南京大学软件学院

《人月神话》读书笔记

软件工程管理作业

王驰焱 MF1832152

2018/9/17

《人月神话》是作者在软件开发领域心得的文集。本人的读书笔记按论点的不同方向对内容作了重新整理，提炼其中的重要观点。

目录

笔记：关于软件开发

第一章：焦油坑

作者用焦油坑来比喻伴随软件开发过程的痛苦，引出主题。

论点：个人程序与编程系统产品间存在明显差别

如下图所示，一方面要产品化，另一方面要构件化，这使得成本倍增（书所所言为 9 倍）。这意味着不能以单以个人编程的生产率来评估大型系统的编程活动。

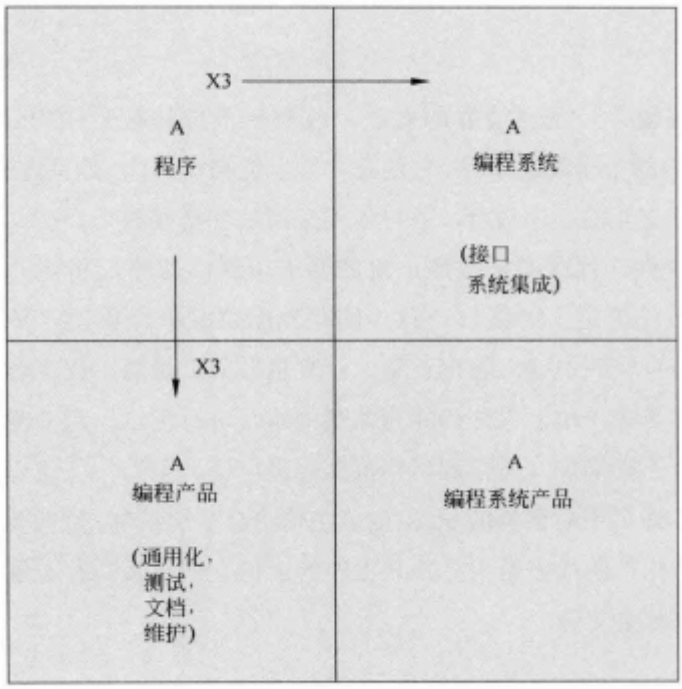

图 1-1　编程系统产品的演进

论点：编程行业的乐趣和苦恼

“幸福的家庭都是相似的，不幸的家庭各有各的不幸”，我读的时候更关注也更同意作者所讨论的烦恼，大致如下：

a) 需要追求完美
b) 工作中对他人的依赖
c) 创造性活动之外琐碎的工作（比如写文档，捉虫子，改别人写或者自己几年前写的代码）
d) 千辛万苦做出来，却发现没有价值

笔记：进度控制

第二章：人月神话

论点：项目延迟的主要原因——缺乏合理的进度安排

a) 对估算技术缺乏有效的研究
b) 错误地将进度与工作量混淆
c) 对估算缺乏信心
d) 对进度缺少追踪和监督
e) 意识到可能发生 delay 时，下意识地去堆人力

展开：错误的思考方式或假设

有时候安排进度时会**过于乐观**：设想一切都将运作良好，每一项任务仅花费他所应该花费的时间。然而，我们所要考虑的不仅仅是“实现”，构思和交流同样重要；我们下意识地认为实现是易于驾驭的，而构思是有缺陷的。实际上，在复杂任务中，“一切正常”的概率是非常小的。

人力成本具有欺骗性（人月神话）。人数与时间的互换仅适用于任务可分解，不需要交流的情况，而软件开发是一项系统性工作。

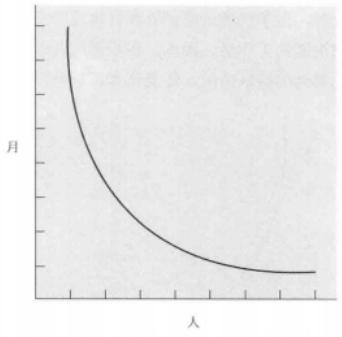

图 2-1 人员和时间之间的关系——完全可以分解的任务

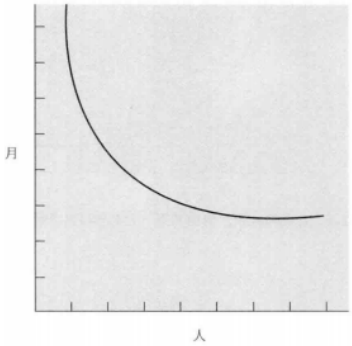

图 2-3 人员和时间之间的关系——需要沟通的可分解任务

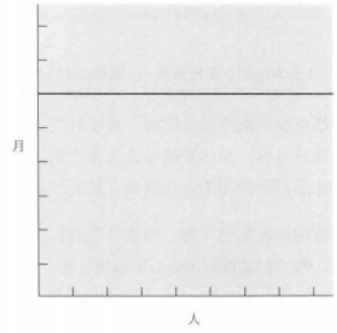

图 2-2 人员和时间之间的关系——无法分解的任务

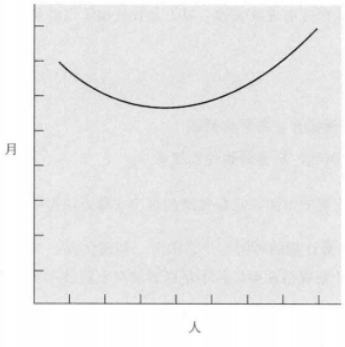

图 2-4 人员和时间之间的关系——关系错综复杂的任务

如图所示，软件开发往往是第三或第四种情形，在项目进度出现问题时试图以投入更多人力的方式来挽救往往不能带来所希望的结果（培训需要成本、任务需要重新划分、额外系统测试）。更好的做法是重新安排进度、削减任务。

对测试进度的安排往往不合理。在早期进度策划时，就应该安排足够多的系统测试时间。作者的对进度安排的建议是 1/3 计划，1/6 编码，1/4 构件测试和早期系统测试以及 1/4 系统测试。（考虑到作者以大型系统为出发点以及实际运作中人员分工的情况，我觉得这个数字是可信的。）

对进度**估算过于空泛**。有时候安排进度的人员会为满足客户给出的不合理的安排或仅凭直觉进行估算。作者给出的解决方案是生产率图表、缺陷率图表、估算规则等，并且在客户（或老板）的压力面前坚持自己的估计。我觉得实际上很多所谓敏捷开发团队对进度的估算就是相当空泛的，一种很实际的方法是让开发人员参与进度制定的确认过程。

第八章：胸有成竹

讨论：关于项目估算

a) 编码大约只占问题的 1/6 左右
b) 构建独立小型程序的数据不适用于编程系统产品
c) 工作量 = 常数 * 指令数量^1.5（也就是说随着代码规模的增长，工作量是呈指数增长的）

作者指出，项目估算可能会对每个人年的技术工作时间作出不现实的假设（也就是说未考虑会议、假期、机器故障等耗费的时间）。

在这一章中，作者就软件开发生产率给出了若干较为可靠的统计数字。

第十四章：祸起萧墙

论点：项目进度落后是一天天积累下来的

a) 进度控制中的里程碑必须是具体的、特定的、可度量的事件，能够清晰定义。
b) 要定期修订进度。
c) 每一天的滞后是量变导致质变的过程。
d) 滞后信息不应该被隐瞒，要减少角色冲突，公开状态真相。
e) 对计划和控制职能投入适度的人力资源。

笔记：人力与沟通

第三章：外科手术队伍

前提：优秀程序员与普通程序员之间存在巨大的效率差异

矛盾：精干的高手队伍无法满足大型系统的时间要求

解决：Harlan Mills 的人力方案

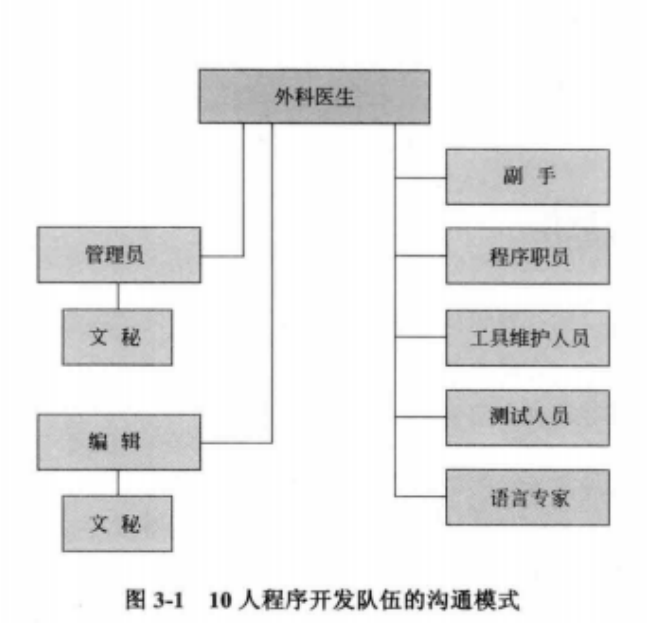

图 3-1　10 人程序开发队伍的沟通模式

值得注意的是，这里的“程序职员”不是负责写代码的那个人，“外科医生”才是 coder。关键在于，以上述思路构建大型系统的开发团队，200 人的团队中实际需要协调的开发者仅仅是 20 人。不过我觉得对于中小型项目而言，这样的架构可能过于冗余了。

第六章：贯彻执行

问题：如何保障每个人听到、理解并执行架构师的决策？

1. 手册或书面规格说明
   - 版本迭代
   - 清晰、完整、准确
   - 一致性、完整性、整体性

2.在规格说明中使用形式化定义（外部功能）
- 较大程度上脱离自然语言的不确定性
- 缺点是不易理解，可用记叙性文字辅助
- 值得注意的是，实现（程序化仿真，我觉得可以理解为原型）本身也是一种形式化定义，但可能会导致过度的约束

3.模块间通过接口整合
4.会议（例会）
- 提前分发建议书，重点是创新
- 增进对项目的了解程度
- 对问题更敏锐
- 在界线内外同时寻求解决方案
- 建议书能集中注意力
- 明确架构师的权力，避免妥协和拖延

5.年度大会
- 解决堆积起来的未能解决的问题，每个人都能参与和表达

6.多重实现（使得定义更加整洁规范）
7.电话会议
8.产品测试
- 及时、仔细

第七章：为什么巴比伦塔会失败

大型项目中的交流问题的解决方案

a) 非正式途径（电话，放现在就是各种即时通讯软件了）
b) 会议
c) 工作手册
d) 改良组织架构

展开：工作手册

定义:对项目必须产出的一系列文档进行组织的一种结构,包括目的、外部规格说明书、接口说明、技术标准、内部说明和管理备忘录。

作用：追溯历史（如设计思路等）；控制信息发布（确保信息能到达需要它的人手中）。

作者提到了微缩胶片这种形式来管理工作手册，当然现在都可以数字化了。同时作者也提到每个编程人员只需了解自己负责的部分（先决条件是精确、完整地定义接口）。

展开：组织架构问题

目的：减少所需的交流和合作。

方法：人力划分和限定职责范围。

形式：树状管理结构。

同时作者也探讨了技术主管与产品负责人可能的关系。

笔记：关于系统设计

第四章：第四章：贵族专制、民主政治和系统设计

问题：如何保障系统设计的概念完整性？

宁可省略一些不规则的特性和改进，也不提倡独立和无法整合的系统，哪怕他们其实包含着很好的设计。

目标是易用性，而评价准则是功能与概念复杂度的比值。同时易用性需要设计上的一致性和概念上的完整性。

不过作者也提出了这样的矛盾：概念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来实现，但进度压力却要求很多人来开发。解决方法有：设计方法和具体实现分工；采用前面提到的“外科手术队伍”的形式组建团队。

需要注意的是，架构和实现同样具有创造性，遵从架构师的主导地位只是为了更高的效率和更好的系统概念上的完整性。

此外，在等待体系结构设计的过程中，设计实现人员仍然有很多事情可以做。

第五章：画蛇添足

问题：如何约束架构师？

答：（与开发人员间）彻底、谨慎、和谐的交流。

如果估算过高，可以削减设计或采用成本更低的实现方法，但对于软件开发而言，后者是对开发人员的 做事方式做出挑战。

实际上，架构师应当建议而不是支配开发人员，能灵活地接受变通的实现。

另外要**避免过度设计**。（就好像现在经常谈论的“伪需求问题”）。

第十一章：未雨绸缪

论点：变化是必然的，关键是如何应对

1.不变的变化（变化的来源）：

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（感觉这是对“改需求”这种说法最令我信服的说辞了）。

不必盲目接受所有的客户需求，但事先为可能的变化做准备总是有必要的

2.为变更设计系统

- 细致的模块化
- 可扩展的函数
- 精确完整的模块间接口设计
- 完备的文档
- 调用队列和表驱动的技术
- 使用高级语言和自文档技术（减少变更引起的错误）
- 变更的阶段化（版本控制）

3.为变更计划组织架构

4.软件维护问题：回归测试成本高昂

5.在维护工程中，**所有的修改都倾向于破坏系统的架构，增加了系统的混乱程度**（熵）

第十三章：整体设计

建议：从整体上考虑系统设计

1. 从设计上就避免或减少 bug 的产生
   - 细致的概念定义
   - 仔细的规格说明
   - 规范化的功能描述说明
   - 规格说明要交给外部测试小组检查完整性和明确性
   - 自上而下设计
   - 结构化编程
2. 构件单元测试
   - 调试技术：交互式调试
   - 测试用例设计
3. 系统集成调试
   - 确保构件已经过测试
   - 搭建充分的测试平台
   - 控制变更
   - 一次添加一个构件
   - 阶段化、定期变更

笔记：关于文档

第十章：提纲挈领

论点：文档的跟踪维护是项目监督和预警的机制。

作者举了几个行业的例子，分析文档的重要组成部分。

问题：为什么要有文档？

- 书面记录决策是必要的（分歧明朗，矛盾突出）
- 作为同其他人的沟通渠道
- 作为数据基础和检查列表

第十五章：另外一面

讨论：好的文档是什么样的

不同用户需要不同级别的文档。

- 使用程序
- 验证程序
- 修改程序

在作者看来，流程图被过分鼓吹了，高级语言的使用使得流程图显得落后了。

同时作者还提到了**自文档技术**，其使得文档与代码保持一致。

笔记：其他

第九章：削足适履

讨论：程序本身的成本

作者提到，需要从整体上控制预算、把握规模、确切定义功能（避免冗余设计）。空间预算需要创造性，同时需要考虑空间-时间的折中。

（我的观点：不过随着硬件设施的显著提升，这个问题可能不再显得那么突出了，但是这不代表这个问题不再重要，特别是在设计运行在受限的硬件平台上的程序的时候。）

第十二章：干将莫邪

论点：开发和维护公共的通用编程工具的效率更高

作者建议配备工具管理人员（负责管理工具、指导使用、编制工具等），并认为对专业工具的开发不能吝啬人力和物力。

几类工具：实用程序、调试辅助程序、测试用例生成工具、文档处理系统。

（值得注意的是，虽然作者更多地考虑了大型系统和硬件平台的情形，但其中提供的思路都是很值得借鉴的。）

此外作者还提到了高级语言（生产率更高，调试速度快）和交互式编程（与批处理系统相比）的重要性。

第十六章：没有银弹

讨论：软件开发中的难点

作者认为，软件开发中困难的部分是规格说明、设计和测试这些概念上的结构，而不是对概念进行表达和对实现逼真程度进行验证。

作者提出了现代软件系统中无可避免的内在特性：

- 复杂性
- 一致性
- 可变性
- 不可见性

作者提到软件开发的次要问题已经通过高级语言、分时系统、统一编程环境等取得了一定的突破，但概念上的根本问题尚没有像硬件领域那样巨大的进展。不过作者也提出了一些方法：

- 需求精炼和快速原型
- 增量开发
- 卓越的设计人员