Lecture2-1 计算机系统的设计和应用是复杂的

1. 常见 CS 课程的 3 个维度

- **THEORY related**：概念，模型，数学，算法，原理， 机制，方法
  - 需要理解抽象的东西
- **STSTEM and TOOLS related**：HW，网络，操作系统，PLs， ide，DBMS，客户端，服务器，虚拟机，云中的容器
  - 需要正确理解、设置和使用它们
- **DESIGN related**：根据需求（问题），需要使用理论和工具来创建 / 实施 / 测试解决方案

2. 软件设计

Complex 和 Complicated

Computer System Design and Application is **Complex**

但是 Complex（错综复杂的）≠ Complicated（复杂难懂的）

- Complex = composed of many simple parts related to one another
- Complicated = not well understood, or explained

软件设计是复杂的且难懂的

- 软件规模增长带来要素数量增长，熵增长必然导致混乱（热力学第三定律）
- 准确定义需求很难（用户不知道要什么，只知道不要什么），需求蔓延是软件项目失败的首要原因
- 不同的设计可以实现同样的功能(系统结构 / 行为 / 功能的关系），判断一个设计是否是好的设计很难（Conway's law: 软件的结构倾向于与开发团队的组织架构具相似性）

Function-Behaviour-Structure 实体

- **Function**：设计对象的目的
- **Behavior**：可以从设计对象的结构派生的属性
- **Structure**：设计对象的组件及其关系

功能与行为相联系，行为与结构相联系，功能和结构之间没有联系

Conway 定律

> 软件组织与软件团队组织相一致的规则
> 通常是这样说的："如果你有四个小组致力于一个编译器，那么你将得到一个 4 种编译器。"
> Organizations which design systems are constrained to produce designs which are copies of the communication structures of these organizations

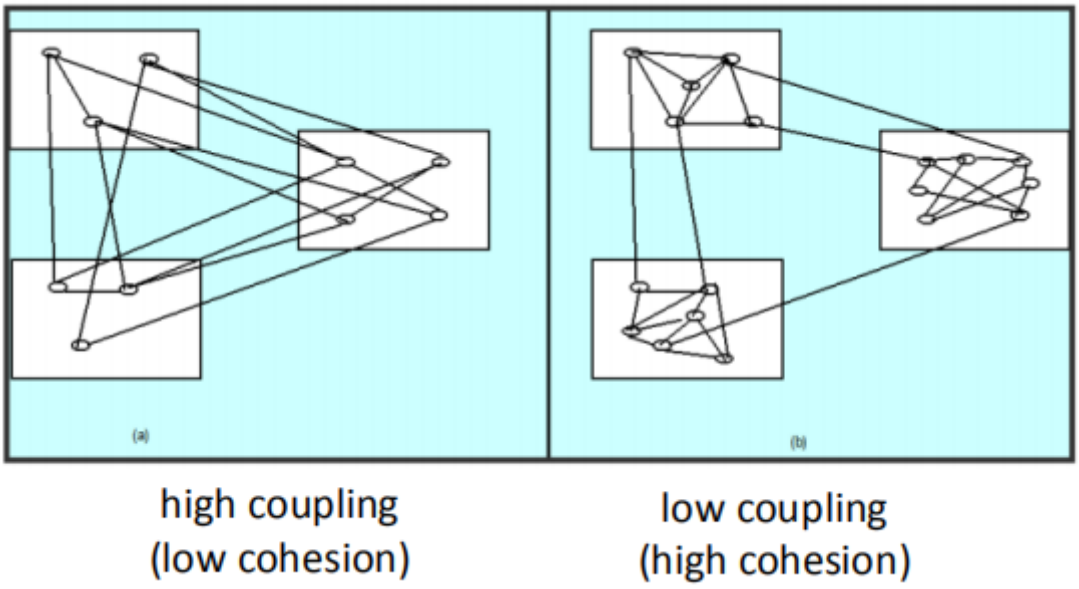
- 它是解释为什么软件开发是困难的原因之一
- 如何判断一个设计的好坏是一个很大的挑战，这很大程度上取决于经验

软件设计的两大挑战

- 如何将系统分解成模块
- 如何管理开发过程中的变更

3. 挑战1 - 如何将系统分解成模块?

关注点分离 Separation of Concerns（Soc）



- High Cohesion within modules：高内聚
- Loose Coupling between modules：低耦合
- Information Hiding：信息隐藏

模块化 Modularity 内聚 Cohesion 和耦合 Coupling

- 一个复杂的系统可以被分成更简单的模块 modules
- 把系统分解成由各个模块组成的效果叫模块化 modular
- **Separation of Concerns**（SoC，关注点分离）
  - 在处理一个模块时，我们可以忽略其他模块的细节
- 模块内都应该是**高度内聚**的
  - 模块可以理解为一个有意义的单元
  - 模块的组件是紧密相关的
- 模块之间应该表现出**低耦合**
  - 模块与其他模块之间的交互很低
  - 可以分别被理解

结构化程序设计

- 一个复杂的系统可以被分成更简单小块，被称为模块
- 分而治之
- 逐步求精
- 将复杂问题做合理的分解，再分别仔细研究问题的不同侧面（关注点）， 最后综合各方面的结果, 合成整体的解决方案

4. 挑战2 - 管理开发过程中的变更

计算机并不关心代码是如何组织的 —— 但人类很关心

更改代码既困难又昂贵，而且因为世界在变化，这是必不可少的，我们想让它变得简单和便宜

- 最小化必须更改的代码量
- 很容易知道哪些代码必须更改
- 将必须更改的代码放在一起
    - 在一个文件中更改 6 行代码通常比在 6 个文件中分别更改 1 行代码便宜

好的编程的评判标准 Criteria for Good Programs

- **Objectives 目的**：有效地（正确地）高效地运行 Run Effectively (Correctly) & Efficiently
- **Approaches 手段**：易于扩展和修改 Easy to be Extended and Modified
- **Pre-conditions 前提**：易于理解 Easy to be Understood

如何让改变更加容易和便宜

- 在定义良好的代码段中**隐藏某些信息**，以便该代码段的用户不依赖于它，并且在它发生变化时也不需要更改
- **函数的调用者不知道函数是如何工作的**：只知道它接受的参数类型和它返回的类型
    - 为了使用数据类型，您不需要知道数据类型是如何实现的，阅读文档就足够了：有哪些操作，它们做什么?
- 这样，就可以编写无需更改的代码，即使实现发生更改（是否可以编写依赖于实现的代码则是另一回事，如果不是，封装的数据类型(封装)）

信息隐藏与数据类型

> 数据类型是一组值（有着有限的存储和特定的格式），并且还有对这些值的操作
> A data type is a set of values (with limited storage and specific formats) and operations on those values.

基本数据类型 Primitive

内置在编译器 / 运行时中定义的操作

- int, double, boolean

用户自定义数据类型 User-defined

使用编程语言定义的操作

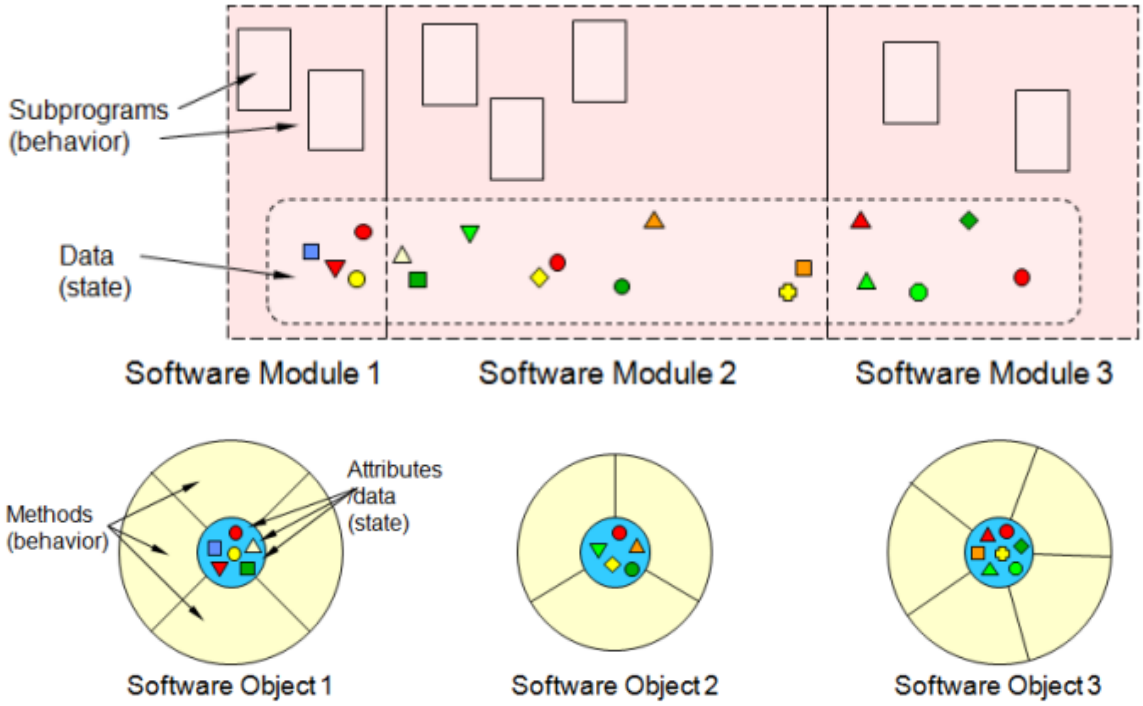
- PrinterQueue, HotelRoom, …
- 在 Java 中，String 其实是定义在 Java 标准库中的

隐藏数据的表示（信息隐藏）

在使用数据类型时，不需要知道数据类型是如何实现的，直接阅读文档：操作是什么，它们能做什么？

然后，你可以写下代码，不必考虑这些数据类型的实现发生了变化，这样的数据类型是封装的 encapsulated

5. 模块 Module vs 对象 Object



- 模块是子程序和数据的松散分组 Modules are loose groupings of subprograms and data
- 对象封装了数据 Objects encapsulate data

面向对象与面向过程

面向过程 Process-oriented

- 面向过程的更直观，因为它是以个人为中心的
    - 思考下一步该做什么，该走哪条路
- 面向流程的不能扩展到复杂的、大规模的问题
    - 大规模的问题需要人们的组织，而不是个人单独工作

面向对象 Object-oriented

- 由于分工的原因，面向对象可能更令人困惑
    - 思考如何将问题分解成任务，分配责任，协调工作
    - 这是一个管理问题
- 面向对象是以组织为中心的
    - 但是，很难设计出好的组织

面向对象的哲学

- 如果设计仅是基于软件目前要做的事情，那么会产生的问题是，未来可能发生很大的变化
- 它工作的领域变化要小得多
- 围绕领域内的事物构建软件，使其更容易理解和维护