软件工程概念整理

Part 1 概述

1. **软件危机：**指计算机软件的开发和维护过程中所遇到的一系列严重问题
2. **软件：**指令的集合，数据结构，软件描述信息（程序+数据结构+文档）
3. **软件工程：**
   - (1)将系统的(systematic)，规范的(disciplined)，可量化的(quantifiable)方法应用于软件的开发、运行和维护，即将工程化方法应用于软件
   - (2)对(1)中所述方法的研究
4. **软件工程分层：**
   - 根基在质量关注点
   - 基础是软件过程层，软件过程把各个技术层次结合在一起，使得合理、及时地开发计算机软件成为可能
   - 软件工程方法：为构建软件提供技术上的解决方法(沟通，需求分析，设计建模，程序构造，测试，技术支持)
   - 软件工程工具：为过程和方法提供自动化或半自动化的支持

Part 2 软件过程

一 软件过程结构

1. **软件过程：**是开发高质量软件产品的一系列相关活动的集合
2. **软件过程的目的：**为各种人员提供一个公共框架，以便用相同语言交流
3. **过程框架：**定义若干框架活动，为实现完整的软件工程建立基础，包括：沟通，策划，建模，构建，部署。一系列**普适性活动**：项目跟踪控制、风险管理、质量保证、配置管理、技术评审及其他活动贯穿软件过程始终。
4. **过程流：**描述在执行顺序和执行时间上如何组织过程框架中的活动：线性流、迭代流、演化过程流(循环)，并行过程流
5. **实践的精髓：**
   - 理解问题(沟通和分析)
   - 策划解决(建模和软件设计)
   - 实施计划(代码生成)
   - 检查结果的正确性(测试和质量保证)
6. **实践通用原则：**
   - 存在价值
   - 保持简洁
   - 保持愿景
   - 关注使用者
   - 面向未来
   - 提前计划复用

- 认真思考

7. **软件过程4个基本活动：**
   - S**oftware specification：定义系统应该做什么**
     - 软件规格需求说明：理解和定义用户需要什么样的服务，确定系统运行和开发时的约束
     - 需求工程过程：
       - Feasibility study：构建系统在技术和经济上是否可行？
       - Requirements elicitation and analysis：用户对系统有哪些要求和期望？
       - Requirements specification：详细的定义需求
       - Requirements validation：验证需求的一致性、正确性
   - **Software design and implementation：定义系统架构和实现**
     - 把软件的规格说明转变为可执行的系统
     - 软件设计：设计出实现了需求规格说明的软件结构
     - 实现：软件结构变成可执行的程序
     - 设计和实现紧密关联，可以交替执行
     - **两个设计：**
       - 总体设计：概括的说应怎样实现目标系统；体系结构
       - 详细设计：具体实现；细化概要设计所生成的各个构件，详细描述程序模块的内部细节
     - **implementation：**
       - 软件设计转成程序代码
       - 方法：以详细设计规格说明书为依据，基于某种程序设计语言编码
   - **Software validation：检验系统是否满足用户需求**
     - 系统符合规格说明，满足用户的期望
     - 包含：检验、审查、测试
   - **Software evolution：随用户需求的变化变更系统**
     - 软件灵活，经常变更
     - 软件维护的开销占整个项目总成本比例较大

二 软件过程模型

(一) 惯用过程模型

1. **建造-修正模型(Code-and-Fix)：**
   - 没有需求分析、设计，直接编码
   - 优点：简单
   - 缺点：难以维护，大规模项目采用很危险
   - 适用：小规模(<200行)且无需维护程序
2. **瀑布模型(Waterfall Model)：**
   - 传统、V模型
   - 系统的，顺序的软件开发方法
   - 适用：需求已准确定义和相对稳定的项目
   - 问题：很少遵循瀑布模型，因为客户难以描述清楚所有需求，在项目尾声客户才能得到可运行的系统

- 解决方法：开始就展示一个目标系统的雏形，让用户评论，逐步修改，直到成功；原型开发

3. **演化模型：**
    1. 原型开发(Prototyping)：
        - 沟通：定义软件目标，明确需求
        - 策划：迅速策划一个原型开发迭代
        - 建模快速设计：集中在最终用户能看到的方面
        - 构建：快速设计产生一个原型
        - 部署交付及反馈：利益相关者评估，根据反馈进一步精炼软件的需求
        - 优点：快速迭代开发，可以不考虑响应时间、错误处理、可靠性、程序质量等。
        - 适用：需求模糊的系统
    2. 螺旋模型：风险驱动型过程模型
        - 循环方式加深系统定义和实现深度
        - 确定一系列里程碑作为支撑点
4. **增量模型：**
    - 适用：初始的软件需求明确，需要向用户快速提供一个有限功能的软件，在后序版本中细化和扩展功能
    - 混合线性和并行过程流，每个增量内部线性，增量过程之间并行
    - 第一个增量为核心产品
    - 优点：用户可以更早期使用软件，软件核心功能得到更充分的测试
5. **并发模型**

（二） 专用过程模型：

1. 基于构件的开发
2. 形式化方法模型
3. 面向方面的软件开发

（三） 统一过程(RUP，Rational Unified Process)

“用例驱动，以架构为核心，迭代并且增量”

1. **3个视图：**
    1. 动态视图(分4个阶段)
    2. 静态视图(过程活动，工作流)
    3. 实践视图(最佳实践)
2. **4个阶段：(阶段内迭代，跨阶段迭代)**
    1. 起始阶段：
        - 与涉众合作，定义系统业务需求(用例)
        - 提出系统大致架构(子系统及功能)
        - 制定开发计划
    2. 细化阶段：
        - 精化、扩展初始用例
        - 扩展体系结构(分析、设计模型)
        - 评审、修订项目计划
    3. 构建阶段：
        - 系统开发和测试
        - 基于分析、设计模型实现构件
        - 单元测试、集成测试、验收测试

4. 转换阶段：
    - 运行环境中部署系统
    - 准备用户手册、问题指南、安装手册等
3. **RUP工作流：**
    1. 核心工作流：业务需求(业务用例模拟业务过程) -> 需求(用户需求) -> 分析与设计 -> 实现(编程实现构件) -> 测试 -> 部署
    2. 核心支持工作流：配置与变更管理；项目管理；环境
4. **RUP最佳实践：**
    1. 迭代式开发：
        - 若干渐进反复过程中得出有效解决方案的迭代方法
        - 先开发优先级高的特性
    2. 管理需求：
        - 获取用户的需求并文档化
        - 跟踪需求变更
    3. 使用基于构建的体系结构：
        - 使用可服用构建来组织系统的体系结构
    4. 可视化建模：
        - 用UML建立可视化模型
    5. 验证软件质量：
        - 软件质量评估内建整个开发过程所有活动中
        - 确保软件产品满足组织质量标准
    6. 控制软件变更
        - 使用变更管理系统和配置管理工具管理软件的变更

三 敏捷开发

（一） 敏捷方法

1. 特点：
    1. 让软件开发团队具有快速工作、响应变化能力的价值观和原则
    2. 是一种思维方式和软件过程方法论
    3. 以人为核心、迭代、增量式的开发方法
    4. 强调软件本身，胜于设计和文档
    5. 基于迭代
    6. 尽快交付可工作的软件产品
    7. 尽快变更软件以满足变化的需求
2. 原则：
    - 客户参与：参与整个开发过程，提供新需求，评估增量
    - 增量交付：确定增量中需求，增量规模小，快速交付
    - 人胜于过程：技术，不拘泥特定过程
    - 拥抱变更：快速响应变更
    - 保持简单：开发过程简单

3. 敏捷三个层次：
    - 理念(敏捷核心思想)
    - 优秀实践(敏捷经验积累)
    - 具体应用

(二) 敏捷优秀实践

1. 技术实践(极限编程，XP)：
    1. 用户故事：根据用户的描述确定用户的需求，并确定需求的优先级，开发团队评估每个需求所需开发周数，将超过3个周数的需求进行细分，重新确定优先级计算成本。
    2. 重构：不改变代码外在行为前提下，对代码做修改，改进程序内部结构
    3. TDD：测试驱动开发
        1. 用户参与测试开发和确认的过程
        2. 使用自动化测试框架
        3. TDD开发流程：编写测试用例 -> 运行测试用例不通过 -> 编写代码 -> 再次执行用例通过 -> 进行重构 - > 执行所有用例通过
        4. 要点：
            - 测试代码简洁，可读性好
            - 测试用例设计完备，覆盖所有功能
            - 功能单元较大，可以分解成小单元逐一用TDD测试
    4. 结对编程：一个考虑设计特定部分的编码细节，另一名确保编码遵循特定标准
    5. 持续集成：团队成员经常集成工作，每个成员每天至少集成一次，每次通过自动化的构建来验证
        - 一次集成时间尽量短，提供完备有效的反馈信息
        - 修复识失败的构建时最高优先级任务
        - 开发人员本地构建成功再提交版本库
        - 开发人员每天至少向版本库提交一次代码
        - 每次构建100%通过测试
        - 持续集成的状态要实时显示给所有人
2. 管理实践(Scrum)：
    1. PO(product owner)
        - 确保team做正确的事
        - 代表利益相关人，对产品投资回报负责
        - 定义产品需求并确定优先级
        - 重构、持续集成环境搭建等内部任务
    2. Scrum Master(Scrum 教练)
        - 确保team做正确的事
        - 辅导团队正确应用敏捷原则
        - 引导团队建立并遵守规则
        - 激励团队
    3. Team
        - 负责产品需求实现
        - 估计工作量
        - 保证交付质量
        - 向PO和利益相关人员演示工作成果

        - 团队自我管理、持续改进
    4. 每日站立会议
        - 准时开始
        - 高效会议(15min，站立)
        - 问题跟踪
3. 团队(Scrum)

Part 3 建模

一 分析

(一) 理解需求

1. **需求工程概念**
    1. 概念：对系统应该提供的服务和所受到的约束进行理解、分析、建立文档、检验的过程
    2. 需求概述：
        - 确定目标系统功能
        - 需求重要性：软件开发的前提和基础；最终目标软件系统验收的标准；避免或尽早剔除早期的错误
    3. 需求工程：
        1. **需求开发**
            - 需求获取
            - 需求分析
            - 编写需求规格说明
            - 需求确认
        2. **需求管理**
            - 需求跟着
            - 需求变更控制
            - 版本管理
            - 需求复用
2. **需求的两个层次**
    1. 用户需求
        - 描述系统能帮用户做什么
        - 建立用户角度的需求
        - 从用户角度描述系统功能和非功能需求，只设计系统外部行为
        - 使用自然语言和图描述
    2. 系统需求
        - 关于软件系统功能和运行约束的更详细的描述
        - 定义系统需实现的功能，描述开发人员需要实现什么
        - 合同的一部分
        - 一个用户需求扩展多个系统需求
3. **需求的两种类型**
    1. 功能需求

        - 系统应提供的功能和服务
        - 具备完整性和一致性
        - 困难
    2. 非功能需求
        - 系统的特性和约束：性能、可靠性、安全性、可用性
        - 比功能需求更关键
        - 分类：
            - 产品需求
            - 组织需求
            - 外部需求
4. **需求获取技术**
    1. 获取信息的来源：文档、涉众、相似系统的规格说明
    2. 获取技术：
        1. 访谈
            - 正式和非正式
            - 封闭式和开放式
        2. 场景
            - 和涉众交流，确定场景
            - 获取包含在场景中的详细信息
        3. 用例
            - UML用例图
            - 用例描述一种交互
            - 每个用例就是一个场景
        4. 原型
5. **需求规格说明SRS**
    1. 用户需求系统需求文档化过程
    2. 用户需求不包含技术信息
    3. 系统需求包含技术信息，是系统设计的基础
    4. SRS依赖于所开发系统的类型和开发过程模型
6. **需求验证**
    1. 验证需求的正确性
        - 一致性
        - 完整性
        - 现实性
        - 有效性
    2. 需求验证技术
        - 人工审查
        - 原型
7. **构建需求模型**
    1. 模型从不同视角描述系统
        - 交互视角：用户交互，系统构件间交互
        - 结构视角：组织结构
        - 行为视角：动态行为

2. 目标：
    - 描述客户需要什么
    - 为软件设计奠定基础
    - 定义软件完成后可以被确认的一组需求
3. 建模方法：
    - 结构化分析
    - 面向对象分析

（二） 结构化分析(SA)

定义:用抽象模型的概念，按软件内部数据传递、变化的关系，自顶向下逐层功能分解的系统分析方法定义系统的需求

1. 数据模型(ER)
2. 功能模型(DFD)
3. 行为模型(状态转换图)
4. 数据字典(DD)

（三） 面向对象分析(OOA)

1. 面向对象基本概念
    1. 对象(object)
    2. 类(class)
    3. 消息(message)
    4. 封装(encapsulation)
    5. 继承(inheritance)
    6. 多态(polymorphism)
2. 统一建模语言UML
    1. 用例图
    2. 类图
        1. 类之间关系
            - 关联(→)
                - 合成：整体对象负责部分对象的声明周期(实心菱形)
                - 聚合：部分和整体之间的关系(空心菱形)
            - 泛化：继承(空心三角)
            - 依赖：(虚线→)
    3. 顺序图

（四） 需求建模

1. 基于场景的方法
2. 基于类的方法
3. 行为和模式

二 设计

（一） 设计概述

1. 数据/类的设计

2. 体系结构设计
3. 结构设计
4. 构件设计

（二） 设计原理

1. 模块化
2. 抽象
3. 逐步求精
4. 信息隐藏

（三） 设计模型

1. 体系结构设计
2. 构件级设计
3. 用户界面设计

三 面向对象方法和UML建模

Part 4 质量管理

（一） 质量概念

1. McCall质量因素

（二） 软件质量保证

1. 技术评审
2. 软件测试
3. 程序正确性证明

（三） 软件测试基础

1. 目标
2. 准则
3. 方法

（四） 软件测试策略

1. 测试步骤
2. 单元测试
3. 自动化测试
4. 集成测试
5. 回归测试
6. 确认测试
7. 系统测试

（五） 软件测试技术

1. 白盒测试
2. 黑盒测试

（六） 软件配置管理

1. 软件配置

2. 软件配置项
3. 基线
4. 配置管理过程
5. 版本控制
6. 变更控制

Part 5 管理软件项目

（一） 项目管理概念

1. 人员
2. 产品
3. 过程
4. 项目

（二） 过程度量与项目度量

（三） 软件项目估算

1. 功能点
2. 代码行