Soft^ware^ Engineering (S.E.)

<BLOCK1: Agile Software development>

1. 引入

Def 软件工程
> An engineering discipline that is concerned with all aspects of software production.

- Software E. (SE) is a layered technology (Tool → Method → Process → Quality focus)

2. Softw Process.

Def Software process
> A software process is a set of structed activities that leads to the production of the software.

系统地说，最符合本课程的：

课程架构

Requirement specification → Development → Test / Deployment → Evolution

Development:
- Analysis (Creat a conceptual model)
- Design (Implementable model)
- Implementation (实现)

Software process model:

① Waterfall model (Sequential approach, 慢, 文档多)

② Evolutionary development (快, 多 versions)

③ The Rational Unified Process (RUP) 已弃

④ Agile Software Development

3. 简单介绍 Agile Software Development (Rapid development and delivery)

Def Agile process
> The processes of specification, design, implementation and testing are concurrent, referred to as an iteration.

① 关于 Agile.

A set of best practices in SE // Focus on code. // iterative approach

visible process // Extreme Programming // Test Driven Development (TDD)

pair programming

注：有PPT一样，提供了「盲人摸象」式学习，哈哈！

4. Requirements.

Def Requirement
> A requirement is a feature that your system must have in order to satisfy the customer

Def Stakeholder
> Any person or organization who is affected by the system in some way and so who has interest

(Requirement ↔ Stakeholder: 相关)

注 Requirement 会变，难定义，是SE中最难的问题，所以↓

① Requirement Engineering (帮工程师理解真正需求).

需求分类
- Functional requirements (Should do)
- Non-Functional requirements (System's properties and constraints)

不同 Requirements 中有 conflicts 是很正常的事，看 stakeholders 的角色 trade-off.

output: Software Requirements Specification (SRS)

What is required of system developers. 应该做什么

② Requirements Capture (Fact-finding)

Background Reading; Interviewing; Observation; Document Analysis.

5. Agile 中的 Requirements.

在 Agile 中，Requirement 又是 user stories. 通俗易懂，任何人都可以写.

Def User story
> User stories are short, simple description of feature, They typically follow a simple template: As a...; I want..; So that..

注 1) 开发团队需要将 story break down into tasks.

2) 长的 story 被称为 Epics，在开发中会将 Epics 拆分为 story.

3) 每个 story 应配有 Acceptance Criteria，是用来检验是否完成需求的 test.

Def 系统待办
> Product backlog, which is a prioritsed list of the functionality to be developed in a product or service

Epic → story → Task.

① 在 backlog 中需要确定 Prioritisation of stories:

- 一种确定的方式：Dynamic system development method (DSDM).

将其分为：Must have, Should have, Could have, Want to have

⇒ MoSCoW

<补充> 好的故事：Valuable, Estimatable, Small, testable ...

② Estimating：估计关于 work 完成的时间、资源等等.

一般方法：T-shirt sizing (小中大); Ideal time (天/时)

Def Story points

> Story points: an arbitrary measure that allows the teams to understand the size of the effort.

注: Team members 有不同能力和经验, 使用精确 time 估计会有较大差异; Arbitrary measure can avoid over-estimating. (比如 T-shirt size, story point) 更好的让员工理解 size of the effort.

③ Prototyping (physical 或 logic)　　设计 technical 可行 ↑　low time, cost.

1) Low-fidelity (低保真): paper and sketch; Quick, Effective

2) Medium-fidelity: 功能有限, 有可点击的交互区域.

3) High-fidelity: 考虑 user interface (UI), user experience (UX) 细节和功能都与最终设计相似.

< BLOCK 2: Development & Test >.

1. Analysis: 为了 precise understanding of requirement, 关注系统内部.

① Conceptual modelling　↕ 领域建模 (struck) 也称为建模

Def 识别概念

> Aim to identify the individual concepts which exist within a problem, described using UML class diagrams

② Analysis Class (三种分类).

1) Entity class: model information that is long-lived and persistent

2) Boundary class (交互/对接): model the interaction (Button, page 等).

3) Control class: Deal with performing tasks, getting/setting data and coordinating behaviour.

③ Class Relationships (两种关系).

1) Association: 类之间的 bidirectional connection (数据可双向流通)
用 multiplicity indicators 表示连接数量: [A] 1 —— 0..* [B]
0..2 (0到2), 0..* (0或更多)　(1个A对应0或更多).

2) Inheritance
subclass will inherit all attributes, methods, relationships defined in any of its superclasses. 可减少冗余代码了.

2. Design & 重要名词.

① Role of Design.

Design transforms the analysis model into a design model that serves as a blueprint for software construction.

② Encapsulation: 限止直接访问, 设置 setting/getting 方法.

③ Modularity: Separate the functionality of a program into independent, interchangeable modules.

④ Coupling (耦合): metric of dependencies between subsystems. {Tight / Loose} ↑ 开发过程

⑤ Cohesion (内聚): metric of dependencies within a subsystem {High ✓ / Low}

⑥ Refactoring (重构): intended to improve non-functional attributes of the software.

⑦ OO 的好处 (Object-Oriented): Easier maintenance, class reusable..

3. Implementation.

goal: mapping Design to code, implement the system in terms of components.

① Build

Def Build

> A "build" is an executable version of the system, normally a part of the system

Def integration build plan

> An integration build plan describes the sequence of builds required in an iteration.

② 再谈 Association

JAVA 没有双向连接, 只有 unidirectional link.

1→1: 都有关, (ex: A→B, A中有B, B不可操作A).

1→n: 依然两者都有关, 但使用 Collections.

4. Testing

① 目的 { Validation testing: software meets its requirements / Defect testing: Discover defects } Build Confidence

② Testing process

unit testing ⟷ system testing ⟷ Acceptance testing

- unit testing: test individual component such as a function or a method.
- system testing: 在类似真实环境中 Test 整个应用.
- Acceptance testing: Stakeholders to confirm the system meet the business needs

<补充Q> 为什么在TDD unit test 中，Refactoring 很重要？

Answer: TDD 开发 code in small independent increments，它将导致 code duplication and poorly structed code.

Regular refactoring can improve the code structure s.t. it's more readable and easy to change

③ Testing：Techniques

1) Black-Box testing (test functional requirements)
   - Partition testing (选择不同参数对function 测试 input)
   - Regression Testing (每个Build的测试全部跑一遍，确保更新后功能正常)

2) White-Box testing (test internal program logic)
   - Basis Path Testing
     - Cyclomatic Complexity：# 决策(◇) + 1
     - → Number of test (精简测试用例覆盖路径)
     - 使得每条语句执行至少一次
   - Mutation Testing
     - make small change to source code: if(a==0) → if(a!=0)
     - if test fails, the test is robust.

5、简介 Test Driven Development (TDD)

TDD：write tests prior to write the production code.

JUnit：支持TDD的 unit test 框架 (自动的).

使用 @Test 表明 test 方法，无需 main 自动执行 (公共，无参，无返回)

使用 assertEquals ("期望", 实际) 来辅助测试.

<BLOCK 3：Architecture & Project Management & Design Principles>

1. Software Architecture

Def 软件架构 ① A Software Architecture is a description of how a Software System is organized

一般用UML描述

注
1) Non-functional requirements 是否架构的成功，Requirement 关联 Architecture
2) 作用：System anaysis，reuse，Communication
3) Agility中的 architecture，越早着手 Architecture 越好，因为不好改.

② Architectural patterns

Def 架构 patterns: Architectural patterns are a means of representing, sharing and reuse ~~patte~~ knowledge.

有利于系统快速开发.

1) Web-~~base~~based architectures
2) Distributed System architecture

   Web-based 的 单点连接 当用户多 表现不好，可用一些 techniques 优化
   - Load balancing：The Distribution of tasks over a set of Resources with aim of making their overall processing more efficient.
   - Cloud computing：delivery of on-demand computing services typically over the internal.
   - Caching
3) RESTful architecture

   Representational State transfer (REST) 有以下6条 Constraints

   Client-Server, Cacheability, Uniform interface

   Statelessness, Layerd system, Code-On-Demand
4) Mobile application architecture

   通常指 layerd architecture，即

   presentation → Business → Data.

2. Project Management

AIM Project management: Ensure the Software is deliverd On time, ~~With~~ Within budget, and With Quality. 时间，预算，质量.

① Software Projects' Distinctions (特性)

1) intangible (Can not see)
2) flexible
3) Not standardalised
4) Many software projects are "One-off" Project.
   - Technologies are change to fast
   - Old experiences may be obsolete.

② Project management activities

1) Project planning

—整体考虑 Constraints, parameters, 建立 milestones, schedule

并不断 iteration 的过程, 可从 Quantity, Validation 等多个角度 plan

Def 里程碑

> Milestones are the recognisable end-points of a process activity

必须将整个 project broken down into basic activities 才可建立 milestones.

Def Scheduling

> Project schedule: Estimate time and resource required to complete activities and organise them into a coherent sequence

其一般有以下要点: Split project into separates tasks, concurrently

Minimize task dependencies.

<如何表示 Schedules>

(a) Task chart

(b) Activity network ⭑ } AOA / AON ⇒ 详见整理 PPT

(c) Bar chart (schedule against calender time)

---

③ Agile project management

传统项目管理是 plan-driven. Agile project 是不同形式:

Adapted to incremental development and the particular strengths of agile methods. (Scrum approach)

EX 一个显著的特点: Short daily stand up meeting.

purpose: 让所有人知道目前 project 正在干什么, 以及出现问题可以及时调整.

活动: share information

share the progress since last meeting

problem and plan

Agile 中, 重点在于 visibility, Communication, trust. *

<补充> AI 项目管理: Data analysis, Risk Prediction, Optimize resource allocation 特色

3. Design Principles (SOLID).

用于 code 的常见思考问题, 才可以写出好的 code (UML 是一种方法).

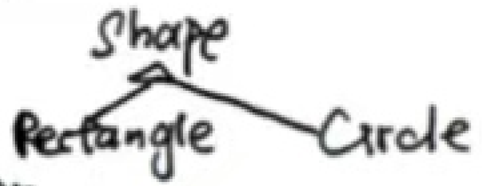
① Single Responsibility Principle (SRP)

Rule: 一个类应专注于单一功能, 非常直观.

② Open-Closed Principle (OCP)

Rule: "Open for extension, closed for modification".

关键在于使用 Abstraction 类. 例如



只用在子类中修改已有的 draw() 方法,

不必单独有一个类对各种类型处理.

③ Liskov Substitution Principle (LSP)

Rule: 子类可随意替换父类而不引起错误.

要点: 1) 子类实现父类方法, 且不修改已有实现方法.

2) 子类不扩展父类行为, 不可增加新方法, 新属性.

3) 返回值, 参数与父类一致.

即一种比 "IS-A" 描述更强的继承规则.

④ Interface Segregation Principle (ISP)

接口应该尽可能小, 以免子类实现 "空" 方法.

即接口小到与具体类需求匹配.

⑤ Dependency Inversion Principle (DIP)

Rule: 高层不依赖低层, 都依赖于抽象;

抽象不依赖细节, 细节依赖于抽象.

高层和低层之间通过抽象(接口)连接, 低层修改不影响高层.

高层修改也不影响低层. 凡高层直接依赖于细层, 则违反 DIP

⑥ Don't Repeat Yourself (DRY) ⊆ OCP.

将重复代码提取成一个函数, 减少重复.

其实是 OCP 的一种表达.

总结: SRP, OCP, LSP, ISP, DIP (DRY ⊆ OCP).

⏟ Solid 设计原则.

<BLOCK 4:

1. Software Design Patterns

Def 设计模式. A design pattern is a general repeatable solution to a commonly occurring problem in software design.

设计模式可以分为3个类型.

① Creational patterns — 关注应该 creat 什么样的类和对象.
- Factory method: 创造类的"构造器", 在不用具体细化对象时即可创建对象.

② Behavioural design patterns — 关注对象之间的通信
- Observer: 当一个对象状态变化时, 所有"Observer"收到通知并自动更新.
- Strategy: 将一组算法封装在独立类中, 可以动态的选择具体使用的算法.

③ Structural patterns — 关注 class and object composition
- (介绍三种 Wrapper 的样例 (外部对象包含内部对象))
- Decorator: 不修改原代码情况下, 动态地为对象添加更多功能
- Adapter: 将未 implement 某接口的类放在 implement 该接口的类(adapter)中, 在兼容下复用类.
- Composite: 用树状结构表示"部分-整体"结构. 并可用处理单个实例的方式处理对象组合. (ex. 文件和文件夹).

另外, 还提了 Architecture design pattern:

model-View-Controller (MVC).

Immutable View (可解决两个变量指向同一对象的问题).

问题: 为什么需要 Design Patterns.

1) Represent Experience.
2) Help Structure Code
3) Reduce Dependency
4) Communication (Vocabulary)
5) In line with design Principle.

2. Open Source Software (OSS) (了解即可)

① 定义及引入.

Def OSS: "Open Source Software" is usually used to mean software which is free of charge, and free of legal restriction on how it can be used.

Def OS: Open Source is a methodology that promotes free redistribution and access to a product's design or ideas and implementation details.

注 开源软件未必免费, 相反的, 开源可以认为是一种"营销手段".
可以用 Cathedral (教堂) 和 Bazaar (集市) 来比作商业 software 和 OSS.

② 其它.

1) Software Freedom (对 OSS 最重要的自由): freedom to run, change, redistribute, distribute modified version
2) Copyright, Copyleft (自由版权, 鼓励 copy).
3) e-voting 不可信: we can not sure the software that is displayed is the software that is actually used.
4) fork: OSS 的一个开发分支, 从主程序中 copy 一个作为单独的自用程序开发.
5) OSS 中, user 和 ~~producer~~ developer 的界限不是明显的.
6) 现代 OSS 大部由公司掌控, 并非个体小组织.

3. Software Development tool. (了解即可).

主要是 Microsoft's Best Practices.

① Revision Control System.
② Daily Build

Def Build: A Build is the process of producing a complete working version of the software for the whole system.

Def 持续集成: The practice of builds involving full testing for every modification is called continuous integration.

③ Bug Database.

按预定义，记录bug的合作.

④ Static and dynamic analysis tools.

with run or without run the code.

⑤ Globalisation and Localisation.

语言的翻译 (1→多, 多→1).

⑥ Other

Documentation Generator (Javadoc), Interactive Development Environments (IDEs)

小总结 design principles, design patterns, tools 都需要一定的上手难度,

但都是必要的，开发一个大型 software 若没有它们，会使后续的维护复杂.

4. Ethics, Risk and Quality Management.

① Algorithms Fair.

1) 主要考虑两种公平 (Parity):

a. Statistical Parity: 不同群体内被判定为有的比例相同.

忽略了个体差异 (Disparate impact), 可能造成discrimination.

b. Predictive Parity: 根据具体数据做决策, 不造成 discrimination.

但是数据本身可能存在 bias.

2) 四种方式 (为了公平):

a. 去 input 部分特征 (可能有相关性)

b. 去 input, → inaccurate

c. add more feature

d. Pre-processing / post-processing 去弥补 bias.

3) 法律层面上

对人产生影响的算法应透明可追溯.

保护他人个人数据

② Risk management.

主要是对不确定性的预防.

1) 风险及分类.

Def Risk: A risk may be defined as the probability of something undesirable happening during software development.

S.E. 中讨论3种风险:

- Project risk (时间, 资源).
- Technical risk (表现, 质量).
- Business risk (软件是或产品不能在市场中给公司带来价值).

} 可能3种组合着发生. ex. knowledge 不足带来延期.

2) 风险管理流程.

所以, 需要对未发生的 risk 进行预测和预防.

Risk identification → analysis → planning → Monitoring (监视该Risk发生…

planning ⇕ Contingencies: alternative plans.

3) Agile Risk management.

不断迭代, test, 错误一般发现及时;

No-long-term planning, 可能较难 indentify risks

③ Quality management.

Aim 重点在于 meet customer's specification, fitness for purpose; "good enough" software

↓ software 不一定完美…

一些常见的对 Quality 的 metric:

Safety, Reliability, reusability, efficiency, understandability ……

Reliability ↓ 特殊环境下仍正常运行

注 从用户视角看, Quality 有 visible 和 invisible 之分, 最重要的是 visible 的 Quality

④ 其它杂乱的, 了解即可.

Software Standards, 一个 rule document, 可能对 Quality 有帮助, 但需要 review regularly, 否则 out of date.

人多开发未必快

Outsourcing (外包) 存在沟通问题.