第1课 软件架构 & 架构师

北京交通大学软件学院

part 1-软件架构

目录

01 01 软件架构的基本概念

02 02 软件架构的收益和挑战

03 03 软件架构的基本策略

04 04 软件架构的整体流程

01

软件架构的概念

什么是架构？

- 从词源上，来自希腊语：
- Architecture = ἀρχιτέκτων
- ἀρχι-"chief" 首要的
- τέκτων"creator" 创造者
- 架构 = 规划、设计、构建建筑的过程和产品，或者其它结构

维特鲁威 (Vitruvius)《建筑十书》

- 写于公元前30年-15年之间（西汉于公元8年灭亡）
- 书中描述了建筑设计的各方面
  - 建筑师的必要素质：理论，时间，道德品质
  - 城市规划，建筑材料，神庙，公共建筑，私人住宅，水利工程，机械工程，天文
- 提出了好建筑的3个核心属性
  - usefulness易于使用：有用，好用；
  - durability持久坚固：健壮，保持良好的状态；
  - elegance美观：给人愉悦感
- 同样适用于软件系统

什么是软件架构？（1）

- 软件架构： 是指系统在特定的环境中具有的结构，体现在其组成元素、关系以及设计和演进原则中的基本概念或属性。架构【ISO/IEC/IEEE 42010:2011, 3.2】
  - 系统的蓝图： 它描述了系统如何组织和构建。
  - 关键决策的集合： 它包含了关于系统结构、行为和实现的重要的设计决策。
  - 指导开发的框架： 它为开发团队提供了一个共同的理解和指导，确保系统按照预期的方式构建和演进。
- 架构描述：
  - 指的是一份清晰、明确的文件，包括模型，文本，图形，它表达了一个软件系统的架构。
- 架构设计：
  - 是指构建、定义或创造一个系统架构的整个过程。 包括识别系统的关键需求、做出重要的设计决策、并最终形成一个清晰、完整的架构描述。

什么是软件架构（2）

- 系统的基本结构，包括：
  - 软件元素
  - 元素之间的关系
  - 元素的属性，关系的属性

架构和设计的区别

- 架构和设计没有显著的区别
- 架构更多的关注：
  - 软件系统的整体结构
  - 系统中重要的设计决策
  - 修改他们将会导致很高的代价
- “所有的架构工作都是设计工作，但并非所有的设计工作都是架构工作”
  - by G.Booch

建筑架构 vs 软件架构

相似性

1. 都是很复杂的系统
2. 都是团队开发
3. 都是给人使用的
4. 都有自己的风格，模式，战术
5. 都受到潮流和趋势的影响

建筑架构 vs 软件架构 区别

01

建筑架构

环境更加稳定

物理可见的产品或服务

受到物理限制，难以改变

具有很长的传统和历史（作为例子）

02

软件架构

环境改变非常快

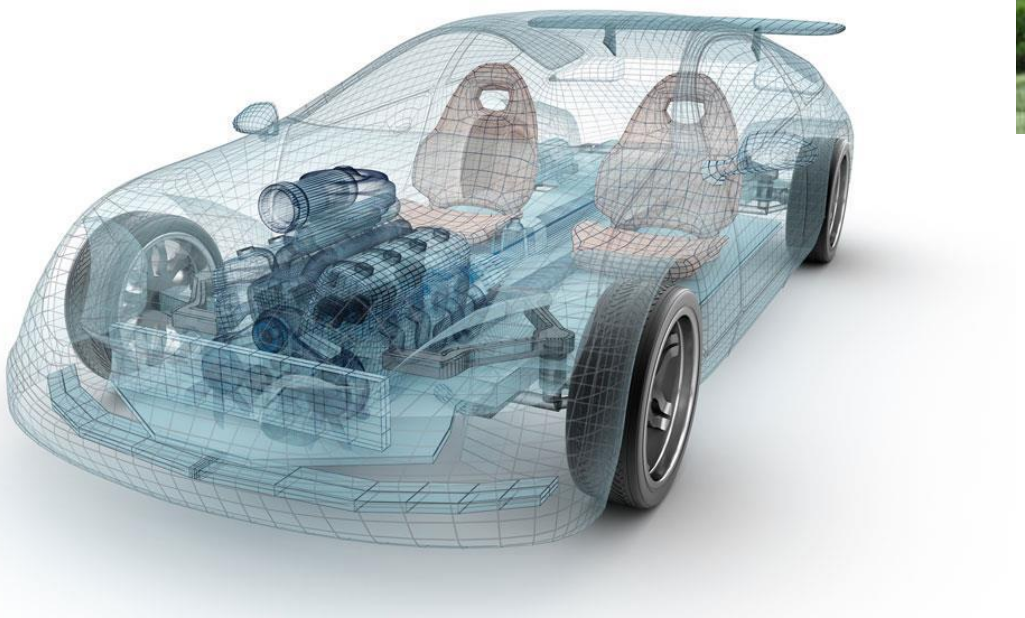
虚拟的产品或服务

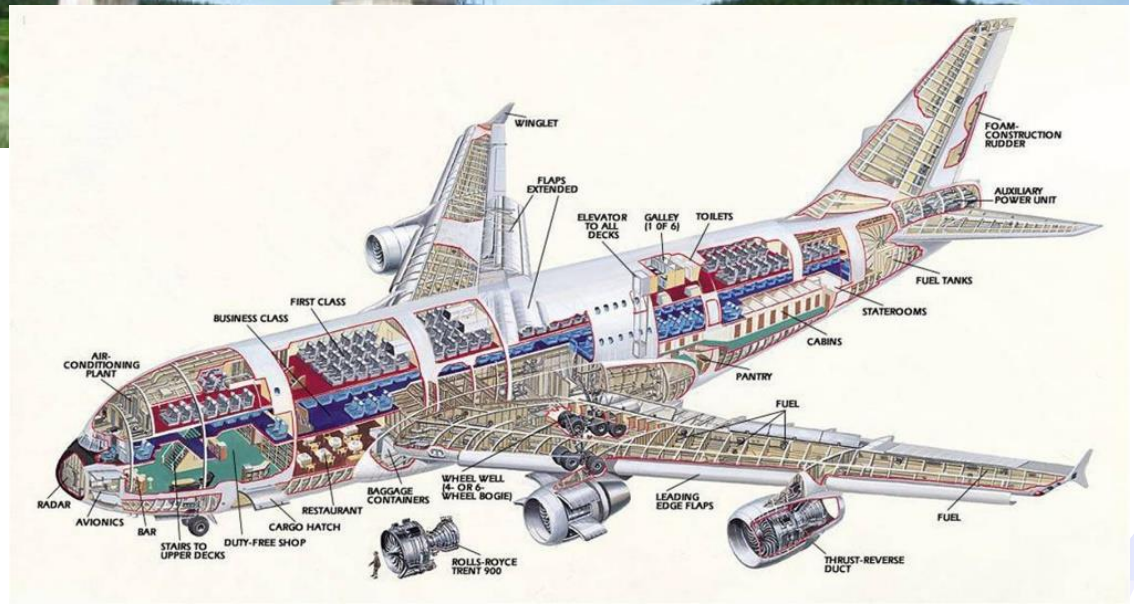
没有物理限制，很容易改变

相对来说比较新的学科（OOP）

其他类似学科

- 土木工程
- 机械工程
- 航空工程





软件架构之外的其它架构

- 业务架构
- 企业架构
- 系统架构
- 信息架构
- 数据架构
- ……
- 共同点：结构和视图

02

软件架构的收益和挑战

软件架构的收益

1. 为团队提供清晰的愿景和路线图
2. 技术领导力以及更好的协调
3. 解答与重要决策相关的问题：质量属性、约束和其他横切关注点
4. 识别并降低风险
5. 方法和标准的一致性：促成结构良好的代码库
6. 为产品发展奠定坚实的基础
7. 提供一种沟通解决方案的结构：以不同的抽象级别面向不同的受众

软件架构的挑战

象牙塔里的架构师（指架构师脱离实际）

缺乏沟通

01

所有决策集中化

瓶颈

02

做出过多决策

推迟决策可能比推翻决策更好

03

前期大型设计

过多不必要的图表和文档

架构设计过程造成的延误

04

03

软件架构的基本策略

敏捷软件开发的软件架构

01

能够对环境做出反应的架构

适应不断变化的需求

也称为演化架构

良好的架构能够促进敏捷性

更好地理解权衡和决策

02

常见的错误（反模式）：

采用敏捷软件开发技术，却创建了非敏捷的软件架构

这是由于过度关注交付功能造成的

软件架构定律

定律1：在软件架构中一切都是取舍；

推论1：如果一个架构师发现了一个设计不是取舍（单纯的好或者单纯的不好），那很有可能他还没有识别出来需要做的取舍

推论2：一切有意义的决定必然都有不利的一方面（损失性能，增加工作量，……）

01

定律2：为什么做 比 怎么做 更重要

质疑一切

记录架构决策

02

Fundamentals of Software Architecture, M. Richards, N. Ford

软件架构的输入

- 设计目标
- 功能需求
- 质量需求
- 约束
- 关注点

架构把握平衡：创新性 VS 应用成熟方法

01

创新

有趣

有风险

能够提出新的解决方案

可能是不必要的

例子：LLM，multi agent

02

应用成熟方法

在熟悉的领域中是有效的

可预见的结果

有时候不是最佳方案

已经被证明是高质量的技术

例子：规则引擎

根据系统类型选择架构策略

创新领域中的绿地系统（新系统）

智能对话机器人，人形机器人控制，……

知道的人比较少，创新性比较强

成熟领域中的绿地系统

传统的企业应用，标准的手机apps

大众熟悉的成熟领域（考勤，资源管理，信息管理）

棕地系统（历史系统）

对历史系统进行修改

架构的成长和变化

- **原则在不断发展变化：**

  云原生系统的可靠性支持

  大数据的实时分析

  前端的复杂化

- **架构师需要持续关注：**

  新的开发技术（go）

  新的架构模式（rag，agent）

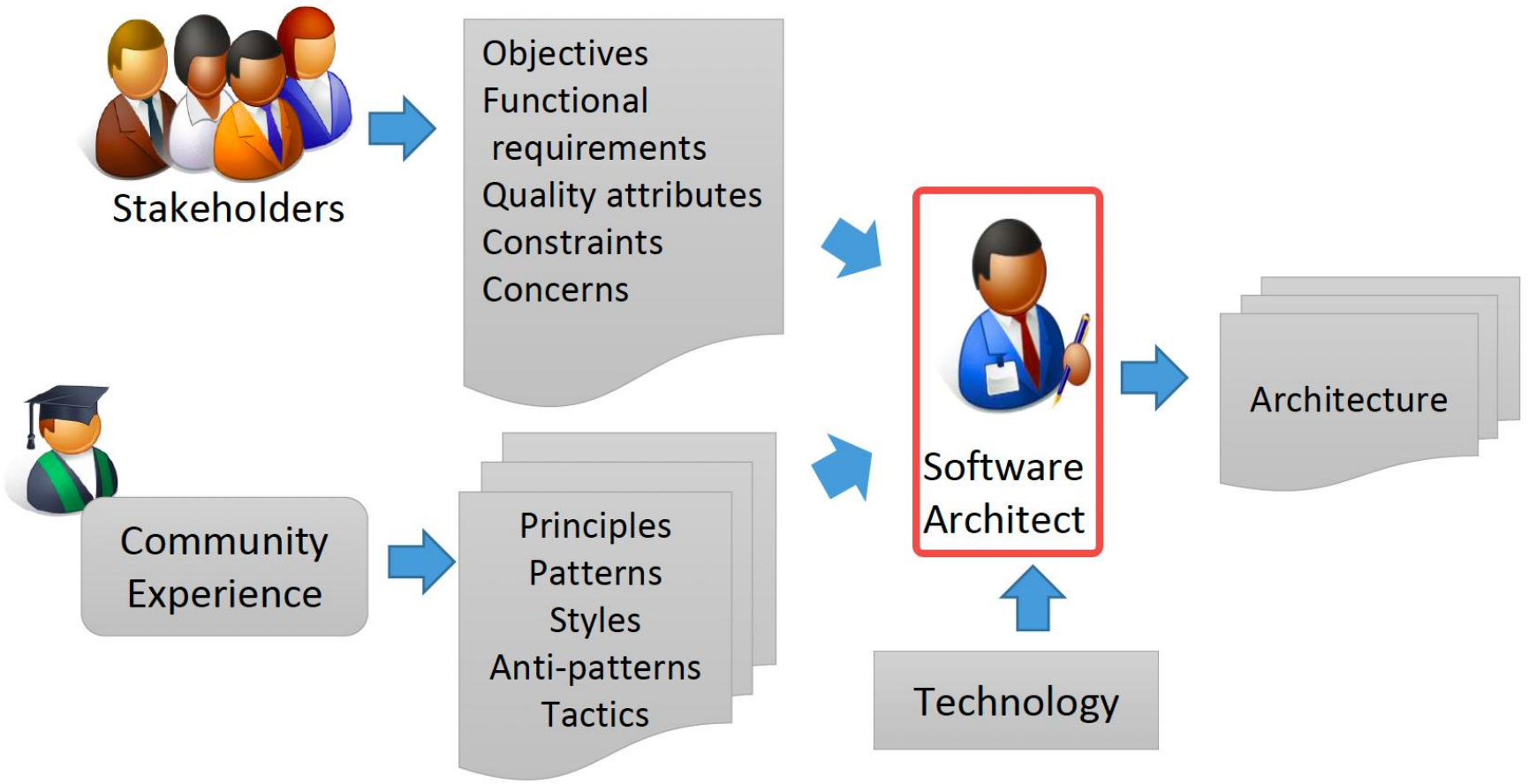
  经验是最好的工具（没有银弹）

  - 自己的经验
  - 来自社区的经验

04

软件架构的整体流程

架构是知识的汇总


Stakeholders
Objectives
Functional
requirements
Quality attributes
Constraints
Concerns
Community
Experience
Principles
Patterns
Styles
Anti-patterns
Tactics
Software
Architect
Technology
Architecture

part 2-架构师

目录

01 架构师的自我修养

02 架构师和研发团队

03 架构师和项目干系人

01

架构师的自我修养

架构师是一个角色，不是一个级别（理论上）

- 对架构师的期望
  - 做出架构决策
  - 持续分析架构
  - 紧跟现有趋势
  - 确保实现符合现有决策
  - 广泛的见识和经验
  - 具备业务领域知识
  - 具备人际交往能力
  - 理解并驾驭职场政治

作出架构决策

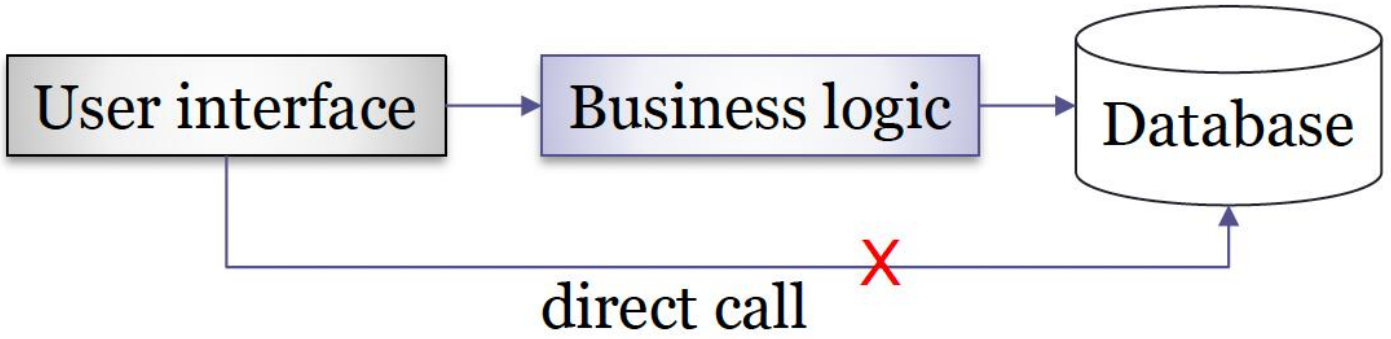
- 定义架构决策和设计原则
- 指导技术决策
- 保留决策记录
- 分析利弊

持续分析架构

- 持续分析架构和技术
- 确保项目的技术成功
- 意识到系统结构性衰退
- 努力保持新代码和旧代码的一致性
  - 将代码按照规范组织成包、文件夹、模块等
  - 定义模块边界、指南、原则，避免错乱
  - 管控测试环境和发布环境

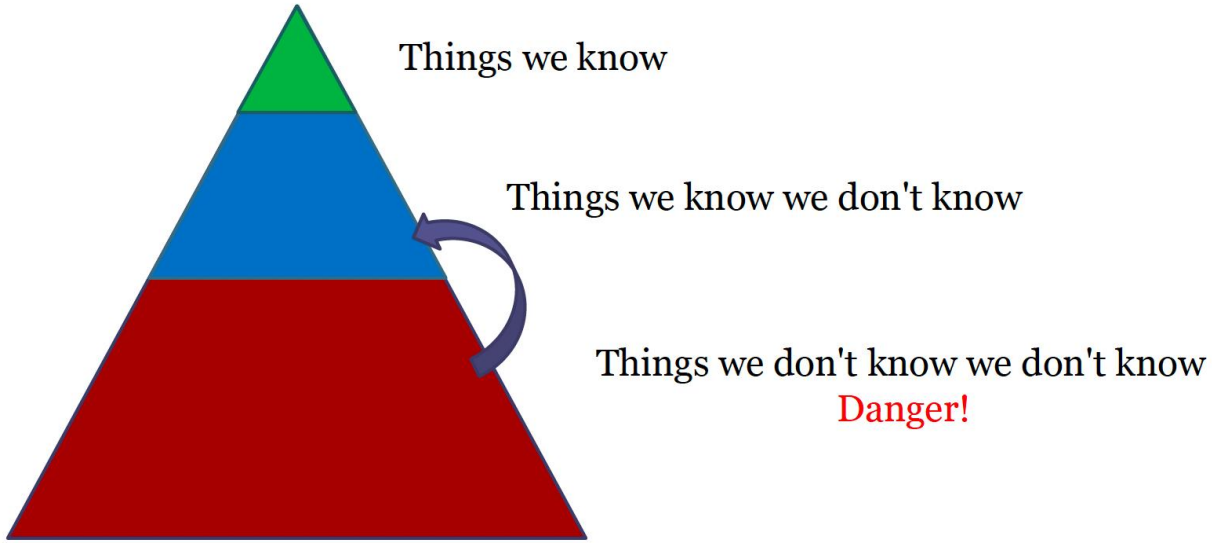
确保符合现有决策

- 架构师通常会施加一些约束
- 例子：
  - 从用户界面访问数据库的约束
  - 开发人员可能会绕过它



紧跟现有趋势

- 了解最新的技术和行业趋势
  - 也要考虑产品本身的需求
  - 架构师做出的决策 = 持久且代价高昂
  - 优秀的架构师知道他们知道什么，以及他们不知道什么



接触多种技术，积累经验

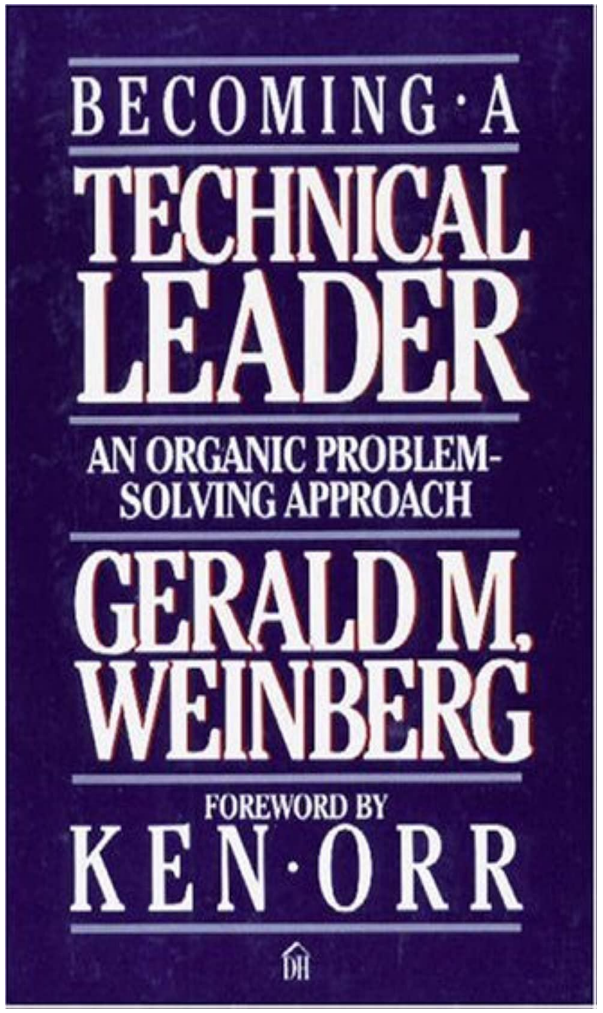
- 接触多种多样的技术、框架、平台、环境等。
- 这并不意味着要成为每个领域的专家
- …但至少要熟悉各种不同的技术
- 技术广度胜于技术深度

业务领域知识

- 期望架构师具备一定程度的业务领域知识
- 理解业务问题、目标和需求
- 使用领域语言与高管和业务用户有效沟通

具备人际交往能力

- 软件架构师 = 领导者
- 团队合作和领导能力
- 技术领导力（不是职位领导力）
  - 具有包容性并进行协作
    - 帮助开发人员理解全局
  - Get hands-on: 亲力亲为
    - 参与交付过程
    - 对底层有理解
    - 编码是工作的一部分
    - 代码审查和指导



G. Weinberg：no matter what they tell you, it's always a people problem

理解并驾驭组织政治

- 理解企业的政治环境，并能够驾驭其中的政治因素
  - 架构决策会影响利益相关者
    - 产品负责人、项目经理、业务干系人、开发人员……
  - 架构师做出的几乎每个决定都会受到质疑
- 需要谈判技巧
  - 展示并捍卫架构
- 软件架构师的电梯演讲
  - 与不同层级沟通

02

架构师与研发团队

软件架构师的主要关注点

- 明确质量属性
  - 明确某个问题的解决方案
- 确定权衡和决策
  - 说明为什么要这样做
- 抑制熵增
  - 为团队定义标准、约定和工具集

架构师如何和团队协作

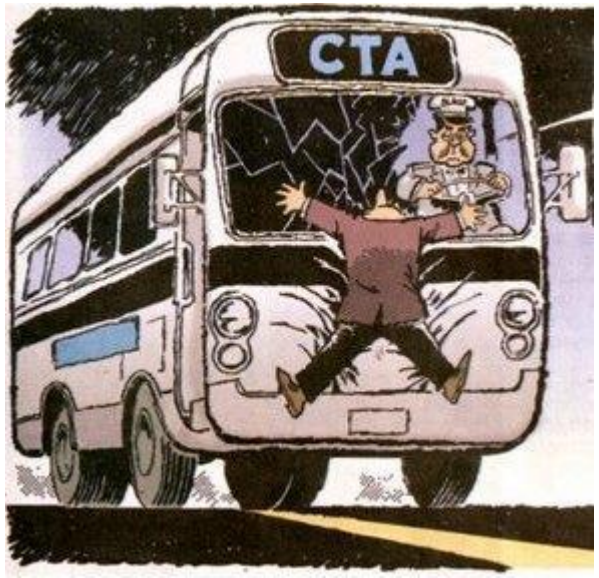
- 软件工程是一项需要团队协作才能完成的工作。
- 涉及：
  1. 社交互动
  2. 架构师的个性
  3. 团队拓扑结构
  4. 团队规模

1 社交互动

- 缺乏安全感
  - 人们害怕别人评判他们正在进行的工作
  - 试图隐藏代码
- 天才神话：
  - 倾向于将团队的成功归功于个人
  - 例如：比尔·盖茨、林纳斯·托瓦兹等。
- 隐藏被认为是有害的
  - 独自工作会增加风险

团队的bus系数

- 指项目中，如果有多少人被“巴士撞倒”（即突然无法工作，例如因意外、疾病、离职等），会导致项目完全停滞。
  - 换句话说，它衡量的是项目对关键人员的依赖程度。
  - 不可预测的生活事件可能发生。
- 团队合作是降低风险的必要条件。
  - 确保至少有 2 个人掌握关键知识。
  - 良好的文档记录也很重要。

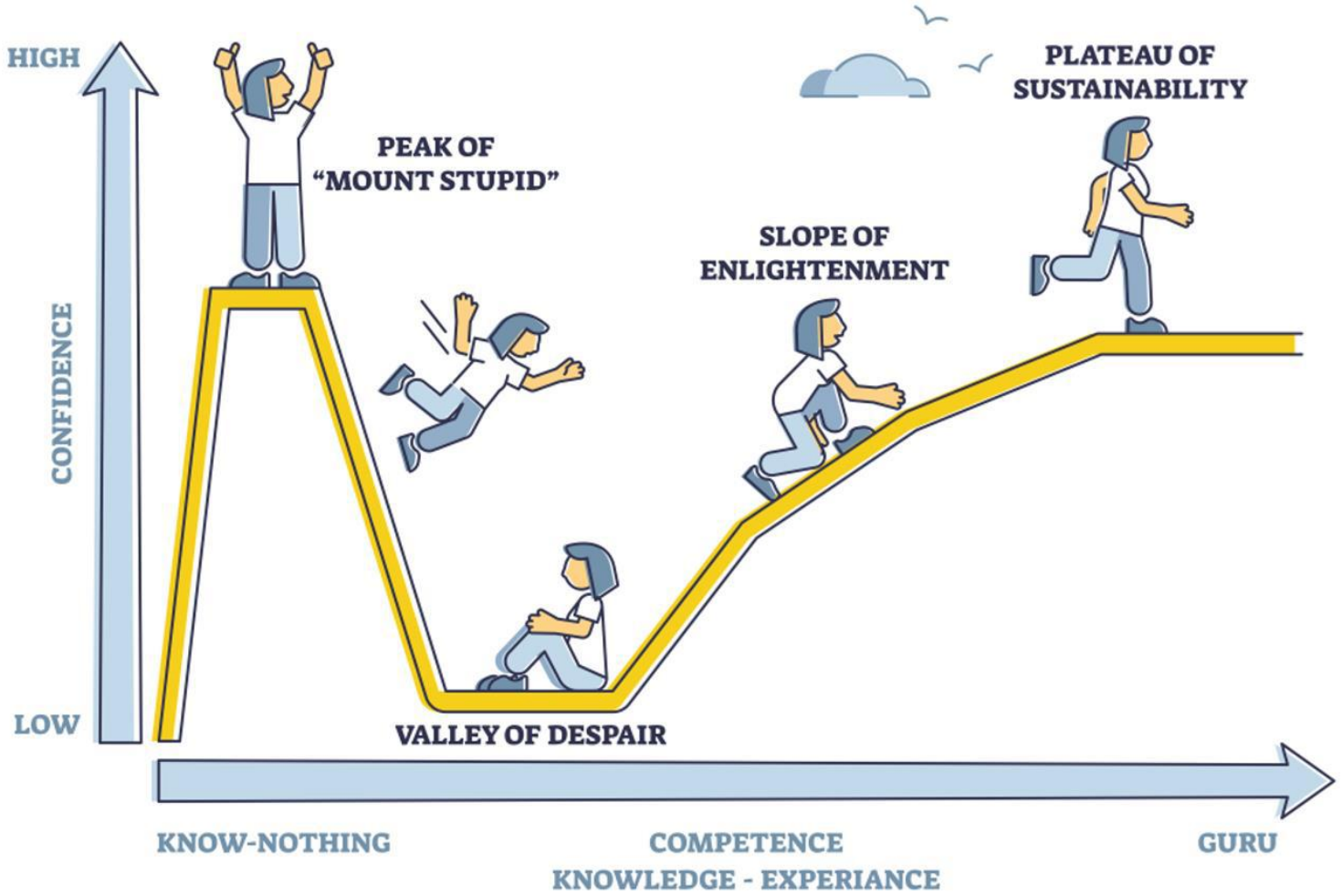


社交活动的3个支柱

- 谦虚
  - 你不是宇宙的中心（你的代码也不是！）
  - 你既非全知全能，也非绝不会犯错。
  - 你应该保持开放心态，不断自我提升
- 尊重
  - 你相信他人是能胜任的。
  - 你相信他人会做出正确的选择。
  - 在适当的时候，你乐于让他们主导。
- 信任
  - 你真心关心与你共事的其他人。
  - 你友善地对待他们。
  - 你欣赏他们的能力和成就。

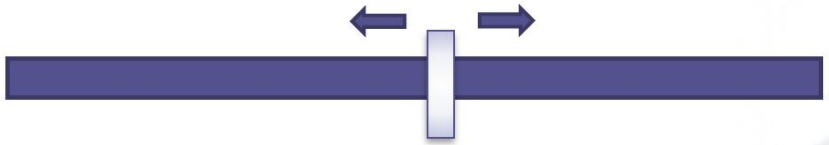
DK效应

- 人在知识有限的情况下容易高估自己的能力
- 造成后果：
  - 糟糕的决策
- 自信力 ≠ 能力
  - good：没试过但可以尝试
  - bad：错了但是坚持



2 架构师的个性

- 高效的架构师 = 在控制狂和袖手旁观者之间权衡



控制狂

1. 参与所有决策
2. 决策过于细致和底层
3. 参与代码开发（造成瓶颈）

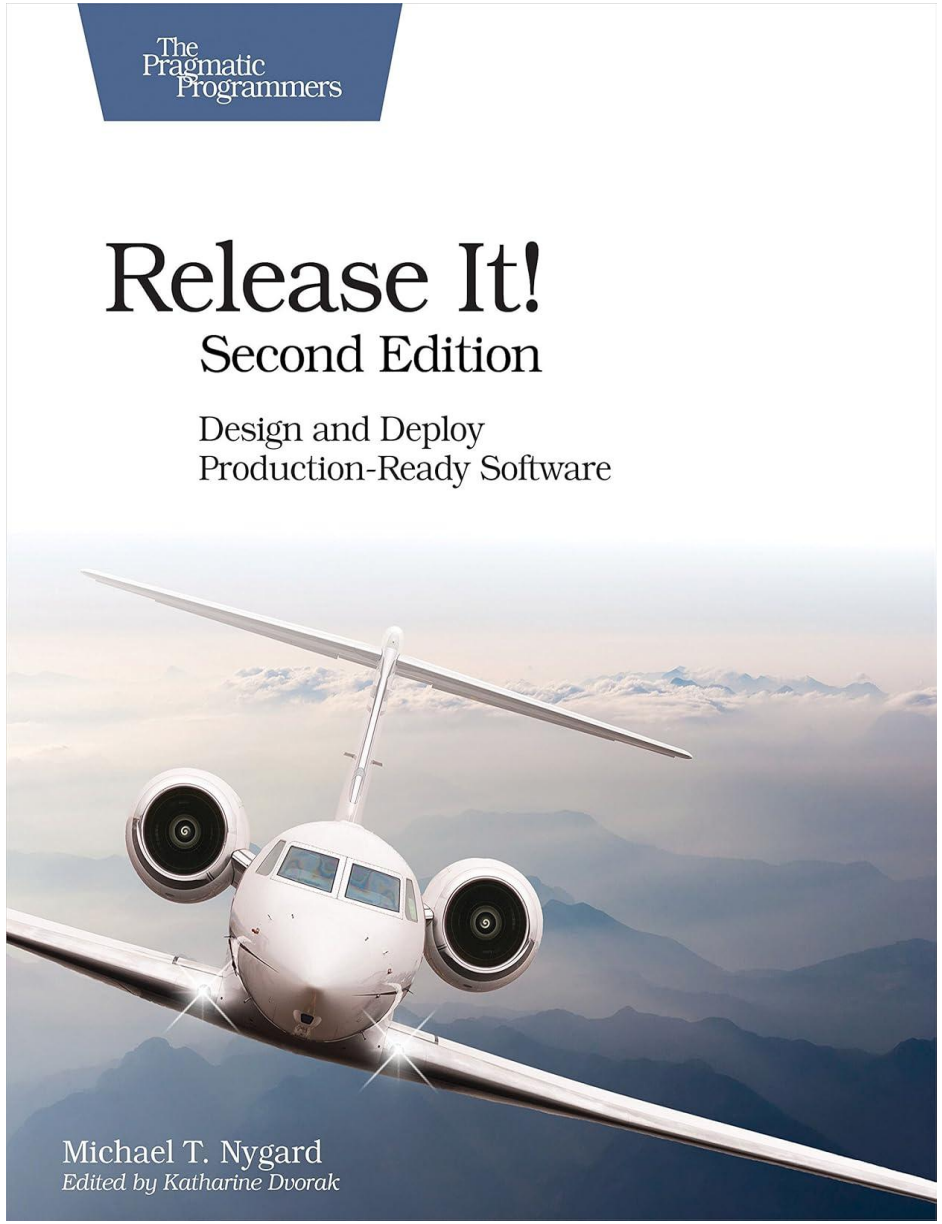
袖手旁观的架构师

1. 与开发团队脱节
2. 从不常驻（频繁更换项目）
3. 只参与最初的图表设计

3 团队拓扑

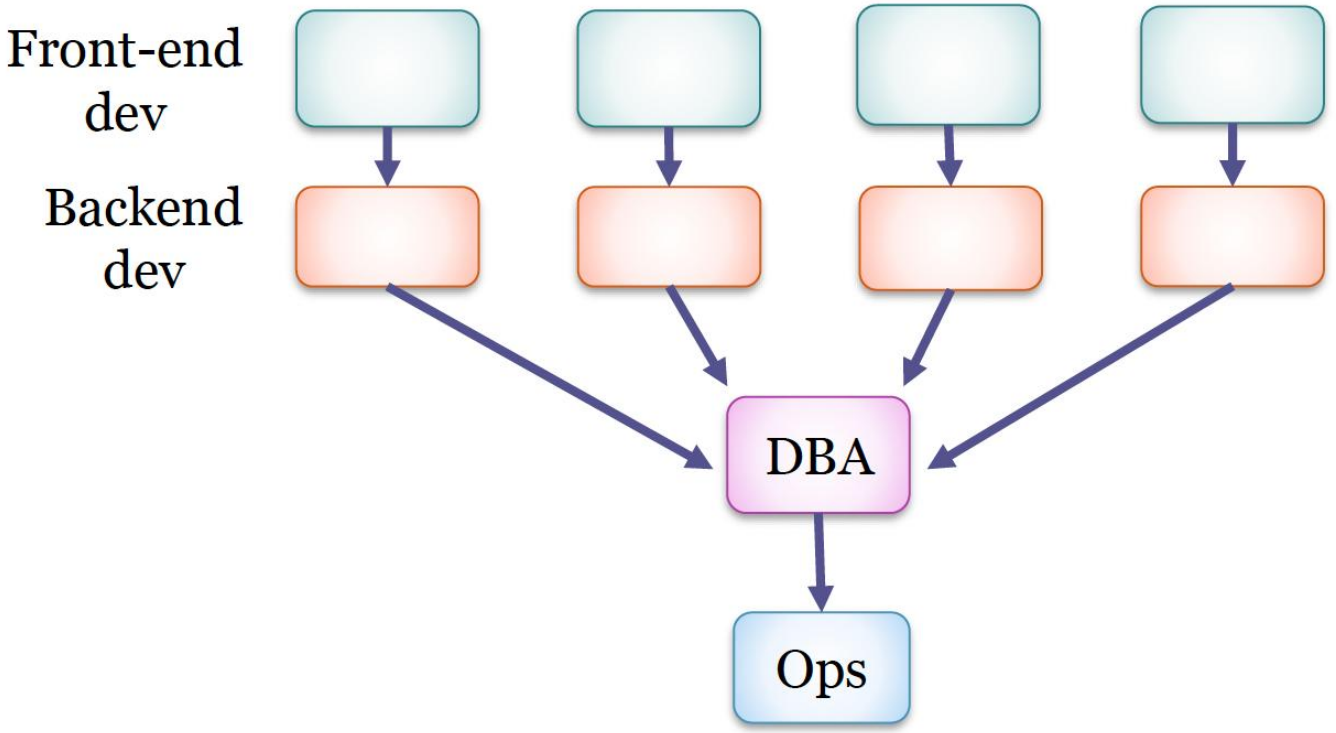
- 团队拓扑会影响系统
  - 沟通结构
  - 团队动态
  - 团队规模

"Team assignments are the first draft of the architecture", M. Nygaard



传统团队的组织方式

- 每个新项目都需要现有的团队
- 例子：4个团队：前端、后端、数据库管理员，运维（Dev Ops）

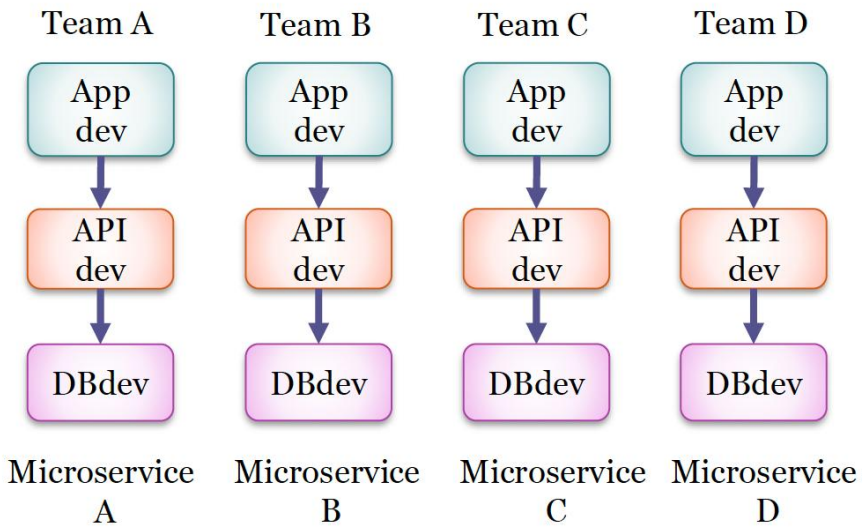


康威定律

- 设计系统的组织…会受到约束，最终产生的设计会复制这些组织的沟通结构。[M.康威，1967]
- 推论：
  - 一个系统的最佳结构受到组织社会结构的影响。
- 例子：
  - 如果有3个团队（设计、编程、数据库），系统自然会有3个模块。

逆向康威定律

- 对团队的组织结构进行演进，以促进期望的架构：
  - 先分解模块，再创建团队
  - 分配给团队的任务要符合团队的风格
  - 微服务的例子



Amazon's principle: You build it, you run it

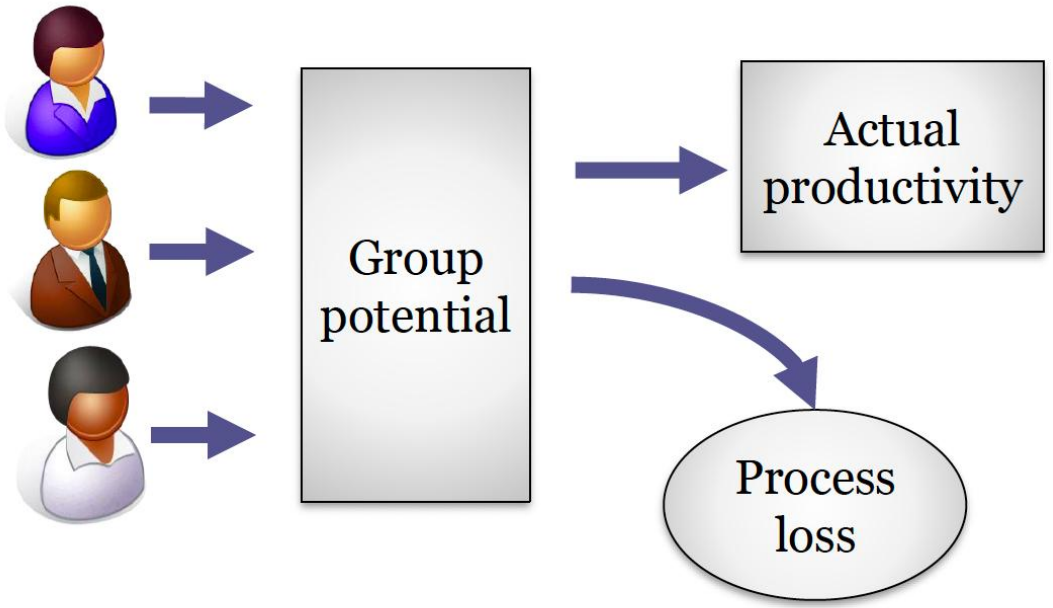
4 团队规模

- 高效的团队规模会影响项目成功。
- 需要注意的一些警告：
  - 过程损失：效率损失
  - 多元无知：保持沉默
  - 责任分散：

2-pizza rule: "if you can't feed a team with two pizzas, it's too large", J. Bezos

过程损失

- 实际生产力通常低于群体潜力
- 一些原因包括：沟通成本、会议等等



Brooks's law. Adding manpower to a late software project makes it later

多元无知（从众心理）

- 团队成员私下里拒绝它，但是表面上表示同意，因为他们认为其他人都同意了，可能是自己遗漏了某些明显的事情。
  - 一些架构决策没有受到质疑（或挑战）。
  - 事实上架构师可能没有考虑全面。

职责分散

- 团队规模扩大会对沟通产生负面影响。
- 一些迹象包括：
  - 边界不清晰：对谁负责什么感到困惑
  - 工作缺乏成就感
  - 事情被遗漏

03

架构师与项目干系人

项目干系人

- 所有参与开发或受系统影响的各方
- 可以是个人、角色或组织
- 通常有不同的关注点
  - 有时是矛盾的（方便 vs 安全）
- 有必要：
  - 理解关注点的性质、来源和优先级
  - 识别并积极地与他们互动
  - 征求他们的需求和期望

干系人（显式或隐式地）驱动架构的整体形态和方向，以满足他们的需求。

识别项目干系人

- 所有以下类型的个人、角色、组织：
  - 有责任了解架构的人（横向团队）
  - 能够对架构发表意见的人（评委）
  - 需要使用架构或代码工作的人（研发）
  - 在工作中需要阅读架构文档的人（QA）
  - 需要对系统或其开发做出决策的（leader，投资人）

项目干系人

01

Internal

Analyst

Designer

Business manager

Developer

Product owner

Auditor

UX designer

Project manager

02

External

Customer

End users

Auditor

Public authority

Suppliers

External service providers

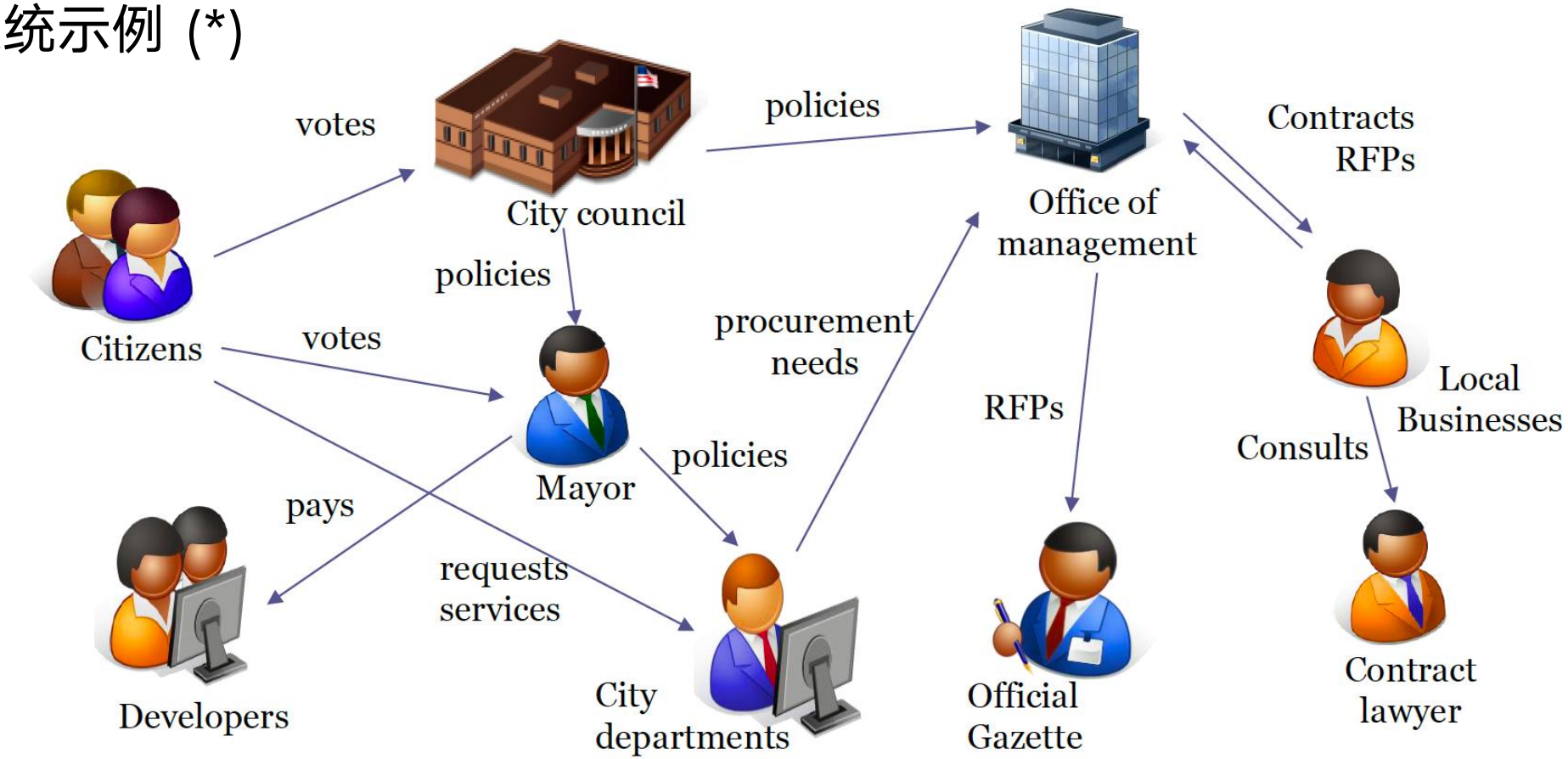
了解项目干系人的期望

- 识别具体需求
  - 目标：实现目标受众更大的满意度
- 避免不必要的工作
- 避免记录无关紧要的事情

角色-姓名	联系方式	对产品的期望（架构相关）

项目干系人地图

- 展示参与系统或受系统影响的人员/角色
  - 包含关系/互动
- 采购自动化系统示例 (*)
  - 收集需求
  - 寻找供应商
  - 下单
  - 合同
  - 收货
  - 发票
  - 报表分析



业务目标

- 以人为本的业务目标
- 通常3-5个
- 结构：
  - 主体/干系人
  - 结果：以可衡量的方式表达需求
    - 如果系统成功，世界会发生什么变化？
  - 上下文
    - 关于目标的一些分析

主体	输出	上下文
市长	减少开支30%	但是不要降低政府部门的服务质量
采购管理部门	分析过去10年的数据	分析历史数据才能预测将来的需求

感谢观看

THANK YOU