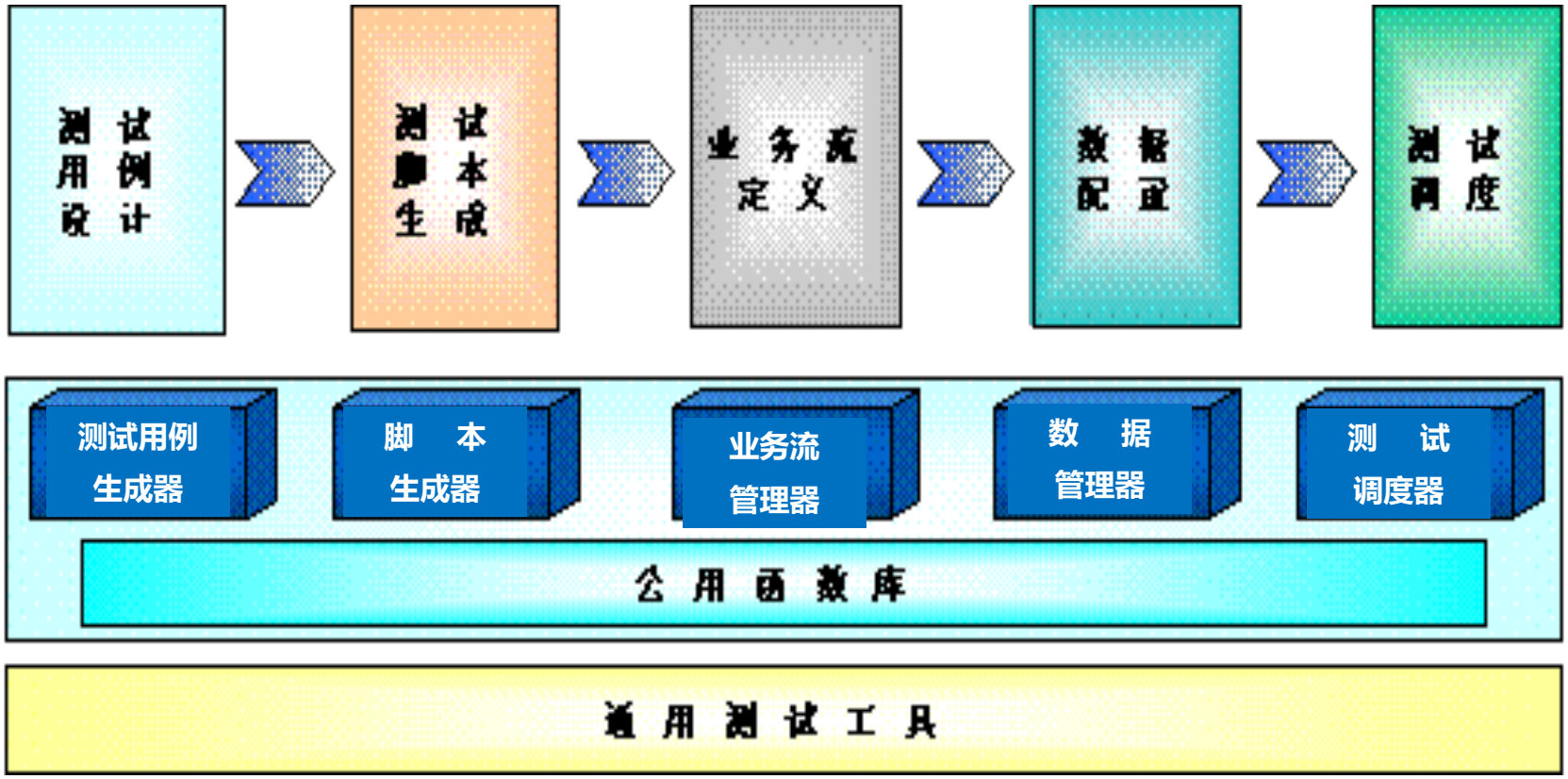


手工测试流程图

存在以下问题：

- 测试过程中，大量精力花在输入数据、执行程序和数据比较上，无法把精力集中到关键问题上。
- 手工测试是一件工作量非常大的、重复的、枯燥的工作。
- 现在的应用系统使用环境比较复杂，通常会运行在不同的操作系统、数据库、中间件等系统软件上，很难组织业务人员对所有环境进行测试。
- 随着迭代开发越来越盛行，测试的频率增加了。传统的手工测试已经满足不了软件开发的需求。
- 参与测试的业务人员不足，无法保证充分的测试，也无法重复。

自动回归测试框架


测试用例设计
测试脚本生成
业务流定义
数据配置
测试调度
测试用例生成器
脚本生成器
业务流管理器
数据管理器
测试调度器
公用函数库
通用测试工具

- 自动回归测试系统功能结构

  自动回归测试系统在通用测试工具上构建了一个自动回归测试框架，框架里包含一个公用函数库和5个主要功能模块。通过自动回归测试框架，自动测试工程师可以完成测试用例设计和自动脚本生成功能，业务人员可以完成业务流定义和数据配置功能，测试执行人员可以完成测试调度和测试管理功能。

下面对5个功能模块进行简单的说明。

1 测试用例设计

- 测试用例是最小的业务分支，它规定了需要操作的对象、步骤、动作和需要使用的数据。测试用例是可复用的，因此在测试用例中的数据应是参数化的，由此测试用例产生的测试脚本也是可复用的。
- 测试用例应由业务人员和技术人员设计。业务人员使用Excel或其它工具来设计，技术人员使用自动回归测试框架来设计。

2 脚本生成器

- 脚本的维护是自动测试系统中维护工作量最大的部分。在自动回归测试框架中，自动脚本生成器可以自动产生健壮的、结构化的、易于维护的脚本。

3 业务流管理器（可选项）

- 自动测试执行的实际上是一个一个的业务流，业务流是由测试用例串联起来的，自动回归测试框架提供以下特性：
  - 业务流管理器可以选择测试用例来组成业务流
  - 在业务流中可支持条件分支，形成树状的业务流
  - 提供业务流管理功能，可以新增、删除、修改业务流
  - 支持在一个业务流中对同一个测试用例的多次引用

4 数据管理器（可选项）

- 数据管理器提供为测试用例指定数据的功能。满足以下要求：
  - 可以在数据管理器上选择需要配置的业务流
  - 选择业务流后，可以选择指定的测试用例
  - 可以显示测试用例所有参数的列表，并提供赋值功能
  - 提供业务流中测试用例间数据映射功能
  - 支持同一个业务流中多次使用同一个测试用例的数据配置

5 测试调度器

- 配置业务流的执行时间，支持预约测试
- 可选择业务流执行
- 执行测试
- 判断并记录测试结果

测试调度器设计，往往包括测试框架和测试环境的设计与实现。

自动化测试还涉及的技术有哪些？

- 缺陷管理
- 脚本管理
- 测试框架与环境设计
- 在嵌入式软件中，还涉及驱动信号生成、系统性能测试（响应时间、资源占用）等技术和方法

Q&A