《人月神话》读后感

姓名：薛晓波　　学号：MF1932216

一、前记

早在我本科阶段的学习中，就曾在多门软件工程专业知识课程中听说过《人月神话》这本久负盛名的著作。然而由于曾经的年轻与浅薄，沉迷于各种“高大上”的编程技能和框架技术，而对于软件工程管理相关的知识往往不屑一顾，因此尽管曾有多位老师力荐，也还一直没有去阅读这本堪称封神的著作。

然而我们总说，缘，妙不可言，作为一名软件工程专业的学生，如同命中注定一般，有些事情是终究无法绕过去的，在软件工程管理这门课程上，它作为作业提上了我的读书日程。庆幸的是，我已不如当初的青涩幼稚，在逐渐深入的学习和实践中让我认识到软件工程并非是单纯的以技术为王的软件开发，在过去的几天时间里我一边回想自己曾经参与过的各种项目或工作，一边翻阅这本著作，深感作为一本经久不衰的经典，确实名不虚传，即使书中尚有许多未理解的内容，仍然觉得醍醐灌顶，大开眼界。

二、作者与背景

由于软件开发、管理等技术的快速更新，在软件领域很少能有像《人月神话》封神的著作畅销几十年而经久不衰。随着软件规模的越来越大，新兴互联网软件的出现，《人月神话》中的很多观点被实践所证明，甚至成为了不会随时代变迁而改变的真理。事实上，我建议每个软件行业的从业人员，无论是软件开发人员，还是项目经理，再或者是架构师，系统分析师等，都该认认真真去品读此书。有人会讲，你之所以觉得大开眼界，只不过是因为你只是个毫无经验的学生而已，其他人未必如此。没错，倘若一个从业多年的资深项目经理初读此书不以为意，那只是因为该书所述的大部分经验与建议，都早已成为软件开发与项目管理领域的常识。由此更能见得作者若干年前的真知灼见。说到作者Brooks，简单介绍一下（以下内容引自维基百科，更多资料可自行搜索查看）：

> Frederick Phillips Brooks, Jr. (born April 19, 1931) is a computer architect, software engineer, and computer scientist, best known for managing the development of IBM's System/360 family of computers and the OS/360 software support package, then later writing candidly about the process in his seminal book The Mythical Man-Month.Brooks has received many awards, including the National Medal of Technology in 1985 and the Turing Award in 1999.

由此，我们便不难理解《人月神话》的写作背景，即作者Brooks在1956~1965 之间，实际领导了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。

三、感悟

关于本书，即使很多没读过的人也能说出其中知名度最高的关于人月神话的观点。之前还在知乎上看过有人举了一个例子：一个人生孩子需要十个月，两个人去生孩子同样需要十个月。这一比喻虽不完全恰当，却也大致说出点内容。事实上，作者在书中这样来描述人月神话：软件开发项目常以人月来衡量工作量，这种度量暗示着人手和时间是可以互换的。这种“人多力量大”的想法是一种一厢情愿的虚妄神话，布鲁克斯法则：向滞后的软件项目追加人手会使得进度更迟缓（Adding manpower to a late software project makes it later）。关于这点，作者解释的为：向软件项目中增派人手从三个方面增加了项目必要的总体工作量：任务重新分配本身和所造成的工作中断；培训新人员；额外的相互沟通。同样，作者也向项目管理人员提供了相应的解决方法：

> 重新安排进度。我喜欢P.Fagg，一个具有丰富经验的硬件工程师的忠告：“避免小的偏差（Take no small slips）”。也就是说，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。

> 削减任务。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后续成本非常高时，这常常是唯一可行的方法。项目经理的相应措施是仔细、正式地调整项目，重新安排进度；或者是默默地注视着任务项由于轻率的设计和不完整的测试而被剪除。

在三四十年前，作者便以他丰富的项目经验和敏锐的观察力、判断力提出这样创造性的论断，确实令人折服。时至如今，关于人力和时间之间的关系并非呈现线性关系，也不能采用人月作为生产率的衡量标准这一结论已得到普遍的认同。当然，作者的论断以及Brooks准则也并非严格成立：在《20年后的人月神话》一文中，作者给出了Boehm的模型和数据以验证之前的说法太过绝对，但是，“这些细致的研究使“异常简化”的Brooks准则更加实用。作为平衡，我还是坚持这个简单的陈述，作为真理的最佳近似，以及一项经验法则——警告经理们避免对进度落后的项目采取的盲目、本能的修补措施。”

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。

许多人都觉得以上观点就是《人月神话》的全部了，并非这样。《人月神话》只是书中一个章节，Brooks准则也只是书中的观点之一。本书所述内容远非于此，它涉及到软件开发与项目管理过程中的方方面面，从开发团队人员配置，到资源的合理配置，到项目文档的撰写以及其他许多内容。实际上，我认为本书取名《人月神话》在某种意义上存在一定的误导性：在《20年后的人月神话》中，作者提炼出了核心观点：**概念完整性和结构师**。

> **概念完整性。**一个整洁、优雅的编程产品必须向它的每个用户提供一个条理分明的概念模型，这个模型描述了应用、实现应用的方法以及用来指明操作和各种参数的用户界面使用策略。用户所感受到的产品概念完整性是易用性中最重要的因素。

> **结构师。**结构师负责产品所有方面的概念完整性，开发用于向用户解释使用的产品概念模型，概念模型包括所有功能的详细说明以及调用和控制的方法。结构师是这些模型的所有者，同时也是用户的代理。

> 在不可避免地对功能、性能、规模、成本和进度进行平衡时，卓有成效地体现用户的利益。

> **将体系结构和设计实现、物理实现相分离。**为了使结构师的关键任务更加可行，有必要将用户所感知的产品定义——体系结构，与它的实现相分离。体系结构和实现的划分在各个设计任务中形成了清晰的边界，边界两边都有大量的工作。

> **体系结构的递归。**对于大型系统，即使所有实现方面的内容都被分离出去，一个人也无法完成所有的体系结构工作。所以，有必要由一位主结构师把系统分解成子系统，系统边界应该划分在使子系统间接口最小化和最容易严格定义的地方。每个部分拥有自己的结构师，他必须就体系结构向主结构师汇报。显然，这个过程可以根据需要重复递归地进行。

所以从人个人观点出发，本书用其中一章的标题命名全书，实则不太妥当，哪怕叫做《项目管理规范》都更能做到统筹兼顾。

最近应课上荣国平老师的要求在学习发布与“中国大学mooc”网上的在线课程《DevOps导论》，在第一章的概论中，老师就提到了《人月神话》，也提到了Brooks总结的软件工程本质性工作的四大难题。我认为全书最精华的部分就是第16章，因为它把我在开发学习隐约的体会用精炼又充满思辨的语言表达出来，不要忘记，这是在40年前。这里简述一下该章的论证思路：

- 所有软件活动可以分成两种任务：根本任务和次要任务，根本任务是打造构成抽象软件实体的复杂概念结构，次要任务是使用编程语言表达这些抽象实体，在空间和时间限制下将它们映射成机器语言。
- 所谓根本（essence）是指软件特性中的固有的困难，所谓次要（accident）是指出现在目前的生产中，但并非与生俱来的困难。那为什么构造异常复杂的抽象概念结构是根本问题呢？这里涉及现代软件系统中的几个无法规避的内在特性：复杂度、一致性、可变性和不可见性。
- 复杂度：计算机存在很多种状态，这使得构思、描述和测试都非常困难。软件系统的状态又比计算机的状态多若干个数量级。同样，软件实体的扩展也不仅仅是相同元素的重复添加，而必须是不同元素实体的添加。大多数情况下，这些元素以非线性递增的方式交互，因此整个软件复杂度要比非线性增长多得多。
- 一致性：物理学家坚信，必定存在着某种通用原理，或者在夸克中，或者在统一场论中。软件工程师却无法从类似的信念中获得安慰，他必须掌握的很多复杂度是随心所欲、毫无规则可言的，来自若干必须遵循的人为惯例和系统。许多情况下，因为是开发最新的软件，它必须遵循各种接口。另一些情况下，软件的开发目标就是兼容性。所以很多复杂性来自保持与其他借口的一致性，对软件的任何再设计，都无法简化这些复杂特性。
- 可变性：软件实体经常会遭受到持续的变更压力。部分原因是系统中的软件包含了很多功能，而功能是最容易感受到变更压力的部分。另外的原因是软件可以很容易地进行修改——它是纯粹思维活动的产物，可以无限扩展。简言之，软件产品扎根于文化的母体中，如各种应用、用户、自然及社会规律、计算机硬件等。后者持续不断的变化着，这些变化无情地强迫着软件也随之变化。
- 不可见性：软件的客观存在不具有空间的形体特征。当我们试图用图形来描述软件结构时，发现它是很多相互关联、重叠在一起的图形。这些图形可能代表控制流程、数据流、依赖关系、时间序列和名字空间的相互关系等。在上述结构上建立的概念控制的一种方法是强制将关联分割，直到可以层次化一个或多个图形。除去软件结构上的限制和简化方面的进展，软件仍然保持着无法可视化的固有特性，从而剥夺了一些具有强大功能的概念工具的构造创意。

读完全书，我惊叹于作者在实际工作中能够提炼出如此多的普遍性真理，更令我惊讶地是这些如今仍然熠熠生辉的论述来自于四十年前，其间计算机硬件性能的提升诚如摩尔定律所言，而软件开发的生存率却远达不到这般飞速，原因正是如上所说之大型软件工程中的根本问题，我相信这本书将在我未来可能数十年的软件开发生涯中长久地影响着我，给予我指导。