CMM/CMMI 的优缺点: 优点该模型提供了大量的以行业最佳实践为基础的背景信息和指导信息,如果使用得当,这些信息可能会很有价值.过程域的层次划分便于以渐进方式进行改进,这会更为有效和持久.过程域的划分使过程改进活动中心更为突出,也更容易管理.有限的范围可以提供更精深的细节,并为与项目有关的过程提供指导.CMMI 目标和共性实践的描述方式使其可以适用于很多类组织和项目.CMMI 提供了成熟度级别(阶段式表示法)和能力级别(连续式表示法)作为确定和评价在过程改进上所取得的进步的方法.CMMI 对量度特别重视,这有助于确定过程改进活动的投资回报.**缺点**
CMM/CMMI 的评估耗资不菲,一个 CMM2 级评估就可能达到数百万之巨,而且耗时很长,过程十分复杂,常常导致效果不太理想.不少企业的认证流于形式,评估完成后就只留下一大堆文档,而真正的软件开发过程却依然故我.而且,CMM/CMMI 只告诉我们应该怎么做,而没有具体地告诉如何做.
CMMI **1.一个模型**:这不是开发模型,这是过程改进的参考模型,CMMI 可以应用在非软件领域.把最佳实践分组排序的意义:刻画了从不成熟到成熟的路线;帮助一个企业定位,理解所处的位置;帮助企业指引改进的方向.过程域:做好软件开发的某一个方面,即把最佳实践分组排序的结果.**2.两种表现形式**:**阶段式**和**连续式** 能力程度:做得怎么样（连续式）;成熟程度:做了还是没做（阶段式）**3.三种不同的应用领域/业务形式**:（1）软件开发领域的 CMMI 模型:CMMI-DEV （CMMI for development）有 22 个过程域（2）服务领域的 CMMI 模型:CMMI-SVC（3）为采购提供的 CMMI 模型:CMMI-ACQ.*SAAS:Software-as-a-Service 软件发布部署 **4.把最佳实践分成 4 个类别**（1）项目(project)管理类型（2）过程(process)管理类型（3）工程管理类 engineering （4）支持管理类型 support. **任何一个过程域一定唯一的属于某一个类别.5.4 个级别** 等级一定是阶段式,对应成熟度 本来有 5 级,但因为第一级没有过程域 把 CMMI-DEV 的 **22** 个过程域分类:

	project	process	engineer	support	
5		OID(OPM)		CAR	一般
4	QPM	OPP			特殊
3	IPM/RSKM	OPD/OPF/OT	RD/TS/PI/VER/VAL	DAR	proactive
2	PP/PMC/SAM/REQM			CM/MA/PPQA	reactive

所有 O 开头的都是过程类的
5 个等级 CMMI 阶段式表现形式的有 5 个成熟级别:1. initial 2.managed(CMM 是 repeated) 3.defined 4.managed(CMM 是 quantitatively managed) 5.optimizing **连续式**表现形式单个过程域有 **6 个能力级别**:0 级 incomplete(因为它不完整),1 级 performed(指某个特定的过程域里的工作你全都做了) **第二级**管理的三个要素:（1）非常明确清晰的目标定义(目标)（2）状态跟踪(状态)（3）纠正偏差的措施(纠偏) **一级到二级**:有管理总比没有好. **OSSP**:组织级标准软件过程 **PDSP**:企业级 **二级到三级**（OSSP->PDSP）是为了共享,让开发经验的共享变成必然.**第四级高级在哪里？(三级到四级** 预测能力加强,定性到定量) 根据偏差进行纠正背后反映了一个模型,就是根据当前状态进行预测的模型；这个模型的依据是历史数据.让这个模型更可靠的方法:增加数据量；分类；消除异常数据.**第五级** 消除影响过程的根本因素. 过程域的成熟(maturity)级别达到 1 级,那么能力(capability)级别必须达到 1 级：…2,…2;…3,…3;…4,…3;…5,…3；**过程域**:一组相关实践的集合,为了满足一个或多个目标 required:必须要实现的 expected:期望实现的 informative:补充的信息.**GP**（generic practices）gp2.1 建立一个指导原则（policy）,在实施过程域的时候指导思想是什么 gp2.2 对过程域的工作做一个规划,怎么做这个过程域 gp2.3 为了确保工作的完成投入足够的资源,人力、环境 gp2.4 有清晰的角色分配和定义 gp2.5 为了让大家做这个过程域要进行一些培训 gp2.6 这个过程域有一些工作产品的产生,如何保存 gp2.7 相关干系人怎么协作 gp2.8 按规划去检查,是否有偏差 gp2.9 客观的第三方的检查 gp2.10 高级管理者怎么参与的 gp3.1 建立一个组织级的规范 gp3.2 保存结果,收集过程改进的资料
二级过程域 1.PP(Project Planning).定义:建立和维护定义项目活动的计划.SG（specific goal）:建立估算;开发一个项目计划;获取计划的认可.软件项目估算,最重要的不是估得准不准,而是估算出来的结果你的管理者、团队等是否接受这个数字；如果是大家都信任的估算,那就是好的估算.**问题:为什么 pp 也有 gp2.2**？计划的计划.为计划做一个计划,现实中有这个必要,因为项目计划涉及到多个工作人员的参与,所以需要一个计划来计划它.**2.PMC(Project Monitoring and Control)**项目监督与控制.状态跟踪加纠偏 SG:根据计划来监督项目；对纠正偏差的措施进行跟踪管理.**里程碑评审 vs 进度评审**:两者看的东西很接近,都得看风险等；前者回答的是这个项目是不是需要继续下去、是否进入下一阶段--客户跟高级经理一起参与；后者回答的是判断项目是不是按照计划在进展--范围小、频率高；团队内部开展,不需要管理者、客户参与 **问题:为什么 pmc 有 gp2.8？** 监控的监控.小组的周例会没有很好的开展；pmc 的 gp2.8 在里程碑评审处做 **3.CM(Configuration Management)**配置管理.建立和维护工作产品的完整性,使用配置识别、配置控制、配置状态报告、配置审计.SG:建立基线；跟踪和控制变更；建立完整性.识别配置项是因为软件开发过程中产生的对象太多,所以要减少要管理的对象.配置管理的关键在于变更请求、对所有的变更进行跟踪控制,而不在于版本控制.版本控制可以完全自动化,配置管理不可能完全自动化.**问题:CM 的 gp2.6**？配置的配置 因为配置也有很多东西需要保存,比如配置项的变更请求、基线的发布记录等 **4.MA(**Measurement and Analysis)度量和分析.SG:把度量和管理目标联系起来；提供度量结果.GQM-goal question metric.**问题:MA 度量和分析中的数据收集和 PMC 中的数据采集分析的差别**是什么？MA 的重点构建了一种获取信息的能力,PMC 是有了获取数据能力之后根据这些数据来决定下一步动作.base measure(基本) VS derived measure(衍生) 缺陷个数:base；严重缺陷个数:base；严重缺陷百分比:derived；缺陷密度:derived；人月:base **问题:**在满足信息需求的基础上,设计一个度量体系时 base 和 derived 哪个多一点？衍生度量多一点,因为收集 base 工作量很大 **5.PPQA**(Process and Product Quality Assurance)过程和产品质量保证 **问题:**客观的评价过程和客观的评价工作产品是期望型的（不是必需型的）哪一个是可以被替换的？客观的评价过程更重要.因为过程的质量决定了最终产品的质量,对过程的评价是通过对产品的评价来进行的,CMMI 的思想是全面管理质量.**问题:**PPQA 的 gp2.9？QA 的 QA QA 活动是有一套标准的,QA 检查表,那么 QA 的执行是否按照定义的流程走是需要检查的,即 QA 的 QA
3. 三级过程域 1.RD(Requirements Development)需求开发.SG:开发客户需求:客户提问题；开发产品需求:开发团队给答案.分析和确认需求.步骤:提问题、给答案、证明答案是对的 2.**技术方案** TS(technical solution) 方案选型、设计、实现 **3.PI** 产品集成 持续集成和集成测试属于 PI,但是反过来就不行,因为 PI 一直管到交付,类似单元测试、集成测试、性能测试都可以理解为 PI 的工作.做产品集成的时候是有策略/顺序的.哪些因素影响顺序？:开发计划、完成模块的质量.sequence 和项目完全相关,procedure 和 criteria 可能在大部分项目中差不多.**问题:待集成的组件可以做集成的标准**有哪些？-必须是稳定版本（配置项的状态是关闭状态、组件来自于配置库而不是工作库）-必须有完整的接口描述-必须有证据证明通过了单元测试 **4.VAL(Validation)**确认 目的:产品组件在即将使用它的环境中是能够满足使用意图的.最终的产品有没有帮客户解决问题（客户需求）VAL 高于 VER
5.VER(Verification)验证 目的:确保被选择的工作产品满足需求.有没有正确的把解决方案给实现(产品需求) **6.OPD** 组织级过程定义 建立了一套标准的流程 **7.IPM** 集成项目管理（6、7 是一对）小组按照标准流程指定小组的执行流程 OSSP-OPD PDSP-IPM **8.OPF** 组织级过程焦点 对 IPM 执行结果进行评价审查,识别强项弱项,进行改进 **9.OPM** 组织绩效管理（原来是 OID 组织级创新与部署）增强版本的 OPF,是量化的去识别强项弱项（measurely improve XXX）**10.OPP** 组织过级程性能 需求 需求评审 设计 设计评审 实现 测试 6 个阶段每个阶段都会注入/解决错误 **问题 1:**验收测试的时候有 5 个错误,现在重做项目,但是不做需求评审（当时需求评审消除了 10 个错误）,最后验收的时候有几个错误？答:可能大于、等于、小于 15 个 **问题 2:**非要估算最后的错误个数,需要哪些信息？答:每个阶段注入的错误个数、每个阶段测试消除的错误个数 baseline:基于子过程的历史数据,计算标准差和均值 model:需求注入-需求评审消除+设计注入-设计评审消除+实现注入-测试消除 **11.QPM** 定量项目管理 OPP 给出管理目标 项目管理管的是子过程,确保每个子过程的能力基线能够实现.

SEMIntro:导论:
1、软件工程定义:1. 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；2. 在 1 中所述方法的研究.
3、软件工程知识域:软件需求、软件设计、软件构造、软件测试、软件维护、软件配置管理、软件工程管理、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、计算基础、数学基础、工程基础
3、软件工程 VS 计算机科学:目标（**SE:**在资源的限制条件下构建满足用户需求的软件系统 **CS:**探索计算和建模方法,改进计算方法）、关注（**SE:**如何为用户实现价值 **CS:**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）**4、软件工程 VS 计算机程序设计:**1.软件工程存在于各种应用中,存在于软件开发的各个方面.而程序设计通常包含了程序设计和编码的反复迭代的过程,它是软件开发的一个阶段；2.软件工程力图对软件项目的各个方面做出指导,从软件的可行性分析直到软件完成以后的维护工作.

Ch1 没有银弹
1、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）**2、根本任务**:打造由抽象软件实体构成的复杂概念结构 **3、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.除非次要任务占了所有工作的 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.（硬件的发展使次要任务越来越容易解决）**4、软件项目的现状**: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）**6、探寻软件产业发展的问题:**按照亚里士多德的观点,将软件开发的问题分成根本的(essence)—软件特性中固有的困难,次要的(accident)—出现在目前生产上的,但并非那些与生俱来的困难.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.**7、软件系统中无法规避的内在特性**:复杂度、不一致性、可变性和不可见性.银弹与软件的内在特性相悖.Brooks 认为软件开发中困难的部分是规格化、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,那么软件开发总是非常困难的,天生就没有银弹.**1.复杂度:**a. 软件实体可能比任何由人类创造的其他实体都要复杂 b.计算机复杂,状态多,构思、描述、测试困难 c.软件实体扩展是不同元素实体的添加,复杂度非线性增长.d.软件的复杂度是必要属性,不是次要因素.e.导致了产品瑕疵、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,影响了产品的可靠性；函数调用变得困难,导致程序难以使用；程序难以在不产生副作用的情况下用新函数扩充；造成很多安全机制状态上的不可见性 f.复杂度引发管理上的问题:全面理解问题变得困难,从而妨碍了概念上的完整性；所有离散出口难以寻找和控制；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）**2.不一致性:**软件开发面对的复杂度往往是随心所欲、毫无规则可言的,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.**3.可变性:**软件实体经常会遭受到持续的变更压力.软件很容易修改（它是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.**4.不可见性:**软件是不可见的和无法可视化的；软件的客观存在不具有空间的形体特征.限制了个人的设计过程,也严重的阻碍了相互之间的交流（UML）.**8、没有银弹主要观点:**相对必要任务而言,除非次要任务占了所有工作的 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.
9、当年的银弹:Ada 和其他高级编程语言、面向对象编程（消除了开发过程中的非本质困难,允许设计人员表达自己设计的内在特性,而不需要表达大量句法上的内容；对于抽象数据类型和层次化类型,它们都是解决了高级别的次要困难和允许采用较高层次的表现形式来表达设计；使得修改局部化,提高了可维护性）、**人工智能**（AI-1:使用计算机来解决以前只能通过人类智慧解决的问题.AI-2:使用启发式和基于规则的特定编程技术.程序被设计成以人类解决问题的方式来运作）、**专家系统**（专家系统是包含归纳推论引擎和规则基础的程序,它接收输入数据和假设条件,通过从基础规则推导逻辑结果,提出结论和建议,向用户展示前因后果,并解释最终的结果.专家系统最强有力的贡献是给缺乏经验的开发人员提供服务,用最优秀开发者的经验和知识积累为他们提供了指导.）、**“自动”编程**（从问题的一段陈述说明自动产生解决问题的程序.大多数情况下所给出的技术说明本质上是问题的解决方法,而不是问题自身）、**图形化编程**（软件非常难以可视化）、**程序验证**（是否能够在系统设计级别、源代码级别消除 bug 呢？是否可以在大量工作被投入到实现和测试之前,通过采用证实设计正确性的“深奥”策略,彻底提高软件的生产率和产品的可靠性？不能保证节约劳动力；程序验证不意味着零缺陷的程序；完美的程序验证只能建立满足技术说明的程序,而这时,软件工作中最困难的部分已经接近完成,形成了完整和一致的说明）、**环境和工具**（这样的工作是非常有价值的,它能带来软件生产率和可靠性上的一些提高.但是,由于它自身的特性,目前它的回报很有限）、**购买和自行开发**（构建软件最可能的彻底解决方案是不开发任何软件；通用软件）、**需求精炼和快速原型**（概念性工作中,没有其他任何一个部分比确定详细的技术需求更加困难；事实上,客户不知道自己需要什么；快速原型明确实际的概念结构,让用户知道他们需要什么）、**增量开发——增长而非搭建系统**（Grow not building；客户；士气；迭代式开发）、**卓越的设计人员**（软件开发是一个创造性的过程）
10、没有银弹的影响:软件开发本质的认识；软件过程

Ch2 大教堂和市集
《大教堂和市集》以大教堂模式和开放市集模式的比喻将自由软件和商业封闭软件区分开来——“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。” 他在文中论证了自由软件不仅仅是一种乌托邦的理想,而是在开发模式上真正代表着“先进生产力”,代表着历史发展趋势的必然.
1、Fetchmail: Linux 开发模式:1.每一个好的软件的起因都是挠到了开发者本人的痒处（“需要是发明之母”长久以来就被证明是正确的）**2.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**（伟大程序员的一个重要特点是建设性的懒惰.他们知道你是因为成绩而不是努力得到奖赏,而且从一个好的实际的解决方案开始总是要比从头干起容易；）**3.计划好抛弃**（你常常在第一次实现一个解决方案之后才能理解问题所在,第二次你也许才足够清楚怎样做好它,因此如果你想做好,准备好推翻重来一次）**4. 如果你有正确的态度,有趣的问题会找上你**（你在思考、审视一些你感兴趣的软件时,你会有新的更好的想法） **5.当你对一个程序失去兴趣时,你最后的责任就是把它传给一个能干的后继者**（在开源软件的开发中）**6.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 7. 早发布、常发布、听取客户的建议**（多数开发人过去都相信这对大型工程来说是个不好的策略,因为早期版本都是些充满错误的版本,而你不想耗光用户的耐心；这种信仰强化了建造大教堂开发方式的必要性；Linus 的创新并不是这个(这在 Unix 世界中是一个长期传统),而是把它扩展到和他所开发的东西的复杂程度相匹配的地步,而且因为他培育了他的协作开发者基础,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） **8. Linus 定律: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**（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,和在长期等待之后并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,或者至少当暴露给上千个热切的协作开发人员,让他们来对每个新发布进行测试的时候,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；**Delphi 效应:**一群相同专业（或相同无知的）观察者的平均观点比在其中随机挑选一个来得更加可靠.**两种版本:**Linus 也做了一些改进,如果有一些严重的错误,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,这个战略还没被大多数 Linux 黑客所仿效,但它应该被仿效,存在两个选择的事实让二者都很吸引人）**9. 聪明的数据结构和笨拙的代码要比相反的搭配工作的更好 10. 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. 11. 想出好主意是好事,从你的用户那里发现好主意也是好事,有时候后者更好. 12 最有突破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. 13.“最好的设计不是再没有什么东西可以添加,而是再也没有什么东西可以去掉.” 14. 任何工具都应该达到预期的用处,但是一个伟大的工具提供你没料到的功能 15.当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作! 16. 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 17. 一个安全系统只能和它的秘密一样安全,当心伪安全 18.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,它们就推广开来了）**19.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高效建造来扩展）(11-16 来自 fetchmail)
4、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.但**如果 Brooks 定律就是全部,那 Linux 就不可能成功**（交流与沟通）.
5、“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.虽然编码是一个人活,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和进行修改的环境的开发人员之后.

6、集市风格的必要的先决条件——1.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.（linux 也不是从头开始的）2.当你开始创建社团时,你需要演示的是一个诺言,你的程序不需要工作的很好,它可以很粗糙、很笨拙、不完整和缺少文档、它不能忽略的东西是要吸引大家卷入一个有趣的项目.市集项目协调人或领导人必须有良好的人际和交流能力 3.能够提出卓越的原始设计思想对协调人来说不是最关键的,但是对他／她来说绝对关键的是要能把从他人那里得到的好的设计重新组织起来 4.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 5.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,但这远远不是全部,个人素质也有关系.

8、Linux 与 Internet:1.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,Linus 是第一个学会怎样利用 Internet 的新规的人.2 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 3. Linux 黑客取得的最大化的“实际利益”不是经典的经济利益,而是无形的他们的自我满足和在其他黑客中的声望 **9、怎么解释开源与 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 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；代码变动和臭虫报告都流经项目的核心,只有在小小的核心团体中全面的布洛克成本才有效.

10、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(假设你相信后者,Linus 和我都不),而仅仅是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.

Ch3 开源软件经济学

很多高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.他们并不是放弃资本主义,而是认为这是个好的商业策略.**1、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就会增加（当计算机变便宜就会有更多人买,由于计算机要有操作系统,于是操作系统的需求就增加,而操作系统的盈利也水涨船高）**2、为什么公司要支持开源？**很多有责任尽量提升股东价值的大型上市公司,投入很多资金支持开放源码软件(通常是负担大型程序团队的开发费用),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. **6、**软件是很容易让硬件普及化的（只要个小小的硬件抽象层,比如像 Windows NT 的 HAL 就只要很小一段程序）,不过硬件想让软件普及化确是难上加难.

Ch4 颠覆式创新

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（sustaining）:根据主要市场的主流客户一直以来看中的性能层面,来提高成熟产品的性能.这些技术可能是突破式的,也可能是渐进式的.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,很少会导致企业失败.**破坏性技术 disruptive:**性能低于主流市场的成熟产品,但拥有边缘客户看中的新特性；基于破坏性技术的产品价格更低、体积更小、性能更简单,而且通常更便于客户使用；最初应用于非主流市场或新的市场,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都是破坏性技术导致的.小的技术改进也可能带来破坏性创新;突破性技术不一定会带来破坏性创新.

管理良好的企业失败的原因:1.企业的资源分布取决于客户和投资者.关键概念:价值网络.一种大环境,包括:特定的产品性能属性；特定的成本结构（盈利模式）；特定的组织能力.价值网络决定了企业内部的资源分配,成熟企业以客户为导向的资源分配和决策流程,决定了它们无法实现转型.2.小市场并不能解决大企业的增长需求 3.无法对并不存在的市场进行分析 4.大型机构能力无法应对破坏性创新,一个机构的能力包括资源、流程、价值观,它独立于机构内部工作人员的能力.**成熟企业的流程和价值观通常都无法进行破坏性创新**.5.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,但是破坏性技术将在日后变得极具竞争力.

应对颠覆式创新:1.把开发破坏性技术的职责赋予存在客户需求的机构.说服企业的每一个人,这对企业的发展有长期战略意义.创建一个独立的机构,使这个机构直接面对确实需要这种技术的新兴客户群体 2.设立一个能够欣然接受较小收益的独立小型机构 3.为失败做好准备

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.可以用鸿沟理论来解释先行者的领先优势.**其他应对破坏性创新时需要考虑的问题:**1.怎样判断出某技术是否具有破坏性？她的性能曲线日后可能与主流市场需求相交汇.2.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 3.应该采取什么样的产品技术和经销策略？新产品理念: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；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:将颠覆性创新与改变行业竞争格局的任何突破混为一谈不可取,因为不同类型的创新需要采取不同的战略.“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.

颠覆式创新容易被忽视或误解的部分:颠覆式创新是一个过程；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,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.相反的,他们应该通过投资持续创新,继续加固与核心客户群的纽带.此外,他们可以创建一个全新的部门来全心全意应对由颠覆现象带来的机遇.

《人月神话》1、进度总成问题:缺乏合理的时间进度是造成项目滞后的最主要原因 1 对估算技术缺乏有效的研究；2 采用的估算技术隐含地假设人月可以互换,错误地将进度与工作量相互混淆；3 由于对自己的估算缺乏信心,软件经理通常不会有耐心持续地进行估算这项工作；4 对进度缺少跟踪和监督.5 当意识到进度的偏移时,下意识（以及传统）的反应是增加人力.这只会使事情更糟,从而进入了一场注定会导致灾难的循环.（合理的做法是不停更新估算）**2、乐观主义:**一切都将运作良好,每一项任务仅花费它所“应该”花费的时间.**3、“人月的不合理性”**:1.成本的确随开发产品的人数和时间的不同,有着很大的变化,进度却不是如此.用人月作为衡量一项工作的规模暗示着人员数量和时间是可以相互替换的.2.人数和时间的互换仅仅适用于以下情况:某个任务可以分解给参与人员,并且他们之间不需要相互的交流.这在系统编程中近乎不可能.3.当任务由于次序上的限制不能分解时,人手的添加对进度没有帮助.4.对于可以分解,但子任务之间需要相互沟通和交流的任务,必须在计划工作中考虑沟通的工作量.5.沟通所增加的负担由两个部分组成,培训和相互的交流.每个成员需要进行技术、项目目标以及总体策略上的培训.这种培训不能分解.6.相互之间交流的情况更糟一些.7.一对一交流的情况下,所增加的用于沟通的工作量可能会完全抵消对原有任务分解所产生的作用.（此外新人加进来要重新分工,原来的一些半成品也要丢掉）**4、系统测试:**由于乐观主义,通常实际出现的缺陷数量比预料的要多得多.不为系统测试安排足够的时间简直就是一场灾难：直到项目的发布日期,才有人发现进度上的问题；此时此刻的延迟具有不寻常的、严重的财务和心理上的反应.

5、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

6、空泛的估算:还没有可靠的估算技术出现；在基于可靠基础的估算出现之前,项目经理需要挺直腰杆,坚持他们的估计,确信自己的经验和直觉总比从期望派生出的结果要强得多.

7、一个软件项目落后于进度:1.重新安排进度 2.削减任务.在现实情况中,一旦开发团队观察到进度的偏差,总是倾向于对任务进行削减.当项目延期所导致的后续成本非常高时,这常常是唯一可行的方法

《人件》伟大的管理者懂得人本质上是不可管理的,软件成功的本质是使每个人朝着同一个方向努力并且使他们热情高涨,任何事物都不能阻止他们前进.

管理人力资源 由于我们以团队、项目或者其他紧密协作工作小组的形式来完成工作,我们大多数人是在从事人类交流的职业.我们的成功源自于所有参与者良好的人与人之间的互动,我们的失败则归因于这种互动的缺失.项目需要的投入越夸张,成员就越应该学习如何更好地协作,对这份工作的热爱也会变得更重要.项目越是需要在一个无法完成的固定时间交付,项目团队就越不能缺乏频繁的头脑风暴,或者项目组聚餐之类的活动来帮助团队形成一个统一的整体.压力不会让人工作得更好—只是工作得更快.质量,远远不只是最终用户的要求,而是达到高产能的一种方法.一个组织,如果为了质量一毛不拔,那么收获的质量也将会一文不值.“质量一如果时间允许”这种策略会导致产品不会有任何质量可言.管理者的作用不是让大家去工作,而是创造环境,让大家可以顺利开展工作.

创建健康的办公环境 只要员工还拥挤在嘈杂、低效、干扰不断的环境里,任何除了环境外的改进都是徒劳的.度量体系往往会变成对大家的威胁,反而增加大家的负担.为了让度量这个理念发挥应有的潜力,管理层必须足够主动安全地把自己从这个圈子里摘出去.这就意味着个人的数据不会被传递到管理层手中,而且组织中每个人都心知肚明,收集的个人表现数据只能用于个人提升.度量体系就是自评估的一个练习,只有处理过的平均值才会交给老板看.创造一个合理的工作环境,成败与否可以由一些公认的标志来决定.一个显而易见的成功标志就是门.如果环境有足够多的门,员工就可以有选择地控制噪声和干扰来满足工作的变化. **雇佣并留用正确的人** 创新依靠领导力,而领导力又需要创新.二者互为依存,若一项能力不足,就会影响另一项能力.即使最好的创新也需要一点离经叛道才能产生影响:离经叛道的领导力.创新者自己可以不是一个伟大的领导者,但必须有人是.在这个创新过程中,这种离经叛道的领导力带来的是时间-让一个关键人物不去做产生利润的工作（这可能是提出建设性的反对意见）,而去尝试处于萌芽中的想法-同时,为了让创新发挥效力,即使再难,也需要对组织进行重组.你想雇一个人来制造一种产品,这种产品很可能跟他以前做的类似.你当然需要看看候选人以前生产的产品样品.大家留在这样的公司,是因为它让你意识到公司总是期待你留下来.公司为你的个人成长投入巨大依靠脑力劳动的公司必须认识到他们在人力资本上的投资是至关重要的. **高效团队形成** 建立一个健康的组织需要管理者建立对质量的执着追求,提供诸多满意的闭环,建立精英意识,允许和鼓励差异性,维护和保护成功团队,提供战略而不是战术方向.闭环就是整体的每一部分皆需要一致满足的“过程”. **改造企业文化和快乐地工作** 风险管理的本质不是让所有风险都消失,而是确保风险发生时有相应的应对措施.应对措施应该提前就经过规划和演练了.

《软件管理沉思录》

计划类型的计划过程：项目团队建设和激励；如何说服管理者采用最佳实践；个人职责、承诺和过程.

管理你的项目 为了完成工作,首先要知道什么是你要努力去实现的.当团队面临巨大交付压力时,一定不能让步,要坚持制定一份计划.两类计划:1.阶段计划:基于时间段的计划,阶段计划关心在这一时间段内你准备如何利用时间 2.产品计划:基于行动的计划,比如开发一个程序或撰写一份报告.一份合格的产品计划应当包括三项内容:1.将要生产的产品规格和重要的性能指标；2.估算工作所需的时间3.进度预测.随着消费者对产品理解的成熟,唯一限制你增加新功能或新特性的就是你的交付能力.

管理你的团队 形成凝胶型团队的一个必需条件是:所有成员对团队的全身心投入.其他几个条件是:1.有可量化的目标:包括详细的计划、功能目标、质量要求等.2.团队的目标必须代表一种意义重大的挑战 3.目标必须能被追踪,并且进度必须以一种可见的方式呈现.团队成员还必须能从团队整体成就中分辨出他们个人的表现,当个人的工作在团队中看不出来时,他们就不会那么努力了.团队中所有成员都必须感到任务是可以完成的,同时理解他们的角色和责任,并就如何完成任务达成一致.

管理你的领导 独断专行的管理方式会让员工失去动力,而且其生产效率要低于最适宜工作环境下的生产效率.**为过程改进给出战略性理由**:清晰地阐述你的提议是什么；理解当前的业务环境：找出高层管理者当前的关注点；对改进的合理性进行初步的检查；以两三个原型为起点制定计划；估算前期一次性的引入成本；测定后期的可能成本；记录下可用的收益数据；估算预期的节约数额；确定如何衡量实际收益；评估改进对高层管理者最关心的问题可能带来的影响；找出提议的改进给组织可能带来的其他益处；准备一个清晰、简短的陈述.最好的领导是告诉他的团队成员应当如何完成工作却不插手具体事务.**管理者希望你能做到的:**按日程进度利用已有的资源去完成此项工作；让你生产的产品同时满足明确的和暗含的需求；让管理者随时了解你的团队的工作进展情况；及时让管理者了解出现的问题并采取正确的对策；与组织中其他所有部门协调一致的工作.

管理你自己 你的任务是要激励团队全力以赴地投入工作.要想做到这一点,你必须信任团队中的所有成员.作为领导者,你工作很重要的一个方面就是保持团队目标清晰明确,确保团队每一位成员都知道他当前的工作对实现目标有什么帮助.你的精力、热情和自律就是榜样.**领导力低下的症状**:1.高层管理者在思考问题时以自我为中心、眼界短视狭隘.2.“官僚惰性”:目前存在的都被认为是合理的,任何的增加或改变要想得到批准都要付出艰苦的努力.3.管理者没有能力及时作出有效的决策.4.某些领导风格太执着于变化,而导致组织永远处于动荡之中.**管理**是利用资源达到某种结果,而**领导**则是激励人们实现某个目标.领导和管理者之间最主要的区别是,管理者命令员工服从他们的指令,而领导者是带领他们完成任务.领导力有两个关键因素,那就是身先士卒,而且相信你的士兵正跟随着你.做好你自己的工作,激励你的团队,而不是要把你的时间浪费在挑剔工作环境有多么糟糕的这类事情上,甚至是你感觉事实的确如此时也不要这么做.

《黑客与画家》黑客是如何成长的以及他们看待世界的一些观点；黑客怎样做出自己的成果,这些成果又是怎样对全世界产生了影响；黑客的工具（编程语言）和工作方法,这是黑客文化的基础和核心.

开源软件开发

高质量:可用性不理想；发布的频率过高造成缺乏成熟的设计；参与的人员太多使得设计的质量不高；文档质量不高；软件越来越复杂而难以控制；集成和版本控制的成本增加；过多冗余的功能.

高开发效率:需求获取及分析:不存在需求不明或与用户难以沟通的问题；设计:不一定要经历从概要设计到详细设计的过程；实现:开发者分散开发,并行工作；可进行快速的 bug 修复.

高强度测试:可以有效地利用开源社区来查找和改进软件的缺陷,测试团队庞大,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**（如果有足够多的眼睛,所有的错误都是浅显的）

敏捷软件开发宣言:个体和互动高于流程和工具,工作的软件高于详尽的文档,客户合作高于合同谈判,响应变化高于遵循计划.也就是说尽管右项有其价值,我们更重视左项的价值.

敏捷的本质:敏捷的本质,是承认软件开发的复杂性.而且承认,这种复杂性,达到了这样的一种程度:“无法通过足够充分的前期准备,而消除后续的风险.甚至于,前期准备得越是充分,后续的风险越大”;软件开发具有复杂性和可变性.

高适应性:产品角度:适用于需求不明确并且快速改变的情况,如系统有比较高的关键性、可靠性、安全性方面的要求,不完全适合；组织结构的角度:组织结构的文化必须支持谈判、人员少而精并且彼此信任、开发人员所作决定得到认可,环境设施满足成员间快速沟通之需要；适用于 30 人及以下较小队伍.

敏捷 VS 计划驱动:计划驱动的工作方式应该是先定义计划,然后按照计划来行事.按照这个意思来说,所有项目都应该是计划驱动的项目.常用敏捷方法都有估算和计划环节,如果不是希望制定的计划来指导（驱动）工作,做计划干嘛？如果计划不是这个作用,那做计划本身岂不是违背了“拒绝浪费”这一敏捷指导思想？可能有一种对计划驱动的理解是到底计划能否允许改变,个人认为,不管是允许改变还是不允许改变,一定有背后的缘由的.做软件开发的人脑袋都不笨,不管是环境决定了能变更还是不能变更,背后一定是有足够的理由的!谁敢说自己一定比做类似飞机导航软件或者大型操作系统（通常认为不适用敏捷方法的环境）的软件工程师强？

TDD:在开发功能代码之前,先编写测试代码,然后只编写使测试通过的功能代码,迭代该过程,直到所有的测试通过,然后重构代码优化.（不可运行-可运行-重构）.测试用例用来表达需求.TDD 实践问题:测试范围的确定；TDD 导致大量的 Mock 和 Stub；Test Case 并没有想像中的那么简单:测试用例的完备与否,测试代码本身逻辑的正确与否都依赖于程序员.

SCRUM:是一个框架,包括 Scrum 团队及其相关的角色、事件、工件和规则.框架中的每个模块都有一个特定的目的,对 Srum 的成功和使用都至关重要.Scrum 框架包括三个角色(产品负责人、Scrum Master、团队）四个仪式（Sprint 计划会议、每日站会、Sprint 评审会议、Sprint 回顾会议）三个物件（产品 BackLog,Sprint BackLog,燃尽图）.具有可操作的过程管理,而 xp 具有较强的操作性.

中国实施敏捷可能会遇到的困难:1.文化冲击,敏捷导致原来项目管理模式改变,推广敏捷很大程度上会触及到企业文化的变革,企业可能无法快速适应 2.当遇到困难的时候,他们会选择性的回到其习惯的开发方式,或者根据自己的情况进行裁减,选择性地实施,可能就会变成半敏捷的方式 3.学习成本

估算 概念:根据软件的开发内容、开发工具、开发人员等因素对需求分析、软件设计、编码、测试与整个开发过程所花费的时间、费用及工作量的预测.确定开发时间和开发成本的过程.**估算内容:**规模估算、工作量估算、成本估算、进度估算、风险估算、缺陷估算、资源估算等.明确一点:估算的偏差是必然会存在的,估算不在于精确而在于有用.**为什么要估算？**估算是一种手段,为了计划、项目管理.

估算方法及优缺点:代码行:直观,可以作为衡量工作量的标准； 在项目结束之前不能得到精确的数据.类比:估算较为准确；要依赖实际经验和类似历史项目.专家:不需要历史数据,适合新项目；主观,专家技术带来误判.自顶向下:估算工作量小,速度快；估算不全面,盲目性大.自底向上:准确性高；会遗漏系统级工作量,估算值往往偏低.参数模型:客观,结果是可重复的；没有采取校准结果往往不准确.功能点分析:基于用例,可保持与需求变化的同步；加权调整需要依赖个人经验. **专家 vs 模型:**1.专家有天然的优势,可以掌握更多的信息,并且可以更加灵活的运用信息 2.构建准确的估算模型很困难,缺少相关性的分析以及学习数据集的数量和质量都会带来严重的影响.

影响估算准确度的因素:1.对项目信息的掌握不完全 2.太过乐观 3.人的主观性 4.缺乏优秀的估算工具 5.缺乏历史数据的支持

软件过程管理基本定律: 如果开发过程可以在统计控制下,改进过程就可以导致更好的结果.将整个软件开发任务作为一个可控制,可度量,可改进的过程.统计控制背后的基本原理是度量.度量是昂贵的有破坏性的,度量的越少越好,只度量必须.软件过程提高软件能力:1 知道开发过程所处的状态 2 开发一个期望的过程的原型 3 建立提高过程质量的举措的列表并按优先级排序 4 为实施这些举措定计划 5 提供资源来实施计划