怎么做架构

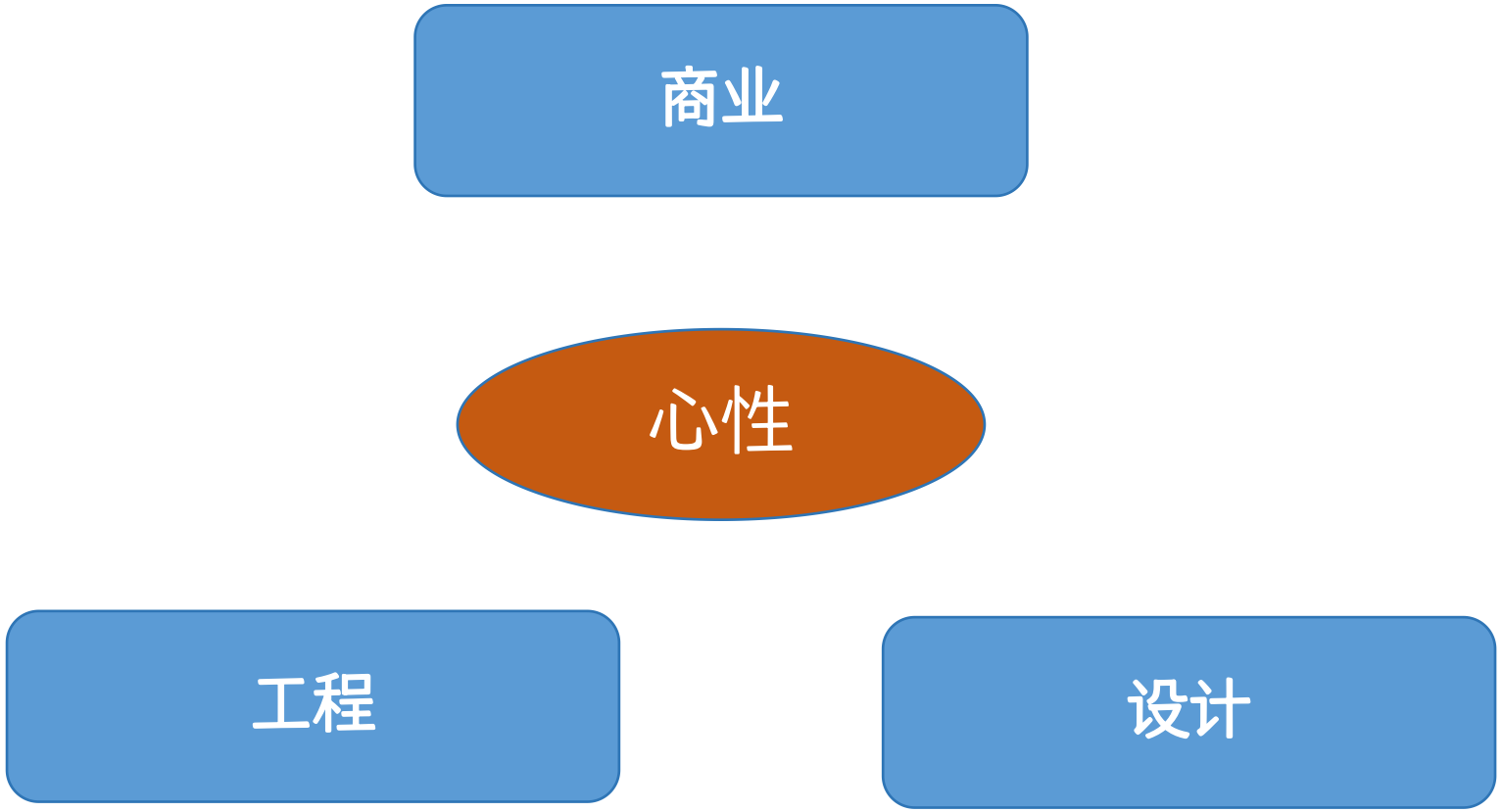
1024实训营3期结业

@xushiwei

1024实训营

- 缩短从校园到企业的距离
- 培养优秀的工程师
- 为什么实训周期是 3 个月？

人才的能力象限



心性

- 韧性：是否能够长期坚持为一个目标去努力
- 要性：对成功的渴望
- 空杯：倾听、迭代认知的能力

工程

- 工程是一门有关于如何“把事做成”的学问
- 工程可以和所有学科交叉（工程+X）

- 模块：将复杂问题拆解成独立子问题
- 版本：迭代逼近目标（节奏）
- 分工：团队协同

设计

- 设计不是指“产品包装”，而是一门有关于“决策”的学问
  - 要做什么事，以及把事做成什么样子

- 组织需要设计
- 产品需要设计
- 架构需要设计

产品与架构

- 初级产品经理和初级架构师都喜欢**做加法**

- 但是**顶级产品经理**做的是**减法**
- 而**顶级架构师**做的是**乘法**

产品设计：少做加法，做减法

- **少就是指数级的多**
- 卖点越多的产品其实就是没有卖点
- 架构的**开闭原则**对产品同样适用
  - 产品要满足的需求可以是无限的，但**产品功能必须是封闭（有边界）**的
    - 例：Computer

什么是架构设计?

- 架构从手法上就是模块拆解
- 连接与组合
- 实体 (Entity) 的边界问题

架构设计：少做加法，做乘法

- 模块的**规格高于实现**
  - 时序图是基于规格串起来的用户故事，用伪代码比UML图成本更低

- 假设你要做一个网站，把页面1交给A，页面2交给B，然后开干，这是典型的一次做加法的过程

- 把整个网站需求分解为 A、B、C、D、E 等完全正交无耦合、甚至和你的网站原始需求不直接挂钩的**通用模块**，用极短的桥接代码1把这些通用模块组装起来完成页面1，桥接代码2完成页面2，这才是做乘法
  - 而判断乘法做的好不好的标准非常简单：**桥接代码的代码量越少**，乘法的威力越大，你的**架构设计的能力也就越强**

何为乘法?

- 每个模块都是独立子业务

- 它独立看有更大的需求边界
- 它独立看很有前景（很多人都有类似的需要）

从 Go+ 的实现看架构拆解

- Go+ 源代码 => Go+ AST
  - gop/scanner + gop/parser + gop/ast（为什么是 3 个？）
  - gop/tpl/scanner + gop/tpl + gop/ast（另一种可能的实现路径）
- Go+ AST => **Go AST => Go 源代码** => 可执行程序
  - gop/cl + gogen + go compiler (go, llgo or tinygo)
  - 到 Go 源代码不是必须的，但是是当前最合适的（为什么这么说？）
- **Go 源代码 => Go AST** => Go SSA
  - go/scanner+parser + tools/go/packages + tools/go/ssa（都不是我们的）
- Go SSA => LLVM SSA => 可执行程序
  - llgo/cl + llgo/ssa + goplus/llvm + clang（只有非常局部的一块是我们的）

从 Go+ 的实现看架构拆解

- 一个逆直觉的重要结论：

- 模块切分，通常和需求并不形成对应关系
  - 如果对应，往往反而说明模块划分和团队分工是有问题的

- 需求分析要做什么？
  - 需求未来可能的发展方向预判（防止过度设计）
  - 洞察需求的内在逻辑关联（模块切分的基础）