# 人月神话读后感

来源：网络 作者：梦里花开 更新时间：2025-03-14

*第一篇：人月神话读后感人月神话读后感、《人月神话》是预言了未来还是扼制了未来?事实是：我们目前的许多工程知识，——无论是从书上看到的，还是从实践中经验到的——大多未曾脱离《人月神话》之所言，人月神话读后感。我在开篇中说《人月神话》“是一本...*

**第一篇：人月神话读后感**

人月神话读后感

、《人月神话》是预言了未来还是扼制了未来?

事实是：我们目前的许多工程知识，——无论是从书上看到的，还是从实践中经验到的——大多未曾脱离《人月神话》之所言，人月神话读后感。

我在开篇中说《人月神话》“是一本可怕的书”。然而我感受恳挚的可怕之处在于：现今凡是论及工程(且不要让人感受是离经叛道)，那么所解说的定然是Brooks的这么的经验以及由此推出的见解，可能在不违拗这些经验和见解上的一些翔实的实作措施!我们全然不顾书中所言是假象，还是性质的推论，可能只是假象归纳的一个(未必准确的)答案。尽管这些答案大多数时候都好像预期地展目前你的切实工程中：

原文中还有众多相仿的见解、假象和答案，都成为了切实工程中的既存假象。先民们所说的圣人以及通神者，皆因他们多数时候在准确地预言自己的切实。只有当这个“多数时候”变成半点的时候，先民们才会置疑圣人和通神者的力气。其实我们懂得并未曾预言未来的人，大多数时候是两种情形导致的假象：

他做出了准确的推断;

你主观地跟随了他对未来的设定。

后者是风险的。大师们预言了未来也就改换了未来，即便未来未必“该当”好像他所预言的那样。

但万一这种预言的前提不准确，那么未来定然脱离这种波及而回到它该当的事态上去。好像我们看到的另一些事实一样，有许多假象阐明，我们正在归来工程\*\*\*的道路上摸索前进。我们也觉察，在大多数情形下，先哲们的预言在实践中被检讨着，只是偶尔“不太灵光”。下表则列出一些不同的例子：

注1：我例举了爽利的一些见解，并不阐明我是Ap/Xp的fans。Ap/Xp的问题另论，在这里，我只是解释存在一种不同的信念。

注2：Brooks尔后确认“定然丢弃原型”是一个不太准确的见解。

注3：Brooks在这里未曾犯讹谬，只是他所谈论的是狭义的流程图，而我们例举的时序图则更广义。

我们追忆上一细节，在《人月神话》中的那“31%的答案”的前提——也即便那7%的性质中，如下两项是显明猜忌的(也是重要置疑)：

目标的性质：是大型工程，是系统项目，而不是过程

个体的性质：是私利性的其实早就有人意识到个体的性质“未必全是私利的”，尊重这些个体就会带来一些收获。例如Ap正是因为更尊重开发人员的禀性与力气，以及互相间的配合而获得了效率的晋级。

再进一步地说，既然Brooks设定了“大型工程或系统项目”这么的目标，并给出了一些答案。那么在“小那么一点点的”工程项目中，是不是这些答案就无须定了呢?例如Brooks的众多提倡，对于某些目标——例如你要用为期三个月的工夫开发一个的产品——就并不是很管用;可能大约无法厉行——例如你的群体总共只有6个人，连“外科手术式的群体”都组织不起来，读后感《人月神话读后感》。

Brooks的答案对于同样的目标，以及在他所述的“性质”未能发生改换时，还是比拟管用(或有厉行的可能性)。因而上述一些例外，总是在上述的“7%的性质”被抵赖或被改换的情形下获得的。因而我们提出的问题是“如何抵赖或改换”这些难以撼动的性质。然而在我看来，Brooks早曾经在最佳位置上，给出了撬动它们的一个支点：

Brooks感受发生“自力更生小型过程”与“编程系统产品”是不同的问题。

Brooks谈论的编程系统产品的规模究竟有多大呢?我想起码该当是以IBM 360为参看的。不过书中在引用Joel Aron(IBM在马里兰州盖兹堡的系统技巧主管)的例子时说，“大型意味着过程员的数目超过25人，将近30,000行的号召”。而按照《人月神话》的数据：人均效率800号召/人年，则这个“大型项目”该当必需1.5年能力告终。另外，还必需大约一倍的人工，来负责除开代码之外的测验、管教、文档和沟通等工作。

好的，万一你有一个“(起码)50人，开发一年半”的项目，那么你能够先接受Brooks的答案去实践一下：起码你能够有工夫来谈论工程问题，也能够组建那样规模的群体。然而，难道只有这么的“大型工程”才算得工程，而“小那么一点点”的就不算吗?切实是，我们一方面在做着“小那么一点点的”工程项目，另一方面在听着全副业界嘈杂着“为更大规模的工程”而准备的工程理论。我们总在实践Brooks的“答案”可能“预言”，而淡忘这些答案的前提：

Brooks的经验源自对IBM 360等大型项目标实践与分析;

Brooks所述的工程是要获得编程系统产品;

Brooks感受编程系统产品的工作量可能是自力更生小型过程的9倍(在告终大约雷同功能的情形下)。

事实上我们目前的软件工程的进展是被驾驶了，而不是被预言了。从性质上来说，Brooks在《人月神话》中只是谈论了大型工程的厉行，以及相应规模下的群体创立。而我们，便按照这么的设定来摆开了全副软件工作的工程化厉行。

促成这种现状的，并不但仅是一本书的能力，还在于商业的能力。因为只有在这么扩展开来的工作环境中，才可能有商业时机。——即便那些工程顾问与厉行专家历来未曾厉行过“50人，开发一年半”这么的项目，凡是他们能报出Brooks的名字，能谈及某些工具在应付“大型项目”中的获胜经验，他们就曾经获胜了一半了。

为什么“爽利”之初颇受争议?为什么爽利对一些中小型的群体显得管用和可厉行?为什么当这些争议被摆在现在的获胜平息尔后，传统工程的理论家们却不忘恨恨地评上一句：那是一种不能(或难以)利用于大型工程的措施呢?!

因为万一大家都很“爽利”，都只做比这些大型工程“小那么一点点”的工程，那么传统工程的专家们就失业了。反到来，只有把工程做大，大到“爽利”错过了含义，而“宏伟”变成了性质的时候，传统工程就可感受任何失利找到借口：看啊，Brooks就说过“未曾银弹”嘛。

**第二篇：人月神话读后感**

3110402157\_ 王森\_软件11

2《人月神话》读后感

在阅读这本书之前,已经很多次听到关于人月神话这本书以及他的作者Brooks的消息了.在软件领域, 《人月神话》具有深远影响力而且畅销不衰.这一次,正好老师的作业要求我们阅读这本书,我终于使有机会阅读这本经典之作了.在这过去的几个星期里面,一点一滴的阅读这本书,粗略的了解了这本书.首先,让我印象深刻的是《人月神话》提出的两条著名的法则：

1、人月神话：向一个已经延后的项目中投入更多的人力资源只会让它更延后。人月神话看上去这么浪漫的名字，原来并不是真的说神话故事，作者阐述的主要观点是在软件开发项目上项目进度和增加人员这两个概念是不能互换。虽然已经时隔20多年了，这本书依然给我震撼，一是让我惊讶的是，美国20年前软件项目所面临的问题，在我们现在依然如此，糟糕的情况没有改变，大家仍旧在焦油坑里挣扎，而且看上去没有解决办法.当读到“是当意识到进度的偏移时，下意识（以及传统）的反应是增加人力。这就像使用汽油灭火一样，只会使事情更糟。越来越大的火势需要更多的汽油，从而进入了一场注定会导致灾难的循环。这让我明白了一个重要的道:理项目的进度是不能够光靠人力的增加来推进的.2、没有银弹:没有任何技术或管理上的进展，能够独立地许诺十年内使生产率、可靠性或简洁性获得数量级上的进步。

虽然现在有不少人对他的观点持反对或不同意见，但我始终觉得他的观点是对的——根本和次要问题的划分以及定义。作者认为软件开发困难的部分是概念的结构，如规格化、设计和测试等概念的结构，而不是概念的表述和实现概念，虽然实现概念可能占用了小于90%的时间，就如现今的软件开发一样，系统分析通常占用的整个项目开发时间不超过20%，而80%的时间花在编程上一样。

这两个原则已经在过去的几十年间得到了验证.我相信在未来,它以依旧是成立的.另外,在焦油坑那一章里面,有一句话让我难以忘怀:岸上的船,如同海上的灯塔,无法移动.是呀, 过去几十年的大型 系统开发就犹如这样一个焦油坑，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，其中只有非常少数的项目满足了目标、时间进度和预算的要求。各种团队，大型的和小型的，庞杂的和精干的，一个接一个淹没在了焦油坑中。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都能被解决，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。对问题的麻烦程度，每个人似乎都会感到惊讶，并且很难看清问题的本质。不过，如果我们想解决问题，就必须试图先去理解它。这就是生活真理。要想解决一件事，首先要了解事情的始末。提出问题就是解决问题的答案。

人月神话还让我了解到, 软件系统可能是人类创造中最错综复杂的事物.往往一个很小的功能，实在也需要开发职员的架构设计方面的完善，对 其它模块的影响及扩展,以及代码编写工作。用户在前台可能看到的只是几个文字，实际是中开发职员昼夜奋战的结果。很多时候，客户的需求修改，在他们眼里看起来是如此地Easy,可他们却忽视了很多他们看不到的因素.总而言之，《人月神话》是一部IT界的神话，经久不衰.它就像是一颗“银弹”，教会我们如何去消灭软件项目这只“人狼”，指引着每个IT从业者认真开发,开拓进取.人月神话将带领IT界的精英们创造一个又一个IT界的神话..

**第三篇：人月神话读后感**

人月神话读后感

二十九年前(1975)﹐IBM大型电脑之父──Fred Brooks 出版一本书﹕“The Mythical Man-Month”。收集了他在1960年代领导1000多人共同发展OS/360大型软件系统的心得和经验。该书是论文集﹐其中有一篇文章叫“The Mythical Man-Month”﹐他就以此作为书名。在1956~1965 之间﹐Brooks实际领导IBM 360 大型电脑的开发计划﹐包括硬体结构及庞大的OS/360作业系统在内﹐因之他具有IBM 大型电脑之父的尊称。由于OS/360是多达1000位程式师共同合作的大型软件开发工作﹐让他深刻了解到大型软件开发的技术和管理上所面临的种种困难和挑战。于是﹐他就将其领导开发OS/360软件系统的经验心得收集在这本书里。人们常拿Man-Month（多少人﹐做多少个月）来计算软件的工作量﹐但是Brooks发现软件的开发工作是需要人与人之间密切沟通的﹐使得设计工作不易分割﹐所以Man-Month 为单位的计算方法是有问题的(mythical)。也就得出著名的Brooks法则── 「对于进度已落后的软件开发计划而言﹐若再增加人力﹐只会让其更加落后。」(Adding manpower to a late software project makes it later)这是该书名称的涵义。

看完此书后，我发现人月神话无处不在，其实在我们做

软件工程来说，此书已经渗透进去了。本书作者为人们管理复杂项目提供了颇具洞察力的见解，既有很多发人深省的观点，也有大量的软件工程实践。本书对我触动最大的，一是保持设计的概念完整。无论对小软件还是大软件，都必须由一个设计师主导，最多两个人讨论来共同完成软件的整体设计。作为一个软件，一个系统，必须有一个清晰明确的概念模型，大家都在这个框架下工作，所有的创新发展都必须与基本的概念相吻合。具体的实现人员可以细化概念，但只有总设计者才有否定与发展基本概念的权力。需要注意的一点是，即使是总设计师一直是同一个人，他脑海中所认为理所当然的规则或者概念，很可能由于没有明确的文档化，而没有成为所有开发者共同的概念。概念的完整性，对于很多小规模软件，由于开发人员不多，开发经理一般都能控制住所有的代码，概念完整性在组织层面就维持住了。但要注意以后的Bug修改，功能扩展的时候，也要时刻留意与最初的设计是否概念上相容。对于大规模的软件系统，则必须通过树状组织结构，层层控制，总设计师还是一到两人，每一层都有对下层的绝对把握能力。

二是“一个拿2倍工资的人，生产率可能是其他人的10倍。”不知道其他公司的程序员们如何看。我觉得，作为公司，应该给最好的人最好的待遇，或者说给比目前更高的待遇。组建一个团队，最好的就是那种精英团队。微软就是这

种思路吧，把最聪明的人集中在一起，想不成功都难。

三是进度落后与增加人力。记得当年看《C＋＋编程思想》，Bruce说“十个妇女不能在一个月内生下小孩”（大意），于我心有戚戚焉。而本书作者Brooks得出的结论是对我是震撼性的：“向进度落后的项目中增加人手，只会使进度更加落后”。以前，增加人手基本是挽救进度落后项目的主要办法。这个办法行不通的话，难道只有“加班”一条路了？如果不想加班，不想削减功能，不想推迟发布日期，那么。。。唯一的方法还是只有….加人。加足够的人。而且不要逐步加入，一定要一次性加入。要小心的是，新加入的人可能对原来的组织造成冲击，或者对原来的设计有不同意见（特别是加入的人中有比较强大的设计者）。那么，就当作，新组建了一个团队吧。交流，培训新人，就设计达成一致，继续向者目标前进。

不同的社会经验，不同的思想状态，对读本书的心得也不一样。在此我说说书中许多非常好的观点。

1.外科手术队伍The Surgical Team

项目经理在项目的初期必须清楚的估计项目的人月运作模式（时间、人力在项目各阶段的分配），例如什么时候需要出什么样成果，决定了什么时候需要什么样的人加入项目，这是项目经理的责任。

2.贵族专制，民主政治Aristocracy,Democracy,System

要获得概念的完整性，设计必须由一个人或具有共识的小组来完成。

3.画蛇添足The Second-System Effect

讲述的基本都是基于IBM 360操作系统以及编译程序等方面的经验，讲述如何避免开发第二个系统的风险，作者认为开发第二个系统的设计师设计出来的系统是最危险的,因此要求他们自律。

4.贯彻执行Passing the word

印象比较深刻的是“体系结构设计人员必须为自己描述的任何特性准备一种实现方法，但他不应该支配具体的实现过程。”

5.巴比伦塔会失败Why did the Tower of Babel Fail?讲述巴比伦塔会失败的原因是缺乏交流。

6.胸有成竹Calling the Shot

主要讲述如何计算编程时间，以及提出几个人的经验算法。讲述的各种算法可能都不太适合与现在的高级语言，但Portman的观点仍然适合现在，即编程人员实际的编程时间只有50%，其他的时间都花在了无关的琐碎事情上。

7.削足适履Ten Pounds in a Five-Pound Sack

主要讲述程序占用的空间等，在70年代比较突出，但现在好多了。

8.提纲擎领The Documentary Hypothesis

说明文档的作用

9.未雨绸缪Plan to Throw One Away

唯一不变的是变化本身。在大型项目中，项目经理需要有两个和三个顶级程序员作为技术轻骑兵，当工作繁忙最密集的时候，他们能急驰飞奔，解决各种问题。

10.干将莫邪Sharp Tools

主要讲述项目中管理好各种工具的重要性，项目经理首先要制定一种策略，让各种工具成为公用的工具，这样才能使开发、维护和使用这种工具的开发人员的效率更高，这种工具可能是开发人员开发出来的，也可能是使用现有的，可能是通用的，也可能是专用的或个人偏好的。比如：文档编写工具、开发工具（包括各种不同开发平台）、调试工具、测试工具、数据库工具、版本管理、项目管理工具等。

11.整体部分The Whole and the Parts

特别是这句话\"BELL实验室监控系统项目的V.A.Vyssotsky提出,“关键的工作是产品定义。许许多多的失败完全源于那些产品未精确定义的地方”,细致的功能定义，详细的规格说明，规范话的功能描述说明以及这些方法的实施，大大减少了系统中必须查找的BUG数量。虽然这句话的意思只是说明精确定义产品将减少BUG的数量，但我看到了系统分析的最重要的工作——产品定义。

12.祸起萧墙Hatching a Catastrophe

这章节说明使项目进度拖后的最大原因不是重要的事件，如新技术、重组等，而是一些琐碎的小事，每件小事只耽误半天或一天时间，但这种小事多以后，将使项目的进度严重拖后。项目对于公司就如程序对测试工程师一样，如果不了解它，它就是一个黑盒子，如果不打开这个黑盒子，你可能永远不知道盒子里面有什么。

13.另外一面The other face

本章说明程序的另一面——文档。

不了解，就无法真正拥有——歌德，作者引用的歌德的话来描述文档对客户的重要性，提出客户需要什么样的文档以及文档的格式和包含的内容，指出当时存在的大多数文档只描述了树木，形容了树叶，但没有整个森林的图案。想想，这种情况在现在仍然没有改变。于是作者提出了两个观点：

1.流程图：流程图是被吹捧得最过分的一种程序文档。许多程序甚至不需要流程图，很少程序需要一页以上的流程图

2.自动文档化（self-documenting）的程序：提出文档与程序合为一体，能很好的解决文档与程序分开造成的文档过时的问题，并说明了在程序中加入文档的一些方法和技巧。

14.没有银弹-软件工程中的根本和次要问题(No Silver Bullet-Essence and Accident in software Engineering)

人狼是传说中的妖怪，只有银弹才能杀死他。作者认为软件项目具有人狼的特性，因为软件项目也可能变成一个怪物，一个落后进度、超出预算、存在大量缺陷的怪物。作者通过软件系统的内在特性复杂性、一致性、可变性和不可见性来分析说明了软件天生就没有银弹。作者试图通过分析软件问题的本质和很多侯选银弹的特征来探究其中的原因。他行动的第一步是将大块的“巨无霸理论”替换成“微生物理论”。这个变化的过程告诉你，进步是逐步取得的，伴随着辛勤的劳动，对规范化过程应进行持续不懈的努力，而这个努力的过程相应的就诞生了软件工程。

15.再论《没有银弹》No Silver Bullet Refired

看完再论《没有银弹》后，虽然作者说有不少人对他的观点持反对或不同意见，但我始终觉得他的观点是对的——根本和次要问题的划分以及定义。作者认为软件开发困难的部分是概念的结构，如规格化、设计和测试等概念的结构，而不是概念的表述和实现概念，虽然实现概念可能占用了小于90%的时间，就如现今的软件开发一样，系统分析通常占用的整个项目开发时间不超过20%，而80%的时间花在编程上一样。

**第四篇：《人月神话》读后感**

学号：0967020449姓名：张小波班级：软件工程专升本3班

《人月神话》读后感

在软件领域中,很少能有像《人月神话》一样具有深远影响力和畅销不衰的著作。Brooks 博士为人们管理复杂项目提供了最具洞察力的见解,既有很多发人深省的观点,又有大量软件工程的实践,影响着一代又一代….通过阅读《人月神话》，我从中学到了一些东西：

首先，开发一个项目，我们错误的认为用人月这个工作量单位来估计和进行进度安排。成本的确随开发产品的人数和时间的不同，有着很大的变化，进度却不是如此。因此我认为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，而在系统编程中近乎不可能。当任务由于次序上的限制不能分解时，人手的添加对进度没有帮助。调试、测试的次序特性，许多软件都具有这种特征。因为软件开发本质上是一项系统工作——错综复杂关系下的一种实践——沟通、交流的工作量非常大，它很快会消耗任务分解所节省下来的个人时间。从而，添加更多的人手，实际上是延长了，而不是缩短了时间进度。

对于编程，有其乐趣和苦恼。创建事物的快乐，开发对其他人有用的东西的乐趣，将可以活动、相互啮合的零部件组装成类似迷宫的东西，这个过程所体现出令人神魂颠倒的魅力，面对不重复的任务，不间断学习的乐趣，工作在如此易于驾驭的介质上的乐趣——纯粹的思维活动，其存在、移动和运转方式完全不同于实际物体。将做事方式调整到追求完美，是学习编程的最困难部分；由其他人来设定目标，并且必须依靠自己无法控制的事物（特别是程序）；权威不等同于责任实际情况看起来要比这一点好一些；真正的权威来自于每次任务的完成任何创造性活动都伴随着枯燥艰苦的劳动，编程也不例外 人们通常期望项目在接近结束时，（bug、工作时间）能收敛得快一些，然而软件项目的情况却是越接近完成，收敛得越慢产品在即将完成时总面临着陈旧过时的威胁。

开发一个软件，我们要有合理的时间进度，开发人员要少而精，概念完整性必须考虑在内，要尽量做到尽早交流和持续沟通。

同时，文档形成了关键的枢纽，每个项目管理的工作都围绕着它们运转，它们是经理们的主要个人工具。对于计算机硬件开发项目，关键文档是目标、手册、进度、预算、组织机构图、空间分配、以及机器本身的报价、预测和价格；对于大学科系，关键文档类似：目标、课程描述、学位要求、研究报告、课程表和课程的安排、预算、教室分配、教师和研究生助手的分配；对于软件项目，要求是相同的：目标、用户手册、内部文档、进度、预算、组织机构图和工作空间分配。即使是一个小型项目，我们都会要求书写相关文档，对每个关键文档的维护提供了状态监督和预警机制并且本身就可以作为检查列表或者数据库。

良好的工作手册和组织架构可以开发出更加符合用户的需求。手册、或者书面规格说明，是一个非常必要的工具，尽管光有文档是不够的。手册是产品的外部规格说明，它描述和规

定了用户所见的每一个细节；同样的，它也是结构师主要的工作产物。

形式化定义是精确的，它们倾向于更加完整；差异得更加明显，可以更快地完成。但是形式化定义的缺点是不易理解。记叙性文字则可以显示结构性的原则，描述阶段上或层次上的结构，以及提供例子。它可以很容易地表达异常和强调对比的关系，最重要的是，它可以解释原因。在表达的精确和简明性上，目前所提出的形式化定义，具有了令人惊异的效果，增强了我们进行准确表达的信心。

通常，开发一个软件我们还会设立规模目标，控制规模，发明一些减少规模的方法——就如同硬件开发人员为减少元器件所做的一样。规模预算必须与分配的功能相关联； 在指明模块大小的同时，确切定义模块的功能。培养开发人员从系统整体出发、面向用户的态度是软件编程管理人员最重要的职能。

调试，是一种检验程序中的方法。然而调试是系统编程中很慢和较困难的部分，而漫长的调试周转时间是调试的祸根。它可以持续一个很长的时间，从而可能影响项目的交付日期。

为了更好的控制进度，我们需要制定一个严格的进度表来控制项目的进度表，进度表由里程碑和日期组成。里程碑必须是具体的、特定的、可度量的事件，能进行清晰能定义。进度表有时可以根据进展情况进行适度的修改。

产品测试时每个产品在提交给用户的一道程序。而这项工作主要由产品测试机构/小组根据规格说明检查机器和程序，充当麻烦的代言人，查明每一个可能的缺陷和相互矛盾的地方。每个开发机构都需要这样一个独立的技术监督部门，来保证其公正性。产品－测试小组则是顾客的代理人，专门寻找缺陷。不时地，细心的产品测试人员总会发现一些没有贯彻执行、设计决策没有正确理解或准确实现的地方。出于这方面的原因，设立测试小组是使设计决策得以贯彻执行的必要手段，同样也是需要尽早着手，与设计同时实施的重要环节。

一个已开发的项目，我们需要对它进行后期维护。其维护基本上不同于硬件的维护，它主要由各种变更组成，如修复设计缺陷、新增功能、或者是使用环境或者配置变换引起的调整而且维护总成本通常是开发成本的40％或更多。维护成本受用户数目的严重影响，用户越多，所发现的错误也越多。在每次修复之后，必须重新运行先前所有的测试用例，从而确保系统不会以更隐蔽的方式被破坏。其实，对于一个项目，我们要尽量做到完美，减少以后的维护困难和成本。

在计算机技术进步的同时，计算机相关学科知识也在飞速发展。兴趣太多，令人兴奋的学习、研究和思考的机会也太多——多么不可思议的矛盾啊！这个神奇的时代远远没有结束，它依然在飞速发展。更多的乐趣，尽在将来。

**第五篇：《人月神话》读后感**

～-6-23　字数：1345当我捧起《人月神话》，马上就被深深的吸引了。书中很多细微之处都对我的思维造成了冲击。上一本给我类似感觉的书是那本四人帮的《设计模式》，已经很久没有看到这么好的书了，郑重推荐。把感触比较深的几点记下来，顺便整理一下自己的思路，与大家分享。1，保持设计的概念完整。无论对小软件还是大软件，都必须由一个设计师主导，最多两个人讨论来共同完成软件的整体设计。作为一个软件，一个系统，必须有一个清晰明确的概念模型，大家都在这个框架下工作，所有的创新发展都必须与基本的概念相吻合。具体的实现人员可以细化概念，但只有总设计者才有否定与发展基本概念的权力。需要注意的一点是，即使是总设计师一直是同一个人，他脑海中所认为理所当然的规则或者概念，很可能由于没有明确的文档化，而没有成为所有开发者共同的概念。在其他开发者编码的时候，就可能会生成与概念相抵触的东东（模块，功能，算法），导致整体结构的恶化。这个时候总设计师一定要即时发现，做出更正。概念的完整性，对于很多小规模软件，由于开发人员不多，开发经理一般都能控制住所有的代码，概念完整性在组织层面就维持住了。但要注意以后的Bug修改，功能扩展的时候，也要时刻留意与最初的设计是否概念上相容。对于大规模的软件系统，则必须通过树状组织结构，层层控制，总设计师还是一到两人，每一层都有对下层的绝对把握能力。我以前参加过一个15人左右的项目组，就是分为两层。感觉整体概念完整性的控制效果还不错。我没有更多人数项目的具体实践经验，希望以后能有机会参与比较大的项目。2，“一个拿2倍工资的人，生产率可能是其他人的10倍。”我和我的同学，一个小公司的技术总监聊起这个，他也是十分的认同。不知道其他公司的程序员们如何看。我的同事中有一个牛人，做出的贡献特别大，应该相当于我们公司普通的十个程序员，不过工资最多也就是普通程序员的二倍。是不是有些不公平呢？我也说不清楚。因为那些普通程序员也十分的努力。不过，我觉得，作为公司，应该给最好的人最好的待遇，或者说给比目前更高的待遇。组建一个团队，最好的就是那种精英团队，大家都是牛人，效率会特别高。微软就是这种思路吧，把最聪明的人集中在一起，想不成功都难亚。3，进度落后与增加人力。记得当年看《C＋＋编程思想》，Bruce说“十个妇女不能在一个月内生下小孩”（大意），于我心有戚戚焉。而本书作者Brooks得出的结论是对我是震撼性的：“向进度落后的项目中增加人手，只会使进度更加落后”。以前，增加人手基本是挽救进度落后项目的主要办法。这个办法行不通的话，难道只有“加班”一条路了？但长期加班是对个人的摧残，我更愿意利用业余时间去看书，例如看这本“人月神话”。：）如果不想加班，不想削减功能，不想推迟发布日期，那么。。。唯一的方法还是只有….加人。加足够的人。而且不要逐步加入，一定要一次性加入。要小心的是，新加入的人可能对原来的组织造成冲击，或者对原来的设计有不同意见（特别是加入的人中有比较强大的设计者）。那么，就当作，新组建了一个团队吧。交流，培训新人，就设计达成一致，继续向者目标前进。感触还有很多，以后如果有机会再写。不过，我决定去买本英文版回来，收藏，以后再多看几次。

本DOCX文档由 www.zciku.com/中词库网 生成，海量范文文档任你选，，为你的工作锦上添花,祝你一臂之力！