0.No Silver Bullet：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。

0.1 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。除非次要任务占 9/10，即使全部次要任务的时间缩减到零，效率也无法提升 10 倍。

0.2 摩尔定律：每隔 18 至 24 个月，集成电路上的晶体管数量大约会翻倍，成本降低

0.3 软件开发的根本困难：软件特性中固有的困难——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验证；次要困难：生产上的困难，而非与生俱来的。

0.4 软件的内在特性：**复杂性、一致性、可变性、不可见性**

0.4.1 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;(2)数字计算机本身就比人类建造的大多数东西复杂;(3)软件实体的扩展也不仅仅是相同元素重复添加，而必须是不同元素实体的添加;(4)软件的复杂度是必要属性，不是次要因素。所以不能像其他学科一样抽掉复杂度构建简化模型；（5）函数的复杂度造成函数调用困难、结构复杂度造成难以括扩充、结构复杂度造成安全机制状态上的不可见性；（6）复杂度引发管理问题。

0.4.2 一致性：软件是人造的，必须遵循认为惯例和系统，即接口的一致性。为了维持这种一致性又会带来很多复杂性：各种软硬件交互的密切配合；系统协作导致牵一发而动全身；部分修改导致全局受影响。

0.4.3 可变性：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。

0.4.4 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；这限制了设计过程和交流；0.5 当年的银弹：高级编程语言、面向对象编程、人工智能、专家系统、自动编程、图形化编程、程序验证

1.1 大教堂与市集（Eric Steven Raymond）1.1.1Linux：是一种自由和开放源代码的类 UNIX 操作系统内核；1.1.2 开源许可证：GPL BSD MIT Mozilla Apache 和 LGPL

1.1.3Linus 定律：“足够多的眼睛，就可让所有问题浮现”，即只要有足够多的开发者、测试者和用户参与代码审查与测试，软件中的错误就能很快被发现、定位并容易地被修复。 1.1.4**“Delhpi 效应”：**一群相同专业的(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。1.1.5 两种模式：**大教堂模式**——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；**集市模式**——是并行的、点对点的、动态的开发模式 1.1.6 开源运动：1998 年 2 月 3 日，硅谷的一次会议上，开放源代码（Open Source）被正式提出，后来发展成为开源运动。1.2 开源经济学 1.2.1 替代物品与互补物品：替代物品：替代物品是首选商品太贵时会改买的另一种东西。鸡肉就是牛肉的代替物品。互补物品：通常会和其它产品一起购买的产品。车和汽油是互补物品。计算机硬件和操作系统是互补物品，这也是微软的一大盈利业务

2.人月神话

2.1 造成软件开发项目失败的原因：(1)进度；(2)乐观主义；(2)人月的不合理性

2.2Brooks 定律：（1）向进度落后的项目中增加人手，只会使进度更加落后。（2）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。

2.3 交流与沟通的成本：(1)人月可以交换的前提是任务可以分解且他们不需要沟通；(2)当任务由于次序的限制不能分解时，人手的添加对进度没有帮助；(3)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；(4)沟通所增加的负担由两个部分组成，培训(不能分解，线性变化)和相互的交流(如果任务的每个部分必须分别和其他部分单独协作，则工作量按照 n(n-1)/2 递增)。因为：因为软件开发本质上是一项系统工作——错综复杂关系下的一种实践——沟通、交流的工作量非常大，它很快会消耗任务分解所节省下来的个人时间。从而，添加更多的人手，实际上是延长了，而不是缩短了时间进度。

3.软件过程——CMM（Capability Maturity Model）是一种软件过程改进的模型

(1)初始级(initial)。特征：过程是临时的、混乱的。成功依赖于个人英雄主义。一旦遇到危机，所有规矩都会被抛弃，只顾编码和测试 。改进：建立基本的项目管理（控制承诺、成本、进度）、管理层监督、质量保证以及变更控制。

(2)可重复级(Repeatable)。特征：建立了基本的项目管理控制，使得以前的成功可以被重复 。实现了“承诺控制”。改进：建立过程小组专注于过程改进；定义过程架构；引入软件工程方法和技术（如代码审查、库控制系统）。

(3)已定义级(Defined)。特征：组织已经将过程标准化并文档化，确保了一致的执行 。即使在危机中，团队也会坚持使用已定义的过程 。改进：目前仍是定性的，缺乏数据衡量过程的有效性，建立过程测量的最小集，建立过程数据库，评估产品质量 。

(4)已管理级(Managed)。特征：引入了全面的过程测量，超越了简单的成本和进度，进入了质量和效率的量化管理阶段 。警示：切勿使用过程数据来评价或比较个人/项目，否则数据的可靠性会崩溃。数据的目的是为了照亮产品和改进过程 。

(5)优化级(Optimizing)。特征：组织拥有了基于数据来持续“微调”和优化过程本身的能力 。重点转向缺陷预防和自动化的数据收集 。案例：通过数据分析（如审查与测试发现 Bug 的成本对比），优化资源分配。**改进的逻辑：**先解决混乱，建立基本的项目管理（进入 L2）。将成功的经验标准化为过程（进入 L3）。通过数据量化过程的质量和效率（进入 L4）。利用数据反馈循环来持续优化过程（进入 L5）。

4.Scrum(背景：全面 VUCA（易变、不确定、复杂、模糊）时代)

4.1 起源：1986 年竹内弘高《新型新产品开发策略》首次提出"Scrum"术语；1995 年 Jeff Sutherland 团队在 Easel 公司完成首个完整实践,在奥斯汀举办的 OOPSLA 95 上，Sutherland 和 Schwaber 联合发表了论文首次提出了 Scrum 概念。

4.2 定义：**Scrum 不是构建产品的一种过程或一项技术,而是一个框架**,在这个框架里可以应用各种流程和技术。

4.2Scrum 33355 4.2.1 **三大支柱：**透明：过程中的关键环节必须对负责产出的人显而易见。没有透明的检视会产生误导和浪费。

检视：经常性地检查 Scrum 工件和迈向目标的进度，以便发现偏差。

适应/调整：如果检视发现过程出现偏差或产出不可接受，必须尽快调整过程或材料。

4.2.2 五个价值：承诺：愿意对目标做出承诺；专注：全神贯注于 Sprint 的工作和目标；开放：对工作中的挑战和进展保持开诚布公；尊重：尊重每位成员的能力和独立性；勇气：有勇气做正确的事并处理棘手的问题

4.2.3 三个角色：Scrum Team 足够小保持灵活，足够大可以在 一个 Sprint 中完成重要的工作，通常只有 10 人或更少。Scrum Team 是跨职能的：具有创造价值的全部技能；自管理的：在团队内部决定谁做什么、何时做以及如何做

开发人员：负责每个 Sprint 结束时交付潜在可发布的产品增量，是自管理的跨职能团队

Product Owner：负责最大化产品的价值，管理产品待办列表 (Product Backlog)。

Scrum Master：负责推广和支持 Scrum，帮助团队理解理论、实践、规则和价值，清除障碍。**4.2.4 三个工件**：产品 Backlog：产品需求的有序列表，是产品需求的唯一来源

Sprint Backlog：选入当前 Sprint 的产品待办项，加上交付这些项的计划

增量：当前 Sprint 完成的所有产品待办项的总和，以及之前所有 Sprint 产生的增量价值

4.2.5 五个事件：Sprint：一个固定长度的事件（通常 2-4 周），包含其他所有事件；Sprint 计划会议：规划本次 Sprint 要做的工作

每日 Scrum 会议：每天 15 分钟，开发团队同步进度并规划未来 24 小时的工作；Sprint 评审会议：Sprint 结束时检视增量，调整产品待办列表；Sprint 回顾会议：检视团队自身的工作方式，并制定改进计划。

4.3 用户故事：停止写出完美文档的“执念”，使用 who、what、why 格式写 user story

4.3.1Ron Jeffries 的 3C 原则：卡片(Card) 在一堆卡片上写下你期望的软件特性； 交 谈(Conversation)聚在一起对要开发的软件进行深入讨论；确认(Confirmation)对完工条件进行确认

4.3.2DoD**“Definition of Done”**用于描述一个用户故事、任务或功能何时可以被认为真正“完成”。它是团队对完成工作的标准化定义，确保开发过程中每个增量都符合质量要求并准备好交付。

4.3.3 **行为驱动开发**（Behavior-Driven Development，BDD）：通过定义软件的行为来驱动开发过程，增强团队对需求的理解，并确保开发的软件满足业务目标。

4.3.4：用户故事 INVEST 原则

(1)独立性（Independent）要尽可能的让一个用户故事独立于其他的用户故事。

(2)可协商性（Negotiable）一个用户故事的内容要是可以协商的，用户故事不是合同。

(3)有价值（Valuable）每个故事必须对客户具有价值（无论是用户还是购买方）。

(4)可以估算性（Estimable）开发团队需要去估计一个用户故事以便确定优先级，工作量，安排计划。(5)短小（Small）— 一个好的故事在工作量上要尽量短小，至少要确保的是在一个迭代或 Sprint 中能够完成。

(6)可测试性（Testable）—一个用户故事要是可以测试的，以便于确认它是可以完成的。每个用户故事都应像标准快递箱——独立封装（Independent）、地址清晰（Valuable）、体积适中（Small）、运费明确（Estimable）、可追踪（Testable）、允许改派（Negotiable）。这样的标准化包装，才能让需求运输既高效又可靠。

大规模敏 SAFe：为大企业提供一个分层次的敏捷实施蓝图。2011 年发布。强调通过统一节奏和里程碑来协调多个团队，批评者认为 SAFe 层级过多，引入大量新角色和流程。

LeSS（Large-Scale Scrum）核心理念是“Large-Scale Scrum is Scrum”，不引入额外的复杂框架，而是通过扩展 Scrum 的基本原则和实践来适应多团队环境。

4.3 估算：估算是很困难的，因为这是在预测未来；估算经常是错误的，但是估算过程仍然是有用的；团队共同完成估算可以让大家建立对工作一致的理解；根据收益递减原理，不应在估算上花太多的时间。4.4 估算单位：选取可识别的最小用例为 2 个 story point.其它估算都是相对值，在所有 sprint 中保持该相对值一致；估算速度时，估计能在一个迭代周期内能够完成的 story point 数量。

5.XP(Extreme Programming) 极限编程一种敏捷开发方法诞生于 20 世纪 90 年代后期(Kent Beck)：把有益的实践做到极致

5.1XP 变更成本曲线：随着时间的推移，更改程序的成本呈指数增长，在一段软件中修复一个问题的成本会随时间呈指数增长。这导致传统方法强调“早期做好一切”，试图避免后期修改。但 XP 认为变更成本曲线可以变平，实现技术：

(1)在技术方面，面向对象是一项关键技术;(2)简单设计，没有额外的设计元素;(3)自动化测试，所以我们有信心知道我们是否意外地改变了系统的现有行为;(4)重构技术;(5)CI/CD:及时发现问题。XP 鼓励频繁交付和持续部署。“Beck 的曲线并非无视规律，只是把反馈周期从几个月缩短到几分钟，结果就没有机会让成本成指数增长”平坦曲线的前提是严格执行 XP 实践。对于那些未充分实践 XP 核心，后期改动一样痛苦。

XP 价值观：Communication 交流 Simplicity 简单 Feedback 反馈 Courage 勇气

5.2XP 实践：1.计划游戏：快速制定计划。业务人员决定范围、优先级、发布的组成和日期，开发人员决定估算后果流程和安排；2.小发布周期 Short Releases：尽可能频繁地将产品发布到生产环境，快速获得用户反馈，降低发布风险；3.隐喻 Metaphor：用简单的、共同的词汇或比喻来描述系统的运作方式，让大家都听得懂；4.简单设计：只设计当前需要的功能，不进行过度设计；5.减少浪费，使系统保持轻量，便于适应变化；6.测试：先写失败的测试用例，再写刚好能通过测试的代码，最后优化代码结构；7.重构：在不改变

外部行为的前提下，调整代码的内部结构；8.消除技术债务，保持代码整洁；9.结对编程：即时的代码审查，提高质量，促进知识共享，减少单点故障；10.代码集体拥有制：任何人在任何时候都可以修改任何代码。没有“这是我的模块，你不能动”的说法；11.持续集成：开发人员每天多次将代码合并到主干，每次合并都触发自动构建和测试；40小时工作制；现场客户；12.编码规范

5.3TDD、敏捷测试

敏捷测试：我们将在编码前逐分钟编写测试。我们会永久保留这些测试，并经常一起运行它们。我们还会从客户的角度推导出测试。测试应当是独立且自动的，与其他测试互不影响且测试是自动化的。；做有价值的测试：测试那些可能会出问题的东西；测试像赌博；只应该编写那些能带来回报的测试；程序员测试：写代码前先写测试；客户测试

TDD：通过先写测试再写实现的逆向流程驱动代码设计：三阶段循环：红（失败测试）→ 绿（通过实现）→ 重构（优化代码），终极目标：创建可测试、可维护、最小化的代码。标准流程：编写失败测试；快速实现通过；逐步泛化；持续重构；核心价值：缺陷预防；设计引导；降低耦合；活的文档；设计：XP 涉及大量的设计工作，设计是为了长期轻松修改软件，倡导演进式设计而非计划设计

简单设计四大准则：通过所有测试；消除重复；清晰表达意图；最小化元素数量

YAGNI（"You aren't gonna need it"）原则：拒绝未经验证的灵活性和零成本的超前开发

5.4 持续集成：持续集成 (Continuous Integration, CI) 是一种软件开发实践，团队成员频繁地（通常每天至少一次）将他们的代码更改集成到共享的代码库主线中。每次集成都会触发一次自动化的构建过程，该过程包括编译、链接以及运行一套全面的自动化测试套件。是尽早发现并解决集成过程中的问题

CI 步骤：获取最新代码→本地构建与开发→集成前检查→本地构建通过与推送

持续集成的好处：降低风险；促进改进与发布

特性分支：拒绝长期存在的特性分支，强调每天多次提交集成到主代码库

持续交付与持续部署：每次主线构建通过所有自动化测试和验证后，软件处于可发布状态，但最终部署到生产环境通常还需要一次手动触发；而在持续部署中，这个手动触发步骤也被自动化了：只要主线构建成功通过了部署流水线中的所有阶段（包括所有自动化测试和检查），它就会自动被部署到生产环境中，无需人工干预。

6.看板：起源于制造业，是一种通过使用可视化、拉动式系统来优化流程中价值流动的策略。看板由三种协同实践组成：定义并可视化工作流程→主动管理工作流程中的事项→改进工作流程。这些看板实践统称为看板系统；参与看板系统价值交付的人员称为看板系统成员。地位：Kanban 已被普遍视为与 Scrum、XP 等并列的重要敏捷实践框架。Kanban 实现敏捷和 DevOps 的流行框架之一；

可视化：把工作内容贴在白板上，使得工作对所有人可见。

可视化工作流：Kanban 系统成员对于流的明确且共同的认知，就称为 “工作流的定义。(1)在工作流中移动的各个价值单位的定义。这些价值单位被称为 “工作项”;(2)在工作流中“开始”与“完成”工作项的定义(3)工作项自开始到结束时流动所经过的一个或多个已定义状态。(4)如何从开始到结束间控制 WIP 的定义;(5)有关工作项如何在每个状态中从开始到完成的明确的政策;(6)一份用来预估一个工作项从开始到完成所应花费的时间的 “服务水平期望”。

在制品限制：**在制品限制 (WIP Limit):** 给每个工作阶段设 WIP 限制是 Kanban 重要元素。通常在列顶标注数字表示上限。强制团队避免多任务过载，减少排队等待。当某列达上限，需暂停拉入新任务，优先完成现有任务。

7.DevOps 矛盾：软件开发 (Dev) 与 IT 运维 (Ops) 领域的传统模式弊端，两者各自为政，目标与激励机制存在天然冲突。这种“割裂”的墙导致了效率低下、交付延迟和频繁的相互指责，促使业界深刻反思并寻求新的协作范式。

Iac((Infrastructure as Code-IaC)基础设施即代码：使用代码化、声明式的方式来管理和配置计算、网络、存储等基础设施。价值：一致性与可重复性；版本控制与审计；自动化与效率；关键实践：自动化配置管理；容器；容器编排

安全左移：实现内建安全，而非事后补救。在开发和集成环节就主动发现并修复安全问题，而非等到最终测试阶段；第三方依赖包的安全；代码本身的安全：开发人员在写代码时，工具能自动发现逻辑漏洞或不规范的写法

DORA 指标：DORA 指标体系是衡量团队软件交付能力的事实标准，包括交付速度和交付质量两个类别；交付速度：部署频率：成功向生产环境发布代码频率；变更前置时间：代码提交到成功部署至生产环境的时间

交付质量：变更失败率；平均恢复时间 MTTR；金丝雀发布：将新版本发布给一小部分用户，在确认没有问题后，再逐步扩大范围，直到覆盖所有用户。

驱动目标：严控 DORA 指标中的“变更失败率”

发布流程：金丝雀部署→并行对比→数据分析→决策(扩大流量还是终止发布)

NetFlix 的实践：数亿用户、数千工程师、成千上万微服务，传统流程难以承载

平台工程:提供了标准化的环境与工具链，为“快”速交付奠定基础。

混沌工程:在系统和文化层面注入了强大的“韧”性基因，使系统不惧变更

金丝雀发布：为高速变更提供了智能化的“稳”定性保障，确保每次发布都安全可控。这套体系的协同，最终实现了 DORA 指标中两大核心目标的统一：极致的部署频率与极低的变更失败率、极快的服务恢复时间。

8.敏捷概述：敏捷的本质：承认软件开发的复杂性，并承认这种复杂性已达到“无法通过足够充分的前期准备，而消除后续的风险。敏捷软件开发是当前应对模糊需求、快速变化需求的最佳方式。（拥抱复杂，直面变化）

关注点：传统观点强调遵循计划、按时按质不超预算；敏捷观点强调软件创造的价值

敏捷宣言：

个体和互动高于流程和工具：业务人员和开发人员必须相互合作

工作的软件高于详尽的文档：持续不断的及早交付有价值的软件

客户合作高于合同谈判：开发团队和客户之间要保持尽可能公开和顺畅的对话

响应变化高于遵循计划：欣然面对需求变化，即使开发后期也一样。敏捷过程掌控变化。

敏捷软件开发知识体系：(1)价值观(敏捷宣言：简单、反馈、沟通、勇气、尊重)、(2)原则：快速反馈、及早交付有价值的软件使客户满意、以简洁为本；(3)方法：XP、Scrum、看板;(4)实践：TDD、CI/CD、结对编程；(5)工具

协同演进范式

9.新方法学

本质区别：敏捷方法是“适应性”而非“预见性”，传统方法遵循流程，拒绝变化，而**敏捷欢迎变化**，他的目标即适应变化；**敏捷方法是面向人的**，而非面向过程的。传统方法强调过程，敏捷方法强调人的重要性。软件开发中所有的工作几乎都是设计，创造性的过程是不太容易计划的，因此，可预见性是个不可能达到的目标。

不可预见过程的控制：要随时知道我们在开发中的情形处境，这需要一个诚实的反馈机制来不断准确地告诉我们。这种机制的关键之点是“迭代式”开发方法。迭代式开发的要点是经常不断地生产出最终系统的工作版本，这些版本逐步地实现系统所需的功能。

理由：没有什么比一个集成了的、测试过的系统更能作为一个项目扎扎实实的反馈。

适应性的客户：这类适应性过程需要与客户建立一种新型的关系。固定价格合同需要稳定的需求，即一个可预见性过程；通常敏捷方法是固定时间和价格，而让范围能够可控制地变化。

面向人的过程的管理：(1)实施敏捷型过程的一个关键之处是让大家接受一个过程而非强加一个过程(2)另一点是开发人员必须有权作技术方面的所有决定。

10.精益软件开发：精益软件开发的 **7 大支柱**：消除浪费：识别并根除一切消耗资源但不为客户创造价值的活动；内建质量：质量无法在后期“检查”出来，必须在开发过程中的构建。TDD、CI、结对编程；创建知识：软件开发是学习与发现的过程，代码仅是知识的快照。PDCA 科学循环环(Plan-Do-Check-Act): 将开发视为科学实验，每次迭代都是创造知识的机会；知识基础设施：通过代码评审、Wiki 文档和“社区实践”促进隐性知识的传播，防止“重新学习”的浪费推迟决策：将关键决策推迟到“最后责任时刻”,以在信息最充分时做出最优选择；快速交付：目标是缩短周期时间 (Cycle Time)，而非让工程师“工作得更快。严格限制在制品数量 (WIP)；尊重人：在知识工作中，人是唯一的生产资产。尊重是效能的基础。内在驱动力与心理安全(将错误视为学习)

全局优化：局部优化往往导致全局受损。打破部门壁垒，对对最终客户价值负责，而非对中间环节的局部指标

7 种浪费：

部分未完成的工作	数周未合并的 git 分支	Mura（不均衡）：根源，工作流的剧烈波动，期末工作量激增
额外的功能	开发炫酷的 UI	
额外的流程	无人阅读的冗长文档	Muri(超负荷):结果，系统或个人承载能力超限，通宵赶工
交接	设计与编码交接后不沟通	
任务切换	同时进行 3 门课的大作业	
延迟	等待队友 Code Review	muda(浪费):必然产物，超负荷导致错误率上升和效率下降
缺陷	截止前才集成测试，有 bug	

实践的桥梁：价值流图：需求→开发→测试→部署，目的：识别浪费的强有力工具，描绘从概念到交付的全过程	看板系统：代办→开发中→评审中→完成，目的：实现"拉动系统"和"连续流"的工具,核心机制：限制在制品，这强迫团队协作解决瓶颈，将焦点从最大化个人忙碌转向最大化价值流动效率

真题：1.为什么这 **XP4 个实践必须互相协作和互补**？XP 的核心在于反馈环和勇气，这四个实践构成了一个相互支撑的安全网：简单设计是目标，保持系统最简化；TDD 是手段和保障它保证了代码在设计之初就是可测的，并且为重构提供了安全网；重构是路径。随着需求增加，必须不断优化代码结构以保持“简单设计”；持续集成是频率。它确保重构和新代码不会破坏现有系统。

2.如果只进行一个实践，会导致哪些问题？仅重构如果没有 TDD 的测试覆盖，重构极易引入 Bug，导致系统崩溃，开发者不敢动旧代码；仅 TDD 如果写了测试但不重构，代码会变得臃肿、难以维护，测试代码本身也会变成巨大的技术债务；仅持续集成如果没有 TDD 支持，CI 只是在自动编译烂代码，无法尽早发现逻辑错误；如果没有简单设计，集成过程会非常痛苦且缓慢；仅简单设计如果不进行重构，随着新功能加入，设计很快会腐化变复杂；没有 TDD，无法验证设计是否满足需求。

3.Scrum 的优势与局限性：优势：(1)适应变化：基于经验主义，通过透明、检查和调整来应对需求的不确定性；(2)价值驱动：优先交付高价值需求，确保投资回报率最大化;(3)高可见性： 进度和障碍对所有人透明。局限性：缺乏工程实践；对团队要求高

4.Scrum 项目管理：将传统的项目管理职责分散到了角色和迭代过程中：

通过“团队角色分工”实现管理：抄写 4.2.3

通过“迭代管理 (Sprint)”实现管理：抄写 4.2.5

5.**Scrum 方法在开发阶段对工程实践的规定不足，会导致哪些问题？**(1)技术债务累积：为了赶 Sprint 结束的截止日期，团队可能会写出低质量代码。没有重构和 TDD 的强制要求，代码库会迅速腐化。(2)不可持续的速度：项目初期看似很快，但随着 Bug 增多和代码复杂度失控，开发速度呈断崖式下跌，无法维持迭代的节奏。(3)“完成”的假象：但由于缺乏自动化测试，集成时才会发现大量回归错误，导致项目后期陷入“集成地狱”。(4)重构恐惧：没有测试覆盖网，团队不敢修改旧代码。

6.XP 特点与优势。

特点：极限将被证明有效的实践推向极致；反馈：强调短周期的反馈环；人本与沟通：强调面对面沟通。核心优势：极高的代码质量。极速响应需求变更的能力。减少文档负担，聚焦可工作的软件。

保证软件质量：TDD、结对编程、现场客户。保证开发速度：测试和结对，减少了返工时间、简单设计与重构始终保持代码简洁、持续集成让软件随时可用可发布。

7.Scrum 与 XP 如何在管理和技术上融合？Scrum 的管理外壳+XP 的工程内核：管理层使用 Scrum 来管理需求列表、Sprint、组织会议，解决“做什么”和“团队协作”的问题。技术层：在 Sprint 内部，团队严格遵守 XP 的工程实践：TDD、简单设计、持续集成、重构等。