一般产品：功能、质量、结构

通用工程： 收益-风险；过程-结果；少数-多数

- **风险 vs 收益**
- **过程 vs 结果**
- **少数 vs 多数**

工程师的特点：

- 人道无害
- 雇主
- 实事求是，恪守公心，严守纪律，蓄意学习

软件特点：复杂性；商品属性；可变性；*功能的契合性

软件家族：版本序列；产品线；产品家族；成品组件

软件的功能

- 包括基本功能、高级功能和定制功能等
- 开发人员需要根据产品需求和设计要求进行开发和测试，以确保软件的功能和特性能够满足业务需求和用户需求。

软件的质量

1. 用户反馈：软件的质量需要通过用户反馈进行评估，包括用户满意度、用户体验等。
2. 软件的自适应：软件需要具备自适应性，即能够根据用户的需求和环境变化进行调整和优化。
3. 移植性：软件需要具备良好的移植性，即能够在不同的平台和操作系统上运行，并且能够兼容各种硬件和软件环境。
4. 功能稳定性：软件的各种功能需要保证稳定性，即能够在各种情况下正常运行，并且不会出现崩溃和异常情况。
5. 性能效率：软件的性能效率需要保证，即能够在各种数据量和负载情况下保持良好的性能表现。

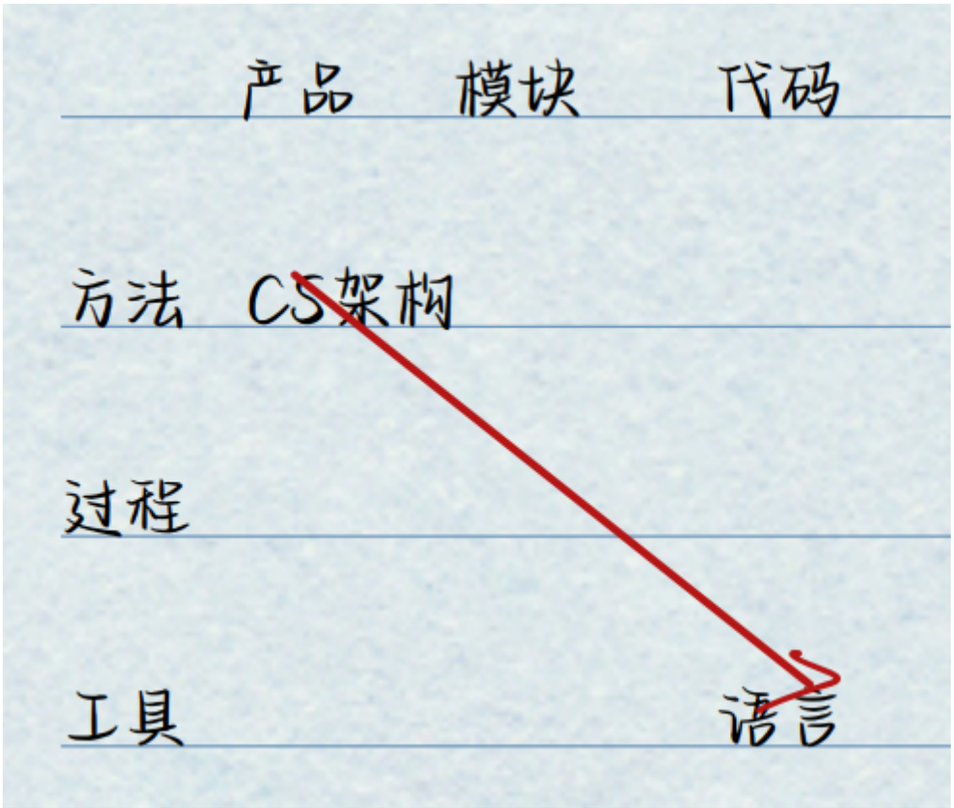
- **在软件工程领域中，没有银弹能够解决所有问题。** 没有银弹的核心原因是软工非常**复杂**，比过程更重要的是人

软件天花板

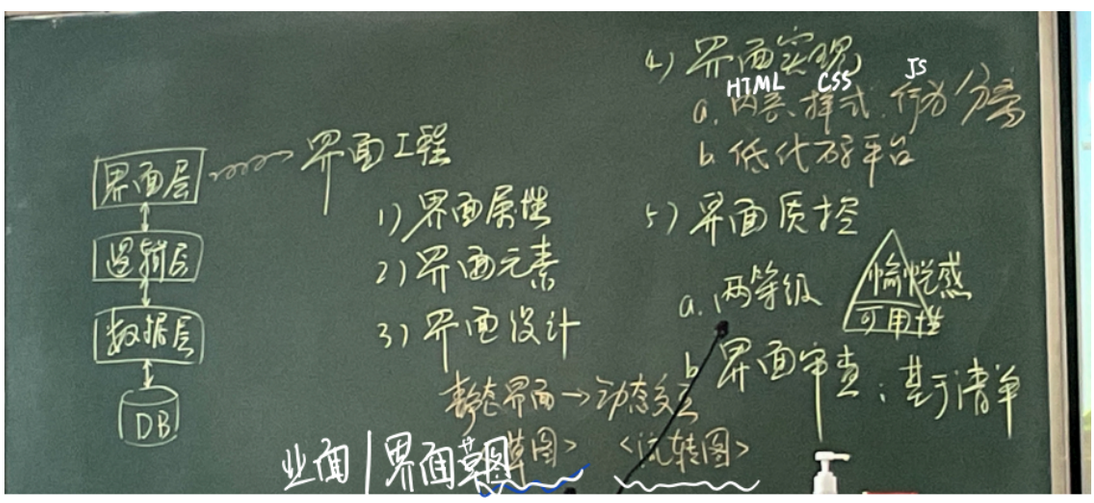
- **天花板的核心是复杂度太高**，进而规模的上升导致复杂度更快的增长，而人类的能力无法承受复杂度的增长；
- 期望设计一个结构，使得复杂度线性增长而非超线性增长，这属于架构师的工作了
    - 如何设计架构?
    - **判断哪个架构最好?**
        - 适合产品的设计
        - 可行性、可构造性
        - 可变性（持续）**【与实体工业的区别】**
        - **因为软件有如下特性**
            - **复杂性**
            - **可变性**
            - **商品属性**
    - **软件三层金字塔（从上到下）：质量、功能、结构**
    - **软件结构形态**
        - **单体架构**
        - **模块集成**：模块化是一种常见的软件架构设计方法，它能够将系统拆分为多个独立的模块，每个模块之间的关系简单明了。这种方法适用于变更频率较低的中小型软件项目。

        - 缺点：模块化使得关系数量上升，复杂性超线性增长
        - 使用与变更频率低的中小场合
    - **（大型）平台服务【微服务】**：平台服务是一种分布式系统架构，能够实现热插拔和冗余存储，同时具备高可用和高可伸缩性。这种方法适用于大型复杂的软件系统。
        - 分布式
        - 热插拔
        - 冗余存储
    - **系统乌合：取源于“乌合之众”**
        - "乌合之众"是指没有明确目标和领导的人群，他们的行动和思想容易受到外界影响。系统乌合设计方法的缺点可能包括以下几个方面：
            1. 缺乏统一的规划和控制
            2. 不兼容和冲突问题
            3. 维护成本高
            4. 可扩展性受限
- **平台+服务会成为主流**
- **平台就是一个软件产品**
- **从产品为中心 -> 服务为中心**
    - 提高开发效率
    - 可扩展性和可维护性
    - 降低系统的复杂性，使得系统更易于维护
    - 提高系统的可用性和稳定性、
    - 灵活（热插拔）
- **未来：平台会被垄断；服务会被分散；云服务一定是主流**
    - 对服务端/云端：最上游被垄断
        - 资金
        - 时间
        - 布局
        - **功能相同，适者生存，仅留胜者**
    - 对开发者和中小公司
        - 快速开发，省时、省钱、省力
        - 培训成本降低
        - 不同的服务可以使用相同的平台（大模型）
    - 对非cs领域
        - 省钱
    - 对程序员而言会被淘汰，我们需要：
        1. 大局观
        2. 对行业的见解和规划
        3. 领导能力
        4. 冲破性的技术革新
- 分久必合，合久必分：微服务（now）和serverless（future）。
    - 微服务架构是将一个大型的应用系统分解为多个小型的服务，每个服务都可以独立部署和扩展。而Serverless架构则更进一步，将服务拆分为更小粒度的函数，不需要自己管理服务器，只需要编写和部署函数即可。
- **服务 = 设计+开发+测试+托管**
    - **平台面向领域**
        - 实现领域共性需求
        - 实现领域特定架构
        - **一个领域只需要一个平台**
    - **高度集成一体化** Devops：高度集成一体化的DevOps则是指在服务的开发、测试、部署和运营过程中，采用高度集成的工具和流程，实现快速迭代和持续交付。这种DevOps方式可以提高服务的效率和质量，同时也可以降低开发和运维的成本和复杂性。

- **（如何设计平台）开发大规模平台**
  - 知识
  - 经验：架构设计经验
  - 领域需求
- **设计审查-设计质量（贯穿整个软件生命周期）-架构质量**
- **架构**
  - 架构分为**抽象架构和具象架构**
  - 跨模块型缺陷MCD
    - 解决：代码审查；测试验证；接口设计和模块化设计
  - **架构质量：**
    - 经验视角
    - 主观视角
    - 客观视角（结构；缺陷）
  - **架构质量评估：**
    - 从**经验视角、主观视角和客观视角**来评估软件系统的架构质量
    - 编码之前
      - 设计原理：模块化、高内聚低耦合、单一职责
      - 设计共识：设计模式、架构模式、UML建模
      - 在这个阶段，可以通过评审、审查等方式来评估系统的架构质量，确保软件系统的设计符合最佳实践和标准。
    - 演化之中
      - 结构视角：结构偏差；评估软件系统的结构是否符合设计原则和设计共识，以及模块之间的耦合和内聚关系是否合理
      - 缺陷视角：评估软件系统中存在的缺陷和问题，例如跨模块型缺陷（MCD）
  - 软件开发过程：设计->架构
  - **架构中心开发方法：**
    - 业务驱动
    - 复用优先
- **设计**
  - 软件设计质量：K、R、E
  - 设计于维护
    - 维护的概念
      - 维护是指在软件、硬件或其他系统运行期间，对其进行修复、更新和改进的过程。维护可以分为预防性维护和修复性维护两种类型。预防性维护是指在问题发生之前采取的措施，以防止问题的出现。而修复性维护则是指在问题已经出现之后，对其进行修复和改进。
    - 维护实践（独立维护）
    - 设计从何而来：逆向工程
      - 逆向工程是指通过分析已有的产品或系统，来了解其设计和实现的过程。通过逆向工程，维护人员可以了解一个产品或系统的内部结构和工作原理，从而可以二次开发或进行维护。
    - 设计如何使用：CIA
  - 设计于演化
    - 演化概念
    - 演化实践；演化开发
    - 架构恶化
- **设计审查需要考虑的点：**
  - **设计审查清单**
    - **Y轴：方法、过程、工具**

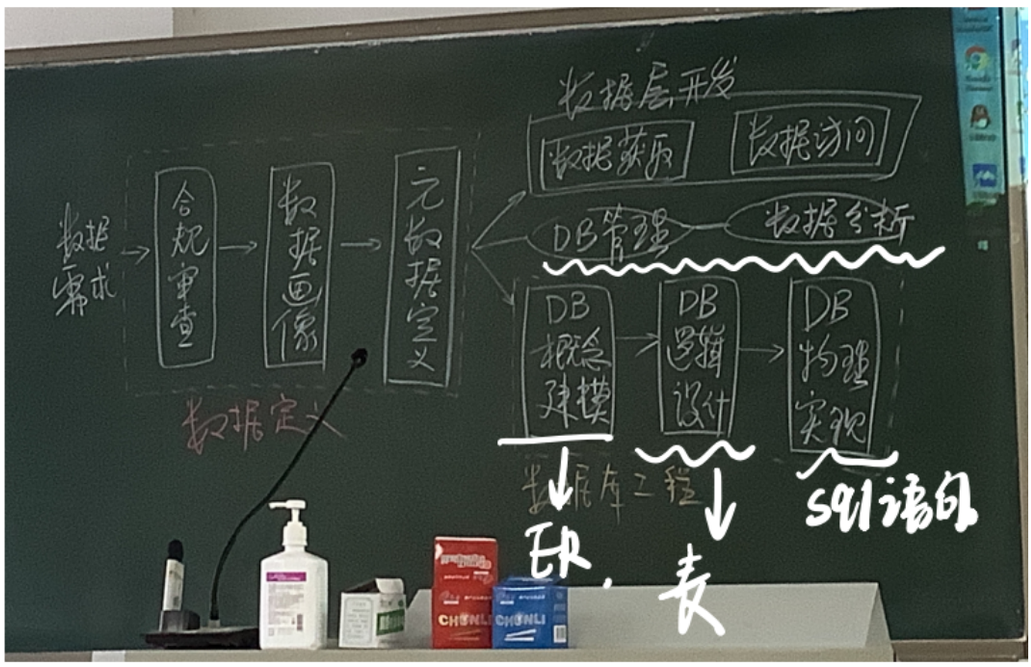
        - **X轴：产品、模式、代码**
    - 交互设计审查清单
    - 

    - 设计审查是基于清单的自查自纠
    - 软件系统的质量控制：审查+测试

- **软件架构模式：**界面层、逻辑层、数据层和DB
- 

    1. 界面层（Presentation Layer）：负责与用户交互，展示数据和处理用户的输入。在Web应用中，通常是Web页面或Web应用程序。
    2. 逻辑层（Application Layer）：负责处理业务逻辑，包括数据处理、计算、校验等。通常是一个独立的应用程序或服务。
    3. 数据层（Data Layer）：负责处理数据的存储和访问，包括数据库和数据访问层。在传统的软件架构中，数据层通常是一个独立的数据库，而在现代的云架构中，数据层常常被拆分成多个分布式存储和处理服务。
    4. 数据库（DB）：负责存储应用程序的数据。可以是关系型数据库（如MySQL、Oracle等）或非关系型数据库（如MongoDB、Redis等）。
- **界面工程：**
    - 界面属性：颜色、字体、排版
    - 界面元素：按钮、文本框、下拉框
    - 界面设计
        - **静态（草图）、动态（流程图）**

- 界面实现
  - **内容、样式、行为分离（MVC或MVVM等设计模式）**
  - **低代码平台**
- 界面质控
  - 两个等级：**愉悦感和可用性**
  - **界面审查：基于清单**

- **数据工程：**
  - 关系型数据库、NoSQL数据库、数据仓库、数据湖
  - 数据需求：**使全部功能得以执行的所有数据**
  - 

  - **数据定义**
    - 合规审查（**前提**）：合法性和安全性
      - 数据合规官（Data Compliance Officer，**DCO**）是一个组织中负责数据合规的专业人员
    - 数据画像：对数据进行深度分析和可视化，以便更好地理解数据。数据画像可以帮助人们快速了解数据的特征、质量和价值，从而更好地利用数据。
    - 定义元数据：对数据进行描述和分类
  - **数据层开发**
    - 数据获取
    - 数据访问
  - **DB管理、数据分析**：备份、恢复、性能优化
  - **数据库工程**
    - DB概念模型：ER图
    - DB逻辑模型：表
    - DB物理实现：sql语言和文件系统
  - **生产数据**：实际生产中产生的数据，包括企业、机构或个人在业务过程中产生的所有数据。生产数据是实际业务活动的结果，包括交易数据、客户数据、产品数据、供应链数据等，是企业和机构的重要资产之一。
  - **元数据**：**先导型Proactive Metadata，后生型Reactive Metadata**
  - **数据本体**（Data Ontology）是指描述数据和数据之间关系的概念模型，是知识表示和知识管理的一种方式。数据本体通常包括一组定义良好的概念和关系，用于描述领域中的实体、属性和关系，并提供一种共享和重用这些概念和关系的方式。
    - 实体：描述领域中的实体，如人、物、事等。
    - 属性：描述实体的各种属性，如名称、年龄、性别等。
    - 关系：描述实体之间的各种关系，如父子关系、雇佣关系等。

- Q：获得数据前需要做什么？
    1. 需要什么数据？
    2. 能收集么？能商用吗？
    3. 有能力保护数据吗？
    4. 基于什么条件来收集？
    5. 数据怎么使用？（合规审查，数据合规官DCO）
- **管理的三重境界：**
    - 人：都需要经理调控
        - 工程师
        - 客户、投资人
        - 用户
    - 事
        - 变更未知，不想1+1算1w次那样简单
        - 风险预测未知；时间、成本管理困难
        - CTO不管理？ 不行
        - 胶冻团队：临时组建的跨部门或跨职能团队，在短时间内迅速响应某项任务或项目，快速协作完成任务并在完成后解散的一种团队形式。
    - 物
        - 产品
        - 文档（制品）
        - 议素
            - 待实现和决议的内容
            - 带移除修复的缺陷
- **软件管理三部分**
    - 产品管理
    - 项目管理
    - 过程管理
- **软件管理的三种项目**：
    - **软件开发项目**
        - 项目规划
            - **进度规划**
                - **项目估算**
                    - 项目规模
                    - 技术难度
                    - 人力资源
                - **软件里程碑**
            - **质量规划**
                - **质量保障（QA）**
                    - 质量标准和规范
                    - 过程管理
                    - 质量培训
                - **质量控制（QC）【审查+测试】**
                    - 质量检查和测试
                    - 缺陷管理
                    - 质量评估审查
        - **项目监管**
            - 项目风险
            - 项目团队
        - **项目改进**

- **软件运营项目**
- **软件维护项目**
  - 逆向开发，开发者小队+维护二次开发者小队
- **软件演化项目**
  - 常规
  - 非常规
- **开发过程需要考虑的：规模 -> 工作量 -> 工期、用人 -> 成本**
- 规模增长 + 软件具有复杂性和频变性，导致软件有三高：**价值、成本、风险 【考点】**
- 这些因素都会对软件开发的工作量、工期和成本产生重要的影响，做决策规划之前需要确认和考虑：
  - 立项谈判
  - 目标产品
  - 合同价格
  - **交付日期**
  - **完工标准**