Chapter 2 软件工程Software Engineering

开发软件前必须知道的事实：

- 要明确问题
- 设计是关键活动
- 软件必须展现高质量
- 软件必须可维护

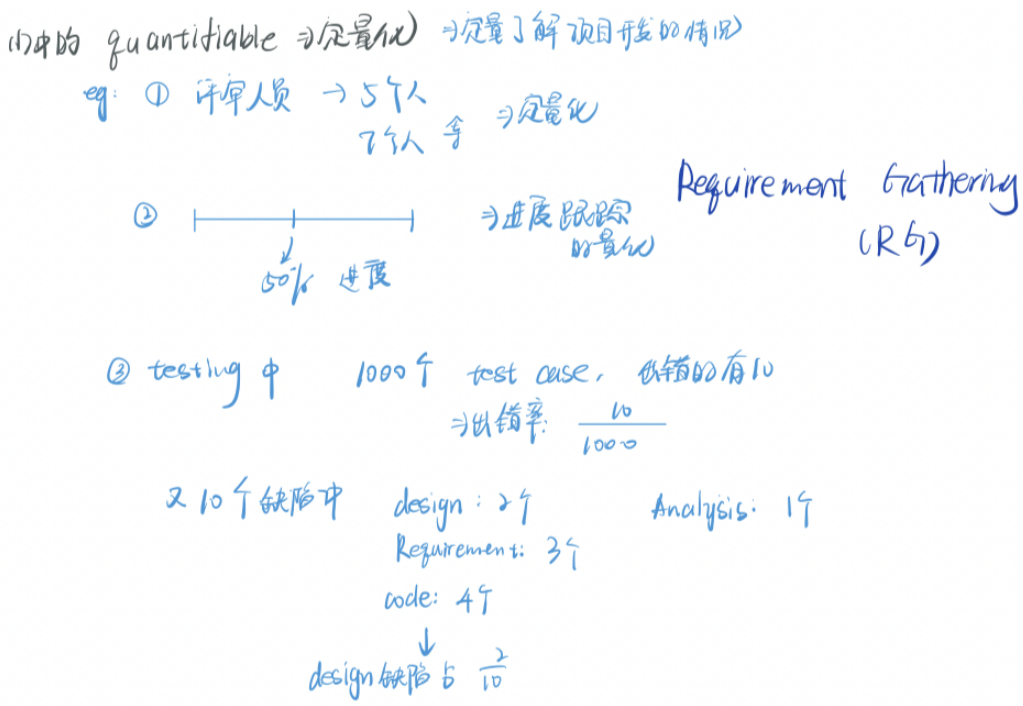
质量保证的一些做法:**review, SQA（software quality application）(使用checklist)**

2.1 定义

※ IEEE 的软件工程定义：

1. SDQ(系规量) 将系统的systematic、规范的disciplined(遵守各种规则，如设计模式，编码规则等）、可量化quantifiable（软件工程里最重要的概念之一）的方法应用于软件的开发、运行和维护； 也就是说，**工程在软件中的应用The application of engineering to software，以开发，使用，维护软件**
2. 对(1)中的研究方法

可量化quantifiable的例子(定量了解项目的开发情况）：三个人公司实习，把设计转为代码，系统对你们代码量测试，发现每一个人千行代码平均错误为20、10、5个，这就是通过软件工程思想进行可量化管理

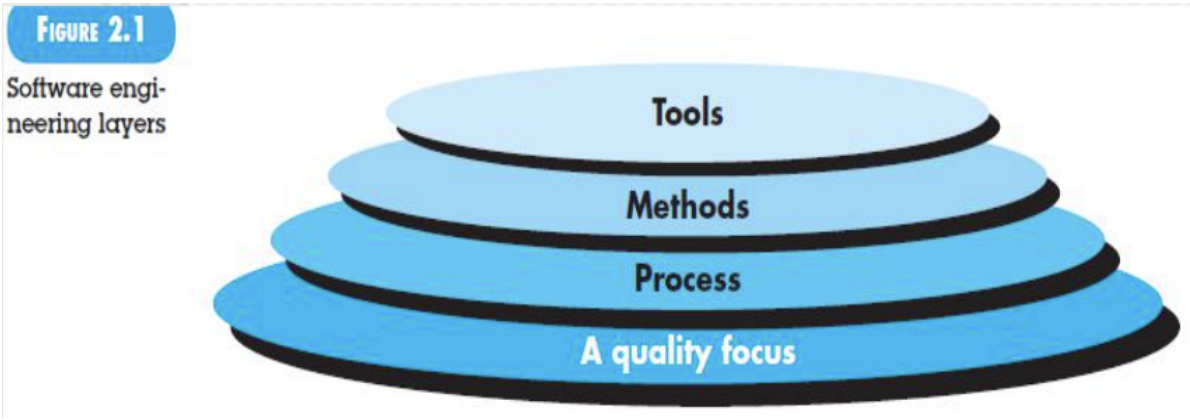


把所有软件工程的工具通常为CASE(计算机辅助软件工程）： computer-aided software engineering

为过程和方法提供自动化或半自动化的支持

※软件工程层次

对软件工程进一步的抽象分成4个层次：



- Q质量: 为什么要有软件工程⇒因为要提高软件的质量，***质量是根基***
- P过程化：回答了如何提高质量的问题
- M方法：方法贯穿于过程中，eg：**UseCase建模，数据库设计方法，架构方法，分析方法，设计方法，测试方法**等多种方法
- T工具：现在很多方法和技术都工具化了

2.2 软件过程Software Process

（重要）

对Process的分解：Process—细分为若干→Activity—细分为若干→Action—细分为最小的→task

task是最小单元，原子性，高内聚

粒度划分：

- Activity: 力求实现一个广泛的目标——e.g：stakeholder的沟通
- Action: 包含一组产生主要工作产品的任务——e.g：架构设计；一个action要有主要工作产品work product(e.g 架构模型)
- Task: 侧重于产生实际结果的小而明确的目标——e.g：单元测试

提问：不同activity, activity的不同action，action的不同task之间的关系?(start-start/start-finish/finish-start)

回答：实际情况下，三个关系都有可能发生

一个例子：

Activities:RG过程的一个Activity

Actions: 小的project，比如上报需求到负责人，开一次腾讯会议

Tasks：比如：制定会议室时间等

架构的三大任务：体系架构设计，接口设计，数据与数据结构设计

2.2.1 过程框架Process Framework

两大过程：

- 框架构成Framework Activity: 正常的开发活动（需求调研，需求分析，设计，coding等）
- 伞型活动Umbrella Activity

milestone：里程碑，经过framework activity和umbrella activity后可以交付给下一个阶段

※软件工程的通用过程框架包括五项活动: **CPMCD**

1. 交流communication：需求收集requirement gathering& 需求模型诱导 elicitation

2. 计划Planing：planning是一个大的，阶段的计划（如人员安排等），有很多template
3. 建模Modeling：包括分析建模(Analysis modeling)，概要设计(high-level design)， component design 组件设计(detailed design),概要设计(high-level design)
4. 构建Construction：coding & testing 写代码&测试,测试包括单元测试，系统测试
5. 部署Deployment：delivery & maintenance & feedback 交付，维护，反馈

   软件开发过程包括很多迭代，并不是线性顺序的模型，而是循环往复

※ **Standard Process包含的Activity**

1. **requirement elicitation 需求诱导**
2. **requirement analysis modeling 需求分析建模**
3. **architecture design 架构设计**
4. **component design 组件设计**
5. **coding 写代码**
6. **unit testing 单元测试**
7. **integration testing 集成测试**
8. **system testing 系统测试**
9. **acceptance testing 验收测试（用户在开发环境下的测试）**
10. **Release & Delivery，Feedback & Support 发布，部署，反馈，支持**

对应CPMCD：

Communication: 1

Planning:(umbrella activity)

Modeling: 2、3、4

Construction: 5、6、7、8、9 coding-5 testing 6-9 （UISA)

Deployment:10

2.2.2 伞型活动Umbrella Activities

作用：

保障development activity有high quality，贯穿整个项目，不是单独活动

控制进度，保障质量，规避风险

导致changing的三个原因：

- corrected:
- adaptive: 适应性修改，比如法律等
- enhanced: 增强功能，性能等

典型的伞活动：

1. **Software Project Tracking & Control** 软件项目跟踪控制（如：人员计划，发挥成员特长能力，了解人员是tracking，调整人力是control）
2. **Risk Management** 风险管理(quantity,是定量管理,risk exposure风险曝光度=probability(0-1)**loss(0-10)概率**损失(例如，人员流失跳槽风险，新技术，云平台不熟悉，数据库不熟悉等...))

   (如何想到的：经验，原有项目，头脑风暴)，算出的风险从高到低排序，并制定方案.

   风险管理三个阶段：risk management是task但也是umbrella activity

   1. I risk identification: 风险定位（头脑风暴，相关经验，专家建议）

2. M risk mitigation: 风险缓解
3. T risk tracking: 风险追踪

风险管理：eg：拼多多经常发布新版本，新版本发布前要进行测试
⇒ 要做风险管理。

又如在开发过程中也要做风险管理

需求分析 建模时评审方向不对 ⇒ 很多缺陷没有被发现 ⇒ 风险
在此处做错 后面会走越来越多

风险管理之风险识别的方法 (risk identification)

1. 头脑风暴
2. 跟团队类似项目的经验
3. 专家建议 (experts advice)

形成一个 list

↓ 还可计算

risk exposure 风险的暴露度。

⇒ risk exposure = probability × loss
可能性 × 损失

eg: probability: 0.5, loss: 5

risk exposure = 0.5 × 5 = 2.5

⇒ 然后把 risk 按 risk exposure 从大到小排序

风险缓解 (risk mitigation)

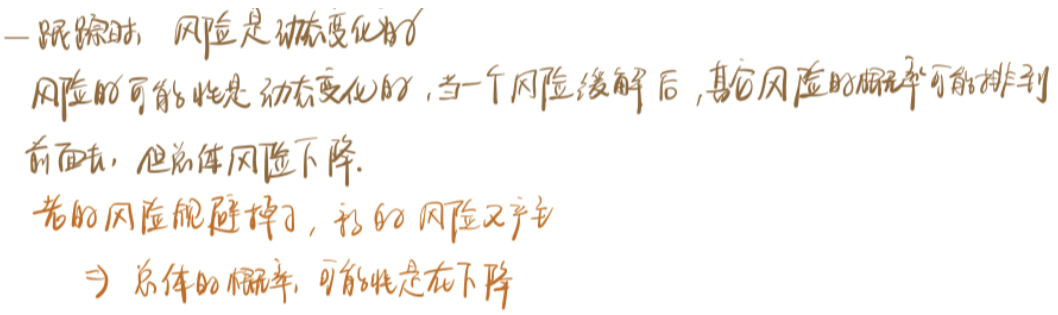
eg: 新版本回归测试中，用把之前的测试用例跑一遍

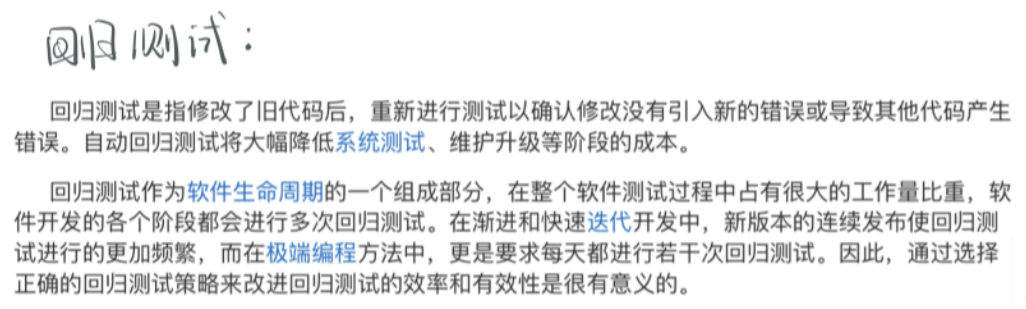
针对哪个接口用哪些用例 ⇒ 根据规约进行查询，找出对应的测试用例

再针对新的接口增加一些测试用例 ⟶ 风险缓解

又如为了提高质量 ⇒ 工具的购买，使用，培训 ⇒ 风险缓解

风险跟踪 (risk tracking)





3. **Software Quality Assurance** 软件质量保证——定义和实施确保软件质量所需的活动 ;(编程管理，代码规范，需求规约等）有Checklist
4. **Technical Review** 技术审查——特殊的quality assurance (比如代码Review，需求Review等)
5. **Measurement** 测量；对schedule进行测量，对不同时间和schedule的偏差进行度量，对发现的缺陷进行度量，对人员工作量进行度量等
6. **Software configuration** 软件配置管理；比如用 Github 进行管理
7. **Reusability Management** 可重用性管理(苹果的复用做的很好，通过标准化可以降低成本)---前提是重构，让代码和类标准化(可以复用)；
8. **Work product preparation & production** 工作产品准备和生产——包括创建工作产品所需的活动，例如模型、文档、日志、表格和列表。

区分:measurement和metric----发现errors,ABCD任务errors计数是measure; ABCDE任务的errors求平均数等数学模型处理是metric（例子:项目进度,安排4个人分工协调，到时间时测量每个人完成度是measure，计算总体完成度是metric，以此制定下阶段目标)

（TODO）

2.2.3 流程调整Process Adaptation

(被调整和被裁减）应该是敏捷和适应性强的（针对问题、项目、团队和组织文化）。 因此，一个项目采用的过程可能与另一个项目采用的过程有很大不同

考试→给例子，应用概念来描述

※以下的活动可以在不同项目中做适应性调整

- activity, action, task的总体流程(flow)以及它们之间的相互依存关系
- action和task在每个framework activity中的定义程度
- work product的定义要求的差别
- QA质量保证Activity实施的方法
- 项目跟踪控制和控制类Activity实施的方法
- 过程progress描述的的严格和详细的程度
- stakeholder的参与度（可能是使用完反馈，也可能是直接干预开发）
- 团队的自主程度

- 类似项目有没有类似经验

(TODO)

2.3 软件工程实践

2.3.1 实践的精髓(Essence of practice)

4步:

- 1.理解需求Understand the problem——
  - 交流(requirement gathering)
  - 计划(planning)

  实质：需求调研，需求工程(requirement engineering)

  具体问题:

  1.谁是项目的**利益相关者**;

  2. **不确定因素，数据、功能和特性**（特性还包括一些非功能需求，如安全性，performance，都很重要);
  3. 问题能否划分，需求能否分解，**减小粒度**(在UML里和Use Case,Class Diagram相关)；

  4.问题可以**可视化表现**吗(图)，**分析模型(analysis model)**能否建立;
- 2.计划解决方案Plan a solution——
  - analysis
  - design modeling(Use Case, Class Diagram, Sequence Diagram, Activity Diagram, State Diagram)
  - 核心是：**Architecture Design & Component Design**

  概况：根据上一个阶段的结果，进行建模（数据，功能等角度），提供一种解决方案（比如：框架，接口，界面，数据库的设计）

  具体问题：

  1. 有无**相似的模式**可以复用，（设计模式，相似项目，数据，功能，非功能需求等)；

  2.**类似的问题有无解决方法**，有无**可复用**的解决方案元素；

  3. 子问题能否被定义，**子问题**的解是否显而易见的（软件架构）；

  4.能否以一种能够有效实施的方式来**表示解决方案**？能否创建**设计模型(design model)**？
- 3.实施解决方案Carry out the plan
  - code generation, implementing solution 代码生成

  具体问题：

  1.解决方案是否符合计划？ 源代码是否可以追溯到设计模型（你的**代码要对应设计架构**，比如说，哪部分代码对应了某功能的微服务）

  2.解决方案的每个组成部分(component)是否**正确**？ 设计和代码是否已经过**审查（review 是 Umbrella Activity）**，或者更好，是否已将正确性证明应用于算法
- 4.检查结果精确度Examine the result for accuracy
  - testing 测试
  - quality assurance 质量保证

  具体问题:

1.是否可以**测试**解决方案的**每个组成部分?**(单元测试,每个类都有状态图State Diagram,利用状态图进行测试)，是架构设计的重要依据) 是否实施了合理的**测试策略(testing strategy)**

2.解决方案是否产生符合所需数据、功能和特征的**结果**？ 软件是否根据所有利益相关者的要求进行了**验证(**validate->也叫system testing系统测试)?

2.3.2 一般原则General Principles

1. V存在价值—— 项目的价值是什么，能不能带来生成效益
2. S保持简洁KISS—— 架构简单清晰明了
3. V保持视图—— 保持每个阶段可见，可视化，using类图或其他工具
4. U关注使用者——软件做完后有没有受众面，对客户有作用（输入输出）
5. F面向未来——可扩展性，比如,接口不写死, 类便于修改维护
6. R**提前计划复用——**代码尽量标准化，提前考虑组件复用性前瞻性，降低成本
7. T认真思考

2.4 软件开发误区Myth

管理误区：针对维持预算、防止进度延误和提高质量的压力

Myth-Reality

1. 即使有软件工程这个开发框架也可能会遇到很多问题

customer myth

1. 需求不能随意更改，敏捷开发中需要等到下一个迭代周期才可以更改需求（stories）

Practitioner’s myths

1. 代码要写的慢而精
2. review SOA去找error
3. Software engineering is not about creating documents. It is about creating a quality product.

ppt_c2看风险管理，myth部分