# 构建之法阅读报告

作者由一个推论和一个类比开始讲起，从而引出了软件工程的相关概念(是什么、难题和目标)；接着讲到了单元和回归测试、效能分析及个人开发流程；紧接着讲到了软件工程师成长之路上的个人能力衡量与发展、思维误区职业发展，和如何正确把握学习区；再接着讲到了合作编程中应该考虑的因素：代码规范、代码复审和结对编程；然后讲到了开发团队和流程，介绍了TSP,MVP,MBP以及RUP的这几种开发流程。学生详细阅读了构建之法的前五章，下文是对书中前五章作者重要观点的总结。

## 一个推论

作者由 `程序 = 算法 + 数据结构` 得到了 `软件 = 程序 + 软件工程` 和 `软件企业 = 软件 + 商业模式` 的推论。程序（算法和数据结构）是基本功，但是在算法和数据结构之上，软件工程决定了软件的质量；商业模式决定了一个软件企业的成败。

## 一个类比

作者将软件事业的发展类比航天事业的发展，指出软件产业也经历过四个阶段：玩具阶段，业余爱好阶段，探索阶段和成熟的产业阶段。对于软件产业来说，从打印出"hello world"开始，就找到了一个很好玩的玩具；然后经过一些已知知识的积累，可以用某些编程语言写出一个网站了；再然后随着知识、经验的沉淀，一些新技术被开发，软件产业持续向前发展；最后随着技术的逐渐成熟，我们有了成熟的银行软件系统、电子商务系统和操作系统。

## 软件工程是什么

作者给软件工程下了一个定义，`软件工程是把系统的、 有序的、 可量化的方法应用到软件的开发、 运营和维护上的过程`。软件工程应该包括：`软件需求分析、软件设计、软件构建、软件测试和软件维护`。软件工程给是一个多学科交叉的知识体系，涉及到了`计算机科学、 计算机工程、 管理学、 数学、 项目管理学、 质量管理、 软件人体工学、 系统工程、 工业设计和用户体验设计`。

## 软件工程中的难题

- 复杂性：软件的各个模块之间相互依赖，随着系统和模块的增多，这些关系的数量往往以几何级数的速度增长；
- 不可见性：软件工程师可以看得见源代码，但是机器在执行机器码，我们无法清除的看到自己的源码如何具体的在机器上运行；
- 易变性：软件看上取好像比较容易修改，但是应对多变的需求，考虑到软件自身的复杂性，正确的修改软件是一件很困难的事情；
- 服从性：软件要运行在操作系统之上，所以要服从系统中的规则，另外还需要遵循行业准则以及用户要求；
- 非连续性：许多软件系统在输入上很小的变化会引起输出上极大的变化。

## 软件工程的目标

目标是创造足够好的软件。Bug少，用户满意度高，可靠性强，软件流程质量好，容易维护。具体对软件工程师的要求是：1）能研发出符合用户需求的软件；2）通过一定的软件流程，在预计时间内发布足够好的软件；3）能证明所开发的软件是可以维护和继续发展的。

## 单元测试

在现代软件开发的过程中，都是团队作业的，那么团队各成员之间的工作有较强的依赖关系，所以为了保证每个开发模块的质量，在开发过程中每隔一段时间都要进行必要的单元测试。作者提出单元测试应该`应该准确、 快速地保证程序基本模块的正确性`。

验证单元测试好坏的标准有：

- 单元测试应该在最基本的功能／参数上验证程序的正确性；
- 单元测试必须由最熟悉代码的人（程序的作者）来写；
- 单元测试过后， 机器状态保持不变；
- 单元测试要快（ 一个测试的运行时间是几秒钟， 而不是几分钟）；
- 单元测试应该产生可重复、一致的结果。
- 独立性：单元测试的运行／通过／失败不依赖于别的测试， 可以人为构造数据， 以保持单元测试的独立性。
- 单元测试应该覆盖所有代码路径；
- 单元测试应该集成到自动测试的框架中；
- 单元测试必须和产品代码一起保存和维护。

## 回归测试

当我们修正了某一个bug后，要做回归测试，这样做可以：1）验证新的代码的却改正了缺陷；2）同时要验证新的代码有没有破坏模块的现有功能。良好的单元测试是回归测试的基础，如果模块的基本功能都不能满足要求，回归测试任务也进行不下去。

## 效能分析

为了让自己的程序跑的又快又好，要进行程序性能分析和改进。在优化我们的程序之前，应该进行程序的效能分析。分析方法有：抽样和代码注入。`如果我们不经过分析就盲目优化，也许会事半功倍`

## 个人开发流程(PSP)

PSP(Personal Software Process)的目的是记录工程师实现需求的效率。

- 计划阶段：明确需求，充分考虑时间和成本
- 开发阶段：
    - 分析需求
    - 编写设计文档
    - 和同事审核设计文档
    - 明确代码规范
    - 详细设计
    - 具体编码
    - 代码复审
    - 测试
- 记录用时：记录每一个开发步骤的消耗的时间
- 测试报告：针对测试编写详细的测试报告
- 计算工作量
- 时候总结
- 提出过程改进的计划

## 个人能力的衡量与发展

初级软件工程师的成长历程：１）积累软件开发相关的知识，提升技术技能；２）积累问题领域的知识和经验；３）对通用的软件设计思想和软件工程思想的理解；４）提升职业技能；５）实际成果。

软件开发的工作量和质量的衡量指标：１）项目/任务有多大；２）花了多少时间；３）质量如何；４）是否按时交付。（实际项目中，稳定、一致的交付时间是衡量一个员工能力的重要方面。）

TSP(Team Software Process)对团队成员的要求：１）交流；２）说道做到；３）接受团队赋予的角色并按角色要求工作；４）权利投入到团队的活动；５）按照团队流程的要求工作；６）做好准备工作；７）理性的工作。

## 软件工程师的思维误区

- 分析麻痹：不要妄图弄清楚所有细节、所有依赖关系后再动手，这样会有无从下手的感觉；
- 不分主次：不经分析，想马上修复所有主要的和次要的依赖问题；
- 过早优化：不要在局部问题上过于纠结花大量时间对其优化；
- 过早泛化：首先需要做的是解决当前特定问题，而不是妄图解决所有类似问题；
- 避免画扇面的情况：踏踏实实干事，停止一味创新的臆想。

## 软件工程师的职业发展

引用书中一位大佬的话：

> 任何事情，当你仔细探究，你就会理解它的量和质；当你对一个领域的神韵足够了解，并开始链接这个领域的表现形式和实现细节的时候，任何一个领域都是会变得隐忍入胜的。

做任何事，从事任何职业都应该保持一颗敬畏之心，端正态度，并付诸实际行动，那么个人的职业前景也会不错的。

知识＋技能

书中的软件工程师职业知识知识评估：https://www.cnblogs.com/xinz/p/3852177.html

## 正确把握学习区

在我们的学习过程中，要正确划分舒适区、学习区和恐慌区，正确把握学习区，踏踏实实，稳步前进。

## 代码规范

- 代码风格规范：主要是文字上的规定，代码风格的原则是简明、易读且无二义。具体在实际项目中应该要本着“保持简明，让代码更容易读”的原则来设计风格规范。书中提到的风格规范有(p70-74)：
    - 缩进：４个空格
    - 行宽：限定100个字符
    - 括号：在复杂的表达式中一定要用括号来表明运算的优先级
    - 断行与空白的 {} 行：每个'{' 和 '}' 都应该独占一行
    - 分行：不要把多条语句放在一行上，特别是不要把多个变量定义在一行上
    - 命名：命名应该能体现变量的类型，还应该简明清晰的表达该变量的用途
    - 下划线：下划线一般用作分割变量名字中的作用域和变量的语义，一般不做他用
    - 大小写：驼峰规则
    - 注释：一定要注意注释要随着代码的更新而变化，避免misleading 的注释
- 代码设计规范：设计牵涉的内容很多，如程序设计，设计模式，模块关联等方方面面。书中提到的设计规范有(p75-84)：
    - 函数：只做一件事，并且要做好
    - goto：函数要有单一的出口，为了达到这个目的，可以使用goto，只要有利用程序逻辑的清晰体现。

- 错误处理：当程序的主要功能实现后，需要给代码加一些错误处理，但是这百分之二十的工作往往需要全部项目百分之八十的时间。
    - 参数处理：所有的参数都要验证正确性
    - 断言：当觉得某事肯定这样时，就可以使用断言
- 处理类：
    - 使用类来封装面向对象的概念和多态
    - 对于有显示的构造函数和析构函数的类，不要建立全局的实体
    - 仅在有必要的时候使用类
    - 按照public、protected、private的测序来说明类中的成员
    - 运算符：有必要时，自定义操作符
    - 异常：了解异常处理的详细机制
    - 继承和委托：尽量使用委托来减少依赖。
- 代码复审：看代码是否规范，看是否解决了问题（最基本的复审手段就是同伴复审，简便易行）
    - 自我复审：用同伴复审的角度来严格要求自己
    - 同伴复审
    - 团队复审：有严格的规定和流程，适用于关键代码

绝大部分的情况下，我们可以找到相似问题的设计方案，我们可以参考其中的编程规范和设计原则。

## 结对编程

极限编程过程中，可以让我们每时每刻都处在代码复审的状态，这样的效率和正确度都很高。结对编程就是一对程序员平等的互补的进行开发工作，一起工作，一起分析和设计，一起测试，一起编码，一起写文档....  结对编程中因为有随时的复审和交流，程序的质量取决于水平较高的那一位程序员。这样会减少程序中的错误，程序的初始质量会高很多，这样就会省下很多以后修改和测试的时间。

怎样进行结对编程 , 作者在书中第87页有提及。总之两人编程需要磨合，如何给予正确的反馈很重要，我们在工作中需要对同伴的工作进行反馈，表达感谢，阐明要求，指出不足等。这样的编程方式虽然很极限，但是如果正确使用，可以极大的提高开发效率。

## 团队和流程

### 1 写了再改模式

- 优点：不需要太多其他准备和相关知识，上来就撸代码，写写改改，也许就出来了。
- 缺点：缺点就是这个方法基本无法解决具有实际需求的软件。
- 用途：
    - 只用一次的程序
    - 看过了就扔的原型
    - 一些不实用的演示程序

### 2 瀑布模型

- 局限：
    - 各步骤分离，但是实际开发过程中各环节紧密相连；
    - 回溯修改很困难甚至不可能；
    - 产品的原型最终才出现，但是我们需要尽早知道产品的原型并试用。
- 适用范围：
    - 产品的定义非常稳定；使用的技术非常成熟；
    - 产品模块之间的借口、输入输出能很好地用形式化的方法定义和验证；

- 负责团队分属不同的机构，不可能做到频繁交流。

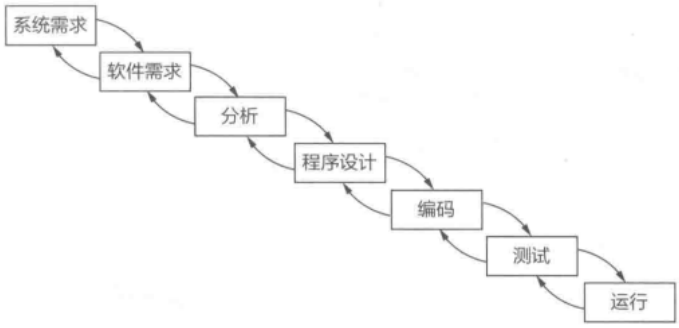


图 5-9　相邻步骤的回溯

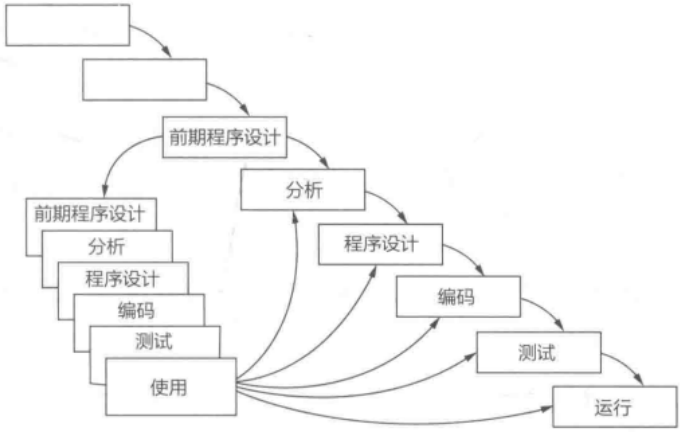


图 5-10　收集反馈并改进

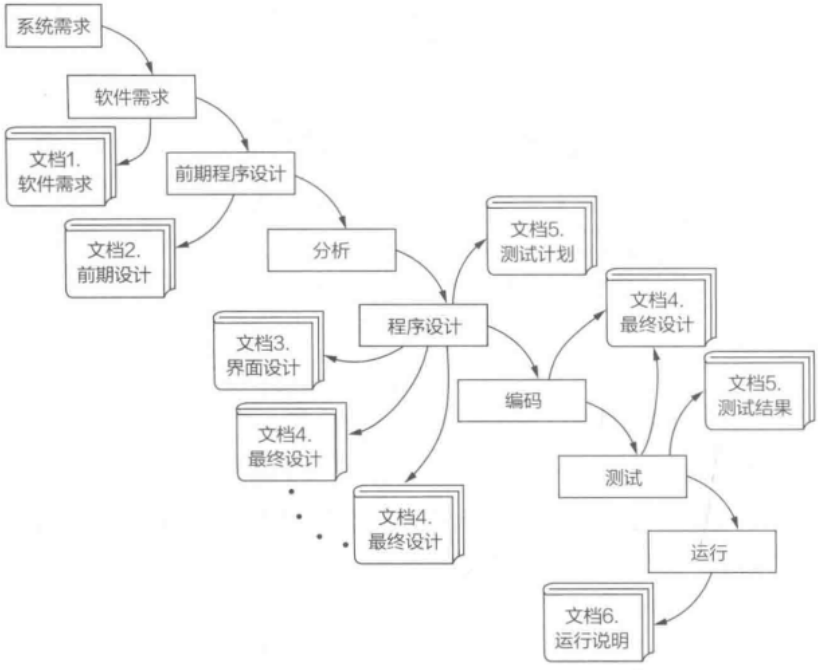


图 5-11　6 种文档

## 3 瀑布模型的变形

- 生鱼片模型

  解决了各个步骤之间分离的缺点，但是无法界定每一个阶段的结束。


图 5-12　生鱼片模型

- 大瀑布带着小瀑布

  解决了不同子系统之间进度不一，技术要求迥异，需要区别对待的问题。但是需要把各个子系统统一到最后做系统测试的阶段，难度大；另外用户只有到了最后才能看到结果，用户等待时间长，不便于反馈。

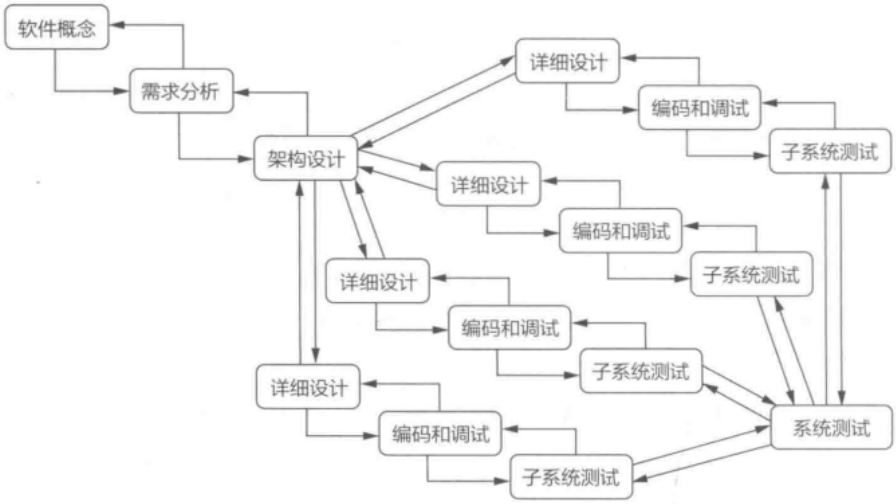


图 5-13　子瀑布模型

## 4 统一流程(RUP)

RUP把软件开发的各个阶段整合在一个统一的框架里。在一个项目里，团队的各种成员要在不同的阶段要做不同的事情，我们称之为Discipline或者Workflow。

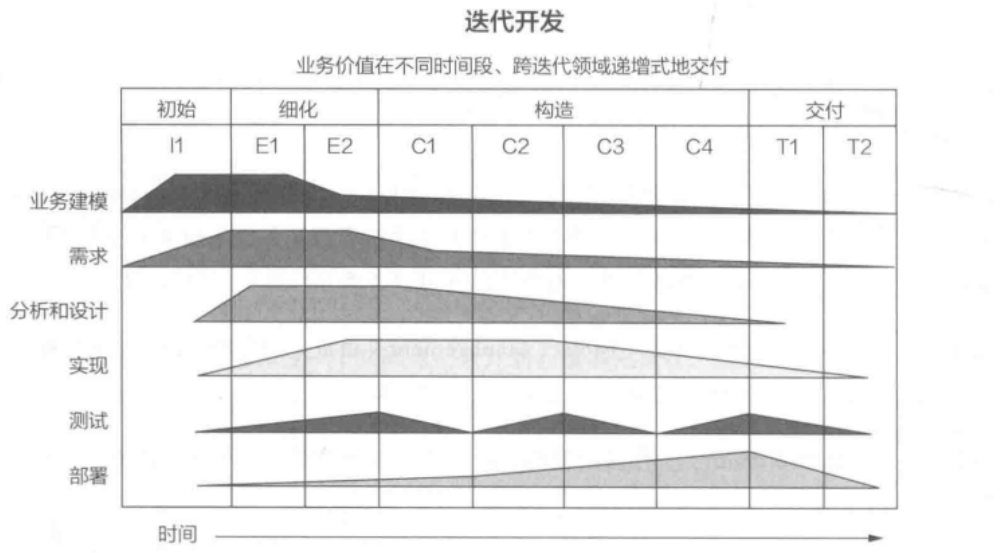


图 5-14　RUP 的工作流（纵轴）和开发流程的各个阶段（横轴），图中的阴影面积代表不同角色在各个阶段的参与程度。由于阴影面积起起伏伏，这个图又被称为 RUP 驼峰图 [14]

## 5 渐进交付的流程，MVP和MBP

- MVP：最小功能子集
- MBP：最强最美产品

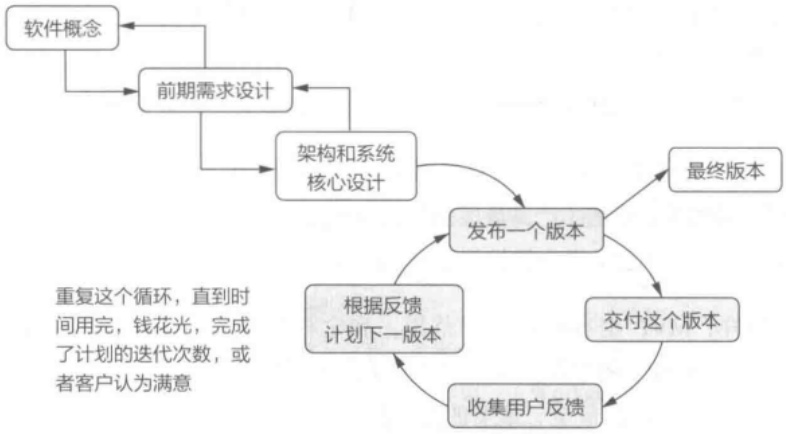


图 5-15　不断演进的 evolution 循环

## 6 TSP的原则

Team Software Process

- 采用标准化作业，使得流程中可以重复的部分有可以衡量的标准；
- 团队每个成员都有对项目的足够理解，包括团队目标，自己的角色定位，产品定位；
- 尽量使用主流开发框架，使用成熟的技术，不要一味求新；
- 收集更多的数据，利用这些数据得到有利于项目改进的信息，团队可以基于这些数据来进行理性的决策，这样走的更稳；
- 团队每个人都要有自我管理的能力，整个team也要采取更加协同的开发方式；
- 详细按照计划，认真落实每一步，出现问题，及时沟通，利用团队的智慧及时修正偏差；
- 小组开发过程中要有严格的质量把关，争取在软件的早期发现问题；
- 设计阶段要做全面细致的设计工作，可以参考已有比较成熟的案例，设计过程中如果出现瑕疵，那么项目开发进程会荆棘密布，难以行进。