文章
姚 鑫 · 四月 11, 2021 阅读大约需 16 分钟

人月神话

人月神话

焦油坑

1. 编程系统产品开发的工作量是供个人使用的，独立开发的构件程序的9倍。我估计软件构件产品化引起了3倍工作量，将软件构件整合成完成系统所需要的设计，集成和测试又强加了3倍工作量，这些高成本的构件在根本上是互相独立的。
2. 编程行业“ 满足我们内心深处的创造渴望和愉悦所有人的共有情感 ”，其提供了五种乐趣：

- 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，不断学习的乐趣。
- 纯粹的思维活动。

3. 同样，这个行业具有一些内在固有的苦恼：

- 将做事方式调正到追求完美是学习编程的最困难的部分。
- 由其他人设定目标，并且必须依靠自己无法控制的事物，权威不等同于责任
- 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，软件项目能收敛得快一些，然后，情况却是越接近完成，收敛得越慢。
- 产品在完成前总面临着陈旧过时的威胁；只有实际需要时，才会用到最新的设想。

人月神话

1. 缺乏合理的时间进度是造成项目滞后的主要原因，它比其他所有因素的总和影响还大。
2. 所有的编程人员都是乐观主义者“ 一切都将运作良好 ”
3. 由于编程人员通过纯粹的思维活动来开发，我们期待在实现过程中不会碰到困难。
4. 但是，我们的构思本身是有缺陷的，因此总会有bug。
5. 围绕着成本合算的估计技术，混淆了工作量和项目进度。人月是危险和带有欺骗性的神话，因为它暗示了人员数量和时间可以互相替换的。
6. 在若干人员中分解任务会引发额外的沟通工作量-培训和相互沟通。
7. 关于进度安排，我的经验是1/3计划，1/6编码，1/4构件测试以及1/4系统测试。
8. 作为一门学科，我们缺乏数据估计。
9. 我们对自己的估计技术不确定，因为在管理和客户的压力下，我们常常缺乏坚持的勇气。
10. Brooks法则：为进度落后的项目增加人手，只会使进度更加落后。
11. 向软件项目中增派人手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：额外的互相沟通。

外科手术队伍

1. 同样有两年讲演而且在收到同样培训的情况下，优秀的专业程序员的生产率是较差的程序员的10倍。
2. 小型，精干队伍是最好的-思绪尽可能少。
3. 实际上，绝大数大型编程系统的经验显示，一拥而上的开发方法是高成本，速度缓慢，低效的，开发出的产品无法进行概念上的集成。
4. 一位首席程序员，类似于外科手术队伍的团队架构提供了一种方法-既能获得由少数头脑产生的产品完整性，

又能得到多位协助人员的总体生产率，还彻底减少了沟通的工作量。

贵族专制，民主政治和系统设计

1. “ 概念完整性是系统设计中最重要的考虑因素。 ”
2. “ 功能与理解上的复杂程度的比值才是系统设计的最终测试标准 ”，而不是丰富的功能。
3. 为了获得概念完整性，设计必须由一个人或者具有共识的小型团队来完成。
4. 将体系结构方面与具体实现相分离是获得概念完整性的强有力方法。
5. 如果要得到系统概念上的完整性，就必须有人控制这些概念。实际上是一种无序任何歉意的贵族专制统治。
6. 概念上统一的系统能更快地开发和测试。
7. 体系结构，设计实现，物理实现的许多工作可以并行。

画蛇添足

1. 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。
2. 结构师如何成功地影响实现：

- 牢记开发人员承担创造性的实现责任；结构师只能提出建议。
- 时刻准备着所指定的说明建议一种实现的方法，准备接受任何其他可行的方法
- 对上述建议保持低调和平静。
- 准备对所建议的改进放弃坚持。
- 听取开发人员在体系结构上改进的建议。

3. 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。

贯彻执行

1. 即使是大型的设计团队，设计结果也必须由一个或两个人来完成，以确保这些决定是一致的。
2. 必须明确定义体系结构与先前定义不同的地方。
3. 处于精确性的考虑，我们需要形式化地设计定义，同样，我们需要记叙性定义来加深理解。

为什么巴比伦塔会失败

1. 巴比伦塔项目的失败是因为缺乏交流以及交流的结果 - 组织。
2. “ 因为左手不知道右手在做什么，从而进度灾难，功能的不合理和系统缺陷纷纷出现。 ”由于存在对其他人的各种假设，团队成员之间的理解开始出现偏差。
3. 团队应该以尽可能多的方式进行互相之间交流，非正式地进行简要技术陈述的常规项目会议，共享的正式项目工作手册。
4. 项目工作手册 “ 不是独立的一篇文档 ” 它是对项目必须产生的一系列文档进行组织的一种结构。
5. 项目所有的文档都必须是该结构的一部分。
6. 需要尽早和仔细地设计工作手册结构。
7. 事先制定良好结构的工作手册 “ 可将后来书写的文字放置在合适的章节中 ” 并且可以提高产品手册的质量。
8. 每一个团队成员称该了解所有的材料。
9. 实时更新是至关重要的。
10. 共享的电子手册是能达到所有这些目标的，更好的，更加低廉的，更加简单的机制。
11. 团队组织的目标是为了减少必须的交流和协作量。
12. 组织中的交流是网状么，而不是树状结构， 因此所有的特殊组织机制（往往体现为组织结构中的虚线部分）都是为了进行调整，以克服树状组织结构中交流缺乏的困难。
13. 每个子项目具有里那个领导角色 -
    产品负责任，技术主管或结构师。这两个角色的职能有很大的区别，需要不同的技能。
14. 两个角色的任意组合都可以是非常有效的。

胸有成竹

1. 仅仅通过对编码部分时间的估计，然后乘以其他部分的相对系数，是无法得出对整项工作的精确估计的。

2. 构建独立小程序的数据不适用于编程系统项目。
3. 当使用适当的高级语言时，程序编制的生产率可以提高五倍。

削足适履

1. 除了运行时间意外，程序所占据的内存空间也是主要的开销。
2. 从系统整体出发以及面相用户的态度是软件编程管理人员最重要的职能。
3. 为了取的良好的空间-
   时间折中，开发队伍需要得到特定的某种语言或机型的编程技能培训，特别是在使用新语言或者新机器时。
4. 编程需要技术积累，每个项目需要自己的标准组件库。
5. 精炼，充分和快速的程序往往是战略性突破的结果，而不仅仅是技巧上的提高。
6. 这种突破常常是一种新型算法。

提纲挈领

1. 在一片文件的汪洋中，少数文档成为了关键的枢纽，每个项目管理的工作都围绕着他们运转。
2. 对于计算机硬件开发项目，关键文档是目标，手册，进度，预算，组织结构图，空间分配以及机器本身的报价，预测和价格。
3. 对文档进行规范化。
4. 项目经理的基本职责是使每个人都向着相同的方向进行。
5. 项目经历的主要日常工作是沟通，而不是做出决定，文档使各项计划和决策在整个团队范围内得到交流。

未雨绸缪

1. 第一个开发的系统对于大多数项目并合用。它可能太慢，太大，而且难以使用，或者三者兼而有之。
2. 系统的丢弃和重新设计可以一步完成，也可以一块块地实现，但这是必须完成的步骤。
3. 开发人员交付的是用户满意陈谷，而不仅仅是实际的产品。
4. 用户的实际需要和用户感觉会随着程序的构建，测试和使用而变化。
5. 软件产品易于掌握的特性和不可见性，导致它的构建人员面临着永恒的需求变更。
6. 目标上的一些正常变化无可避免，事先为他们做准备总比假设他们不会出现要好的多。
7. 高级语言的使用，编译时操作，通过引用的声明整合和自文档技术能减少变更引起的错误。
8. 程序员不愿意为设计书写闻到那股，不仅仅是因为惰性，更多的是源于设计人员的踌躇 -
   要为自己尝试性的设计决策进行辩解。
9. 为变更组建团队比为变更进行设计更加困难。
10. 只要管理人员和技术人才的天赋允许，老板必须对他们的能力培养给予极大的关注，使管理人员和技术人才具有互换性，特别是希望在技术和管理角色之间自由地分配人手的时候。
11. 对于一个广泛使用的程序，其维护总成本通常是开发成本的40%或更多。
12. 维护成本受用户数目的影响。用户越多，所发现的错误也越多。
13. 缺陷修复总会以20%-50%的几率引入新的bug。
14. 每次修复之后，必须重新运行先前所有的测试用例，确保系统不会以更隐蔽的方式被破坏。
15. 所有修改都倾向与破坏系统的架构，增加了系统的混乱程度。即使是最熟练的软件维护工作，也只是延缓了系统退化到不可修复的混乱状态的进程，以致必须要重新进行设计。

干将莫邪

1. 抛开理论不谈，一次分配给某个小组的连续的目标时间快被证明是最好的安排方法，比不同小组的穿插使用更为有效。
2. 主程序库应划分为 1 一系列独立的私有开发库 2 正处在系统测试下的系统集成子库 3
   发布版本。正式的分离和进度提供了控制。
3. 高级语言不仅提高了生产率，还改进了调试，bug更少，而且更容易寻找。

整体部分

1. 许许多多的失败完全源于那些产品未精确定义的地方。
2. 有时必须回退，推翻顶层设计，重新开始。
3. 系统调试所花费的时间会比预料的更长。

4. 系统调试的困难程度证明了需要一种完备系统化和可计划的方法。
5. 开发大量的辅助调试平台和测试代码是很值得的，代码量甚至可能有测试对象的一半。
6. 必须有人对变更和版本进行控制和文档化。

祸起萧墙

1. 一天一天的进度落后比起重大灾难更难以识别，更不容易防范和更加难以拟补。
2. 李成碧必须是具体的，特定的和可度量的事件，能进行 清晰的定义。
3. 慢性进度偏离是士气杀手。
4. 进取对于杰出的软件开发团队是不可缺乏的必要品德。
5. PERT饿准备工作是PERT图使用中最有价值的部分。
6. 第一份PERT图总是很恐怖。
7. 每个老板同时需要采取行动的异常信息以及用来进行分析和早期预警的状态数据。
8. 状态的获取是困难的，因为下属经理有充分的理由不提供信息共享。
9. 老板的不良反应肯定会对信息的完全公开造成压制，相反，仔细区分状态报告，毫无惊慌地接收报告，决不越俎代庖，将能鼓励诚实的汇报。

另外一面

1. 对于软件编程产品来说，程序向用户所呈现的面貌- 闻到那股，与提供给机器识别的内容同样重要。
2. 即使是完全开发给自己使用的程序，描述性文字也是必须的，因为他们会被用户-作者所遗忘。
3. 培训和管理人员基本上没有向编程人员成功地灌输对待闻到那股的积极态度 -
   文档能在整个生命周期对客服懒惰和进度的压力起促进和激励作用。
4. 大多数闻到那股只提供了很少的总结性内容。必须放慢脚步，稳妥地进行。
5. 流程图是被吹捧的最过分的一种程序文档。
6. 如果这样，很少有程序需要一页纸以上的流程图
7. 程序员改人员所使用的文档中，除了描述事件如何，还应阐述它为什么那样，对于加深理解，目的是非常关键的，即使是高级语言的语法，也不能表达目的。
8. 软件系统可能是人类创造中最错综复杂的事物。
9. 软件工程的焦油坑在将来很长一段事件内仍然会使人们举步维艰，无法自拔。

- 从程序到编程产品至少是开发程序时间的三倍，需要有完备的文档，每个人都可加已使用，修复和扩展。

职业的乐趣

- 这种快乐是一种创造事物的纯粹快乐，特别是自己进行设计。
- 其次，这种快乐来自于开发对他人有用的东西。内心深处。我们期望我们的劳动成果能够被他人使用，并能对他们有所帮助。
- 第三，快乐来自于整体过程体现出的一股强大的而魅力，并收到了预期的效果。
- 持续学习的快乐，来自于这项工作的非重复特性，人们所面临的问题总有这样那样的不同，因而解决问题的人可以从中学习新的事物。
- 程序员就像诗人一样，几乎仅仅在单纯的思考中工作。程序员凭空地运用自己的想象，来建造自己的“ 城堡 ”。

职业的苦恼

- 苦恼来自于追求完美。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。
- 其次，苦恼来自由他人来设定目标，供给资源和提供信息。
- 对于系统编程人员而言，对其他人的依赖是一件非常痛苦的事情。他依靠其他人的程序，而这些程序往往设计的并不合理，实现出拙劣，发布不完成整，文档记录很糟糕。
- 寻找琐碎的bug是一项重复行的劳动，往往枯燥沉闷。

人月

- 用人月作为衡量工作的规模是一个危险和带有欺骗性的神话。
- 人数和时间的互换仅仅适用于，他们之间不需要相互交流。
- 沟通所增加的负担：培训和相互交流。在一个不可分解的任务时，人数的增加反而会使得项目时间增加。

- 软件开发本质是一项系统工作 - 错综复杂关系下的一种实践，沟通，交流的工作量非常大，添加更多的人手，实际上延长了而不是缩短了时间进度。

系统测试

- 软件任务的进度安排。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，所有的构件完成。
- 不为系统测试安排足够的时间，简直就是一场灾难。

进度灾难

- 向进度落后的项目增加人手，只会使得进度更加落后。
- 项目的时间依赖于顺序上的限制
- 人员的最大数量依赖于独立子任务的数量

问题

- 一流人才组成的小型，精干的队伍，而不是那些几百人大型团队，这里的“ 人 ”当然暗指平庸的程序员
- 优秀的程序员和较差的程序员之间生产率的差异，实际测量出的差异领人吃惊，最好的和最差的表现在生产率上平局能为10：1，在编程速度和空间上具有5：1的惊人差异
- 简言之，20000美元/年的程序员的生产率可能是10000美元/年的程序员10倍。
- 实际上，绝大多数大型编程系统的经验显示出，一拥而上的开发方法是高成本的，速度缓慢的，低效的，开发出的是无法在概念上集成的产品。

组织架构

- 我认为产品负责任作为管理者是更合适的安排。

控制规模

- 更深刻的教训体现在以上的经验中，项目规模本身很大，缺乏管理和沟通，以至于每个团队成员认为自己是争取小红花的学生，而不是勾践系统软件产品的人员。为了满足目标，每个人都在局部优化自己的程序，很少会有人听再来，考虑一下对客户的整体影响。

文档

- 技术，周边组织机构，行业传统等若干因素凑在一起，定义了项目必须准备的一些文书供祖宗。对于一个刚从技术人员中任命的项目经历来说，这减脂是一件彻头彻尾，令人生厌的事情。
- 慢慢地，他逐渐认识到这些文档的某些部分包含和表达了一些管理 方面的工作。每分文档的准备工作是集中考虑，并使各种讨论意见明细化的主要时刻。如果不这样，项目往往会处于无休止的混乱状态中，文档的跟踪维护是项目监督和预警的机制。文档本身可以作为检查列表，状态控制，也可以作为回报的数据基础。

未雨绸缪

- 对于大多数项目，第一个开发的系统并不合用。它可能太慢，太大，而且难以使用，或者三者兼而有之。要解决所有的问题，除了重新开始以外，没有其他的办法。即开始一个更灵巧或更好的系统，系统的丢弃和重新设计可以一步完成，也可以一块块地实现，所有大型系统的经验都显示，这是必须完成的步骤。而且，新的系统概念或新技术会不断出现，必须构建一个用来抛弃的系统，因为即使是最优秀的项目经历，也不能无所不知地在最开始解决这些问题。
- 我从不建议顾客所有的目标和需求的变更必须，能够或者应该整合到设计中，项目开始时建立的基准，肯定会随着开发的进行越来越高，甚至开发不出任何产品。
- 程序员不愿意为设计书写文档的原因，不仅仅是因为惰性或者时间的压力。相反，设计人员通常不愿意提交尝试行的设计决策，再为他们进行辩解。“ 通过设计文档化，设计人员讲自己暴露在每个人的批评之吓，他必须能够为他书写的一切进行辩护。如果团队加固因此收到任何形式的威胁，则没有任何东西会被文档化，除非架构是完全受到保护的。 ”
- 对于一个广泛使用的程序，其维护总成本通常是开发成本的40%或更多。令人吃惊的是，该成本受用户数目

的影响很大。用户越多，所发现的错误就越多。
- 一个有趣的循环，上一个版本中被发现和修复的bug，在新的版本中仍会出现，新版本中的新功能会产生新的bug。解决了这些问题以后，程序会正常运行几个月，接着，错误率会重新攀升，这是因为用户的使用达到了新的熟练水平，开始运用新的功能，这种高强度的考验查处了新功能中很多不易察觉的问题。
- 用在修复原有设计上瑕疵的工作量越来越少，而早期维护活动本身所引起的漏洞的修复工作越来越多，随着时间的推移，系统变得越来越无序，修复工作迟早会失去根基。每一步前进都伴随着一步后退，尽管系统在理论上一直可用，但实际上，整个系统已经面目全非。
- 系统软件开发是减少混乱度的过程，所以它本身是处于亚稳态的，软件维护是提高混乱度的过程。即使是熟练的软件维护工作，也只是放缓了系统退化到非稳态的过程。此时就需要大规模重构。

工具

- 就工具而言，即使是现在，很多软件项目仍然像经营一家五金店。每个骨干人员都仔细地保管自己工作生涯的一套工具集，这些工具成为个人技能的直观证明，正式如此，每个编程人员也保留着编辑器，排序，内存信息转储和磁盘空间使用程序的巩固。
- 所以在前面关于软件开发队伍的讨论中，我建议为每个团队配备一个工具管理人员。这个角色管理所有通用工具，能知道他的客户和老板如何使用工具。同时，他还能编制老板需要的专业工具。

文档

- 面对那些文档“ 简约 ”的程序，我们中的大多数人都不免曾经暗骂那些远在他方的匿名作者。因为，一些视图向新人慢慢地灌输文档的重要性，旨在延长软件的生命期，克服惰性和进度的压力。但是，很多次尝试都失败了，我想可能是由于我们使用了错误的方法。
- 每个用户都需要一段对程序进行描述的文字。可是太多数文档只提供了很少的总结性内容，无法达到用户要求，就像是描绘了树木，形容了树皮和树叶，但确没有一幅森林的图案，为了得到一份有用的文字描述，就必须放慢脚步，稳妥前行
- 流程图是被吹捧的最过分的一种程序文档，事实上，很多程序甚至不需要流程图，很少有程序需要一页纸以上的流程图。
- 现实中，我从来没有看过一个有经验的编程人员，在开始编写程序之前，会例行公事地绘制详尽的流程图。在一些要求流程图的组织中，流程图总是事后才补上的。

如何培养杰出的设计人员

- 尽可能早地，有系统地识别顶级的设计人员，最好的通常不是最有经验的人员
- 为设计人员指派一位职业导师，负责他们技术方面的成长，仔细地为他们规划职业生涯。
- 为每个方面指定和维护一份职业计划，包括与设计大师的，经过仔细挑选的学习过程，正式的高级教育和短期的课程，所有这些穿插在设计和技术领导能力的培养安排中。
- 为成长中的设计人员提供交流和激励的机会。

面向对象

- 因为OO和各种复杂语言的联系已经很紧密，人们并没有被告诉OO是一种设计方式，并xang他们讲授设计方法和原理，大家只是被告知OO是一种设计方法，并向他们讲授设计方法和原理，大家只是被告知OO是这一种特殊工具，而我们可以用任何工具写出优质或低劣的代码。除非我们给人们讲解无任何设计，否则语言所起的所用非常小，结果人们使用这种语言做出不好的设计，没有从中获得多少价值。而一旦获得的价值太少，它就不会流行。
- 面相对象技术包含了很多方法学上的进步，OO的前期投入很多 ，主要是重新培训程序员用很新的方法进行思考。面相对象在整个开发周期中得到了应用，但是真正的获益只有在后续开发，拓展和维护活动中才能体现出来。Coggin说：面相对象技术不会加快首次或第二次的开发，产品族群中第五个项目的开发将会异乎寻常的循序。
- 为了预期中的，但有写不确定的收益，冒着风险投入资金是投资人每天在做的事情。不过，在很多软件工资，这需要真正的管理勇气，一种比技术竞争力或者优秀管理能力更少有的精神。我认为极度的前期投入和收益的推后是使OO技术应用迟缓的最大原因。

重用的情况

- 大多数的丰富经验的程序员都拥有自己的私人开发库，使用大约30%的重用代码来开发软件。公司级别的重用能提供70%的重用代码量，它需要特殊的开发库和管理支持。公司级别的重用代码意味着需要对项目中的变更进行统计和度量，从而提高重用的可信程度。

#InterSystems IRIS