37讲过程：规模与协作——规模化的过程方法

在学校时，你学习编程，写写课程作业的代码，但你想过真正的行业里，公司中的规模化开发方式是怎样的吗？在上一篇《核心：安全与效率》的文中，你应该还记得我讲的那个电站的例子，那么编写课程作业的代码就像搭建的 “酒精灯电站”，而工业级的规模化开发才是建设 “真实电站” 的方式。

工业级规模化的程序系统开发包括了一系列的过程，而这一系列过程的起点是：需求。

需求与调度

需求，有时会有很多不同的表达形式，包括：客户的诉求、用户的请求、老板的要求，但这些不同的表达形式，不一定是真正的需求。

客户的诉求，更多来自传统甲、乙方关系的场景，在软件工程过程中有一个子过程——需求工程——去对客户的诉求进行分析和提炼，并转化为需求文档。用户的请求，更多来自互联网 toC 的场景，通过洞察大量用户请求中的共性去提炼并转化为真正的产品需求。老板的要求，更多是因为老板也可能是你的产品用户之一，但这个用户的特殊之处在于，他为这个产品买单。所以，他的要求无论合理与否都能很容易地变成需要开发的需求。

正因为需求的来源多，表达形式也多，因而真实情况是 “需求” 似乎总是源源不绝，但是真正的需求往往隐藏在这些诉求、请求与要求的表象之下。这是关于 “需求” 的第一个困难点。如果我们总是能找出真正的需求，那么也许需求也就没那么多了。但现实往往是我们不能，那么需求过载的场景就会常常发生。

这时，你就会面临第二个困难，如何对过多的需求进行排序？

为什么需要排序？我们进行需求排序的原因是，在有限的资源下我们想要达到如下目标：

- 最大化用户、客户和老板的整体满意度；
- 最大化价值与产出，让最多的资源投入到最有价值的需求上。

只有当用户需求被快速地满足时，他们才会感到满意。但在有限资源限制的条件下，我们不可能让所有用户的需求都能被快速

满足。面对这样的矛盾，我们也许可以学习、借鉴下**操作系统的资源调度策略**。

我用了好多年的 Mac 和 iPhone，发现它们相对同等资源配置的 Windows 和 Android 机，在满足用户使用的响应性方面要好得多，特别是在使用了几年之后，这种差距会更明显。

在同等硬件资源配置的情况下，Mac 和 iPhone 表现得更好，只可能是其操作系统的资源调度策略实现更优秀。通常，操作系统需要同时执行多个应用程序时，它的执行调度策略就会在多个应用程序间不停切换，有如下几种常见的调度策略：

1. **先来**先执行
2. 执行起来**最快**的先执行
3. 占用资源**最少**的先执行
4. 释放资源**最多**的先执行
5. **高优先级**的先执行

当资源充足，只用策略 1 就足够了，但更多情形下需要综合后 4 种策略。比如：老板的要求天生自带高优先级，需要先执行；而一些小需求，优先级不高，但执行快，占用资源少，随着它们排队的等待时间延长，优先级可以逐步提升，以免消耗完用户的等待耐心，形成负面评价。

当用户同时运行的应用程序确实太多时，操作系统发现资源无论如何调度都不够了，它有一个选项是提供了资源消耗的监视器，来提示用户主动停掉一些同时运行的应用，而最后的底线是操作系统主动杀掉一些应用程序以释放资源，以保障系统还能正常地运转下去。那么我们在调度需求时，是否也能以透明的资源消耗监视提示用户主动控制需求或选择 “杀” 掉需求，然后还不降低用户的满意度呢?

需求调度，可以像操作系统一样运转，形成一个规模化的需求调度体系，并通过多种调度策略来平衡需求的响应性和投入产出的价值最大化。

设计与开发

紧接需求之后的过程是：设计与开发。

成为程序员后，你一开始可能会习惯于一个人完成系统开发，自己做架构设计、技术选型、代码调测，最后发布上线，但这只适合代码量在一定范围内的系统开发模式。在一定范围内，你可以实现从头到尾“一条龙”做完，并对系统的每一处细节都了如指掌，但当系统规模变大后，需要更多的人共同参与时，整个设计和开发的模式就完全不一样了。

一定规模的系统，下面又会划分子系统，子系统又可能由多个服务构成，而每个服务又有多个模块，每个模块包含多个对象。比如，我现在所在团队负责的产品，就由数个系统、十数个子系统、上百个服务构成，这样规模的系统就不太可能光靠一个人来设计的，而是在不同的层次上都由不同的人来共同参与设计并开发完成的。

规模化的设计思路，一方面是自顶向下去完成顶层设计。顶层设计主要做两件事：

- 一是去建立系统的边界。系统提供什么？不提供什么？以怎样的形式提供?
- 二是去划定系统的区域。也就是系统的层次与划分，以及它们之间的通信路径。

今年世界杯期间，读到万维钢一些关于 “足球与系统” 的文章，感慨原来系统的顶层设计和足球运动十分类似。按文中所说，足球的科学踢法是：“球员必须建立 ‘区域（zone）’ 的观念，每个球员都有一个自己的专属区域”，通过区域的跑位来形成多样化的传球路线配合。

而系统的区域划分，也是为了让系统内部各部分之间的耦合降低，从而让开发人员在属于自己的区域内更自由地发挥。而在区域内的 “控球”“传球” 与 “跑位”，就更多属于开发人员个体能力的发挥，这个过程中区域的大小、边界都可能发生变化，进而导致区域之间的通信路径也跟随变化。这样的变化，就属于自底向上的演化过程。

所以，**规模化设计思路的另一面，就是要让系统具备自底向上的演化机能**。因为，自顶向下的设计是前瞻性的设计，但没有人能做到完美的前瞻性设计；而自底向上的演化机能，是后验性的反应，它能调整修复前瞻性设计中可能的盲点缺陷。

记得，我曾经看过一个视频名字大概是 “梅西的十大不可思议进球”，视频里的每一个进球都体现了梅西作为超级明星球员的价值，而在前面提及的万维钢的文章中，有一个核心观点：“普通的团队指望明星，最厉害的球队依靠系统”。其实二者并不矛盾，好的系统不仅依靠 “明星” 级的前瞻顶层设计，也指望 “明星” 级的底层演化突破能力。

所以，一个规模化的系统既要靠前瞻的设计视野，也依赖后验的演化机能，这样才可能将前瞻蓝图变成美好现实。

测试与运维

完成了设计与开发之后，系统将进入测试，最后发布上线进入运行与维护阶段。

在前面需求、设计与开发阶段的规模化过程中，都存在一个刚性扩展的问题，也就是说，如果提出的需求数量扩大一倍，那么需要去对接、分析和提炼需求的人员理论上也要扩大一倍；如果提炼出的需要进入开发的需求也翻倍，相应开发人员只增长一倍那已经算是理想情况了，这说明系统的正交与解耦性做得相当完美了，所有的开发都能并行工作，不产生沟通协调的消耗。

但真实的情况不会那么完美，需求的产生与爆发很可能是一种脉冲式的，而企业一般会维持满足需求平均数量的开发人员，当需求进入脉冲高峰时，开发资源总是不够，然后就会过载，进入疯狂加班模式。

开发完成后，进入测试与线上运维的循环阶段，这个阶段与前面阶段的不同之处在于：需求提炼、设计开发基本都只能由人来完成，而测试、运维的很多步骤可以通过制作工具来完成自动化过程。所以，这个阶段随着规模化过程实际可以是一个柔性扩展的阶段。

但它从来不是一开始就是柔性的，因为制作工具也有一个成本考虑。比如，在我做的这个系统发展的早期，系统架构简单、部署规模也很小，基本所有的测试与运维工作都是通过人肉来完成的，这样的效率也不算低，唯一的问题是对测试人员而言，大量的工作都是低水平的重复，不利于个人成长。

随着后来的业务快速增长，系统增长越过某个规模临界点，这时人肉负载基本饱和，效率已没法提升，制作工具是唯一的出路。你可能或多或少都知道一些现代化的工业流水线，而在软件开发过程中，“测试与运维” 的运转体系是最可能接近工业流水线方式的。

因此，以测试为例进行规模化的最佳方式，就是打造一条 “测试机器” 流水线，而我在《转化：能力与输出》一文中写到了关于打造 “机器” 的三个核心点，这里再强调一次：

- 流程规则
- 工具系统
- 规范共识

围绕这三个核心点，我们再来看看 “测试机器” 如何打造?

从开发提测，机器自动下载提测版本分支代码，进行构建编译打包，实施代码规范性检查测试，通过后发布测试环境，进行分层次的各类自动化专项测试。如：用户终端层、通信协议层、服务接口层、数据存储层的各项测试，全部通过后，生成相应的测试报告，进入下一步发布流程。这就是测试体系的“流程”，而“规则”就是其中定义的各种测试项检查约束。

上述流程中涉及的“工具系统”包括：代码规范检查工具、终端 UI 的自动化测试工具、通信协议与服务端接口调用的模拟工具、数据一致性校验工具、测试报告生成工具、测试 Bug 统计分析与收敛趋势等可视化展现工具，等等。

最后，“规范共识” 是整个团队对这个流程环节、里面具体规则的定义以及 Bug 分类等方面达成的共识，这决定了这台 “测试机器” 运转的协调性与效率。

测试通过后，发布到线上就进入了运维阶段，行业里已经有大量的关于 DevOps 的分享内容，而它的本质也就是打造了一台“运维机器” 流水线，这和我上面描述的 “测试机器” 运转类同，只是有不同的规范共识、流程规则和工具系统，便不再赘述了。

到了规模化的测试与运维阶段，看一个团队的水平，就是看这台 “机器” 的制作水准与运转效率。

在程序系统的开发过程中，当系统的大小和复杂度到了一定的规模临界点，就会发生从量到质的转变，规模不同，相应的需求调度、设计开发、测试运维的过程也都不同了。

量级变了，逻辑就不一样了。

每一个具备一定规模的组织都有对应的规模化工程过程，而且这个过程的形成受公司文化、团队构成、组织架构，甚至业务特性共同决定。那你所在组织的规模化过程是怎样的？这个过程系统如何运作的？欢迎你在留言区分享。

精选留言

godtrue

普通的团队指望明星，最厉害的球队依靠系统。

团队中个体越强当然越好，不过要各司其职具有协作精神，否则产生的摩擦就会更大。不过个体的强大往往会带来更多的自主空间的诉求，不利于管理了！

2018-11-06 21:27

作者回复

水不够深，鱼都大了，也有问题，彼此的生存空间都受到了挤压

2018-11-07 20:23

third

规模化，长期来看，就是降低成本的系统
用最少的资源，达成最大的价值
需求与调度，真实的需求，同时把需求排序

设计与开发，顶层设计，划定边界，区域和通信路径

测试与运维，善于利用自动化工具

2018-10-28 00:54

艾尔欧唯伊

我们项目的自动化测试用例都是研发写的。。。。项目的功能点，文档全都是研发给梳理。。。最后测试反馈一个表格叫我们写上个用例。。。

需求更简单了，找个对标的产品，抠抠图，写个抽象得不能再抽象的文字描述，然后叫研发先做出来再说。。。

2018-10-27 09:26

作者回复

，那只用研发就够了，还要测试和产品干哈

2018-10-28 12:32

ComputerGeek

请问老师有没有推荐的“测试机器” 开源工具。

2018-10-31 12:16

作者回复

测试领域我不够专业，建议看另一个测试专栏，详尽的覆盖了我文中提到的关于测试机器的内容

2018-10-31 19:12