



《软件工程》

Software Engineering





华中科技大学计算机学院

Email: cherryyuanling@hust.edu.cn

Office: N1-413

Name: 袁 凌

专属QQ群:



群名称: 2017级软件工程
群 号: 782525694



《软件工程》参考书

1. 《软件工程（第4版）》 张海藩 人民邮电出版社
2. 《现代软件工程：构建之法》 邹欣 人民邮电出版社
3. 《软件工程：实践者的研究方法》
郑人杰等译 机械工业出版社
(< Software Engineering: A Practitioner's Approach>, Roger
S. Pressman)



课程基本要求

- 1.课堂小练习：课堂随机练习上交；
- 2.课程大作业：上机讨论检查+实验文档。

成绩：

大作业（80%）+ 小练习（20%）

注：大作业相关任务书和文档随后会发布。



在有些人眼里，今天的软件开发似乎已成为简单的事情，已有了不少很好的开发工具和软件库，软件开发人员训练有素，都强烈渴望去编写很酷的软件，可以在几天的时间里编写出一个相当复杂的软件。但为什么有一些软件能够得到用户的喜欢，而另一些则不能？为什么有些软件能够在市场上成功，而有些则受到冷落？由此可见，开发软件并不一定难，难就难在如何开发有用的软件。

微软凌小宁博士

2019/9/12



我最大的心得是，一个产品一定要找到能够真正适用的场合，不能只是为了技术而从事技术，为了研究而进行研究，却不管用户对你所研究的技术和产品有没有需求。否则，无论你的技术是多么优秀，多么先进，恐怕你的产品在市场上都无法获得成功。

微软张益肇博士



课程的性质、目的与任务

软件工程是计算机科学与技术专业的一门专业核心课程。通过本课程的学习，使学生掌握系统的软件开发理论、技术和方法，使用正确的工程方法开发出成本低、可靠性好并在机器上能高效运行的软件，为今后从事软件开发和维护打下坚实的基础。



课程主要内容

本课程比较全面、系统地介绍软件工程的概念、技术与方法。

主要包括：软件工程概述、软件生存周期及软件需求分析、软件设计方法、软件测试技术等。

软件工程和下列的学科相关：

计算机科学、计算机工程、管理学、数学、项目管理学、质量管理、软件人体工学、系统工程、工业设计和用户界面设计。



各 章 学 时 安 排

知识点	学时	章	学时	章	学时	章	学时
软件基本概念	3	结构化方法	8	面向对象方法	8	软件项目管理	2
软件工程概念 生命周期模型	3						
理论课学时 24个学时； 8个学时讲授理论知识， 4个学时进行项目展示。							



第一篇 软件工程与软件过程

传统工程




水利工程 建筑工程 机械工程

新兴工程



气象工程 生物工程 软件工程

本章将对软件的特点、软件的危机、以及软件工程学科的形成、软件生命周期等方面的问题和基本概念



课堂练习 15 分钟

- 请拿出纸和笔，或者打开电脑，启动你的编程环境（**IDE**）



Program vs. Software/Service

- 小学老师要每周给同学出300道四则运算练习题。
- 这个程序有很多种实现方式:
 - ☐ C/C++
 - ☐ C#/VB.net/Java
 - ☐ Excel
 - ☐ Unix Shell
 - ☐ Emacs/Powershell/Vbscript
 - ☐ Perl
 - ☐ Python
- 两个运算符，100 以内的数字，不需要写答案。【估计时间1】
- 现在估计写这个程序需要的时间
- 马上把代码写出来
- 需要写答案，并且保证答案在 0..100 之间 【估计时间2】
- 有条件的请马上把程序和运算结果算出来



软件：很多细节

- 这个老师发现你的程序很好用，其他老师都想要一份，有些新的要求：
 - 题目避免重复
 - 可扩展性
 - 可定制（数量/打印方式）
 - 具体定制
 - 是否有乘除法，是否有括号，数值范围，加减有无负数
 - 除法有无余数，是否支持分数，打印中每行的间隔可调整
- 现在估计做好这个软件需要多长时间。



更多需求

- 然后你发现全国的老师都有这个需求
- 甚至有好些家长都想要这个
 - 。 。 。
- 校长希望你写一个网站满足大家的需求，希望能接受上万个用户同时访问
- 现在估计完成这一软件服务需要的时间



程序 vs. 软件

- Program = data structure + algorithm
- Software = Program + Software Engineering
- Software Company =
Software + Business Model

程序有道德么？ 软件公司是否有道德？



当出问题的时候...

■ Kid's toy （小孩玩具坏了）

- get a new one

■ Hobbyist （业余时间搞的程序崩溃了）

- we had fun, move on...

■ Early prototype （创业的软件失败了）

- learn the lesson, find another way, and keep going...

■ Professionals （专业软件崩溃了）

- 股票交易软件，火车票购票网站，
- 四则运算练习网站（有一万个学生每天要在这里做作业）
- people's life, money



Quiz:

If a feature in a commercial product has 1-in-a-million chance of usage, do you want to build it and educate customer about it every time they use the product?

- A. don't even plan it.
- B. cut it if we don't have time.
- C. Build it, but don't need to tell users
- D. Build it, tell uses about it

The answer is:





软件的概念

软件

software

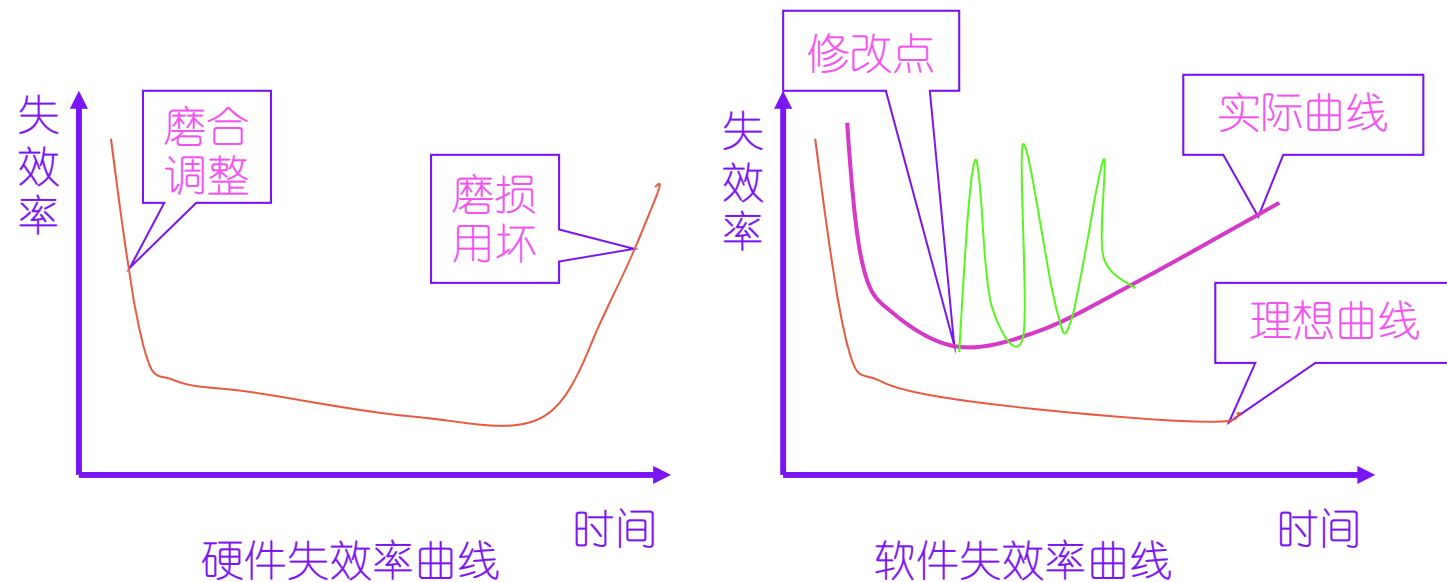
soft+ware

软制品
(软体)

软件是计算机系统中与硬件相互依存的另一部分。
它包括程序、数据及其相关文档的完整集合。

软件特点

- 软件是一种逻辑实体，而不是具体的物理实体
- 软件的生产与硬件不同
- 在软件的运行和使用期间，没有硬件那样的机械磨损，老化问题






软件的特性

- 复杂性
- 不可见性
- 易变性
- 服从性
- 非连续性
- 软件开发会越来越容易么？
- **No Silver Bullet / 没有银弹**
 - 没有一种大规模提高软件开发效率的快速办法，将来也没有



复杂性

- 软件可以说是人类创造的最复杂的系统类型。
- 而软件工程师通常一次只能看到30—80行源代码（相当于显示器的一屏），他们的智力、记忆力和常人差不多。软件的各个模块之间有各种显性或隐性的依赖关系，随着系统的成长和模块的增多，这些关系的数量往往以几何级数的速度增长。



不可见性（Invisibility）

- 软件工程师能直接看见源代码，但是源代码不是软件本身。软件以机器码的形式高速运行，还可能在几个CPU核上同时运行，工程师是“看”不到自己的源代码如何具体地在用户的机器上被执行的。
- 商用软件出现了错误，工程师可以看到程序在出错的一瞬间留下的一些痕迹（错误代号、大致的目标代码位置、错误信息），但是几乎无法完整重现到底程序出现了什么问题。
- AI 模型的问题，我们如何去验证呢？



易变性（Changeability）

- 易变性（Changeability） 软件看上去很容易修改，修改软件比修改硬件容易多了。人们自然地期待软件能在下面两种情况下“改变”：

- a) 让软件做新的事情；
- b) 让软件适应新的硬件。

但是与此同时，正确地修改软件是一件很困难的事情。



服从性（Conformity）

- 服从性（Conformity） 软件不能独立存在，它总是要运行在硬件上面：
 - a) 它要服从系统中其他组成部分的要求；
 - b) 它还要服从用户的要求、行业系统的要求（例如银行利率的变化）。



非连续性（Discontinuity）

- 人们比较容易理解连续的系统：增加输入，就能看到相应输出的增加。
- 但是许多软件系统却没有这样的特性，有时输入上很小的变化，会引起输出上极大的变化。



软件的其他特性

新特性：

- 有许多不同的程序设计语言
- 软件工具和软件开发平台
- 存在许多不同的软件开发流程
- 软件团队中存在许多不同的角色
- 软件通常既可以存储在磁带上，也可以存储在CD/DVD上
- AI 可以写程序， GitHub 社区

但是这些非本质、临时的特性并不能决定软件工程的本质问题。例如，有人发明了一种新的程序设计语言，或者又出现了一个新的软件开发流程，或者网上出现了又一个程序员技术社区……这些事并不能改变软件工程的根本难度，这也是著名的“没有银弹（No Silver Bullet）”论断所阐述的道理。

User's Perspective

- 小汽车
 - QQ
 - Hyundai (现代)
 - BMW
- 新车出厂, 它们都质检合格
- 它们的质量有区别么?
- 但是它们找到了各自合适的顾客





软件分类

1、按软件的功能进行划分

系统
软件

应用
软件

支撑
软件



支撑软件

一般类型： 文本编辑程序 文本格式化程序	支持需求分析： PSL/PSA 问题描述语言 关系数据库管理系统
支持设计： 图形软件包 结构化流程图绘图程序	支持测试： 静态分析器 测试覆盖检验程序
支持实现： 编辑程序 连接编辑程序	支持管理： 标准检验程序 库管理程序

2、按软件的规模进行划分

按开发软件所需的人力、时间以及完成的源代码行数。

类别	参加人数	研制期限	产品规模(源代码行数)
微型	1	1-4周	约500行
小型	1	1-6周	约2000行
中型	2-5	1-2年	5000-50000行
大型	5-20	2-3年	5万-10万行
甚大型	100-1000	4-5年	100万行
极大型	2000-5000	5-10年	1000万行



微软企业文化宗旨部分内容

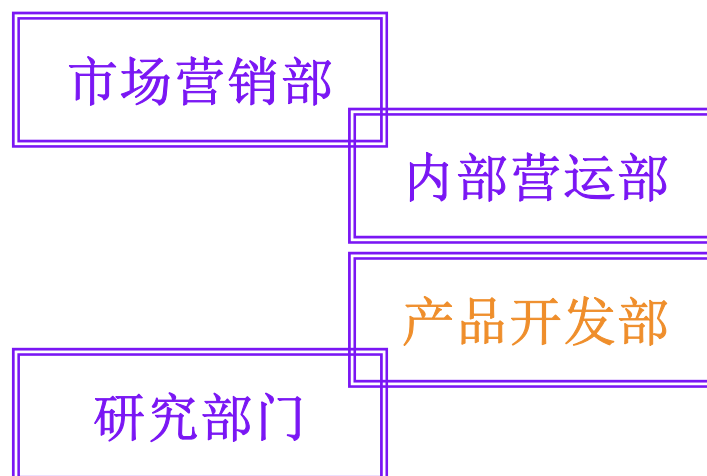
Wake up every day with a feeling of passion
for the different technology will make in people's
Life.

每天醒来的时候，要对技术给生活造成的改变
始终拥有一份 激情。



微软的软件产品开发过程

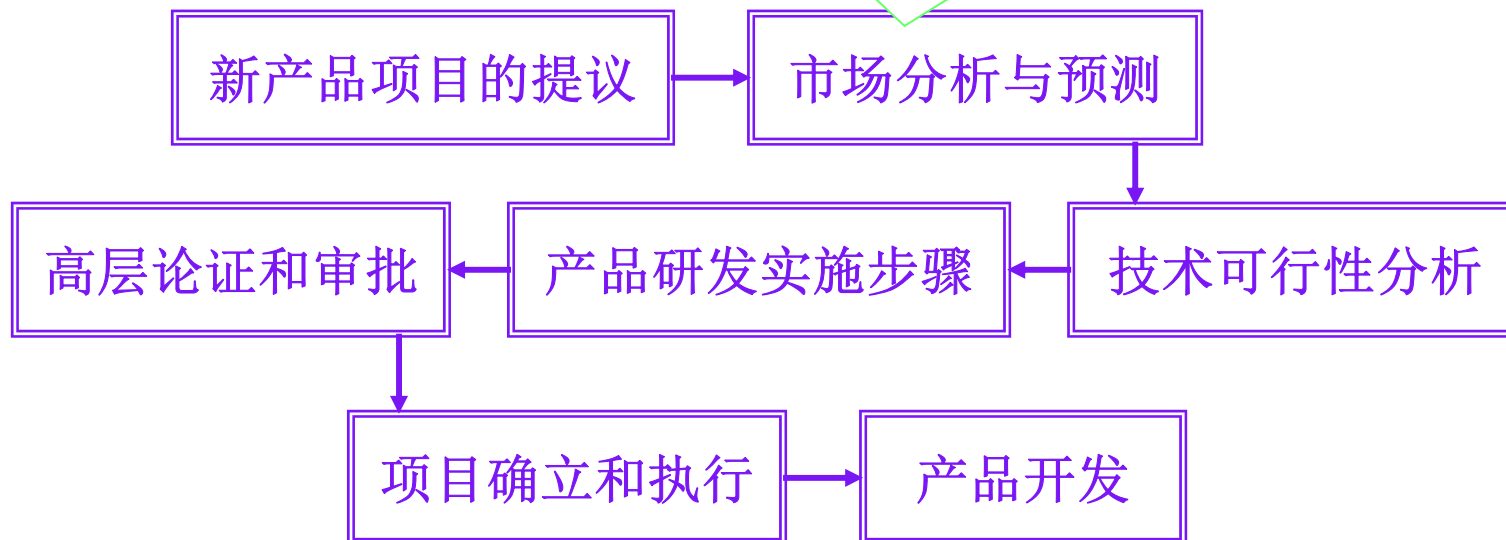
一、微软的组织结构



二、新产品的产生过程

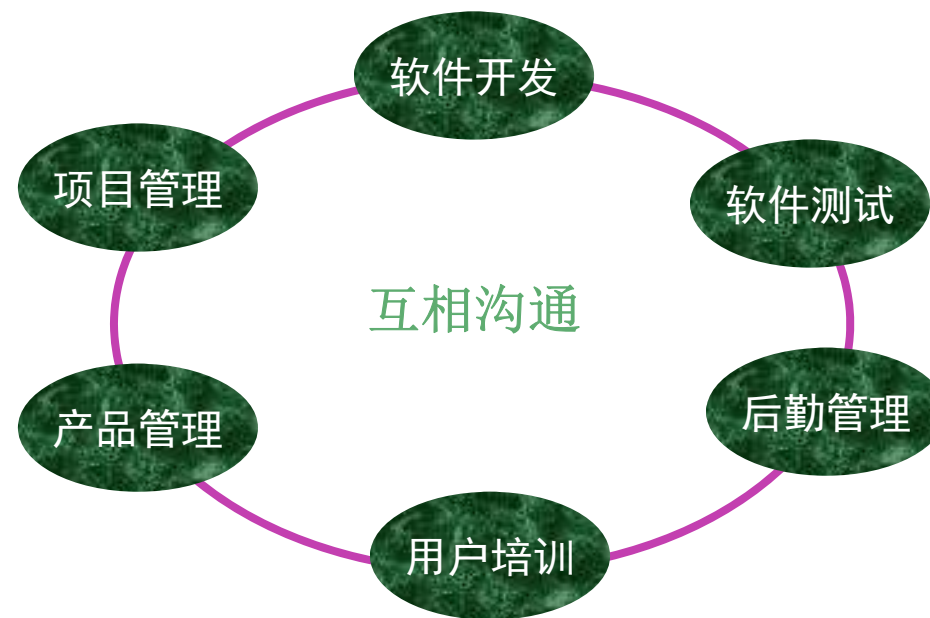
我最大的心得是，一个产品一定要找到能够真正适用的场合，不能只是为了技术而从事技术，为了研究而从事研究，却不管用户对你所研究的技术和产品有没有需求。否则，无论你的技术是多么优秀、多么先进，恐怕你的产品在市场上都无法获得成功。

----- 微软张益肇





三、微软的产品团队



----- 微软产品团队的组成

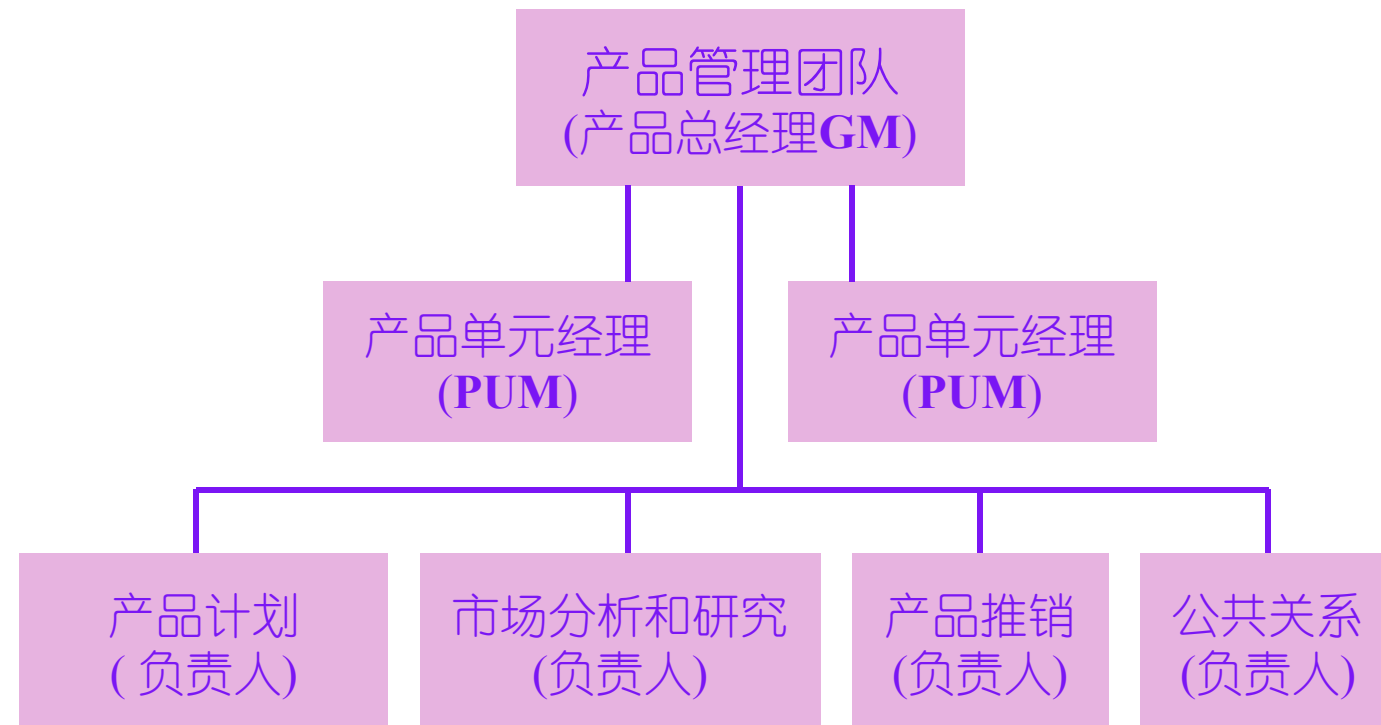


各个团队的角色及主要目标

团队名称	分担的任务
产品管理	确定产品的远景，获取并确定用户的需求，开发并维护商业安全，满足用户的需求
项目管理	制定开发功能规范，在团队内进行沟通和协商，维持产品进度并报告产品状态，保证能够尽快尽好地在产品约束条件下发布产品
软件开发	开发出满足设计规范和用户要求的产品
软件测试	开发测试策略和计划，保证在解决了所有已知问题后再发布产品
用户培训	保证使用文档要全部很清楚地写出来，提高用户使用产品的技能，保证大多数用户都能够充分利用产品的功能
后勤管理	保证产品能够平稳地发展

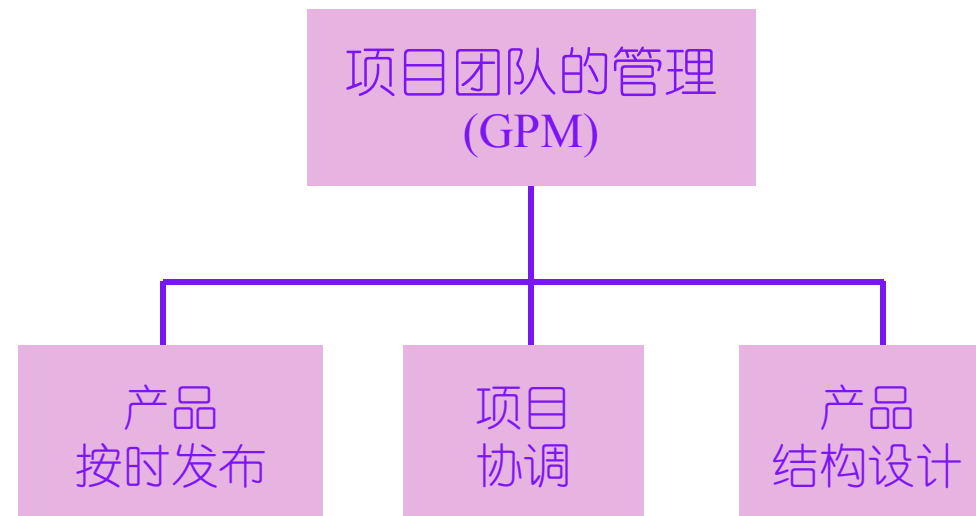


产品管理团队的组织机构 Product Management Team



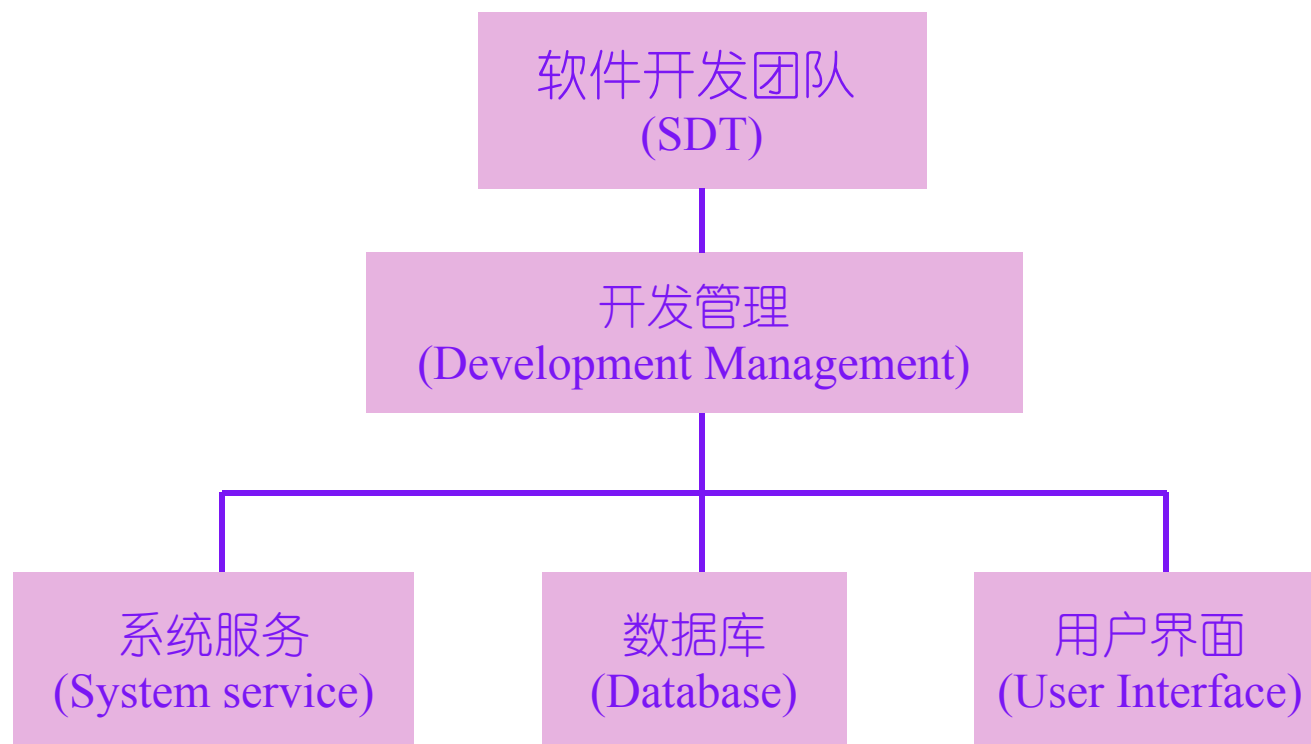


项目管理团队的组织机构 Program Management Team



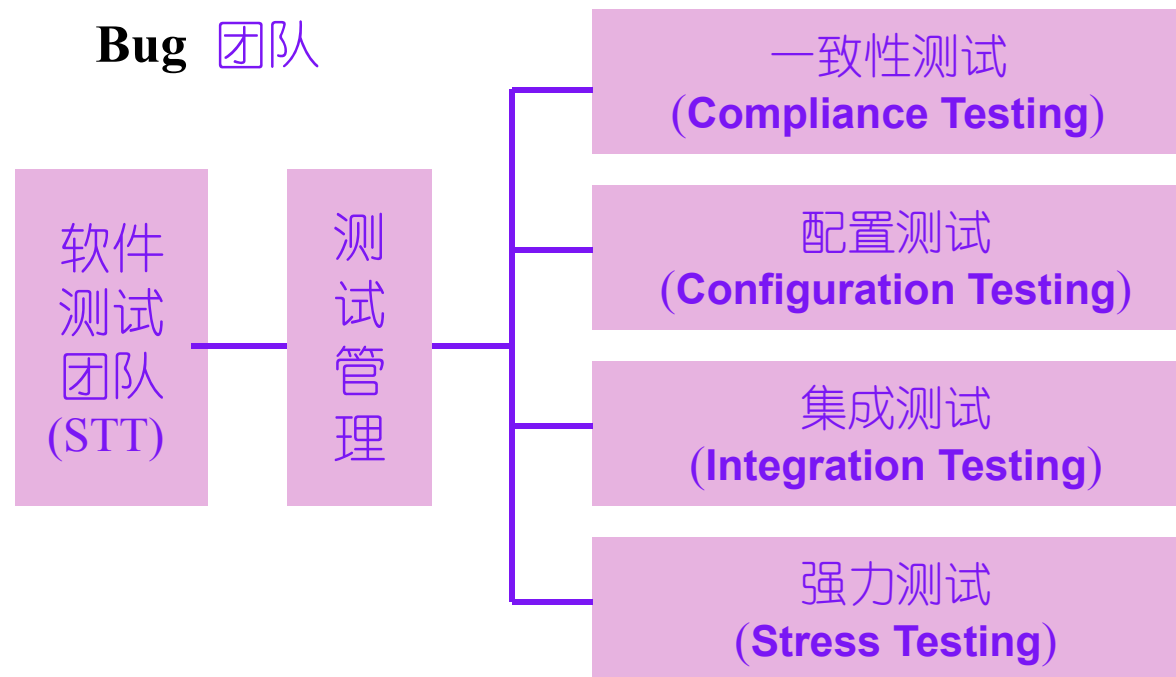


软件开发团队的组织机构 Software Development Team





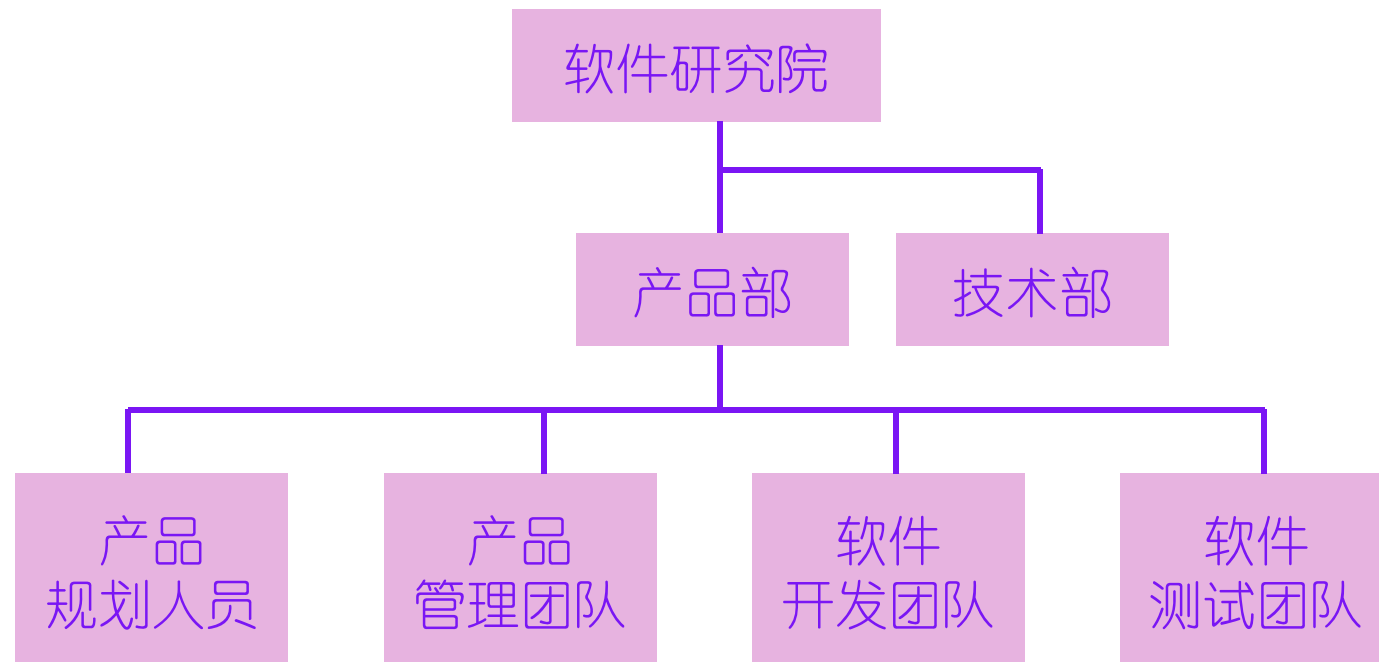
软件测试团队的组织机构 Software Testing Team





Exchange 2000 和Windows 2000 中的人员结构

	Exchange 2000	Windows 2000
项目经理	25人	约 250人
开发人员	140人	约1700人
测试人员	350人	约3200人
测试人员/开发人员	2.5	1.9



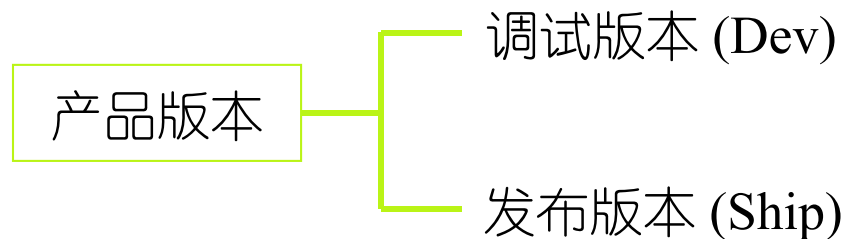
----- 微软现代软件开发的典型体系



四、微软的软件开发特点

管理上

- (1) 文档齐全，项目规范清楚
- (2) 开发人员相互阅读其他人新编写代码
- (3) 所有代码都有清楚的注释





软件开发第一阶段各团队分担的任务

团队名称	分担的任务
产品管理 项目管理 软件开发	分析是否应该做这个新产品，尽量证明该产品是值得做 设计新产品的目标，以及具体的实现方法 开发一些技术原型，检验新产品是否有意义，并向大家 展示新产品未来的样子。另外还要对开发过程中一些大的 结论提出意见
软件测试 用户培训 后勤管理	判断该新产品是否有用、是否可接受 分析用户的需求 对长期的支持管理提出建议



软件开发第二阶段各团队分担的任务

团队名称	分担的任务
产品管理	概念设计和市场推销计划 / 进度表
项目管理	逻辑设计、功能规范，以及总体计划 / 进度表
软件开发	物理设计和开发计划 / 进度表
软件测试	设计评估和测试计划 / 进度表
用户培训	用户性能支持设计和用户培训计划 / 进度表
后勤管理	对长期的支持管理提出建议

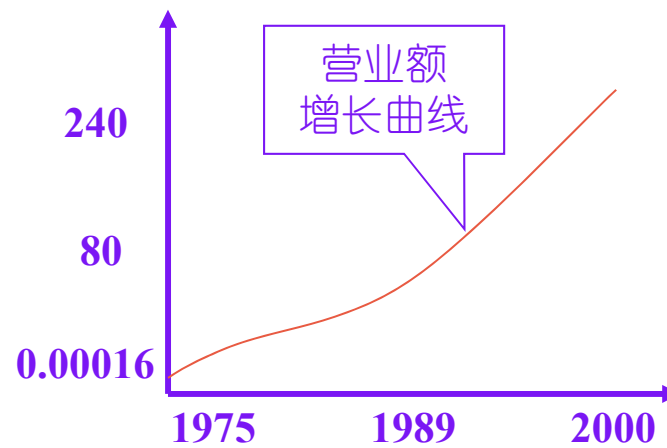


3、按软件开发划分

软件项目开发

软件产品开发

3、软件的演变（即从传统软件开发到现代软件开发）





1.2 软件工程的观念

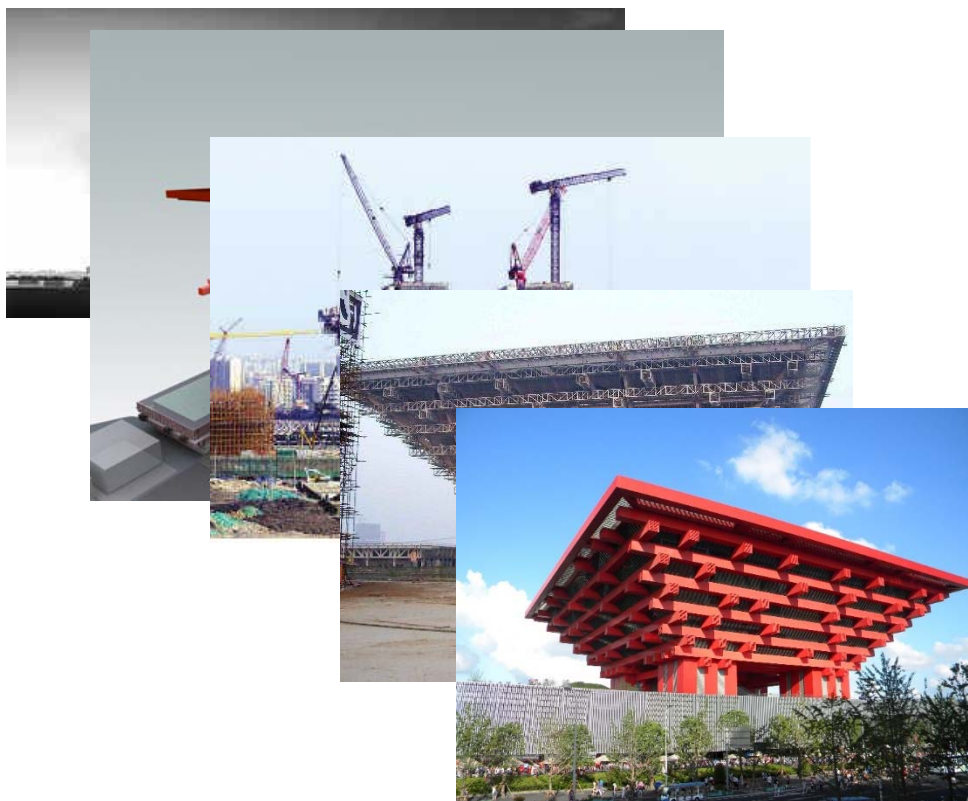
1、“软件工程”----Software Engineering

于1968年 NATO 组织在
德国召开的一次会议上提出

是把软件当作一种工业产品，要求 “采用工程化的
原理与方法对软件进行计划、开发和维护 ”。

“工程” 在各种行业都有

- 构想
- 分析
- 建设
- 交付
- 运行





软件工程和计算机科学的关系

- 计算机科学中的理论
 - 大多可以从形式上证明，和数学，离散数学，数理逻辑密切相关
- 计算机科学的实践
 - 和数据，以及其它学科结合，把各种理论运用在“计算和通讯”的场景中
- 软件工程
 - 和人的行为，现实的需求息息相关。软件工程的研究目标（软件的开发, 运营, 和维护）都有“人”出现
- 人
 - 项目需求的提供者，
 - 软件的开发人员，
 - 软件的用户



计算机的科学 vs. 工程

Science

- long-term
- idealism
- certainty
- perfection
- generality
- separation
- unification
- originality
- formality
- correctness

Engineering

- short-term
- compromise
- risk
- adequacy
- specificity
- amalgamation
- diversity
- best practice
- intuition
- dependability



不同侧重点

表 1-2 计算机科学和软件工程的侧重点

计算机科学	软件工程
发现和研究长期的、客观的真理	短期的实际结果 (具体的软件会过时)
理想化的	对各种因素的折衷
确定性，完美，通用性	对不确定性和风险的管理，足够好，具体的应用
各个学科独立深入研究，做出成果	关注和应用各个相关学科的知识，解决问题
理论的统一	百花齐放的实践方法
强调原创性	最好的、成熟的实践方法
形式化，追求简明的公式	在实践中建立起来的灵感和直觉
正确性	可靠性



软件工程的知识领域

- 软件工程的三大类基础知识领域：
 - 计算机基础；数学基础；工程基础。
- 软件工程这个学科包含了12个知识领域（Knowledge Area, KA）：
 - Software Requirements
 - Software Design
 - Software Construction
 - Software Testing
 - Software Maintenance
 - Software Configuration Management
 - Software Engineering Management
 - Software Engineering Process
 - Software Engineering Models and Methods
 - Software Quality
 - Software Engineering Professional Practice
 - Software Engineering Economics



2 软件危机

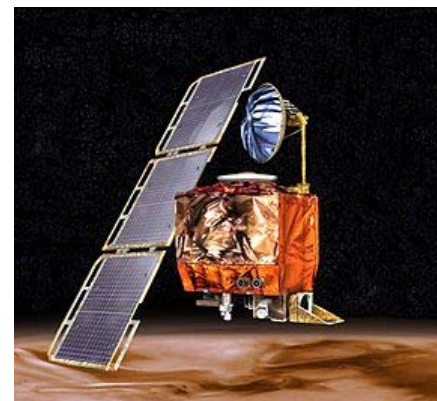
一、软件危机的产生

- ◆ 20世纪60年代中期之后，随着大型软件开发的需求不断增加，而软件技术的发展跟不上实际需求，造成大型软件开发过程中的三大难题：
复杂程度高、研制周期长、正确性难以保证三大难题。这些堆积的问题由于没有找到有效的解决办法，形成了人们难以控制的局面，出现了所谓的“软件危机”。
- ◆ 包括两方面的问题
 - 如何开发软件，以满足对软件日益增长的需求
 - 如何维护数量不断膨胀的已有软件

例子 - 火星气候探测者号

1999年“火星气候卫星”事故

- 总成本是3亿2,760万美元
 - 19,310万美元来研发
 - 9,170万美元来发射
 - 4,280万美元来进行探测任务
- 1998/12/12 发射
- 十个月之后，1999年9月23日，卫星抵达火星大气层，错误的导航参数造成卫星坠入大气层烧毁





事故的调查

1. 软件外包公司对于 **mission-critical** 的软件模块有很完备的检查和测试，但是对于其他模块则没有完备的管理
 2. 程序员写了一个不重要的**log** 功能，其中用英制 (磅* 英尺) 表示力，但是 **NASA** 用 “牛顿” = 千克*米/(秒*秒)
 3. 外包公司接到一个新的工程，他们进行了软件重用，**log** 功能中记录的力被重用为导航功能的输入参数，成为 **mission-critical** 的模块。
 4. 这个新的工程由发包公司 **Lockheed** (洛克希德公司) 交给了客户 **JPL** (喷气推进实验室)
 5. 火箭带着卫星发射了，在10个月的飞行中，**JPL** 可以每天两次启动小推进器，来调整太空船的航向，在这一过程中，有人发现了导航功能的一些不正常现象，于是 -
 1. **JPL** 发邮件给 **Lockheed**, 说 – 这个模块有些参数看起来好像不正常...
 2. **Lockheed** 回邮件...
 3. **JPL** 再发邮件...
 4. 最后没有人再发邮件了
 6. 后来, **JPL**的同志们认为, **Lockheed** 的同志们估计已经搞定了。 **Lockheed** 的同志们认为, **JPL** 的同志们没再追问这个问题，可能已经不是问题了
- 十个月之后，卫星抵达火星大气层，错误的导航参数造成卫星坠入大气层烧毁。



错误根源？

■ 错误1:

- 一个模块从 non-mission-critical 变成 mission-critical 没有经历必要的复审和测试。

■ 错误2:

- 这个问题从来没有收录到NASA 的错误跟踪系统 (Bug tracking system), 只是在email 中交流, 导致最后没有人对这个问题负责。在错误跟踪系统中, 总得有一个人“拥有”这一个bug, 这样可以避免推诿责任。



我们碰到的bug

- 软件的行为和用户的期望值不一样
 - 崩溃**Crash**, 死锁
 - 数据丢失, 错误
 - 达不到预期功能
 - 难用
- 期望值从哪里来?
 - 用户/顾客/产品/文化



Bug? 取决于你看问题的角度

- 伙计取下壁上挂的一块乌黑油腻的东西，请他们赏鉴，嘴里连说：“好味道！”引得自己口水要流，生怕经这几位客人的馋眼睛一看，肥肉会减瘦了。肉上一条蛆虫从腻睡里惊醒，...
- 伙计忙伸指头按着这嫩肥软白的东西，轻轻一捺，在肉面的尘垢上划了一条乌光油润的痕迹，像新浇的柏油路，一壁说：“没有什么呀！”顾尔谦冒火，连声质问他：“难道我们眼睛是瞎的？”大家也说：“岂有此理！”...
- 肉里另有两条蛆也闻声探头出现。伙计再没法毁尸灭迹，只反复说：“你们不吃，有人要吃——我吃给你们看——”店主拔出嘴里的旱烟筒，劝告道：“这不是虫呀，没有关系的，这叫‘肉芽’——‘肉’——‘芽’。”方鸿渐引申说：“你们这店里吃的东西都会发芽，不但是肉。”
- Bug? Or 肉芽?



二、软件危机的定义及其表现形式

软件危机是指在计算机软件的开发与维护过程中所遇到的一系列严重问题。主要表现在：

- (1) 对软件开发成本与进度估计不准确
- (2) 软件开发人员与用户交流不充分
- (3) 软件质量不佳
- (4) 软件缺乏适当的文档资料
- (5) 软件成本不断日益增长
- (6) 软件开发生产率提高的速度跟不上硬件的发展速度



三、解决软件危机的途径

(1) 应该对计算机软件有一个正确的认识

1983年，IEEE对软件的定义：计算机程序、方法、规则、相关的文档资料以及在计算机上运行程序所必需的数据。

(2) 充分认识软件开发是一种组织良好、管理严密、各类人员协同配合、共同完成的工程项目

(3) 推广使用实践中开发软件的成功技术和方法

(4) 开发和使用更好的软件工具



3 软件工程的定义

- ◆ Fritz Bauer为软件工程给出如下定义：“软件工程为了经济地获得可靠的且能在实际机器上有效地运行的软件，而建立和使用的完善的工程化原则。”
- ◆ 1983年，IEEE对软件工程所下的定义是：“软件工程是开发、运行、维护和修复软件的系统方法。”
- ◆ 1993年，IEEE给出了一个更全面的定义：
软件工程是：①把系统化的、规范的、可度量的途径应用于软件开发、运行和维护的过程，也就是把工程化应用于软件中；②研究①中提到的途径。



4 软件工程的基本原理

1983年，B. W. Boehm提出了软件工程的7条基本原理：

- (1) 用分阶段的[生命周期计划](#)严格管理
- (2) 坚持进行阶段[评审](#)
- (3) 实行严格的产品控制
- (4) 采用[现代程序设计技术](#)
- (5) 结果应能清楚地[审查](#)
- (6) 开发小组的人员应该[少而精](#)
- (7) 承认不断[改进](#)软件工程实践的必要性

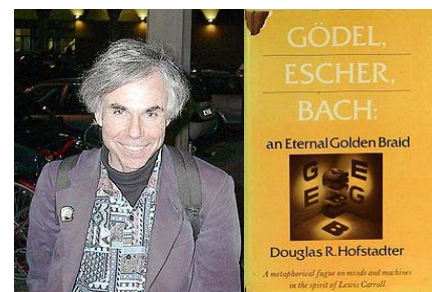


软件工程的规律

- 没有银弹：“不会有任何单一软件工程上的突破，能够让程序开发的生产力得到一个数量级（**10倍**）的提升。”
- 如果多种软件工程上的突破结合起来，能否让软件开发的效率得到**10倍**的提高呢？

软件工程的规律

- 时间估计霍夫斯塔特定律：
 - 实际时间总是比预期要长，即便你考虑到了霍夫斯塔特定律。
 - 这个规律如何应用到实际中呢？
- 增加人手：
 - 向进度落后的项目中增加人员，会让项目更加落后。
 - 有数量的关系么？
 - 如果是增加专家呢？
 - 搞 996 呢？





规律...

- 发展于1980年代的Cocomo 模型，认为某种项目的时间花费（人月）遵守这样的公式：
 - **$\text{Person*Month} = 2.4 * \text{KLoC}^{1.05}$**
- 其中，KLoC 表示一千行代码，例如，如果估计的代码量是10万行（KLoC =100），那么时间花费就是 $2.4*100^{1.05} = 302$ 人月。
- 如果你的团队有15人，那就要花20个月的时间。在2018年的实际项目中，项目经理真的会用 2.4，和1.05 来进行计算么？

规律?...





5 软件工程学

- ◆ 软件工程包括技术和**管理**两方面的内容，是管理与技术的紧密结合。通常把在软件生命周期全过程中使用的一整套技术的集合称为方法学。
- ◆ 软件工程方法学包括三个要素：**方法**、**工具**和**过程**。
 - **方法**：完成软件开发的各项任务的技术方法，“如何做”，例如，项目计划与估算、软件系统需求分析、数据结构、系统总体结构的设计、算法的设计、编码、测试以及维护等；
 - **工具**：为方法的运用提供自动或半自动的软件支撑环境；
 - **过程**：为了获得高质量的软件所需要完成的一系列任务的框架



5 软件工程学

广泛使用的有：结构化方法学和面向对象方法学

- 结构化方法学：把软件生命周期全过程依次划分为若干个阶段，顺序地逐步完成每个阶段的任务，可大大提高软件开发的成功率。
- 缺点：
 - 当软件规模较大时，或者对软件的需求是模糊或随时间变化时，往往不成功
 - 维护较困难
- 原因：将数据和处理人为地分离成两个独立的部分，增加了软件开发与维护的难度



5 软件工程学

面向对象方法=对象+类+继承+用消息通信

- 对象：作为融合了数据及其操作的软件构件
- 类：对具有相同数据和操作的一组相似对象的定义
- 继承：在类等级中，下层派生类自动拥有上层基类的数据和操作
- 消息：对象之间仅能通过发送消息互相联系

优点：

简化了软件的开发和维护工作，提高了软件的可重用性

- 适合开发互联网时代的系统软件和应用软件
- 编程语言为Java、C++、PowerBuilder、Delphi、VB等



1.3 软件生存周期 (SW life cycle)

把软件从产生、发展到成熟、直至衰亡为止

软件生命周期由：
软件定义
软件开发
软件维护 三个阶段组成。



软件生存周期定义

- 定义 (Definition)

- 指软件产品从提出开发要求开始，经过开发、使用和维护，直到最后不再能够使用的全过程，包括软件定义、软件开发、运行维护三个时期。

- 各时期的任务

- 软件定义：

- 确定总目标、可行性、系统功能、项目成本及进度
- 包括问题定义、可行性研究、需求分析三阶段

- 软件开发：

- 具体设计和实现所定义的软件
- 包括概要设计、详细设计、编码和单元测试、集成测试四阶段

- 运行维护：

- 使软件持久地满足用户的需要



各阶段的基本任务

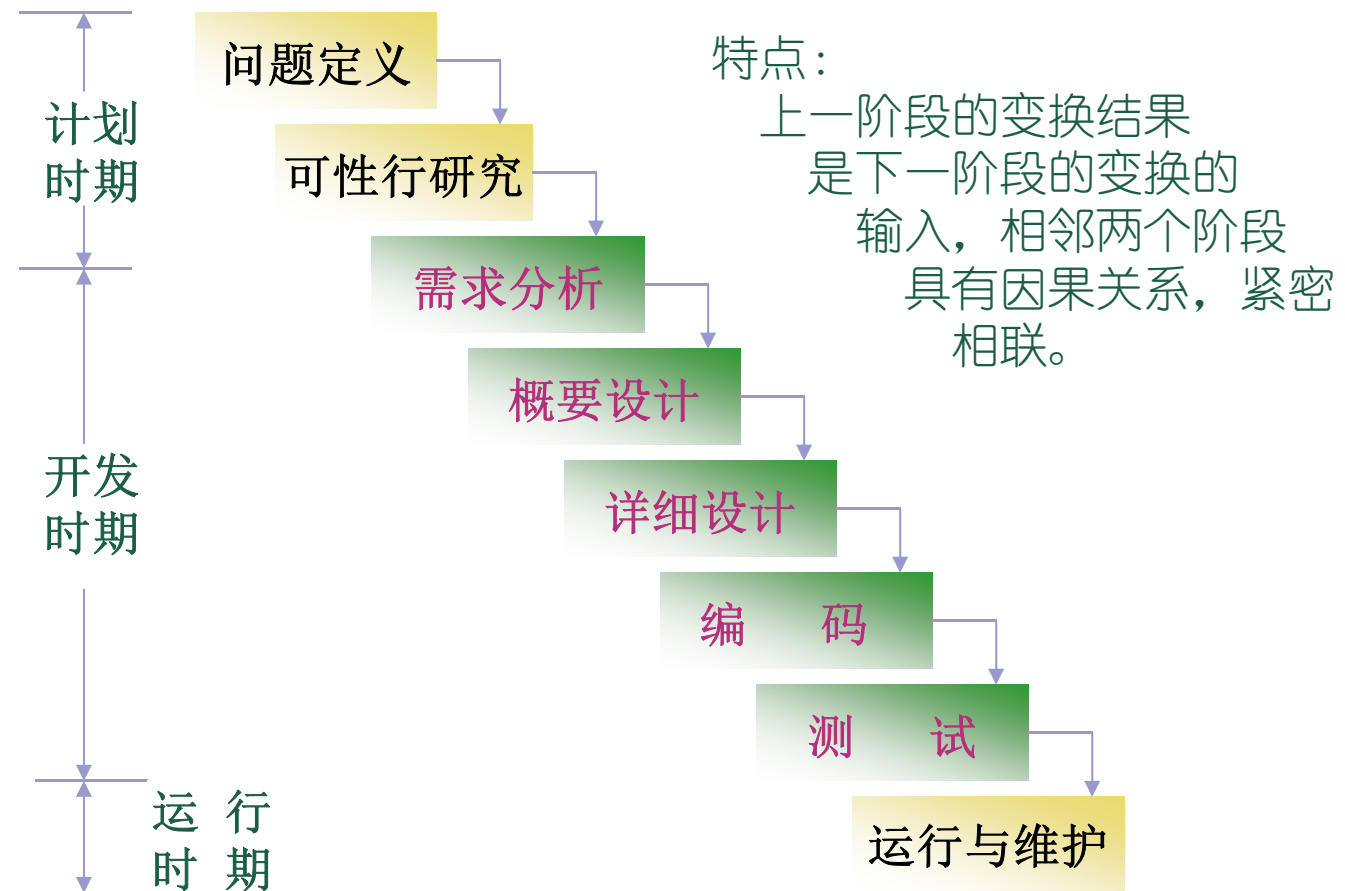
- 问题定义 (Definition)
 - 是什么 ???
- 可行性研究 (Feasibility Research)
 - 是否有切实可行的方法 ??? (从技术、经济、社会因素等方面分析) 【可行性报告】
- 需求分析 (Requirement Analysis)
 - 做什么 ??? (对目标系统提出完整、准确、清晰、具体的要求) 【需求规格说明书 SRS】



各阶段的基本任务

- 软件设计 (Software Design)
 - 怎么做 ??? 【软件设计说明书】
- 编码和单元测试 (Coding & Unit Testing)
 - 写程序模块，完成系统 【源程序清单】
- 集成测试 (Integrated Testing)
 - 质量保证，包括集成测试和验收测试 【测试报告】
- 运行/维护 (Running/Maintenance)
 - 安装后的进一步完善，包括：改正性维护、适应性维护、完善性维护、预防性维护 【维护文档】
 - 实质上是经历了一次压缩和简化了的软件定义和开发的全过程

软件生存周期模型（瀑布模型 Waterfall Model）





One Example -制造汽车

- 你(用户) 提出要发动机, 车身, 车窗, 方向盘, 加速踏板, 刹车, 手刹, 座位, 车灯
- 生产商按照瀑布模型流程给你生产, 六个月后交付。
- 看到样车后...
- 你提出 – 我当初忘了一件小事, 要有倒车灯
 - 当倒车的时候, 倒车灯会亮
- 生产商说:
 - 我要重新设计车尾部, 加上倒车灯, 把车底拆开, 安装线路, 修改传动装置把倒车档和倒车灯联系起来。。。我得重新开始
- 你说: 这不是很小的一件事么?



瀑布模型

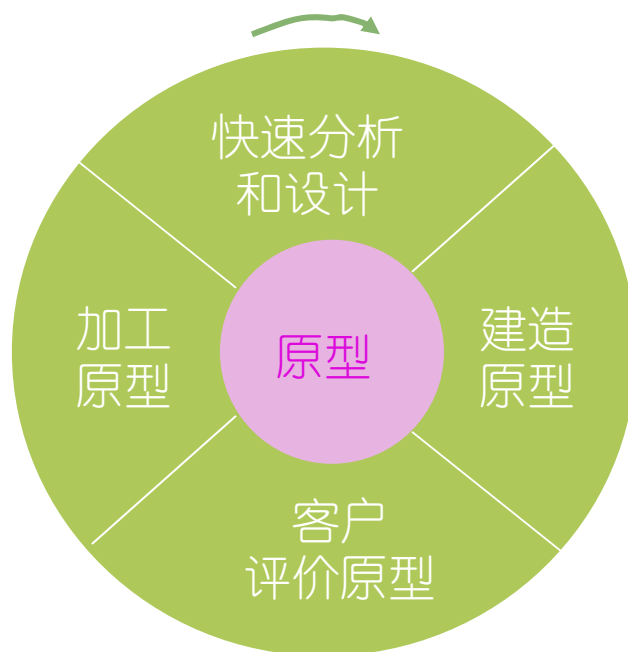
- 特点：
 - 相邻阶段间具有顺序性和依赖性
 - 推迟实现的观点
 - 质量保证的观点（认真做好各阶段的文档和评审工作）
- 优点：
 - 迫使开发人员采用规范的方法；
 - 严格地规定了各阶段必须提交的文档；
 - 要求各阶段的产品必须经过质量验证。
- 不足：
 - 开发初期就需指明系统的全部需求
 - 开发期长，一旦修改，则损失惨重
 - 不支持软件复用、集成技术
- 适合于软件需求很明确的小型软件项目开发



原型模型 (Prototype Model)

原型：是指模拟某种产品的原始模型

墨刀：在线产品原型设计与协作平台 <https://modao.cc/>



- 1、原型系统仅包括未来系统的主要功能，以及系统的重要接口。
- 2、为了尽快向用户提供原型，开发原型系统时应尽量使用能缩短开发周期的语言和工具。



原型模型 (Prototype Model)

优点:

- 原型系统已通过与用户交互得到验证，据此产生的规格说明正确反映了用户需求，后续阶段不会因为发现了规格说明文档的错误而进行较大的返工
- 开发人员通过建立原型系统学到很多东西，设计和编码阶段发生错误可能性小
- 本质是“快速”，应尽可能快地建造原型系统，以加速软件开发过程，节约成本

适用:

- 客户和开发者必须承认原型是为定义需求服务的，必要的时候要丢弃原型，实际的软件系统必须以质量为目标

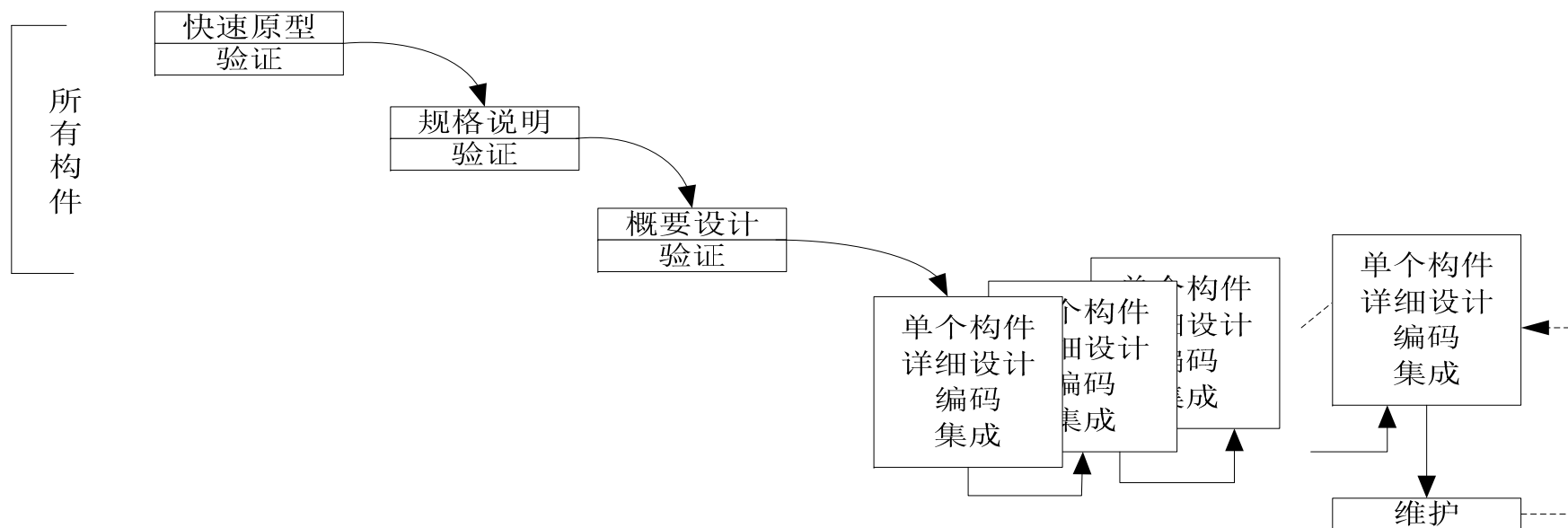


原型模型 (Prototype Model)

缺点:

- 当开发者告知用户整个系统需要重建以提高软件质量，客户可能会不愿意，觉得只需稍加修改即可，从而使得软件开发管理陷入失效；
- 开发人员为使原型快速运行起来，往往在实现过程中采用折衷的手段。比如使用不适当的语言或低效算法等，时间长了会适应这些选择，从而造成系统的不完善。

增量模型





增量模型

增量模型将软件产品作为一系列增量构件来设计、编码、集成和测试

- ① 开发人员一个构件接一个构件地向用户提供产品，每次用户都能得到一个满足部分需求的可运行产品，直到最后一次得到满足全部需求的产品
- ② 能在较短时间向用户提交可完成一些有用工作的产品
- ③ 逐步增加产品功能可以使用户有时间学习和适应新产品
- ④ 软件体系结构必须开放，便于增量构件的集成，可维护性明显好于封闭结构软件



增量模型

实例：字处理软件

- 第一个增量：提供基本的文件管理、编辑和文档生成功能；
- 第二个增量：提供复杂的编辑和文档生成功能；
- 第三个增量：提供拼写和语法检查功能；
- 第四个增量：提供高级页面排版功能。

特点：

- 第一个增量是核心产品，用户根据使用此产品进行评价，从而制订下一个增量计划；
- 类似快速原型模型，具有迭代的性质。但增量模型侧重每个增量都提交一个可操作的产品，早期增量是最终产品的片段版本。



增量模型

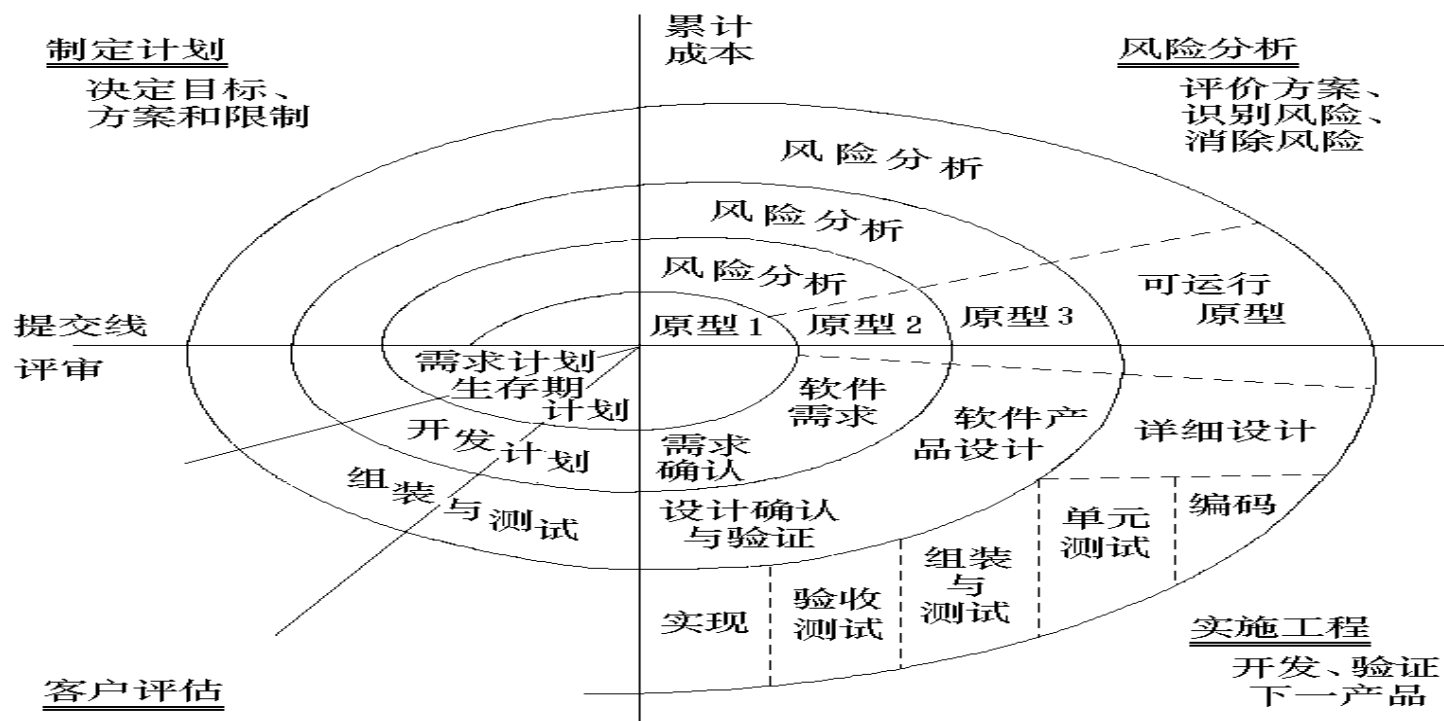
适用：

- 在项目既定的商业要求期限之前不可能找到足够的开发人员
- 可规避技术风险，例如，若一个系统需运用某个新型硬件，但不能确定此硬件的交付日期，可在早期避免使用这个硬件，这样可保证部分功能按时交付给最终用户，不至于造成延期

缺点：

- 要求软件具有开放式结构

螺旋模型



- 软件风险在不同程度上损害软件开发过程和软件产品质量，在软件过程中必须识别和分析风险，并且采取适当的措施消除或减少风险，例如：构造原型



螺旋模型

特点：

- 在软件过程的每个阶段之前都增加了风险分析过程的快速原型模型
- 在每次迭代中逐步完善，开发不同的软件版本
- 螺旋的每圈都会跨过策划区域，此时可调整项目计划，并根据交付后用户的反馈调整预算和进度，项目经理还调整完成软件开发需要迭代的次数

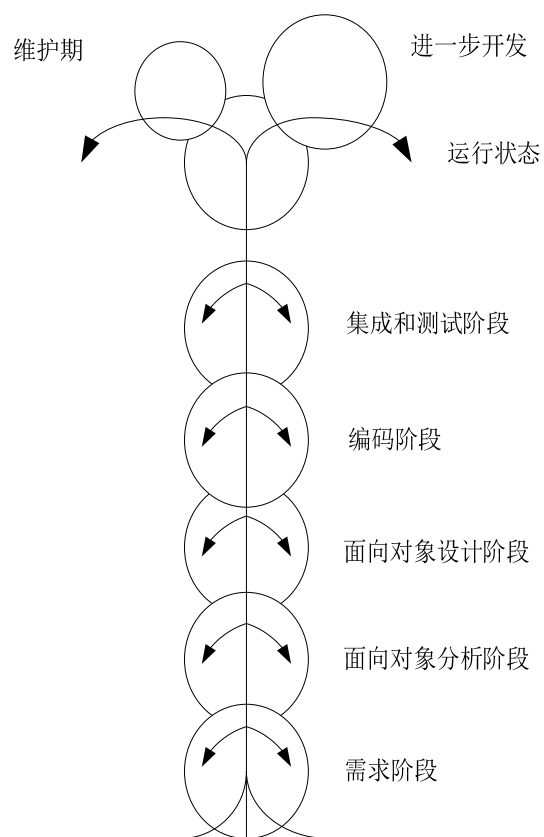
适用：

- 适用于开发大型软件和系统
- 项目的所有阶段始终考虑技术风险，如果适当应用该方法，可在风险变成问题之前化解风险

缺点：

- 以大量的风险评估专家来保证成功，如果较大的风险没有被发现和管理，肯定会发生问题

喷泉模型



- 软件过程各个阶段之间的迭代或一个阶段内各个工作步骤之间的迭代，在面向对象范型中比在结构化范型中更常见。



喷泉模型

特点:

- 连续两个活动之间存在迭代
- 用面向对象方法开发软件时，在分析、设计和编码等开发活动之间不存在明显边界，保证各项开发活动之间的无缝过渡
- 典型的面向对象生命周期模型



Team Software Process

- 优秀的模式和流程有什么共同点呢？
- Team Software Process (TSP) 的原则：
 - 1. 使用妥善定义的流程，流程中的每一步都是可以重复、可以衡量结果的；
 - 2. 团队的各个成员对团队的目标，角色，产品都有统一的理解；
 - 3. 尽量使用成熟的技术和做法；
 - 4. 尽量多地收集数据（也包括对团队不利的数据），并用数据来帮助团队做出理性的决定；
 - 5. 制定切合实际的计划和承诺，团队计划要由负责具体执行的的角色来制定（而不是从上级而来）；
 - 6. 增加团队的自我管理能力；
 - 7. 专注于提高质量，争取在软件生命周期的早期发现问题。最有效提高质量的办法是做全面而细致的设计工作（而不是在后期匆忙修复问题）。

思考：

- 你们小组的开发模式，流程有这些特点么？



小结

- 软件概念
- 软件特点
- 软件危机
- 软件工程概念
- 软件过程
- 软件生命周期模型