## 《人月神话》阅读笔记

### 焦油坑

在这一章节作者阐述了过去几十年的大型系统就像一个焦油坑，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。表面上看单独的一个问题都很好解决，但是当他们交织在一起的时候团队的行动就会变的越来越慢。然后为了阐述这种情况的原因，作者向我们介绍了编程这个职业的乐趣，阐述了编程这个职业其实首先是一种创造事务的快乐就像小孩用粘土为爸爸办公室捏制铅笔盒没有本质的区别，但是程序毕竟和诗歌不同，程序能独立的产生输出，能够打印结果，绘制图形，发出声音。阐述完编程这个职业的乐趣后，又向我们展示了编程这个职业的苦恼，首先编程必须追求完美，任何异地地方的错误都有可能导致结果出现问题；其次，编程人员很少能控制工作环境和工作目标；再次，对于系统编程人员，需要依靠其他人的代码，但是其他人的程序往往会有缺陷，难以理解和修改；最后就是概念设计是很有趣的，但是，寻找琐碎的bug确是一项重复的工作；最后，也是相当无奈的就是当产品完成的时候，产品却显得过时了。

### 人月神话

这一章节作者首先提出：缺乏合理的时间进度是造成项目滞后的首要原因。然后提出了出现缺乏合理时间进度的原因：乐观主义，人月，系统测试，空泛的估算，重复估算进度。

乐观主义导致系统安排背后的第一个假设是：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。但是编程设计不同于硬件或者工程上的其他设计，人的思考总会是有缺陷的，因此总会有bug，从而导致一切正常的概率变得非常小。

人月指的是在估计和进度安排中使用的工作量单位。但是在进度中使用人月作为工作量单位是及其不合理的。人月只适用这样一种情况：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互。这在系统开发中近乎不可能。

在时间进度中，系统测试进度的安排常常是编程中最不合理的部分。通过作者的研究，大部分项目没有为系统测试安排足够的时间。这样就导致，有些项目直到项目发布的时候才有人发现进度上的问题。

然后则是则是空泛的估计，很少的数据支持，加之完全借助软件经理的直觉，这样的方式很难产生出健壮可靠的和规避风险的估计。

最后一个因素则是重复调整进度，当项目延期的时候，项目经理不断的增派人手（相信以人月的方式解决问题），但是根据Brooks的原则，向落后的项目增加人手，只会使进度更加落后。

### 外科手术队伍

作者在这一章节引用了Harlan Mills的提议：建议大型项目的每一个部分由一个团队解决，但是该队伍以类似外科手术的方式组建，而非一拥而上。当面对几百人的大项目的时候，扩建的成功依赖于这样一个事实，即每个部分的概念完整性如果想要得到彻底的提高，决定设计的人员是原来的七分之一或者更少。作者认为整个系统必须具备概念上的完整性，要有一个系统结构师从上至下进行所有的设计。

### 贵族专制、民主政治和系统设计

在本章作者指出：概念的完整性要求设计必须由一个人，或者非常少数互有默契的人员来实现。创造性的工作并非这些人独有的，在给定体系结构下的设计实现，同样需要同编制技术说明一样的创造性、同样的思路和活跃的才华。

### 画蛇添足

在本章作者提出了结构师的交互准则和机制，如结构师应尽早和持续与客户进行交流以使他能获得更好的成本意识，这也让开发人员获得设计的信心，并且不会混淆了他们之间的工作。同时作者还提出了一些其他建议。

作者同时指出，对于第一个项目较为成功的结构师而言，他们的第二个系统将是他们所设计最危险的系统。因为他们往往会过分设计第二个系统，向系统添加过多的修饰功能和想法。这就造成了画蛇添足。作者提出，为了避免画蛇添足，结构师可以有意识关注那些系统的特殊危险，运用特别的自我约束准则，来避免功能上的修饰，并根据系统基本理念的变更舍弃一些功能。

### 贯彻执行

世界的规律就是一切往着混乱的方向发展，所以我们往往需要耗费很大的精力将上层的决策向基层贯彻执行。作者首先在这一章提出：文档的规格说明—手册，是一个非常重要的工具，它应该包括用户能在界面上看到的一切，同时还要避免描述用户看不见的事物。虽然有了持久化的文字和代码，语言的交流会更有利于概念的确定，不至于在不同的人那里产生歧义，所以正式的会议和随意的交流都是必须的。书中还提出了多重实现、电话日志和产品测试的方法。

### 为什么巴比伦塔会失败

巴比伦塔的制造是一个神话故事，但是其中的道理却对今天人们的协作有着重要的启示。软件系统的开发完全通过计算机执行，为什么还是很少有远程协作的企业，这是因为远程协作很容易导致交流的缺失。大型的软件项目开发需要团队中的每个人能及时了解到整个团队在做些什么，这就需要经常的交流。交流的方式可以通过非正式的电话、网络，也需要正式的会议和工作手册。

### 胸有成竹&削足适履

通过对一些项目的数据的分析，作者得出了：即使不考虑相互沟通，开发人员仅仅回顾自己以前的工作的情况下，这些数字仍然显示出工作量是规模的幂函数。所以需要“削足适履”，即进行规模控制，和对空间预算做出预判。

### 提纲挈领

本章作者提出：书面记录决策是必要的，只有记录下来，分歧才会明朗。所以从一开始就应该认识到文档的普遍性和重要性，那么就可以将文档作为友好的工具利用起来。

### 未雨绸缪

试验性开发确实会造成精力的消耗，或许大量的测试方案最终还会被舍弃，但是我们必须这样做。实际上如果不进行方案的实验，正式的开发反而可能遭遇返工和混乱的拆补，会严重分散重新开发人员的精力和信心，甚至影响用户对产品的信任。这世界上唯一不变的就是变化本身，我们必须有未雨绸缪的能力，对未来可能产生的变化做出提前的设计，甚至对组织架构也需要进行提前的计划来规避变化造成的风险。除了开发，运营维护也需要适应变化，这就需要我们提高代码的质量和可读性，完善测试过程，保证系统在调整的过程中能够尽量少地引入更多问题。前人的智慧告诉我们，缺陷是永远存在的，我们需要通过质量管理放缓系统混乱度的提高。

### 干将莫邪

软件的开发离不开工具，从需求的分析，到系统的设计，到程序的编码，到  
构建、测试、发布和维护，我们要善于利用工具来提高我们工作的效率和质量。

### 整体部分

产品的概念完整性在使它易于使用的同时，也使开发更容易进行以及bug更不容易产生，因此我们要使用上述的方法来保障产品的概念完整性。面向对象编程的“封装”思想和结构化编程的“精化”思想对于整个软件开发过程的各个粒度同样适用。整体的顺利运行离不开各个组成部分的优化。编码时各个信息隐藏的模块需要完成各自的任务，再通过接口互相配合。测试时需要从最小的单元测试开始，每一粒度都测试完全时，整个系统的运行才有保证。当系统出现问题，需要找到问题的发生点，这时就需要将问题在不同的模块和粒度上分解测试，最终找到问题的症结。

### 没有银弹

作者指出：人们希望找到一种方法能完美解决这个问题。然而，经过多年的探索，我们发现，这样的方法根本不存在，于是催生了软件工程这个学科来针对软件过程中的每个细分进行方法论的指导。在解决这个问题的过程中，人们也做了很多努力，提出了高级编程语言、面向对象编程、人工智能、专家系统、图形化系统、程序验证、工作站等方式，这些对于软件工程的发展都产生了重要的影响。人月是一个神话，现如今软件工程却是真实地在解决软件过程中的问题，提高软件产品的质量。研究人员和实践人员的不断探索或许永远无法一劳永逸地解决所有问题，但是从中积累地经验却能够有效地指导我们更好地应对大型软件系统的实现与管理。