摘要：可能出乎很多人意料之外的一个事实是，微服务很多核心理念其实在半个世纪前的一篇文章中就被阐述过了，而且这篇文章中的很多论点在软件开发飞速发展的这半个世纪中竟然一再被验证，这就是康威定律。

## 概述

可能出乎很多人意料之外的一个事实是，微服务很多核心理念其实在半个世纪前的一篇文章中就被阐述过了，而且这篇文章中的很多论点在软件开发飞速发展的这半个世纪中竟然一再被验证，这就是康威定律(Conway's Law)。

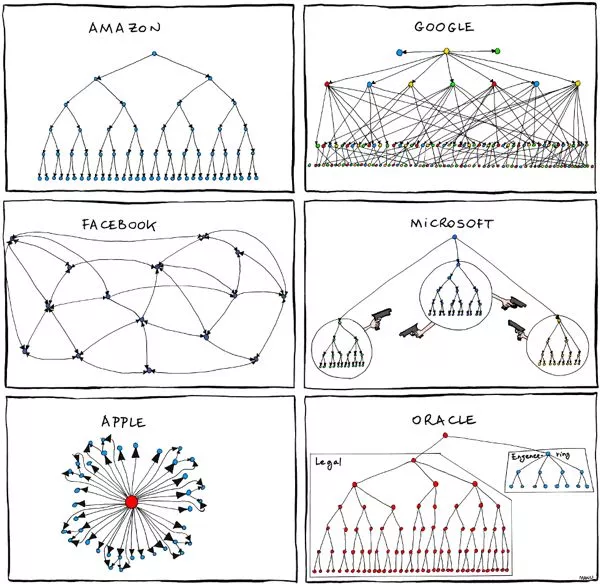


在康威的这篇文章中，最有名的一句话就是：

Organizations which design systems are constrained to produce designs which are copies of the communication structures of these organizations. - Melvin Conway(1967)

意思是：

设计系统的组织，其产生的设计等同于组织之内、组织之间的沟通结构。看看下面的图片，再想想Apple的产品、微软的产品设计，就能形象生动的理解这句话。



用通俗的说法就是：组织形式等同系统设计。

## 康威定律详细介绍

一些核心观点，如下：

### 第一定律（人是复杂社会动物）

组织的沟通和系统设计之间的紧密联系，在很多别的领域有类似的阐述。对于复杂的系统，聊设计就离不开聊人与人的沟通，解决好人与人的沟通问题，才能有一个好的系统设计。相信几乎每个程序员都读过的《人月神话》（1975年，感觉都是老古董了，经典的就是经得起时间考验）里面许多观点都和这句话有异曲同工之妙。

比如《人月神话》中最著名的一句话就是：

Adding man power to alate software project make sit later—Fred Brooks,(1975)

Boss们都听到了吗？为了赶进度加程序员就像用水去灭油锅里的火一样（无奈大家还是前赴后继）。

为什么？人月神话也给出了很简洁的答案：沟通成本=n(n-1)/2，沟通成本随着项目或者组织的人员增加呈指数级增长。是的，项目管理这个算法的复杂度是O(n^2)。举个例子

5个人的项目组，需要沟通的渠道是5\*(5–1)/2=10

15个人的项目组，需要沟通的渠道是15\*(15–1)/2=105

50个人的项目组，需要沟通的渠道是50\*(50–1)/2=1,225

150个人的项目组，需要沟通的渠道是150\*(150–1)/2=11,175

所以知道为什么互联网创业公司都这么小了吧，必须小啊，不然等CEO和所有人讲一遍创业的想法后，风投的钱都烧完了。

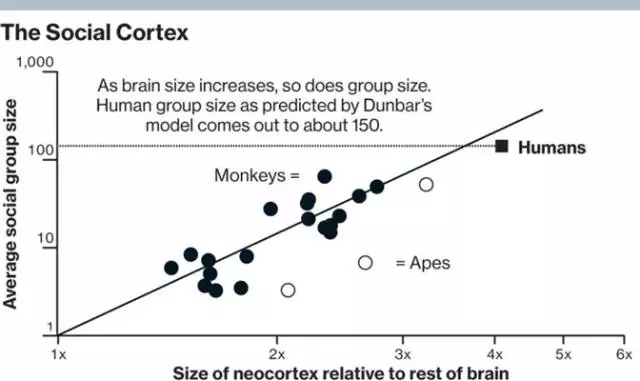
举一个非常有意思的理论，叫“Dunbar Number”，这是一个叫Dunbar生物学家在1992年最早提出来的。最初，他发现灵长类的大脑容量和其对应的族群大小有一定关联，进而推断出人类的大脑能维系的关系的一些有趣估计。举例来说

亲密（intimate）朋友:5

信任（trusted）朋友:15

酒肉（close）朋友:35

照面（casual）朋友:150



是不是和上面的沟通成本的数字很貌似有关联？是的，我们的大脑智力只能支持我们维系这么多的关系。（大家都知道这不是程序猿擅长的领域，在开发团队里，这个值应该更小，估计和猿差不多-\_-凸）

沟通的问题，会带来系统设计的问题，进而影响整个系统的开发效率和最终产品结果。

### 第二定律（一口气吃不成胖子，先搞定能搞定的）

Eric Hollnagel（埃里克·霍纳格尔）是敏捷开发社区的泰斗之一，在他《Efficiency-Effectiveness Trade-Offs》一书中解释了类似的论点。

Problem too complicated? Ignore details. Not enough resources? Give up features.

--Eric Hollnagel(2009)

系统越做越复杂，功能越来越多，外部市场的竞争越来越剧烈，投资人的期待越来越高。但人的智力是有上限的，即使再牛逼的人，融到钱再多也不一定招到足够多合适的人。对于一个巨复杂的系统，我们永远无法考虑周全。Eric认为，这个时候最好的解决办法竟然是——“破罐子破摔”。

其实我们在日常开发中也经常碰到。产品经理的需求太复杂了？适当忽略一些细节，先抓主线。产品经理的需求太多了？放弃一些功能。

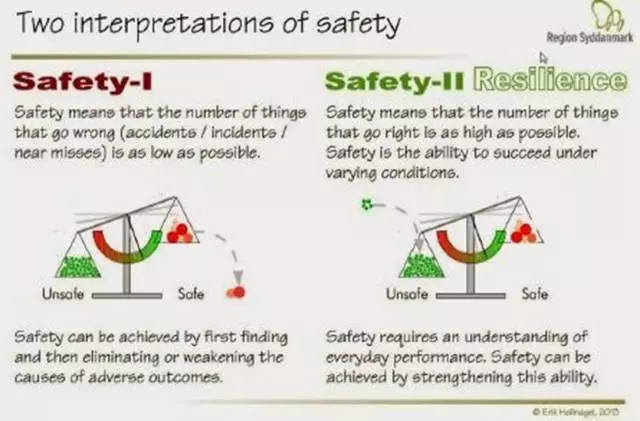
据说Eric被一家航空公司请去做安全咨询顾问，复杂保证飞机飞行系统的稳定性和安全性。Eric认为做到安全有两种方式：

常规的安全指的是尽可能多的发现并消除错误的部分，达到绝对安全，这是理想。

另一种则是弹性安全，即使发生错误，只要及时恢复，也能正常工作，这是现实。

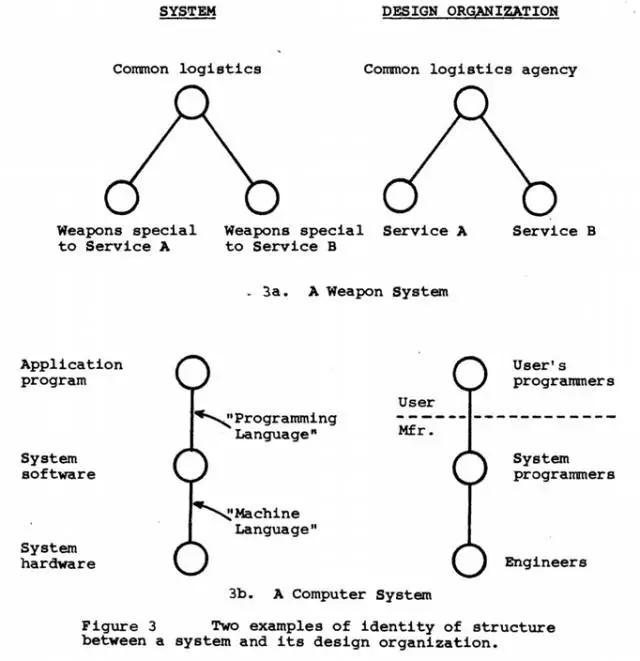
对于飞机这样的复杂系统，再牛逼的人也无法考虑到漏洞的方方面面，所以Eric建议放弃打造完美系统的想法，而是通过不断的试飞，发现问题，确保问题发生时，系统能自动复原即可，而不追求飞行系统的绝对正确和安全。

下面的图很好的解释了这个过程：

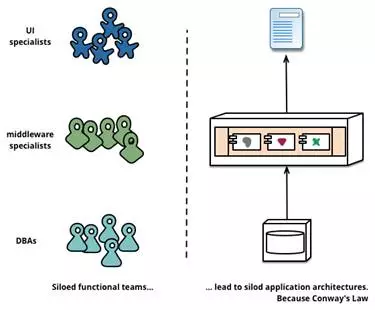


这不就是持续集成和敏捷开发吗？的确就是。另一方面，这和互联网公司维护的分布式系统的弹性设计也是一个道理。对于一个分布式系统，我们几乎永远不可能找到并修复所有的bug，错误流淌在分布式系统的血液里。解决方法不是消灭这些问题，而是容忍这些问题，在问题发生时，能自动回复，微服务组成的系统，每一个微服务都可能挂掉，这是常态，我们只要有足够的冗余和备份即可。即所谓的弹性设计（Resilience）或者叫高可用设计（High Availability）。

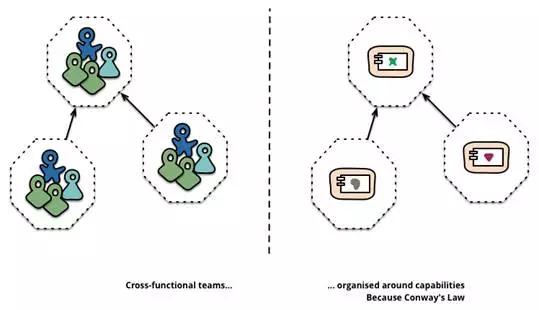
### 第三定律（种瓜得瓜，做独立自治的子系统减少沟通成本）



这是康威第一定律组织和设计间内在关系的一个具体应用。更直白的说，你想要什么样的系统，就搭建什么样的团队。如果你的团队分成前端团队，Java后台开发团队，DBA团队，运维团队，你的系统就会长成下面的样子：



相反，如果你的系统是按照业务边界划分的，大家按照一个业务目标去把自己的模块做出小系统，小产品的话，你的大系统就会长成下面的样子，即微服务的架构



微服务的理念团队间应该是inter-operate（互操作）, not integrate（不集成）。inter-operate是定义好系统的边界和接口，在一个团队内全栈，让团队自治，原因就是因为如果团队按照这样的方式组建，将沟通的成本维持在系统内部，每个子系统就会更加内聚，彼此的依赖耦合能变弱，跨系统的沟通成本也就能降低。

### 第四定律（合久必分，分而治之）

前面说了，人是复杂的社会动物，人与人的通过非常复杂。但是当我们面对复杂系统时，又往往只能通过增加人力来解决。这时，我们的组织一般是如何解决这个沟通问题的呢？Divide and conquer（分而治之）。大家看看自己的公司的组织，是不是一个一线经理一般都是管理15个人以下的？二线经理再管理更少的一线？三线再管理更少的，以此类推。（这里完全没有暗示开发经理比程序猿更难管理）

所以，一个大的组织因为沟通成本/管理问题，总为被拆分成一个个小团队。

最后，康威定律告诉我们组织沟通的方式会在系统设计上有所表达，每个经理都被赋予一定的职责去做大系统的某一小部分，他们和大系统便有了沟通的边界，所以大的系统也会因此被拆分成一个个小团队负责的小系统（微服务是一种好的模式）

## 康威定律如何解释微服务的合理性

了解了康威定律是什么，再来看看他如何在半个世纪前就奠定了微服务架构的理论基础。

人与人的沟通是非常复杂的，一个人的沟通精力是有限的，所以当问题太复杂需要很多人解决的时候，我们需要做拆分组织来达成对沟通效率的管理。

组织内人与人的沟通方式决定了他们参与的系统设计，管理者可以通过不同的拆分方式带来不同的团队间沟通方式，从而影响系统设计。

如果子系统是内聚的，和外部的沟通边界是明确的，能降低沟通成本，对应的设计也会更合理高效。

复杂的系统需要通过容错弹性的方式持续优化，不要指望一个大而全的设计或架构，好的架构和设计都是慢慢迭代出来的。

带来的具体的实践建议是：

* 我们要用一切手段提升沟通效率，比如slack，github，wiki。能2个人讲清楚的事情，就不要拉更多人，每个人每个系统都有明确的分工，出了问题知道马上找谁，避免踢皮球的问题。
* 通过MVP的方式来设计系统，通过不断的迭代来验证优化，系统应该是弹性设计的。
* 你想要什么样的系统设计，就架构什么样的团队，能扁平化就扁平化。最好按业务来划分团队，这样能让团队自然的自治内聚，明确的业务边界会减少和外部的沟通成本，每个小团队都对自己的模块的整个生命周期负责，没有边界不清，没有无效的扯皮，inter-operate, not integrate。

做小而美的团队，人多会带来沟通的成本，让效率下降。亚马逊的Bezos有个逗趣的比喻，如果2个披萨不够一个团队吃的，那么这个团队就太大了。事实上一般一个互联网公司小产品的团队差不多就是7，8人左右（包含前后端测试交互用人等，可能身兼数职）。

再对应下衡量微服务的标准，我们很容易会发现他们之间的密切关系：

1. 分布式服务组成的系统
2. 按照业务而不是技术来划分组织
3. 做有生命的产品而不是项目
4. Smart endpoints and dumb pipes（我的理解是强服务个体和弱通信）
5. 自动化运维（DevOps）
6. 容错
7. 快速演化