每个程序员都该知道的五大定理

原创：伯乐在线 伯乐在线 2017-09-15

(点击上方蓝字，快速关注我们)

> 编译：伯乐在线/孙腾浩
> 如有好文章投稿，请点击 → 这里了解详情

定律-或称法则，可以指导我们并让我们在同伴的错误中学习。这篇文章中，我将介绍我每次设计或实现软件时出现在我脑海的五大定律。其中有些和开发有关，有些和系统组织有关。它们可以帮助你成为合格的软件工程师。

墨菲定律

> “凡事可能出错，就一定出错。”

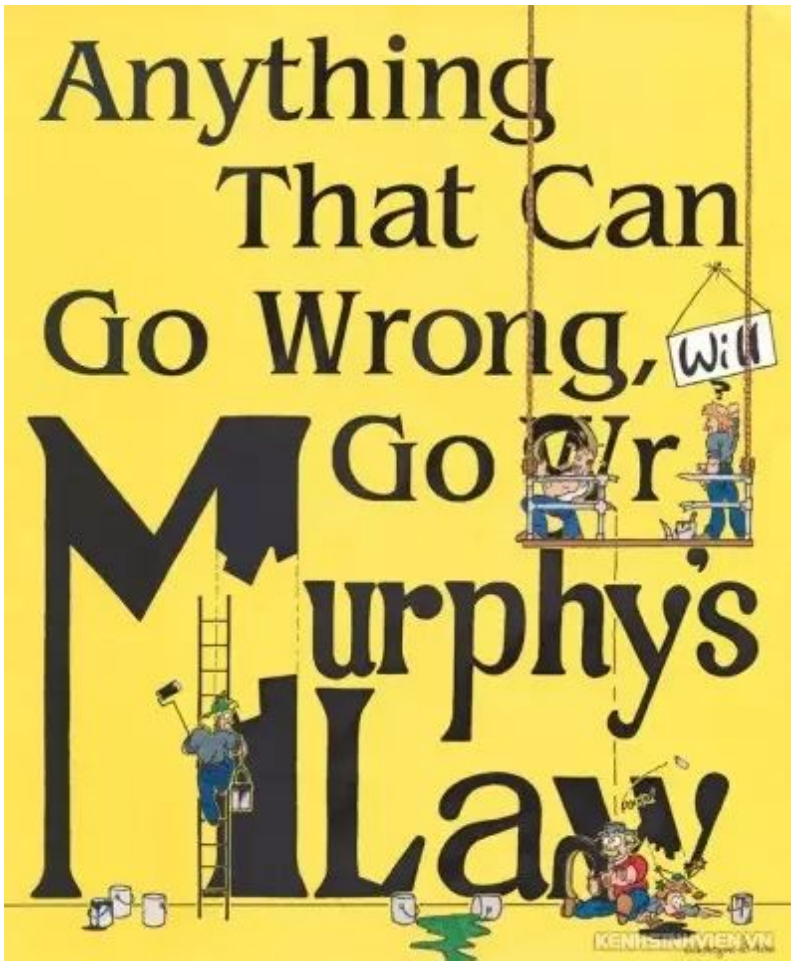
这条定律来源于 Edward Murphy —— 一名航天工程师在 50 年代初对火箭测试失败的回应。这条定律给我们的启示是永远在系统关键地方使用防御性设计，因为系统某些地方总会出错！

这条定律很容易引入软件工程领域。当你将软件暴露给终端用户，他们会创造性地输入一些出人意料的内容，使系统宕机。所以你需要让你的软件足够健壮，能够检测并警告非预期行为。

当你在机器上运行软件时，任何地方都有可能发生问题 —— 从硬盘上的系统到数据中心的电力供应。所以你必须确保你设计的架构在每个层级都可以应对故障。

我曾经有机会领略过几次墨菲定律。 举个例子，我曾经在一个批处理框架中使用字符串“null”来表示空值，我并不认为这有问题，直到有个名字叫“Null”的用户提交了一个交易订单，我们的报表流程中断了几个小时…… 还有一次，在另一个项目中。当所有东西都准备好部署到生产环境了，突然 Azure 基础设施故障导致我们运行自动化脚本的服务器宕机了。

现实世界中的经验教训提醒着我生活的艰难 —— “凡事可能出错，就一定出错”。 所以，心中牢记墨菲定律，设计健壮的软件。



Knuth定律

> “在（至少大部分）编程中，过早优化是万恶之源。”

这条定律也是 Donald Knuth 的经典语录之一，它告诫我们不要过早优化应用程序中的代码，直到必须优化时再优化。

的确，简单易读的源码可以满足 99% 的性能需要，并能提高应用的可维护性。最开始使用简单的解决方案也让后期性能出现问题时更容易迭代和改进。

垃圾自动回收的编程语言中，字符串的连接常常是过早优化的例子。在 Java 或 C# 中，String 对象是不可变的，我们学会使用其他结构动态创建字符串，比如 StringBuilder。但事实上直到你分析完个应用程序前，你并不知道 String 对象创建了多少次并对性能的产生多大影响。所以首先编写尽可能整洁的代码，之后在必须的时候再优化，往往这样做更有意义。

然而，这条规则并不应该阻止你去学习编程语言的性能权衡和正确的数据结构。并且，正如所有其他性能问题，你在优化前要测量开销。

North定律

> “每一个决定都是一次权衡”

好吧，我承认这是取自 Dan North 的演讲 Decisions,Decisions，它目前还不是公认的定律。 但这条语录影响了我做的每个决定，所以我把它放在这。

开发者日复一日的生活中，我们每天都做无数个大大小小的决定。从命名变量到自动化（手动）任务，再到定义平台架构。

这条语录强调无论你做的选择是什么，你总会放弃一个或多个选项

但这不是最重要的。 最重要的是理智地做出决定，了解其他选项，清楚你为什么不选择它们。你要始终根据当前你掌握的信息来权衡并做出决定。

但是如果后来你了解到新的信息，并发现之前的决定是错误的，这也没关系。关键是记清楚你为什么做出那个决定，重新评估新的选项之后再做出新的理智的决定。

重复一遍

> “每一个决定都是一次权衡”

所以，做出选择并对所有选项心知肚明。

Conway定律

> “系统设计的架构受限于生产设计，反映出公司组织的沟通架构”

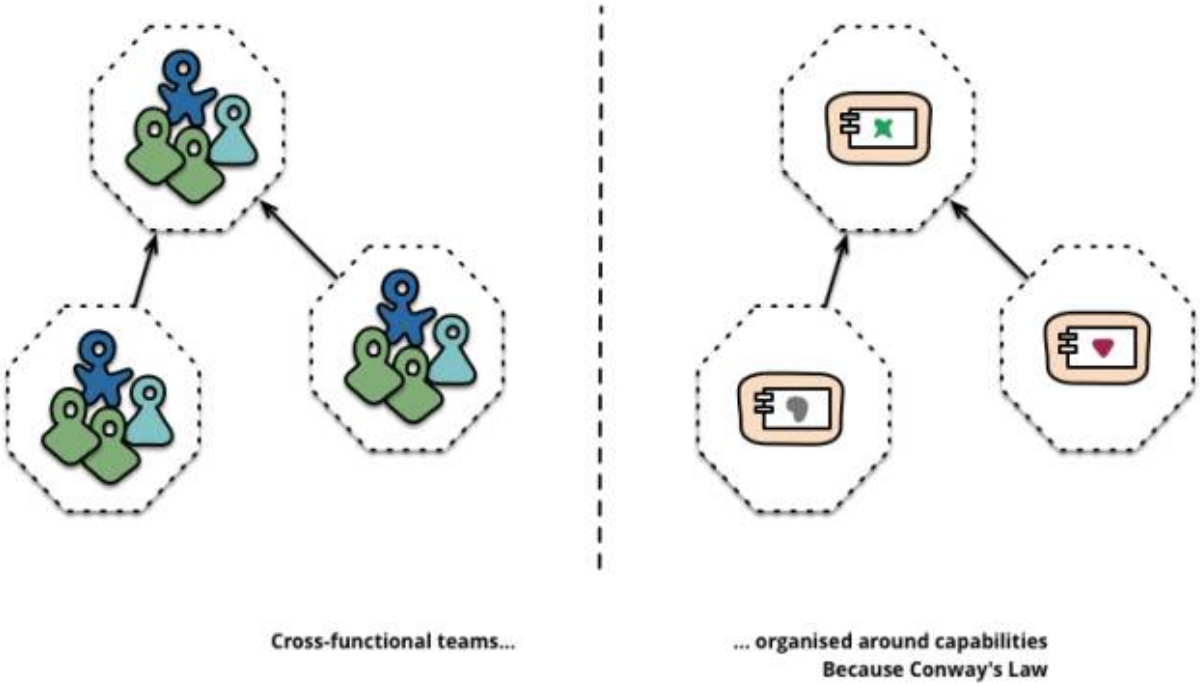
在 60 年代，一位名叫 Melvin Conway 的工程师注意到公司组织结构影响到他们开发的系统的设计。他用一篇论文描述了这个观点，并命名为“Conway定律”。

这条定律很适用于软件开发领域，甚至体现到代码层面上。交付软件组件的各个团队组织结构直接影响到组件的设计。

举个例子，一个集中式的开发者团队会开发出各组件耦合的整体应用。另一方面，分布式的团队会开发出单独的（微）服务，每一部分关注点分离清晰。

这些设计没有好坏之分，但它们都是受到团队沟通方式的影响。在全球有大量独立开发者的开源项目，通常是模块化和可重用库，这就是很有说服力的例子。

如今，将大的集成应用解耦成微服务已成趋势。这很棒，因为这可以加速交付使用项目。但你也应该牢记 Conway定律，在公司组织构建中投入与技术开发同样多的工作。



琐碎定律（帕金森琐碎定律）

> “组织成员投入大量精力到琐碎的事情上。”

这条定律论点是在会议中花费的时间与事情的价值成反比。的确是这样，人们更愿意把注意力和观点放在他们熟悉的事物上，而不是复杂的问题上。

帕金森给出一个例子，一场会议中，成员们讨论两件事：为公司建核反应堆和为员工建车棚。建反应堆是一件巨大而复杂的任务，没有人能完全掌控全局。他们完全信赖流程和系统专家，并很快接受了项目。

另一边，建车棚是一般人都可以做的，每个人都可以对颜色有意见。事实上，每个会议成员都会表达自己的意见，使得建车棚的决议所花费的时间远远超过建反应堆的。

这条定律在软件行业十分出名，这个故事随后也被称为车棚效应

举个例子，开发者会花费更多时间到讨论正确缩进或函数命名，而不是讨论类的职责或应用架构。这是因为每个人都能认知几个字符的变动，但项目架构的变动则需要巨大的认知负载

你能注意到的车棚效应的另一个例子是 Scrum 演示。不要误会我，我喜欢演示，我认为这是一个很好的机会来面对用户并获得对应用程序的反馈。但通常 Scrum 演示过程中的讨论会转向琐碎问题，而不是审视全局。这些讨论也很重要，但你应该注意权衡更重要更复杂的问题。

一旦你了解这种规律，你将在会议和交流中发觉这种行为。 我并不是让你在每次讨论中避免“小”问题，提高你的意识可以帮助你关注真正的问题，并为这些会议做好准备。

结论

这五条定律只是我们行业中总结出的教训中一些例子。随着软件开发经验的增长，我们将会学会更多。 尽管其中某些定律现在看起来是常识，我始终坚信了解这些原则可以帮助你识别这些模式并做出反应。

伯乐在线

分享伯乐在线博客的热门和经典文章

微信号：jobbole

长按识别二维码关注

伯乐在线 旗下微信公众号

商务合作QQ：2302462408

阅读原文