软件架构之禅 - The Zen of Software Architecture

小鹿豆包

Published with GitBook

目錄

软件架构之禅 Zen of Software Architecture.

本文通过生动易懂的实例，对软件架构做出分析。

如果你是一位正在修Software Architecture的留学生，那么你会发现，课程中许多晦涩难懂的概念，你都可以在这里找到对应的中文解释.

第一章 软件架构的总体认识：万事开头难！-- An Overview of Software Architecture.

掉书袋的说，软件架构，是conceptual fabric that defines a system。

通俗的来讲呢，比如说你的公司想要开展一项新的业务，你的老板肯定不能拍桌子拍脑袋想怎么做就怎么做吧？

他得跟策划经理说：“来，现在咱们要开展女仆咖啡业务，你看看给我一个策划吧！”。

项目经理听了话就得干啊，女仆咖啡啊，好，得有几个部门，分别做啥，怎么分工，怎么协作，要啥资源等等。

总之啊，策划经理得给大老板一个方案，利用手头的资源，怎么才能把女仆咖啡做到最好最强。要不说策划经理不好当呢，凭空YY的能力差一点儿都不行！

下面我们来看看，策划经理得怎么写这个策划案！

万事开头难，策划案要写啥呢？是不是事无巨细的把能想到的细节都写进去就行啦？比如，女仆咖啡项目嘛，咱公司得招聘一百个年轻貌美的小姑娘，这些小姑娘哪个的胸围得D以上，哪个得长得像刘亦菲，把这全写进去？万一老板想尝个鲜，想要贫乳眼镜娘呢？你怎么办？

重写策划案？当然不行啦！哦，你老板让你写个策划，你交一个都是小姑娘资料的文档，你老板觉得你是什么人啊！老板想看啥？

老板想看啥？！老板想看张三负责找小姑娘，李四负责找酒店，王五负责发广告，赵大麻子负责那啥有关部门呀！张三一定确定了要他找小姑娘，你要冷不丁的不让他找了，你想他能乐意吗？

所以说啊，策划案，就是要写一旦系统建立起来，就难以更改的部分。这就是：Architecture focuses on thoses aspects of a system that can not be changed easily once the system is established.

软件架构的总体认识：项目经理老大难 Why is software architecture so complicated?

老板一发话，项目经理就得干啊！项目经理头发都掉光了，这是为啥呢？我们总结有三点。

- 第一点：Young field。项目经理没干过女仆咖啡啊，说实话，他连去都没去过，咋写策划案呢？
- 第二点：High user expected。客户有客户的要求，我来你这儿，首先，我的口味你得知道，我的时间你得安排好，我非五星级咖啡不喝，再有就是，我来这儿还不能让我老婆知道。项目经理头大啊！
- 第三点：Software cannot executed independently。说白了，咱搞个女仆咖啡业务，总得租人家的商铺吧？你试试跟老板说：老板啊，租商铺太麻烦了，要不你盖一个大厦得了。看老板不抽你。

那怎么办呢？咱们接着往下看！

软件架构的四大主要问题 -- 4 Main Issues Of Software Architecture

问题都得一点儿一点儿解决，我们有个比较机智的项目经理S，S说过一句名言：“所有的问题，都可以分成两种，好解决的，不好解决的。” S的这句话说得真是太对了啊！咱就把问题分成两大类：

Incidental difficulties 和 **essential difficulties**。 顾名思义啦。

那不好解决的问题都有啥呢？S又说了：“我把不好解决的问题分四类，第一类，**complexity**；第二类，**comformity**；第三类，**changeability**；第四类，**intangibility**。”

看不懂吧？唉，我看S是zhuangbility。我来解释解释吧。

Complexity

- 第一类，复杂性。这个很好理解，咱业务做得越大，涉及的机构、人、事务就越多，当然就越复杂啦。同样的，软件体积越大，复杂性也嗖嗖的飚上去啦！

Comformity

- 第二类，适应性。啥叫适应性呢？一句话：“橘生淮南则为橘，生于淮北则为枳”。见人说人话，见鬼说鬼话。咱公司不得依赖有观不门生存吗？软件在操作系统上跑，也是这个道理。

Changeability

- 第三类，易变性。要在东北做女仆咖啡，那你得了解东北人的习性，譬如胸大，要去上海开女仆咖啡，那你得按上海人的口味走，譬如腰细。你总不能说，我们公司不提供腰细的姑娘，要不你们上海人改改口味，喜欢胸大的得了？这也就是说，每当有啥变更，人们总会想，别的不好改，就软件先改改吧！

Intangibility

- 第四类，无形性。这个也不难理解，比如回家了，你老婆问你：老公，你们这次项目是怎么做的呀？你要是个建筑师，那很简单啦，拿一个房屋模型，你老婆一看就明白了。做策划不一样，你很难跟你老婆形容明白，也很难用物理的模型展现给她（他~）。

其实说白了，就是复杂！复杂！再复杂！我们要解决的首要问题，就是软件架构它太复杂啦！

五大解决措施 -- 5 Main Solutions Of Software Architecture

咋整？请出S经理吧："复杂性，森破！美国的华莱士，我跟他们谈笑风生！五条！你们照着做就行啦，拿衣服！"哪五条呢？

---

1. High level language

比方说你是老板，要搞个项目，现在有俩人：甲，有黄金履历，以前做过好多次女仆咖啡，有口碑保证，手里的资源多；乙，初出茅庐实习生一个。你会让谁做呢？软件也是这个道理，高级语言的封装性好，手册多又易读，当然选高级语言啦。

---

2. Development tools & environment

人和动物的区别是啥？使用工具啊！啥都不想说了，你感受感受吧！

---

3. Component based reuse

项目为啥复杂呢，人多啊！要是你发现，诶？谈五星酒店和谈四星酒店，这步骤都差不多啊，不如都让李四去谈好啦！

---

4. Develope startegies

诸葛亮为啥这么吊？司马懿为啥这么吊？张良范蠡为啥这么吊？锦囊妙计在手，还怕打不赢吗？

---

5. Emphasis on Design

俗话说的好，只要功夫深，铁杵磨成针。俗话又说的好，三思而后行。懂了吗？

好啦，跟我念口诀吧：HDCED！HDCDE！

上文说到，**Empahsis on Design**，怎么个Emphasis法呢？

S经理还有话："我把Design分四步，FPDP！"

"……说人话！"

"Feasibility stage, priliminary stage, detailed disign stage, planning stage."

"此话怎讲？"

"F：众人拾柴火焰高，先找到所有可行的主意再说。**P**：找最好的。**D**：把这个主意的细节补充上去。**P**：看看能不能满足需求，再做做分工。**"**

两点重要补充：豆经理有话说 -- 2 important tips for the complexity of Software Architecture

S经理的话说完了，豆经理补充两点。豆经理是个实在的小姑娘，她说：

---

1. Level of discourse

这是第一点。Treat application as a whole, break big problems into subproblems, think above the disired level。

---

2. Separation of Concern

要做到这一点，你需要做两件事。

- 第一件：decompose problem into independent part。
- 第二件: in architecture, separating components and connectors。

只有这样，才能做到关注点的分离。在关注一个部分的时候，不会受到其它问题的干扰。

但是要做到这两件事并不容易，主要干扰有俩： **scattering** 和 **tangling**。

---

scattering

scattering是啥呢？如果有这么一样东西，广泛分布在每个component里面，在看这个的时候，很难做到分离。比如log，日志这玩意儿，谁都得写。

---

tangling。

再者说tangling，如果我们关注的一样东西，同很多components都有交互，那么是不是也很难做到关注点分离呢？比如我们要关注性能问题，这就不可能是一个两个组件决定的。

软件架构分解三大元素 -- Components, Connectors & Configurations.

软件架构的组成有三大元素：Components, Connectors 以及 Configurations.

在这里，为了严谨起见，我们首先来看一下这三大元素的定义。

Components

Architectural entity that

1. encapsulate a subset of functionalities,
2. restrict access via explicit interface,
3. has explicit environmental dependencies.

Connector

Architecural entity tasked with effecting and regulating **interactions** between components.

Configuration

A set of specific **associations** between components and connectors.

简单来说:

component就是一个个工作小队，每个小队都负责项目的一些工作.

connector是小队之间沟通设备，负责有效沟通和调整沟通。

configuration则表示，我要怎么安排这些小队和沟通设备，谁和谁沟通，咋沟通？

以上概念来自滑铁卢大学 *Concepts are from the University of Waterloo*

软件架构的重要名词解释 -- Important Terms In Software Architecture

这一章呢，我们对后文中一些十分重要的名词、概念作出解释。

Topology goal

Maximize cohesiveness within each component. Minimize coupling between components.

Abstraction

Concept not associated with an instance. focus on key issues and eliminate unncessary details. control abstraction, data abstraction.

Decomposition

Top-down abstraction. break the problems into independent parts. describe each component. Criteria: implementing teams, application domain, parallization.

Architecture representation

- Why: software architecture is fundamentally about facilitating technical communications between different stack-holders.
- Property: ambiguity, accurency, precision.

Prescriptive

- How the system will be built.

Descriptive

- How the system is actually built.

Conceptual

- Meaningful relationships

Concrete

- Actual relations.

为了避免上面两对概念的混淆，我们进行如下的进一步解释。

Prescriptive 和 Descriptive 更关注的是理想模型和实际模型之间的差距。

Conceptual 和 Concrete 关注的是，理想模型是一个细节不完整的模型，主要就是规范了重要的关系，而实际模型是这个东西的具体实现。

Architectural Degration:

- **Drift**: introduction to changes that are not captured in the current architecture but don't violate it.
- **Erosion**: violate.

软件架构的四个重要视角 -- 4 Important Architecture Views.

在软件架构的领域，我们有许多重要的视角。

为何有不同视角

- 关注的东西不同

  Different views focus on different subset of components and relationships.

- 关注的角度不同

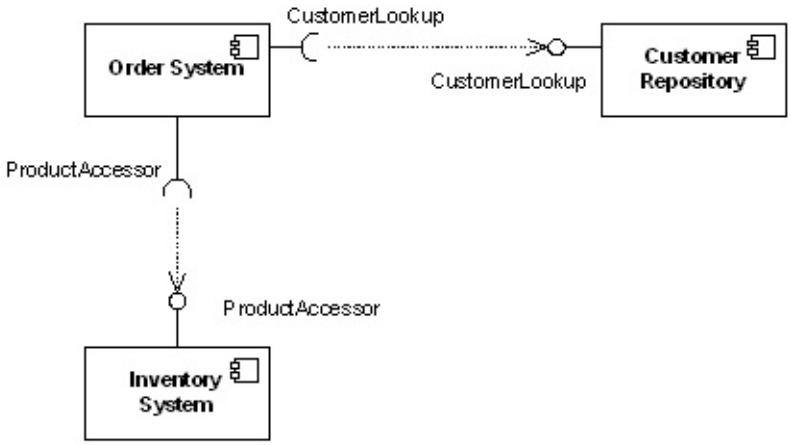
  A view usually focus on a specific concern or a specific scenario.

接下来，我们就来介绍四个重要的软件架构视角 ^-^。

Component diagram

Components and relationships. require explicit APIs.

For example:

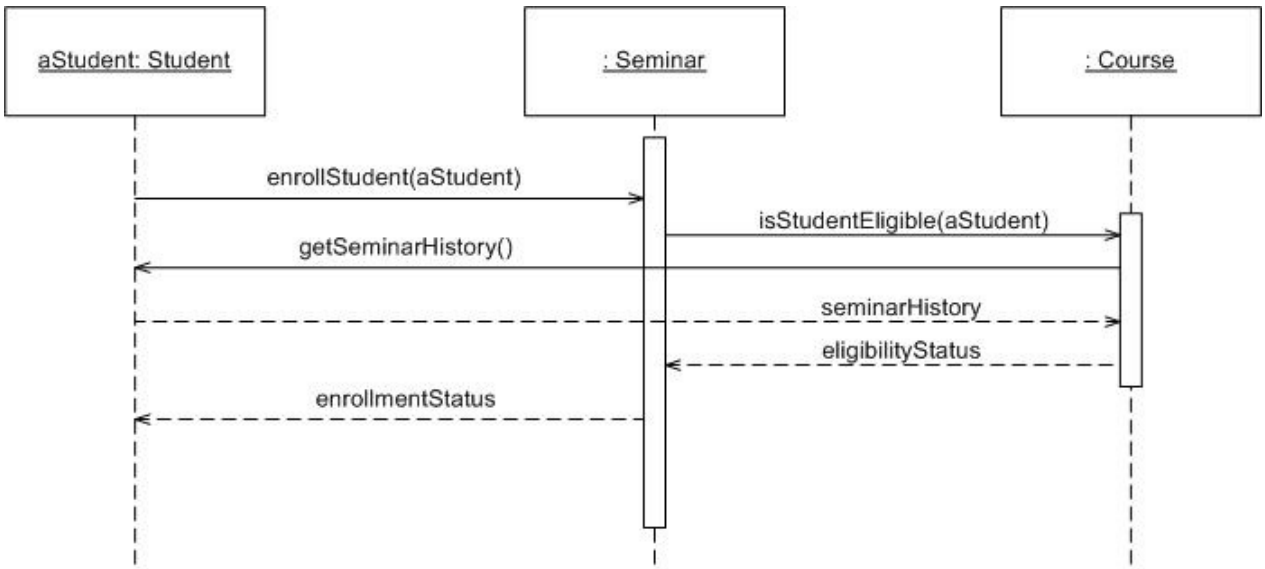


Sequence diagram

Inner-component collaborations.

Capture *behavior* of subsystem in a run-time of a specific scenario.

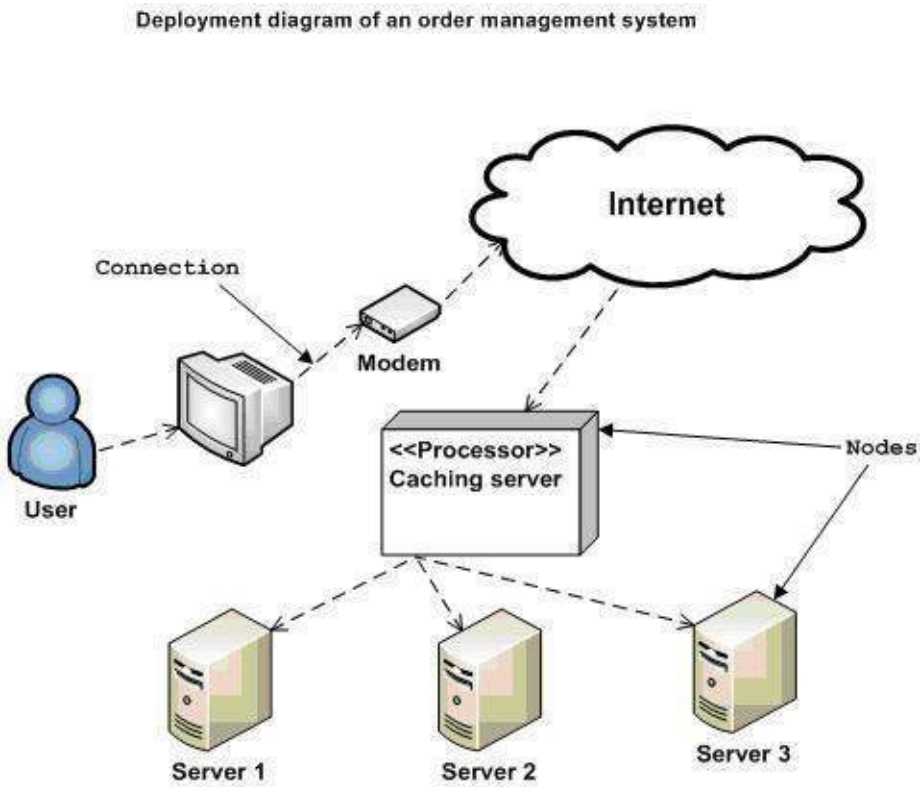
For example:



Deployment diagram

mapping of physical devices.

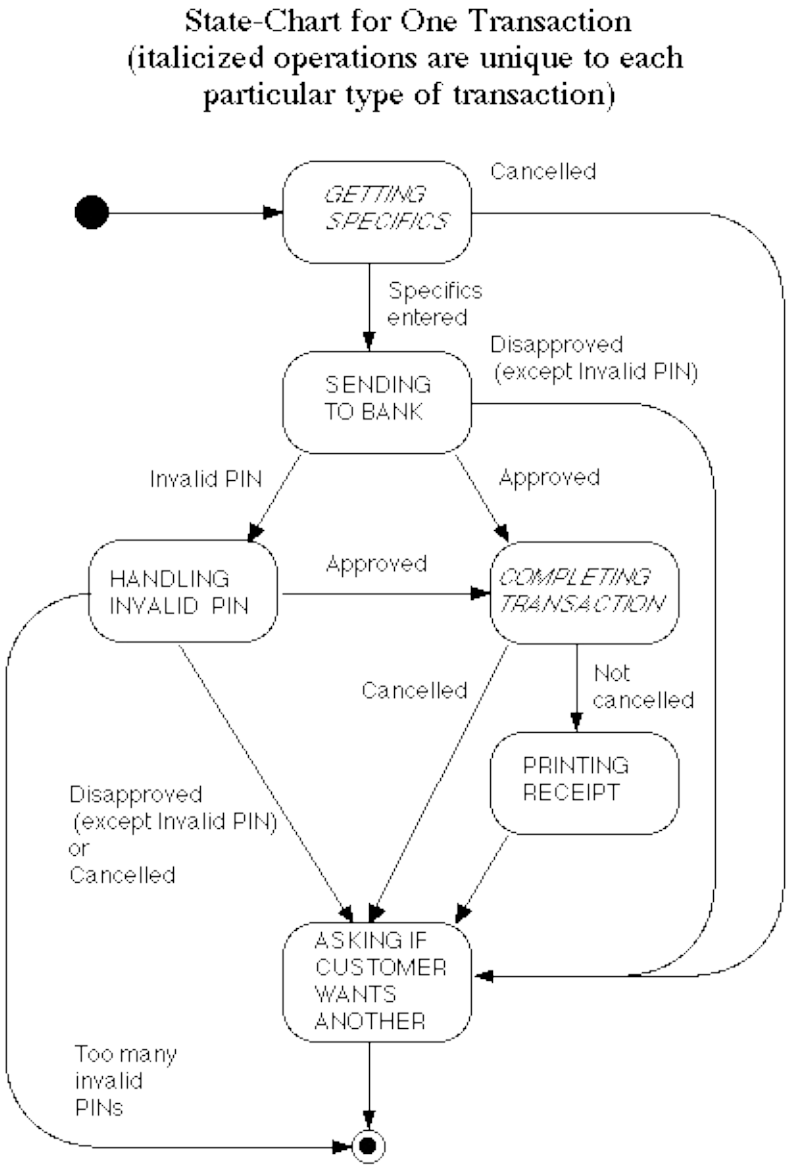
For example:



State-chat diagram

formal description of system behavior.

For example:



Images are from the internet. Concepts are from the University of Waterloo

非功能性需求：功能易做，感觉难抓！ Functional Properties & Nonfunctional Properties.

在软件架构设计过程中，对于软件的需求评估是必不可少的重要环节。

对于需求，我们只需要理解两个名词就行了：

- 功能性需求
- 非功能性需求

功能性需求

What shall the system **do**?

这个系统需要 做 什么？

非功能性需求

What shall the system **be**?

这个系统应该是 怎样 的？

也就是说，非功能性需求代表了一个软件实现其功能性的方式。这可以说是在软件架构设计中非常重要的。这一章，我们主要就来讨论非功能性需求。

NFPs are constraints on the manner in which the system implements and delivers its functionality.

非功能性需求之效率 -- Efficiency in NFP

效率其实是一个非常模糊的概念，我问甲效率是啥，甲说“干的好呗”，我又问乙，乙说“干的快呗”。

在软件架构的非功能性需求里，效率的定义是这样的：

The ability of a system to meet the performance requirement.

一个系统实现性能需求的能力！说白了就是又快又好，要啥给啥。

任何一个需求的分析，我们都要从软件架构的三大元素上去理解：Components, Connectors 和 Configurations.接下来就让我一一道来！

Components

先拿工作小队开刀。比如，女仆咖啡项目的客户接待组。这个组一次负责接待一个客户。客户接待组要有五个条件。

- **1. Small**

  第一，小！

  俗话说得好，一个和尚挑水喝，两个和尚抬水喝，三个和尚没水喝。组越小，越有利于管理。

- **2. Interaction simple and compact**

  第二，对外接口简单紧凑。

  这就是说，我们的小组同别人说我们要干啥时候，应该说“给我客户的资料，我可以同客户进行沟通，并反馈给你沟通的结果。”，而不是“給戙窓戶の濱料，戙岢苡茼窓戶進听溝嗵，還岢苡哏窓戶竊迭，婄窓戶吃飯喝酒，并仮饋給你溝嗵の結課。”瞧，是不是烦死了？

- **3. Multiple interfaces for one functionality**

  第三，给同一个功能提供多种接口。

  这就是说，每一个客户接待组，都得有俩洽谈员得，一个讲中文，一个讲英文。否则如果客户是个老外，要跟我们沟通还得特地请翻译，那岂不是很没效率？

- **4. Separate data from processing components**

  第四，数据与处理数据分离。

  “我们不生产水，我们只做大自然的搬运工。”分类组的工作，是根据一个女仆服务员的三围、颜值、性格，把她归为“热辣滚烫美眉”、“高冷气质女王”、“清纯软妹”中的一个。这个组并不会储存女仆服务员的数据，而是当有女仆服务员的数据传入时，进行类别划分。

- **5. Separate data from meta data**

  第五，数据与元数据分离。

  这也很好理解，客户问，你们公司有多少间空桌呀？难道我们还现派人挨个去数吗？当然要把已经数好的个数直接告诉客户啦！

Connectors

再说说沟通设备。要保证效率高啊，沟通设备可得谨慎选择。

- **1. Carefully select connectors. Be careful of broadcast connectors**

  首先，每个小组只负责一部分的工作，如果我使用一个大喇叭，有点啥事儿都用大喇叭广播出来，是不是无关的小组也会分心，导致效率降低呢？

- **2. Encourage asynchronous interaction**

  其次，我们鼓励异步交流（asynchronous interaction）。

  客户要看女仆服务员资料，客户接待组向资料组要资料，但是在资料组把资料找到之前，客户接待组并不需要傻等着，他仍然可以继续与客户沟通关于预约位置的问题。这样做大大提升了效率。

- **3. Be wary of location/distribution transparency**

  第三，警惕location transparency。

  这就像是，客户要看女仆服务员资料，客户接待组向资料组要资料，资料组如果不知道女仆服务员的资料在哪儿，就要给资料组打电话，资料组接到电话后，找到资料，给客户接待组送过去。这样做不仅耗时，还要担心，要是资料组的人正在午休，那可咋整？

Configurations

最后说说配置。两条。

- **1. Keep frequent collaborators “close”**

  第一条，客户接待组和资料组总是沟通，为了提高效率，客户接待组和资料组离得越近越好，这样跑腿的时间大大减小好。

- **2. Consider the efficiency impact of selected styles**

  第二，选择架构风格的时候，一定要考虑效率因素啊！

非功能性需求之复杂性 -- Complexity in NFP

复杂性并不难理解，我们还是先来看一下它在软件架构中的定义：

Property that is proportional to the size of system. volume of constituent elements, their internal structure and inter-dependencies.

接下来，我们从三大元素方面，一一解释。

Components

还是从工作小队开刀，降低复杂性，还得参考我们之前提出的，对抗复杂性的五点要求！

- 1. **Separate concerns**

     第一，关注点分离，原因上文已经说过啦！

- 1. **Isolate functionality from interactions**

     这是啥意思呢？客户要看小姑娘资料，客户接待组向资料组要资料，资料组如果不知道小姑娘的资料在哪儿，就要给资料组打电话要资料。但是接待组还需要知道电话是怎么打过去的、声音信号是怎么转化成电信号的吗？

- 1. **insulate processing from data format changes**

     比如说我们要处理的数据是小姑娘的年龄，然而不同类别的小姑娘使用的是不同的concrete class。那么我们在处理的时候，只需要使用小姑娘Interface或者abstract class来获得小姑娘的年龄就可以，并不需要特殊判断具体的类别。这样，代码的复杂度会大大降低。

- 1. **ensure cohesiveness**

     一个道理，一个团队干一件事。

Connectors

沟通设备。

- 1. **Isolate interaction from functionality**

     这就是说，一部电话只需要操心怎么把电话两端的人连起来就可以，不需要操心它们连起来之后要说什么。这样才能保证分工明确，降低复杂度。

- 1. **restrict interaction provided by each connector**

     啥意思呢？一部电话，就只能打电话，一部传真机就只能发传真，电话不能发传真，传真机也不能打电话。这样分

工明确的规定沟通设备的作用，使得沟通设备十分Focus，避免了overlap等的复杂度上升。

Configurations

配置。all about dependencies.

- 1. **eliminate unnecessary dependencies**

  去除不必要的依赖，能够增加concern separation的程度，更加降低复杂性。

- 1. **use hierarchical composition**

  使用层级的结构，因为使用层级结构，每层只跟它自己上下方的沟通。要操心的事儿少，且明确。

非功能性需求之扩展性,多向性，可移植性 -- Scalability, Heterogeneity, Portability in NFP

First of all，没有规矩不成方圆！我们先来熟悉一下概念。

---

scalability

The capability of a system when meet a larger size/scope of requirements.

Heterongeneity

The ability of a system to be executed with parts of the system.

---

portability

The ability of a system to be executed on different platforms while retaining the functional and non functional properties.

一样滴，我们从三大元素来作出分析。

Components

工作小队。

- **1.** 工作小队要小

  因为小，所以做的事情专一，当数据量增大的时候，对其影响则不大。

  这是怎么说呢？举两个栗子吧！

  - 方案一，每个工作小队由三人组成，进行接电话任务，一次做一个，做完了接下一个。
  - 方案二，一个工作小队由100个人组成，这个工作队负责所有接所有的电话。

  当打电话的人从10个增长为10000个的时候，如果采取方案一，我们只需要多建立一些工作小队即可，而若采取方案二，则需要修改的部分则大大增加。

- **2.** 接口要简单

  举个栗子，我们现在提供一个借口，有俩功能：

  - 获取客户电话
  - 获取客户喜好

  现在的客户口味变得很快，客户喜好指数级增长。我们需要修改获取客户喜好的方法，才能满足新的需求。那么所有涉及这个接口的类都要重写。很显然，有俩功能的接口涉及更多的类。

- **3. avoid unnecessary heterongeneity**

这是为啥呢？有的时候我们的确要做一些多余的事情，来保证在一些部分缺失的情况下，仍然可以正确运行。例如，为了避免资料组放假带来的麻烦，客户接待组也需要知道资料存放位置。然而，我们是否有必要，为了避免电话公司倒闭带来的麻烦，而开一家本地电话公司吗？显然是不需要的。

- **4. distributed data**

  数据不存放在一块儿。这样，即使某一部分的数据丢失了，整体数据损失仍然很小。而且分布存储数据使得存储空间的可扩展性增强。

- **5. replicate data**

  数据备份。这样，在有限的运行环境下，我们仍然能够保证获取足够的数据。

Connectors

沟通设备。

- **simple and explicit**

  简单又精确！

- 需要警惕直接依赖与间接依赖

  另一个是，需要警惕直接依赖与间接依赖。

  比如，接待组依赖资料组，而资料组又依赖信息组去收集资料。这样接待组就间接依赖信息组，直接依赖资料组。这两者都是我们要警惕的。

Configuration

配置。

- 第一，我们要避免瓶颈。什么是瓶颈呢？就是当一个小组负责所有的数据处理。
- 第二，把数据放在需要数据近的地方。例如，文件室应该在资料组附近。再例如，信息数据应该存放在负责处理这些数据的主机上。
- 第三，我们需要location transparency。这就是说，组件是不需要关心数据存放问题的，保证了在数据规模增大的时候，组件仍然能够工作。

非功能性需求之革命性 -- Evolvability in NFP

一样滴，我们来熟悉一下概念！

---

evolvability

The ability to adapt to new requirements.

Components

工作小组，同复杂性相同。

Connectors

沟通设备。

- flexible。因为我们要随时准备革命，所以沟通设备必须也是灵活的。
- clear responsibility。只有分工明确，那么在我们更新一项事务的时候，我们才知道到底要更新啥。

Configurations

配置。

- 避免模糊的沟通设备。如果一个沟通设备，你不能确定他具体负责啥，那就更别提要进行革命了。
- 再有就是location transparency，因为新的要求可能带来数据位置的改变。