THE Mythical Man-Month

32周年中文纪念版

Frederick P.Brooks.Jr.
[美] 弗雷德里克·布鲁克斯/著
UMLChina翻译组 汪颖/译

人月神话

[国内实战体验精华册]

清华大学出版社

目 录

人月神话 年金

Ed Yourdon

有些书对于读者和作者像是年金，它们年复一年的分红。《人月神话》就是这么一本书。我直到在 1994 年接到 Brooks 教授的电话，才真正完全赏识它。

原因来自他的电话。电话中他说，出版商请他修订他在 1975 年第一次出版的这本书。我立刻表示了我的一点妒忌和羡慕；因为我的出版商从未叫我修订一本出版接近 20 周年的书。确实，我甚至表示了这样的意见：当代的软件工程师不会考虑阅读这本如此古老的书，因此，可能无法卖出一本。“哦，不”，Brooks 教授回答道：“《人月神话》到目前为止，每年稳定的卖出 10 000 本。”

自此书出版至今，我已读过 1975 版至少四遍；在接到 Brooks 电话后，我再次将它从书架上拿下来，重读了一遍。但这次，是有特殊目的的：实际的原因来自于他的电话，Brooks 教授告诉我，目的是发现自这本书 1975 年出版以来，计算机领域是否有有意义的事件发生。

我必须申明，这样的问题是相当难以回答的。Brooks 继续告诉我，他现在已基本离开软件工程社团，将他的大部分专业精力投身于虚拟现实领域的研究。所以，在准备再版这本书时，他想知道：什么已经

改变了，什么还没有？他的原书中哪些是正确的？哪些是错误的？哪些是不切题的？

当然，我不是向他提供同样信息的唯一的人：我的一些同僚、大量的业界领袖、作者、顾问和摇旗呐喊者都被邀请回答这个问题……。我们所有人很高兴地做了。并且，像读者所期望的一样，我们的回答被 Brooks 教授处理、分析、过滤、综合进那个非凡的新版本——它是真正的国际财富。

原书的内容仍在那儿，现在新增了四章。它们是 Brooks 教授对他的思想的反思和对我们反馈的反应。新增的第一章由恰到好处地浓缩的原书主题组成，包括可以被称为书的中心论点的东西：大型编程项目忍受管理问题，这些问题因为劳力的分配而与小项目有本质上的差别；软件产品的概念完整性是如此关键；概念完整性是困难的但可达到的。第二章小结了二十年后 Brooks 对这些主题的观点。第三章是他 1986 年首发于 IEEE Software 的经典报告：《没有银弹》的重印。最后一章是对 Brooks1986 年的断言“在未来十年，依然没有银弹”的思考。

年轻的软件工程师、吝啬的研究生以及懒惰的软件老手常请我推荐当前为止最好的软件图书。“如果我带着仅有的一本计算机书在沙漠荒岛上，”他们问：“应该是哪本书？”这是个荒谬的问题，但人们坚持要个答案。假如你真的被放逐到这样的小岛上(或者你决定躲藏到这样的地方去避免未来软件崩溃的恐惧！)，《人月神话》应该紧随着你。

作者介绍：

Ed Yourdon 是方法学大师。他的《结构化设计》、《面向对象分析》、《面向对象设计》等书两次引领了结构化分析设计和面向对象分析设计方法学的潮流。

读者感言

—关于《人月神话》20周年纪念版

读书有感——人月神话

May 29，2004

看到小女子介绍《人月神话》这本书，请不要以为是罗曼史小说或者传奇故事。

它是一本二十多年前出的、讲三十年前软件专案管理问题与经验的书，原名是 The Mythical Man-Month。其中的很多事例，现在仍存在；其中很多经验，现在拿来还很有指导意义。

我们曾经接过急迫且大型的网站建置项目，但是进展上还算顺利，这是由于系统工程师十分有经验，在分工、次序与掌控上，都做了很好的安排。

但在最近的一个朋友的经验中，却是遇到网站机制改版的问题，在时程与资源运用上不断延迟，还好是内部机制，不用向客户交代，否则蒙受的将不仅仅是金钱上的损失。

我们知道时程延迟，但原因在哪里？为何会发生？

《人月神话》的第二章，应该是这本书的精华，也是书名的由来。看完这个章节，我发现作者道出很多软件专案上的管理关键，而软件专案是不可以完全套用传统产业的管理方式的。

第一个关键 乐观

首先作者提出了一件常常发生的事情，这也是一个在我朋友身上发生的故事："程式设计师都是乐观的家伙。"而且我发现越年轻越乐观，但是无可避免的这是个年轻的产业，很少会遇到很有经验的程序设计人员。这些年轻人，即使我们告诉他这样是会有问题的，他也不会有任何改变。

所以"他们所犯的第一个错误是假设一切都会进行得很顺利"。可是没有想到一个 bug，就可能花上一天时间也无法解决。然后，就延迟了后面本来假设可以顺利进行的部分。

第二个关键 人月

人月，是我们预估和排定时程用的，但是作者提出一个前提："使用人月必须要在人力与工时可以互换的状况下，而且要当工作可以切割，投入工作的人不用沟通，人力与工时才能互换。"就是说要可以互换，才能使一个人做 30 天，与 30 个人做一天的结果一样，不然单纯用人月去估算时程，一定会有误差。

为什么呢？首先软件项目有其连续性。作者举了个例子，他说，生小孩就是要九个月，你叫多少个妈来生都是一样。第二点是因为即使工作可切割，但是需要沟通，越多人投入就需要多的沟通与教育的时间，所以一个人做 30 天的工作不可能用 30 个人做一天来完成。

所以，Brooks 定论说，在一个时程已经落后的软件项目中增加人手，只会让它更加落后。

因此，要让一个项目顺利进行，首先要有良好的时程规划，考虑所有的因素，其次，程序人员要有勇气不要妥协，一定要坚持自己预估的时程。就像是厨师一样，即使外面的客人在等着上菜，一只鸡得烤多久就要烤多久，不能因为妥协就用大火烤焦了。

外科手术团队

这个篇章看完突然想到研究所的项目管理课程，三个学分，却是我们一个学期的研究重心；四个实际运作的项目，大家分组进行。教授不断地制造项目危机，比如公司被并购、人员流失、项目形式改变和客户不断施以压力等等，但最后大家还是顺利地结案。

这是个难以忘怀的经验，组员们的不够默契在起初的确造成危机，但是，由于大家的素质还算整齐，很快就建立沟通的语言。加上大家熟悉协同科技概念与网络沟通，可以做到分时分地作业，也使得项目进行比较顺利。

但当我看完这篇外科手术团队后，我发现其实还有个成功的关键，就是同学们各有专长，所以分工很容易；又因为有选出 leader 作为掌控者，所以不会乱。大家不会做重复的事情。只要每个人按照时程交差，工作就可以顺利完成。

作者认为一个外科手术中，有操刀的大夫，有护士、麻醉师等等，各司其职。而不是像屠夫团队，每个人都得拿刀。一个团队中只有一个人操刀，其他人扮演支援的角色。这样整件事情就会出自一个脑袋，也不需要不断地沟通协调。

读这些篇章我比较有感觉，写出来跟大家分享。

最后，作者还有一个“人月”的概念很重要，这会影响到组织的公平性：一个好手，花一天可以做完一件事，但同样一件事，庸才却需要花三四天以上，甚至加班赶工也不能完成。如果照这样去计算人月，一定会差之千里。

我这几天很烦！（产品概念完整性）

2004-06-18

我这几天很烦！

一是因为现在做的项目处于测试阶段，由于一些原因，导致现在发现了很多关于模块交互方面的问题。要将这些模块“组装”成一个像样的系统，这些问题必须解决，目前只能靠自己与其他开发人员商量解决！

这个项目在设计的时候有七个人参与，应该是开发部所有的开发人员包括在内。当时先是讨论整个框架的设计，然后将各个模块分下来，一人完成一两个，再接着开会讨论。讨论来讨论去，讨论的差不多了，也就是每个人将自己设计的模块接口写成电子表格（模块内部的设计有些有文档，有些没文档，因为每个模块的设计人员基本上就是这个模块的编码人员，所有其他人不会太多关心），然后每个人就开始实现自己的模块。后来因为人手不够，又招了几个写代码的人员，分给各个模块的负责人员协助编码。事实上在真正进入编码阶段时模块的负责人只剩下三个，那么其他的人呢？两个离职，一个是公司领导，

不参与实际编码，还有一个因为考博，没时间亲自参与开发。而剩下的三个人中有一个人是这个开发小组的组长，协调开发小组的开发工作，当然也不参与实际的开发。

可以说在整个开发过程中没有一个真正的产品负责人(或者称之为总设计师)，用于负责产品的整体结构设计、各模块的交互关系、功能设计和实现方案的取舍。这个人必须十分了解整个产品及设计方案，他可以不写代码，因为维护整个产品设计方案的工作贯穿于开发全过程，工作量是很巨大的。他也就是维护产品概念完整性的那个人！

其实，除了缺乏真正的产品负责人外，存在的问题还有很多！近期我会抽时间把它整理下来，寻找问题的根源到底在哪里，以避免再犯同样的错误！

还有，我去年买的《人月神话》，当时没看懂。前段时间拿出来，我终于把它完整地看完了，而且感触颇多。

为什么《人月神话》在20多年后还能再出新版，而且又一次引起业界巨大的轰动，就是因为我们没有吸取前人在20多年前总结的经验教训，还在一次次地重犯同样的错误！

……

这个项目是一个二次开发包。完了之后紧接着有很多项目都要依赖它。二次开发包做成这样，不难预测以后依赖此开发包的项目又将使人进入新的“焦油坑”。我现在也不知道什么时候开发第二个版本，至少今年不太可能。

加班！加班！

关于我们的思考——“项目开发”及读《人月神话》有感

项目开发终于结束了。

按项目流程，我应该写一份《项目开发总结报告》。拿来“GB856T—88”标准文档，有框架指导我该写些什么，但是怎么能让这些框架协束缚了自由的思想呢？于是我决定换一种形式。

项目开发接近尾声的时候，也是最关键的时候，有人送来一本《人月神话》。我很幸运能在这个时候读这本书，因为它带给我很多思考，关于项目，关于团队，关于我们……

……

四、完美与放弃

在《人月神话》中有一段被截取，称为“程序员的苦与乐”，在网上广为流传。Brooks 大师用简短的篇幅，描绘出整个程序世界的苦与乐。而在这次项目开发中，我有两个苦恼体现的比较明显。

一是追求完美。“因为计算机是以这样的方式来变戏法的：如果口语中的一个字符、一个停顿，没有与正确的形式一致，魔术就不会出现(现实中，很少的人类活动要求完美，所以人类对它本来就不习惯)。实际上，我认为，学习编程最困难的部分，是将做事的方式向追求完美的方向调整。”

看来我再次做出了正确的选择。我是个追求完美的人，而且我也一直认为程序员就应该有这种本性。在写程序的时候，甚至对一个变

量名我都要反复斟酌，直到选择一个我认为最合适的。我并不认为这是在浪费时间：一是有助于对程序的理解和维护，好的程序本身就是注释；二是可以减少错误发生的可能。这次开发因为时间短，我尝试采用设计→编码→编译的方式来写程序，经常把一个几千行代码的模块写完之后才开始调试。这种方式效果不错，因为对设计考虑得比较充分，设计上没出现什么问题，基本上都是一些拼写上的错误。不过有一个错误却令我苦恼了很久。因为一个结构的成员变量名与函数参数的变量名一样，而这个参数又在多处使用。写的时候，也可能是拷贝代码的时候，就很容易把结构名给忘记或多加了一个结构名，而这时又不会有语法上的错误。吸取了这次教训，我把整个程序检查了一遍，并做了一些修改。这段程序我参考了一位大师的原型，令我欣慰的是，他的下一个版本和我的程序一样做出了同样的一些修改。

另一个苦恼来自“设定目标、供给资源和信息提供。编程人员很少能控制工作环境和工作目标。”《人月神话》后面的章节还提到：“结构师获得了所有创造发明的快乐，剥夺了实现人员的创造力”；“实现同样是一项高级的创造性活动。具体实现中创造和发明的机会，产品会因为指定了外部技术说明而大为减少，相反创造性活动会因为规范化而得到增强，整个产品也一样。”告诉我们，程序员要学会放弃一部分乐趣，整个项目也一样。这是我读《人月神话》感触最深的一点。

设计网络认证的时候，本打算用一种简单的认证方式实现，因为这不是重点。Y 提出用证书加自己写的 Socket 类实现认证的通讯过程。这是一个很好的创意，但 VC 写出类无法在 Delphi 里使用，BCB 也不行。封装，回调……大家都搞晕了，只剩下两个字：“放弃”。

由于时间比较紧，在产品的功能上我们也放弃了一些我们认为是很有创意的，但需求中没有提到的部分。后面的开发中也有类似的问题，每个人都按自己的喜好调用和提供接口，可以说是八仙过海，各显神通。问题主要在于交流。

五、善于交流

……

2．交流方式

人月之所以不能成为神话，正是因为增加人手的同时也增加了人与人之间的沟通成本。

我们在项目一开始就通过 QQ 群组进行沟通，后来又搭建了企业内部协作平台，而且还有不定期的会议。与大师所说的三种交流途径（非正式途径、会议、工作手册）不谋而合。但是，执行的效果却不很理想。是大师说的不对？不是。是我们没有利用好这几种途径。我们很好地利用会议，解决很多细小方面的误解；过多地使用 QQ 群组，一是在登陆 QQ 的时候消息太多而没看仔细，二是这种方式针对小的误解是很有效的，但对于一些系统的全面理解必须通过文档；很少使用协作平台，没有仔细阅读文档的习惯，也没有写出一个完整文档的习惯。

即使在 QQ 群组的交流中，我们也没有把问题表述清楚。经常看到："某某，你那里有问题？""哪里？什么问题？"接着有人回答："不可能啊？"什么事情都可能发生的，只要能满足条件。域名都会不易而飞，还有什么是不可能的呢？

……

九、结束语

本文只是我在项目开发和读了《人月神话》之后的一点感想，对于本文一些观点的正确与否欢迎与您一起讨论。本文只是按照项目流程，挑主要的部分叙述了我在开发中体会到的一些感想。这次开发让我学到太多太多的东西，将使我终身受益。还有很多的想法，如团队建设、整体设计、开发模式……希望以后能与大家多多交流，相互学习。

《人月神话》这本书我推荐大家都去读一下。它不是良药，不能为你解决实际工作中的问题，但它可以给你带来很多思考，让你变得更加成熟。软件工程是为开发软件服务的，标准不是目的，只是手段。《人月神话》没讲标准，但它为怎样做好一个项目提供了一些启发。它会增强你的自信心，当有人反驳你的观点的时候，你可以告诉他："Brooks 大师如是说！"

我的"人月神话"

……

"人月神话"这个题目来自于《人月神话》一书，内容也来自于《人月神话》一书，只不过换成了我所经历的。

我编写代码已经数年了。说到编写代码的能力，我还是比较自信的。但是，领导终究是领导，始终会有没有学过管理或者尝试以流水线理论来主导软件开发的领导。在国内，这样的领导并不少见，其共同之处在于：目标产品是定下期限要完成的，所有的东西是可以以人

手定量的，而人员配备“按照正常情况足够满足开发进度”。在这一前提下，形形色色的变种会出现。

有些是不能明确说出功能列表；有些则是没有能力知道开发中可能遇到的障碍点在哪里；有些则不知道团队人员的真正能力；更有些是集以上之大成。在有些时候，我也是其中的一员。

说到这里，我要简单阐释一下：我们所有的进度都是以“人月”代码产量来衡量的，而实际情况是，增加“人”并不能缩短“月”的量。

一个目标产品本身能有多大的代码量大致不会和预算的相差很多。这是我的经验，当然也有一些连代码量也估算不准的 LEADER。我们通常最终会将代码量分解到每个模块，并且根据程序员的工作能力来分配进度要求。在很多情况下，遇到进度失衡的时候，第一反应是增加人手。但是事实上增加人手的项目只有不到 10%的能准时完成。很多情况下，增加进去的人手并不能真正进入工作，因为模块已经无法细分一小块出来给新加入的人手。

而人月代码产量本身就不是一个固定的值。我的最高编写记录可以达到 1600 行/天。那么，真的就是 32000 行/月了吗？不！更多时刻的代码产量在 200-300 行/天。也有很多一个算法就花费 1 天。变得只有 100 行/天的情况。进行根据最近 3 年的状况，5000 行/3 月是比较客观的量。这是 C/C++的速度，也是我的速度，其他程序员有这样的效率吗？真正能超过的并不多见。即使是这样的代码效率，也并不适合将计算进入商业产品的进度进行考虑，因为很多难点并不是因为降低人月代码产量就能够攻克的。

我本人目前比较倾向的时间分配，也是比较真实的时间分配，没有难点的时间分配，如下：

20% 代码编辑

30% DEBUG

30% 文档

20% 保留时间

这就说明即使在没有已知难点的状况下，有20%的保留时间仍然有必要。因为很有可能1个小小的数学逻辑就能让你忙上一天。这并不是不专心，是疏忽导致的，而从来就没有人能避免疏忽。

了解了这个神话，我们就可以采取主动行动。

1. 首先，不要低估任何一个产品的难度，难度估计得高点总是没有错的。(我曾经犯过多次这样的错误)这样，在确定任务进度前争取更多的时间。

2. 很显然，既然有可能在任意时刻发生问题，为什么不提前多干点儿呢？很少有人愿意这样。但是我的经验是一定要提前多干。在最近的2个项目中，我都是提前很多时候完成了大部分的工作。90%的东西完成了，而产品交付时间则剩下1个月。眼看可以轻松了，却仍然忙着攻克最后的难点，到了最后一天才真正完成任务。险得很！按照时刻表完成进度的程序员都一定会翻船。

《人月神话》中写道好的程序员可能效率比糟糕的程序员高 10倍。在我的人月神话中确实有这样的例证。当时团队中就有一天无法完成一个极度简单功能的PROGRAMMER，但是在人月理论中，这样的人也照样要占着进度表的一条。