**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。

**软件工程与计算机程序设计**：软件工程存在于各种应用中，存在于软件开发的各个方面。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。

**没有银弹**

**结论**：1. 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；2. 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；3. 软件开发在根本上就是困难的。

**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升。

**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）

**根本困难——软件特性中固有的困难**：我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。

**现代软件系统的4个内在特性**：

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，软件实体很复杂，没有两个软件是相同的（如果有我们会将它们合并）；b) 数字计算机本身就比人类建造的大多数东西复杂；计算机拥有大量的状态，这使得构思、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。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。抽掉复杂度的软件实体描述常常也去掉了一些本质属性。e)**复杂度问题造成软件产品开发问题**：沟通困难，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，妨碍概念上的完整性；难以寻找和控制所有离散出口；引起大量学习和理解上的负担（信息隐藏）
2. **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，来自若干必须遵循的认为习惯和系统（VS物理中往往有通用原理）。软件开发面对的人，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**：a)软件实体经常会遭受到持续的变更压力。（相较于汽车、建筑，包含很多功能的软件可以很容易地进行修改——它是纯粹的思维活动产物，可以无限扩展）；b)软件的变更：人们要求扩展，更改功能；硬件的变化；c)软件与真个社会连成一体，社会在变，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（区别于机械制图等）；软件的客观存在不具有空间的形体特征（没有一种2/3维的图形可以描述软件）；限制了个人的设计过程，阻碍了相互的交流——UML。

**当年的银弹**：Ada和其他高级编程语、面向对象编程（简化了设计人员对软件的表达，解决了较高层次的次要困难）、人工智能、专家系统、“自动”编程、图形化编程（软件非常难以可视化）、程序验证、环境和工具（有限）、购买和自行开发（通用软件）、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。

**大教堂与市集-开放源代码软件**

**自由软件与商业封闭软件**：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。

1. 每个好的软件工作都开始于搔到了开发者本人的痒处。2. 好程序员知道改写什么，伟大的程序员知道该重写（和重用）什么。3. 计划好抛弃，无论如何，你会的。4. 如果你有正确的态度，有去的问题会找上你。5. 当你对一个程序失去兴趣时，你最后的责任就是把他传给一个能干的后继者。6. 把用户当做协作开发者是快速改进代码和高校调试的无可争辩的方式。7. 早发布、常发布，听取客户的建议。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地找出并被一些人纠正（教堂长期测试——失望；多人测试、更多的更正）。原因——**Delphi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。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. 如果你像对待最宝贵的资源一样对待你的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。 （你在开发中碰壁了，头破血流，反省后才能得到最好的解决方案）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会带来你从来预期不到的用处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，所以它就推广开来了）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。

**集市风格的必要先决条件**：1. 不能以市集模式从头开发一个软件；2. 创建社团需要的是有趣的项目；**组织者**：1. 协调人要能够将其他人的好设计重新组织起来；2. 协调人必须有良好的人际和交流能力。

**Brooks定律**：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，如果就是全部，那Linux就不可能成功。

**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。

**开源软件经济学**

**代替物品和互补物品**：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（计算机与操作系统）

**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。例子：IBM的PC构架利用现成的零件、MS-DOS是IBM的互补物、xBox使用普及的PC硬件和DirectX、IT咨询公司IBM普及企业软件，开放Netscape让服务器赚钱、Sun和Hp让软件普及化（Gnome）由硬件赚更多的钱。盈利需要在软件和硬件上有一个优势，通过让另一个普及化进行收益。

**开源软件获利方法**：1、需要获利空间向其上下游两方面延伸，并且和运营商确定合适的商业模式（可采用我方技术，服务入股，运营商设备网络投入折成股份，根据比重对客户所产生的移动网络费用及使用量来分红）2、针对更加上游客户，提供更多功能，包括类似GPS定位，设定10米的相隔距离，当双方不知不觉走散时，其移动电话会自动报警等功能，为客户提供更多增值服务，以增加获取利润的空间。

**人月神话**

**缺乏合理的时间进度是项目滞后的主要原因**：1. 对估算技术缺乏有效的研究；2. 错误地将进度和工作量相互混淆（将人月互换）3. 由于缺乏信心，经理不会持续进行估算；4. 缺少对进度的跟踪和监督；5. 进度偏移时，下意识要增加人力。

**乐观主义**：所有系统编程的进度安排背后的第一个假设是：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。

**人月**：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。 因此我认为用人月作为衡量一项工作的规模是一个危险和带有欺骗性的神话。 它暗示着人员数量和时间是可以相互替换的。仅适用于任务可以分解给相互之间不需要交流的参与人员。**沟通负担**：培训和相互交流。

**为什么添加人手无效**：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。 因此添加更多的人手， 实际上是延长了，而不是缩短了时间进度。**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。**Brooks法则**：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。

**敏捷软件开发**

**软件成功的传统观点**：按时完成、费用不超过预算、符合设计规格；不太成功：可以运行、超期、超出预算、少于设计规格；失败的：取消的项目。

**软件成功的敏捷观点**：为客户创造价值。以提升收益为首要目标。

**敏捷软件开发：**推动开发迭代；强调面对面交流；将可运行的软件作为评判开发过程的主要标准。

**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；软件开发需求可变性“实际上具有其合理性”。

**敏捷软件开发宣言：**我们一直在实践中探寻更好的软件开发方法，身体力行的同时也帮助他人。由此我们建立了如下价值观：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。

**敏捷适用性：**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、较小规模。

**敏捷缺陷：**缺乏结构和必要文档;初级开发者难以掌握;早期软件设计不够;要求公司文化变更;合同签署方面问题;有时会没有效率;项目开始时无法估算(投标问题);范围剧烈变更的危险(没有详细的需求文档);功能驱动,无法表示非功能需求;缺乏文档,设计;没有达到宣称的效果——大规模提高生产力。

**注**：敏捷软件开发人作用的认识:“敏捷型方法”是面向人的而非面向过程，它们试图使软件开发工作顺应人的天性安排，而非逆人，它们强调软件开发是一项愉快的活动，实施敏捷开发的一个关键之处是大家接受一个过程而非强加过程，通常软件开发过程是管理人员决定这种过程经常受到抵制，特别是管理人员已经脱离软件开发过程很长时间了，而接受一个过程需要一种自愿能力，这样大家就能积极参与进来，正由于IT行业技术发展变化之快，所以管理层必须信任和依靠当前的开发人员。

**敏捷宣言遵循的原则：**我们最重要的目标，是通过持续不断地及早交付有价值的软件使客户满意；欣然面对需求变化，即使在开发后期也一样；为了客户的竞争优势，敏捷过程掌控变化；经常地交付可工作的软件，像个几星期或一两个月，倾向于采取较短的周期；业务人员和开发人员必须相互合作，项目中的每一天都不例外；激发个体的斗志，以他们为核心搭建项目；提供所需的环境和支援，辅以信任，从而达成目标；不论团队内外，传递信息效果最好效率也最高的方式是面对面的交谈；可工作的软件是进度的首要度量标准；敏捷过程倡导可持续开发；责任人、开发人员和用户要能够共同维持其步调稳定延续；坚持不懈地追求技术卓越和良好设计，敏捷能力由此增强；以简洁为本，他是激励减少不必要工作量的艺术；最好的架构、需求和设计出自自组织团队；团队定期地反思如何能提高成效，并以此调整自身的举止表现。

**敏捷VS计划驱动：**适应性,短计划VS可预期(严格遵循预先计划的需求、分析、设计、编码、测试的步骤顺序进行，变更代价高昂导致难以应对变更)；短周期VS长周期；可运行软件进行度量VS可交付产物或步骤成果进行度量；敏捷仍有严格的规范,与最终产物有关。

**Scrum**

**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。

**产品负责人（PO）**负责最大化产品以及开发团队工作的价值；**Scrum Master**负责确保Scrum被理解并实施；**Scrum团队**(3-9)模式的目的是最大限度地优化适应性、创造性和生产力。

**产品BackLog**：Scrum的核心，是按照重要性排序的需求或故事的列表（客户语言描述的客户需求user strory），添加类别、组件字段，保持在业务层面。**Sprint BackLog**：由Sprint计划会议决定，是产品Backlog的一个快照。有信息页和白板两种形式。燃尽图表示生产速度，提醒团队适当修改生产速度。

**Sprint计划会议**：会议之前需要进行Sprint计划会议准备工作（SM的工作）；会议需要全员参与，产品负责人也必须参与。在会议之前，要确保产品backlog有序。会议内容：确定Sprint长度（3周），Sprint目标（业务术语），决定Sprinr中要包涵的Story（Sprint Backlog，团队决定），对每个Story进行讨论（索引卡），估算（单位故事点，纸牌估算法——集体估算，Delphi效应；真正参与工作的人作出的估算较高），将Story拆分为Task。

**Daily Meeting**：不超过15分钟，三个问题：干了什么，准备干什么，遇到什么苦难。更新燃尽图。

**Sprint演示会议**：Sprint结束于演示。让别人了解、团队得到认可、交流、得到他人的反馈、驱使团队真正完成工作。需要一个checklist。

**Sprint回顾会议**：仅次于Sprint计划会议，主题：如何在下一个Sprint中做的更好？1-3小时。轮流发言，各自讲出好坏。对比预估和实际生产率，讨论。SM总结。——可以使用白板。

**Scrum结合其他技术**：Scrum多数实践是非技术的。Scrum和XP（TDD，测试驱动开发）；技术架构、界面（首先设计导航）、数据库；持续集成；编码规范；结对编程简单设计与重构。

**XP特点：**短开发周期,提供及早的、具体的、持续的反馈;增量计划方式;能够灵活安排功能的实现,以对变化的业务需求做出反应;由程序员、客户和测试人员编写的自动测试来监控开发进度,支持系统的演化,并尽早发现缺陷；通过口头沟通、测试和源代码交流系统结构和意图;演化的设计过程贯穿整个系统生命周期;依赖于能力普通但能积极参与的程序员之间的紧密协作;各种实践兼顾项目成员的短期直觉以及项目的长期利益。**XP解决开发风险:**进度延迟(XP提倡短发布周期);项目取消(XP中的最小发布必须是满足最大商业意义的);系统恶化(XP中创建并维护一整套自动测试);缺陷率(XP测试由开发者和客户共同完成);业务误解(XP提倡业务人员成为团队成员);业务变更(XP缩短了发布周期,因此在一个单独的发布周期中几乎没什么变更);错误特性太多(XP坚持只解决最高优先级的任务);人员流动(XP要求程序员估算自己工作所需时间并完成；XP同样鼓励团队中的相 互沟通)

XP的基础和价值观是交流、朴素、反馈和勇气。

**RGP课程讨论**

**开源软件**

**概念**：开放源代码、软件修改、拷贝、再发布

开源是指一套**开发方法**，而自由软件则是一场社会运动。自由软件强调自由的软件精神。

**高效率：**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

**质量问题：**可用性不理想：开发者缺乏在可用性，HCI方面的设计经验，常常会模仿商用软件的设计，缺乏自主创新，导致可用性和用户满意度不高；发布的频率过高造成缺乏成熟的设计：大部分开源的项目都没有得到详细并有计划性的设计；参与的人员太多使得设计的质量不高；文档质量不高：开源软件的文档质量并没有商业软件那么好，商业软件有完善的技术文档和用户手册，但很多时候看文档比看源码更有效率；软件越来越复杂而难以控制：参与开发的人员众多，并且开发者们都往往按照自己的需求以及意愿来进行开发；集成和版本控制的成本增加；过多冗余的功能。

**安全问题:** 由于开源的混乱和代码审查工作的不到位，导致很多被安插了后门的代码存在。

参与同一个开源软件的各个版本的开发工作的有很多人员，他们之间没有充分的直接交流接触，而各个人员独特的设计逻辑并不为大家所充分了解，导致一些暗藏后门的逻辑无法被审查出来。

**开源的发展：**开源的意义就是为了挖掘新的产业模式，盈利模式是开源发展的根本。

**敏捷开发**

**敏捷VS计划驱动：**计划驱动开发：通过从需求到已完成的代码等一系列代表物来推动软件开发的过程，明确地依赖于明确的步骤。敏捷开发：针对传统的瀑布开发模式的弊端而产生的一种新的开发模式，目标是提高开发效率和响应能力。适应性,短计划VS可预期(严格遵循预先计划的需求、分析、设计、编码、测试的步骤顺序进行，变更代价高昂导致难以应对变更)；短周期VS长周期；可运行软件进行度量VS可交付产物或步骤成果进行度量；敏捷仍有严格的规范,与最终产物有关。

**个人理解**：在这里所谓的计划驱动开发指的是敏捷之外的传统软件开发方式。

区别：应用1.主要目标：敏捷目标是快速交付价值和响应变更，快速变更的环境中，反应式的态度具有优势，但是有些风险。计划的目标是可预见性、稳定性和高可靠性，提前行动的态度对于稳定的环境非常有效，认证需要有计划和规格说明。2.规模：敏捷最适合规模较小的项目。计划驱动对于大型、复杂的项目来说是必需的。3.环境：敏捷适合频频变更的环境，但有一些风险。计划驱动开发需要稳定性。管理1.客户关系：敏捷提倡专职的、在一起工作的客户，重中之重是客户代表和用户之间的接口。计划驱动依赖于合同和规格说明，重中之重是开发者和客户之间的接口。2.计划和控制：敏捷者把计划看作一种达到目标的手段，是“计划行为驱动”，而非“计划驱动”。3.项目沟通：敏捷依赖于频繁的、人和人之间的沟通。计划驱动开发依赖于显示的、文档化知识。技术：1.需求：敏捷软件开发把非正式的、由用户指定优先级的素材作为需求。2.设计：敏捷软件开发提倡简单设计3.测试：敏捷编码之前开发测试，增量地测试。计划驱动开发测试的是规格说明。

人员1.客户：敏捷强调专职的、工作在一起的客户代表，而计划驱动开发则依赖于大量预先的，客户和开发者之间的合同计划和规格说明方面的工作。2.文化：敏捷者喜欢更多的自由度，计划驱动者需要清晰的过程和角色。

**RGP观点：**敏捷和计划驱动不是对立的，敏捷也需要计划，只不过敏捷的计划周期较短，如Scrum中的sprint。

**XP实践：**①计划博弈：通过结合使用业务优先级和技术评估来快速确定下一个版本的范围。当计划赶不上实际变化时，就应更新计划；②小发行版本：将一个简单系统迅速投产，然后以很短的周期发布新版本；③隐喻：用有关整个系统如何运行的简单、众所周知的故事来指导所有开发；④简单设计：不必要的复杂性一旦被发现就马上去掉；⑤测试驱动⑥重构：程序员重新构造系统以去除重复、改善沟通、简化或提高柔性；⑦结对编程：所有的生产代码都是由两个程序员在同一台计算机上编写的；⑧集体所有权：任何人在任何时候都可以在系统中的任何位置更改任何代码：⑨持续集成：每天多次集成和生成系统，每次都完成一项任务；⑩每周工作40小时：一般情况下，一周工作不超过40小时，不要连续两个星期都加班；⑥现场客户：在团队中加入一位真正的、起作用的客户，他将全职负责回答问题；⑥代码规范：程序员依照强调通过代码沟通的规则来编写所有代码。

**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哟，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。

**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的改变，在企业里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