人月神话读书笔记

推荐理由

《人月神话》这本书几年前就听别人说是本很经典的软件开发方面的书，这本书的成功之处在于他思想的前卫性，以至于不只是软件行业的人在读。从这本书的内容来看，对于一个项目经理来说肯定会有更大的收获，这本书主要是针对软件开发管理方面的内容，这主要原因可能是因为作者以前就是项目的管理者，他是站在管理者的角度写的。即便这样，对于一个从来没有参与过真实项目开发，更没有领导过团队的我还是有一定的吸引力。这本书里面为了论证某一观点，会举出许多实际的项目作为证据，这些例子也许对于作者那个年代的人来说很好理解，但是放在30年后来看这些例子又有些陈旧和难懂了。

本书要点

- 焦油坑

  > *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，但是当他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。*

- 人月神话

  > *在众多软件项目中，缺乏合理的时间进度是造成项目滞后的最主要原因，它比其他所有因素加起来的影响还大。*

- 外科手术队伍

  > *如何在有意义的时间进度内创建大型的系统？*

- 贵族专制、民主政治和系统设计

  > *对于计算机系统而言，尽管它们通常没有花费几个世纪的时间来构建，但绝大多数系统体现出的概念差异和不一致性远远超过欧洲的大教堂。*

- 画蛇添足

> *如果将制订功能规格说明的责任从开发快速、成本低廉的产品的责任中分离出来，那么有什么样的准则和机制来约束结构师的创造性热情呢？基本回答是结构师和建筑人员之间彻底、仔细和谐的交流。*

- 贯彻执行

> *假设一个项目经理已经拥有形式规范的结构师和许多编程实现人员，那么他如何确保每个人听从、理解并实现结构师的决策？*

- 为什么巴比伦塔会失败

> *巴比伦塔是人类继诺亚方舟之后的第二大工程壮举，但巴比伦塔同时也是一个彻底失败的工程。*

- 胸有成竹

> *即使在不考虑相互交流沟通，开发人员仅仅回顾自己以前工作的情况下，这些数字仍然显示出工作量是规模的幂函数。*

- 削足适履

> *由于规模是软件系统产品用户成本中如此大的一个组成部分，开发人员必须设置规模的目标，控制规模，考虑减小规模的方法。*

- 提纲挈领

> *在一片文件的汪洋中，少数文档形成了关键的枢纽，每件项目管理的工作都*

围绕着它们运转。它们是经理们的主要个人工具。

- 未雨绸缪

  “

  为舍弃而计划，无论如何，你一定要这样做。

- 干将莫邪

  “

  项目经理应该制订一套策略，并为通用工具的开发分配资源。与此同时，他还必须意识到专业工具的需求，对这类工具不能吝啬人力和物力。

- 整体部分

  “

  如何开发一个可以运行的系统？如何测试系统？如何将经过测试的一系列构件集成到已测试过、可以依赖的系统？

- 祸起萧墙

  “

  通常载货来自白蚁的肆虐，而不是龙卷风的侵袭。

- 另外一面

  “

  公共应用程序的用户在时间和空间上都远离它们的作者，因此对这类程序，文档的重要性不言而喻。

精编书摘

- 焦油坑

    - 编程系统产品开发的工作量是供个人使用的、独立开发的构建程序的九倍。软件构件产品化引起了3倍工作量，将软件构件整合成完整系统所需要的设计、集成和测试又强加了3倍的工作量，这些高成本的构件在根本上是相互独立的。
    - 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”，提供了五种乐趣。
    - 同样，这个行业具有一些内在固有的苦恼。
- 人月神话
    - 缺乏合理的时间进度是造成项目滞后的最主要原因，它比其他所有因素加起来影响还大。
    - 良好的烹饪需要时间，某些任务无法在不损害结果的情况下加快速度。
    - 所有的编程人员都是乐观主义者：“一切都将运作良好”。
    - 由于编程人员通过纯粹的思维活动来开发，所以我们期待在实现过程中不会碰到困难。
    - 但是，我们的构思是有缺陷的，因此总会有bug。
    - 我们围绕成本核算的估计技术，混淆了工作量和项目进展。人月是危险和带有欺骗性的神话，因为它暗示人员数量和时间是可以相互替换的。
- 外科手术队伍
    - 同样有两年经验而且在受到同样的培训的情况下，优秀的专业程序员的工作效率是较差程序员的十倍。
    - 小型、精干队伍是最好的——尽可能的少。
    - 两个人的团队，其中一个项目经理，常常是最佳的人员使用方法。
    - 对于真正意义上的大型系统，小型精干的队伍太慢了。
    - 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、速度缓慢、不充分的，开发出的产品无法进行概念上的集成。
    - 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，还彻底地减少了沟通的工作量。
- 贵族专制、民主政治和系统设计
    - “概念完整性是系统设计中最重要的考虑因素”。
    - “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”，而不仅仅是丰富的功能。
    - 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。
    - “对于非常大型的项目，将设计方法、体系结构方面的工作与具体实现相分离是获得概念完整性的强有力方法。”
    - “如果要得到系统概念上的完整性，那么必须控制这些概念。这实际上是一种无需任何歉意的贵族专制统治。”
    - 纪律、规则对行业是有益的。外部的体系结构规定实际上是增强，而不是限制实现小组的创造性。
- 画蛇添足
    - 尽早交流和持续沟通能使结构师有较好的成本意识，以及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。
    - 结构师如何成功地影响实现:

        - 牢记是开发人员承担创造性的实现责任；结构师只能提出建议。
        - 时刻准备着为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。
        - 对上述的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。
    - 第二个系统是人们所涉及的最危险的系统，通常的倾向是过分地进行设计。
    - 为功能分配一个字节和微妙的优先权值是一个很有价值的规范化方法。
- 贯彻执行
    - 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。
    - 必须明确定义体系结构中与先前定义不同的地方，重新定义的详细程度应该与原先的说明一致。
    - 出于精确性的考虑，我们需要形式化的设计定义，同样，我们需要记叙性定义来加深理解。
    - 必须采用形式化定义和记叙性定义中的一种作为标准，另一种作为辅助措施；它们都可以作为表达的标准。
    - 设计实现，包括模拟仿真，可以充当一种形式化定义的方法;这种方法有一些严重的缺点。
- 为什么巴比伦塔会失败
    - 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。
    - 团队应该以尽可能多的方式进行相互之间的交流：非正式、常规项目会议，会上进行简要的技术概述、共享的正式项目工作手册。
- 胸有成竹
    - 仅仅通过对编码部分的估计，然后乘以任务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。
    - 构建独立小型程序的数据不适用于编程系统项目。
    - 程序开发呈程序规模的指数增长。
    - 一些发表的研究报告显示指数约为 1.5。
    - Portman的ICL数据显示相对于其他活动开销,全职程序员仅将50%的时间用于编程和调试。
    - IBM的ron数据显示，生产率是系统各个部分交互的函数，在1.5K千代码行/人年至10K千代码行/人年的范围内变化。
- 削足适履
    - 除了运行时间以外,所占据的内存空间也是主要开销。特别是对于操作系统，它的很多程序是永久驻留在内存中。
    - 即便如此，花费在驻留程序所占据内存上的金钱仍是物有所值的，比其他任何在配置上投资的效果要好。规模本身不是坏事,但不必要的规模是不可取的。
    - 软件开发人员必须设立规模目标，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。
    - 规模预算不仅仅在占据内存方面是明确的,同时还应该指明程序对磁盘的访问次数。

    - 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。9.6 在大型的团队中,各个小组倾向于不断地局部优化,以满足自己的目标,而较少考虑队用户的整体影响。这种方向性的问题是大型项目的主要危险。
- 提纲挈领
    - “前提:在一片文件的汪洋中，少数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。
    - 因此,即使是小型项目，项目经理也应该在项目早期规范化上述的一系列文档。
- 未雨绸缪
    - 化学工程师已经认识到无法一步将实验室工作台上的反应过程移到工厂中，需要一个实验性工厂(pilot planet)来为提高产量和在缺乏保护的环境下运作提供宝贵经验。
    - 对于编程产品而言，这样的中间步骤是同样必要的,但是软件工程师在着手发布产品之前,却并不会常规地进行试验性系统的现场测试。
    - 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。
    - 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。这是个必须完成的步骤。
    - 将开发的第一个系统——丢弃原型——发布给用户，可以获得时间，但是它的代价高昂——对于用户，使用极度痛苦；对于重新开发的人员，分散了精力；对于产品，影响了声誉，即使最好的再设计也难以挽回名声。
    - 因此，为舍弃而计划，无论如何，你一定要这样做。
- 干将莫邪
    - 项目经理应该制订一套策略，以及为通用工具的开发分配资源，与此同时，他还必须意识到专业工具的需求。
    - 开发操作系统的队伍需要自己的目标机器，进行调试开发工作。相对于最快的速度而言,它更需要最大限度的内存，还需要安排一名系统程序员，以保证机器上的标准软件是即时更新和实时可用的。
    - 同时还需要配备调试机器或者软件，以便在调试过程中，所有类型的程序参数可以被自动计数和测量。
    - 目标机器的使用需求量是一种特殊曲线：刚开始使用率非常低,突然出现爆发性的增长，接着趋于平缓。
    - 同天文工作者一样，系统调试总是大部分在夜间完成。
    - 抛开理论不谈，一次分配给某个小组连续的目标时间块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。
    - 尽管技术不断变化，这种采用时间块来安排匮乏计算机资源的方式仍得以延续20年，是因为它的生产率最高。
- 整体部分

    - 在编写任何代码之前，规格说明必须提交给测试小组，以详细地检查说明的完整性和明确性。开发人员自己不会完成这项工作。(Vyssotsky)
    - “十年内[1965~1975]，Wirth的自顶向下进行设计[逐步细化]将会是最重要的新型形式化软件开发方法。”
    - Wirth主张在每个步骤中，尽可能使用级别较高的表达方法。
    - 好的自顶向下设计从四个方面避免了bug。
    - 有时必须回退，推翻顶层设计，重新开始。
    - 结构化编程中，程序的控制结构仅由支配代码块(相对于任意的跳转)的给定集合所组成。这种方法出色地避免了bug，是一种正确的思考方式。
- 祸起萧墙
    - 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。
    - 如果里程碑定义得非常明确，以致于无法自欺欺人时，程序员很少会就里程碑的进展弄虚作假。
    - 对于大型开发项目中的估计行为，政府的承包商所做的研究显示：每两周进行仔细修订的活动时间估计，随着开始时间的临近不会有太大的变化；期间内对时间长短的过高估计，会随着活动的进行持续下降；过低估计直到计划的结束日期之前大约三周左右，才有所变化。
- 另外一面
    - 对于软件编程产品来说，程序向用户所呈现的面貌与提供给机器识别的内容同样重要。
    - 即使对于完全开发给自己使用的程序，描述性文字也是必须的，因为它们会被用户-作者所遗忘。
    - 培训和管理人员基本上没有能向编程人员成功地灌输对待文档的积极态度——文档能在整个生命周期对克服懒惰和进度的压力起促进激励作用。
    - 这样的失败并不都是因为缺乏热情或者说服力，而是没能正确地展示如何有效和经济地编制文档。

作者简介

Frederick P. Brooks, Jr.是北卡罗来纳大学Kenan-Flagler商学院的计算机科学教授，北卡来罗来纳大学位于美国北卡来罗来纳州的查布尔希尔。Brooks被认为是“IBM360系统之父”，他担任了360系统的项目经理，以及360操作系统项目设计阶段的经理。凭借在上述项目的杰出贡献，他、Bob Evans和Erich Bloch在1985年荣获了美国国家技术奖（National Medal of Techology）。早期，Brooks曾担任IBM Stretch和Harvest计算机的体系结构师。

在查布尔希尔，Brooks博士创立了计算机科学系，并在1964至1984年期间担任主席。他曾任职于美国国家科技局和国防科学技术委员会。Brooks目前的教学和研究方向是计算机体系结构、分子模型绘图和虚拟环境。