《人月神话》读书笔记

本书要点：

用人月作为衡量一项工作的规模是一个危险和带有欺骗性的神话。在众多软件项目中，缺乏合理的进度安排是造成项目滞后的最主要原因，它比其他所有因素加起来的影响还要大。这大概是这本书以《人月神话》为标题的原因，它旨在为人们管理复杂项目提供具有洞察力的见解，书中既有很多发人深省的观点，又有大量软件工程的实践。本书内容来自 Brooks 博士在 IBM 公司 SYSTEM/360 家族和 OS/360 中的项目管理经验，该项目堪称软件开发项目管理的典范。

内容简介：

在很多方面，管理一个大型的计算机编程项目和管理其它行业的大型工程很相似(比大多数程序员所认为 idea 还要相似)；另一方面它又有差别（比大多数职业经理人所认为的差别还要大）。由于人员的分工，大型编程项目碰到的管理问题和小项目碰到的管理问题区别很大；关键需要的是维持产品自身的概念完整性。该书主要对软件工程领域所涉及到的工程管理的知识进行了系统的阐述。

精编书摘：

焦油坑

过去几十年的大型系统开发就犹如一个焦油坑，很多大型和强壮的动物在其中剧烈地挣扎。他们中大多数开发出了可运行的系统—不过只有极少数的项目满足了目标、进度和预算的要求。表面上看起来好像没有任何一个单独的问题会导致困难，每个问题都能获得解决，但是当它们相互纠缠和累积在一起的时候，团队的行动就会变得越来越慢。

人月神话

在众多软件项目中，缺乏合理的进度安排是造成项目滞后的最主要原因，它比其他所有因素加起来的影响还要大。

乐观主义

系统编程的进度安排背后的第一个错误的假设是：一切都将运作良好，每一项任务仅花费它所“应该”花费的时间。

人月

用人月作为衡量一项工作的规模是一个危险和带有欺骗性的神话。它暗示着人员数量和时间是可以相互替换的。

人数和时间的互换仅仅适用于以下情况：某个任务可以分解给参与人员，并且他们之间不需要相互的交流。

系统测试

在进度安排中，顺序限制所造成的影响，没有哪个部分比单元调试和系统测试所受到的牵涉更彻底。

对于软件任务的进度安排，以下是我使用了很多年的经验法则：

1/3 计划

1/6 编码

1/4 构件测试和早期系统测试

1/4 系统测试，所有的构件已完成

特别需要指出的是，不为系统测试安排足够的时间简直就是一场灾难。

空乏的估算

观察一下编程人员，你可能会发现，同厨师一样，某项任务的计划进度，可能受限于顾客要求的紧迫程度，但紧迫程度无法控制实际的完成情况。

开发并推行生产率图表、缺陷率图表、估算规则等等，而整个组织最终会从这些数据的共享上获益。

或者，在基于可靠基础的估算出现之前，项目经理需要挺直腰杆，坚持他们的估计，确信自己的经验和直觉总比从期望派生出的结果要强得多。

重复产生的进度灾难

向进度落后的项目中增加人手，只会使进度更加落后。

在新的进度安排中分配充分的时间，以确保工作能仔细、彻底地完成，从而无需重新确定时间进度表。

在现实情况中，一旦开发团队观察到进度的偏差，总是倾向于对任务进行削减。当项目延期所导致的二次成本非常高时，这常常是唯一可行的方法。

外科手术队伍

如果在一个 200 人的项目中，有 25 个最能干和最有开发经验的项目经理，那么开除剩下的 175 名程序员，让项目经理来编程开发。

大型项目的每一个部分由一个团队来解决，但是该队伍以类似外科手术的方式组建，而非一拥而上。也就是说，同每个成员截取问题某个部分的做法相反，由一个人来完成问题的分解，其他人给予他需要的支持，以提高效率和生产力。

扩建过程的成功依赖于这样一个事实，即每个部分的概念完整性得到了彻底的提高—决定设计的人员是原来的 1/7 或更少。所以，可以让 200 人去解决问题，而仅仅需要协调 20 个人，即那些“外科医生”的思路。

可以认为整个系统必须具备概念上的完整性，要有一个系统架构师从上至下地进行所有的设计。要使工作易于管理，必须清晰地划分体系结构设计和实现之间的界线，系统架构师必须一丝不苟地专注于体系结构。

贵族专制、民主政治和系统设计

概念的完整性

我主张在系统设计中，概念完整性应该是最重要的考虑因素。也就是说为了反映一系列连贯的设计思路，宁可省略一些不规则的特性和改进，也不提倡独立和无法整合的系统，哪怕它们其实包含着许多很好的设计。

对于非常大型的项目，将设计方法、体系结构方面的工作与具体实现相分离是获得概念完整性的强有力方法。

结构师的工作，是运用专业技术知识来支持用户的真正利益，而不是维护销售人员、制作者所鼓吹的利益。

概念的完整性的确要求系统只反映唯一的设计理念，用户所见的技术说明来自少数人的思想。实际工作被划分成体系结构、设计实现和物理实现，但这并不意味着该开发模式下的系统需要更长的时间来创建。经验显示恰恰相反，整个系统将会开发得更快，所需要的测试时间将更少。同工作广泛的水平分割相比，垂直划分从根本上大大减少了劳动量，结果是使交流彻底的简化，概念完整性得到大幅提高。

画蛇添足

尽早交流和持续沟通能使结构师有较好的成本意识，使开发人员获得对设计的信心，并且不会混淆各自的责任分工。

第二个系统是人们所设计的最危险的系统，通常的倾向是过分地进行设计。

为功能分配一个字节和微秒的优先权值是一个很有价值的规范化方法。

为什么巴比伦塔会失败

巴比伦塔建造失败的主要原因有两个方面，一个是缺乏了交流，另一个则是交流的结果—组织。他们无法相互交谈，从而无法合作。当合作无法进行时，工作陷入了停顿。
团队可以通过三种途径进行相互之间的交流沟通—非正式途径、会议、工作手册。

没有银弹

软件生产率在近年来取得的巨大进步来自对人为障碍的突破，例如硬件的限制、笨拙的编程语言和机器时间的缺乏等等，这些障碍使次要任务实施起来异常艰难。

我建议：仔细地进行市场调研，避免开发已上市的产品；在获取和制定软件需求时，将快速原型开发作为迭代计划的一部分；有机地更新软件，随着系统的运行、使用和测试，逐渐添加越来越多的功能；不断挑选和培养杰出新生代的概念设计人员。

没有任何技术或管理上的进展，能够独立地许诺在生产率、可靠性或简洁性上取得数量级的提高。

解决管理灾难的第一步是将大块的“巨无霸理论”替换成“微生物理论”，它的每一步—希望的诞生，本身就是对一蹴而就型解决方案的冲击。它告诉工作者进步是逐步取得的，伴随着辛勤的劳动。对规范化过程进行不懈的努力。由此，诞生了现在软件工程。

作者将软件开发比作人狼，而将提高软件开发效率的方法比作银弹。作者预言未来十年，不可能找到一种有效地银弹将软件开发效率提高一个甚至几个数量级。事实证明确实如此。现在已经出现很多快速开发平台（如：图形化编程、测试驱动），或者是模板框架开发（如：wordpress），但很少有可以真正不写代码就能满足所有需求的平台或者框架。没有银弹的断言进一步说明了软件开发的复杂性，一致性，可变性和不可见性，但是追求组件复用和技术跨平台，可以封装部分由平台或者底层实现差异而带来的复杂度。

虽然没有银弹，虽然这是个巨大的焦油坑，但是我们痛苦并快乐着，一次次看着它不断突破自己，实现技术上的飞跃，一切都是未知而新奇的，这就足够支撑我们坚持下去了。