导论

软件工程定义：将系统化的、严格约束的、可量化的方法应用于软件的开发、运行和维护，即将工程化应用于软件；对上述方法的研究。**软件维护：**软件开发完成后，对软件的问题进行解决，增加功能，与开发阶段有很重要的联系。**软件工程知识域：**软件需求、设计、构造、测试、维护、配置管理、工程管理、软件工程过程、软件工程模型与方法、软件质量、软件工程职业实践、软件工程经济学、计算、数学、工程基础。**软件工程 VS 计算机科学：**目标（在资源的限制条件下构建满足用户需求的软件系统 VS 探索计算和建模方法，并改进）、产品、进度和时间表、关注（如何为用户实现价值 VS 软件本身运行原理（时间空间复杂度、算法正确性））、变化程度、需要的其他知识（SE：相关领域知识 CS：数学）**软件工程 VS 计算机程序设计：**软件工程存在于各种应用中，存在于软件开发的各个方面。程序设计通常包含了程序设计和编码的反复迭代过程，是软件开发的一个阶段；软件工程为图对软件项目的各方面做出指导，从软件的可行性分析直到软件完成后的维护工作。

没有银弹

主要思想：没有任何一种单纯的技术或管理上的进展，能够独立地承诺十年内使生产率、可靠性或简洁性获得数量级上的进步；所有大家看到的技术、管理方法都不会给软件开发带来意想不到的效果；软件开发在根本上就是困难的（Brooks 认为根本困难是固有的概念复杂性）**根本任务：**打造由抽象软件实体构成的复杂概念结构**次要任务：**使用编程语言表达这些抽象实体，在空间和时间限制内将它们映射成机器语言。**没有银弹的主要观点：**相对必要任务而言，除非次要任务占所有工作的 9/10，否则即使全部次要任务的时间缩减到零，也不会给生产率带来数量级上的提高。（即使硬件发展使次要任务越来越容易解决）**软件项目现状：**常常看似简单明了的东西，却有可能变成一个落后进度、超出预算、存在大量缺陷的怪物。（软件开发 VS 硬件开发：不是软件发展慢，是硬件发展太快）**探寻软件产业发展的问题：软件开发中的根本问题**软件特性中固有的困难，规格化、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。**次要问题**出现在目前生产上的，但并非与生俱来的困难。一个相互牵制关联的概念结构是软件实体必不可少的部分，它包括数据集合、数据条目之间的关系、算法、功能调用等。这些要素本身是抽象的，体现在相同的概念构架中，可以存在不同的表现形式。尽管如此，它仍然是内容丰富和高度精确的。**软件系统中无法规避的内在特性：**复杂性、不一致性、可变性、不可见性。**复杂性：**1.软件实体可能比任何由人类创造的其他实体都要复杂，没有两个软件是相同的（如果有，会将它们合并）；2.数字计算机本身就比人类建造的大多数东西复杂。计算机拥有的大量状态使得构思、描述和测试都非常困难。软件系统的状态又比计算机系统状态多若干个数量级；3.软件实体扩展是不同元素实体的添加，复杂度非线性增长；4.软件的复杂度是必要属性，不是次要因素，抽掉复杂度的软件实体描述常常也去掉了一些本质属性。5.*复杂度造成的软件开发问题：*沟通困难导致产品瑕疵、成本超支和进度延迟；列举和理解所有可能的状态十分困难，影响产品的可靠性；6.函数调用变得困难，导致程序难以使用；7.程序难以在不产生副作用的情况下用新函数扩充；8.造成很多安全机制状态上的不可见性；9.复杂度引发管理上的问题：全面理解问题变得困难，妨碍了概念上的完整性；10.所有离散出口难以寻找和控制；11.引起了大量学习和理解上的负担，使开发慢慢演变成了一场灾难（对策：信息隐藏策略）**不一致性：**软件开发面对的复杂度往往随心所欲、毫无规则可言，来自若干必须遵循的人为惯例和系统。软件开发面对的是人，不是上帝；很多复杂性来自保持与其他接口的一致。**可变性：**软件实体会遭受到持续的变更压力。软件很容易修改（纯粹思维活动的产物，可以无限扩展）；软件的变更来自于人们要求扩展、更改功能和硬件的变化，软件与整个社会联成一体，后者在不断变动，它强迫软件也跟着变动。**不可见性：**软件不可见，无法可视化；软件的客观存在不具有空间形体特征。这限制了个人的设计过程，阻碍了相互之间的交流。**当年的银弹：**Ada 等高级语言、OOP、人工智能、专家系统、“自动”编程、图形化编程、程序验证、环境和工具、工作站、购买和自行开发、需求精炼和快速原型、增量开发——增长而非搭建系统、卓越的设计人员**没有银弹的影响：**软件开发本质的认识；软件过程。

开源软件

大教堂与市集：一种是封闭的、垂直的、集中式的开发模式，反映一种由权利关系所预先控制的极权制度；而另一种则是并行的、点对点的、动态的开发模式。自由软件不仅是乌托邦的理想，而在开发模式上真正代表“先进生产力”，代表历史发展趋势的必然。**建造教堂和集市模式的核心区别**：在建造教堂模式的编程模式看来，错误和编程问题是狡猾的、阴险的、隐藏很深的现象，花费几个月的仔细检查，也不能给你多大确保把它们都挑出来的信心，因此很长的发布周期、和在长期等待之后开没有得到完美的版本发布所引起的失望都是不可避免的；以市集模式观点来看，在另一方面，我们认为错误是浅显的现象，或者至少当暴露给众多热切的协作开发人员，让他们来对每个新发布测试的时候，它们很快变得浅显了，所以我们经常发布来获得更多的更正，作为一个有益的副作用，如果你偶尔做了一个笨拙的修改，也不会损失太多。*两种版本：*Linux 内核的版本在编号上做了些处理，让用户可以自己选择是运行上一个“稳定”的版本，还是冒遇到错误的险而得到新特征。存在两个选择的事实让二者都很吸引人。**Linux 开发模式：**每一个好的软件的起因都是挠到了开发者本人的痒处；好程序员知道该写什么，伟大的程序员知道该重写（和重用）什么；计划好抛弃，无论如何，你会的（*常常在第一次实现一个解决方案之后才能理解问题所在，第二次才足够清楚怎样做好它*）；如果你有正确的态度，有趣的问题会找上你（*在思考、审视一些你感兴趣的软件时，会有新的更好的想法*）；当对一个程序失去兴趣时，最后的责任是把它传给一个能干的后继者（*在开源软件的开发中*）；把用户当作协作开发者是快速改进代码和高效调试的有效方式；早发布、常发布、听取客户的建议（*尽量早尽量频繁的发布是 Linux 开发模式的一个重要部分。Linux 的创新并不是这个，而是把它扩展到和他所开发的东西的复杂度相匹配的地步，而且因为培育了协作开发者的基础，比其他任何人更努力地充分利用了 Internet 进行合作，所以这确实能行*）；如果有一个足够大的 beta 测试人员和协作开发人员基础，几乎所有的问题都可以被快速找出并被一些人纠正；聪明的数据结构和笨拙的代码比相反的搭配工作得更好；如果像对待最宝贵的资源一样对待 beta 测试员，他们就会成为你最宝贵的资源；想出好主意和从你的用户那里发现好主意都是好事，有时候后者更好；最重要和有创新的解决方案常常来自于你认识到你对问题的概念是错误的；最好的设计不是再没有什么东西可以添加，而是再没有什么东西可以去掉；任何工具都应该能以预想的方式使用，但是一个伟大的工具提供没料到的用法；写任何种类的网关型程序时，多费点力，尽量少干扰数据流，永远不要抛弃信息，除非接收方强迫这么做；如果你的语言一点也不像是图灵完备的，严格的语法会有好处；一个安全系统只能和它的秘密一样安全，当心伪安全；要解决一个有趣的问题，请从发现让你感兴趣的问题开始（*最好的开发是从作者解决每天工作中的个人问题开始的，因为它对一大类用户来说是一个典型问题，所以能推广开来*）；如果开发协调人员有至少和 Internet 一样好的媒介，而且知道怎样不通过强迫来领导，许多头脑将不可避免地比一个好（*自由软件的将来将属于那些知道怎样玩 Linus 的游戏的人，把大教堂抛之脑后拥抱市集的人，这并不是说个人的观点与才气不再重要，而是，自由软件的前沿将属于从个人观点和才气出发的人，然后通过共同兴趣自愿社团的交流建造来扩展*）。**Linus 定律：**如果有足够多的眼睛，所有的错误都是浅显的；如果有一个足够大的 beta 测试人员和协作开发人员的基础，几乎所有的问题都可以被快速的找出并被一些人纠正。**Delphi 效应（神庙效应）：**一群同样内行的（或同样无知的）观察者的平均观点比在其中随机挑选一个来得更加可靠。**Brooks 定律**：向一个进度落后的软件项目中增加开发人员只会让它更加落后，他声称项目的复杂度和通讯开销以开发人员的平方增长，而工作成绩只是以线性增长，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要大。**集市风格的必要的先决条件：**不能以一个市集模式从头开发软件，我们可以以市集模式测试、调试和改进，但是以市集模式从头开始一个项目是非常困难的；当你开始创建社团时，你需要的是一个有趣的项目；协调人要能把从他人那里得到的好的设计重新组织起来；市集项目的协调人或领导人必须有良好的人际和交流能力；为了建造一个开发社团，你需要吸引人，你所做的东西要让他们感到有趣，而且要保持他们对他们正在做的工作感到有趣，保持他们对他们正在做的工作感到高兴。技术方面对达成这些目标有一定帮助，但这不是全部，个人素质也有关。**交流与沟通：**在《人月神话》中，Fred Brooks 观察到的 Brooks 定律被普遍当作真理，但**如果 Brooks 定律就是全部，那 Linux 就不可能成功。“忘我(egoless)的编程”**中，Weinberg 观察到在开发人员不顾固保守自己的代码，鼓励其他人寻找错误的地方，软件改进的速度比其他地方有戏剧性的提高。虽然编码是个人活动，真正伟大的工作来自利用整个社团的脑力。在一个封闭项目中只利用自己脑力的人会落在知道怎样创建一个开放的、进化的，成百上千的人在其中查找错误、进行修改的环境的开发人员之后。**Linux 与 Internet：**Linux 是第一个有意识的成功的利用整个世界作为它的头脑库的项目，Linux 的孕育和万维网的诞生相一致并不是一个巧合，而且 Linux 在 1993-1994 的一段 ISP 工业大发展和对 Internet 的兴趣爆炸式增长的时期中成长起来，Linux 是第一个学会怎样利用 Internet 的新规的人；Linux 世界的行为更像一个自由市场或生态系统，由一大群自私的个体组成，它们试图取得(自己)最大的实效，在这个过程中产生了比任何一种中央计划都细致和高效的自发的改进的结果；Linux 黑客取得的最大化的“实际利益”不是经济利益，而是无形的自我满足和在其他黑客中的声望，（有人会说他们的动机是利他的，但这忽略了利他主义本身就是利他主义者的一种自我满足的形式）。**解释开源与 Brooks 定律的矛盾**：用 Internet 沟通代价很小；开源项目的通讯结构是核心开发者、beta 测试人员、协助开发者。**Brooks 定律基于一个前提：**每个人都与其他所有人交流。但是在开源项目中，外沿的开发者做的实际上是平行分离的子项目，彼此交流很少；代码变动和 bug 报告都流经项目的核心，只有在小小的核心团体中全面的 Brooks 成本才有效。**结论：**也许最终自由软件文化将胜利，不是因为协作在道德上是正确的或软件“囤积居奇”在道德上是错的，而仅仅因为商业世界在进化的军备竞赛中不能战胜自由软件社团，因为后者可以把更大更好的开发资源放在解决问题上。

开源经济学

很多商科技公司投入巨额资金发展开源软件，而通常开源软件本身免费。这些公司并不是放弃资本主义，而是认为这是个好的商业策略。**替代物品&互补物品：**替代物品是首选商品太贵时会改买的另一种东西，互补物品是通常会和其它产品一起购买的产品。当商品的价格下降时互补物品的需求会增加。开源也不能免于经济法则。**为什么公司要支持开源：**可以用互补物的原理来解释。聪明的公司试图让产品的互补物普及化，产品的需求就会上升而你就可以卖贵一点然后赚更多钱。要让开源软件成为自己产品的互补品而不是替代品。*例子：*IBM：企业软件—IT 顾问；微软：MS-DOS—PC；Xbox：游戏—普及的 PC 硬件和 DirectX；Netscape：服务器—Web 浏览器；Sun：推广并发展免费软件让软件普及化；推广 Java 及其架构和让硬件普及化；Sun 和 HP 研究 Gnome：让软件普及化然后由硬件赚更多钱。**软件与硬件的关系**：软件很容易让硬件普及化，而反过来很难。软件不能互换，除非切换成本为零，否则无法完成互换。

高科技市场技术采用生命周期

连续性创新与非连续性创新：任何时候，我们面对的新产品需要我们改变自己一贯的行为模式，或者需要对我们目前依赖的产品或服务进行改变时，这种创新在学术上称为非连续性创新或破坏性创新。相对的创新称为连续性创新或延续性创新，指产品正常升级，不需要我们改变行为。两者之间是连续的，很多创新不容易明确区分。创新的形式与技术难易程度没有关系，但创新的形式与商业价值有巨大的关系。传统行业引入非连续性创新速度很慢，并且伴随惊恐；高科技行业则经常性地引入非连续性创新。**技术采用生命周期：**创新者（2.5%）是新技术的热衷者，科技是他们生活中的最大乐趣，而并不在意这些技术能够在他们的生活中提供什么功能。创新者并不多，是大家公认的最有能力对新技术进行早期评价的人；对企业非常重要，他们的认同是下一步市场开发的先决条件，是生命周期的守门人。早期采用者（13.5%）是高瞻远瞩的有远见者，拥有洞察力将新技术与战略机遇结合起来。这些人是企业和政府里的创新者。他们想用非连续性创新产品开辟一个新的未来，希望能成为第一个开发产品潜力的人，并将它发展成无法超越的巨大竞争优势。早期大众（34%）是实用主义者，对技术持中立态度，相信演变而非革命。只有看到新产品能够提高工作效率，又能听到周围他们信任的人的意见时，才会接纳新产品。后期大众（34%）是保守主义者，对非连续性创新有一种本能的抗拒，对于他们是否能从创新产品中得到益处表示悲观。与新的进步来说，他们对传统更加信任。落后者（16%）是怀疑主义者，讨厌高科技产品，而且很高兴远离一个旁冷眼观看那些被人们鼓吹的高科技产品的结局。**鸿沟：**当真正有价值的非连续性创新产品进入市场后，首先欢迎它的是由新技术热衷者和有远见者构成的早期市场，随后就陷入鸿沟，销售萎靡，几乎进入滞销状态。鸿沟是由早期市场的有远见者和实用主义者之间的差异构成的。有远见者相信直觉、支持变革，愿意冒险，认为实用主义者太平庸；实用主义者认为有远见者太冒险。**必须跨越：**高科技财富主要来源于主流市场，跨越鸿沟是企业的必然选择。**高科技市场的技术采用生命周期模型：***早期市场：*一个激动人心的时期，因为消费者是那些新技术热衷者和有远见者，成为首批购买新产品的消费者；*鸿沟：*这是一个令人沮丧的时刻。这时早期市场变小，而大众市场却仍然不能接受不成熟的新产品；*保龄球道：*这是在大众市场内寻求产品立足之地的时期。企业可以根据消费者的具体需求，生产整体产品来满足他们的需要；*龙卷风暴：*这是一个大众接受时期，因为此时的大众市场已经过渡到了使用新产品的阶段；*主流：*这是一个市场发展繁荣的时期。因为社会基本接受新的产品，而下一步的目标则是更好的挖掘潜力。*生命终止：*高科技产品生命终止来的太快。**跨越鸿沟的策略：**提供完整产品，即能够保证满足目标消费者需求的最少量产品。最核心的要求是满足真实用户的真实需要。在未能验证完整产品前，首先要和其它合作伙伴一起提供完整产品，互相借鉴。在完成产品验证后，设法获得投资，消除所有竞争对手。*D-Day 营销策略：*将大量的精力和资源集中在一个有限的市场空白中。关注一个细分市场比大规模的投入推广效果要好很多。**保龄球道：**产品已经在大众市场内有了立足之地，但是还没有被整个市场接受。对很多消费者而言此时的产品存在优势，但还没有强烈的理由购买，可以继续使用旧的产品。虽然产品跨越鸿沟，但还没有证明它对整个市场而言都是完整产品，此时应当设法获取收入，争取市场领导者地位，这对争取实用主义者的购买将有很大作用。**市场领导者：**一旦市场领导者出现，在自由市场上秩序将自然出现。通常市场领导者的市场份额较大，为第三方厂商提供了巨大市场，因此*第三方厂商*愿意支持市场领导者。当第三方厂商出售产品和服务时，完善了市场领导者的产品。客户通过购买第三方厂商产品，使得他们在市场领导者产品上的投资更加有价值，从而推动更多地购买市场领导者产品。市场领导者有权利将同样的产品以更高的价格出售，获取超额利润，甚至它的产品有缺陷。由于出货量大，它的单位成本低；实用主义者愿意购买市场领导者的产品，因为市场领导者的产品不一定是最好的，但一定是最稳定的。因此领导者的营销成本也很低。消费者、第三方厂商、市场领导者三者形成*正反馈*。**龙卷风暴：**特别的需求在某一个特定时间点大爆发出来，标志着事件是形成了独立的产品类别。此阶段，一家公司成为市场领导者，其它竞争对手全部成为备份，市场领导者的员工会非常繁忙。在龙卷风暴阶段，硬件厂商关注供应链管理，互联网厂商关注运维质量。对 IT 部门和实用主义者来说，新技术是需要的，但是采用新技术不能过早也不能过晚。过早意味着风险，过晚使得公司处于不利的竞争地位，个人将失去竞争能力，公司将陷入高科技产品的生命周期尾端，维护成本会大大增加。**大猩猩、黑猩猩和猴子：**市场领导者成为大猩猩，一到两家市场领导者的竞争者成为黑猩猩，其它占有少量市场份额的企业被称为猴子。这种市场格局是由实用主义者支持市场领导者的本性决定的。没有这样的领导者，市场将不稳定。在市场领导者的确立中，口碑极端重要。一旦实用主义消费者开始支持和购买市场领导者的产品，这个过程是一个自反馈过程，可以确保市场领导者获取它的产能所能支撑的市场份额。大猩猩无法吃掉所有市场，黑猩猩更为和善，用户可以和它们商议价格、服务、安装等。猴子在龙卷风暴时进入市场，不需要进行基础架构探索和研发，不需要教育市场和客户。最好的策略是克隆大猩猩的产品，卖便宜一些。市场也希望能有低成本的替代品。**主衝：**大规模半导体集成电路的发展使得产能很快会超越需求，购买选择权回归到客户，厂商又要为争夺客户而努力。因此必须改变市场策略。此时的研发策略是完整产品+1，多考虑产品的附加值，向自己要做什么才能在不大规模增加成本的情况下，让客户愿意付更多的钱。停留在主衝的时间由替代的非连续性创新的出现时间决定。主衝过后，市场会收缩为服务产业。**两种组织架构模型：***复杂系统：*客户多为大型企业，基础数量少，交易次数少，平均价格高，客户需求定制化；*大量交易：*客户多为消费者，企业核心技术不需要是高科技，而是符合大众市场的一般标准品。客户群以百万计，个别客户不是焦点，单次交易金额少，频率高，总数大。*研究/设计/采购/制造/市场/销售/服务策略：*定性方案/定量分析、模块或子系统的整合/可整合产业支撑解决方案的模块、边际价值/平均价值角度、自我适应方法论/确定的流程、价值观协调/品牌与促销、高接触说服/低接触配销、无限制咨询服务/封闭式交易。

创新者的窘境

颠覆是指一个仅有有限资源的新生公司，逐渐具有向占据优势的大企业挑战实力的过程。颠覆式创新源于低端或新市场的立足点。颠覆式创新精华在于其对管理短视的警醒，但市场变化多端，除了理论还需要进行详实的竞争战略分析。**为什么管理良好的企业会遭遇失败？**所有失败案例都具备一个共同点，那就是导致企业失败的决策，恰好是在领先企业被广泛普为世界上最好的企业时做出的。解释：它们实际上一直管理不善；这些遭遇失败的企业的管理已经做到了极致，但它们在大获成功后做出决策的方式，却最终埋下了它们日后失败的种子。**良好的管理：**听取客户的意见；大力投资客户表示希望得到进一步改善的技术；争取更高的利润率；以更大的市场而不是更小的市场为目标。**延续性技术：**它们都根据主要市场的主流客户一直以来所看中的性能层面，来提高成熟产品的性能。这些技术有可能是突破式的；也有可能是渐进式的。大部分技术改进都是延续性技术。最具突破性、最复杂的延续性技术，也很少会导致领先企业失败。**破坏性（颠覆性）技术：**性能要低于主流市场的成熟产品，却拥有一些边缘客户所看中的新特性。基于破坏技术的产品通常价格更低、性能更简单、体积更小、而且通常更便于客户使用。最初应用于非主流市场或新市场，积累经验并得到足够投资后，提高产品性能，最终占领主流市场。率先进入一个新的市场具有先发优势。领先企业的失败大部分由破坏性技术导致。**成熟企业无法很快使用破坏性技术的原因：***企业的资源分布取决于客户和投资者：*绩效最好的企业都建立了扼杀没有得到客户认可的理念的成熟体系。他们很难投入足够的资源来开发破坏性技术，他们发现其最终无法得到客户的认可时，一切为时已晚。*价值网络：*一种大环境，企业正是在这个大环境下确定客户需求，并对此采取应对措施，解决问题，征求客户的意见，应对竞争对手，并争取利润最大化。特定的产品性能属性：每一个价值网都会按照重要性的高低对不同的产品性能属性进行排序，从某种程度上来说，价值网的界定就是由这种独特的排序所决定的；特定的成本结构：也叫盈利模式。公司的成本和盈利，以及要求的毛利率；特定的组织能力：Nokia 的核心人员来自电信和通信行业：苹果的核心人员来自计算机行业。Nokia 的核心流程是工厂制造-渠道销售，核心能力是制造：苹果的核心流程是产品设计-制造外包-品牌营销，核心能力是产品。价值网络决定了资源分配，成熟企业以客户为向导的资源分配和决策流程，决定了企业内部高层管理者无法真正实现企业转型，主导力量来自机构以外的力量。**面对破坏性创新，成熟企业的管理决策过程（硬盘行业例子）：**破坏性技术首先由成熟企业研制成功，由成熟企业的工程师利用非正规渠道获得资源研发，而非企业高层发起。针对新的市场，性能有降低；市场营销人员收集公司主要客户的反馈：它们不需要；成熟企业加快对延续性技术的开发步伐以满足主流客户；新企业已经出现，破坏性技术市场在反复尝试中逐渐成型，可能是由那些在成熟企业不得志的工程师创办的；新兴企业向高端市场转移；成熟企业无能为力。*小市场并不能解决大企业的增长需求：*要维持股价，为员工创造内部晋升机遇，成功的企业必须不断发展。成熟企业在试图进入更新、更小，但日后注定将发展壮大的市场时，会遇到困难。为了保持增长率，这些企业必须专注于大市场。*无法对并不存在的市场进行分析：*详实的市场研究数据和良好的规划以及之后的按计划执行的流程，构成了良好管理的基本特征。对于那些强调良好管理的企业，它们面对破坏性技术是变得束手无策，因为它们要求的数据并不存在。*大型机构能力无法应对破坏性创新：*一个机构的能力独立于机构内部工作人员的能力而存在。成熟企业的流程和价值观通常无法进行破坏性创新。*能力框架（RPV）：*资源：人员、资金、技术、客户、供应商等；流程：制造过程、产品开发、采购、资源分配等；价值观：在确定决策优先级别时遵循的标准。哪个订单更重要，哪个客户更重要，可接受的毛利率，规模（股价）。*技术供给可能并不等同于市场需求：*成熟企业往往可以提供更好的技术性能。尽管最初的性能较弱，但破坏性技术将在日后变得极具竞争力。**应对颠覆性创新：***把开发破坏性技术的职责赋予存在客户需求的机构：*只要企业能够明确的知道它们的客户需要的最终产品是什么，企业就愿意为技术上存在风险的项目投入巨额资金。但面对客户明确说“不”的破坏性技术时，管理者有两种选择：说服企业内的每个人，这对企业的发展有长期战略意义；创建一个独立机构，使这个机构直接面对确实需要这种技术的新兴客户群体。*设立一个独立的、满足小型市场发展的机构；设立一个能够欣然接受较小收益的独立小型机构；为失败做好准备：*不要在第一次就用尽所有资源，因为很难在第一次尝试中就抓住正确方向。迭代，快速反馈，调整方向。**为什么不能在市场明确后投入力量进入市场：**数据证明先行者有巨大的领先优势；可以使用鸿沟理论来解释。**其它应对破坏性创新时需要考虑的问题：***怎样判断出某技术是否具有破坏性？*它的性能曲线日后有可能与主流市场需求相交汇；*新产品的市场到底在哪里？*往往不在主流市场；*无法通过市场调研得到，必须进入市场，不断尝试；*这是一个学习计划，不是一个执行计划，必须做好失败 2-3 次的准备；*应该采取什么样的产品技术和经销策略？*新产品设计理念：体积要小，结构简单，使用方便；能以较低成本迅速对产品特色、功能和外形进行变更的产品平台；确定一个较低的定价点；破坏性产品将会重新定义主要经销渠道，经销策略的第一步就是寻找或创造新的经销网络。**颠覆式创新理论容易被忽视或误解的部分：***颠覆式创新是一个过程：*“颠覆式创新”常被理解为处于某一固定阶段的产品或者服务，而事实上，这个名词所指的应该是产品或服务的发展过程；*颠覆者往往建立与优势企业非常不同的商业模型；一些颠覆式创新成功了，但并非所有：*误解：<u>一家公司的成功被视为是否符合颠覆式理论的决定因素</u>。事实上，成功与否不是颠覆式理论定义中的元素：并不是所有颠覆式途径都指向成功的出口，也并不是所有成功人士走的都是颠覆式创新之路；*“要么颠覆，要么被颠覆”的口头禅会误导我们：*当颠覆式现象发生时，优势企业并不需要立刻回应，他们需要注意不要采取过激措施、破坏仍有利可图的现有业务。相反，应该通过投资持续创新，继续加固与核心客户群的纽带。可以创建一个企业部门来全心全意应对颠覆现象带来的机遇。调研显示，新企业的成功很大程度上取决于将新业务与核心业务分开。这也意味着优势企业将同时运营两种完全不同的业务。

软件工程与管理

软件工程的核心问题：软件工程的管理视角，**成功是否可以复制**？**1.软件过程**，软件过程视为了实现一个或者多个事先定义的目标而建立起来的一组实践的集合，这组实践之间往往有一定的先后顺序，作为一个整体来实现实现定义的一个或者多个目标；**2.生命周期模型**，对软件过程的一种认为的划分。软件工程的技术视角，问题是否可以解决得更好。**软件项目管理概念：**三大关键要素，目标、状态纠偏；典型三大目标，成本、质量、工期；软件项目管理是应用方法、工具、技术以及人员能力来完成软件项目，实现项目目标的过程。估算、计划、跟踪、风险管理、范围管理、人员管理、沟通管理等等。广义软件过程：理论基石：软件产品和服务的质量，很大程度上取决于生产和维护该软件或者服务的过程的质量。广义软件过程包括技术、人员以及狭义过程。**生命周期模型与软件过程：**区别和联系生命周期模型是对一个软件开发过程的人为划分；生命周期模型是软件开发过程的主框架，是对软件开发过程的一种粗粒度划分；生命周期模型往往不包括技术实践。**典型生命周期模型：**瀑布模型、迭代式模型、增量模型、螺旋模型、原型法等等。软件过程管理：管理对象是对软件过程；管理的目的是为了让软件过程在开发效率、质量等方面有着更好性能绩效；软件项目管理：产品生产管理 ；软件过程管理：流水线的设计、建设、维护、优化以及升级改造。**软件开发本质难题是：**不可见性、复杂性、可变性、一致性；进一步分析：三个本质难题因项目而异、四大本质难题相互促进、本质难题变化带动软件方法（过程）演变。**软件发展三大阶段：软硬件一体化阶段（50 年代~70 年代）**软件完全依附于硬件（软件应用典型特征：软件支持硬件完成计算任务，功能单一，复杂度有限，几乎不需要需求变更；软件开发典型特征：硬件太贵，团队以硬件工程师和数学家为主），软件作坊（软件应用典型特征：功能简单，规模小；软件开发典型特征：很多非专业领域的人员涌入软件开发领域，高级程序语言出现，质疑权威文化盛行；典型实践：code and fix）；**软件成为独立的产品（70 年代~90 年代）**，（软件应用特征：摆脱了硬件束缚（OS），功能强大，规模和复杂度剧增，普通人成为软件用户，需求多变，兼容性要求，来自市场的压力；典型软件过程和实践：1.形式化方法 2.结构化程序设计和瀑布模型（问题和不足：形式化在扩展性和可用性方面存在不足，瀑布模型成为一个重文档，慢节奏的过程）3. 成熟度模型）；**网络化和服务化（90 年代中期迄今）**，（软件应用特征：功能更复杂，规模更大，用户数量急剧增加，快速演化和需求不确定，分发方式的变化（SaaS）；软件过程与实践：1. 迭代式，大型软件系统的开发过程也是一个逐步学习和交流的过程，软件系统的交付不是一次完成，而是通过多个迭代周期，逐步来完成交付；2. 雪鸟会议和敏捷宣言，个体和互动胜过流程和工具，可以工作的软件胜过详尽的文档，客户合作胜过合同谈判，响应变化胜过遵循计划，尽管右项有其价值，我们更重视左项的价值。3. XP（eXtreme Programing） 方法偏重于一些工程实践的描述，SCRUM 管理框架和管理实践，Kanban 精益生产（丰田制造法）的具体实现可视化工作流、限定 WIP、管理周期时间。4. 开源软件开发方法：是一种基于并开发模式的软件开发的组织与管理方式 代码管理：严格的代码提交社区审核制度 内部开源（inner source）众包(Crowdsourcing)）**从软件发展的三大历史阶段以及软件过程的演变当中，我们可以总结出哪些规律性的东西？**关于人月神话，管理方法与技术的结合，软件过程滞后于软件应用的发展——问题驱动，迭代式是主流方法，对方法标签敏捷或者非敏捷是有害的。**当前软件现状：软件应用典型特征：**进一步服务化和网络化，用户需求多样性进一步凸显，软件产品和服务的地位变化，错综复杂的部署环境，近乎苛刻的用户期望（多：功能丰富，快：快速使用，及时更新，好：稳定，可靠，省：用户的获得成本低，最好免费）。**软件开发典型特征：**空前强大的开发和部署环境——XaaS，IaaS，PaaS，SaaS，FaaS；盛行共享文化；潜在支撑获得了长足进步（AI，Bigdata， Cloud，etc.）。**典型 DevOps 实践和方法：**方法论基础是敏捷软件开发、精益思想以及看板 Kanban 方法，以领域驱动设计为指导的微服务架构方式，大量虚拟化技术的使用，一切皆服务 XaaS(X as a Service)的理念指导，构建了强大的工具链，支持高水平自动化。

CMMI 集成能力成熟度框架

来源于三个模型：软件工程 sw-cmm、系统工程 EIA/IS、集成化产品和过程开发 IPD-CMM。**CMMIv1.2 分为哪三个集群：**面向开发的 CMMI（CMMI-DEV）面向采购的 CMMI（CMMI-ACQ）面向服务的 CMMI（CMMI-SVC）。**CMMI 的成熟度等级表示：**1）**阶段式成熟度等级表示法：**初始级、已管理级、已定义级、量化管理级、持续优化级；2)**连续式表示法：**不完整级、已执行级、已管理级、定义级、量化管理级、优化级。**一种模型：**CMMI 是一个由 SEI 发布的过程域模型。一组最佳实践依据不同的目标分组为过程域，将这些过程域排序，就描绘出了一条路径，从不成熟到成熟的演化路线图（因此 CMMI 不是开发模型）。再根据过程域的特征，归纳出模型。**两种表现形式：***阶段性表现形式*是有无的问题，由成熟度维度来表现。***初始级：***检测是否为初始级：如何面对危机？遵循过程还是单纯的编码和测试。企业对项目的目标与要做的努力很清晰，项目的目标得以实现。但是由于任务的完成有很大的偶然性，企业无法保证在实施同类项目的时候仍然能够完成任务。企业在一级上的项目实施对实施人员有很大的依赖性。改进：加强项目管理；保证遵守过程，质量保证小组；变更控制。***可重复级：***组织面对新的挑战（新的方法、工具，新的项目）时将存在危机。企业在项目实施上能够遵守既定的计划与流程，有资源准备，权责到人，对相关的项目实施人员有很大的依赖性。改进：加强项目管理；保证遵守过程，质量保证小组；变更控制。***可重复级：***组织面对新的挑战（新的方法、工具，新的项目）时将存在危机。企业在项目实施上能够遵守既定的计划与流程，有资源准备，权责到人，对相关的项目实施人员有很大的依赖性。改进：加强项目管理；保证遵守过程，质量保证小组；变更控制。

员有相应的培训，对整个流程有监测与控制，并与上级单位对项目与流程进行审查。企业在二级水平上体现了对项目的一系列的管理程序。这一系列的管理手段排除了企业在一级时完成任务的随机性，保证了企业的所有项目实施都会得到成功。改进：建立过程小组，提高软件开发过程；建立软件开发过程的架构；引入软件工程的方法和技术。***定义级***：过程定性，很少数据说明过程的有效性。企业不仅能够对项目的实施有一整套的管理措施，并保障项目的完成；而且，企业能够根据自身的特殊情况以及自己的标准流程，将这套管理体系与流程予以制度化这样，企业不仅能够在同类的项目上生到成功的实施，在不同类的项目上一样能够得到成功的实施。科学的管理成为企业的一种文化，企业的组织财富。改进：建立过程可度量的内容，来确定过程的质量和成本；建立过程数据库；收集维护数据并分析；评估软件质量，通知管理层。***管理级***：最大的问题是收集数据的成本。过程数据用于描述软件的开发过程，用于过程改进，而不能用于横向比较项目和个人。企业的项目管理不仅形成了一种制度，而且要实现数字化的管理。对管理流程要做到量化与数字化。通过量化技术来实现流程的稳定性，实现管理的精度，降低项目实施在质量上的波动。改进：支持自动化的收集过程数据；使用数据分析和改进过程。***优化级***：帮助管理者发现哪里需要帮助，怎样提供支持；使专家可以用精准、量化的术语进行交流；提供框架，使专家了解工作表现以及如何改进。企业的项目管理达到了最高的境界。企业不仅能够通过信息手段与数字化手段来实现对项目的管理，而且能够充分利用信息资料，对企业在项目实施的过程中可能出现的次品予以预防。能够主动地改善流程，运用新技术，实现流程的优化。*连续性表现形式*是好坏的问题，由能力水平来表现。(incomplete; performed; managed; defined; quantitatively managed; optimizing) 连续式针对单个过程域；阶段式是官僚教条的，一定程度调和矛盾。**CMMI 的三个应用领域**：CMMI-DEV（开发）、CMMI-SVC（服务）、CMMI-ACQ（采购）**CMMI2 和 CMMI3 关注的目标不同：** 2 级关注项目级别的管理，以每个里程碑的管理为重点，期望项目能够按照计划达成目标。3 级关注于组织的所有项目按照统一的标准过程执行，使用和维护组织过程财富库，以项目的里程碑的内部可见性的管理为重点，期望组织能持续稳定地产出高质量的工作产品。**一套方法已经很熟练但对项目经理来说为什么要改变？（二级向三级改进的原因）** 改进的出发点在于促进共享，这也是必然发生的，然而共享经验和教训需要一个组织级的过程，这样才能使得共享必然发生。**CMMI GG（公共目标）与 GP（公共实践）：L1：** GG1 Achieve Specific Goals 实现特定目标 GP1.1 Perform Specific Practices 实施特定实践；**L2：** GG2 Institutionalize a Managed Process 制度化已管理过程 GP2.1 Establish an Organizational Policy 建立组织政策 GP2.2 Plan the Process 计划过程 GP2.3 Provide Resources 提供资源 GP2.4 Assign Responsibility 分配职责 GP2.5 Train People 培训人员 GP2.6 Control Work Products 控制工作产品 GP2.7 Identify and Involve Relevant Stakeholders 识别并纳入相干关系人 GP2.8 Monitor and Control the Process 监控过程 GP2.9 Objectively Evaluate Adherence 客观评价遵循程度 GP2.10 Review Status with Higher Level Management 与高层管理人员评审状态 **L3：** GG3 Institutionalize a Defined Process 制度化已定义过程 GP3.1 Establish a Defined Process 建立已定义过程 GP3.2 Collect Process Related Experiences 收集过程相关信息

CMMI-DEV 中各过程域按类别与级别的划分：（见右表）

需求管理	REQM	风险管理	RSKM
过程与产品质量保证	PPQA	需求开发	RD
配置管理	CM	技术解决方案	TS
度量与分析	MA	产品集成	PI
项目计划	PP	验证	VER
项目跟踪与监控	PMC	确认	VAL
供应商协议管理	SAM	决策分析与解决方案	DAR
组织过程焦点	OPF	量化项目管理	QPM
组织过程定义	OPD	组织过程绩效	OPP
组织培训	OT	组织绩效管理	OPM
集成项目管理	IPM	原因分析与解决方案	CAR

	Process	Project	Engineering	Support
5	OPM			CAR
4	OPP	QPM		
3	OPD OPF OT	IPM RSKM	RD TS PI VER VAL	DAR
2		PP PMC REQM SAM		CM PPQA MA

其他 CMMI 问题

项目需求管理的目的在于管理项目产品及产品组件的需求，并界定这些需求与项目计划及工作产品间的差异。**软件需求的评审主要关注的方面：** 需求规格说明的正确性；需求规格说明的可实践性；需求规格说明的完整性；需求方案的可行性和成本预算；需求的质量属性；需求的可实施性；需求包含的用例文档；需求评审会的过程和结束标准。参与评审者应包括各级客户、开发人员和测试人员。**PP GP2.2 和现实意义：** 为计划做计划，需要协调人员时间和资源；现实中，项目计划的计划不是临时定义的计划，是固定好的一个模板。规定了会议等的时间和流程，以提前协调人员。**确定软件生命周期模型的意义：** 生命周期模型的定义是确定工作量和成本的基础，不同的生命周期工作量的分布往往是不同的，而且项目历史数据也是在一定的生命周期模型下得到。**项目估计的作用：** 是项目成功的关键，有助于项目的时间管理、资源管理、成本管理和风险管理。**一般估计的因素：** 规模估计、工作量估计、人力资源估计、成本估计。**PMC 跟踪和监控的对象：** 软件工作产品的规模、项目的软件工作量和成本、项目所使用的重要的计算机资源、项目的软件日程、软件工程的技术活动、与项目的费用、资源、日程和技术方面相关的软件风险。**需要配置管理的原因：** 现代软件开发复杂度高；众多的开发人员；文件及相关资源多种多样；多个发布版本；多种平台；软件在不同地点开发。**MA 与 PMC 中数据搜集的区别：** MA 的重点是构建了获取信息的能力，PMC 是获得数据之后进行决策下一步的动作。**软件配置管理（SCM）包括的内容：** 人员职责、权限、配置库结构、备份策略、配置项计划、基线发布计划、培训计划等。**CMMI 的评估方法：** 评估方法叫 SCAMPI，是用于过程改进的标准 CMMI 评估方法；SCAMPI 方法有三种类型：Class A:凡是按体系要求的项目都需要按体系要求做，评估的时候采取抽样评估；Class B:评估试点项目与体系文档、CMMI 模型的符合度；Class C:评估完成的过程体系与 CMMI 模型的差距。**度量分析的作用：** 对项目组：提供了信息，为管理者提供决策信息，告诉管理者项目所处在什么阶段，将来应该进行相应的项目管理决策；对组织：为组织及过程改进，提供决策信息，知道组织过程处于什么阶段，将来的改进方向。**集成就绪检查表的内容：** 集成环境（计算机、编译环境、工具等）已准备好；待集成的部件完成了需求规格说明中的功能和接口；各部件的源代码正确版本在配置库中……有证据证明已经过同行评审；已通过单元测试；稳定版本：配置管理中对应的项应该是关闭状态；必须来自于配置库；必须要有完整的接口描述。

来自博客

成熟度等级背后的动机： ML1->ML2：这个动机是明显的，有管理总比没有管理或者无序管理来得好，尤其是对于有着复杂、不可见、多变、一致性要求等本质挑战的软件开发工作来说，团队需要有方向，有具体实施步骤，有调整，这些往往就是所谓管理的基本要素了；ML2->ML3：动机也是易于理解的。标准化的目的是提供了一种合理成本下的优秀实践、优秀工具、成功经验等等在组织内部共享的机制。一旦有了标准过程定义，上述的这些“优秀”，在组织内部的流传将成为一种必然机制；ML3->ML4：这背后的动机有点耐人寻味。ML4 真好吗？那些量化的基线和数据模型真的可靠吗？恐怕很难有肯定的答案。软件开发是智力工作，人的因素影响太大了；而恰恰是这个因素，历史数据没那么可靠。所有统计方法都必须假设某种规律可以通过历史数据的分析中获得，同时新的场景与历史非常类似。这个在传统行业中很容易实现，但是软件开发很明显不属于这类工作。ML4->ML5：看起来其实是更加缥缈的想法。估计实践者在做缺陷帕累托和根本原因分析的时候，很少能找到通过缺陷类别就能避免或者消除某类缺陷的手段吧？**CMMI 是模型的有关误解：** 所谓 CMMI 模型，是指 CMMI 刻画了软件团队/组织从不成熟到成熟的每个阶段的特征——即所谓的路线图 roadmap。与实际的开发模型没关系。这个路线图其实也是 CMMI 模型最为精华的部分，甚至都可以在很多其他的领域借鉴。*推论之一：CMMI 模型需要适当裁剪以适应公司的实际情况* CMMI 模型不需要裁剪，模型本身仅仅刻画成熟度路线图上不同阶段的特征。大部分公司都不具备能力来裁剪这个模型，真要裁剪，也是应该由 CMMI 的模型的提出方和维护方 SEI 干。真正需要裁剪的是公司内部定义的组织级开发流程和开发规范，这个需要裁剪以适应具体的项目场景，与 CMMI 模型的裁剪是完全不同的概念。*推论之二：CMMI 模型太重了，不适合互联网时代的轻量级开发*这个说法的错误之处在于，不一定是 CMMI 重或者轻，而是，CMMI 根本就不是开发模型。*推论之三：CMMI 模型只适合大公司、大项目，不适合小项目*首先没人检验过；其次，项目的大小衡量本身也缺乏值得信赖的参考依据；最后，接受这种说法的人还是把 CMMI 当成是一种特殊的开发模型。*推论之四：CMMI 模型只适合需求不变或者很少变化的场合，不适合需求不确定、变化很多的场合* CMMI 不是开发模型，与需求变化与否无关，谈不上适应或者不适应。**CMMI 与敏捷的对立：** 这种说法是错的。最根本的原因是 CMMI 不是开发过程，而大部分敏捷则是具体的开发过程。两者根本就是风马牛不相及的事物，不具备冲突的基础。所以不存在两者之间的权衡和借鉴。此外，也不存在 CMMI 的抽象是其不足。所有的模型都是抽象的，抽象恰恰是模型的本质特征之一。模型通过抽象来强化特征与目标之间的关系，这才能帮助我们理解其内在机理，指导具体实践。**那么 CMMI 和敏捷到底有没有差异呢？** 首先，前者不是开发过程，而后者是开发过程，这是最直接最根本的差异。其次，CMMI 存在所谓的标准化，不管是评估方法还是实施办法都有标准化的趋势。而敏捷往往拒绝标准（追求灵活）。再次，作用，即让不熟悉的第三方认可上有差异。尽管 CMMI 目前的现状不乐观，但是，毕竟这种方式提供了一些有价值的线索来了解某个软件组织的能力和成熟度的可能。而这一点敏捷过程还无法提供。有一种说法，CMMI 是主要是组织级过程，而敏捷是项目小组的过程。应该说这种说法有一定的问题，敏捷也可以是组织过程，CMMI 也可以只是关注在小组级别（2 级）。**好方法的特征：** 必须包括具体实践；必须体现整理管理框架；应当具备有 reactive 向 proactive 转变的潜力；必须有人熟悉。**软件项目该度量吗？** 软件项目是否需要度量，完全取决于团队和相关干系人实现项目管理目标的意愿是否足够强。如果想不折不扣的实现项目的目标，那么就必须要度量。所获得数据的意义不仅仅在数据本身，同时体现在对项目相关人员对心理影响上。相反，如果对项目目标不实现与否并不在意，那么我们说，度不度量，并没有太大差异。例如，如果项目交付时间并不是严格限定，只需要在一个大致范围内即可，在这种情况下，进度的度量意义就不大了。**什么是质量管理？ 什么是质量？** IEEE 的定义质量是“与软件产品满足规定的和隐含的需求能力有关的特征或者特性的全体”。Tom Demarco 定义是 “软件产品可以改变世界，使世界更加美好的程度。”非正式场合可以定义质量为多重属性的集合，例如，功能正确、界面友好，响应快、运行稳定，扩展简单，等等。**什么是管理？** 管理至少应该有三方面的内容：清晰的目标定义；正确的跟踪当前状态；产生效果的纠偏措施。在一个完整的管理过程中，这三方面缺一不可。**那什么是质量管理呢？** 事实上，在软件项目当中，一般都是以缺陷管理来替代质量管理的。这样的作法往往都是基于一种基本的质量观，即用户对软件产品的首要期望是功能正确。这个质量属性目标与缺陷密切相关，因而，一定程度上以缺陷管理替代质量管理也是有意义的。

读书笔记

《人月神话》焦油坑： 大型系统的开发表面上没有单独的问题能够导致困难，但是众多小问题纠缠在一起，团队的行动就会非常缓慢，最终举步维艰，淹没在焦油坑中。**人月神话：** 人月：估算技术假设一切都将运作良好；假定人月可互换，将进度和工作量混淆；估算工作不会持续进行；缺少进度跟踪和监督；进度落后时增加人力，沟通和学习成本只会使事情更糟。项目时间分配：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，所有构件已完成。**外科手术队伍：** 项目团队的合理配置。每个人承担对应的职责，每种职责都由对应的人员来负责，从而提高整个团队的效率。然而，这样的队伍只适合小型团队，大型项目团队还需要结合对项目的分解确定分工。**贵族专制、民主政治和系统设计：** 从完整性和可维护性上考虑，“专制”是必要的；但“专制”并不意味着扼杀了其他成员的创造性。其他成员依然可以提出自己的见解，同时在自己的工作领域发挥创造性。**画蛇添足：** 开发第一个系统时，总能发现新的想法或问题，但往往会使得“第二个系统”过于臃肿而难以使用。要解决，需要架构师和项目经理做到“自律”，或者团队中存在开发过“第二个系统”的成员，从而能更加清晰地了解到这个问题。**贵御执行：** 文档化使大规模团队开发的项目具备概念上的完整性，会议还保证信息的透明化。**为什么巴比伦塔会失败：** 复杂项目如果缺乏团队人与人之间的交流，最终一定会崩溃。

《人件》帕金森定律： 很平庸的人作了管理，那么摆在它面前的只有三条路：退位给有能力的人；使用比自己更优秀的属下；运用比自己还平庸的手下。第一条路和第二条路一般是个有欲望的人，都不会采取，那么只有第三条路了。所以，手下的人如此效仿，就演变成整个阶层都是这些平庸的人组成。很多公司都有这种情况，尤其是在政府。**质量的提升会带来成本的降低：** 质量的提升意味着 bug 或缺陷的减少，这样当然就减轻了维护的成本，而软件的长期维护是一个公司最大的成本。**没有快速提高生产力的捷径。工作环境很重要**，在工作环境上省钱很不明智，会大大影响智力劳动者效率。最好是几个人的小房间或者小隔间，减少噪音和干扰。**善待开发者**，让开发者有归属感，降低人员流动性。高人员流动性是很大的成本。**胶冻团队：** 有更高的凝聚力，各担其职，效率更高。胶冻团队的自杀：对手下有防范意识；官僚作风；物理上的分离，阻碍团队交流；分割员工时间；产品质量要求降低；假的截止期限（很容易让人知道不实际，潜意识的加班就成了必须的加速手段，会带来开发者身心上的排斥）；不注意维持团队。**开心地工作**。**会议是为了针对某一问题达成共识**，不是出于这种目的的会议都是一种仪式而已。**未雨绸缪：** 大多数项目开发的初始系统并不合用。要解决问题，必须丢弃，重新开发更灵巧或更好的系统。**整体部分：** 测试和防范 bug 的方法：防范 bug 的定义、测试规格说、自顶向下的设计、结构化编程、单元测试和集成测试等。**祸起萧墙：** 对计划和控制职能进行适度的技术人力投资非常值得赞赏。计划和控制小组作为监督人员，明确地指出了不易察觉的延迟，并强调关键的因素。

《软件管理沉思录》交付高质量的产品： 规模和复杂性不断增加，发生严重问题的可能性也在加大，对软件质量提出了更大挑战。质量是永无终点的旅程。**为高质量项目制订计划：** 最难以制订计划的时候也是最需要计划的时候。**计划必须满足五条基本要求：** 易于理解、清晰明白、详细具体、精确缜密、准确无误。不能使计划准确无误，那就常作计划。**高效团队的基本要素：** 团队致力于共同的目标、团队合作会比个人独立工作表现更出色。**团队遇到的七个常见问题：** 无效的领导、缺乏妥协或合作、缺少参与、拖延和缺乏信息、低劣的质量、功能蔓延、无效的对等评估。**高效团队必备的四个条件：** 团队凝聚力、富有挑战性的目标、反馈、共同的工作架构。**做一位高效的团队成员：** 优秀的团队成员会做任何需要做的事，而不用要求和指示。每个人都献出他所知道的一切，并确保意见被他人听到并理解了。**讨论项目并捍卫你的计划：** 当觉得期限太紧时，要有理有据地说明为什么需要更长的时间，这就是制订计划的作用。与管理者商谈需要注意时宜。**控制你的工作：** 一个明确的过程会帮助你提高。各种小的事件会中断工作，导致一周真正工作的时间很少。项目团队要学会管理压力。文书工作等软件开发工作的附带工作尽量交由支持人员处理。**学会领导：** 领导者的行为、情绪会很大程度上影响团队，影响最终成败。遇到风险和疑虑要与团队一起分析。领导者的行为人共睹，并会被团队成员视为榜样。墨守成规和过分改变都会导致领导力低下。软件开发是创造性的，更需要领导而不是管理。**领导力的两个关键因素：** 身先士卒并相信你的士兵正跟随着你。内在激励或变革性领导能够真正让员工产生热情。**没有银弹：** 软件危机，带来软件研发的变革的神奇武器是不存在的。

《黑客与画家》黑客与画家最像，他们共同之处在于他们都是创造者。 他们本质上都不是在做研究，而是试图创作优秀的作品。与其说计算机是科学，不如说是艺术。要战胜大公司，需要从一个新兴领域进军。包括黑客在内的创作者解决生计的方法是找一份白天工作，然后在其余时间开发优美的软件。优秀的软件要求对美的狂热追求。**不能说的话：** 那些被认为说出了真相的言论容易招致麻烦。思考不该做的事情是很好的脑力训练，同时帮助大脑想出优秀作品。**良好的坏习惯：** 黑客的动机是满足好奇心。不服从管教是黑客成为优秀程序员的原因之一。黑客通过研究当前的技术来构想下一代技术，而版权法设置了障碍，禁止了外部人员通过了解内部细节产生新构想。**另一条路：** 互联网软件更容易、便宜、机动、可靠，通常也比桌面软件更强大。互联网软件可以频繁发布，所以没有版本号的概念。没有明确计划的开发方式适用于小型的由优秀可靠程序员组成的开发团队。按服务付费是互联网软件天然的收费模式。**如何创造财富：** 最好的致富办法就是自己创业或者加入创业公司。创业公司中程序员不受打扰地工作，生产效率大大提高。生产大家需要的东西就是创造财富。财富可以创造出来，并非只能转移。如果一家公司能够按照贡献付薪，它将取得巨大成功。公司越小越容易测量贡献值。让自己的技术难以复制，是对抗大公司最好的办法。尽量早地把公司卖掉。

《精益创业》MVP 在市场和需求高度不确定的情况下，应该用尽可能小的可行性原型产品来进行产品和需求的验证，即 MVP。其中最核心最重要的思想就是提出假设——用最小的代价验证假设。这个过程可以是初创企业的早期，也可以是成熟企业开拓新业务，甚至应用在个人生活中。验证过程中，需要强调的是，用可量化的数据来判断事实，而不是模糊的主观判断，即当下提倡的 data-drive。尤其要注意的是，衡量的指标应该是明确的，和事实相关的，而不是模糊的有多种解释的，或是与具体事实缺乏紧密关联的。测试之后，依据指标事实，确定调整的方向，甚至转型。**精益创业的名称来源于精益生产。** 精益的思考方法大大改变了供应链和生产系统的运作方式。它的原则中包括了吸取每位员工的知识和创造力、把每批次的规模缩小、实时生产和库存管理，以及加快循环周期。精益生产让全世界懂得价值创造活动和浪费之间的差异，揭示了如何由内而外地将质量融入到产品之中。**创业者无处不在。** 所谓的新创企业就是在充满不确定性的情况下，以开发新产品和新服务为目的而设立的个人机构。…… 精益创业的方法可以运用到各行各业，任何规模的公司。**创业即管理。** 新创企业不仅代表了一种产品，更是一种机构制度，所以它需要某种新的管理方式，特别是要能应对极端不稳定的情况。**经证实的认知。** 创业者可以通过频繁的实验检测其愿景的各个方面，这种认知是可以得到验证的。**开发——测量——认知。** 所有成功的新创企业的流程步骤都应该以加速这个反馈循环为宗旨。**创新核算。** 为了提高创业成果，并让创新者们负起相应责任，我们需要关注那些乏味的细枝末节：如何衡量进度，如何确定阶段性目标，以及如何优先分配工作。这需要为新创企业设计一套新的核算制度，让每个人都肩负职责。

敏捷的误区

敏捷项目没有计划： 由于产品需求的不确定性、甚至是未知的，敏捷项目团队很少能在项目之初建立一份类似于 WBS 任务分解的进度表和甘特图，但敏捷项目依然是有计划的，和传统的进度计划不同，敏捷的计划不是关注在完成项目的一个个活动或者说任务，比如说需求分析、概要设计、详细设计，模块一编码等等，而是关注在客户的需要，关注客户价值的优先级，其计划的对象是用户要求的功能，例如用户故事，计划活动的产出是一个设置了优先级的用户需要的功能列表。敏捷计划分为以下几个层次：愿景–制定产品的长远目标；路线图–制定实现长远目标的分步实施计划；发布–制定一次发布的目标；迭代–制定一次迭代的目标；每日计划–制定每天的工作目标。其计划的过程是一个持续的过程，从项目开始时制定产品的愿景，到每个迭代开始时制定迭代计划，敏捷项目的计划不断的细化，不断的根据变化而调整，是 Just-In-Time 的计划。**敏捷就是追求速度：** 敏捷实际是关注实现客户的价值，而这一价值体现在“可工作的软件”之中，这其实是对质量的要求，它意味着交付的软件是客户需要的并且质量稳定的，是同时对需求质量和开发质量提出要求。另外，因为市场的变化会促使客户重新调整需求，以获取最大的价值，因此敏捷强调“响应变化”，迅速调整策略，以帮助客户迅速对市场变化做出响应。**敏捷是放之四海而皆准的开发模式：** 敏捷开发模式被互联网企业广泛采用的重要的原因有两个：产品的功能升级要新换代非常快，大家都必须要在最短的时间内抢占市场，吸引用户，而需求往往又不是非常的明确，甚至有时只是一个 idea，需要市场的反馈；产品的升级是可控的，即便是带着一定缺陷的产品发布（又称为“灰度发布”），我们都可以在后台悄悄的升级系统或修改 BUG，对于用户来说，任何时间打开浏览器都可以看到最新的产品，因此对用户的影响是最小的，甚至用户是不感知的。对于那些需要安装到用户使用的终端（电脑、手机、平板等）的应用来说，这样的升级方式可能就会导致客户的反感、投诉和客户流失。**敏捷是一个过程：** 敏捷不是一个过程，是一类过程的统称，它们有一个共性，就是符合敏捷价值观，遵循敏捷的原则。敏捷的价值观如下：个体和交互 胜过 过程和工具；可以工作的软件 胜过 面面俱到的文档；客户合作 胜过 合同谈判；响应变化 胜过 遵循计划。建立敏捷联盟的 17 位大师所创立的敏捷方法包括：极限编程，Scrum，特征驱动开发，动态系统开发方法，自适应软件开发，水晶方法，实用编程方法。这些方法统称为敏捷方法。 每个人都可以从敏捷宣言和原则出发，明确问题，找出一些解决方法，形成自己的过程。要实用而不是要机械式的规范。让开发人员理解并实施，体验到敏捷的好处，而不是盲目机械地实施规范。CMMI 的最高境界（Level 5）也是要求组织有足够的灵活性适应外界环境的变化，灵活的运用规范，而不是教条主义。**敏捷只适用于小型项目和团队：** 敏捷实践确实很多发源于小型的项目团队，但并不是说敏捷只适合小型项目团队。其实，早期的 Scrum 项目就已经有在 500 多人的大型项目中成功实施的案例。可能是由于大多数的敏捷团队一般都希望在 5～9 人的规模，并且希望团队成员在同一个工作区域，所以很多时候被认为不适合跨地域，跨团队的大型项目的开发。其实，5～9 人的团队规模是一个类似于一个战斗小组的规模，这个团队小组负责完成一个共同的目标。对于一个小型的项目而言，可能只需要一个这样的战斗小组就可以了，而对于一个大型的项目，我们就可以建立多个这样的战斗小组来完成项目目标。在 Scrum 中，就有 Scrum Scaling，通过把一个大型项目团队合理分解为多个小型的 Scrum 团队，每个团队都负责一个相对独立的模块或者功能，再配合其他的敏捷实践，比如持续集成，Scrum of Scrums 等，加强团队之间的协作，从而确保项目的成功。所以，将敏捷实践应用于大型的、复杂的项目是完全可以的。**敏捷开发 == 快速开始编码**：敏捷开发强调迭代，鼓励开发人员用代码说话，不过绝对不鼓励没想明白就写代码。符合敏捷开发思想的流程往往主张在一个稳定的基础之上迭代完成各种功能。如果基础都不牢固，迭代就无法进行，整个开发过程就退化成不断重写的过程，浪费开发时间。敏捷开发实际非常重视“设计”，并且对开发人员的设计水平提出了极高的要求，既要“持续重构”又不能“过度设计”，稍有不慎就会陷入反复推翻已有代码的窘境。对于内功不够的开发人员最好还是想好再写代码，设计的时候慢一点没关系，尽量少的做无用功才是最重要。**敏捷是反文档的**： 文档只是为了达成目标的一种手段，如果这种手段是低效的，那就换一种手段。可是完全抛弃了文档，怎样解决沟通的问题？难道你想每次沟通都完全用手比划，用嘴说，跟不同的人重复表述同样的想法，那样更是低效的。 应该清楚文档的本质是把知识显性化。在一个项目中存在很多需要沟通的知识，知识具备两种形态，显性的和隐性的，传统的观念是尽量把隐性知识显性化，即文档化，而忽略了这其中的代价（特别是更新同步文档的代价）。 因此，在实施敏捷的时候，需要在团队内明确哪些知识是必须显性的，这些知识可以通过文档交流。哪些知识是可以隐性的，这些知识则完全可以通过口头的方式进行交流，以达到沟通的最佳效率。 文档不是目的，有效沟通才是目的。就工作量而言，不写文档，减少写文档时间，看起来确实可以加快开发速度，但实际会严重伤害项目。互联网应用开发不需要似传统大型软件开发那样，按照软件工程，花大量时间去写文档。不要为了写文档而写文档，而是为了开发而写文档。不要让文档成为开发的负担，而是成为帮助。现在工作的流动性很大，没有文档，当中一两个开发人员离职，如何后续处理。没有开发文档，在架构和关键技术上没有想清楚，贸然而上，开发的永远只是个 demo 产品。没有任何文档，极容易导致推出后，疲于奔命地修修补补，每天一个版本。敏捷开发是快，但是快不等于敏捷开发。**敏捷是自由的，无约束的：** 敏捷强调的是自组织团队，发挥人的能动性，以动力代替压力，让人有绝对自由的错觉。但是应该清楚，凡事都是要讲究一个平衡，人也是两面的，消极的一面和积极的一面同时并存，绝对的自由会放纵人得懒惰和自由。敏捷并非是绝对自由，无约束的。作为管理者，有一个职责，就是引导团队成员用自己积极的一面去压制消极的一面，不能放任团队中出现搭便车的现象，否则将打击整个团队的士气。如果实在无效，那就只能将其排除出团队了，这个惩罚够有约束力吧？

其他问题

两个关于开源的谬论： 有一个普遍认同但不准确的观点：任何对开源软件的修改或自定义都必须公开发布。另一个经常被引用的谬论是：开源不能与专有软件一起使用。支持这种观点的人认为，免费软件条款应包含与闭源代码许可条款不兼容的规定。**测试：** 25. 当前最好的软件测试也只有 55%到 60%的覆盖率。26. 即是 100%的测试覆率仍然是远远不够的。27. 测试工是必备的，然而甚少被用。28. 自动化测试被夸大了，多数测试不能被自动化。29. 由程序员编写的，嵌入在代码里的调试代码是对测试 I 具的很好补充。**描述你所理解的互联网开发模式和互联网商业模式请具体谈谈 google 和百度的商业模式：** 百度竞价排名服务百度竞价排名是百度首创的一种按效果付费的网络推广方式，用少量的投入就可以给企业带来大量潜在客户，有效提升企业销售额。搜索+呼叫广告（呼叫广告按通话效果付费，即有了意向客户进行才收费，如果与搜索引擎的竞价排名进行一定的融合，更能够给客户带来一定的效果，这样也会使得百度摆脱搜索竞价欺骗的骂名）。谷歌商业模式：Google 广告解决方案针对广告客户：Google AdWords 仅当有人点击您的广告时，您才需要支付费用。针对网站发布商：Google AdSenseGoogle AdSense 是一个快速简便的方法，可以让各种规模的网站发布商为他们的网站展示与网站内容相关的 Google 广告并获取收入。由于所展示的广告与用户在您的网站上浏览的内容相关，或与您网站内容所吸引的用户的个性和兴致相符，您终于可以在充实网页的同时，透过网页为你带来经济效益。Google 的纯云商业模式。开源名义下的商业新探索（推广 Android，其目的是什么呢？目的其实就是“组织管理遍布全球的信息资源” Google 的服务都是基于互联网的，所以，Google 最终要实现这个计划，必须得让上网变得越来越方便，让人们随时随地连接上网络，无论是有线还是无线。毫无疑问，无线领域必将成为下一个爆发的信息通道，而把持住这个通道则将成为 Google 的最重要战略。Google 手机更重大的一个意义在于，手机的发展从之前硬件的开发过渡到了软件的开发，都是建立在 Android 这个平台基础之上的，必须符合 Android 的标准，这样，Google 就可以顺理成章地变成行业标准的制定者。其实，Google 是在开源的名义下，正在进行一场新商业模式的探索。史特斯说道，这种商业模式是独一无二的。毋庸置疑，搜索广告只是其中一种形式。）