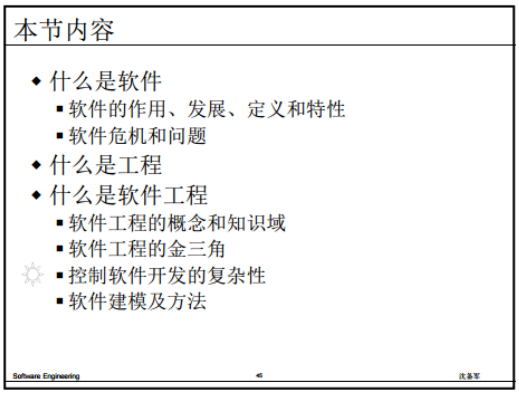
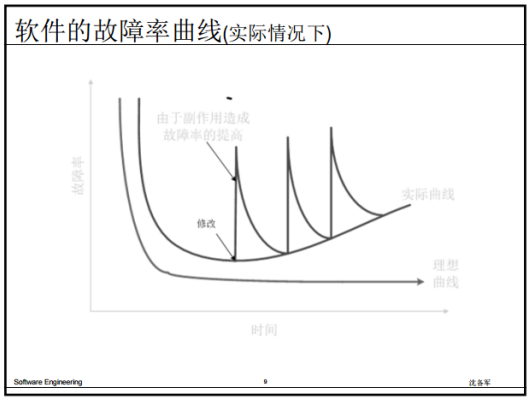
01- 软件工程引论

- 

- 软件的定义：计算机系统与硬件相互依存的另一部分，包括程序、相关数据及其说明文档。
- 软件的特征：软件开发不同于硬件设计；软件生产不同于硬件制造；软件维护不同于硬件维修
- 硬件的故障率曲线：浴缸曲线
- 软件的故障率：
- 

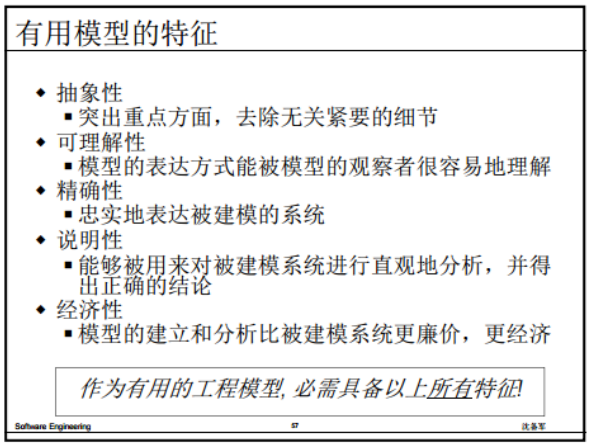
- 软件面临的新挑战：软件复杂性增加；软件规模不断扩大；软件环境的变化；遗留系统的集成和复用；软件开发的高质量和敏捷性要求；分散的开发团队的协同

- 软件危机的主要表现： 教材 p6
- 软件危机的原因：一方面与软件本身的特点有关；另一方面是有软件开发和维护的方法、过程、管理、规范和工具不正确有关

- 软件开发是一门工程
- 软件工程是为了经济地获得可靠的且能在实际机器上高效运行的软件而建立和使用的工程原理；是将系统的、规范的、可量化的方法应用于软件的开发、运行和维护，即将工程化应用于软件；是应用计算机科学和数学的原理来经济有效地解决软件问题的一种工程
- 软件工程的知识域：教材 p10

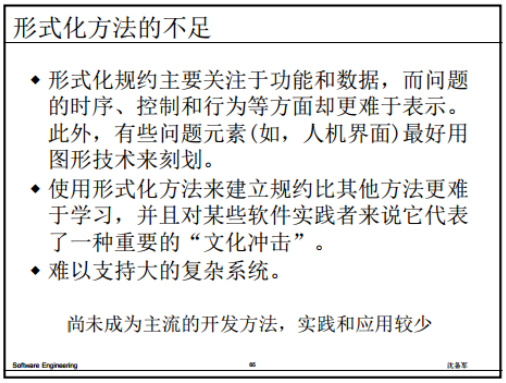
- 软件工程的金三角：人，技术，管理
- 软件工程技术：系统工程，需求，设计，编码，测试，运行和维护
- 控制软件开发的复杂性主要来源于技术的复杂性（不断发展，使用多项技术），需求的复杂性（模糊，不断变化），人的复杂性
- 软件建模是软件工程的核心技术

- 有用模型的特征

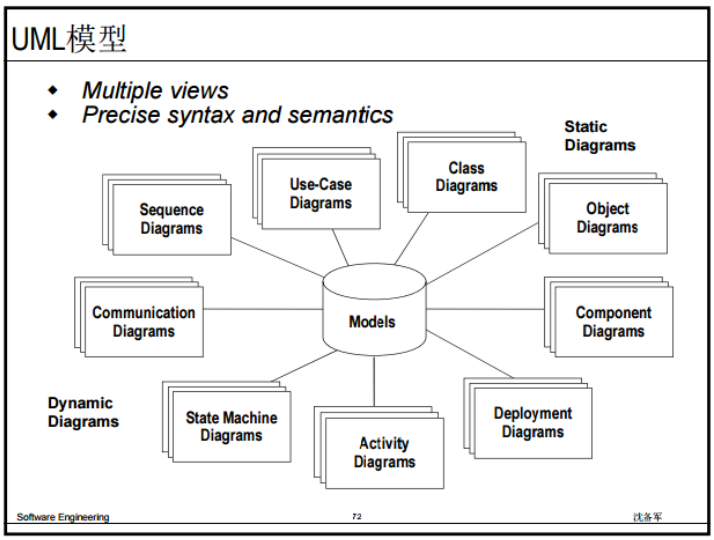


-
- 软件的三种模型：教材 p48
  - Computation-independent model(CIM),
  - platform-independent model(PIM),
  - Platform-specific model(PSM)
- 软件建模的方法：第三章
  - 形式化方法
  - 结构化方法
  - 面向对象方法
  - 基于构件的软件开发方法
  - 面向服务方法
  - 模型驱动的开发方法
  - 敏捷建模方法

- 形式化方法：是基于数学的技术开发软件，如集合论，模糊逻辑，函数
- 形式化方法的好处：无二义性，一致性，正确性，完整性
- 形式化方法的不足：



- UML 模型



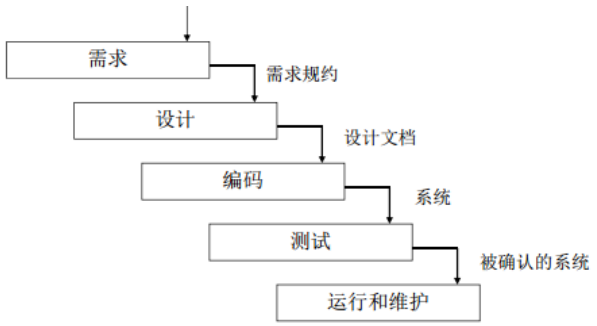
02- 软件过程

-

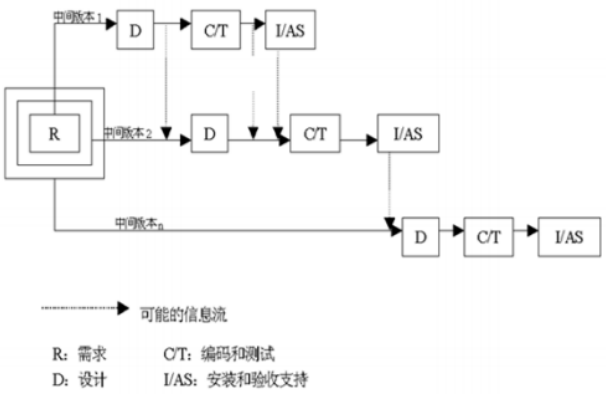
- 软件过程，也成为软件生存周期过程，是指软件生存周期中的一系列相关过程
- 软件生存周期模型，又称软件开发模型，是软件生命周期的一个框架，规定了软件开发、运作和维护所需的过程、活动和任务
  - 线性顺序模型
  - 增量式模型
  - 演化模型

- 瀑布型 waterfall

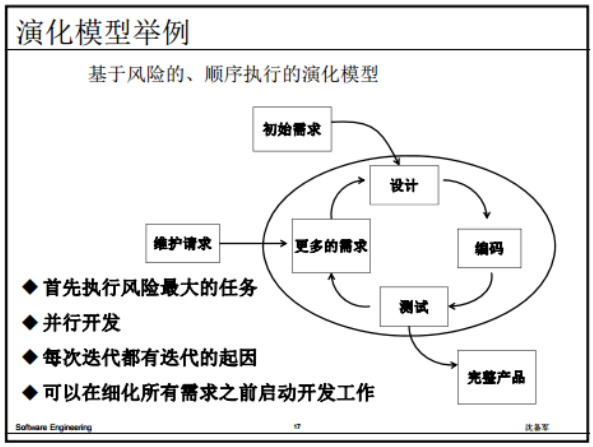
-



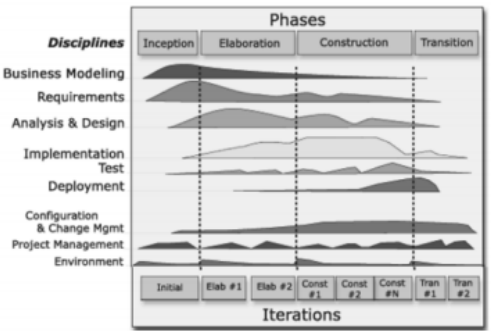
- 瀑布型的特点，使用时机，价值，风险和缺点
- 增量型 incremental：构造一系列可执行的中间版本



- 特点：重点在中间版本，一个一个迭代的做，每一个中间版本都可以用
- 演化型 evolutionary：迭代开发的方法，渐进的开发各个可执行版本，逐步完善软件产品，是目前采用最广泛的模型
- 演化模型和增量模型的区别：需求在开发早期不能被完全了解和确定，在一部分被定义后开发就开始了，然后在每个相继的版本中逐步完善



- 现代的软件过程都采用演化模型
  - 统一软件过程 RUP
  - 敏捷过程 SCRUM，XP 等
  - 净室软件过程 Cleanroom
- 演化模型的子类
  - 原型模型 prototyping
  - 螺旋模型 spiral
- 统一软件过程 RUP



- **RUP是一个风险驱动的、基于UML和构件式架构的演化开发过程 。**

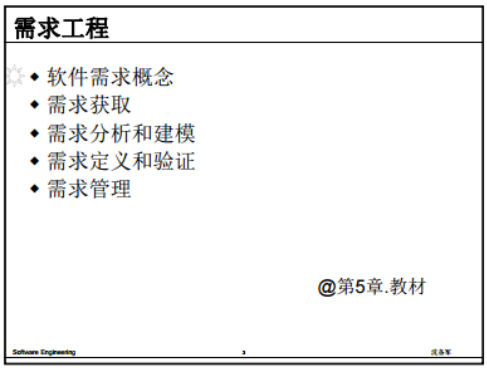
- RUP 的四个阶段：先启，精化，构建，产品化
- 迭代计划策略：增量式，演进式，增量提交，单词迭代（退化为瀑布模型）

- 敏捷过程

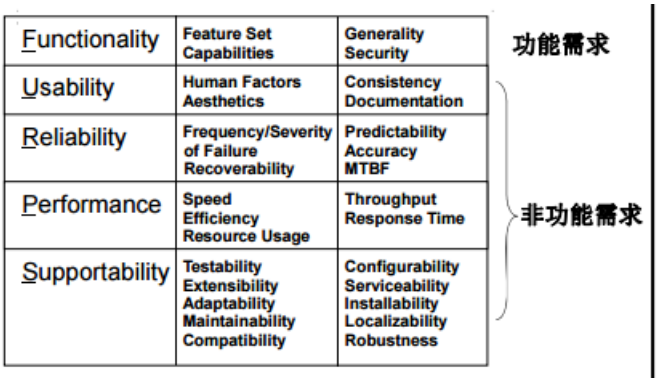
XP SCRUM Crystal ASD dx
MSF FDD DSDM Lean Development

-
- Scrum 核心准则：迭代开发，自我管理
- Scrum 的 4 个主要的工件：产品待办事项列表，sprint 待办事项列项，发布燃尽图，sprint 燃尽图

03- 需求工程

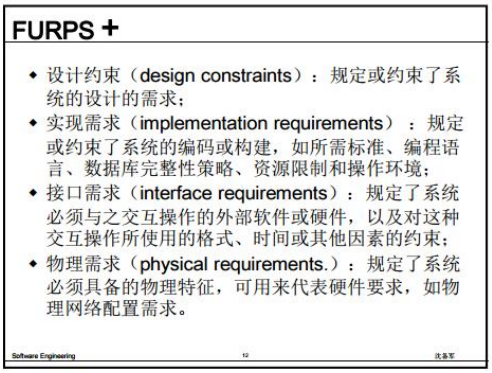


-
-
- 需求：系统必须符合的条件或能力
- 软件需求：用户对目标软件系统在功能、行为、性能、设计约束等方面的期望，内容包括 FURPS+



-
- 功能性、易用性、可靠性、性能、可支持性
- 功能性需求：特性、功能、安全性（概要功能需求及详细功能需求）
- 易用性需求：许多实现“用户友好”的因素
  - 人员因素，美观，用户界面的一致性，联机帮助和环境相关帮助，向导，用户文档，培训材料

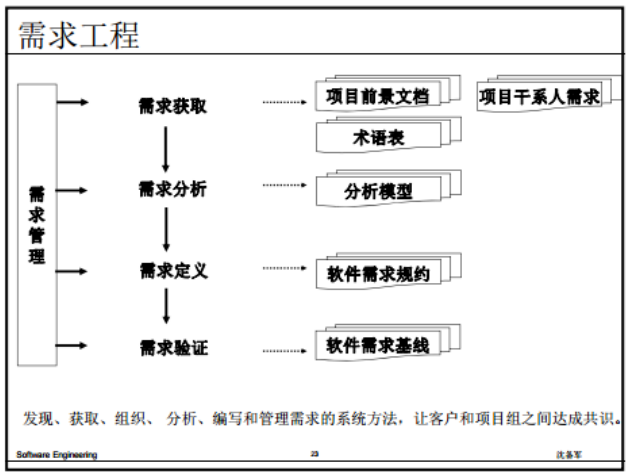
- 可用性需求
  - Training time requirements, measurable task times
  - User abilities (beginner/advanced)
  - Comparison to other systems that users know and like
  - Online help systems, tool tips, documentation needs
  - Conformity with standards
    - Examples: Windows, style guides, GUI Standards

- 可靠性需求：软件无故障执行一段时间的概率
  - 故障的频率，可恢复性，可预见性，准确性，MTBF（平均失效间隔时间）
    - 可靠性需求
      - Availability (xx.xx%)
      - Accuracy
      - Mean time between failures (xx hrs)
      - Max. bugs per/KLOC (0-x)
      - Bugs by class - critical, significant, minor
- 性能：功能需求基础上规定的性能参数
  - 速度，效率，可用性，准确性，吞吐量，响应时间，资源使用情况
    - 性能需求
      - Capacity
      - Throughput
      - Response time
      - Memory
      - Degradation modes
      - Efficient use of scarce resources
        - Processor, memory, disk, network bandwidth
- 可支持性：进行系统测试，安装，扩展，移植，本地化等工作时所需工作量的大小
  - 可测试性，可扩展性，可适应性，可维护性，兼容性，可配置性，可服务性，可安装性，可本地化
    - 支持性需求
      - Languages, DBMS, tools, etc.
      - Programming standards
      - Error handling and reporting standards
- FURPS+：设计约束，实现需求，接口需求，物理需求
  - 

- 软件需求的三个层次：
  - 系统需求（业务需求，产品需求）
  - 项目干系人需求（原始人需求），项目前景文档（概要需求），软件需求规约（详细需求）

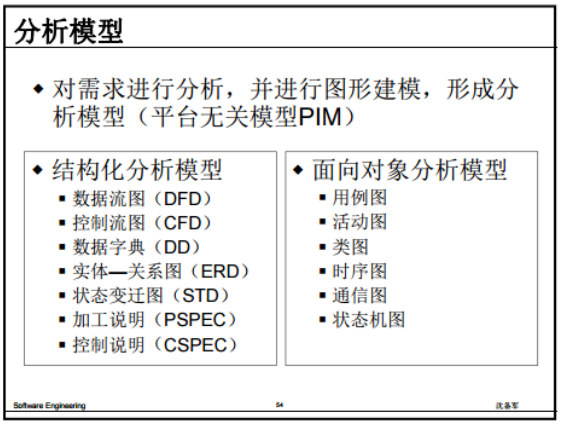
- 优秀需求具备的特性
  - 单个优秀需求应具有的特性：完整性，正确性，可行性，必要性，划分优先级，无二义性，可验证性
  - 多个优秀需求应具有的特性：完整性，可理解性，一致性，可跟踪性

- 需求工程



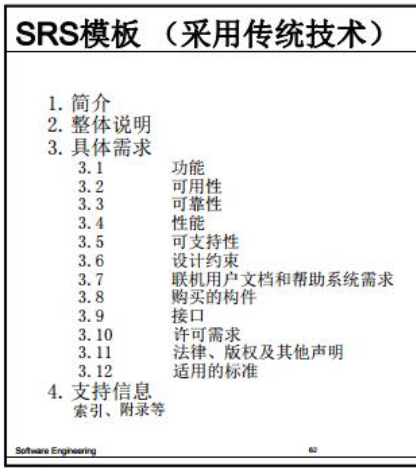
-
- 需求获取
  - 分析问题及根源，识别项目干系人，识别项目的约束，获取常用术语
  - 识别需求的来源，收集需求，产品定位，撰写产品特性，定义质量范围
  - 定义文档需求，建立项目范围，划分特性优先级
  - @Vision 文档

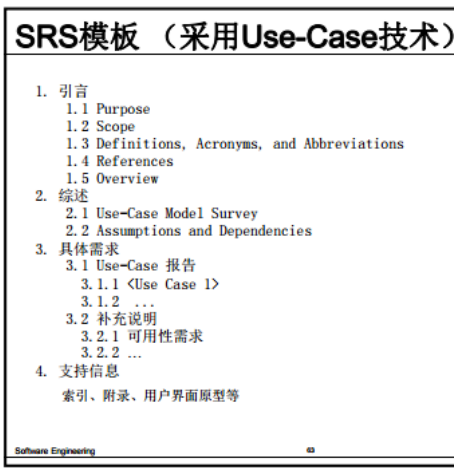
- 需求分析和建模



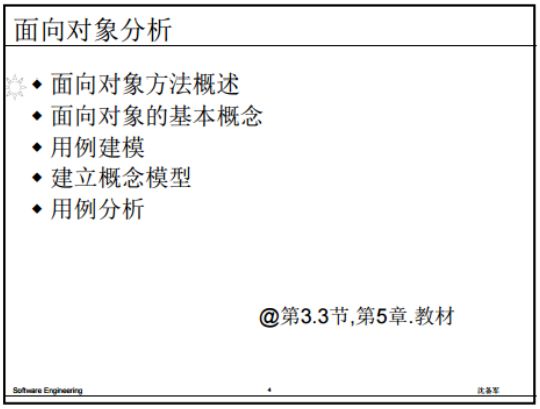
-

- 需求定义和验证



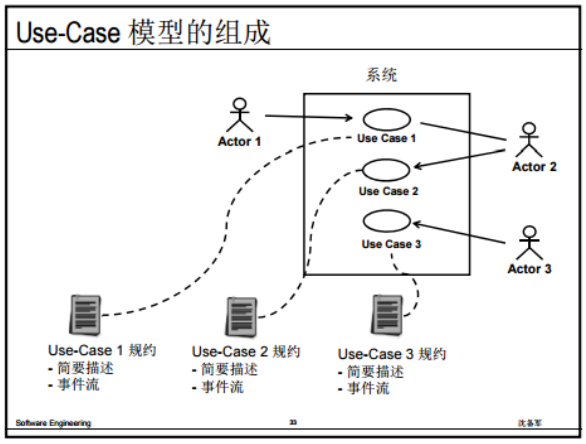


04- 面向对象分析



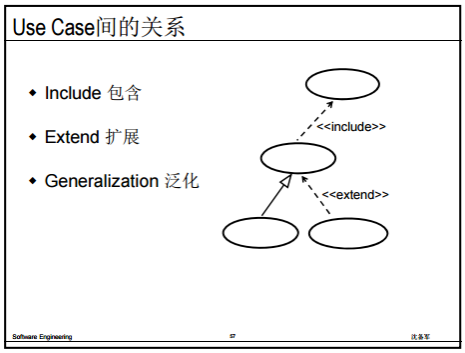
- 面向对象的基本原则
  - 抽象 abstraction，封装 encapsulation，模块化 modularity，层次 hierarchy
- 类和对象的关系
  - 类是对象的抽象定义，类并不是对象的集合
- 包 package
  - Organize elements into groups
  - Contain other model elements
- 面向对象分析的步骤
  - 用例建模
  - 建立概念模型
  - 用例分析

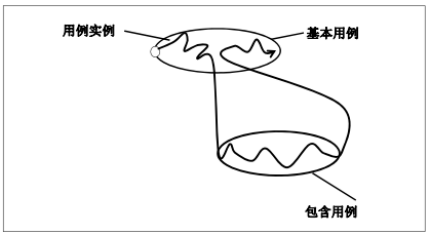
- 用例建模
  - 识别 actor 和 use case，画 use case 图
  - 编写 use case specific
  - 优化 use case 图的结构

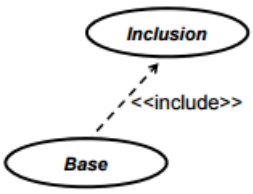
- 

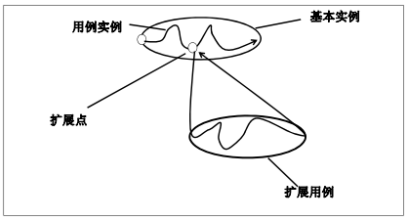
- 通信-关联：actor 和 use case 间的通信渠道，箭头表示谁启动通信

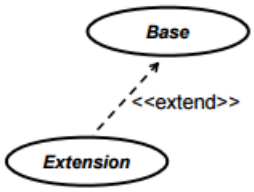
- Use case specification 用例归约
  - 用例名，描述，基本流，关系（actors 和 use case 的），活动图，用例图，特殊说明，前置条件，后置条件，其他图（交互、活动等，用户界面框图），非功能需求，业务规则，扩展点
- 事件流（分为基本流和备选流）
- 活动图：活动图可以划分泳道
- 前置条件：use case 启动的约束条件，并不是触发 use case 的事件
- 后置条件：当 use case 结束时，后置条件一定为真；降低用例事件流的复杂性并提高其可读性；陈述在用例结束时系统执行的动作

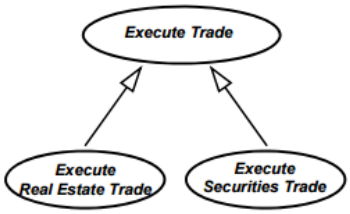
- 优化 use case 图的关系
- Use case 间的关系：包含 include，扩展 extend，泛化 generalization
- 

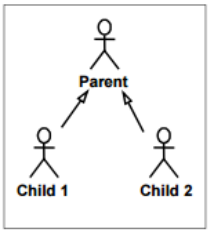
- Include 关系
  - 基本用例包含某个包含用例
  - 包含用例定义的行为将显示插入基本用例
  - 到达插入点时执行包含用例
  - 



- Extend 关系
  - 从基本用例到扩展用例的连接
  - 当扩展用例的行为插入基本用例；只有当扩展条件为真时才进行插入；在一个命名的扩展点插入基本用例
  - 当到达扩展点并且扩展条件为真时执行 get news ---> get quote
  - 



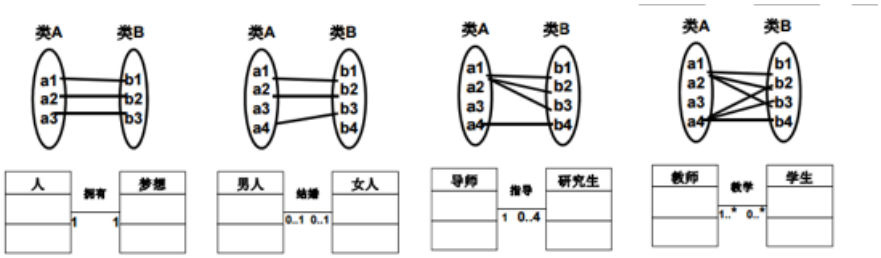
- Use case Generalization 关系
  - 从子用例到父用例的关系，在父用例中描述通用的共享的行为，在子用例中描述特殊的行为，共享共同的目标
  - 

- Actor generalization 关系
  - Actors 有公共的属性，多个 actors 和 use case 交互时可能有公共的角色或目的
  - 

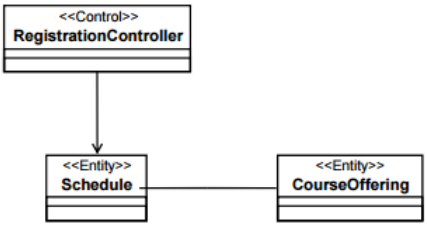
- 建立概念模型
  - 识别 conceptual class 概念类
  - 建立 conceptual class 之间的关系
  - 增加 conceptual class 的属性
- 概念类图的作用：to model vocabulary of system；collaboration；a logical database schema
- 识别概念类
  - Wirfs-brock 名词提取过滤法

- 类之间的关系
- 关联 association
  - 多重度 multiplicity

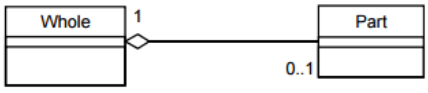
Unspecified	
Exactly One	1
Zero or More	0..*
Zero or More	*
One or More	1..*
Zero or One (optional value)	0..1
Specified Range	2..4
Multiple, Disjoint Ranges	2, 4..6

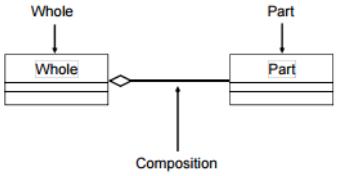


- Roles：instructor，prerequisites，department head



- Navigability：
- 两个类之间关联同时可以有多条

- Aggregation 聚合
  - Aggregation is “a-part-of” relationship 是一部分
  - 多重度还是照常表示
  - 

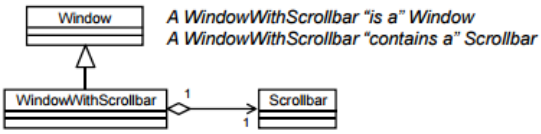
- Composition 组合
  - 组合关系中，部分依赖于整体存在，整体消亡部分也消亡
  - 

聚合　　组合

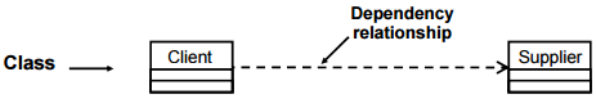
例：公司和雇员　　例：订单和订单项

-

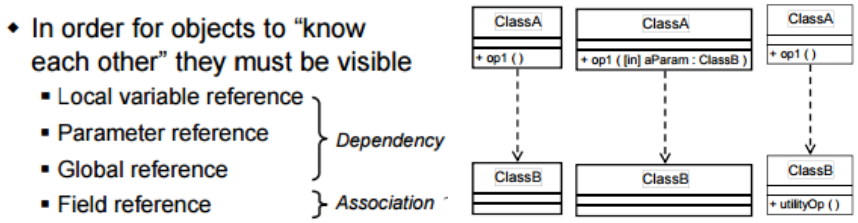
- Generalization 继承/泛化
  - 定义抽象的层次
  - 一个类可以继承自多个类
  - Liskov 替换原则 LSP，子类型应该能替换基类型
  - 继承和聚合
    - 继承表示 a kind of，聚合表示 a part of



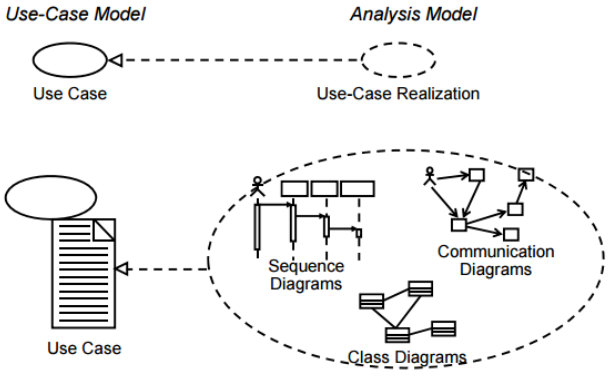
- Dependency 依赖
  - 依赖不是结构性的，是使用性的，一个类的改变可能会影响到另一个类
  - 依赖关系种类：访问 access，绑定 bind，调用 call，创建 create，派生 derive，实例化 instantiate，许可 permit，实现 realize，精化 refine，发送 send，替代 substitute，跟踪依赖 trace，使用 use



  -
  - 依赖和关联
    - 关联是结构性的，依赖是非结构性的

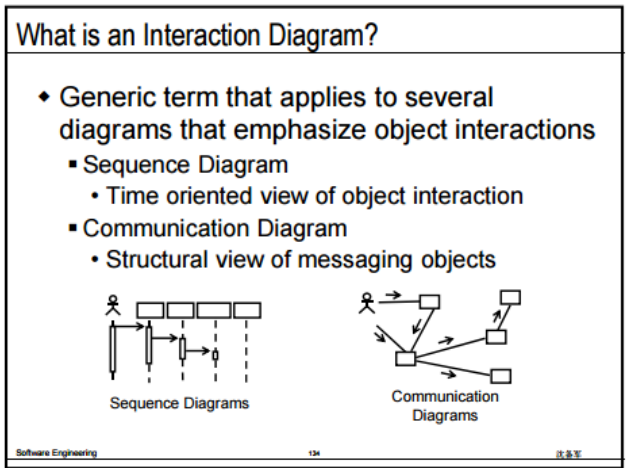


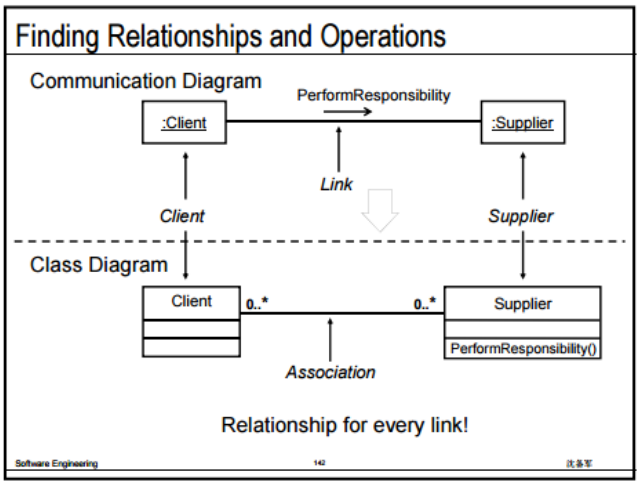
- 用例分析
  - 识别出用例实现
  - 针对每个用例实现：识别出分析类；建立时序图，生成通信图；对照通信图建立类图，完善每个分析类的属性和操作
- 用例规约



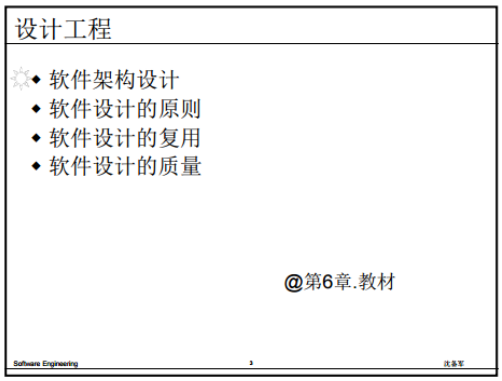
-

- 识别出分析类 analysis class
  - 实体类，控制类，边界类
  - 不同的衍型 stereotype 可采用不同的图标，用《》区别
  - MVC 设计模式，model view control – 实体类，边界类，控制类
- 实体类 entity class
  - 存储管理系统中的信息，是 environment independent 的
- 边界类 boundary class
  - 每个 actor/use case 对都要有一个 boundary class，是 environment dependent 的
  - 用户接口类，系统接口类，设备接口类 device
- 控制类 control class
  - 每一个 use case 定义一个控制类

- 交互图 interaction diagram
  - 时序图 sequence diagram
  - 通信图 communication diagram
  - 

  - VOPC 类图，根据通信类生成
  - 

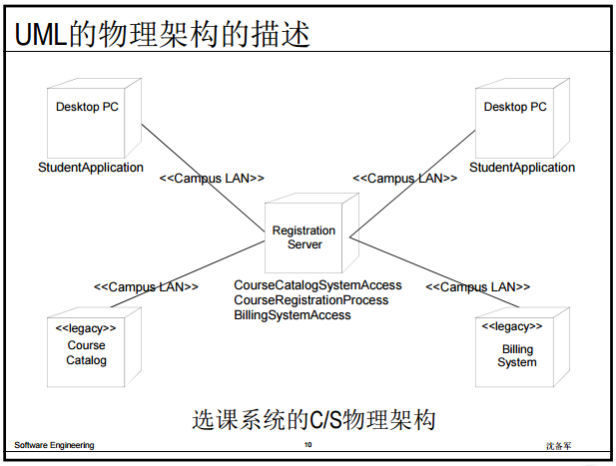
05- 设计工程

- 

- 设计步骤：
  - 架构设计，又称概要设计，它定义了软件的全貌，记录了最重要的设计决策，并成为随后的设计与实现工作的战略指导原则
  - 详细设计：又称构件级设计，它在软件架构的基础上定义各模块的内部细节，例如内部的数据结构、算法和控制流等，其所做的设计决策常常只影响单个模块的实现

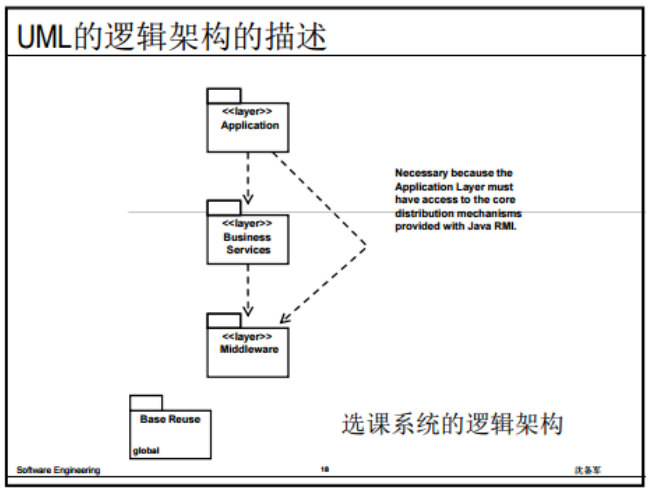
- 架构设计的内容
  - 定义软件的物理架构
  - 定义软件的逻辑架构
  - 定义软件的数据架构
  - 选择软件质量属性的设计策略

- 定义软件的物理架构，即架构的部署视图
  - 针对分布式系统，需要定义物理架构，刻画计算机或处理节点以及网络间的关系
  - 重点考虑可靠性的性能等非功能需求的支持

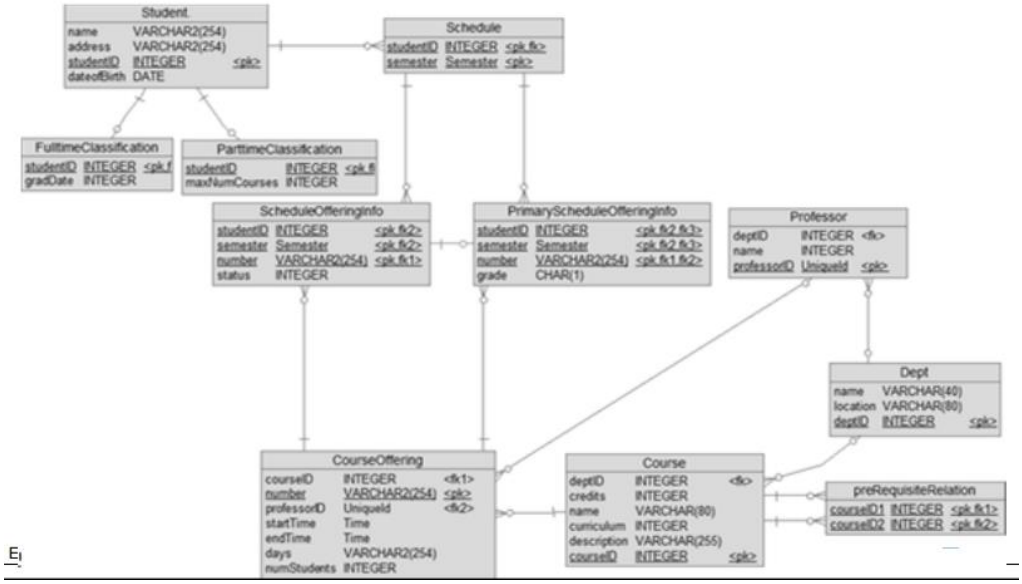
- 

- 定义软件的逻辑架构，即架构的逻辑视图
  - 把软件划分成为多个模块（又称构件），模块和模块相互协作，共同完成软件的需求规约。这些模块将部署在物理架构的统一节点或不同节点上

- 重点考虑功能需求的支持
- 所有软件都需要定义逻辑架构

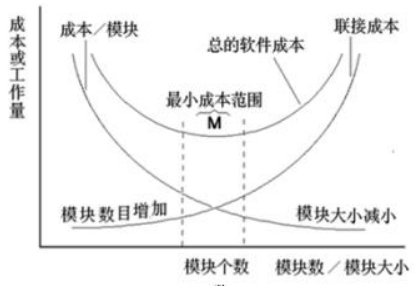


- 定义软件的数据架构，即架构的数据视图
  - 针对以数据为中心的软件

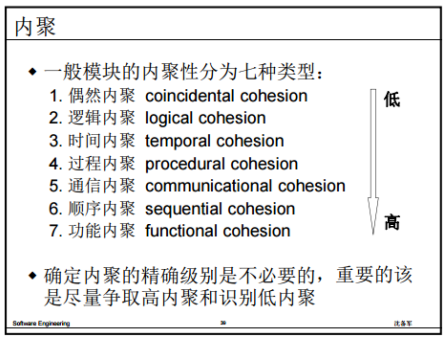


- 其他视图：
  - 针对多进程多线程软件，需要定义进程视图
  - 针对高安全性软件，需要定义安全视图
  - 针对高易用性软件，需要定义界面视图
  - 针对网站，需要定义页面导航视图
- 一个好的架构的特点
  - Resilient 能复原的，有弹性的
  - Simple
  - Approachable 可接近的，可亲近的
  - Clear separation of concerns
  - Balanced distribution of responsibilities 责任平衡分配
  - Balanced economic and technology constraints 平衡经济和技术约束

- 软件设计的原则
  - 抽象：过程抽象，数据抽象，对象抽象
  - 分解
  - 模块化
- 软件模块化的好处
  - 把复杂软件变简单，更容易实现
  - 把易变的部分和稳定的部分分开，模块可插拔可替换，应对需求变更和技术升级
  - 支持多人协同开发
  - 支持迭代式开发
  - 模块复用和模块组装，加快开发时间，节省开发费用，提高产品质量
  - 促进形成大规模开发的软件产业——软件工厂
- 模块化的成本



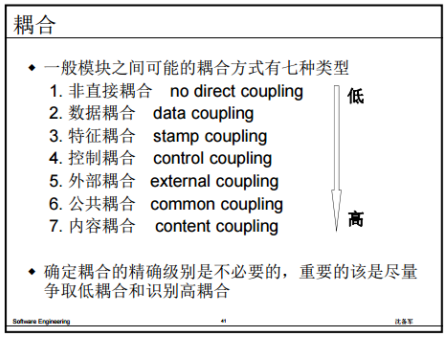
- 信息隐藏：每个模块的实现细节对于其他模块来说应该是隐蔽的
- 模块独立：模块完成独立的功能并且与其他模块的接口
  - 指标一：内聚，cohesion，是一个模块内部各个元素彼此结合的紧密程度的度量
  - 指标二：耦合，coupling，是模块之间的相对独立性的度量
  - 模块独立追求高内聚和低耦合
- 内聚和耦合
-


内聚

- 一般模块的内聚性分为七种类型：
  1. 偶然内聚 coincidental cohesion
  2. 逻辑内聚 logical cohesion
  3. 时间内聚 temporal cohesion
  4. 过程内聚 procedural cohesion
  5. 通信内聚 communicational cohesion
  6. 顺序内聚 sequential cohesion
  7. 功能内聚 functional cohesion

低 ↓ 高

- 确定内聚的精确级别是不必要的，重要的该是尽量争取高内聚和识别低内聚


耦合

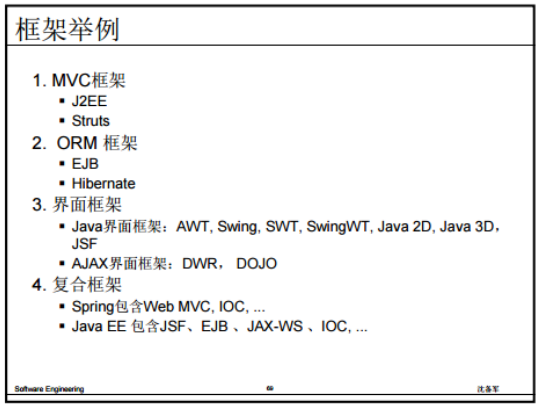
- 一般模块之间可能的耦合方式有七种类型
  1. 非直接耦合 no direct coupling
  2. 数据耦合 data coupling
  3. 特征耦合 stamp coupling
  4. 控制耦合 control coupling
  5. 外部耦合 external coupling
  6. 公共耦合 common coupling
  7. 内容耦合 content coupling

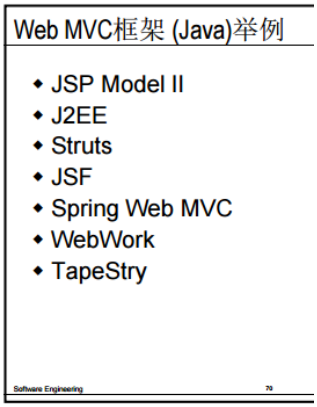
低 ↓ 高

- 确定耦合的精确级别是不必要的，重要的该是尽量争取低耦合和识别高耦合

- 软件设计的复用
- 常见的架构风格
  - 通用结构：分层，管道与过滤器，黑板
  - 分布式系统：客户端/服务器，3 层架构，代理
  - 交互式系统：模型-视图-控制器（MVC），表示-抽象-控制
  - 自适应系统：微内核，反射
  - 其他：批处理，解释器，进程控制，基于规则

- 框架

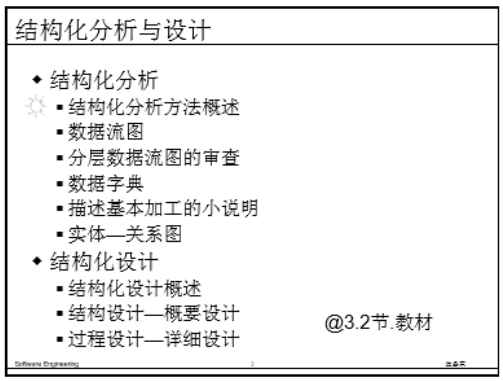




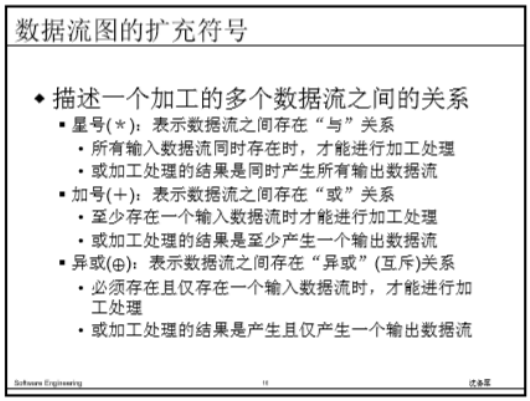
- 软件设计的质量
  - 设计应当模块化，高内聚，低耦合
  - 设计应当包含数据、体系结构、接口和构件的清楚的表示
  - 设计应根据软件需求采用可重复使用的方法进行
  - 应使用能够有效传达其意义的表示法来表达设计模型

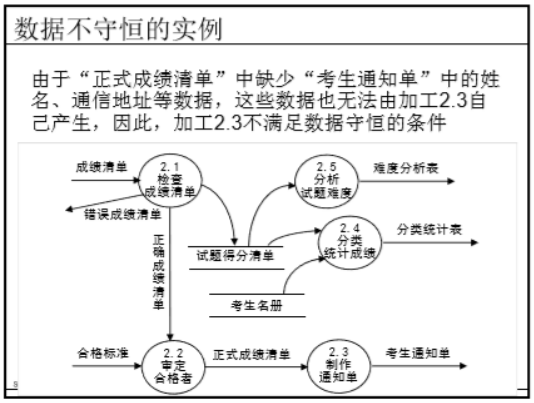
- 7 中软件设计的坏味道
  - 僵化性 rigidity，脆弱性 fragility，牢固性 immobility，粘滞性 viscosity
  - 不必要的复杂性 needless complexity，不必要的重复 needless repetition 晦涩性 opacity

07- 结构化分析与设计



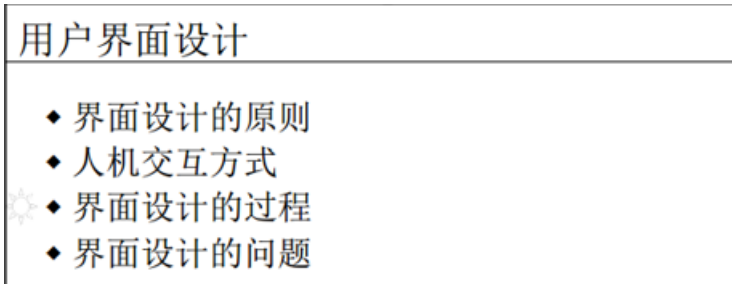
- 结构化方法：结构化分析 SA，结构化设计 SD，结构化编程 SP
- 结构化分析模型
  - 数据字典，数据流图，实体-关系图，状态转换图

- 

- 分层数据流图的审查：检查图中是否存在错误或不合理的部分
  - 一致性：分层 DFD 中不存在矛盾和冲突：父图与子图平衡，数据守恒


  - 完整性：分层 DFD 本身的完整性，即是否有遗漏额数据流，加工等元素

08- 用户界面设计

- 

- 教材 6.6 章

- 界面设计的原则
  - 易学性，用户熟悉性，一致性，教材 p200
  - 减少意外，易恢复性，提供用户指南，用户多样性，教材 p200
- 界面设计的黄金规则
  - 让用户驾驭软件，不是软件驾驭用户
  - 减少用户的记忆
  - 保持界面的一致性 教材 p201

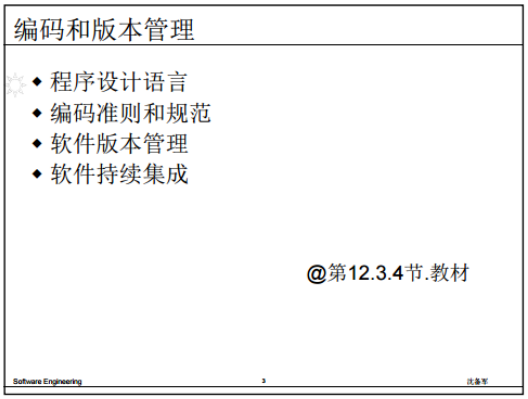
- 常见的人际交互方式
  - 问答式对话
  - 直接操纵
  - 菜单选择填表
  - 命令语言

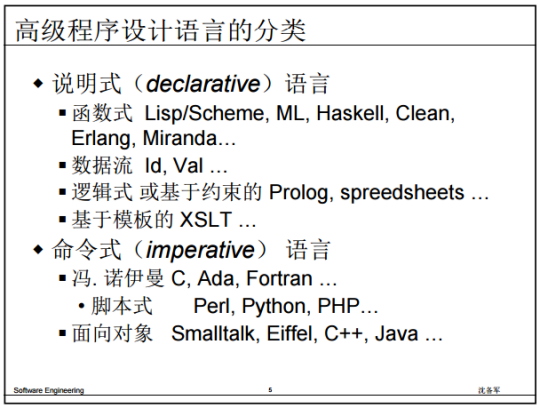
- 用户界面设计过程是“迭代”的，包括：用户分析，界面设计，界面原型开发，界面评估
  教材 p202-203
- 根据用户的特点设计人机界面
  - 用户分类：外行型，初学型，熟练型，专家型

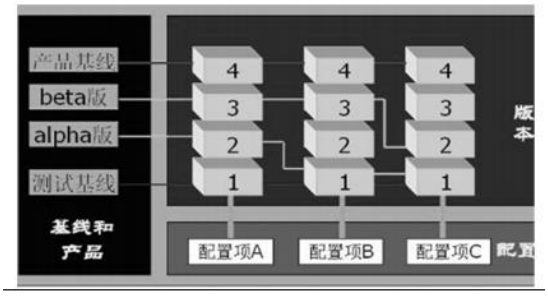
- 界面设计的问题
  - 系统响应时间：
    - 从用户执行某个控制动作到软件做出响应的时间
    - 需要考虑响应时间长度
    - 需要考虑响应时间可变性
    - 即使反馈操作信息（进度条等）
  - 帮助设施
  - 出错处理
    - 防错处理
  - 菜单和命令交互
  - 应用系统的可访问性
  - 国际化
    - 特别要留意：文字编码（unicode），颜色，货币、度量单位，日期格式，人的名字、电话号码、通信地址，风俗习惯，阅读顺序或习惯
  - 合理的布局和合理的色彩
  - 移动界面设计的问题

- 移动界面世界的挑战
  - 尺寸小，界面应简单，只显示重要的内容；根据用户的行为特点和位置推荐用户喜欢的信息；减少不必要的文本输入
  - 移动设备种类多，界面尺寸不同，要求界面有自适应性
  - 用户的层次、爱好和习惯众多，易用性要求更高，提供个性化服务
- 创新：利用移动设备的感知系统
  - 多点触控，地理定位，运动方向，手持方向，语音输入，照相摄像
  - 实时通知，设备连接，刷卡扫描，电子标签，陀螺仪

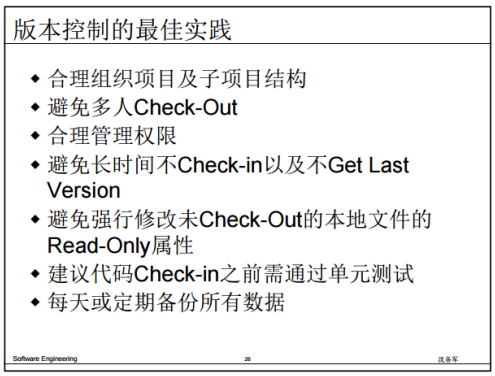
09- 编码和版本管理

- 

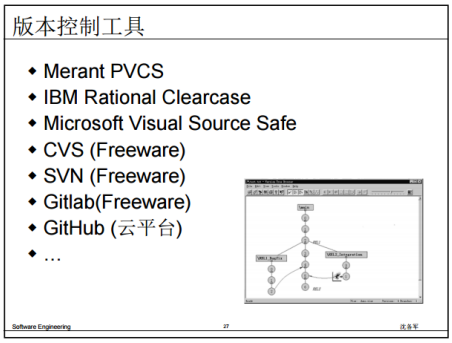
- 

- 软件版本管理
  - 软件版本管理的基本实体是软件配置项（Software Configuration Item）
    - 一个软件配置项是在软件生存周期所产生或使用的一个工作产品或一组相关的工作产品，包括代码，文档，模型，数据
  - 版本 version，每个配置项的版本演化历史可以形象的表示成为图形化的版本树
  - 基线 baseline
    - 基线是项目储存库中每个配置项版本在特定时期的一个快照，它提供一个只是标准，随后的工作基于此标准并且只有经过授权后才能变更这个标准
    - 

  - 版本控制：管理每个配置项的变更履历，控制基线的生成
  - 配置库 CM Repository
    - 存储配置管理信息及软件配置项的版本信息
  - Check in 和 check out



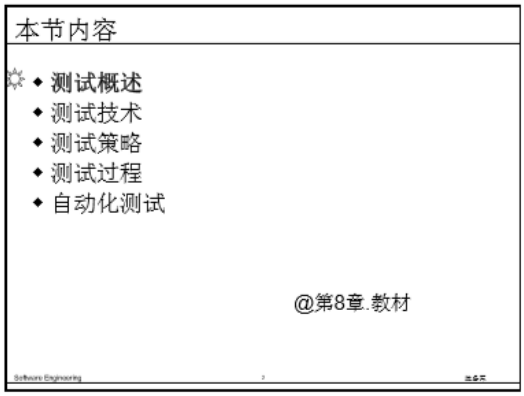
- 版本控制工具



- 本地版本控制系统
- 集中化版本控制系统：CVS, Subversion， Perforce
- 分布式版本控制系统 git

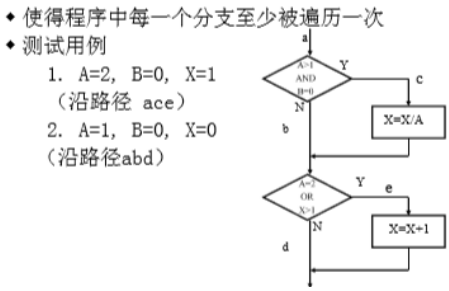
- 持续集成
- 持续集成是一种软件开发实践，即团队开发成员经常集成它们的工作，通常每个成员每天至少集成一次，也就意味着每天可能会发生多次集成。

10- 软件测试



- 测试的目的是发现软件的错误，从而保证软件质量；
  - 测试与调试的却别在于，测试可以定位和纠正错误，并且保证程序的可靠运行

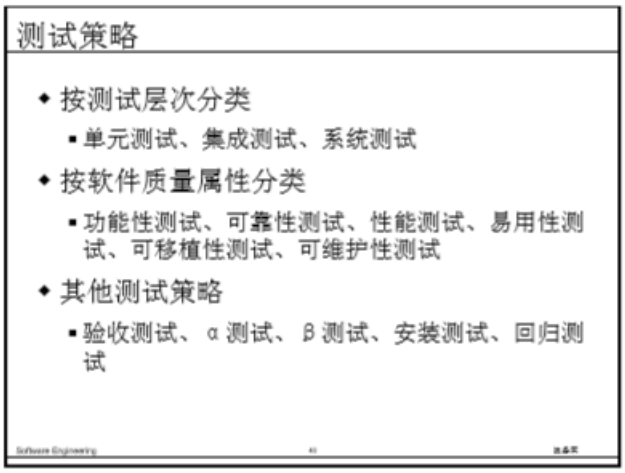
- 白盒测试，又称玻璃盒测试
  - 语句覆盖法：使得程序中每一个语句至少被遍历一次
  - 判定覆盖（分支）：使得程序中每一个分支至少被遍历一次



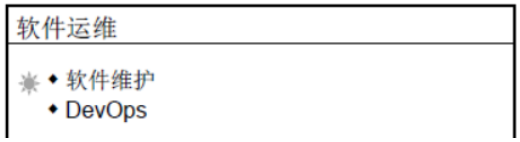
  - 条件覆盖：使得每个判定的条件获取各种可能的结果
  - 判定/条件覆盖：使得判定中的条件取得各种可能的值，并使得每个判定取得可能的结果
  - 条件组合覆盖：使得每个判定条件的各种可能组合都至少出现一次
  - 路径覆盖：覆盖程序中所有可能的路径
- 黑盒测试
  - 等价类划分法：等价类划分的办法是把程序的输入域划分成若干部分，然后从每个部分中选取少数代表性数据当作测试用例。输入的数据划分为合理等价类和不合理等价类
  - 边界值划分法：取边界；如果程序规格说明中提到的输入或输出域是个有序的集合（顺序文件，表格等），就应注意选取有序集的第一个和最后一个元素作为测试用例
  - 错误推测法：猜测被测程序在哪些地方容易出错
  - 因果图法
- 例：三角形测试

1、你所设计的测试样本中，是否包含有效的不等边三角形（例如：1,2,3不算）？
2、是否包含有效的正三角形？
3、是否包含有效的等腰三角形（2，2，4不算）？
4、是否包含三个有效的等腰三角形，而这三个三角形拥有相同的三边长，只是边长排列的顺序不同（例如：3,3,4、3,4,3和4,3,3）？
5、是否包括其中一个输入值为0的测试样本？
6、是否包括其中一个输入值为负数的测试样本？
7、是否包含三个输入值都大于0的测试样本，且其中有两个输入值的和等于第三个输入值？
8、是否包括三个测试样本，而这三个样本都与第7条所述的测试样本彼此相同，只是输入值的排列顺序不同（例如：1,2,3、1,3,2和3,2,1）？
9、是否包含三个输入值都大于0的测试样本，且其中有两个输入值的和小于第三个输入值（例如：1,2,4获12,15,30）？
10、是否包括三个测试样本，而这三个样本都与第9条所述的测试样本彼此相同，只是输入值的排列顺序不同（例如：1,2,4、1,4,2和4,2,1）？
11、是否包括0，0，0这个测试样本？
12、是否包括输入值不为整数的测试样本？
13、是否包括输入值的个数不为三的测试样本（例如：指输入两个整数值或是输入超过三个整数值）？
14、对于你所写下的测试样本，除了指定了输入的三个值之外，你是否还写下了他们预期的输出结果？

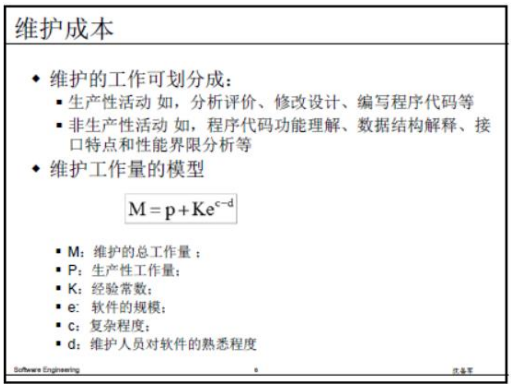
- 测试策略

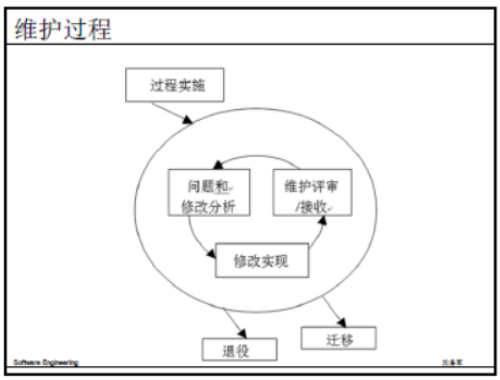


11- 软件运维

- 

- 

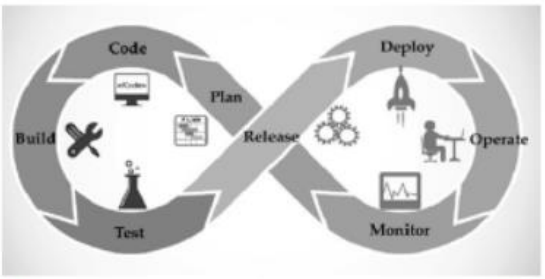


- 

- 软件维护技术：程序理解，逆向工程，再工程

DevOps

- DevOps（Development & Operations）开发运维一体化
  - （持续交付）快速实现一行代码的变更，到软件交付到用户手中
  - （自动化）从代码提交到最终交付用户只需要按下按钮，自动化每一个工作环节，及时收到用户反馈



DevOps过程

Plan：项目规划、设计等

Code：代码开发和审阅，版本控制等

Build：项目构建、打包

Test：单元测试、集成测试、负载测试等

Deploy：应用的配置与部署

Operate：应用的再配置、再部署、升级等

Monitor：应用运行时监控、最终用户体验