Chapter 7 软件工程实践原则Principles that guide practice

※ 考点：给一个例子进行分析

software engineering practice是一系列principle, concepts, methods和tools(方法的工具化，大趋势)

principles that guide practice建立了对软件工程进行的基础

7.1 软件工程知识

现代软件开发技术一般有3年的半衰期；但软件工程principles没有3年半衰期(变化慢）。这些软件工程原则可能在整个职业生涯中为专业程序员提供服务。

7.2 核心原则Core Principles

作用：

1.在流程级别process level，指导软件团队在执行framework activity和umbrella activity时，导航流process flow，并产生一组软件工程work product；

2.在practice level，核心原则建立了一个值和规则的集合，即在分析问题，设计解决方案，实施解决方案并测试解决方案时，该规则是指南，并最终部署软件

7.2.1 引导Process的Principles

对什么样的过程模型Process Model均适用

※可能考选择题

1. **Agile**过程要灵活agile(尽量用敏捷模型开发)

   > *PS：比如对agile model,要求文档不要太多，注重和客户沟通；对非agile model，可以保证每个增量increment之间并行开发，缩短每个增量的开发周期，在人员组织交付工作产品上均灵活些（也就是说，对非敏捷开发模型也有敏捷原则）*

2. **Quality**在每一步产生迭代work product时注重质量

   > review→spike→review→demo，对milestone的work product需要进行评审

3. **Adapt**做好适配的准备

   > 选择process model后可以不断调整以适应项目（如裁减等）

4. **Team**建立高效团队
5. **Communication**建立沟通和协调coordination机制
6. **Change**管理变更

   > 比如：一个类经历了N次迭代，在N+1次迭代时需要修改，则需要管理前n次迭代中它所涉及到的类

7. **Risk**风险评估

> 比如：人员跳槽

8. **Value**产品要有价值

7.2.2 引导实践Practice的原则

软件工程实践具有单一的overriding目标 - 提供可准时，高质量的操作软件，其中包含满足所有利益相关者需求的功能和功能。

1. **D&C**分而治之

> 比如：按业务功能划分

2. **Abstraction**理解和进行分析抽象

> 比如：把requirement用use-case抽象表达；把数据库设计按entity，加上属性找relation，画出E-R图进行抽象，并最后进行代码的抽象

3. **Consistency**前后一致尽量做到
4. **Transfer**注意信息的转换

> 项目开发是一个信息转换过程，比如比如：input->system(本身就是信息转换过程)->output

5. **Modularity** 构建表现出有效模块化的软件(类就是模块化)，即类的定义
6. **Pattern**寻找模式

> 遇到难以解决的问题，寻找已有模式或自己创造pattern，这样是为了提高产品质量

7. **Perspective** 当可能时，从许多不同的角度来看，代表问题及其解决方案
8. **Maintain** 版本控制，保证人员调整后能接上工作

7.3 引导每个framework activity的原则

CPMCD

7.3.1 Communication沟通的原则

对应requirement gathering，communication非常重要

1. **Listen** 听
2. **Prepare** 交流时要准备好
3. **Facilitate** 要有一个负责人来协调推动
4. **F-to-F** 最好面对面交流
5. **Notes** 交流记笔记
6. **Collaboration** 在需求调研时通力协作
7. **Modularize** 专注；模块化你的决策(大家发生冲突时，把发生冲突的部分分解，模块化继续讨论，这点挺重要的)
8. **Graph** 画图抽象
9. **Move-on** 同意，或不同意，或有不清晰现阶段无法弄明白的地方，继续并记录在案日后解决；
10. **Win-Win** 沟通不是竞赛，追求双赢

7.3.2 Planning计划的原则

过度计划浪费时间效率，但缺少计划会混乱

软件团队中的每个人都应该参加规划活动

1. **Scope** 确定项目的范围和边界才能planning
2. **Stakeholder** 要和甲方紧密交流，囊括进计划
3. **Iterative** planning是迭代的
4. **Estimate** 基于已知的来进行估算
5. **Risk** 计划时需要考虑风险

> 对于已经识别的风险，把风险管理作为一个任务在计划中体现；对于未识别的风险，留一定的缓冲buffer空间（风险+预备处理方案）

6. **Realistic** 现实点
7. **Granularity** 做计划时要定义计划的粒度，并调整粒度

> 比如：一个大任务→分成n阶段→阶段分成子阶段→细分小粒度task

8. **Quality** 制定计划以保证质量（比如:review)
9. **Change** 描述如何适应变更

> 比如：敏捷模型可以更好适应变更

10. **Track&Adjust** 跟踪计划，并根据需要实时调整计划

> 比如：敏捷模型明天都要进行跟踪(燃尽图)

7.3.3 Modeling建模原则

在软件工程工作中，可以创建两类模型：需求模型和设计模型。需求模型（也称为分析模型analyse model）代表客户要求通过将软件描绘在三个不同的域中：信息域(数据建模UML)，功能域（Activity Diagram）和行为域(State和Sequence Diagram)。设计模型包括：概要设计&详细设计

即：数据建模，功能建模，行为建模

总体原则：

1. **Software** 软件团队的主要目标是构建软件，而不是创建模型，模型不用太过细致
2. **Light** 要多少才建模多少，少建模型，开发过程轻量级
3. **Simple** 模型尽量简单
4. **Amenable** 模型要易于在不同迭代过程中进行变更（变更时要做版本控制，是前面提到的）
5. **Purpose** 模型都要有清晰的目的
6. **Adapt** 调整模型，适应待开发系统
7. **Useful** 建立实用的模型，不必完美（强调敏捷，为下一阶段做准备）
8. **Dogmatic** 构建模型的方法不必死板，不用全部按照规范，只要能辅助理解需求即可；如果它成功通信了，怎么表示是次要的
9. **Instinct** 直觉告诉你模型不太ok时，仔细检查
10. **Feedback** 及时反馈（通过daily meeting等）

具体来说，分两个：需求建模和设计建模

1️⃣需求建模原则

1. **Domain** 首先要进行数据建模产生类图
2. **Functions** 定义好软件功能。即：需要进行功能建模

> eg:下订单的功能，在类图中找到涉及的类，用活动图表示逻辑，再用泳道图表示逻辑中涉及的类

3. **Behavior** 行为建模

> 借助State Diagram（注重State Transition) 和时序图Sequence Diagram

4. **Layered** 尽可能细化分层
5. **Detail** analyse的task应该从基本信息转向实施细节，为后面设计建模和代码服务

2️⃣设计建模原则

设计模型为软件创建提供各种不同的系统视图(不同角度去理解系统，帮助coding)

设计建模：

- 概要设计建模
  - 软件体系结构设计
  - 数据设计
    - 数据库
    - 数据仓库
    - 数据结构
    - 数据字典
  - 接口设计
- 详细设计建模（为每个类每个方法进行设计）

1. **Traceable** 设计应追溯到需求模型（和前面的内容前后一致，Strive for consistency)，也就是说，设计建模以分析建模作为依据
2. **Architecture** 概要设计中的软件体系结构（看上面）设计十分重要
3. **Data** 数据设计（看上面）与功能设计一样重要
4. **Interface** 内部接口和外部接口设计
5. **UI** 用户界面设计，要求简易且面向用户需求（一般用prototype model)
6. **Component** 组件设计应该在功能上独立
7. **Loose-coupled** 组件之间，组件与外部环境之间松散耦合

> 比如：方法调用中形参，实参尽量简单而少，模板之间的参数尽量简单

8. **Understandable** 设计模型应该容易理解
9. **Iterative** 设计应迭代进行
10. **Agile** 设计模型不应该拖累排除敏捷方法

7.3.4 Construction构建的原则

复习：Construction

- coding
- testing

- unit testing单元测试
- integration testing集成测试
- system testing系统测试(又叫validation testing)
- acception testing验收测试

1️⃣coding原则

要关注编程风格，语言方法集等

A. 写代码前：

1. **Problem** 理解问题（详细设计）
2. **Principles** 理解基本的设计原则和概念（看详细设计对不对）
3. **Language** 语言选择，满足要构建的软件的需求以及它将运行的环境
4. **Environment** 环境选择（IDE等）
5. **Test** 写代码前把测试用例写进去，测试先行（单元测试Unit Testing先创建）

B. 写代码时

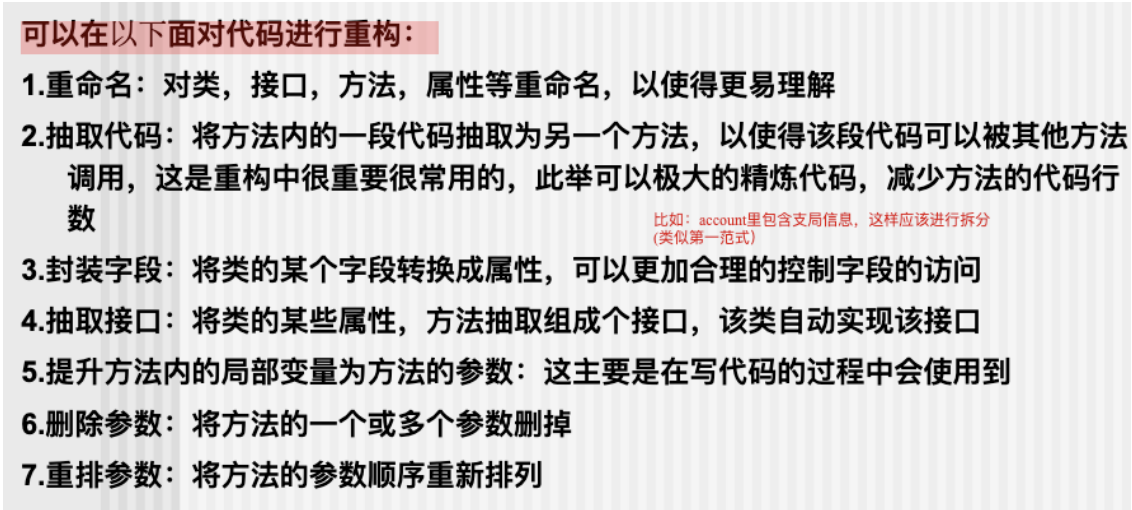
1. **Structured** 结构化编程以写算法（结构化编程就是条件循环语句等）
2. **Pair** 结对编程
3. **DataStructure** 选好数据结构，要满足设计需求
4. **Arch&&Interface** 了解软件体系架构，创建与之一致的接口（也就是概要设计部分的文档内容）
5. **Simple** 尽可能保证条件逻辑简单
6. **Nested** 嵌套循环不要太深，要保证嵌套循环的可测试性（尽量避免，但不可避免时要使得嵌套循环可以较简单地被测试）
7. **Name** 变量命名要规范有意义
8. **Document** 要有注释（不用太细）
9. **Layout** 代码布局要好，增强代码的可读性（比如缩进空行等）

C. 写代码的第一个代码版本完成后

1. **Walkthrough** 进行一次代码审查

> walkthrough: 走查。只读评审(e.g. Review)的一种方法，如一行一行地review

2. **Test** 单元测试，手机错误
3. **Refactor** 重构代码



2️⃣testing原则

主要指动态测试

测试的要求：

1. **Definition** 测试是执行程序的过程，其中包含查找错误

   ↓(good)

2. **Probability** 一个良好的测试用例可以高概率找到尚未发现的错误

   ↓(successful)

3. **Uncover** 成功的测试是揭示尚未发现的错误的测试

原则

1. **Traceable** 所有测试都应追溯到客户要求Requirement
2. **Long** 在测试开始之前，应长期计划测试。
3. **Pareto** 帕累托原则适用于软件测试
4. **SmallToLarge** 测试应该开始从“小”中并进展到测试“大” (输入不可能覆盖全部情况，从小开始)
5. **Impossible** 穷尽Exhaustive的测试是不可能的
6. **Module** 应用于系统中的每个模块，测试工作与其预期的故障密度fault desity相称

   > 假设一千行代码有3-5个fault(也就是fault density故障密度，为度量值)，测试的时候是7-8个，则说明超过了度量值，是有问题的

7. **Static** 静态测试技术static testing techniques可以产生高结果
8. **Pattern** 跟踪缺陷并寻找通过测试未覆盖的缺陷模式。也就是说，这些缺陷是否满足一些pattern
9. **Correctly** 也要考虑演示软件行为正确的测试用例

7.3.5 Deployment部署原则

复习：

部署包括

- delivery 交付
- support 支持
- feedback 反馈

regime：运维团队

1. **Expectation** 处理好客户的预期
2. **Assembled** 一个完整的部署包delivery package必须被汇集并测试
3. **Support** 必须在软件交付之前建立支持制度support regime
4. **Material** 必须向用户提供教学材料（文档）
5. **Bug** 先修bug再deliver

7.4 Work Practice

Isklod建议了超越编程语言和特定技术的10个概念，十分重要的特征。其中一些概念形成了欣赏软件工程在软件过程中的作用所需的先决知识（基础知识）

1. Interface （内部接口，外部接口和用户接口(界面)）
2. Conventions(规范) and templates (编程规范，功能、数据、行为建模的规范等；文档的模板，UseCase图构建的模板)
3. Layering (分层，抽象化)
4. Algorithmic complexity

5. Hashing
6. Caching
7. Concurrency
8. Cloud computing
9. Relational database