1 概论与核心思想

2019年4月10日 22:43

0 概论

什么是“软件”？

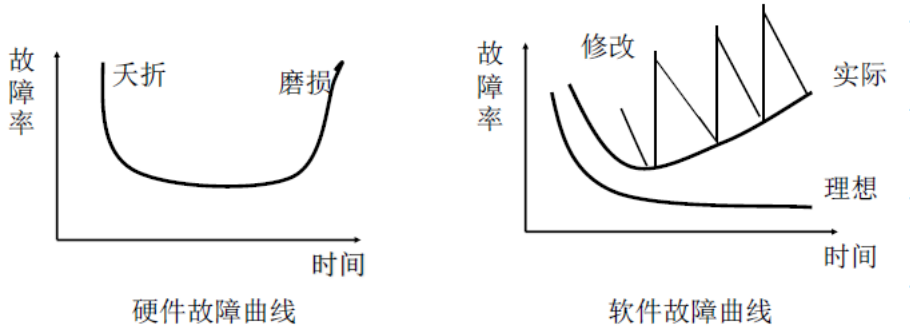
软件(Software)：一组对象或项目所形成的一个“配置”，由程序、文档和数据等部分构成

- 程序(program)：可被计算机硬件理解并执行的一组指令，提供期望的功能和性能；
- 数据(datastructure)：程序能正常操纵信息的数据结构；
- 文档(document)：与程序开发、维护和使用有关的图文材料

软件的四大特征

- 复杂性(complexity)
  - 软件要解决的现实问题通常很复杂，数据、状态、逻辑关系的可能组合导致了软件本身的复杂性；
  - 软件无法以“制造”的方式被生产，只能采用手工开发方式，这是一种人为、抽象化的智能活动(智力密集型)，与人的水平密切相关，人类思维的不确定性导致了开发过程的复杂性；
- 不可见性(invisibility)
  - 尚未完成的软件是看不见的，无法像产品一样充分呈现其结构，使得人们在沟通上面临极大的困难，难以精确的刻画和度量。
- 易变性(changeability)
  - 软件所应用的环境由人群、法规、硬件设备、应用领域等因素汇集而成，而这些因素皆会频繁快速的变化。
- 一致性(conformity)
  - 各子系统的接口必须协同一致，而随着时间和环境的演变，要维持这样的一致性通常十分困难。

软件为何需要不断的变化?

- “变化”是永恒的主题:
  - 软件必须不断的变化以适应新的计算环境或新技术的发展;
  - 软件必须通过不断的功能增强以实现新的业务需求;
  - 软件必须通过扩展以与其他软件系统进行互操作;
  - 软件必须被不断的重构以使其生命周期得以延续;
  - 

  - 软件不会磨损和老化，但维护困难
- 遗留系统(Legacysystem)：仍在使用中的软件系统，可满足客户需求，但很难以“优雅的”方式对其进行演变以适应新需求或新环境;
- 60%的软件维护费用用于向遗留系统增加新功能，17%用来修正遗留系统中的bug

软件为何无法被“制造”？

软件工程师:

- 对于软件系统，因为技术或业务发生了变化，在建设过程中(所有需求和设计完成后)，需求需要做重大修改的情况并不罕见。如果把这种情况放到修桥的事情上，相当于当桥的地基打好后，再把桥的搭建位置移动。

软件开发技术的发展过程

- 1950-1960年代:
  - 软件=程序(Program)
  - 面向过程的软件=算法(Algorithm)+数据结构(DataStructure)
- 1970年代:
  - 软件=程序(Program)+文档(Document)
  - 软件=程序(Program)+文档(Document)+数据(Data)
- 1980-1990年代:
  - 面向对象的软件=对象(Object)+消息(Message)
- 1990年代-至今:

    - 面向构件的软件=构件(Component)+框架(Framework)
    - 面向服务的软件=服务(Service)+消息(Message)+总线(Bus)

NoSilverBullet（没有银弹）by F.Brooks

- 复杂的软件工程问题无法靠简单的答案来解决!
- 所有的软件研发都包括了essentialtask和accidentaltask：
    - 前者是去创造出一种由抽象的软件实体所组成的复杂概念结构；
    - 后者则是用编程语言来表现这些抽象的实体，并在某些空间和速度的限制之下，将程序对应至机器语言。
- 软件项目平常看似单纯而率直，但很可能一转眼就变成一只时程延误、预算超支、产品充满瑕疵的怪兽；
- 人们渴望有一种银弹(silverbullet)，能够有效解决软件研发中的两大困难：
    - 软件本身在概念建构上具有先天的困难，即如何从抽象性问题发展出具体概念上的解决方案。
    - 将概念构思施行于计算机上所遭遇到的困难。

软件危机(SoftwareCrisis)：

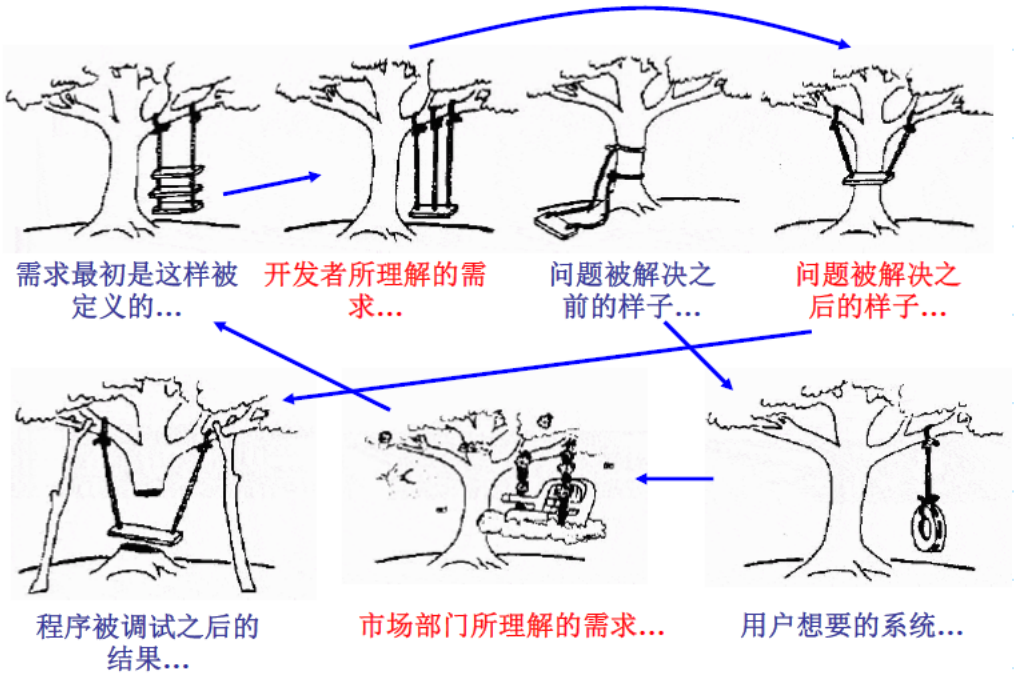
- 计算机软件的开发和维护过程所遇到的一系列严重问题
- 软件危机的表现：
    - 对软件开发成本和进度的估算很不准确，甚至严重拖期和超出预算；
    - 无法满足用户需求，导致用户很不满意；
    - 质量很不可靠，经常失效；
    - 难以更改、调试和增强；
    - 没有适当的文档；
    - 软件成本比重上升；
    - 软件开发生产率跟不上计算机应用迅速深入的趋势。

为何出现混乱的软件开发这种情况?

- 客观上：
    - 随着软件规模的急速软件产品开发的复杂度和难度随软件规模呈指数级别增长，传统的软件开发方法已经不可用了
- 主观上：软件开发人员缺乏工程性的、系统性的方法论
    - 程序员具有编程的能力，但对软件开发这一过程性较强的任务却缺乏足够的工程化思维；

- 对软件开发的一些认识的误区：软件神话(Softwaremyths)；
- 没有将“软件产品研发”与“程序编码”区分清楚；
- 忽视需求分析、轻视软件维护；

软件通常是按这样的方式构造出来的

- 

软件神话(softwaremyths)

关于软件及其开发过程被人盲目相信的一些说法

- 影响到几乎所有的角色：管理者、顾客、其他非技术性的角色、具体的技术人员；
- 看起来是事实的合理描述(有时的确包含真实的成分)、符合直觉，并经常被拿来做宣传；
- 实际上误导了管理者和技术人员对软件开发的态度，从而引发了严重的问题；
- 经典书籍《人月神话》

“软件工程”的提出

- 计算机应用发展 → 软件数量多规模大 → 软件成本高质量低
  个体化软件开发方法 → 软件维护困难
  → 软件危机 → 软件工程

“软件工程”术语的产生

- 1968年NATO的科学委员会在德国召开的一次计算机软件国际会议上，对软件开发的方法、技术进行了广泛的讨论，首次提出了“软件工程”的概念

  软件工程是为了经济的获得能够在实际机器上高效运行的可靠软件而建立和使用的一系列工程化原则;
- (IEEE)软件工程是:

  将系统性的、规范化的、可定量的方法应用于软件的开发、运行和维护，即将工程化应用到软件上;
- (国务院学位委员会)软件工程是:

  应用计算机科学理论和技术以及工程管理原则和方法，按预算和进度实现满足用户要求的软件产品的定义、开发、发布和维护的工程，或以之为研究对象的学科。

什么是“软件工程”？

- 用来开发、管理和维护软件产品的理论、方法和工具；(ISommerville)
- 开发高质量软件的生产过程，力争按期交付、不超过预算成本、尽可能的满足用户需求；(SRSchach)
- 将科学知识实际应用于计算机程序及其开发、操作和维护相关的文档的设计与构造过程；(BWBoehm)
- 技术与管理型的方法，以在预定时间与成本约束条件下系统化的生产、开发、维护和修改软件产品；(RFairley)
- 将相关工具与技术系统化的应用于软件开发的过程当中；(SConger)
- 设计与开发高质量的软件系统的技术；(SLPfleeger)
- 软件工程重要的不是技术而是如何开发软件项目的思想
  - 一种建模活动
  - 一种解决问题的活动
  - 一种知识获取的活动
  - 一种受软件工程原理指导的活动
- 归纳起来，“软件工程”是
  - 范围:
    - 软件开发过程(设计、开发、运行、维护)
    - 软件开发中应遵循的原则和管理技术
    - 软件开发中所采用的技术和工具
  - 目标:

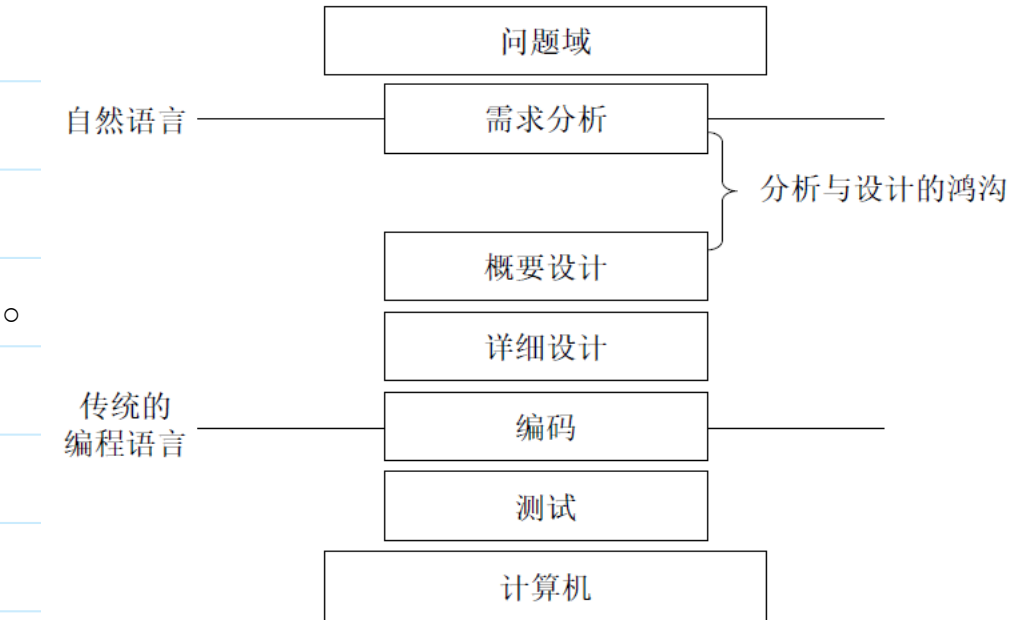
- 高质量
- 按时交付
- 控制成本
- 满足用户需求

软件工程的范畴

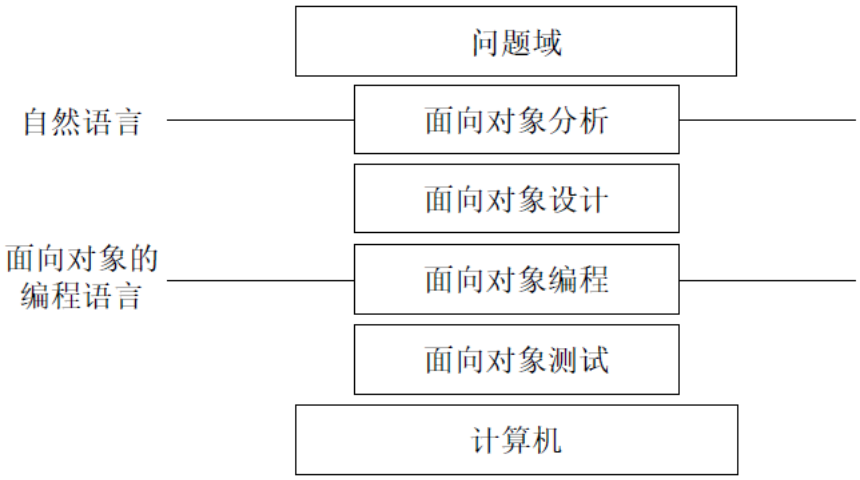
- 软件工程学
  - 软件开发技术
    - 软件开发方法学(方法)
    - 软件工具(工具)
    - 软件过程(过程)
    - 软件工程环境(工具)
  - 软件工程管理
    - 软件管理学(过程、质量)
    - 软件经济学(过程、质量)
    - 软件度量学(过程、质量)

软件开发方法学

- 软件开发方法:
  - “如何做”，Howtodo;
  - 使用预先定义好的一组模型表示方法、良好的设计技术与原则、质量保证标准等方面来组织软件开发的过程;
- 分类
  - 结构化开发方法(StructuredAnalysisandDesignTechnique,SADT)
  - 面向对象开发方法(ObjectOrientedAnalysisandDesign,OOAD)
- 传统软件工程
  - 以结构化程序设计为基础
  - 程序=数据结构+算法
  - 自顶向下：结构化需求分析->结构化设计(概要设计、详细设计)->面向过程的编码->测试



- 面向对象软件工程
  - 以面向对象程序设计为基础
  - 程序=对象+消息
  - 软件分析与对象抽取对象详细设计面向对象的编码面向对象的测试

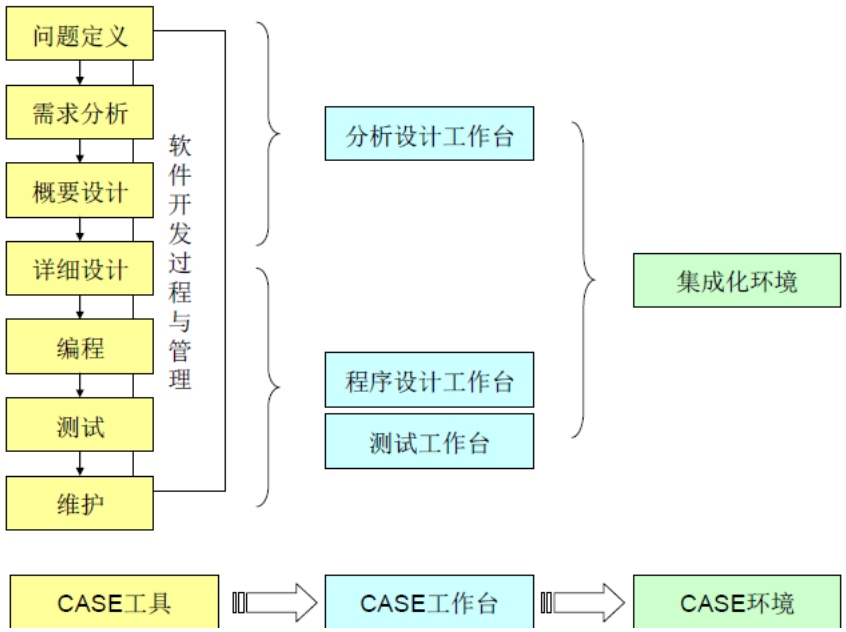


软件工具与软件工程环境

- 工具：自动或半自动的软件支撑环境，辅助软件开发任务的完成，提高开发效率和软件质量、降低开发成本
  - 项目管理工具
  - 需求管理工具
  - 设计建模工具
  - 编程与调试工具
  - 测试与维护工具

- 多个工具集成在一起，形成了软件工程开发环境
  CASE(ComputerAidedSoftwareEngineering)，全面支持软件开发的全过程

计算机辅助软件工程CASE

- 

软件工程的过程：

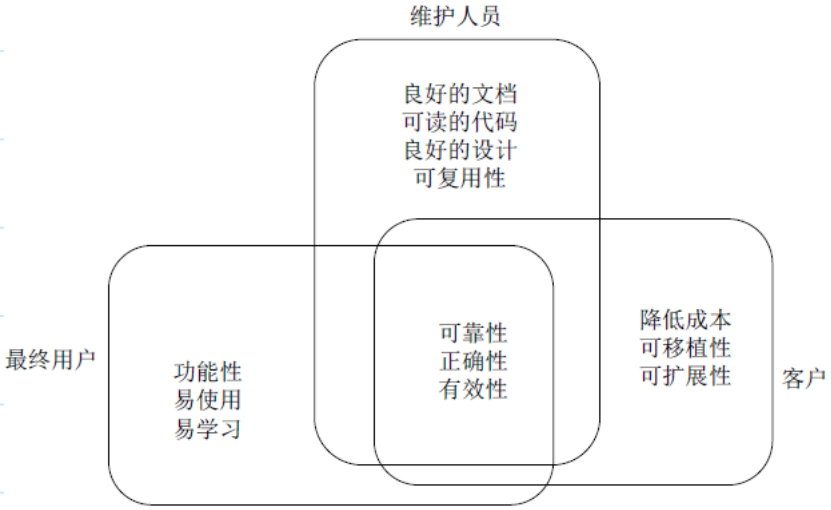
- 管理和控制产品质量的关键；
- 由一系列活动与步骤组成，如需求分析与设计、开发、验证与测试、演化与维护等；
- 定义了技术方法的采用、工程产品(模型、文档、数据、报告、表格等)的产生、里程碑的建立、质量的保证和变更的管理；
- 将人员、技术、组织与管理有机的结合在一起，实现在规定的时间和预算内开发高质量软件的目标。

软件工程管理

- 目的
  - 为了按照进度和预算完成软件开发计划
- 内容
  - 成本估算
  - 进度安排
  - 人员组织
  - 质量保证

软件质量特性

- 软件质量：
  - 软件产品与需求相一致的程度，由一系列质量特性来描述；
  - 并不取决于开发人员自身，通常与客户、用户、维护人员等提出的要求密切相关；

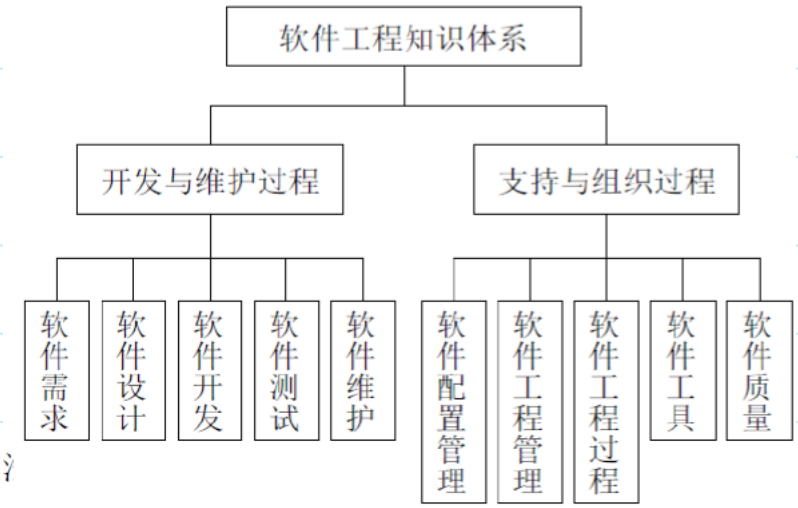


- 软件有“bug”(缺陷)，软件团队的很多人都整天和bug打交道，bug的多少可以直接衡量一个软件的开发效率、用户满意度、可靠性、可维护性
  - 软件的开发效率 开发过程中bug太多了，导致软件无法按时交付；
  - 用户满意度 用户使用时报告了很多bug，纷纷表示对生活影响很大；
  - 可靠性 这个软件经常会崩溃，这个操作系统会死机；
  - 可维护性 这个软件太难维护了，按下葫芦起了瓢，修复了一个问题,另一个问题又出来了。也没有足够的文档
- Bug=软件的行为和用户的期望值不一样；
- 微软的观点：完美的软件在世界上是不存在的，软件工程的一个重要任务就是要决定一个软件在什么时候能“足够好”，没有严重的“bug”，可以发布。

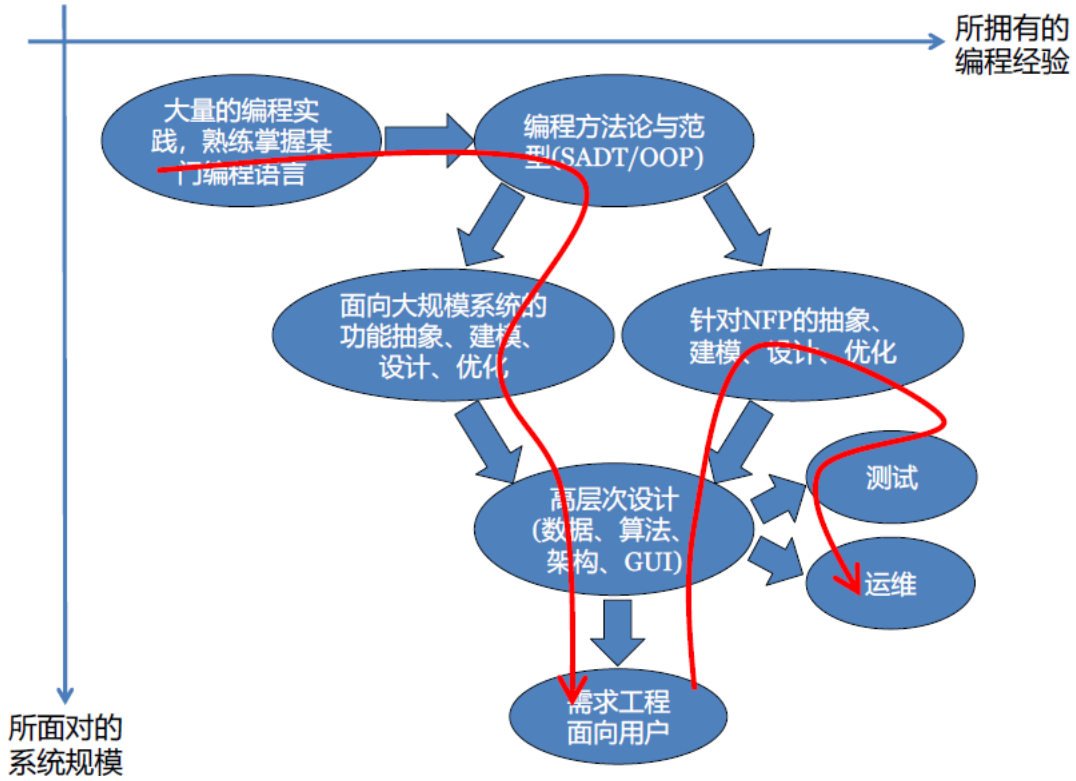
软件工程知识体系SWEBOK(SoftwareEngineeringBodyofKnowledge)

- 美国电子电气工程师学会IEEECS与美国计算机联合会ACM成立了软件工程协调委员会，于1994年开始研究软件工程知识体系（SWEBOK）；IEEECS正式发布SWEBOK30版，成为软件工程知识体系的样板。
  - 为软件工程学科定义清晰的边界；
  - 描述软件工程学科的内容、特征，以及与其他相关学科的关系；
  - 为软件工程课程计划的开发和职业资格认证提供依据；
- SWEBOK包含以下十个知识域，每个知识域下又包含若干个专题：

- 软件需求
- 软件设计
- 软件编码实现
- 软件测试
- 软件维护
- 软件配置管理
- 软件工程管理
- 软件工程过程
- 软件工程工具和方法
- 软件质量



小结：软件工程的知识与技术范畴



1 软件工程的本质

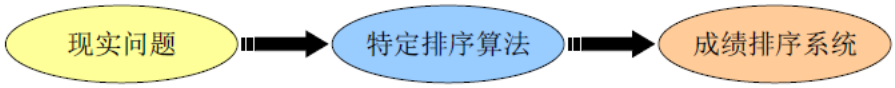
- 不同抽象层次之间的映射与转换

一个小例子

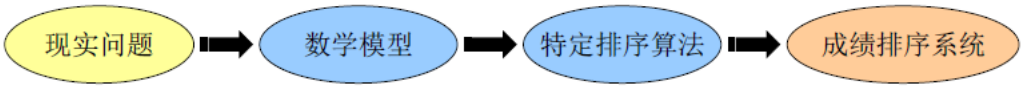
你要开发一段程序，输入班级所有人的成绩，按成绩由高到低的次序进行排序。你该如何去做?

- 方法1：直接写程序；

- 方法2：先设计算法，然后再用程序语言实现；

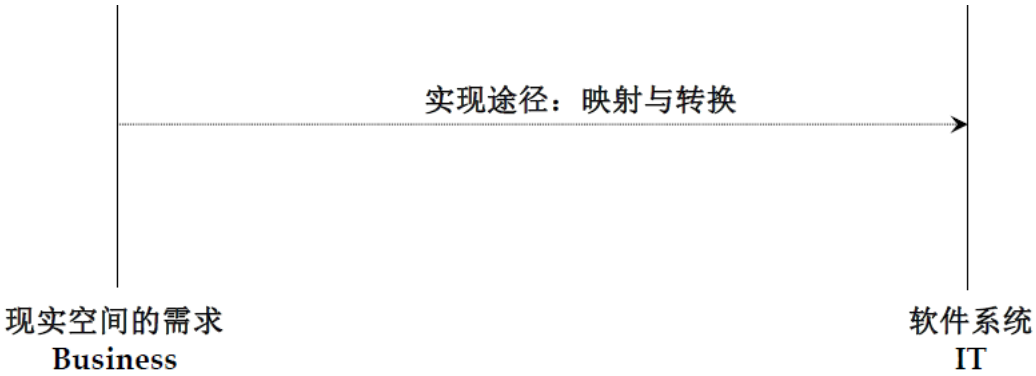


- 方法3：先建立数学模型，然后转换为算法，然后编程实现；

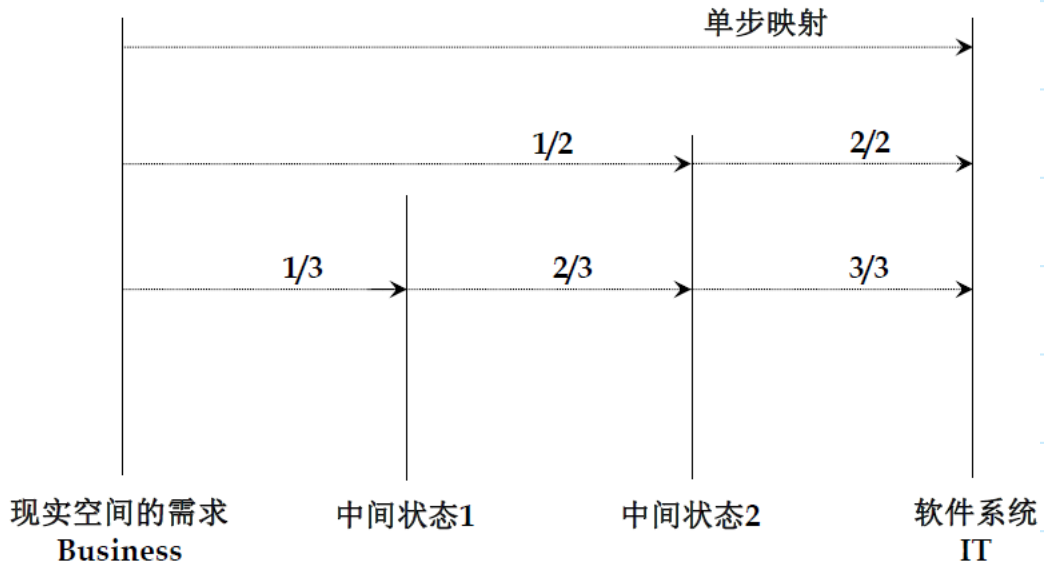


映射与转换

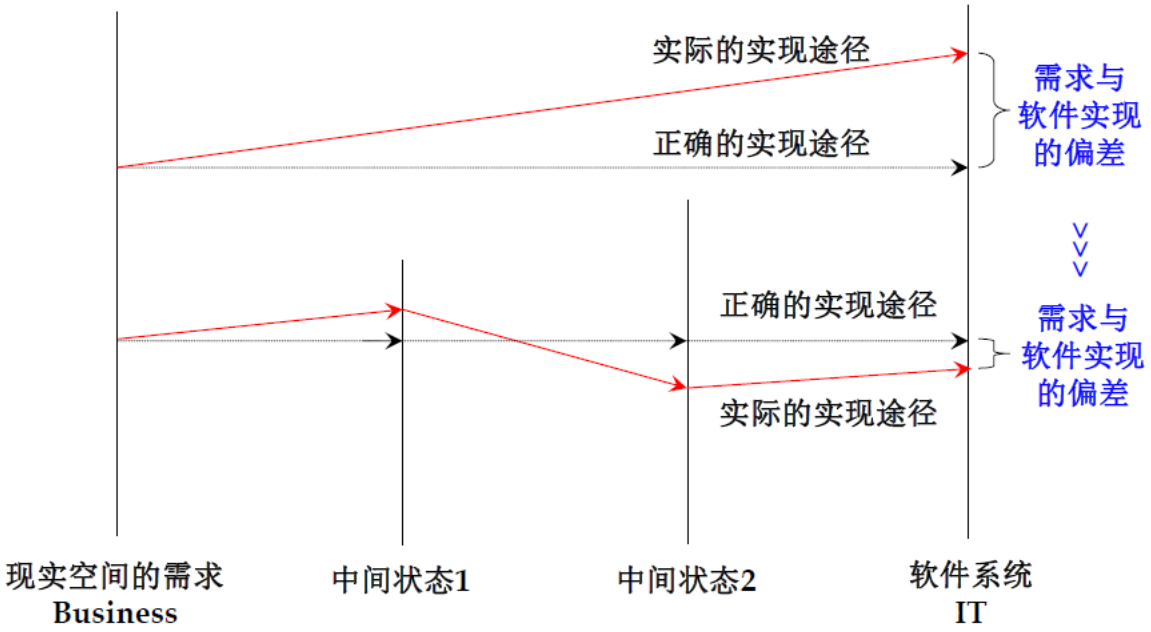
- 任何软件系统开发的共同本质在于:从现实空间的需求到计算机空间的软件代码之间的映射与转换;



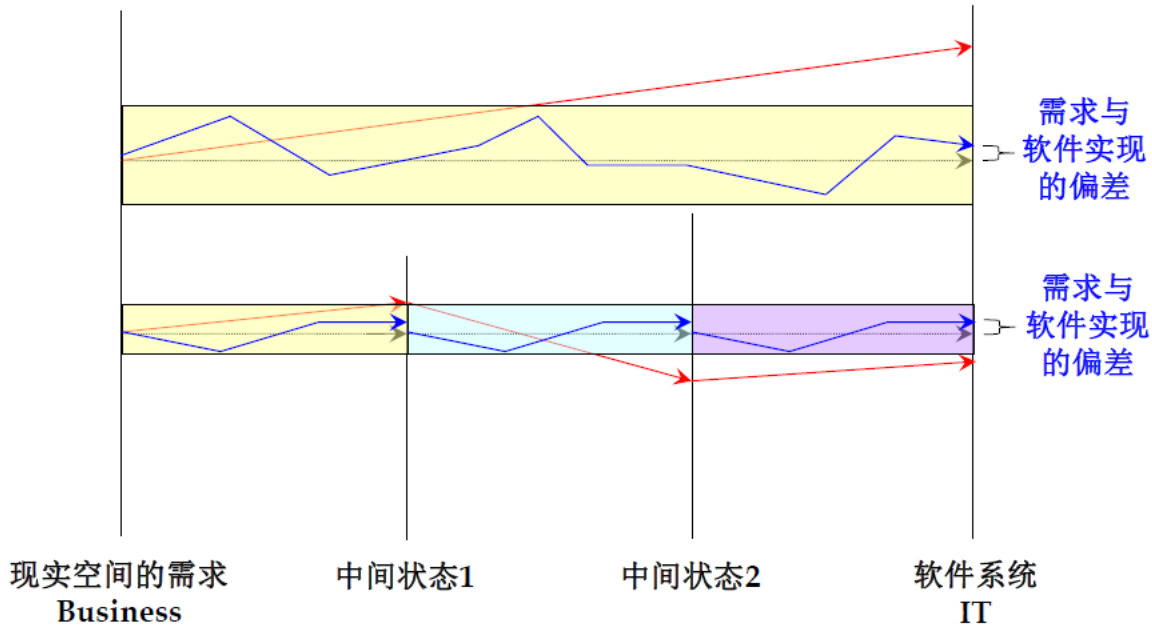
单步映射与多步映射

- 

- 思考：单步映射与多步映射的优缺点分别是什么?

- 

软件工程的作用

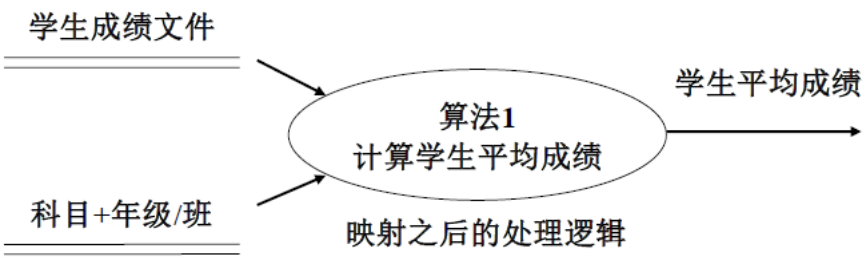
- 

- 软件工程的本质：用严格的规范和管理手段来缩小偏差，通过牺牲“时间”来提高“质量”

软件工程的两个映射

- 概念映射：问题空间的概念与解空间的模型化概念之间的映射

  例如：
  - “学生”->ClassStudent(No,Name,Dept,Grade)
  - “计算机学院大三学生张三”->ObjectStudent(120310101,张三,计算机,大三)
  - “学生成绩”->StructStudentScore(StudentNo,CourseNo,Score)
  - “张三的软件工程课成绩为85分”->ZS_SE_SCORE(120310101,软件工程,85)
- 业务逻辑映射：问题空间的处理逻辑与解空间处理逻辑之间的映射

  例如：
  - 计算某班学生的平均分数->double calculate AverageScore(Struct[]scores){冒泡排序法;}
  - 

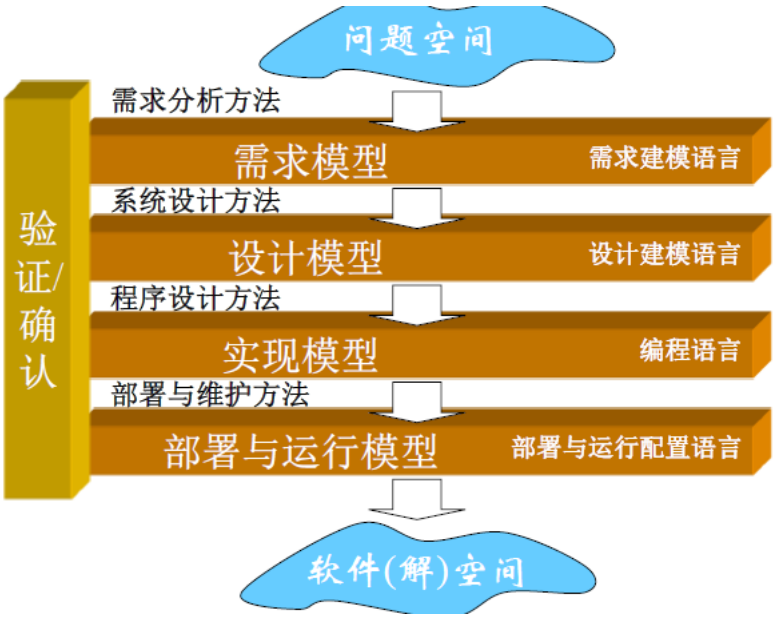
软件工程的作用

为了实现以上两个映射，软件工程需要解决以下问题：

- 需要设置哪些抽象层次——单步映射？多步映射？几步？

- 每一抽象层次的概念、术语与表达方式——公式？图形？文字？
- 相邻的两个抽象层次之间如何进行映射——需要遵循哪些途径和原则？

软件工程：不同抽象层次之间的映射过程

- 

- 需求分析：在一个抽象层上建立需求模型的活动，产生需求规约(RequirementSpecification)，作为开发人员和客户间合作的基础，并作为以后开发阶段的输入。

  现实空间的需求->需求规约
- 软件设计：定义了实现需求规约所需的系统内部结构与行为，包括软件体系结构、数据结构、详细的处理算法、用户界面等，即所谓设计规约(DesignSpecification)，给出了实现软件需求的软件解决方案。

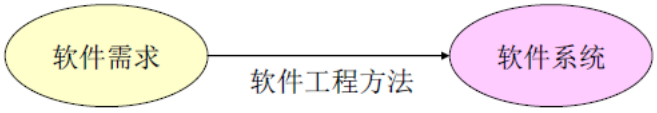
  需求规约->设计规约
- 实现：由设计规约到代码的转换，以某种特定的编程语言，对设计规约中的每一个软件功能进行编码。

  设计规约->代码
- 验证/确认：一种评估性活动，确定一个阶段的产品是否达到前阶段确立的需求(verification)，或者确认开发的软件与需求是否一致(validation)

2 软件工程所关注的目标

软件工程所关注的对象

- 产品：各个抽象层次的产出物；
- 过程：在各个抽象层次之间进行映射与转换；
- 软件工程具有“产品与过程二相性”的特点，必须把二者结合起来去考虑，而不能忽略其中任何一方。



软件工程所关注的目标

- 功能性需求(FunctionalRequirements)：软件所实现的功能达到它的设计规范和满足用户需求的程度
  - 功能1、功能2、…、功能n。
  - 完备性：软件能够支持用户所需求的全部功能的能力；
  - 正确性：软件按照需求正确执行任务的能力；
  - 健壮性：在异常情况下，软件能够正常运行的能力
    - 容错能力；
    - 恢复能力:正确性描述软件在需求范围之内的行为，而健壮性描述软件在需求范围之外的行为。
  - 可靠性：在给定的时间和条件下，软件能够正常维持其工作而不发生故障的能力。
- 非功能性需求(Non-FunctionalRequirements,NFR)：系统能够完成所期望的工作的性能与质量
  - 效率：软件实现其功能所需要的计算机资源的大小，“时间-空间”
  - 可用性：用户使用软件的容易程度，用户容易使用和学习；
  - 可维护性：软件适应“变化”的能力，系统很容易被修改从而适应新的需求或采用新的算法、数据结构的能力；
  - 可移植性：软件不经修改或稍加修改就可以运行于不同软硬件环境(CPU、OS和编译器)的能力；
  - 清晰性：易读、易理解，可以提高团队开发效率，降低维护代价；
  - 安全性：在对合法用户提供服务的同时，阻止未授权用户的使用；
  - 兼容性：不同产品相互交换信息的能力；
  - 经济性：开发成本、开发时间和对市场的适应能力。
  - 商业质量：上市时间、成本/受益、目标市场、与老系统的集成、生命周期长短等。

典型NFR举例：可用性(availability)

- 当系统不再提供其规格说明中所描述的服务时，就出现了系统故障，即表示系统的可用性变差。
- 关注的方面
  - 如何检测系统故障、故障发生的频度、出现故障时的表现、允许系统有多长时间非正常运行、如何防止故障发生、发生故障后如何消除故障、等等。
- 错误检测(FaultDetection)
  - 命令-响应机制(ping-echo)、心跳(heartbeat)机制、异常机制(exception)；
- 错误恢复(Recovery)
  - 表决、主动冗余(热重启)、被动冗余(暖重启/双冗余/三冗余)、备件(spare)；
  - Shadow操作、状态再同步、检查点/回滚；
- 错误预防(Prevention)
  - 从服务中删除、事务、进程监视器。

典型NFR举例：可修改性(modifiability)

- 可以修改什么——功能、平台(HW/OS/MW)、外部环境、质量属性、容量、等；
- 何时修改——编译期间、构建期间、配置期间、执行期间；
- 谁来修改——开发人员、最终用户、实施人员、管理人员；
- 修改的代价有多大?
- 修改的效率有多高?
- 目标：减少由某个修改所直接/间接影响的模块的数量；
- 常用决策：
  - 高内聚/低耦合、固定部分与可变部分分离、抽象为通用模块、变“编译”为“解释”；
  - 信息隐藏、保持接口抽象化和稳定化、适配器、低扇出；
  - 推迟绑定时间——运行时注册、配置文件、多态、运行时动态替换

典型NFR举例：安全性(security)

- 安全性：系统在向合法用户提供服务的同时，组织非授权使用的能力
  - 未经授权试图访问服务或数据；
  - 试图修改数据或服务；
  - 试图使系统拒绝向合法用户提供服务；
- 关注点：抵抗攻击、检测攻击、从攻击中恢复。
- 抵抗攻击——对用户进行身份认证、对用户进行授权、维护数据的机密性、限制暴露的信

息、限制访问；

- 检测攻击——模式发现、模式匹配；
- 从攻击中恢复——将服务或数据回复到正确状态、维持审计追踪。

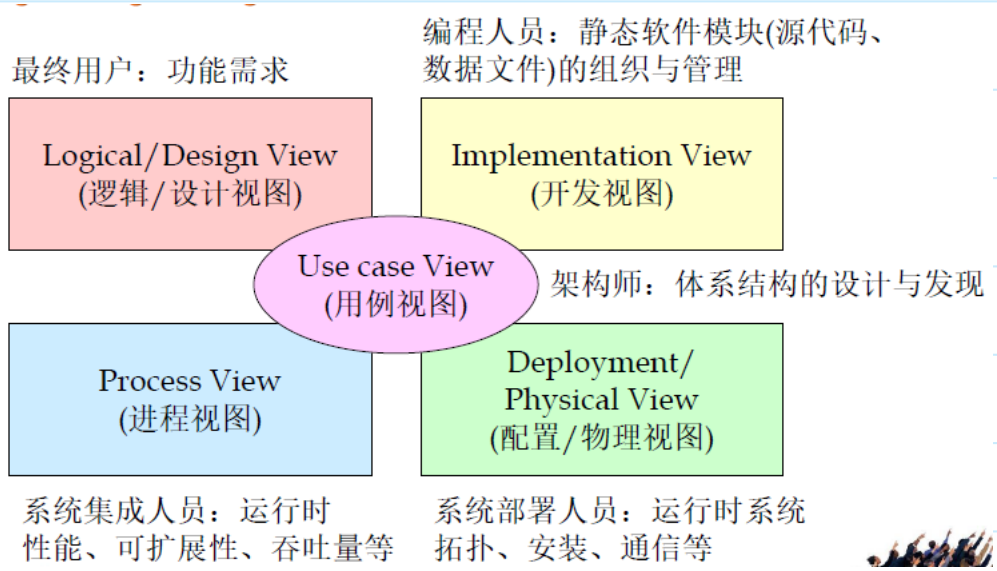
不同目标之间的关系——折中(tradeoff)

- 不同类型的软件对质量目标的要求各有侧重：
  - 实时系统：侧重于可靠性、效率；
  - 生存周期较长的软件：侧重于可移植性、可维护性；
- 多个目标同时达到最优是不现实的：目标之间相互冲突
- 解决思路：折中，足够好的软件

3 软件开发中的多角色

- 在软件开发过程中同样需要多种角色之间紧密协作，才能高质量、高效率的完成任务。
- 顾客企业(Client，甲方)：
  - 决策者(CxO)、终端用户(EndUser)、系统管理员；
- 软件开发公司(Supplier，乙方)：
  - 决策者(CxO)；
  - 软件销售与市场人员；
  - 咨询师、需求分析师；
  - 软件架构师、软件设计师；
  - 开发人员：开发经理/项目经理、程序员；
  - 维护人员。

视角不同，需求各有不同

- 

4 软件工程=最佳实践

- 软件系统的复杂性、动态性使得：
  - 高深的软件理论在软件开发中变得无用武之地；
  - 即使应用理论方法来解决，得到的结果也往往难以与现实保持一致；
- 因此，软件工程被看作一种实践的艺术：
  - 做过越多的软件项目，犯的错误就越少，积累的经验越多，随后作项目的成功率就越高；
  - 对新手来说，要通过多实践、多犯错来积累经验，也要多吸收他人的失败与教训与成功的经验。
- 当你把所有的错误都犯过之后，你就是正确的了。
- 在软件工程师试图解决“软件危机”的过程中，总结出一系列日常使用的概念、原则、方法和开发工具；
- 这些实践经验经过长期的验证，已经被证明是更具组织性、更高效、更容易获得成功；
- 大部分的这些实践都没有理论基础。

最佳实践的例子

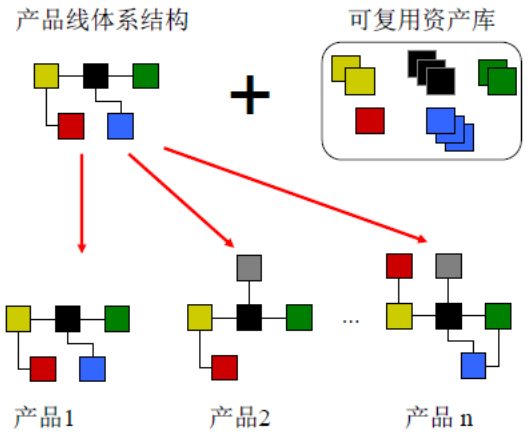
- 软件工程的七条原理(BWBoehm,1983)
  - 用分阶段的生命周期计划严格管理
  - 坚持进行阶段评审
  - 实行严格的产品控制
  - 采用现代程序设计技术
  - 结果应能清楚地审查
  - 开发小组的人员应少而精
  - 不断改进开发过程
- IBMRUP最佳实践原则：
  - 迭代化开发
  - 需求管理
  - 使用基于构件的体系结构
  - 可视化软件建模
  - 持续质量验证
  - 控制软件变更
- 与顾客沟通的最佳实践原则：
  - 倾听；

    - 有准备的沟通；
    - 需要有人推动；
    - 最好当面沟通；
    - 记录所有决定；
    - 保持通力协作；
    - 聚焦并协调话题；
    - 采用图形表示；
    - 继续前进原则；
    - 双赢
- 在你自己展开实践之前，别人的任何经验对你来说都是概念——抽象、空洞、无物
- “最佳实践”：沟通阶段应做的事情：
    - 识别出你需要与客户方的哪些人沟通；
    - 找出沟通的最佳方式；
    - 确定共同的目标、定义范围；
    - 评审范围说明，并应客户要求作出修改；
    - 确定若干典型场景，讨论系统应具备的功能/非功能；
    - 简要记录场景、输入/输出、功能/非功能、风险等；
    - 与客户反复讨论、交换意见，对上述内容进行细化；
    - 与客户讨论，为最终确定的场景、功能、行为分配优先级；
    - 评审最终结果；
    - 双方签字；

5 软件工程的核心概念

- 概念和形式模型
- 抽象层次
- 大问题的复杂性：分治(DivideandConquer)
    - 将复杂问题分解为若干可独立解决的简单子问题，并分别独立求解，以降低复杂性；
    - 然后再将各子问题的解综合起来，形成最初复杂问题的解。
    - 核心问题：如何的分解策略可以使得软件更容易理解、开发和维护?
- 复用(Reuse)
    - 在一个新系统中，大部分的内容是成熟的，只有小部分内容是全新的。
    - 构造新的软件系统可以不必每次从零做起；
    - 直接使用已有的软构件，即可组装成新的系统；
    - 复用已有的功能模块，既可以提高开发效率，也可以改善新开发过程中带来的质量问

题；

- 

Don't re-invent the wheel !
Don't Repeat Yourself
(DRY)!

- 折中(Trade-off)
  - 不同的需求之间往往存在矛盾与冲突，需要通过折中来作出的合理的取舍，找到使双方均满意的点。
  - 例如：
    - 在算法设计时要考虑空间和时间的折中；
    - 低成本和高可靠性的折中；
    - 安全性和速度的折中；
  - 核心问题：如何调和矛盾(需求之间、人与人之间、供需双方之间，等等)
- 一致性和完备性
- 效率
- 演化(Evolution)
  - 软件系统在其生命周期中面临各种变化；
  - 核心问题：在设计软件的初期，就要充分考虑到未来可能的变化，并采用恰当的设计决策，使软件具有适应变化的能力。
  - 即：可修改性、可维护性、可扩展性；
  - Continuing change and increasing complexity