大型企业信息系统的架构设计

SD2C-2009

演讲文字整理

2010-5

前言

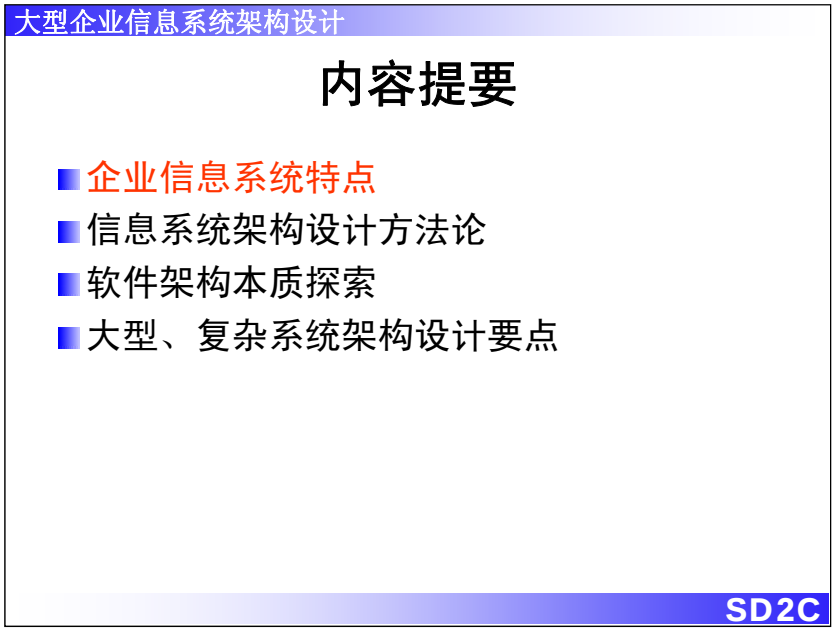
这是我在 CSDN SD2C 2009 上所做的《大型企业信息系统架构设计》的演讲的文字整理，这次演讲内容实际上很多，单纯看 PPT 难以全部了解，而我又没有时间写很多篇博客去系统性地介绍我在这方面的观点。好在 CSDN 对此次讲课内容有录音，我就此想出了一个偷懒的办法：把我的演讲内容直接整理成文字，除了方便阅读而做一些字面上的调整之外，力求保持原样，因此有些地方看着并不那么顺畅，也不可能像文章那样严谨，只求把我当时所讲的内容忠实反映出来，供大家参考。

由于平时很忙，整理工作做了几页就中断了，后来多亏博文视点的编辑卢鹄翔以及其他人接过了这个工作，替我做了整理。我最后根据录音做了一遍校对，基本上是保持了原貌。现在再看里面的部分内容，有些显得太简略，由于时间限制，没有展开讨论，恐怕说的不是那么清楚。还有一些内容，根据我的最新研究已经已经有了较大的调整，不过为了保持原样，也就不进行任何修改了。总之就是分享一下我的观点，抛砖引玉吧。

1. 标题

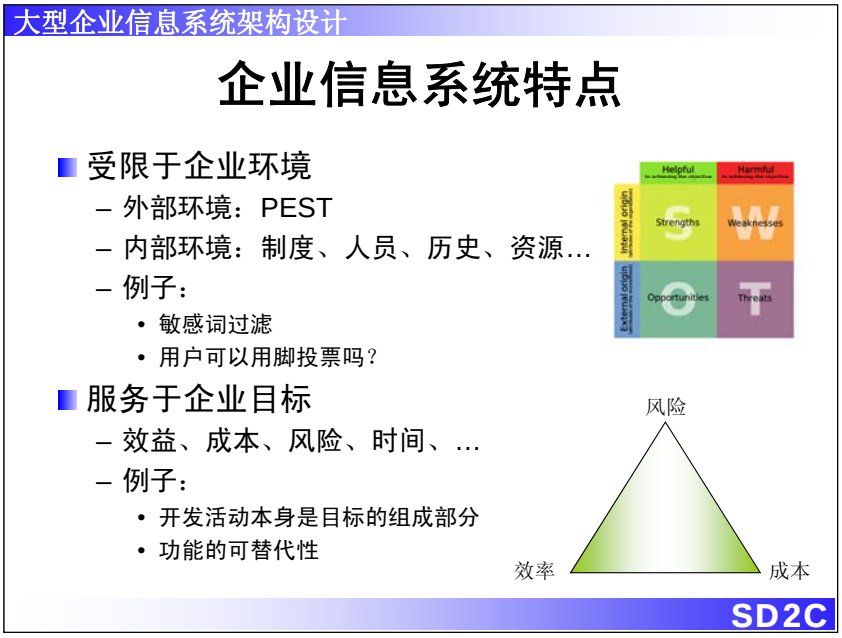
今天讲的主要内容是“大型企业信息系统的架构设计”，从标题来看，这个概念主要分为三个部分：第一个是“企业信息系统”，我们首先要给这个概念做个界定；第二个呢，是核心问题，也就是我们要做“架构设计”；第三呢，我们还要强调一下“大型”系统的架构设计，看看有什么特殊的地方。

2. 内容提要



今天的主要内容大概分四个部分，第一部分就是我刚才说的，企业信息系统有什么特点，要简单介绍一下。中间两个部分大致是讲一下架构设计的内容，包括一些方法论的介绍，也包括我们简单做一些探索性的工作。最后呢，我们回到这个“大”字上，就是系统足够大的时候，架构设计的要点到底在什么地方。现在互联网的时代来临了，大家都知道互联网系统处理的数据量一般都比较大。我们要看一看，企业信息系统“大”的时候，是不是和互联网系统有共同的特点，还是说有可能是完全不同的东西。

3. 企业信息系统特点



好，首先讲一下企业信息系统的特点，这一部分只有这一张幻灯片。企业信息系统的特点，从本质上来讲，有两个角度：首先是受限于企业的环境，其次是服务于企业的目标。从这两个宏观的角度来讲，其实互联网的系统也是一样的：互联网的系统，总也要受制于互联网这个环境，也要服务于企业在互联网上经营的目标。那么他们的区别在什么地方呢，我们来仔细看一下。

首先，我们讲企业的外部环境——因为环境一般分为外部环境和内部环境，说到外部环境呢，大家知道这个吧：“PEST 分析法”，这是企业外部环境的一个重要分析法。这是四个词的缩写：政治、经济、社会和技术，也就是 political、economical、social 和 technological。这四个方面呢，也就是一个企业在整个社会当中，以及在整个国际化的时代当中所面临的外部环境，等会儿我会举几个例子和大家说一下。外部环境应该说是对企业十分重要的，因为一个企业的发展不可能脱离它所在的外部环境。下面再说内部环境，内部环境就是企业本身内部的构成，比如说：它的制度、它的人员、它曾经的发展历史以及它现在所拥有的资源等等。这两个环境对于企业来说应该说是决定性的因素，既然如此，对于企业信息系统而言也是决定性的因素。比如说，大家有没有听说过 SWOT 分析法，相信有人知道，SWOT 分析法是研究企业发展战略的一个重要分析手段，实际上就是从企业的外部环境和内部环境两个角度来分析企业的发展战略。

等一下都我会讲到，那就是对于企业发展有决定性影响的因素，当然都是对企业的信息系统有决定性影响的因素。

现在我来举个两个简单的例子：首先，互联网的领域内大家可能都知道这个词：“敏感词过滤”。敏感词过滤可能是有一定我们中国特色的东西，因为这是受到我们国内政治环境的影响，这就是我们所讲的外部环境中的第一个——“P”。大家可能会看到，敏感词过滤也许是我们中国特色，也许西方国家不会有。我想说的是，整个互联网的环境，在他初创的时候，都是很宽松的，包括我们国内也一样，所以没有敏感词过滤的这个情况。随着互联网发展的越来越快呢，类似于敏感词过滤这种情况，也就是受到政治因素影响的这种情况，会越来越多。而作为企业的信息系统，由于很早，几十年以来就一直存在和发展，因此实际上很早就开始受到政治因素的影响，这一点我们后面还会讲到。这个例子中，如果我们构造一个搜索引擎，大家可能都知道 Map-Reduce 这种架构，但是如果要一个“带敏感词过滤功能的 Map-Reduce 架构”，应该怎么做呢？比如说你是在哪一个阶段进行敏感词过滤呢？这里就是想说明，实际上包括敏感词过滤这样的政治因素，都有可能对你的信息系统的架构产生非常直接的影响，这是一个例子。

第二个例子呢，不知道大家有没有听说过，在互联网上，“用户可以用脚投票”。也就是说你的网站、软件做的不好，用户的体验不好，他就不来用了，就去别的网站了。但是，我们想一下，在一个企业内部的信息系统会发生这种情况么？如果有一个财务软件做的不好，企业的会计人员能说我不用这个财务软件，我用自己家里拿来一个财务软件来给公司记账么？这是不可能的。那么这说明什么问题呢？我这里是要说明，“用户”这个概念，对于不同类型的信息系统而言，差别是非常大的。对于一个互联网信息系统而言，用户实际上属于这个信息系统的外部环境，也就是它的社会环境。而对于一个企业信息系统而言，用户实际上是企业内部人员这样一个内部环境。所以我们大家应该知道，当你建设一个信息系统的时候，如果你没有把这些重要的因素搞清楚，你连你的用户是内部环境还是外部环境都没有搞清楚的情况下，你设计的信息系统，肯定会出现很大的偏差。好，这些就是我们所讲的，受限于企业环境这样一个要素。

下面呢，我们是要讲“服务于企业的目标”。我们大家都知道，企业建设信息系统，无非就是那么几个目的，提高效益、降低成本、控制风险等等，其实就是这样一些目的。那么我们刚才已经说了，互联网的信息系统，它建设的目的，可能和这个就差别很大，比如说我宣传一个概念，吸引用户，然后我可以拿到 VC，在创业板上市等等。这个途径，和企业建设信息系统的目标，差别是非常大的，在这种情况下，如果我们不足够了解企业建设一个信息系统的目标，我们往往会陷入一种唯技术论状况，就是把这个系统建设得很漂亮、很完美、实现了我作为架构师或者作为程序开发者自己的目标，而没有去贴近企业的目标。

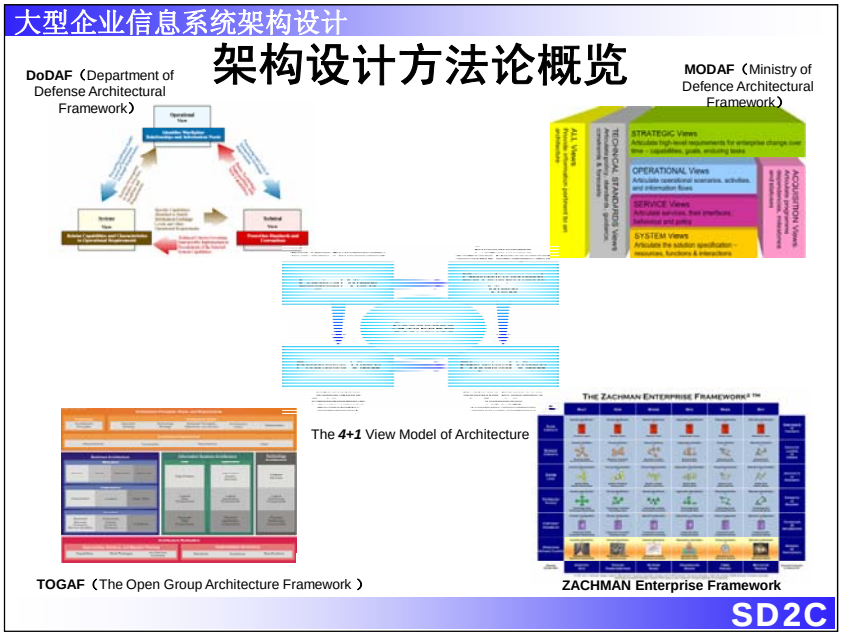
我这里举两个例子，首先是对于企业信息系统而言，开发活动本身也是这个企业的目标的一个组成部分，我想这点大家很容易理解。我们刚才举的例子是说，如果我们建立一个互联网的信息系统，可能它的投资方并不是很关心你这个系统的团队怎么样建设、怎么样开发、采用的工具是什么、管理流程是什么，他只是关心你最后做出来的网站的效果，以及你吸引的用户数。但是对于企业来讲，因为开发活动对于它的目标产生重要的影响，所以他会把开发活动本身也纳入他的效益、成本、风险的考虑当中，所以就会对你的开发活动产生非常直接的控制和影响。也就是说他必须管理你的团队是什么样、你采用的开发工具是什么样、你采用的方法是什么、架构是什么。比如有人说（后面我会举个例子），现在 ROR 很流行，两三年、三四年了，现在应该说已经足够成熟了，那么我想用 ROR 来开发一个企业信息系统可不可以呢？其实首先你就问一问甲方可不可以就行了，如果他不同意，你就不要想用了。当然，后面我们还会看到，还有很多其他因素决定你可不可以用 ROR 来开发企业信息系统。

第二个例子我想讲一下“功能的可替代性”，这一点也是企业信息系统和互联网系统的一个很大的差别。什么叫功能的可替代性呢，就是回到我们企业的目标，我们建设一个系统无非是为了达到这些个目标。那么，这些目标是不是每一个细节都可以用程序这种东西来帮助我们实现呢？通常是不一定的，有时候我们会看到，有很多的所谓系统的需求或者功能，我们要用程序开发出来，其困难程度比我们直接定一个规则，让人采用手工的方式实现，要大得多，因为有些工作并不是很适合于用计算机来做的。这些内容如果你不分析出来的话，

最后简单来说，你会把自己搞死，就是因为你一定要拿计算机去做那些不适合用计算机来做的事，因为用户会给你提出来，说这是我的需求。所以在这方面你必须进行分析，有些工作是完全可以用非信息系统的形态来实现的。这对于我们信息系统的整体架构是有非常重要的意义的，因为经常你会忽视掉这一点，在这种情况下，你会由于为了实现一个很难于实现的系统功能，而破坏了你整个的系统架构。而实际上这个功能，我们把它孤立除去，根本不用信息系统来实现，这种做法是完全可行的。互联网的信息系统普遍而言不具有这样的特点，原因就是因为——回到我们刚才说的那一点——互联网系统的用户与你之间没有任何直接的联系，他们只能通过浏览器，通过网络访问你的系统，你不可能为他们提供一个不通过你网站系统实现的功能，你提供不了，但在企业中是可以的。

所以，大家看我这一张幻灯片讲的很长（我这一部分只有这一页 PPT），就是为了说明，影响企业信息系统的这些因素，我们需要认识到足够清楚，以及他们与互联网信息系统之间有什么区别。这些内容在后面我所要讲的企业信息系统架构设计当中都会有体现。

4. 架构设计方法论概览

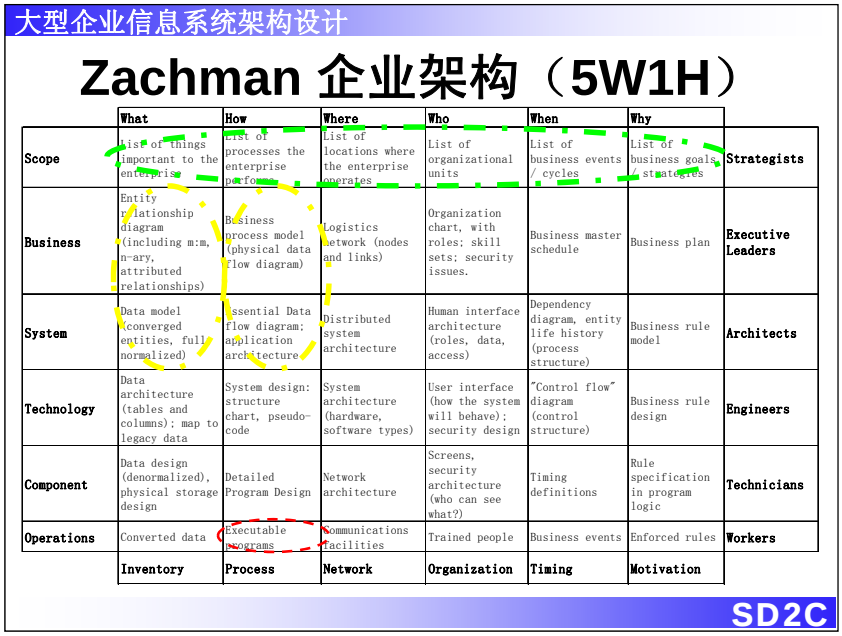


下面我简单介绍一下信息系统架构设计的方法论，为什么要介绍方法论呢，这里就要进入到架构设计领域的特点了。刚才我们说到了，要做企业信息系统的架构设计，最关键是要理解企业的内部环境和外部环境，以及企业的目标。但是从另一方面说，架构设计这个领域，在这二三十年内，越来越成为一个专业的领域。也就是说，已经有很多人做了大量的工作，构建出了一些架构设计的体系，目的就是为了抽象出架构设计这个领域当中（毕竟还是有）很多的客观规律，有很多前人经验的总结。孔子说：“学而不思则罔，思而不学则殆”，刚才我们讲的那些东西，实际上是“思”的过程，就是让大家反思/思考一下，企业的信息系统到底有什么特点。但是光想不学是不行的，我们还是要看一下，别人的方法论，在架构设计领域到底抽象出了哪些共性的东西，这些东西都应该是我们在架构设计过程中可以用到的。下面简单说一下（这部分不是我讲的重点，因为时间也不太够），这五个方法论应该说都是目前比较有名的方法论：

DODAF，这是美国国防部主导的架构体系，MODAF 是英国国防部主导的架构体系，从名字上也能看出来与 DODAF 有一定关系，从 DODAF 借鉴了很多东西，然后自己进行了一些构造。TOGAF，这是 OpenGroup 的架构体系，相当于是一个国际标准化组织的架构体系。Zachman Enterprise Framework，这是一位老先生，Zachman 先生创立的一个架构体系，也有二十多年的历史了。中间这个图呢，是 4+1，不知道有多少人了解这个体系，熟悉 UML、Usecase 等等与

RUP 相关的体系的人可能会了解得比较多一点。这五个架构体系我想大家可以看出来，有两个是国防部的，一个是国际组织的，一个是个人或者说企业的，还有一个是领域专家（Kruchten）的，应该说这五个体系还是有一定代表性的。从这里我们就可以看出来，很多的人，无论是机构、国际组织、公司、个人等等，都有自己的架构方法论。这里就是简单介绍一下，后面主要对两个架构方法论做一下介绍。

5. Zachman 企业架构（ 5W1H ）



首先是 Zachman 的企业架构，我简单列了一下，就是 5W1H。（字体可能比较小，大家下来看 PPT 就可以了）。5W1H 这种模型，我想大家都听说过，不仅仅是企业架构，就是 What、Who、Where、When、Why、How 等等，这样的形态在所有领域的分析模型中都可能会用到。这样的一个体系，它的主要特点是什么呢，我这里没有时间详细地介绍其中每一个点的内容，主要是让大家形成一个印象，后面还会用到：这样一个体系的特点，是它非常适合于分析——非常适合于分析企业信息系统架构的方方面面。这个我想大家应该比较容易理解，任何一个领域内，这种 nW1H 的方法，都是一个非常有效的分析方法，

因为用它可以把方方面面的要素都列出来。大家不要忘了我们一开始讲的内容，也就是影响企业信息系统架构的重要因素——内部环境、外部环境、目标，都在这里有体现。

比如说我们看到头一行（纵向的内容我们等会儿再讲）：What：List of things——你这个企业是做什么的；How：List of Processes——如何做。比如说一个银行和一个保险公司，他们做什么这一点上就肯定是不一样的。进一步而言，如何做，这又是业务流程方面的问题，比如说同样是银行，或者同样是政府，也没有两个流程完全一样的。Where——地理位置，比如说我是一家在中关村某一个写字楼上有一个小办公室的公司，和我是全国范围内有三万六千个网点的银行，他们的信息系统有可能一样么？不可能。Who——人员，或者角色，就是这个企业有多少参与信息系统使用或者建设的人。When——时间，就是说这个企业的业务以及信息系统有没有比较鲜明的时间特征，这个我们可以等会儿再讲。Why——就是终极的，企业的目标，也就是你的企业为什么要做这些事情，以及要这么做的原始驱动力是什么。所以这样一些维度就很有利于分析我们前面所说的企业的内部环境、外部环境等等因素和目标。

那么再看纵向，纵向实际上是一个层次化的结构，就是从宏观一直到微观。Scope 是最粗线条的，我们看到这六个点都是“List”，你只要把要点列出来就可以。再下面就细化了，Business，业务层面的，比如说你有什么样的实体、什么样的业务流程。再往下呢，系统层面，你的数据模型是什么。再往下是技术层面，如何构造这些数据。再往下到组件层面；再往下到执行层面。由于纵向是一个层次化的结构，横向是一个角度的结构，有了这样一个矩阵的结构，就可以把企业信息系统的方方面面全都列在这里。这就符合了我们前面所说的，你要决定一个企业信息系统的架构的时候，必须所要考虑的全部因素都在这里。

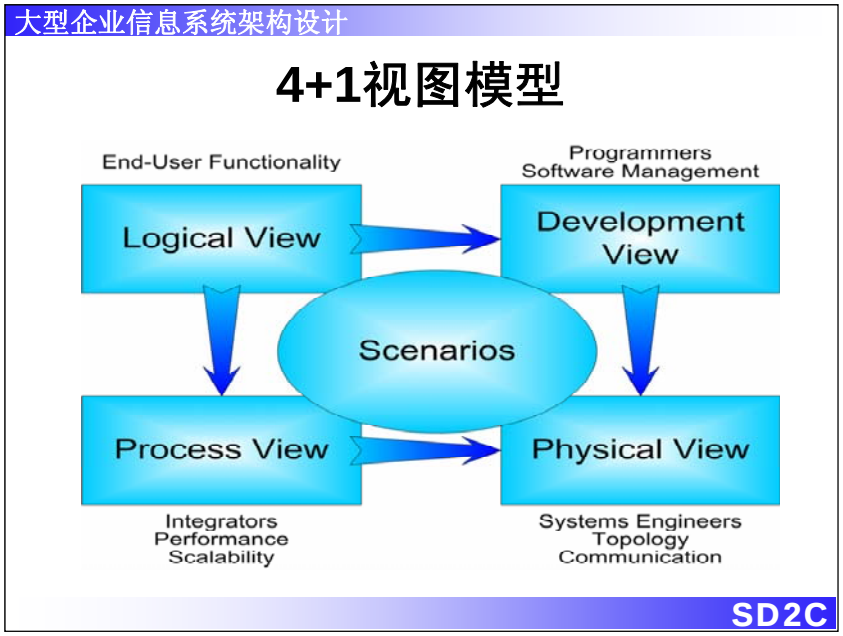
顺便强调一点，就是反过来说，这个框架适合于分析，适合于列出所有必要的内容，但是并不一定适合于设计。我们可以看到，所有的地方都是说企业信息系统应该有什么、有什么，但是怎么做，没有列出来，这是这个框架的特点。

6. Zachman 企业架构

	数据（什么）	功能（怎样）	网络（哪里）	角色（谁）	时间（何时）	动机（为何）
目标范围	列出对业务至关重要的元素	列出业务执行的流程	列出与业务运营有关的地域分布要求	列出对业务重要的组织部门	列出对业务重要的事件及时间周期	列出企业目标、战略
业务模型	实体关系图（包括M：M关系、N-ary关系、归因关系）	业务流程模型（物理数据流程图）	物流网络（节点和链接）	基于角色的组织层次图， 包括相关技能规定， 安全保障问题。	业务主进度表	业务计划
信息系统模型	数据模型（聚合体、完全规格化）	关键数据流程图、 应用架构	分布系统架构	人机界面架构（角色、数据、入口）	相依关系图、数据实体生命历程（流程结构）	业务标准模型
技术模型	数据架构（数据库中的表格列表及属性）、 遗产数据图	系统设计： 结构图、伪代码	系统架构（硬件、软件类型）	用户界面（系统如何工作）、 安全设计	“控制流”图（控制结构）	业务标准设计
详细展现	数据设计（反向规格化）、物理存储器设计	详细程序设计	网络架构	屏显、安全机构（不同种类数据源的开放设定）	时间、周期定义	程序逻辑的角色说明
功能系统	转化后的数据	可执行程序	通信设备	受训的人员	企业业务	强制标准

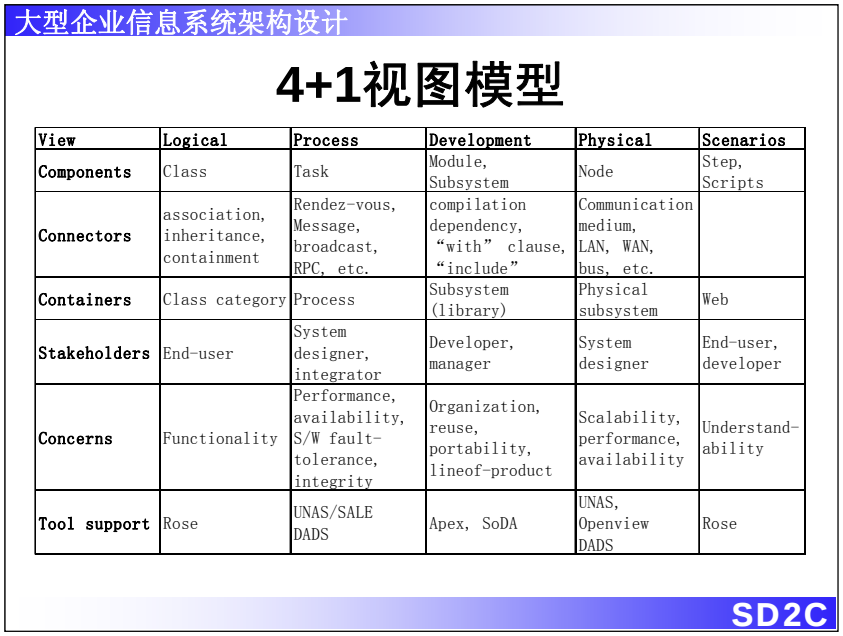
这里是一个中文版的内容，中文不是我翻译的，但我看了应该没什么问题。再简单说一句，就是 Zachman 架构有一个网站，也就是这个公司的网站，这个架构从 87 年开始（演讲中是 85 年，应该是记错了），到现在二十多年的历史了。这个公司本身也提供架构师的培训和认证，我不是给他们做广告，因为我和他们一点关系也没有。我想强调的一点是，这个架构经过二十多年的变迁和他最初的时候也是有很大差异了。我这里也就顺便提出来，因为前两天也有人问到，架构设计到底应该是什么样的规律，是不是可以完全形式化。从这个架构体系也可以看出来，二十多年来它一直在变，可以说最近十几年每年都会推出一个新版本。也就是说架构设计实际上没有绝对的一定之规。大家有兴趣可以去这个网站看一下，为什么我要贴这个网站呢，我不是要给他做广告，是因为这个网站有一个好的地方，网站上就有这个矩阵的一个图，每一个点都是可以用鼠标点进去的，里面会有对相关概念的解释，大家可以通过这个形式去简单学习一下，看看企业架构设计都要考虑哪些因素。

7. 4+1 视图模型



下面一个，4+1 视图模型。有多少人知道的？这个我们可能不会将的太多。这是 Philippe Kruchten 发明的，搞 UML 的人可能都知道。4+1 视图模型把系统分为 5 个视图，Logical View 一般来讲是逻辑视图，Development View 是开发视图（程序员、管理者关心怎么开发这个系统），Process View 是过程视图，因为我们在系统当中会关心一些之行流程、性能、稳定性这样一些内容，Physical View 是最终的物理实现，包括软硬件，包括网络。中间的 Scenario View 是贯穿这四个视图最重要的，以前叫 Use-Case View，即通过 Use-Case 这样需求分析的方法。这个理论是通过 Scenarios（场景）的分析，就可以分析出来逻辑视图、开发视图是怎么样的，过程视图和物理视图是什么样的，这就是它的一个重要思路。

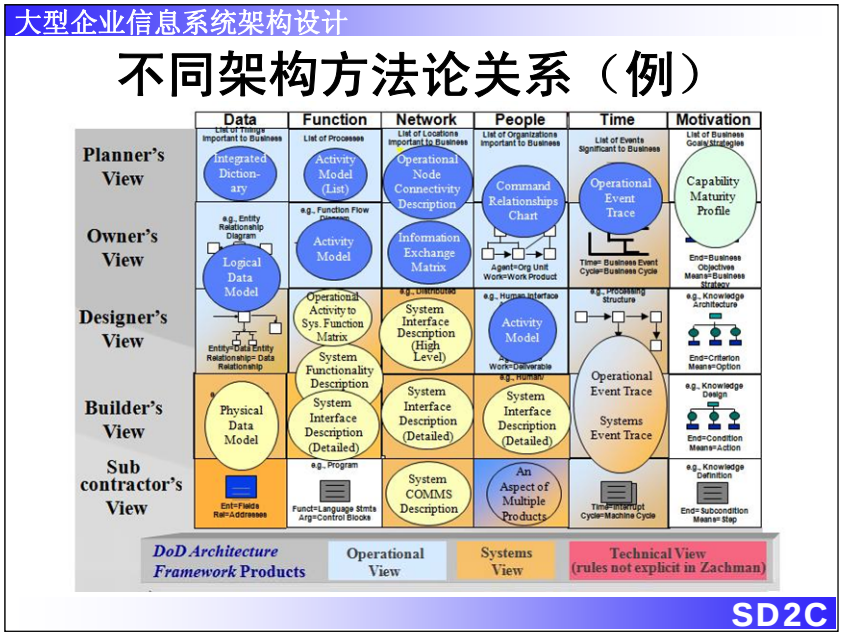
8. 4+1 视图模型矩阵形态



它也有一个矩阵形态，我们看到它横向有五个视图，纵向是每个视图中包含的元素。Components 是它里边包含的实体；Connectors 是它们之间的关系和连接；Containers 是实体的大范畴，比如说这些 class 的归类；Stakeholder 呢，做过 Use-Case 的可能都知道，现在普遍翻译成涉众，也就是会有哪些人参与其中，或者有哪些系统参与其中；Concerns 就是关注点，比如说功能、性能。最后还有一个 Tool support，就是在这样一个视图当中，我可以用什么工具进行设计。这个视图我就不仔细讲了，还是给大家留一个印象，就是跟刚才那个 Zachman 视图比较起来，这样的一个视图很适合于做设计，刚才那个模型很适合于做分析。这是因为它把每一个视图中的关键点都列出来了，这是它重要的一个价值。

因为这两个视图后面我还会一个重要的结合点来讲，所以这边短时间先讲到这样，大家先留下刚才那两个印象，一个是分析，一个是设计。

9. 不同架构方法论关系

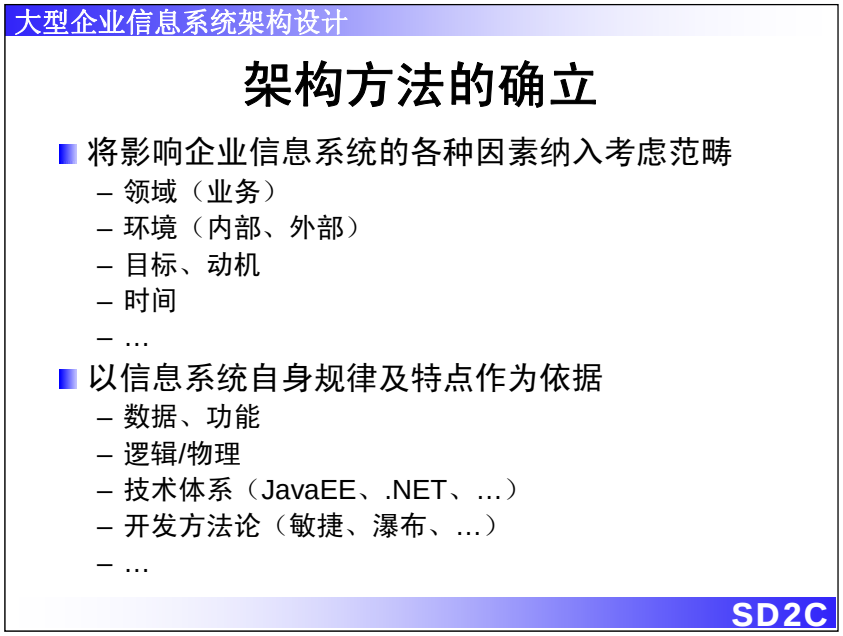


这个地方我简单地过一下，我们知道刚才大家看了五个方法论，实际上其中我们只讲了两个。我们知道这些方法论之间应该还是有一定的关系，因为我们不可以想象说这个世界上有五个甚至十个架构方法论（实际上十几、二十个至少还是有的），如果它们之间都是完全相互隔绝的，任何两个方法论之间都没有相互关系的话，那么根据这些不同的方法论设计出的信息系统，最后难道完全不是一个东西吗，应该不可能。所以说这些架构方法论之间必然是有一定关系的。我们这里简单说一下，这是一个 DODAF 和 Zachman 方法论的一个映射表，从颜色的角度大家可以看到，DODAF 里面有 3 个主要的 view，和 Zachman 之间的 6 个维度有一定的映射关系，浅蓝的底色代表它的 operational view，黄颜色的底色代表它的 system view。

这块就是想和大家简单说一下，因为这一节我们就是简单介绍一下架构方法论。当然在这里我和大家分享一个我个人的观点：为什么刚才我没有讲像 MODAF 这样一个框架？实际上这样一个架构方法论在国际上是非常知名的，不过我要跟大家说一句话：“珍惜生命，远离国防部”。为什么要这样说，因为大家做企业信息系统做多了，可能会反思：我们知道像美国国防部实际上它在整个软件开发体系做了很大贡献，占有很重要的地位。比如 CMM，是它和卡耐基·梅隆合作，等于它委托卡耐基梅隆来做。MODAF 又是它自己设计的，像这样一个东西，在我们整个的软件开发体系中起着重要的作用，但是我要跟大家

说的是，最好大家不要总想去使用美国国防部的这一套东西。因为它所设计出来的东西是具有鲜明的个性的东西——简单地来说就是一句话，世界上只有一个美国国防部，对不对？总不会有两个美国国防部。但是，这世界上有多少企业呢？我们要服务于一个领域的企业，我们去用世界上只有一个人或者一个客户在使用的架构或者开发模型，大家觉得会合适吗？肯定不合适，我只能这么跟大家说。这个话就说到这里，大家可以自己回去再考虑这个问题。

10.架构方法的确立



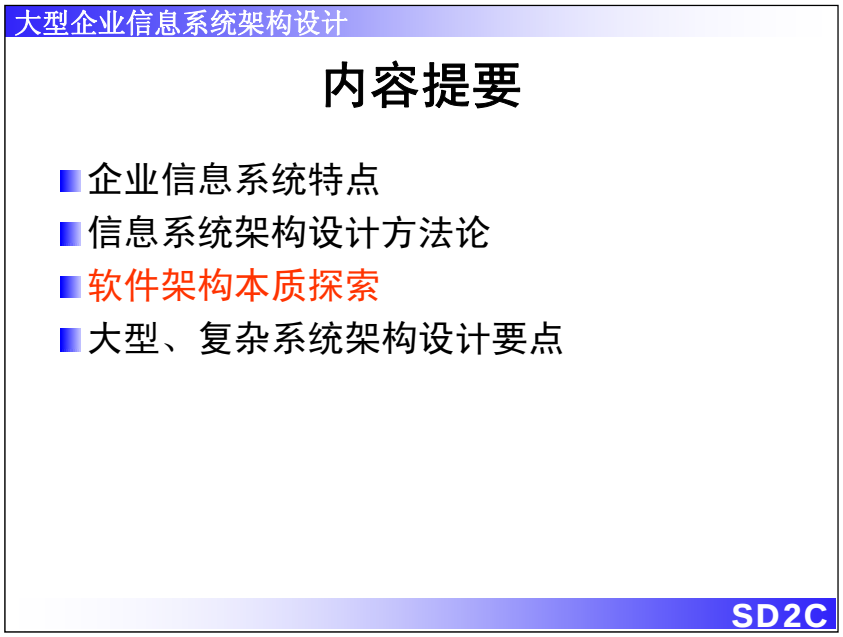
刚才我们实际上讲了两个主要的部分，一个是企业信息系统的架构，信息系统的特点；一个是一些架构方法论。我这里想讲的是，我们作为一个架构师，我们要设计一个企业信息系统最主要的考虑点是哪两个入口：第一，我们首先要将影响企业信息系统的各种因素都纳入考虑范畴。我们刚才所说的领域，这是最重要的。我们经常这么讲，我做银行的信息系统的设计，我知道银行这个业务的领域，我就是做再多年，我的架构功夫再深，我换到一个完全不同的领域，制药业，电力企业，真的能够做好架构设计吗？我相信大家都觉得可能性不大，至少我告诉大家我认为应该是不可能的。不要以为架构设计真的想那些

方法论那样可以完全的抽象出来，作为一个客观的领域，这是不可能的。领域还是最终决定架构的根本因素。那么包括环境，包括目标、动机、时间、后面都会讲到。

再有一个当然就是信息系统的自身规律，这也是一个依据。我们大家都知道 dijkstra 的名言：“算法+数据结构=程序设计”。设计系统架构的时候能不考虑数据和功能的架构吗？同样的，我们现在越来越多的分层方式，决定我们每一点都可有能分为逻辑的架构和物理的架构，包括技术体系是 JavaEE 还是.NET…。如果你作为一个有 15 年经验的 JavaEE 架构师，你说我在什么都不考虑的情况下，我就可以转为.Net 的架构师吗？不可以。这就说明技术体系对架构还是有一定决定的因素，包括开发的方法论都是一样，这一点在后面还会详细地讲到：选择什么样的开发手段，对架构设计而言是很重要的。大家可能都会认为说以往我们所宣传的架构都像那些 view 一样，都像是系统一些静态的东西，其实不是这样的，你设计出的架构再好，开发不出来，你没有人，没有足够水平的人，没有足够的方法可以保证你这个架构开发出来，也是没用的。我这里多讲两句，举一个例子：鸟巢，大家都知道，架构这个词最早就是从建筑业来的。大家知道吗，鸟巢它的每一根梁都是斜的，没有直的梁，可是我们想一想，我们作为一个工人，作为一个建筑工人，我们平时怎么干活？是不是最主要的就是两种，一个就是站着干活，一个是躺着干活。站着干活就是上下左右，我从左到右系一根条，从上到下钉一排钉子。躺着干活是水平面的，就是这样。所以鸟巢就面临一个最大的问题，它每一根梁都是斜的，没有衡平竖直，那么工人怎么干活呢？就是因为这一点，所以在鸟巢这个图形设计出来之后，很多的专家都认为鸟巢做不出来，就说这个架构设计本身就不合理：这个不合理的因素完全不来自于它的承重结构和形态，而是来自于它没法施工。没法施工的架构当然也就是个不合理的架构。这很明显。但是最终鸟巢还是做出来了，为什么？没法施工只是你在某种形态下觉得施工的成本过高而已。我们国家有钱，为了搞奥运会什么都可以做，最后大家知道吧，在电视上看过，工人吊一根吊带，一个梁是 30 度的，人就斜成 30 度，不又成了上下左右了吗。所以说，实际上一个架构设计，它跟你最后能不能做出来是很有关系的。如果我们再突破一点，不是鸟巢，是什么蚁巢蜂巢，也许就真做不出来了。真做不

出来，那它就是不合理的。但鸟巢从某种意义上来说还是合理的架构，什么样的一个合理架构呢？它是你还能够承担这个开发的成本，那它就还是合理的。所以这样就还是回到我们刚才所说的，为什么我会强调开发的重要性呢？因为在企业信息系统当中，开发活动本身是企业整体活动的一个部分。它也要控制成本，也要收回效益和控制风险。所以我们一定要考虑在设计架构的时候，（顺便说只能说一句，因为我们现在讲互联网信息系统的时候，大家普遍都在讲大数据量，多处理，实际上不是这样，而是说在企业当中，你有多少钱干多少事。）你设计的架构必须要满足你的目标，这就是我讲的架构方法的确立。

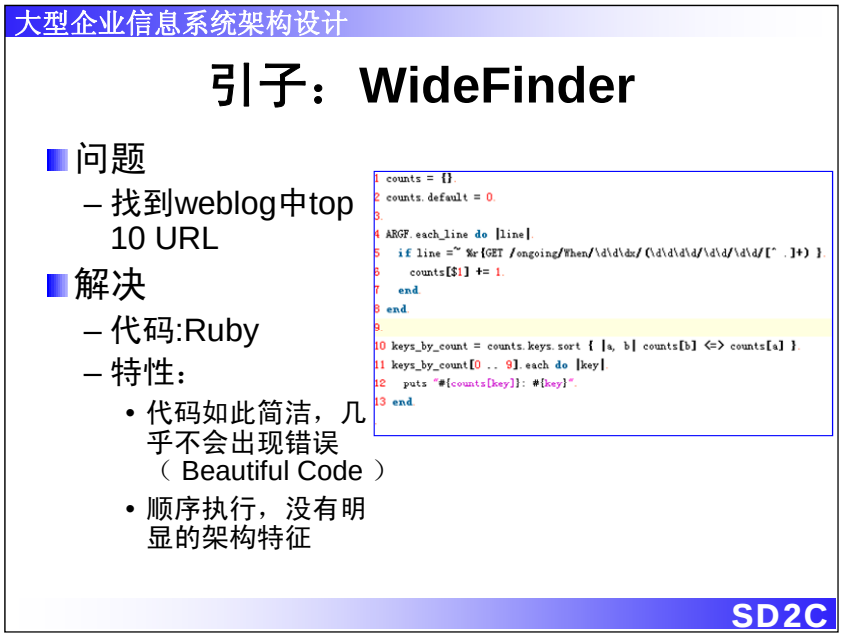
11.软件架构本质探索



其实我刚才所讲的那些主要还是在讲企业信息系统的特点，介绍了两个方法论，主要是为了反映我刚才介绍企业信息系统特点的思路。后面我们将看到这两个方法论怎么用。刚才我说了，今天主要讲三个主题，已经讲了两个了。信息系统的特点，然后架构方法论和架构设计的介绍。中间必须岔开一条道，来讲讲我对软件架构本质的探索。为什么要讲这个，因为在后面讲到大型的信息系统的时候，不可避免的要考虑到一个问题：软件架构的本质是什么？如果

你不抓住本质的话，大型信息系统的设计普遍会出现偏差。

12.引子：WideFinder

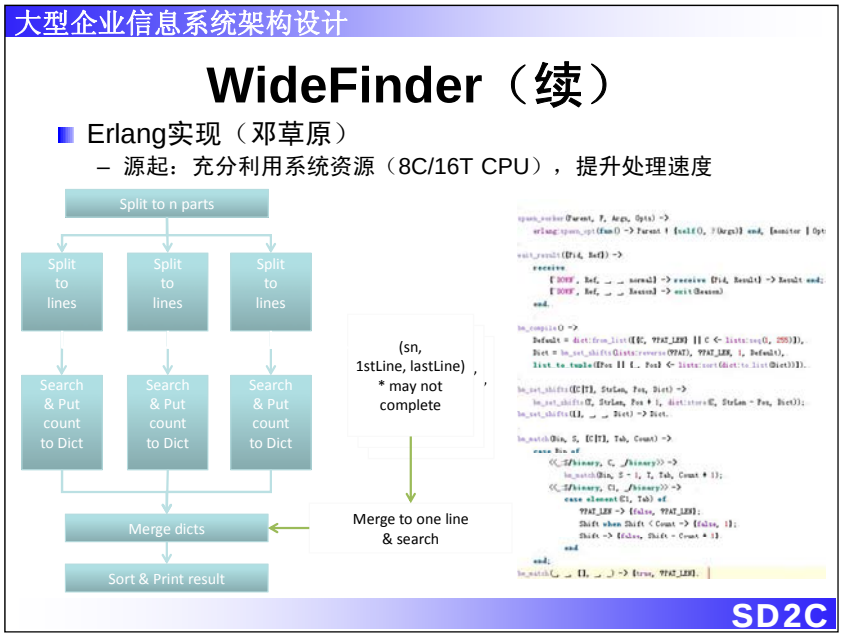


我先简单讲个例子，我不知道有多少人知道 WideFinder 这个例子（不知道邓草原有没有在讲 scala 的时候讲到过？）。其实很简单，就是说在一个网站的日志，如记录了谁在哪个时候访问了哪个 url。这些日志下来每天可能有 500M？1 个 G，2 个 G，都有可能，取决于你网站访问量的大小。那么你想知道的一件事，就是在我这个网站当中哪一个 url 点击最多。比如在一个新闻网站，如果我知道哪个 url 是点得最多的，那我就知道哪条新闻看的人最多。就是这么一个原始的驱动力。那么它的解决是什么呢？WideFinder 的原始的解决有一个 Ruby 的代码，大家看右边这段代码，从一个大型网站的日志当中，找到访问最多的 10 个某种形态的 url，并且把它倒排序打印出来。我说的这句话就是这段代码实现的功能。

我不知道大家对 Ruby 熟不熟，我就不介绍这个内容了，我就告诉大家，一共 13 行代码，还有 2 个空行和 3 个 end，还有 2 个声明语句，也就是说这段代码真正有效的内容也就是这么 5-6 行。那么这个代码有什么样的好处呢？就是我刚才所说的那句话，从一个大型网站的日志当中，找到访问最多的 10 个某种

形态的 url，并且把它逆序的排列出来。实际上就是这段代码所做的事情。也就是说，这段代码和我所描述的我想要达到的目标几乎是一一对应的：对于每一行进行一个判断，如果看到它是这个 url，就对针对这个 url 的 count 加 1。然后做个排序和打印，就是这样。它的好处是什么呢？代码如此简洁，几乎不会出现什么错误。这就是我们平常所说的 Beautiful Code。不知道大家有没有看过 Beautiful Code 这本书，这就是那本书的头一个例子。那么这样一段代码，我想大家会同意说它没有什么架构可言，从第一句话执行到最后一句话，整个代码就结束了。

13.WideFinder（续）



为什么要举这个例子呢？我是为了给大家看另外一个事情。这个实现是 Erlang 实现的一个版本，功能是完全一样的。而且是我们这次的讲师邓草原实现的一个版本。为什么会有这个实现呢？它的一个本质的原因是希望提高分析的速度。因为我有一颗 8 核 16 个线程的 cpu，我希望能够提高处理这个文本和最后打印出结果的速度，所以他就做了这样的一个程序。大家可以看到右边这个程序是 Erlang 的程序，实际上也就是 100 来行，而且还包含了很多各种各样的具体算法实现。所以实际“有效”的代码也并不多，但它的体系结构是这个样子的：就是说我首先要采用把这个文件分而治之的办法，我把这样一个很大

的文件，如 1 个 G 或 10 个 G，把它切成 10 块，然后对每一块进行我刚才所说的搜索 url，并且把它计数。那所有的这些计数最后还要合并，因为是分开执行的，合并完了再排序再打印。那么右边这里面还有一个小的异常情况需要处理，就是说我们要把文件分块的话，它不一定每一块都是整行，而我的 url 搜索是要针对整行来进行的。那怎么办呢？我切成 10 块的话，我得把每一块的头和尾上那些半行的东西找个单独的线程把它们拼起来，然后再进行搜索，然后再跟这里的结果一起进行合并。所以大家看到，虽然是这么一个小的程序，但它已经明显地体现出了一个架构的特征。大家可以同意这一点吗？我们不能说它是一个架构，因为这程序实在太小，但是它很明显有个架构的特征。

14.架构分析

大型企业信息系统架构设计

架构分析

Zachman：

系统	What	How	Where	Who	When	Motivation
Beautiful Code	网站日志 Top 10 URL	日志文本搜索	Local	N/A	Daily (?)	用合适的语言进行简洁的实现
Cao Yuan	网站日志 Top 10 URL	日志文本搜索	Local	N/A	Daily (?)	用合适的语言，充分利用多核CPU提升处理速度

4+1：

系统	Logical	Process	Development	Physical	Scenario
Beautiful Code	一段顺序执行的代码	顺序执行	Ruby: 循环、正则表达式、dict、Sort	Single-core	日志分析
Cao Yuan	主控、分割、拼接、合并	多线程/进程并发	receive/end. erlang:spawn_xxx	Multi-core	日志分析

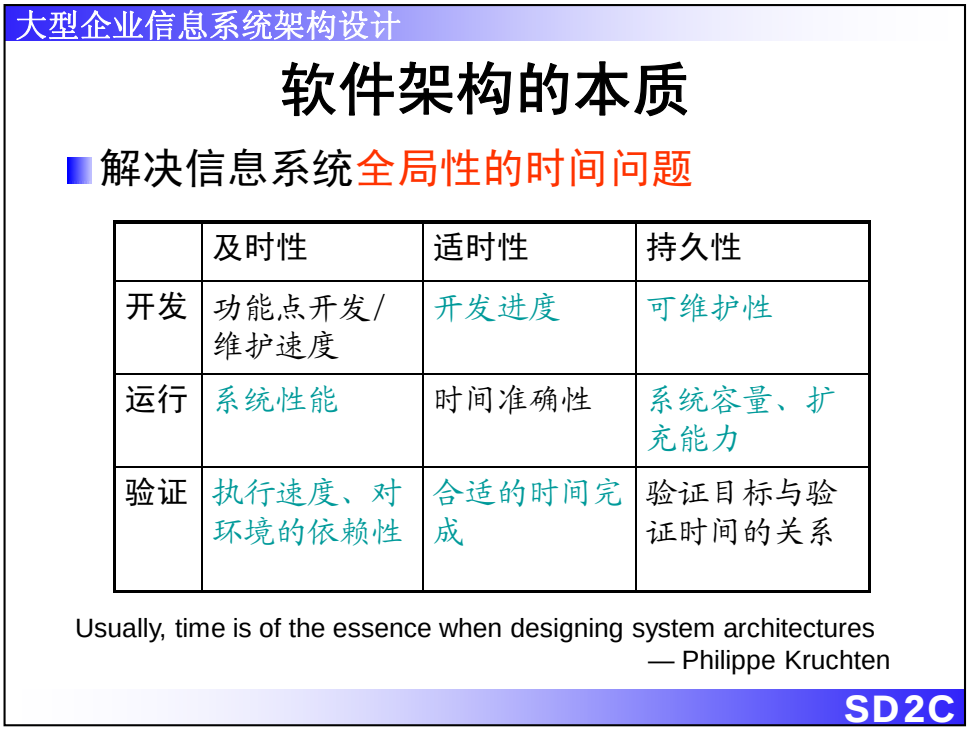
SD2C

好，那我们就来分析一下——因为我们想要探讨软件架构的本质——为什么会有这种情况的发生？首先我们用一个很不精确的分析——因为对每一个架构都有很多内容，我们没法做到意义分析。我们就借用 Zachman 的一个方法，分析一下产生这个原因的影响和现象。也就是说从一个 Ruby 的 beautiful 的代码到这个 Erlang 的架构的代码，产生这个状况的原因是什么？我们刚才说了

Zachman 是适合分析的一种框架，我这里就简单说一下，大家下去有兴趣可以看或者讨论。实际上我们可以看到，最终我们要实现的 5 个要素都没变，唯一变化的是 motivation，也就是动机/意图。就说我们的动机变了，本来我们可能想说就这样一个小程序，我们用一个 beautiful 的 code 去实现，这样的话看的人也容易懂，将来也好修改。后面不行了，我受到时间的压力，我希望提高速度，另外我也有好的机器。所以其实在几乎所有要素没有变的情况下，motivation 变了，导致我们整个信息系统的架构产生变化。这个 motivation 是什么呢？主要就是一条——提高速度。

下面我们再用 4+1 的方法来分析一下。我们可以看到 4+1 跟上面这个正好反过来。它几乎所有的形态都变了。这就可以看出来，我们刚才所说的 4+1 是一个很适合设计的框架，很适合去分析信息系统内部的设计。既然这两个程序差别这么大，那么它们内部所包含的内容当然完全不同。这一点我就不多讲了，逻辑的视图，执行的视图，开发的视图和物理的视图都是不一样的。但是有一点是比较一样的，什么东西比较一样呢，就是 Scenario。要做的场景一样，都是拿来一个文件，对他进行分析，打印出结果。那这说明什么问题呢？场景并不足以决定我们架构的设计。这就是我为什么说 4+1 这个视图在架构这个领域也不足以完全指导我们进行架构的设计，因为它所强调的是通过场景的分析得出结果和内容，那么如果场景的分析不足以导致一些内容的话，那么这样设计出的架构还是有一定的不足之处。

15.软件架构的本质



这里就来到了本次演讲的一个重点了，就是软件架构的本质：软件架构从本质上来讲实际上是解决信息系统全局性的时间问题，这是软件架构的最核心的本质。大家回想一下刚才的那个小例子，实际上说软件架构在绝大部分情况下来讲是一种被迫的行为：如果我不是为了加快处理的速度，我何必要搞这么复杂的程序呢？beautiful code 一共才有 13 行，还有 4 个空行，3 个 end，有效的代码只有 5 行，谁都看得懂。如果我一个信息系统，哪怕就算是有 1 万个功能点，我每个功能点都能像 beautiful code 那样写出来它就实现了用户的目标，它们之间相互没有任何关系，如果那样子的话，我还需要架构吗？肯定不需要。

好，回到这个话题，也就是说，就我们刚才这个例子问题出现在哪？我时间解决不了，我要速度，我用 beautiful code 做不到，我就必须要想办法，那么于是就设计出刚才那样一个小的“架构”来，就可以把这个问题解决掉。

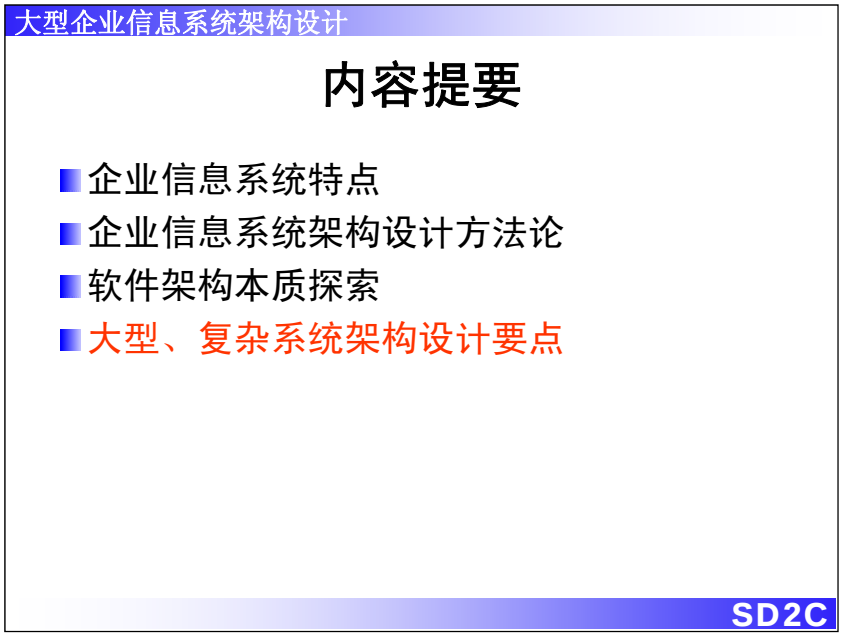
当然这里我要强调一点，刚才我是举了一个例子说明了这件事，我的这句话这个软件架构的本质可不是从这个例子中分析出来的，如果是那样的话，那太可笑了。你搞了一个 WideFinder 的例子，你就说你理解了软件架构的本质了，就是全局性的时间问题，那就太可笑了。这样一个结论，顺便说一下，是我长

期以来对于软件开发的总体过程做了一个全面的语义学的分析，最后得出的一个结论。这个话题太大了，我今天就不讲了。

我还是直接讲主题，全局性的时间问题都有哪些？这是很重要的。大家可能可以从刚才的那个例子可以看到，和大家的直觉一样，普遍来讲要提升性能，我需要一个好的架构，这点大家在直觉上都应该是一致的。那么我告诉大家，系统的性能在于架构的这种作用当中，也就是 1/9 的作用。就是在运行的及时性，及时性就是快，in time，在某一个时间范围之内就把它做完。适时性就是我们所说的 on time，就是准确，就是如果我要在 0 点 0 分 0 秒要做一件事，要触发一个事件，我必须要准确。持久性，lasting，就是随着时间迁移，我的信息系统还能不能满足我的需要。这是时间的 3 个维度。

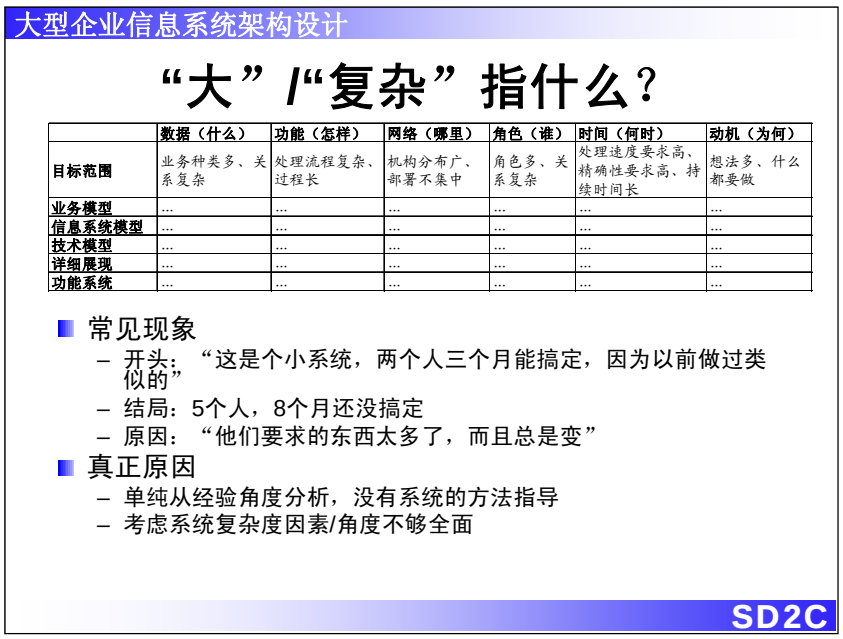
那么我们说全局性的时间问题，都有哪些全局，哪些角度呢？3 个角度。系统软件的开发、系统的运行，这两点大家很容易理解。系统的验证，这点大家以前接触的可能不是很多，我今天也没有时间讲，但我有一点要讲，就是这个维度就是我所说的要对于软件开发总体过程要做一个深入的语义学的分析，才能够得出这 3 个维度。总之我们不讨论这个问题，我们暂且就承认，全局性的时间问题，就是这 9 个点。我这里标出颜色的这几个点，性能，开发进度，可维护性，这是我们大家接触比较多的。像这样的一些内容，如时间准确性是很典型的，在一个小的系统当中时间准确性不是很重要，而在一个大型复杂的系统中时间准确性是十分重要的。这里我没法再去讲太多内容，大家要有这样的一个概念，后面我会用到一些内容来做一些分析。我这里最后讲一句话，Philippe Kruchten，就是我刚才说的 4+1 模型的创始人，他就是讲：Usually, time is of the essence when designing system architectures。他这句话实际上表意是说，时间是架构设计的本质，和我这个上边呼应起来了（总算也不是完全我一个人在这里“独孤求败”）。他这么讲，是出现在一篇文章中，他讲的这个内容从语境上来看，实际上是主要讲开发领域的角度。也就是说设计架构的时候时间很重要，那么实际上我是因为做了一个这么架构的分析，我把架构时间维度扩充到了 9 个，这 9 个维度都是对于信息系统非常重要的维度，这些内容都是要通过架构来保障的。

16.大型、复杂系统架构设计要点



刚才那些东西都讲完了，最后再亮点儿真家伙了。刚才全是分析，分析了半天，这样一些内容都知道了，最后大型复杂的信息系统到底怎么做？

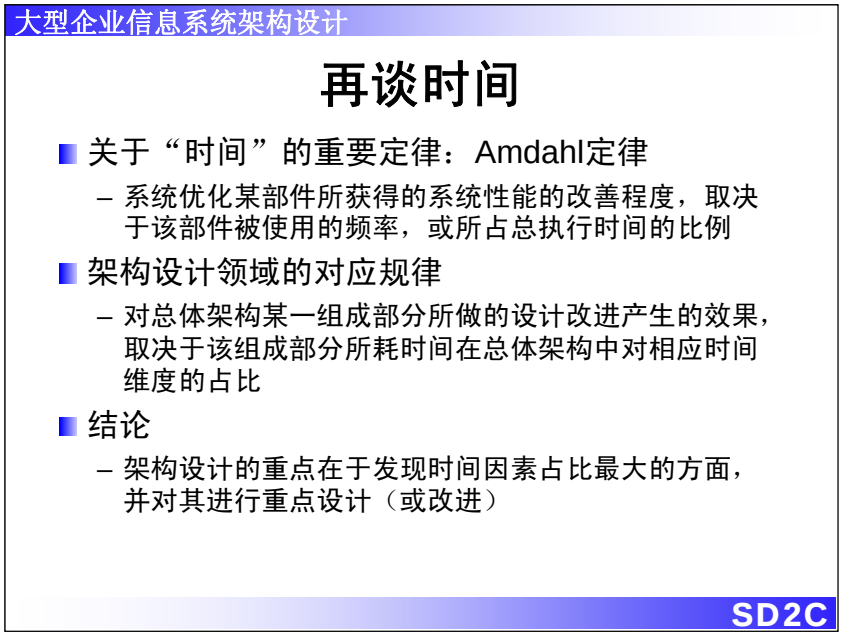
17. “大”/“复杂”指什么



开头已经讲了，第三个要点，就是什么叫大、什么叫复杂？现在把这个问

题理解清楚，我们才能说应该怎么做。很简单，这里是一个 Zachman 的模型，我只列了头一条，后面有时间我们再讨论吧。大是什么呢？What——数据：种类非常多，关系非常复杂。这就是大了。功能：功能特别多，1 万个功能点和 100 个功能点的系统大小能一样吗？不会。网络：刚才说过了，全国范围内的几万个网点的系统，和一个办公室内 5 台机器的系统能一样吗？角色：参与的人很多。时间：包括处理速度，包括精确性，包括持续时间长，大家不要忘记这一点。我刚才讲的那 9 个维度很重要，不要看到一个系统好像小，结果最后它因为处理持续的时间长，它要求这个系统要运行 10 年，20 年，它到那个时候系统就大了，到那个时候你的系统架构设计就可能有问题了。还有，motivation：想通过这个系统达到什么目标，这点很重要，我刚才在第一篇的时候讲过这个例子，功能可能有时侯是会有替代性的，它想达到的目标不一定都能通过系统实现。反过来讲，如果你都要给它做，这个系统可能就会变得巨大无比。从这个角度分析看来，“大”无非就是设计这些。那么很显然，刚才我们都说了，设计一个大系统，必须要有一个手段，把所有这些——比方到底哪些地方大，为什么大——都要分析出来。我举下面这个简单的例子，大家可能都碰到过，大家都知道常见的现象，我一看到这个系统以前做过，功能也差不多，两个人三个月就可以搞定了，什么概念，就是六个人月。结果，做做做，哎呀，他们要的东西太多了，而且还天天变。软件工程老是在讲需求变化，最后呢，搞了五个人八个月还没搞定，五八等于四十，相当于六的七倍。这个成本和时间都相差比较大。为什么呢？从刚才这个我们可以看出，我们还是要有一个方法论，单纯从经验的角度一眼去看这个系统有多大，那是不行的：你看错了怎么办呢？所以还是要有一个系统的手段，把复杂度的因素和角度全都给列出来，然后逐个分析，这样你才能知道一个系统到底是大还是小。我想这是从常识我们大家都能够理解的。

18.再谈时间

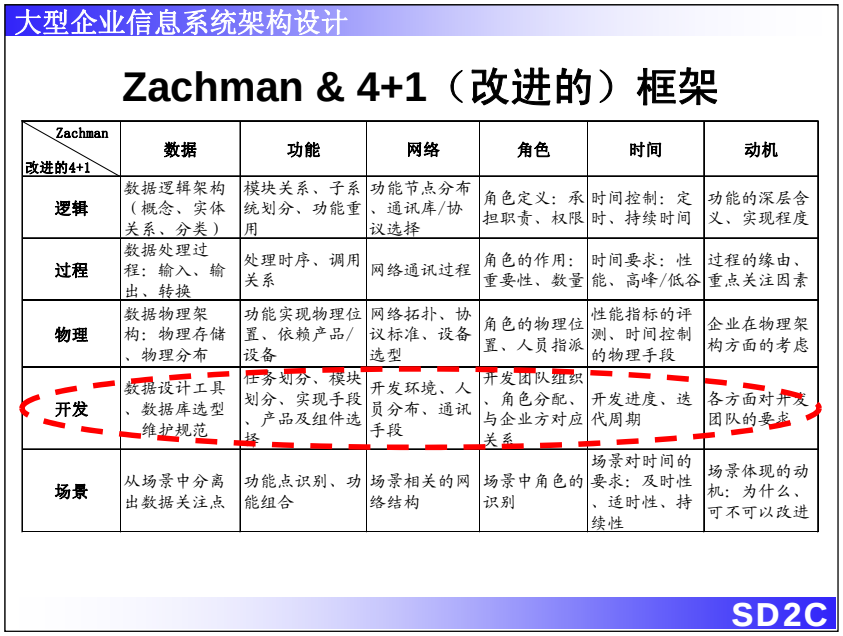


OK，那么我们怎么来做呢？我这里回来要再谈一下时间了，因为这一点是和我刚才说的软件架构的本质是密切相关的。也就是说：软件架构的本质驱动如果是时间的话，那么我们所谓的大和小，复杂和不复杂，也就可以投射到它对时间的影响上。对不对？大家想一想，我不再多做介绍。我们看一下，大家有听说过 Amdahl 定律吧？应该都知道，因为这个定律应该可以说是计算机系统领域的一个至关重要的一个定律。实际上我本人的理解，这不仅是计算机系统本质的定律，而是世间万事万物的一个重要的定律。因为它是跟时间有关系的。Amdahl 定律说得很简单，系统优化某部件所获得的系统性能的改善程度，取决于该部件被使用的频率，或所占总执行时间的比例。大家能理解这个概念吧？我想应该很简单。回到我们刚才所说的那个 WideFinder 的例子，有效代码一共只有 5-6 行，而且是一个顺序执行的循环，所以大家知道如果我想要提程序执行的升速度，我优化其中某一行代码，能起到多大作用呢？对不对，大家能理解吗？因为它是一个顺序执行的循环，如果我要把系统的执行速度提升一倍，就意味着它那个关键一行代码的速度要提升一倍，不然则做不到。同样的在一个大的系统当中，假设说一个三层结构，有页面，有中间应用服务器的数据库层，现在我们发现这个系统慢了，我们应该怎么办呢？当然要分析到底是数据库慢，还是应用服务器慢，还是什么东西慢。也就是说我们总是要找到那个对系统运行总时间影响最大的那个点，我们改善它，能够获得性能的改善是最大

的，大家能理解这个概念吧，非常简单的一个概念，这是 Amdahl 定律告诉我们的。

因为我本人认为架构的本质是时间，所以我把 Amdahl 定律转移到架构设计领域，总结出一个规律：对总体架构某一组成部分所做的设计改进产生的效果，取决于该组成部分所耗时间在总体架构中对应时间维度的占比。因为我们刚才说了，我们把时间分为 9 个维度，我们没法像 Amdahl 定律那样能够计算出来你这个东西改进能够提升多少时间，而是说要具体跟你是改进开发的时间，还是能够改进运行的时间，还是能够改进验证的时间；是改进及时性，还是能改进适时性，还是改进持久性。这里我就得到一个结论，架构设计的重点在于发现时间因素占比最大的方面，并对其进行重点设计（或改进）。我想大家能够理解，从这个推导的关系过来，实际上就是我这里一个重要的结论，因为后面大家会看到整个大型的信息系统的设计我不是太关注于它某个具体的环节涉及什么形态，而是把重点的精力用在发现到底谁会占的时间因素最大，我只要列出这个优先级对它进行重点设计，就可以从相当程度上保证我这个大型系统的设计没有重大问题，没有重大错误。

19.Zachman&4+1（改进的）框架



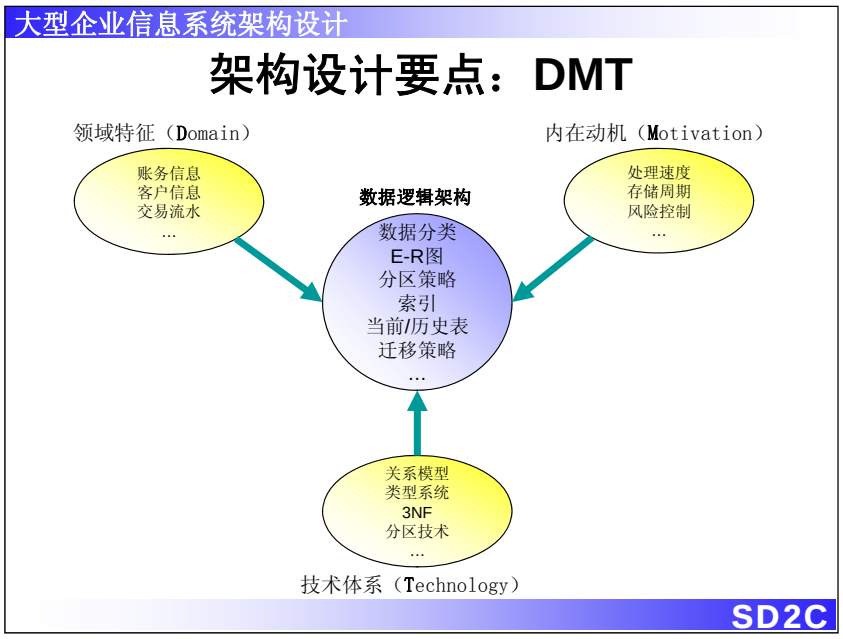
Zachman / 改进的4+1	数据	功能	网络	角色	时间	动机
逻辑	数据逻辑架构（概念、实体关系、分类）	模块关系、子系统划分、功能重用	功能节点分布、通讯库/协议选择	角色定义：承担职责、权限	时间控制：定时、持续时间	功能的深层含义、实现程度
过程	数据处理过程：输入、输出、转换	处理时序、调用关系	网络通讯过程	角色的作用：重要性、数量	时间要求：性能、高峰/低谷	过程的缘由、重点关注因素
物理	数据物理架构：物理存储、物理分布	功能实现物理位置、依赖产品/设备	网络拓扑、协议标准、设备选型	角色的物理位置、人员指派	性能指标的评测、时间控制的物理手段	企业在物理架构方面的考虑
开发	数据设计工具、数据库选型维护规范	任务划分、模块划分、实现手段、产品及组件选择	开发环境、人员分布、通讯手段	开发团队组织、角色分配、与企业方对应关系	开发进度、迭代周期	各方面对开发团队的要求
场景	从场景中分离出数据关注点	功能点识别、功能组合	场景相关的网络结构	场景中角色的识别	场景对时间的要求：及时性、适时性、持续性	场景体现的动机：为什么、可不可以改进

Ok，这就是我前面所讲的一切的内容最终产生的一个依归。我目前还没有给这个框架命名，我暂时就是把它叫做 Zachman & 4+1（改进的）框架。我本人创立的也好，或者设计的也好，发明的也好，或者胡思乱想出来框架也好，这是我做大型信息系统设计所要使用到的框架。这个框架很简单，大家一看就知道。我把 Zachman 横向的 6 个要素排在这，把 4+1 的本来是横向的 5 个要素把它竖过来排在这，好像是说我把这两个框架拼起来合到一个新的框架。抄袭也好，剽窃也好，盗用也好，我们先不管，等会我就会讲到。

好，简单讲一下，这里面的概念很容易理解。比如说，“数据的逻辑架构”，好理解吧；数“据的过程架构”，数据的处理过程；数据的物理架构；数据的开发架构；数据的场景架构，也就是说从数据的场景分析出数据的相应内容，这是因为符合 4+1 的方法，等会我再讲怎么个做法。每一个维度都是这个样子的，大家有兴趣的话可以看这些内容，这些内容只能是我写的了，但我也需要声明一下，这个框架没有那么成熟，像 Zachman 那样 20 多年，4+1 那样大概是 10 几年，我这个框架没几年,以后还要不断地完善，这里面的内容都是我平常日常总结出来的。这带来了一个问题：就是说你把这两个框架这样揉起来，人家里面本来还有一些底层细节，Zachman 框架，4+1 框架底下都有很多细节，你现在把这两个最宏观地维度揉起来的话，会出现一个问题，就是你整个这个框架的层面会非常高，它没有任何底层的细节了。我们大家都知道，架构

师被人们骂的最多一个地方就是，不能落地，空来空走。就是说上来你就给我们分析了这些乱七八糟，你都分析得头头是道，都很清楚，你最后能不能搞定啊，最后设计出来行不行啊。所以说，这样的框架必然有一个配合的层次。

20.架构设计要点：DMT



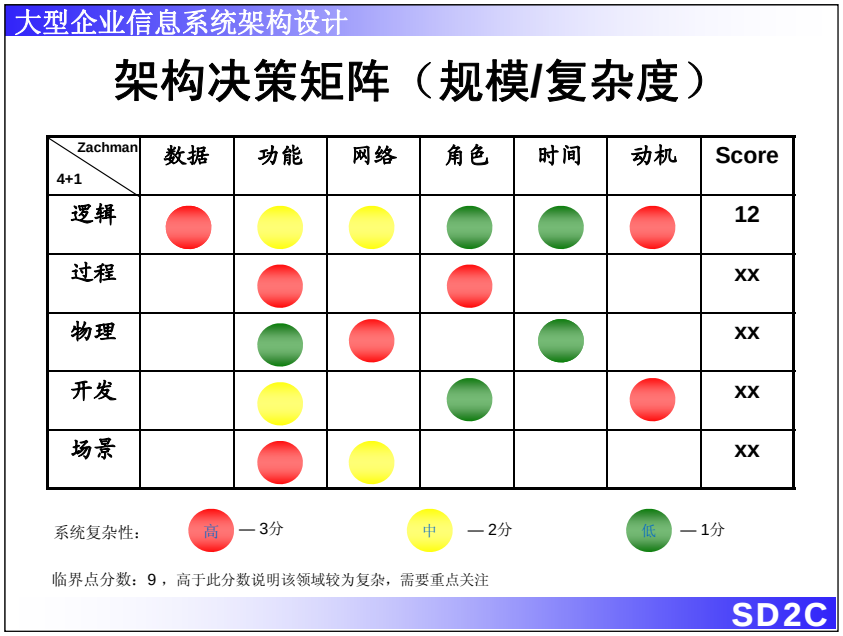
这个框架必然有一个配合的层次，就是里面每一个方块是什么，应该怎么做。这也是我总结出来的内容了，我暂时把它称为 DMT。因为本来我是想把这个框架命名为 DMT，后来一想这个名字太小，因为这实际上是指里面的每一个点，这个东西我还没有命名，我就快速讲一下。

Domain：领域特征，就是我们刚才说的你做什么样的系统，这个东西是有定式的，你别把一个互联网网站，一个共享，一个聊天室的架构说它能够适应银行的业务，不可能的，所以首先要用 domain。我这里举一个例子，就以其中的一个点，数据的逻辑架构来举例。大家知道吧，上面的每一个框，30 个也好，20 个也好，没关系，我举其中的一个例子，数据的逻辑架构。数据的逻辑架构从领域的角度来讲，你有什么样的数据，帐务的、客户的、交易流水——银行的业务。

然后，你的内在动机。你是数据的逻辑架构，你对数据的处理总有动机。你需要它的处理速度要快，你需要它的存储周期要有多长啊，包括你是对数据的风险要有控制，谁能够访问什么样的数据库，能够访问什么样的数据，是这样一个内容。

Ok，第三个维度，T，technology。这个维度是什么呢，是相应的技术手段，是为了解决这两个问题所可能存在的技术手段。比如说，数据，domain 领域模型现在我们普遍都知道，我们都用关系性数据库，那就是说，它解决这个手段，关系模型。然后有类型系统，跟关系模型一样，因为类型系统和关系模型两者之间没有什么本质关系。3NF，这是关系模型中进一步的发展，就是说我为了处理这种数据的复杂度，我有一个范式，好让它们数据之间有一个相应的规范。包括分区技术：数据库分区，大家都知道，这是处理大数据量，提升性能的一个重要的手段。我这个 DMT 的方法，什么意思？就是说每一个点都要有这三个维度考虑，最终得出你的数据架构是什么样子。数据的逻辑架构，数据的分类，画出它的 E-R 图，它有什么样的分区策略，在哪里建索引，要不要分当前表和历史表。大家可能都知道一些点，在某些 OLTP 的形态的系统中，当前表/历史表这种模式用得很多的，包括数据的迁移策略，因为你有存储周期的要求，这种 motivation 使你要定迁移策略,还有很多很多很多。这样的一个形态使你设计出来上面摸索出来的矩阵的 20 个维度当中的一个点，这么使我们避免了空来空走，分析的头头是道，最后做不出东西来这样一个形态。所以我所提出来的架构体系实际上是两个层面，一个层面就是总体的框架，另一个层面就是它内部的设计。

21.架构决策矩阵（规模/复杂度）

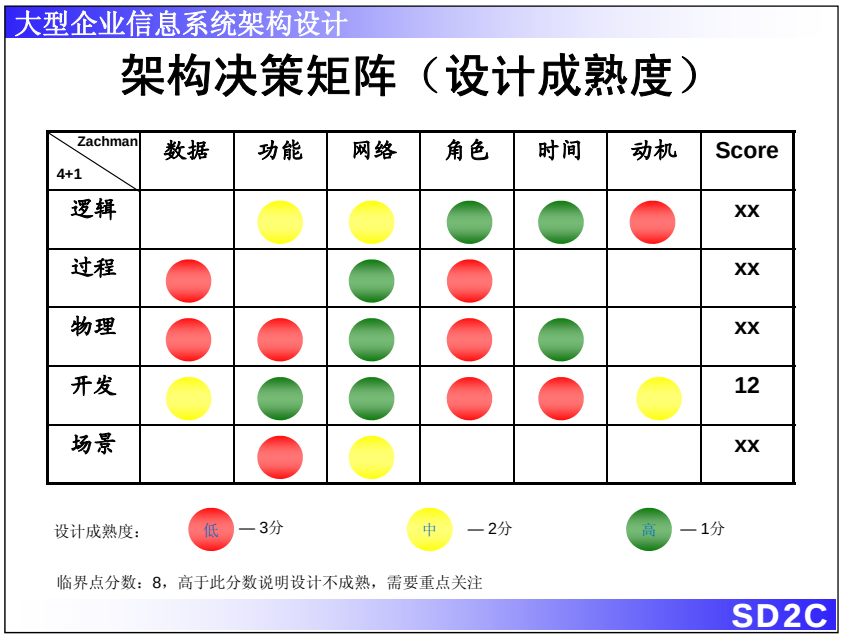


那么这里面我们就要说为什么要有这样的一个东西呢：就是这三个要点，为什么会出现呢？回到我们刚才所说的“大”的概念，因为我们这个信息系统大，所以我必须要知道哪些东西是大的这个因素。那么就是领域特征，账户特别多，客户特别多，这个就是大。内在动机，实际上对于处理的时间要求，实际上和我们所说的系统的复杂和大是一个维度，它是一回事。第三点就是技术体系，是我们用来解决“大”的这个手段。因为我们不能说，分析出来这个系统很大，但是我没招，没这个本事把它解决掉，那要你做架构师做什么呢？对不对。

所以，这样一个形态，我们就说当你有这样的一个方法论的时候你设计一个大型复杂的系统，你要从什么地方入手，我这里讲两个矩阵。第一，叫做架构决策矩阵中的规模复杂度矩阵。也就是说我把这两个维度列出来，然后我根据我刚才所说的 DMT 的方法，主要是 DM 这两个维度，我要看到这个系统中哪些点是最复杂的，因为一个系统并不一定是 20 个维度，或者说 30 个维度全都很复杂，我们刚才说了，架构设计的要点在哪呢？就是我们所说的：要把大和复杂的那些点找出来，优先排列，然后对它们再进行深入的分析。这样的一个手段就使你能够有序地来做这样的一个系统。比如说，如果说这个数据的逻辑架构很复杂；包括它的功能，网络的逻辑架构中等复杂；角色、时间这些呢，比较简单，所以我们可以给它来做一个评分的标准，比如说最复杂的 3 分，中等

复杂的 2 分，低复杂度的 1 分。那么这样的一个情况我们把它放到这里，本身也是可以看到的，这些红色的点就是我们要着力设计的。那么进一步而言，大家知道决策矩阵这个概念吧，decision matrix 这个概念，我们还可以来做加法，就说最终我通过这个加法来得到逻辑架构这部分是 12 分，这个复杂度就可能比较高了。也许因为还有很多空白的加起来还有 4 分，那么就是说我们要定一个标准，就是如果说有一个临界点的分数达到 9 分，就是说这个位置上达到 9 分，那我们就意识到，除去这些单点之外，整个体系内部逻辑架构是十分复杂的。那么我们就应该在这里多花功夫，慎重地请一个有经验的架构师来做这个事情。另外，其实纵向也可以做这个事情，我这里没画，画不下了。就说如果这个维度数值很高的话，就说数据架构很复杂，那么我们就请一位数据设计的高手来做数据的架构，或者说我们自己来在这上面多花功夫，这是我讲的规模复杂度的矩阵。

22.架构决策矩阵（设计成熟度）



下一个我们将设计成熟度的矩阵。这跟刚才那个正好是对应的，因为刚才我们看到一个 DMT 的方法，其中 T 是 technology，也就是用来解决复杂程度的

技术手段，那么这个我们就称为设计成熟度。也就是说，虽然我们有些领域可能很复杂，但是恰好我的这个产品或者说我现在拥有的技术，或者我的团队他们能力很强，恰好能把这个问题解决掉。如果是这样的话，那么这个东西的复杂程度或者复杂因素就会下降。所以我们一定要总结出来，比如说逻辑角色，我们有一个很好的逻辑系统，RBAC 的系统，非常完善，可以把角色的授权什么给解决掉，那么它就是一个很简单的，低分值的东西。所以呢，这个设计成熟度我讲这个分要反过来，越低的得 3 分，中等的得 2 分，高的得 1 分，这样的话，我们也可以得出来我们现在所拥有的所有的技术手段，包括产品，它的成熟度是怎样，能不能满足我们这个整体架构的需要。如果说恰好有一个环节复杂度高，而且成熟度低的情况，那就是对我们整个信息系统当中最危险，最需要重要设计，最有可能影响我们这个系统架构的质量的一个要点。所以说决策矩阵还有一个重要的作用，它可以合并，可以加权，假设说我们这不是 3，2，1 分吗，我们可以把这两个矩阵进行加权，（我就没有再做除了，）乘起来，得 9 分的那就是我刚才那个成熟度最高，复杂度最低的那个点，那你就多留心吧，甚至你请个高手来做吧。对吧。我今天所讲的这个呢，当然跟我的个人经历有关系，就是说都是站在一个总架构师的角度，一个全局的角度来看待这个系统的问题，（等会我会跟大家再说一下）。

23.实例分析：ROR

Ok。基本上来讲今天我所讲的主要的内容都差不多了，就是这样一个架构的框架，加上每一个点的 DMT 的方法，以及它衍生的一些应用，比如说架构的复杂度的矩阵，成熟度的矩阵，你还可以做很多很多的矩阵，因为它是一个框架，这个东西反正我自己也在发展。

下面我举一个简单的例子，也许这个还不能叫例子，什么意思呢？ROR，大家知道 ROR 很流行了这么多年，对于开发效率有很大的提升。那么有些人有时候就会问，那么我也想用 ROR 来做企业的信息系统啊，行不行啊，对不对？我刚才说过了，ROR 做企业信息系统先要问问你的甲方，要是他根本不同意，你别的就不用想了。但是我这里要说的是，我用我的这个框架来分析一下，来看看 ROR 第一它为什么好，第二是它适不适合做企业信息系统。内容我就不讲了，大家可以看到比如说数据的逻辑架构，大家都知道，ROR 没有什么特别复杂的数据逻辑架构，它最主要还是用于数据库。为什么呢？因为大家知道 ROR 的定位是什么，就在这反映出来的，它的逻辑架构的定位 Database Centric Web Application。也就是说，它就是说后台要操作数据库的，基于 web 的应用。所以呢，它的这个数据的逻辑架构其实很简单，它没有太多的什么分类信息，或者别的数据的模型在里头，就是关系模型。其他的也是一样，比如说它的功能架构呢，它就是典型的逻辑架构就是 MVC，包括后面引入的这种 REST，这样的一个形态，就是普通的 MVC 还有基于 REST 的这种形态来做的这种功能的逻辑架

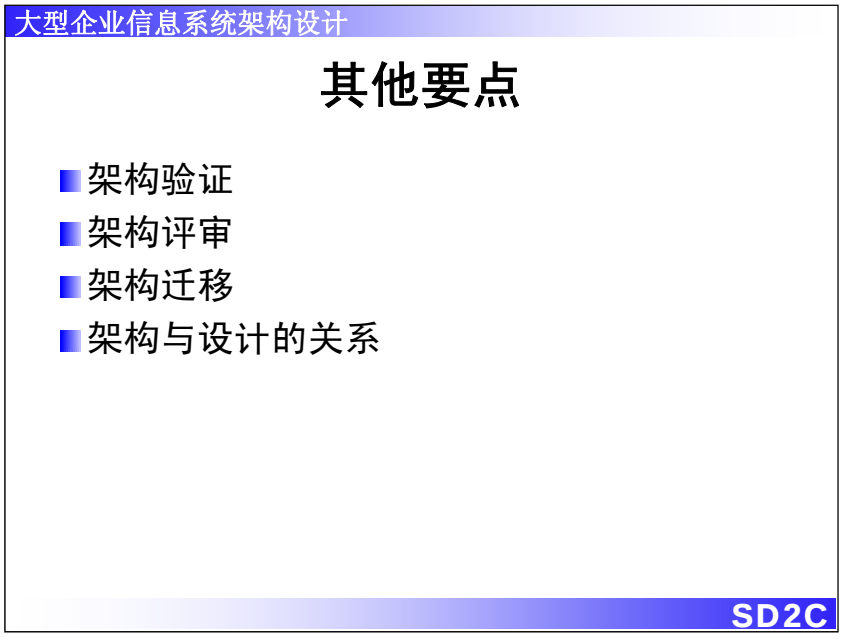
构。

所以整个这些内容我就不讲了，说白了就是一点，我们可以看到，所有的点上面都有 Rails 的这些特色。比如典型的： Rails 它有 performance 来做这个性能的验证，包括有这种 Scaffolding，大家都知道，这是 Rails 一开始最吸引人的地方，因为它提升了效率。为什么呢？因为它的核心的思想就是 DRY 和 COC。这是它的 motivation。包括我们知道场景的架构呢，Rails 没有一个方法论，像 usecase 那样帮助你来分析业务逻辑，但是它有个好东西是什么呢？它有一个 5 分钟做 blog 的视频，然后又出了好几本书，大家都去看了、懂了，说我应该怎么去用这个东西。所以我这里讲的一个例子就是大家可以看到，从架构的角度来讲，用 Rails 能不能来做企业信息系统，实际上是个伪命题，不存在这样的命题。因为你没有说你的企业信息系统的架构要考虑哪些因素，要做成什么样子。你必须要有一个 target，一个目标，然后你就可以去从架构的角度去说，我这个企业信息系统架构重要的点都被 Rails 所覆盖了，它都能帮我解决，那么我用 Rails 来做企业信息系统是可以的。

我举个简单的例子，大家看到我这里划两个杠，说明什么呢？Rails 在网络和角色这两个领域几乎没有任何的作为，就是它的这种原生的 Rails 框架，没有处理这些东西。因为我们说了它是一个 Database Centric Web Application，它不会说是有任何的比如说我的中心的机构是布什么样的应用服务器，然后这个分支机构要布什么样的应用服务器，然后数据又要去做怎么样的同步，怎么样去做本地缓存等等这样的一些机制，它不能去想。角色也是一样，因为做 web 的东西，大家都知道，像购物车网站那样，你登录了，你就是合法用户，没登录就是 anonymous，没有分析说像我们这样复杂的一个角色的架构，做过一个 BPM 的系统，有好几十种角色，有什么发起人，审批人，最终审批人，什么 checked，什么这个那个的都没有。所以，如果你要是做一个系统在这两个方面要求很高的话，那么无非两个可能性，第一，不用 ROR；第二，那你自己做：你把这两个维度能够给它补充上，那 ROR 还是可以用的，那就意味着说你开发的代价可能就会很大。我从这样的一个框架的结构，结合 ROR，可以得出来结论就是说，你从架构的角度去考虑 ROR 能不能做你的系统，需要分析这些问题。大家可以回去看一下，比如说单纯的 Zachman 方法，或者单纯

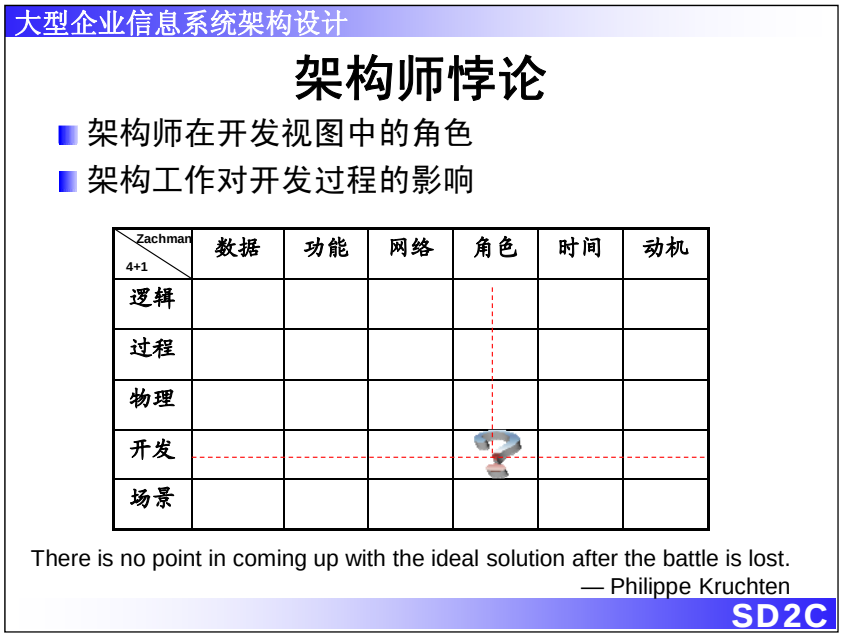
的 4+1 的方法，能不能够做到这样的评判？应该说是要差一些。这也是我举一个实例来说明，我的这样一个架构体系的作用。我给大家简单说一下，就是我刚才说的，如果你决定用 ROR 的话，你就要看那些空白的东西你需不需要，如果说我需要，我现在需要一个 RBAC 授权的机制，那么怎么办呢？无非就是去找，去看看有没有。好，有一个插件，叫做 active-rbac（01：11：48），它也是走的一个 Rails 的 active 的一个系列，它能够很好的和 Rails 结合起来，能够完成角色的授权机制，这样 ok，我放心了，我的角色的问题 ROR 也可以解决。不是原生的 ROR 的机制，但是第三方的插件。但是我告诉大家，这个第三方插件 3 年都没有更新了，大家要用那你们自己去决定。这样的形态就是我们做总体架构设计的核心的一个出发点。

24.其他要点



Ok，基本上的内容都讲完了。最后再说一下，架构设计这个领域太大了，我今天说的这些个内容不说是沧海一粟吧，顶多也就是个管中窥豹。架构的验证今天没说，评审、迁移，还有架构与设计的关系都没有讲，这些领域大家都可以自己去看一下。

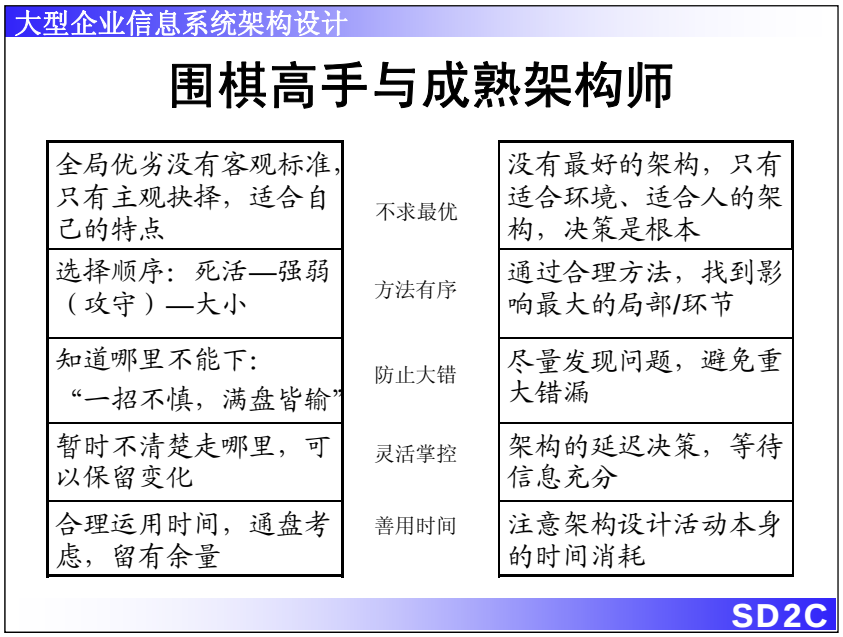
25.架构师悖论



我这里讲一个简单的东西，就是我刚才说的架构师为什么老被骂呢？老说不行呢？这个东西是个本质的问题。不是说架构师不行，架构师的工作本身就带有很强的悖论的性质。大家知道逻辑学或者语义学的悖论吧，所谓的对角线原理，就是说架构设计本身它有一个很重要的问题，它有一个自相关性，就是说架构师他的工作本身应该是在开发这个过程当中的一部分，对不对？可是我们刚才又说了，架构师又必须去设计这个整个的开发体系，所以他在这个里面还要有一个自己的位置，也就说在开发体系的角色当中有一个是他自己——架构师。那在这种情况下问题就很严重，就说我可以去做一个架构很合理的设计，设计得很完美了，最后发现呢，我这个架构其实最重要的时间因素就是在于它的开发过程会很长，等到你分析出来你发现，你得到这个结论花了 3 个月，你自己占用的时间就是你开发体系中最长的一块时间消耗。你不被人骂能行吗？肯定别人都会骂你。你都知道开发这个任务最重，然后你说你们现在都别动，我一个人先做架构设计，先做 3 个月，就是这么的一个问题。所以这里面我就讲说，架构师，因为我们说做架构设计，因为刚才我们都是讲的架构设计。

最后我们来讲讲怎么做架构师，再次引用 Philippe Kruchten 的一句话：There is no point in coming up with the ideal solution after the battle is lost. 等时间都过了，时过境迁，你的系统都已经延迟了，再有完美的架构也没有用了，这就是我们讲的架构师的悖论。

26.围棋高手与成熟架构师



为什么讲悖论呢？大家要重视这个悖论。那怎么办呢？这是我最后的一张 ppt 了。大家听我在这里唠叨可能也有点烦了，我就举一个例子，就是围棋高手和成熟架构师之间的对比。为什么突然冒出一个围棋高手，其实我还有好几个 ppt 是讲围棋和架构设计的关系，时间太紧来不及讲了，砍掉了。

总结五点：第一，不求最优：假设一个围棋的高手在一个局面下，你下每一步棋都是最好的，这是不可能的。因为局面太复杂了，第一没有客观的评判标准，第二你本人适合下什么样的棋呢，你适合于攻杀呢，还是适合于收官子呢，对吧？这是有区别的。所以说，在每一个局面下，不仅仅是要看一个局面本身的状态，还要适合于自己。架构师也是这样。我们刚才说了，没有最好的架构，只有最合适环境，合适人的架构。也就是说，企业你目前干这些事，你

手头就是这样的一个开发团队，那在这种情况下，适合于他们的把这个架构设计出来的决策才是真的。就是说我最终要定下来了，我有 5 个架构的选择，我就定这一个。定这一个的理由已经不是那么客观了，而是普遍的有主观意识。大家可能看过 Martin Fowler 写的《PoEAA》，一上来就提到，普遍业界现在认为架构设计是偏主观领域，而不是客观。

第二点，还是我们刚才所说的“思而不学则殆”，你说不求最优，随便设计，主观嘛，我拍脑袋说，还是不行。还是要有方法论。围棋的高手怎么来做呢？看这个局面下那个最重要。我下一步棋，下完了整个局面崩溃了，为什么？后面会讲到，因为死活出问题了，自己的棋死了，或者说强弱，攻守运转，最其次才是说大小。如果说没有强弱，没有死活的问题，我们下一招棋占大场，占的目数比较多。围棋大家不关心就算了，方法有序就是这个特点，就是我所介绍的这个方法论的一个原始途径。就是说我们要通过一个有序的方法，找到这个刚才我们说的那个方法论的核心。你先找到局部影响最大的那两块，你先把它们做好。就跟围棋一样，死活都没搞清楚——有一个领域对这个系统至关重要，脑子里没有印象，那你还做什么呢？所以必须要有那样一个举证，保证你来做到这些。

第三就是我们说的防止大错。就说在这两点之上，围棋的高手还在选择，因为这两点还不能保证我最终这招棋下在哪。那我怎么办呢？还有一个反过来的想法。我们经常说，围棋高手他不是知道每一步都是最优的，但他知道这个局面下哪一步最差，不能下，下肯定就完蛋。为什么？一招不慎，满盘皆输。我们刚才所说的那个方法论也是这样。就是尽量地发现问题，避免一个重大的错漏。刚才已经说过了，要用这个矩阵，矩阵的最大的作用是什么呢？就是防止你漏掉系统架构设计中的一个重要点。什么问题可以说都比漏掉东西强：系统要上线了，要实施了，突然发现有一个地方我连想都没想过，结果它对这个系统还挺重要，肯定完蛋。这块我刚才说了，有一个像架构评审这样一个内容我都没讲，实际上我们可以看到，有人经常说软件开发是最难的，是最复杂的。我这里也说一句，我现在分析了这么领域，我觉得软件开发是最容易的，因为它什么东西，好多东西还能别人帮助你，好多东西还能后悔，好多东西还能随时跟踪随时检验。因为刚才我还有好多内容都还没讲，真的是这样，很容易。

你盖一个楼，楼塌了——上海那个楼，如果里面住着人呢？所以我们经常说软件困难是因为好多起搏器啊，航天飞机啊，都有软件了，可是你别忘了，软件只是它的一个部分，你还有好多可以验证可以评审的手段。你说一个围棋选手，他下到这要抉择哪一步的时候，说，诶，你们帮我出出主意。这是不行的，违反规则的。架构可以评审啊，你不行你找别人帮你啊，对吧。找 10 个人拿来说你这个架构哪里不好，好像挺丢面子的，实际上架构的评审很重要。你水平不够，你找人帮你看啊，看你哪儿会有问题，对不对？

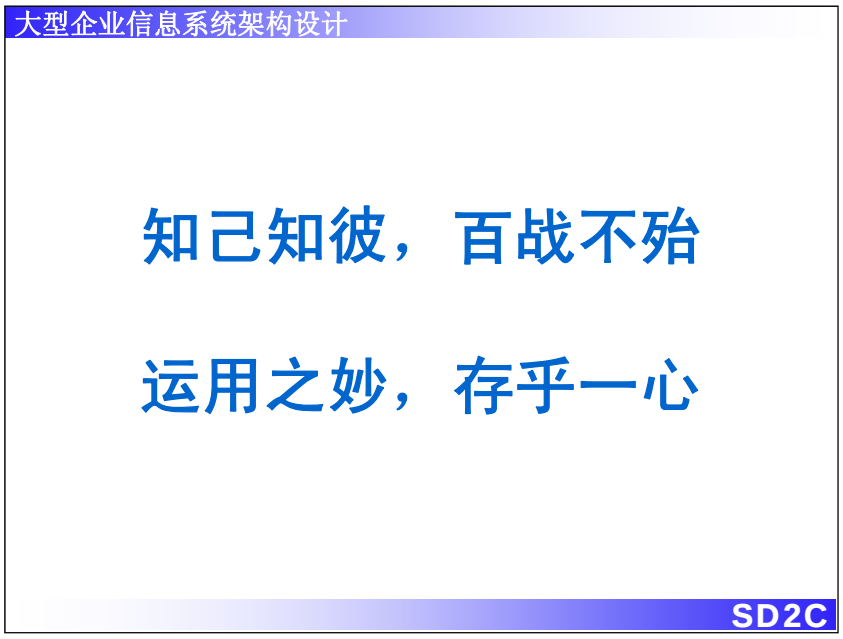
（啊，讲完了，之所以讲这么长是不希望给大家留提问的时间。讲完了就完了，这是我的风格。）

好，最后还有两个。灵活掌控。不多说了，高手知道保留变化。这儿我不知怎么下，先不下，先下别的地方。架构设计有一个重要的原则大家知道吗？延迟决策。延迟决策很重要，就是说，如果我要等待信息充分我才知道这个架构怎么设计的话，那么必要的情况下我们可以延迟。

但是延迟不是无限制的，也就是我们最后一点要讲的，就是要善用时间。我继续唠叨两句。大家可能不知道围棋的读秒，大家知道吗：你不能无休止的下下去，我给你 1 个小时的时间。怎么办呢？下到最后 5 分钟，我要半分钟就逼你下一步棋。大家有下过一种读秒棋么？网络的围棋都有。（我来调查一下…，一个都没有啊。）读秒，怎么读？告诉你，半分钟一步是这么读法。10，20，25，26，27，28，知道吧，它在催促，因为 30 秒要下一步，它要催促你在那个时间一定要下，你如果真正体会过你们就知道那个时候人真是要崩溃了，它催你到 28 还不下的话，那你就输了。为什么呢？因为从手抓到棋子到放到盘大概需要 2 秒钟的时间。如果 28 你再不去抓子，再不下的话，那么等你到落下去之前就到 30 了。在被时间催促的情况下，人的精神状况是极度差，在这种情况下，大家都知道什么古力啊，李昌镐啊，常昊啊这些高手，他们这些人不用说了，世界顶级的架构师，他们在读秒的时候同样会崩溃，同样会犯很多低级的很多业余选手都能够看得出来的错误。为什么？没时间了。软件不是这样吗？先说软件开发本身，眼看离上线还有 15 天，还有 20%的功能还没开发完，拼命敲代码啊，你想想你那个时候代码的质量，肯定是错误百出，架构师也是这样。

我给你 1 个月时间做架构，好像很充裕的，花了 20 天去研究 REST 去怎么设计，怎么用 Reverse Ajax 做聊天系统啊，来提升系统性能啊，你有没有想过在这个 20 个维度当中，这一点是不是最重要啊。你花了 25 天全去搞这些事，你引入了好多新技术，你做了好多新架构，到最后的 5 天里你发现这 20 个维度里还有 15 个你根本没有考虑过。那最后 5 天你就把那些维度就那样了。那个时候一定是你错误最多的时候。所以我们要讲有序的方法论就是这样，就是说我们说架构师的悖论：就是要注意架构设计活动本身的时间消耗，因为我们还有好多手段评审啊，验证啊，都可以去做。所以我们说，不求最优，方法有序，防止大错，灵活掌握，善用时间。做到这些应该说还是可以做到一个比较好的架构师。我本人反正水平也就是这么一回事，也就是一些经验之谈。

27.结束语



最后讲两句，（我是下决心要把 1 个半小时占用完，）最后导出两句兵法的内容，就是总结一下刚才我说的这些内容。“知己知彼，百战不殆”大家都知道——孙子兵法。我刚才所说的，架构设计这套理论，其实核心的地方在这，知己知彼。知己是什么呢？架构成熟度的那个矩阵。知彼是什么呢？架构复杂度的那个矩阵。你总得知道系统是什么样，你自己有几把刷子，然后不就 ok 了吗？百战不殆。殆不殆我不管，反正就这个方法论。“运用之妙，存乎一心”，

这个大家知道吧，一个普遍的兵法，围棋上也经常使用，这是岳飞讲的。岳飞讲的前半句话还有，我记不住了，其实类似的意思是你首先要排兵布阵，把这个阵势弄好，方法论弄好，没问题，但最终的决定者还是你自己，还是你的心，去主观判断去定，说这件事就这么去办了，后面可以有监控，可以有评审，还是可以把这个事做好。

OK，内容讲完了，感谢大家听了我这么半天的唠叨，谢谢大家！