信息科学与技术丛书

Deduction of Perfect Software Development

李智勇 丁静 编著

完美软件开发：方法与逻辑

一本培养软件开发帅才的书；一本帮您在软件开发过程中鸟瞰全局，运筹帷幄，带领团队攻城略地的书；一本通过逻辑和事实直指软件开发本质的书。

- 完美软件开发之解构
- 完美项目管理之解构
- 完美流程之解构
- 完美开发模型之解构
- 完美估算方法之解构
- 完美需求开发之解构
- 完美设计和编码之解构
- 设计和编码的度量与改善

机械工业出版社
CHINA MACHINE PRESS

完美软件开发：方法与逻辑

李智勇　丁 静　编著

机械工业出版社

本书深入剖析了软件开发中主要环节（管理、流程、开发模型、估算、需求开发和设计编码）的运作规律。

在剖析过程中，主要使用演绎法进行推导，同时使用实践中积累的经验对推导出来的结论进行验证。在这一过程中，借鉴了 PMBOK、CMMI、敏捷、功能点方法、面向对象分析与设计等思想或方法的精华内容。

从读者的角度看，本书更适合有一定开发经验，希望在软件开发这个行业有所建树的读者；也适合不仅满足于完成手里的工作，还喜欢透过现象思考本质的人；毕业生可以用这本书来开阔视野，规划自己的发展方向，但有些地方可能会感到不容易理解。

图书在版编目（CIP）数据

完美软件开发：方法与逻辑 / 李智勇，丁静编著. —北京：机械工业出版社，2013.5

（信息科学与技术丛书）

ISBN 978-7-111-42626-4

Ⅰ. ①完… Ⅱ. ①李… ②丁… Ⅲ. ①软件开发 Ⅳ. ①TP311.52

中国版本图书馆 CIP 数据核字（2013）第 109306 号

机械工业出版社（北京市百万庄大街 22 号 邮政编码 100037）

策划编辑：丁 诚

责任编辑：丁 诚

责任印制：乔 宇

三河市宏达印刷有限公司印刷

2013 年 6 月 · 第 1 版第 1 次印刷

184mm × 260mm · 11 印张 · 273 千字

0001—3000 册

标准书号：ISBN 978-7-111-42626-4

定价：35.00 元

凡购本书，如有缺页、倒页、脱页，由本社发行部调换

电话服务

社 服 务 中 心：（010）88361066

销 售 一 部：（010）68326294

销 售 二 部：（010）88379649

读者购书热线：（010）88379203

网络服务

教 材 网：http://www.cmpedu.com

机工官网：http://www.cmpbook.com

机工官博：http://weibo.com/cmp1952

封面无防伪标均为盗版

前　言

在武侠小说中，常会把绝世武功分为两个部分：招式和心法。招式得其形，而心法传其神。从这个角度看，这本书是既讲招式也讲心法的书。

招式繁杂，暂且不提；心法却可以概括。

如果非要用 3 句话来概括本书中所提心法的全部，那么它们是（顺序不可颠倒，有因果关系）：

在尺度中潜在的已经包含本质；尺度的发展过程只在于将它所包含的潜在的东西实现出来。——黑格尔，《小逻辑》

人心唯危，道心唯微，唯精唯一，允执厥中。——《尚书》

横尽虚空，山河大地，一无可恃，而可恃唯我。竖尽久劫，前古后今，一无可据，而可据唯目前。——杨昌济

但这三句话都过于精妙，很难把它们和管理、流程、估算、开发模型、需求开发、设计编码联系起来。本书做的正是这样一种尝试：在把软件作为一个整体进行考察的同时，把精妙抽象的东西和具体的东西结合起来。

把软件作为一个整体考察是因为管理、流程、设计编码等都是影响最终成效的砝码，单纯某一个维度上（比如流程）效能最佳不等于整体效能最佳。

而把精妙抽象和具体相结合则是因为虽然各种理论众说纷纭，但总有些规则凌驾于现象之上，不把握这些规则必然会陷入混乱，进而明于微而昧于巨。反之，精妙的东西又只有通过具体的手段才能实现，无法单独存在。恰如下棋时，虽然每一步都机关算尽，但总是脱不开既定规则，只有有限的结局（或输、或赢、或和），而既定规则的力量又只能实现于每一步具体的操作之中。

作为结果，我们可以讲：

这本书只相信逻辑和事实。虽参照诸多素材（敏捷、CMMI、OO、设计模式等），但主要依靠独立思考才最终塑成体态。纵然错漏难免，但书中所言皆是自主反思所得，虽常有反主流之观点，用心想来却不一定是无稽之谈。

本书直指本质。虽然有的地方略显晦涩难懂，但确实是因为无法简化，并非毫无价值，更非故弄玄虚。同时，为避免偏颇，本书主要使用演绎法，基于以下 4 个预设前提推导各种结论，并用事实进行佐证。

- 软件是一种固化的思维。
- 意识指导行动。

- 项目所能耗费的资源是有限的。
- 重复做同样的工作会降低效率。

本书是培养帅才的书。如果您只想成为一名悍将（比如，C++高手、Android 高手），那本书不太适合您；但如果您想鸟瞰全局，运筹帷幄，带领团队攻城略地，那本书则是很有参考价值的。

本书在一定程度上可以终止某些争议。软件开发这个行业之中如果过度相信经验主义，则事实会证明效果并不好。几十年发展下来，各种理论依然纷争不断，比如，是做架构设计，还是测试驱动；是敏捷，还是 CMMI 等。本书在一定程度上可以包容这些矛盾，恰如黑格尔的辩证法可以包容康德的二律背反一样。

本书是一个开始而非结束。限于作者的眼界、能力、时间等，本书无法终结所提及的所有问题。希望能有志同道合者与作者一同继续研究这个题目，作者也希望能收到各种建议来不断提高自我。

作　者

目　录

第1章　完美软件开发之解构

某哲学家曾经说过：现实世界不过是理想世界的一个苍白摹本。就像完美的圆形只在概念中存在，现实中的圆形只能是对完美的圆形的无限回归。这也即是所谓的完美大多处于虚无之中。

然而完美境界之所以具有永恒的价值，却并非在于其是否可实现，而是在于其可以给不懈努力者以终极的指引。真的完美境界更多地会体现为一种原则，一种规律，一种必然性，它并不以个人的喜好而变动半分，当你背离它时，它会以惨痛的教训让你重新认识到它的存在。而所谓成功项目或者失败项目所昭示的则是对这种规则性的顺应程度。

当我们处于蒙昧之中时，往往并不知道当为不当为之准绳，其结局必然是在沉重中惶恐，在惶恐中希冀，在希冀中重新陷入迷惘。当此情境，唯有理想真知才能让人破妄返真，重拾方向与希望。所以觉悟之后，虽知一去难返，却终究要在虚无中寻找永恒，以完美境界为现实之归宿。对完美之追寻，实是以短暂之身，叩问永恒之道，窥天心而解惑。

软件亦概莫能外——谨以此拉开完美软件开发的序幕。

1.1　完美软件开发的定义

不识庐山真面目，
只缘身在此山中。

——苏轼，《题西林壁》

任何软件开发，其输入的都是需求、工具（编译工具等）和人，这三者在特定的时空背景下受主观的影响而改变的可能性小。

接下来，基于上述三者，通过选定的管理方法、流程、开发模型、需求开发方法、估算方法、设计编码方法等对软件进行构建，期望最终达成多重质量目标。

最终输出的是软件产品。对软件产品的度量至少有两个维度：一是用户层面的，如功能正确，性能优异；二是结构层面上的，如容易维护等。

从输入到输出这一过程受三个维度的因素影响，一是商业因素，二是技术因素，三是“政治”因素。商业因素决定收益与成败，技术因素决定成本和质量，而“政治”因素添加变数。在这里，“政治”是指人与人之间非理性的复杂关系，即“项目政治”，而非“国家政治”。

如果只考虑技术因素即只考虑软件开发自身的内在合理性，那么最终推导出的就是完美方法，掺杂了商业因素和“政治”因素后的真实的方法，将是对完美方法的折中和回归。

完美软件开发是追求这样一种状态：在只考虑技术因素的情况下，在限定的要求、工具、指定的人员状况下（输入），在既定的质量水平下（输出），达到生产效能最高。在这一状态下，任何对管理、流程、估算方法等在尺度上的修正，如果不以降低质量为代价，则将导致生产效能的降低。

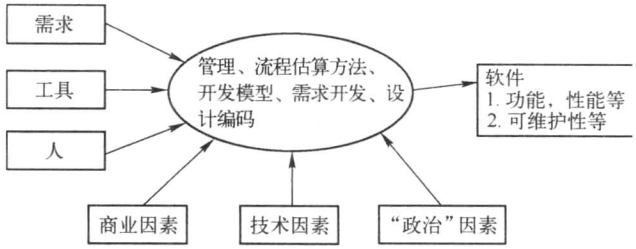

图 1-1　软件开发的基本组成

建模

设定一个范围，把范围内的东西抽象出对应的概念，再标明这些概念间的关系，这样一个过程就是建模。为免晦涩，这里的定义可能和教科书里略有不同，但建模的概念大致如此，并不高深，几乎人人可做。

1.2　完美软件开发的构成

自其异者视之，肝胆楚越也；自其同者视之，万物皆一也

——庄子，《德充符》

软件是一种固化的思维，这一点决定了许多的事情。

从特质上来看，既然软件是固化的思维，那就必然同时具备思维以及思维所承载之物的特质。

- 思维的特质是指思维的澄清通常是渐进的，思维自身是不可度量的，思维的主体一定是人，思维通常由概念和逻辑组成，思维的无边界化（灵活易变）这样的特质。这部分特质是共通部分，同时属于所有软件。
- 思维承载之物的特质是指当思维的对象是数学的时候，思维本身也就具备了数学的特质；当思维的对象是商业逻辑的时候，思维自身也就具备了商业逻辑的特质。

既然思维自身的特质是复合的，那么作为固化思维的软件，其特质必然也是复合的：既有属于所有软件的共同特质，也有特属于某类软件，甚至同其他类软件完全相反的独有特质。也就是说，在软件这一范畴里，两种矛盾的说法同时成立，并不是什么值得惊讶的事情。

既然软件的世界如此多元，那么进行完美软件开发的讨论时，就必须忽略一些细节，才可能在限定的篇幅下，取得有价值的结论。

因此，在本书后续讨论中，将更多的基于思维的特质，而非思维承载之物的特质，来探讨软件。这样得出的结论才更有普适性。

如果说软件是一种固化的思维，那么软件开发无疑是思维固化的过程。在思维固化的过程中有两个层面的问题需要同时解决。

- 思维的主体必然是人，当多个人在一起协作的时候，彼此间的关系如何处理？在这背后隐含的两个分支是管理和流程。
- 在软件开发过程中，从本质来看，事实上只有两个根本步骤：一是弄清楚要做什么（需求开发）；二是对思维进行固化（设计，编码）。对这两个步骤的时序进行各种安排，则产生各种开发模型。为支持这两个步骤能够平滑进行，需要预先进行估算。

上述分解总结起来如图 1-2 所示。

在做出上述分解后，我们可以进一步推断完美软件开发必然包含着两个根本命题：一是上述各个分解步骤自身的最优化；二是上述各个步骤彼此间搭配的最优化。

也就是说现存的大多方法论（CMMI、敏捷等），由于其过度强调某单一维度，同时漠视方法与软件本质间的关联，一定程度上讲其本质是苍白的。

图 1-3 用于说明这种关联的复杂性。

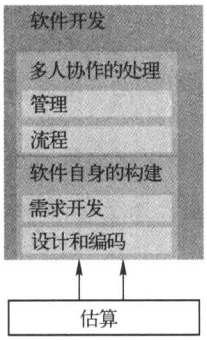

图 1-2　软件开发中各项活动的归类

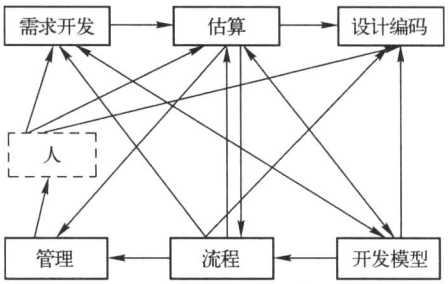

图 1-3　软件开发中各项活动的关联关系

本质与细节

这世上同时存在着两种对立的声音：本质决定成败和细节决定成败。

偏好本质的人喜欢说本质论。

偏好细节的人则喜欢说精细化管理。

但如果在较长的时间轴上考量这两种观点，就会发现它们之间并不真的对立。

本质决定大尺度时间上的走势和必然性，而细节则决定差异（包括短期成败）。

比如说：人的本质特征是能思考，有感情，会衰老，寿命有限等，但区别不同人的却不是这些，而是性格，肤色，发色等细节。

具体来看：软件本质上是只有人才能处理的东西，因此公司中程序员群体的衰落一定会导致软件的衰落，只有优秀的程序员群体，才能保证软件的持久成功，这是必然的。但优秀的程序员却不一定确保当前项目成功，任何细节上的小疏忽，都可能导致软件在市场上崩溃，死锁，进而导致灾难性后果，这就是细节决定成败。

成败自身虽然万众瞩目，对个体而言却只是一种偶然和机遇。当事人可以很努力地平衡本质上的追求（长期视点）和细节上的追求（短期视点），但变更的始终是一种成败可能性。

1.3　完美软件开发的前提

十世古今，始终不离于当念；无边刹境，自他不隔于毫端。

——李通玄

当我们试图对完美的软件开发进行阐述时，事实上总是有两类方法可以帮助我们达成目的。

- 归纳法。即基于项目经验，总结出一定的规律，再推而广之。在没有反例出现之前，

总结出的规律一直成立。

- 演绎法。即基于预设的前提，依照逻辑，推导各种结论。这个时候，只要前提没被推翻，逻辑又没有错误，那么得出的结论就具有必然性，虽然这种必然性何时体现在现实之中比较难以测度。

当前软件行业采用的大多数方法论和结论是基于归纳法的。

但一如我们前面所论及的，软件作为一种固化的思维，必然同时具备思维以及思维承载之物的双重特质。而思维承载之物的特质有时候甚至可能产生矛盾，因此基于归纳法得出的结论，容易偏颇并引发争议。

很显然，从数学类软件上得出的结论与业务流类软件上得出的结论并非是共通的；适用于操作系统内核的经验，也并不一定适合于视频播放软件。

与此同时，人的精力是有限的，这就导致一个人既不太可能经历所有的软件开发领域，也不可能穷尽同一类软件下的所有变数。比如说，A 所经历的 MIS 系统和 B 所经历的 MIS 系统很可能天差地别。

这很可能是软件行业中纷争不断的一个主要原因——每个人都以为自己看到的就是全部，而当另一种与自己不同的观点出现时，这种观点会因与己不同而被判处死刑，但事实上两者可能同时正确，只是正确的边界不同。

为避免这类争议，在这本书里，我们将主要使用演绎法。为得到各种结论，我们将主要基于以下 4 个前提，使用逻辑进行推导。

- 软件是一种固化的思维。
- 意识指导行动。
- 项目所能耗费的资源是有限的。
- 重复做同样的工作会降低效率。

这 4 个前提被假设为公理，将不额外进行说明。而为使结论不显得晦涩和突兀，我们将尽可能使用现实里的例子对逻辑链和结论进行佐证和补充。

为了从这 4 个基本前提推导出属于各个部分（管理、流程等）的相关结论，我们还需要对影响团队创造价值的因素做进一步分解，否则事情会保持在混沌状态，进而无法以正确的尺度判断某一行为所能产生的正、负两方面影响。

如果我们假设一个人的工程素养为 E，一个人的工作意愿为 W，组织所能提供的力量为 O，内耗系数为 M，那么对于一个拥有 n 个人的团队，其在单位时间内最终可能的贡献值可以表示为

$$[(E_1 \times W_1 + O) + (E_2 \times W_2 + O) + \cdots + (E_n W_n + O)] \times M$$

其中 M 的取值可以为 0～1。0 表示完全内耗掉，整个团队完全没有贡献。

注：很多人会对为什么可以用加法累积不同人的贡献这一点产生疑问，比如，架构设计师的贡献和测试人员的贡献为什么可以叠加等，这点将在附录 1 中统一进行说明。

也就是说，单位时间团队总贡献=[（A 的工程素养×A 的工作意愿+组织力量） + （B 的工程素养×B 的工作意愿+组织力量）+…]×内耗系数

其中：

- 工程素养是指人员的工程能力，比如，需求分析能力，架构设计能力，编码能力等。
- 工作意愿是指一个人愿意不愿意工作。工程素养和工作意愿乘起来才是一个人的可能

贡献。这并不难理解，一个能力不好的人，再怎么努力，贡献也不会好；而一个能力好的人，每天蒙事，其贡献也不会好。

- 组织提供的力量是指常说的组织力。比如说，组织内有比较多的重用代码库，那么不管谁都可以从中受益。探讨组织力，会使范畴发散，因此在本书中大多时候我们会忽视这个维度。
- 内耗系数是指人员彼此间贡献耗散的程度。影响因素会比较复杂。比如说，A 开发一个模块，但和 B 开发的模块有 80%重叠，这会产生内耗；再比如说，A 和 B 坐在一起工作就会吵架，这也会产生内耗。

上面这段分析可以总结为如图 1-4 所示的情况。

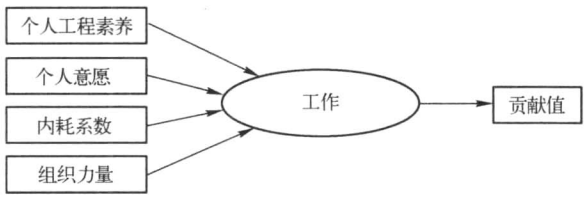

图 1-4　贡献值的分解

那么在指定的时间段，这一团体的生产率为

$$\frac{[(E_1\times W_1 + O) + (E_2\times W_2 + O) + \cdots + (E_n W_n + O)]\times M}{n}$$

上面的公式可以用来定性分析贡献值的大小，但并不能用来分析贡献值的效果。

比如说，某个团队可能在上述 4 方面都没有问题，最终也开发出了比较好的软件，但由于云计算的兴起，公司开发的这类客户端软件已经没有市场。这时候贡献值再好，也体现不出效果。

这意味着要想获得更好的结果，一要保证贡献值尽可能的大，二要保证贡献值使用在正确的方向上。控制方向性虽然重要，但与商业因素等有较大关联，本书中大多时候不会对其进行考量。

演绎法与空谈

经验主义者是不喜欢演绎法的，极端的甚至认为演绎法即是空谈。但我们不应该忘记，《相对论》与《资本论》这两个与演绎法有深刻关联的东西对世界所产生的巨大且深刻的影响。

在软件工程或者项目管理领域中，对演绎法的排斥达到了极点。人们似乎很忌讳谈及形而上学的东西，对经验的推崇则是达到了无以复加的地步。

但事实则证明这并不很有效，几十年下来，软件行业中还是一如既往的纷争不断。

比如说，Linux 之父 Linus Torvalds 每隔一段时间就会站出来贬斥一下 C++（有时还连带上面向对象），并经常用垃圾、很差这样的形容词来描述 C++。而与此同时世界上很多关键系统是用 C++成功开发出来的，拥有许多属于自己的铁杆粉丝。

这似乎很矛盾，也很难判定出是非，但当我们用演绎法来分析这类事情时，就会发现貌似对立的事情并非不可调和。Linus 主要专注的领域是内核，所以他的观点在内核领域里一定是不能漠视的，但这并不意味着 C++和面向对象就不适合游戏，富客户端，数据库程序等。

这类事实的存在意味着，我们应该切换视角，用演绎法来重新审视软件这个行业，而不能只是用有限领域中得来的经验去臆测无限范围中的事物——这是必将陷入矛盾的方法。

1.4 完美软件开发的用途

夫英雄者，胸怀大志，腹有良谋，有包藏宇宙之机，吞吐天地之志者也。

——罗贯中

完美境界是这世界上的终极力量，真的完美，绝不苍白无力。

真正的完美背后是规则的力量。

牛顿第一定律说：任何一个物体在不受任何外力或受到的力平衡时，总保持匀速直线运动或静止状态，直到有作用在它上面的外力迫使它改变这种状态为止。

这无疑是在描述一种完美状态，任何物体当然是受外力的，但谁敢说这一完美状态是苍白无力的。

完美的软件开发状态，其存在意义与上述相同。

具体而言，其真实作用是帮我们俯视全局，对种种问题洞若观火，进而把持解决问题的方向和尺度。

为获得最终的软件产品，中间必然面临种种困难。而在特定场景下，项目所面临的主要矛盾往往不同。比如，如果项目的主要问题是缺人或人员不负责任，那么加大技术培训，增强编码能力，就是缘木求鱼。

为了解决种种问题，事实上需要有人有一个软件开发的全体视图，再从中找到最关键的矛盾，接下来采取具体措施进行解决，这即是包藏宇宙之机，且腹有良谋。

而上述所有关键步骤都需要完美状态作为参照，并把握现实和完美状态间的距离——现实世界中难的并非是没有方法，而是待选方法太多，不好把握其间尺度。这就要求首先要有完美状态下的整体视图，同时不能让采取的措施与完美状态下的规则相背离，否则就会陷入迷途。

智慧与珍珠

佛家有“智珠”之说，“智珠在握”则用来形容有高深的智慧可以应对任何事情。

非常巧合的是，达成“智珠在握”状态的过程与珍珠的形成过程极度类似。

一旦有异物侵入蚌的外套膜，蚌就会不停地分泌珍珠质，最终得到的就是珍珠。

这与智慧的形成过程相似。

一个人对软件开发的全体视图可以浅陋，单薄，但一定要有。这就是形成珍珠的那粒杂质，它可以是沙粒，甚至可能是鸟屎，这都没有关系，关键是要有。

在此基础上，可以读书，可以从实践中反省，最终就会生成璀璨夺目的“珠”。

但如果没有这粒杂质，很多东西就无所依凭，到头来学到的东西就会彼此冲突，知识反而成为一种障碍。

这正是编写本书的目标之一——通过解构和逻辑链来帮助每个人形成自己的那粒杂质。

第 2 章　完美项目管理之解构

项目管理自身并不复杂，却往往被看做一门复杂的学问。究其根本，实是因为为术则日繁，为道则日远。就好比伐树的时候却从剪枝叶做起，必然只见其繁杂，而不见其根本。在这一章里，我们将对管理的根本命题进行解构。

如果你曾经对下面这些问题困惑过，那么在这里，你将找到一份逻辑上说得通的答案。

为什么同样人数，天分又差不多的团队，爆发出来的战斗力却有天壤之别？

为什么看上去很好的量化管理，一旦导入却会天怒人怨？

为什么团队的成员会从朝气蓬勃，变得得过且过？

为什么开源项目没有项目经理，往往也运作得很好？管理真的有价值么？

为什么工厂式的开发不适合软件？

项目经理究竟要不要懂技术？

……

2.1　项目管理的存在意义

君子务本，本立则道生。

——孔子

2.1.1　价值根源

价值是一个可以引起无数纷争的词语。在商业社会中，价值的界定就更为艰难。即使完全相同的两个程序，广为人知的可能价值亿万，没于静室的可能一文不值。所以当我们考察项目管理的价值时，我们必须剥离一些东西，使价值的范畴更加清晰。

在后续各个章节中，对价值进行讨论时，我们将要剥离的是软件的商业价值。也就是说，在完美的世界中，我们关注的是做最好的软件，但并不关注最好的软件是否有市场。后者显然是有意义的，但并不在我们的考察范围之内。

在剔除商业价值之后，我们来看一下软件的一个根本特质。

由于软件是一种固化的思维，而思维固化的媒介必然是代码，因此我们可以讲：

软件与代码相等价，是同一事物的两面。也即说代码可以表征软件的一切价值。

这也就意味着，如果项目管理确实有意义，那么项目管理必须直接或间接地对代码施加影响。

显然的，项目管理很难直接对代码产生影响，这也就意味着项目管理自身并不直接创造价值，必须以他人为媒介才可能最终产生自己的价值。

项目管理者对他人可能产生的影响可以分为两类：一是对个人的影响；二是对组织的影响。

- 对个人的影响可以体现在对消极、排斥异己、懒惰、好高骛远等负面情绪的遏制上；也可以体现在肯定成绩、表达信任这类激励方法上。

- 对组织的影响则可以体现在对谋而无断、职责不清、流程不清这类负面组织行为的遏制上；也可以体现在创建共识、对组织意识形态进行引导这类避免纷争的手段上来。

上述两点正是项目管理的价值根源。

在下一节里，我们将对上述的价值根源做一些定性的分析，但在进行定性分析前，有必要对项目管理这一常用词语的范畴做一点补充说明。因为我们推导出来的价值根源与大多数人的认识可能已经有了较大的偏差。

我们来看一下 PMBOK（Project Management Body of Knowledge，项目管理知识体系）对项目管理的定义：

项目管理就是把各种知识、技能、手段和技术应用于项目活动之中，以达到项目的要求。项目管理是通过应用和综合诸如启动、规划、实施、监控和收尾等项目管理过程来进行的。

——PMBOK

这个定义中的关键词是“各种”，在 PMBOK 中，“各种”被定义为诸如：

法律、金融、标准、规章制度等，甚至也包含文化和社会环境、政治环境等。

PMBOK 和上述的推导结果间的一个显然差异是：PMBOK 强调的项目管理的职责更多的是向外看的，而我们推导结果是项目管理的价值根源在于对内部的人或组织施加影响。

造成这种差异的一个原因是 PMBOK 考虑的是现实的世界，并没有如我们一般剥离商业价值这一维度。而如何在两者之间寻找平衡，则是从完美世界回归现实时需要考虑的问题，需要具体情况具体分析，我们这里就不做进一步讨论了。

2.1.2 定性分析

让我们回到第 3 章中提到的公式：一个人的工程素养为 E，一个人的工作意愿为 W，组织所能提供的力量为 O，内耗系数为 M，那么对于一个拥有 n 个人的团队，其在单位时间内最终可能贡献值可以表示为

$$[(E_1 \times W_1 + O) + (E_2 \times W_2 + O) + \cdots + (E_n W_n + O)] \times M$$

其中 M 的取值可以为 0 到 1。0 表示完全内耗掉，整个团队完全没有贡献。

没有管理者时，在指定的时间段里，这一团体的生产率为

$$\frac{[(E_1 \times W_1 + O) + (E_2 \times W_2 + O) + \cdots + (E_n W_n + O)] \times M}{n}$$

假设加入了一个管理人员，由于管理人员不直接创造价值，生产率的公式变为

$$\frac{[(E_1 \times W_1 + O) + (E_2 \times W_2 + O) + \cdots + (E_n W_n + O)] \times M}{(n+1)}$$

任何管理手段几乎不可能对工程素养 E 产生影响，短期内也很难影响组织力 O，因此如果工作意愿 W 或者内耗系数 M 没有变化，则生产率必会降低。反过来讲，项目管理者必须对工作意愿 W 或者内耗系数 M 施加正面影响才能促进生产率的增长，进而阻止生产效能的降低。

为做进一步分析，我们引入经济学中边际价值和边际价值递减的概念。边际价值是指在其他条件不变的前提下，增加一单位要素投入所增加的产品的价值。边际价值递减是指超过某一水平之后边际投入的边际产出下降。边际价值递减可以帮我们推导出下面的结论。