软件工程管理

C5 软件项目估算

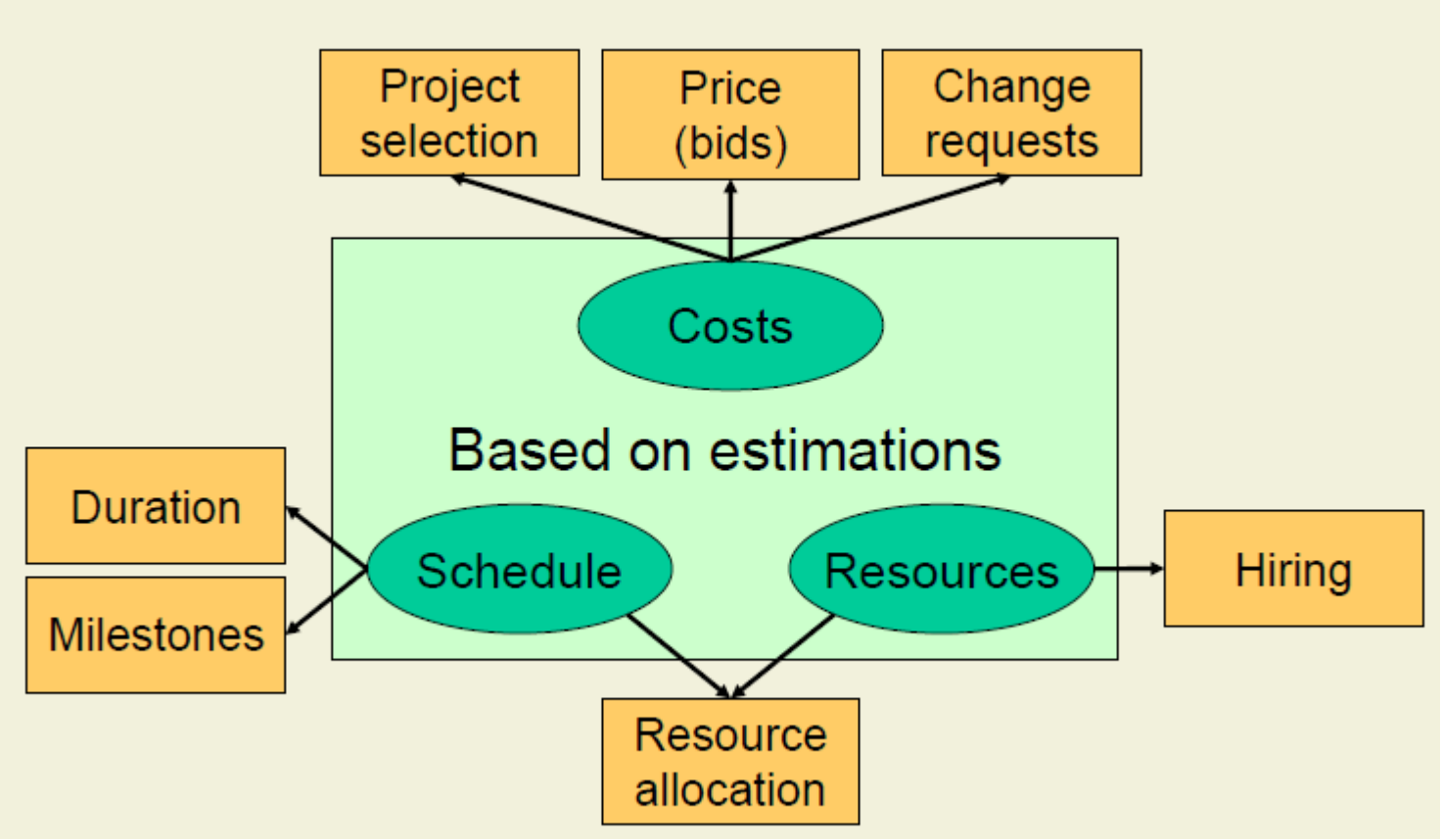
第一次课题联系反馈

- 第一次作业反馈

内容

- 估算概述
- 工作量估算方法
- 成本估算方法
- 估算表示

为什么做估算？


Project selection
Price (bids)
Change requests
Costs
Based on estimations
Duration
Milestones
Schedule
Resources
Hiring
Resource allocation

估算

- 经典错误之一：不现实的预期

- 一个项目最重要的任务是:设定现实的期望。
- 基于不准确估算的不切实际的期望是导致软件失败的一个根本原因。

Software Estimation Woes

- Estimation woes 1 - Estimates as wishful thinking
  - "When will my car be ready?"
  - "By tomorrow afternoon. Thursday morning for sure."
    - *Meaning: In theory, I could fix your car by tomorrow afternoon.* *This* implies:
      - ...that I can find supplies
      - ...that no urgent jobs come up
      - ...that I diagnosed your car's problem correctly
      - ...that I don't get lazy or sick tomorrow, and arrive here on time
      - ...that none of my tools breaks down
      - ...that there won't be weather problems, nor electrical blackouts
    - "And since I can't guarantee all of that, I'm giving myself a half-day buffer. Should be enough."
  - *Real meaning: It'll be ready in two months.*

Software Estimation Woes

- Estimation woes 2 - *Estimates as guessing games*
  - "Scotty, what's the problem with the brake?"
  - "It's broken, captain."
  - "How long will it take to fix it?"
  - "Seven hours. Maybe eight."
  - "Seven hours?! You got fifteen minutes."
  - "Yes sir."
    - *Meaning: I didn't really want an estimate. I wanted you to guess the* answer I was thinking of, you fool.
      - I assume your estimate was a bargaining chip. Maybe you're lazy and wanted to buy some procrastination time.
      - I can make you bend reality if I pressure you hard enough.
  - *Real meaning: It'll be fixed in eight hours. Maybe twenty*

Software Estimation Woes

- Estimation woes 3 - *Estimates as negotiation tools*
  - "Scotty, now what's the problem with the brake?"
  - "It's broken again, captain."
  - "How long will it take to fix it this time?"
  - "… ehem…Twelve days, captain. This one is hard."
  - "What?! That's insane! You got ten hours."
  - "OK sir."
    - *Meaning: I learned to play the game. Whatever I tell you you'll just cut it down irrationally. So I'll blow it up irrationally too.*
      - *NOW my estimate was a bargaining chip. I won't ever give a candid estimate anymore, thanks for the lesson.*
  - *Real meaning: It'll be fixed in eight hours. Maybe twenty*

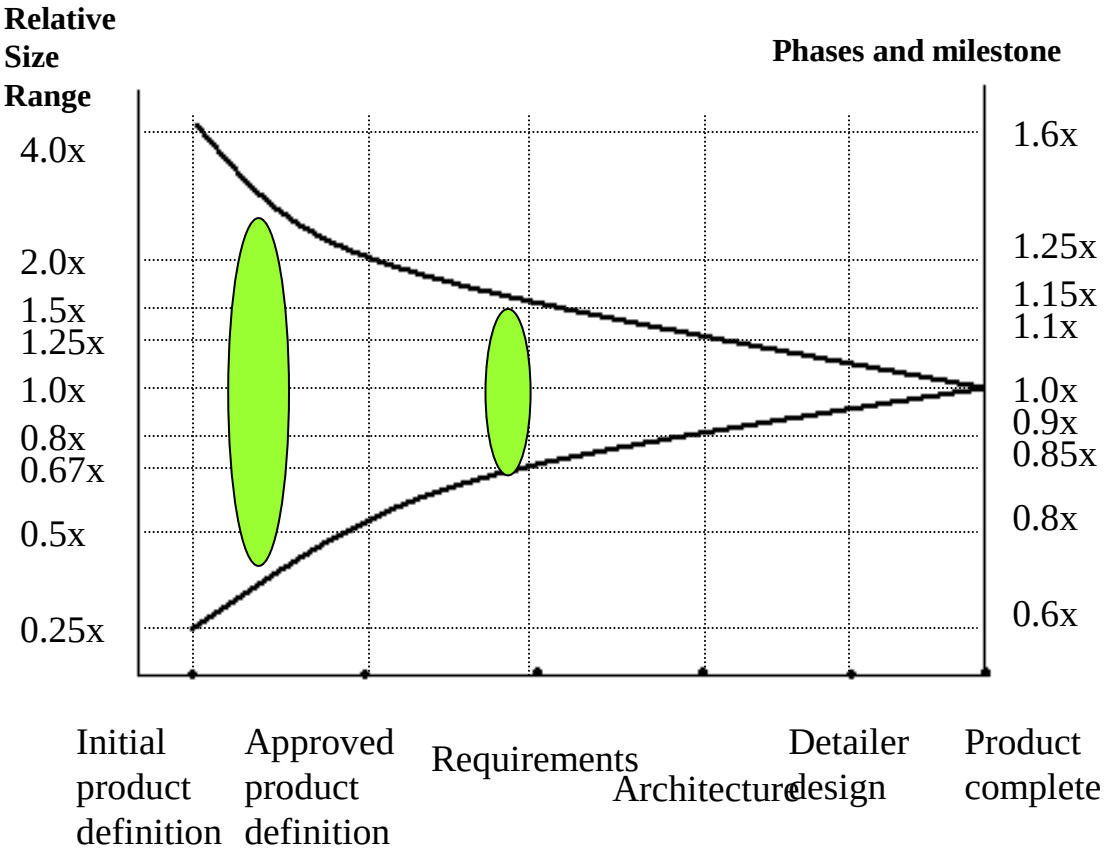
Software Estimation Woes

- Estimation woes 4 - Self-fulfilling prophecies
  - "Hmm, what do you know, the brake seems simpler than I thought this time. This one could actually be fixed in under four hours!"
    - So that means I have six extra hours. Let's see if I can also fix that rattling sound that's been bugging me. And I'll bring the new guy to train him on how to fix the warp drive. And…
    - Meaning: The captain said I had ten hours, so I'll use ten hours.
    - Parkinson's law: Work expands to fill the time available.
      - The reason why almost no project ends before its estimated time
  - Real meaning: It'll be *fixed in eight hours. Maybe twenty*

估算

- 尽管非常难，但是仍然需要做估算
- 只有准确地估算软件的功能，才能比较精确的估算出软件的成本，并制订出合理的进度计划；
- 软件估算对进度与成本控制非常重要！也就是说没有好的软件估算，项目的规划、跟踪和控制就根本无从谈起。
- 但是，一个“准确的估算”是不可能的!估算会有误差!
  - 软件系统的规模、成本只有在软件完成时候才能够准确知道结果。
- 软件估算是随着开发过程而逐渐细化改进的过程。

估算的不确定性



Estimation is a complex matter and can’t be precisely defined.
Usually, there are lots of changeable factors that impact the result.

估算的时机

- 估算进行的太早了，显然误差很大。
- 软件开发是一个逐步细化的过程，在每个阶段，都可能做出影响最终项目成本与进度的决策。
  - 可行性研究
  - 需求说明
  - 供应商和分包商评估
  - 项目计划(迭代)
  - 注意:并非所有这些步骤总是显式执行

估算常见问题

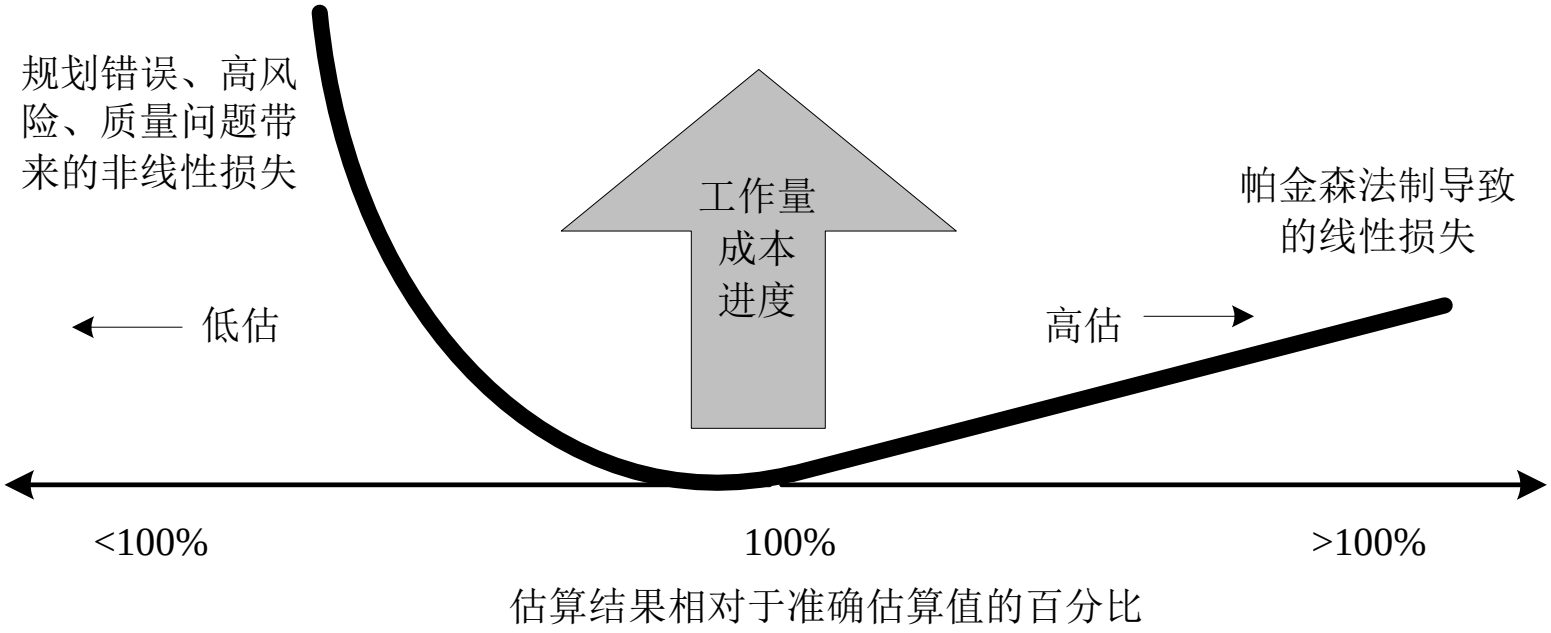
- 基于价格的估算
- 没有准备足够时间做估算
- 不完整或不正确的需求
- 对项目估算时缺乏数据支持，即无历史数据
- 没有考虑到其他重要项目因素
  - 开发环境的稳定性和可用性
  - 团队稳定性
  - 复用因素
- 估算的假设不成立——软件项目生产率与所施加的进度要求无关
- 开发者本身很乐观

项目	设计		编码		测试		合计		生产率
	工作月数	%	工作月数	%	工作月数	%	工作月数	SLOC	
1	3.9	23	5.3	32	7.4	44	16.7	6050	362
2	2.7	12	13.4	59	6.5	26	22.6	8363	370
3	3.5	11	26.8	83	1.9	6	32.2	13334	414
4	0.8	20	2.4	62	0.7	18	3.9	5942	1524
5	1.8	10	7.7	44	7.8	45	17.3	3315	192
6	19	28	29.7	44	19	28	67.7	38988	576
7	2.1	21	7.4	74	0.5	5	10.1	38614	3823
8	1.3	7	12.7	66	5.3	27	19.3	12762	661
9	8.5	14	22.7	38	28.2	47	59.5	26500	445
合计							249.3	153868	617

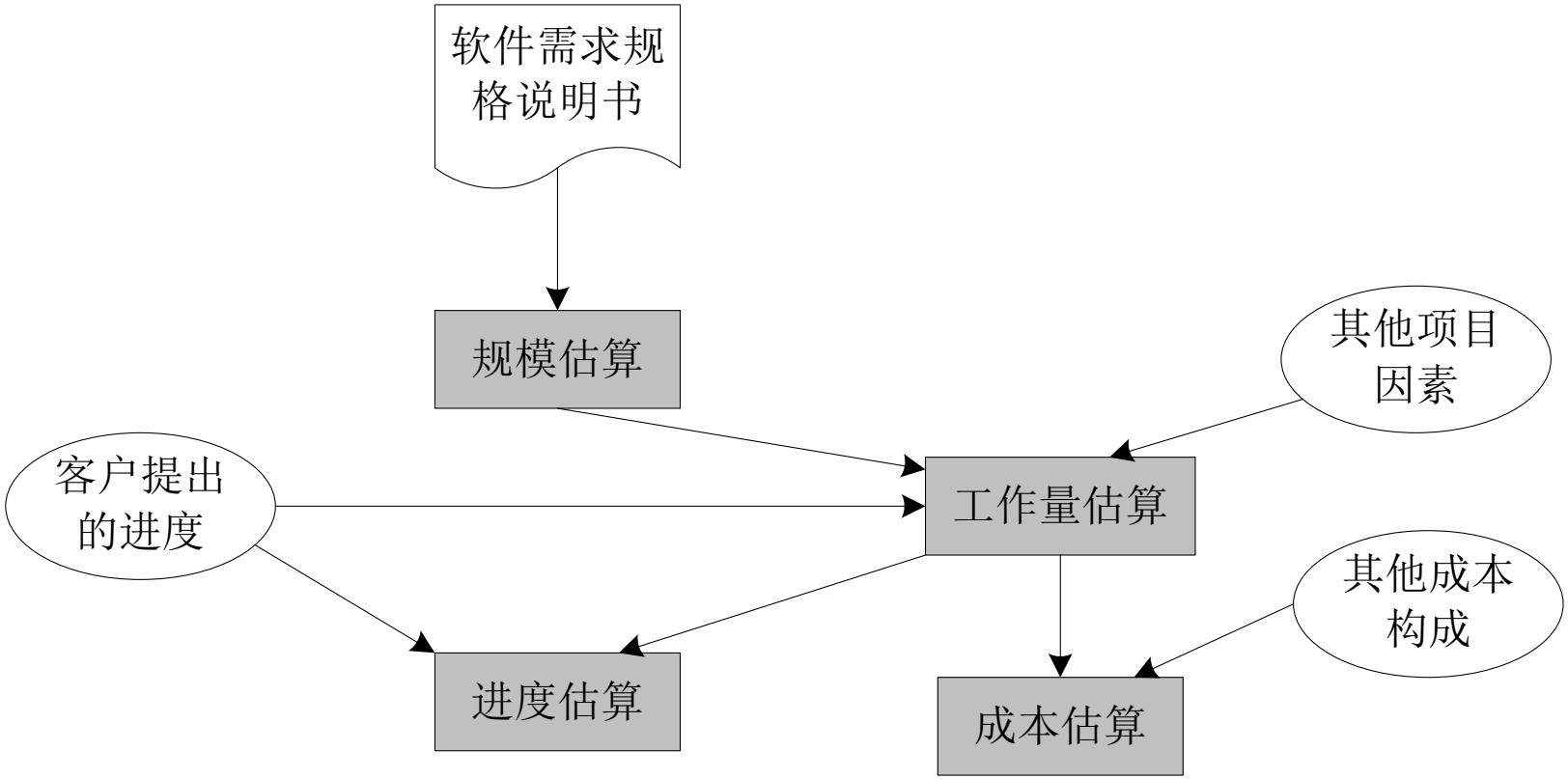
估算对结果的影响

- 高估：
  - 帕金森法则（Parkinson’ s Law）认为“工作总是用完所有可以利用的时间”
- 低估：质量问题
  - 温伯格定律（Weinberg’Law）:如果一个系统不要求是可靠的，那么它可以满足任何其他目标。

估算的影响


规划错误、高风险、质量问题带来的非线性损失
工作量
成本
进度
帕金森法制导致的线性损失
低估
高估
<100%
100%
>100%
估算结果相对于准确估算值的百分比

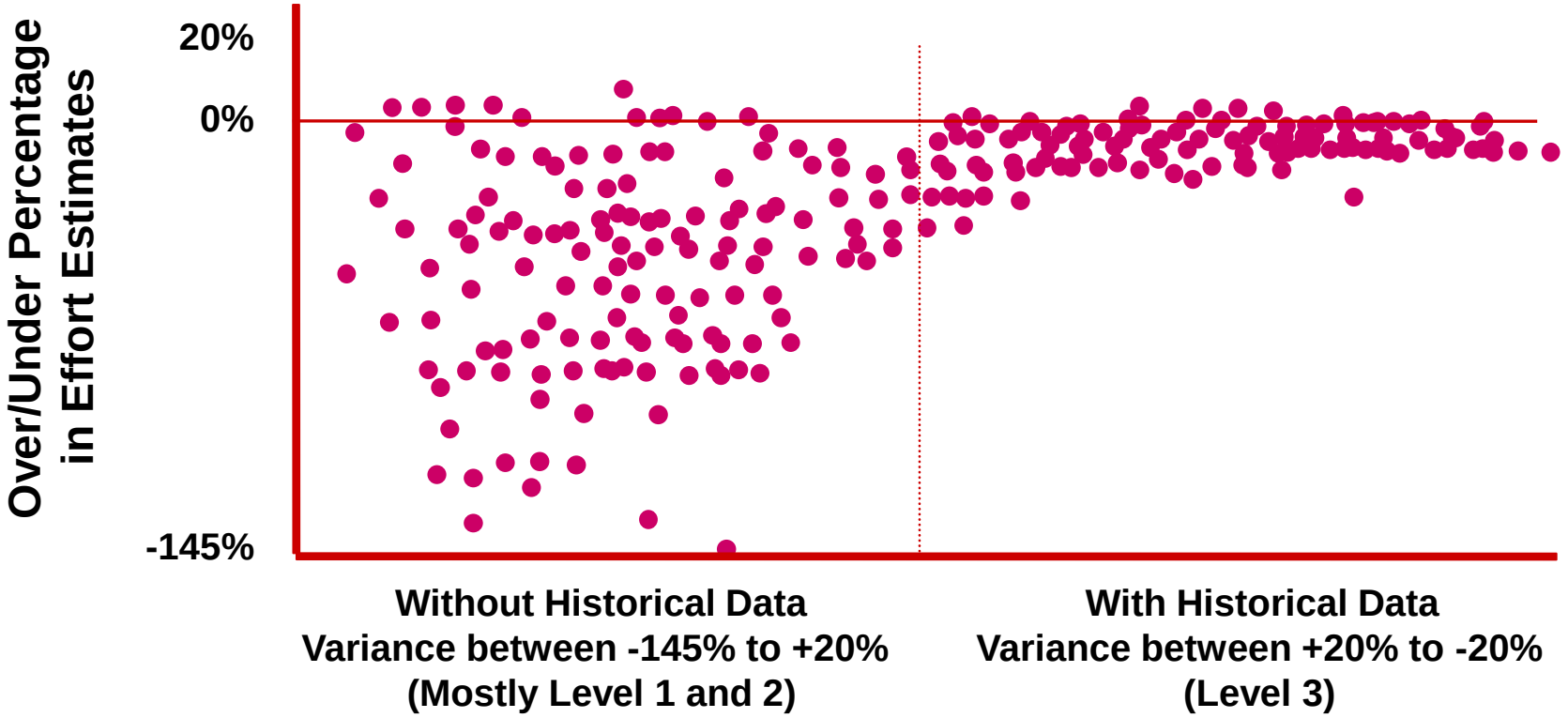
软件项目估算内容



估算过程

- 输入：
  - SOW、WBS
  - 资源单价
  - 历史数据
  - 学习曲线
- 过程：各种估算方法
- 输出：
  - 估算说明文件，其主要包括：
    - 估算出的各种工作量和成本：这是估算的最终结果。
    - SOW和WBS：说明估算的基础。
    - 说明估算的依据和为成本估算所做的任何假设的合理性：重点阐明估算采用的历史数据和资源单价的来源。
    - 估算的误差变动：给出估算结果的误差，也就是公差范围

历史数据



From 120 Projects in Boeing Information Systems

Similar： WPDP- CPDP – transfer learning

Metric is important!

内容

- 估算概述
- 工作量估算方法
- 成本估算方法
- 估算表示

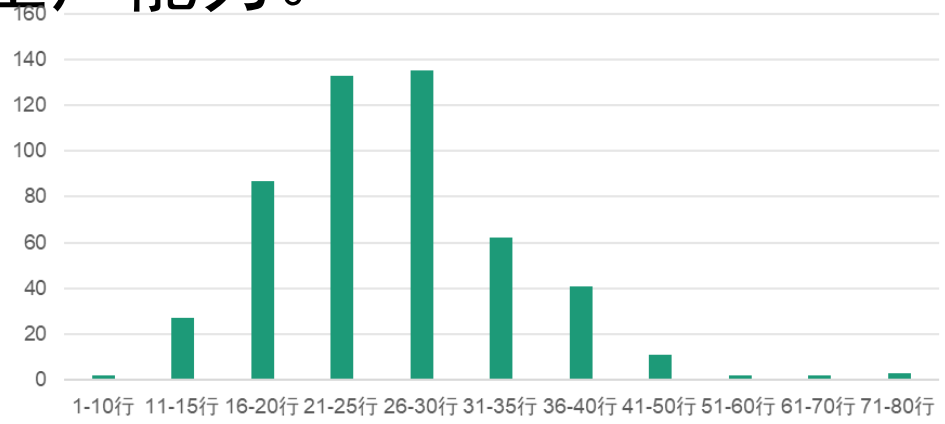
估算方法

- 代码行
- 功能点估算法（Function Point Analysis）
- 用例点估算法
- 自底向上估算
- 类比法
- 专家判断法
- 参数估算法
- 简单估算法(Wideband Delphi)
- 三点估算法
- 其他方法

LOC方法

- LOC(Line of Code)：所有的可执行的源代码行数，包括可交付的工作控制语言语句、数据定义、数据类型声明、输入/输出格式声明等。
  - NCLOC（Non-Commented Source Lines of Code）
  - CLOC（ Commented Source Lines of Code ）
  - LOC=NCLOC+CLOC
- 一代码行（1LOC）的价值和人月均代码行数可以体现一个软件生产组织的生产能力。
- 示例：一个简单

闭包运算学生代码行统计



代码行估算

- 优点
  - 容易理解、便于比较、直观、操作成本低
- 缺点：
  - 早期很难估计
    - 需要较为丰富的开发经验和历史数据的支撑，要求软件分解必须达到相当详细的程度
  - 依赖编程语言和个人编程风格
  - 忽略了项目中很多工作量和成本（例如需求、黑盒测试）
  - 鼓励程序员写冗余代码
  - 面向系统而不是面向用户的

代码行估算

- 一种事后估算，度量指标
- 根据已往的数据对以后的项目产生估算依据
- 程序员平均日生产效率:
  - 100-200 LOC/d
  - 在 Quora 上， Google工程师的回答，有很多人认为一天写 100-150 行代码
  - 20行有效代码
- 代码行应该和缺陷率相结合来衡量代码质量

功能点（Function Points，FP）

- 软件的大小由它所执行的功能的数量和复杂性来衡量
- 用衡量房子的方法来类比
  - 房屋的面积平方米，类比于软件的代码行
  - 房屋的功能：三室两厅两卫，类比于功能点
  - 前者仅是规模，后者体现了尺寸和功能

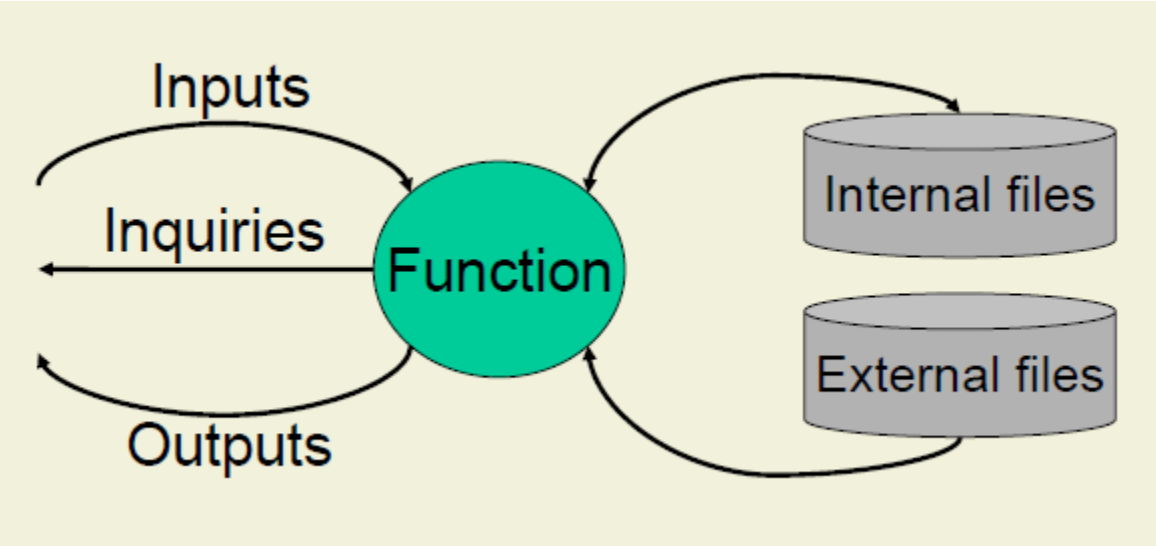
功能点

- 历史：
  - IBM的工程师艾伦·艾尔布策(Allan Albrech)于20世纪70年代提出
  - 从用户的角度来估算软件规模
  - 需求分析阶段基于系统功能的一种规模估算方法
- 功能点构成
  - 5种功能构成
  - 14个复杂度因子

国际功能点用户协会(IFPUG：The International Function Point Users' Group)

5个功能点

- 从软件用户的角度来评估一个软件系统的功能，软件由数据和程序构成的
- 系统功能分为对最终用户不可见的数据功能（内部逻辑文件、外部接口文件）和用户对数据的获取处理的事务功能，即对最终用户可见的交互功能（输入、输出和查询）。



功能点

- 内部逻辑文件（Internal Logical Files，ILF）：在应用程序内部的，用户可识别的、可维护的内部逻辑数据和控制信息。一个逻辑文件可能由单一文件或关系数据库中的单一表组成。
- 外部接口文件(External Interface Files，EIF)：在应用程序边界内被查询，但在其他应用程序中被维护的、逻辑上相关的数据。因此，一个应用程序中的EIF必然是其他应用程序中的ILF。
- 外部输入(External Input，EI)：对用户的输入进行处理的过程。外部输入可以是控制信息，也可是事务数据输入。如果是事务数据，它必须维护一个或多个内部逻辑文件。也就是说那些最后没有保存的中间计算结果和消息发送，都不算作数据输入单元。
- 外部输出(External Output，EO)：向外部发送数据的过程。它的主要目的是通过逻辑处理过程向用户呈现信息。
- 外部查询(External Inquiries，EQ)：是一个输入输出的组合过程。根据用户提出的查询请求，从一个或多个ILF、EIF中取出数据输出到程序外部。其中的输入过程不更新任何ILF，输出过程不进行任何数据处理。EQ不会维护任何一个ILF，也不会改变应用程序的系统行为。

功能点计算过程

- 1.识别系统包含的五种功能点各有多少个
- 2.识别每个功能点的复杂度
  - 复杂、平均和简单，每种复杂度有一个权重值
- 3.计算未调整的功能点（Unadjusted Function Point Count，UFC）
  - 统计各类功能点中各复杂度的功能点数量，分别与对应的权重相乘，得出的总和就是UFC
- 4.计算值调整因子（Value Adjustment Factor，VAF）
  - VAF是基于14个技术复杂度因子（Technical Complexity Factor，TCF）
  - VAF=0.65+0.01*（Sum(Fi)）
  - Fi取值范围0-5，所以VAF结果范围0.65-1.35
- 5.计算调整后功能点
  - FP=UFC*VAF

UFC计算示例

	simple		average		complex	
input	6×	3	2×	4	3×	6
output	7×	4	7×	5	0×	7
inquire	0×	3	2×	4	4×	6
External file	5×	5	2×	7	3×	10
Internal file	9×	7	0×	10	2×	15

UFC＝301

技术复杂度因子

F1	数据通信	F2	分布式数据处理
F3	性能	F4	大业务量配置
F5	事务处理率	F6	在线数据输入
F7	最终用户效率	F8	在线更新
F9	复杂处理	F10	可复用性
F11	易安装性	F12	易操作性
F13	多场地	F14	支持变更

- 每个因子对系统复杂度影响描述分为6级：
  - 0（无影响）
  - 1（偶然影响）
  - 2（有一定影响）
  - 3（一般影响）
  - 4（重要影响）
  - 5（强烈影响）

TCF计算

- TCF=0.65+0.01*(Sum(F_i)), i＝1,2,3,...,14,
- 每个 Fi取值是0到5，所以 TCF 0.65 到 1.35间取值

假设前面例子中此软件项目的技术复杂影响程度都是平均程度，则：

TCF＝0.65+0.01*（14*3）＝1.07

则：**FP＝UFC*TCF=301*1.07＝322**

练习

- 在学院工资系统项目中需要开发一个程序，该程序将从会计系统中提取每年的工资额，并从两个文件中分别提取课程情况和每个老师教的每一门课的时间，该程序将计算每一门课的老师的成本并将结果存成一个文件，该文件可以输出给会计系统，同时该程序也将产生一个报表，以显示对于每一门课，每个老师教学的时间和这些工时的成本。
- 假定报表是具有高度复杂性的，其它具有一般复杂性

练习

	简单	一般	复杂
外部输入	3	0× 4	6
外部输出	4	5	1× 7
外部查询	3	0× 4	6
外部接口文件	5	4× 7	10
内部逻辑文件	7	1× 10	15

UFC＝45

- 外部输出：报表，1
- 内部逻辑文件：成本文件，1
- 外部接口文件：工资文件，人员文件，课程文件，成本文件，4

功能点的简单算法

- 典型权重：
  - 输入 4，输出 5 ，查询 4，内部文件 10，外部文件 10

    （即内部文件是高复杂性，其他都是一般复杂性）
- 估计者根据对复杂度的判断，总数可以用+25%、0、或-25%调整

功能点转化为工作量

- 对于原来的项目，计算生产率：
  - 生产率＝功能点数目/工作量（人日）
  - 则，对于新项目，功能点计算出来后，工作量为：
  - 工作量＝功能点数目/生产率

功能点方法优缺点

- 这种方法的优点：
  - 可以在早期根据功能需求对产品规模进行估算
  - 从客户的角度出发，关注完整的产品
  - 在信息系统、数据库密集系统开发中被广泛认可
  - 遵循标准化的分类和分类程序，并且有基于经验的功能点到工作量转化表存在，而且还能根据实际工作量更新这种经验表
- 劣势：
  - 应用此方法，需要项目中具有有功能点分析经验和知识人员
  - 其考虑复杂性问题的客观性受到质疑，不能应用在逻辑较复杂的实时系统、科学软件开发中，对于此类软件的规模度量，还需要考虑诸如复杂度、状态转换数目等信息。

Reference： 2019，40 Years of Function Points: Past, Present, Future
https://www.ifpug.org/40-years-of-function-points-past-present-future/

简单功能点(SFP)

- 简单功能点协会对该方法进行了改进，然后在2019年被IFPUG收购。
  - 只有两个基本功能组件:基本进程和逻辑文件
  - 它是敏捷过程中故事点的完美“伴侣”
- IFPUG的三个标准
  - Function Point Analysis (FPA)
  - Software Non-functional Assessment Process (SNAP)
  - Simple Function Points (SFP)
- 2023年10月出：Elementary Process and User Stories（从事敏捷软件开发(ASD)工作的专业人员提供了一份新的白皮书）

https://www.ifpug.org/introducing-simple-function-points-sfp/

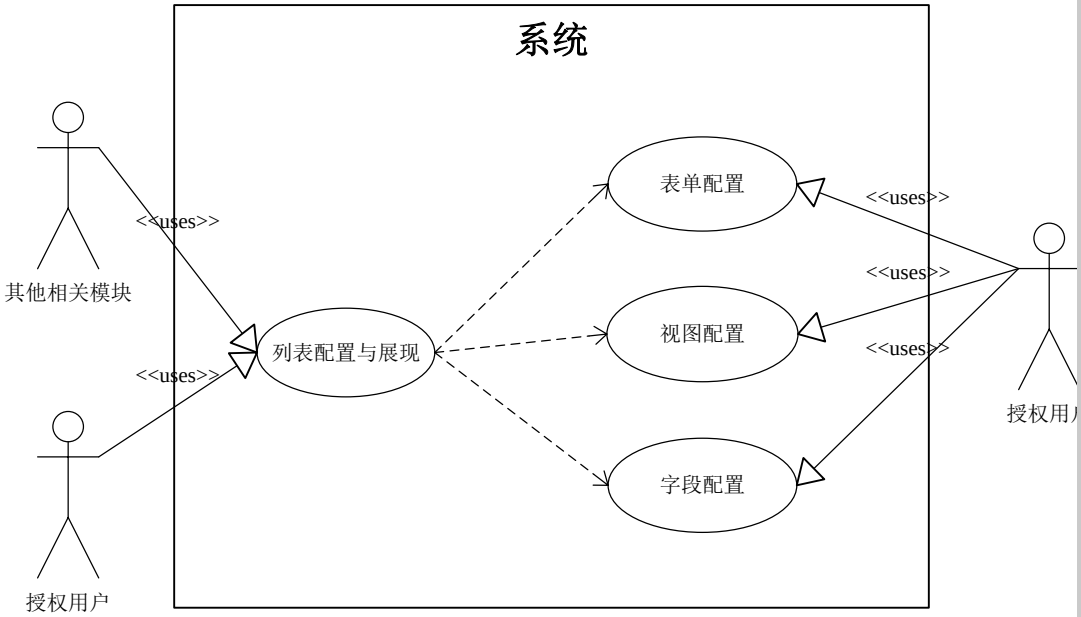
用例点估算法

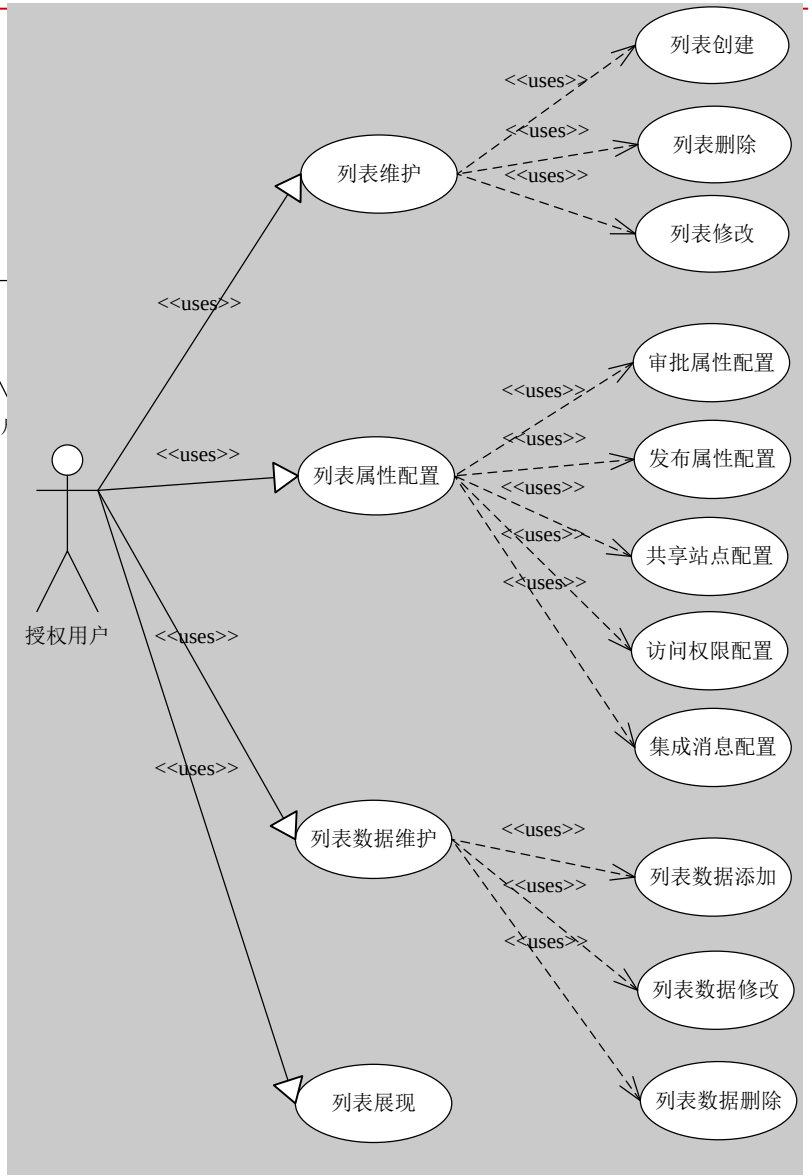
- 用例点方法(use case point method，UCP)是1993年针对功能点方法而提出的一种改进方法
- 面向对象开发方法中基于用例估算工作量的方法。
- UCP的基本思想是利用已经识别出的用例和执行者，根据他们的复杂度分类计算用例点。
- 用例和代码之间存在明显的关系，复杂的用例通常比简单的用例需要更长的时间来编写代码。
- 这种估计方法不包括与项目管理、基础设施和其他支持工作以及验收测试相关的工作，主要是详细设计、实现、单元测试和集成测试的工作量估计。

用例点估算法

- 如果一个项目团队希望用用例点进行估算，他们应该在用户目标级别上编写他们的用例。
- 每个用例都有一个目标，此目标是业务价值的一个基本单元。
- 对于用户目标用例编写的是否合适，有两个判断条件：
  - 首先，用户实现目标的频率越高，交付给业务的价值就越多；
  - 第二，用例通常在单个会话中完成，在实现目标后，用户可能会继续进行其他活动。

用户目标级别用例举例

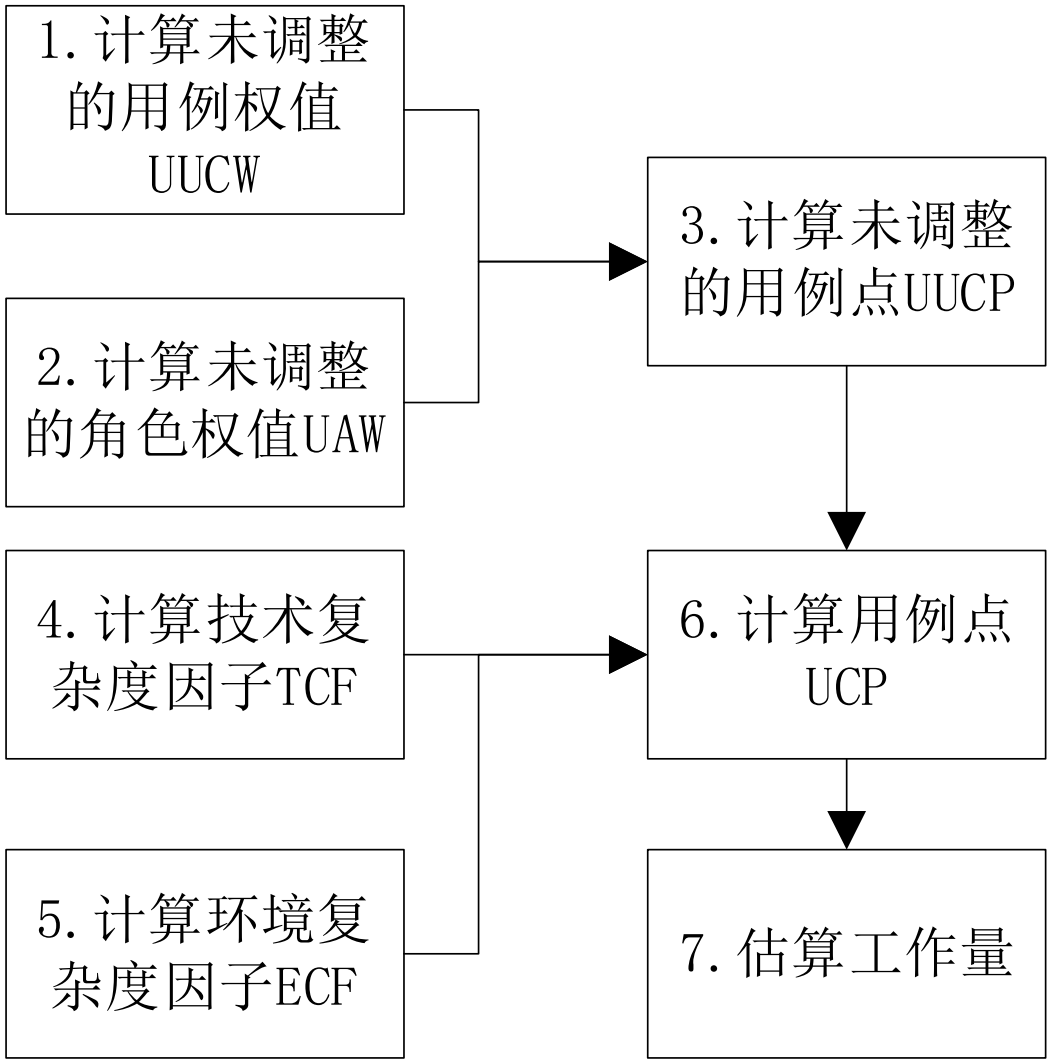




- 非用户级别
  - 系统高层用例

用例点估算步骤

- 7步


1. 计算未调整的用例权值UUCW
2. 计算未调整的角色权值UAW
3. 计算未调整的用例点UUCP
4. 计算技术复杂度因子TCF
5. 计算环境复杂度因子ECF
6. 计算用例点UCP
7. 估算工作量

UUCW

- 步骤1：计算未调整的用例权值（Unadjusted Use Case Weight，UUCW）
  - 根据用例模型确定用例及其复杂度。
  - $UUCW$ = 简单型用例总数 * 5 + 一般型用例总数 * 10 + 复杂型用例总数 * 15

用例类型	描述（事务/场景个数；实现类个数；界面和数据库实体）	用例权值
简单	它的成功场景包含3步或更少的步骤；它的实现涉及不到5个类；简单的用户界面，只涉及单个数据库实体；	5
一般	用例步骤在4到7步之间；它的实现涉及5到10个类；更多界面设计，涉及两个或多个数据库实体。	10
复杂	用例超过7个步骤；它的实现涉及10多个类；复杂的用户界面或处理过程，涉及三个或更多的数据库实体。	15

UAW

- 步骤2：计算未调整的角色权值（Unadjusted Actor Weight，UAW）
  - 根据用例模型确定角色个数及其复杂度。用例中涉及的参与者（角色）是复杂性另一个方面。
  - UAW = 简单型角色总数 * 1 + 一般型角色总数 * 2 + 复杂型角色总数 * 3

角色类型	判定标准	角色权值
简单	通过API与之交互的另一个系统	1
一般	通过基于文本的用户界面进行交互的人，也可能是通过协议进行交互的另一个系统	2
复杂	通过图形用户界面与系统交互的人	3

UUCP

- 步骤3：计算未调整的用例点（Unadjusted Use Case Points，UUCP）
  - 未调整的用例权值(UUCW)和未调整的角色权值(UAW)之和
  - $\mathrm{UUCP} = \mathrm{UUCW} + \mathrm{UAW}$

TCF和ECF

- 步骤4. 计算技术复杂性因子（Technical Complexity Factor，TCF）
  - 13个技术复杂性因子Ti，每个因子给出评估结果（0到5，0表示此因子和项目无关，3表示影响一般，5表示因子对项目强烈影响），每个因子有一个权重值。
  - 例如因子“并发”
  - $\mathrm{TCF}=0.6+\left(0.01*\sum_1^{13}T_i*Weight(T_i)\right)$
- 步骤5. 计算环境复杂性因子（Environmental Complexity Factor，ECF）
  - 8个环境复杂性因子Ei，每个因子给出评估结果（0到5），每个因子有一个权重值。
  - 例如因子“需求稳定性、应用领域经验”
  - $\mathrm{ECF}=1.4+\left(-0.03*\sum_1^{8}E_i*Weight(E_i)\right)$

UCP和EE

- 步骤6：计算调整后的用例点（ Use Case Points，UCP）
  - $UCP = UUCP * TCF * ECF$
- 步骤7：估算工作量（Estimate Effort， EE ）
  - 需要把UCP转换成有意义的人月或者人时度量的工作量。
  - 引入每个用例点的开发时间，即项目生产率因子（Productivity Factor，PF）。PF一般在15-30之间，即一个用例点需要15-30人时，建议值是每个用例点20小时。
  - EE=UCP*PF
  - 但是更好的确定PF的方法是：计算项目组织过去的项目的历史平均值。例如，如果五个最近的项目包括2000个用例点，并代表44000个小时的工作，则表示组织的平均生产率是每个用例点22小时(44000÷2000=22)。
  - 从组织度量角度，每个项目完成后都应该存储PF值作为以后项目的参考数据。

用例点估算

- 优点：
  - 估算过程可以自动化。一些用例管理工具会自动计算系统中用例点的数量。
  - 用例点有可能产生可靠的结果，因为它一方面是基于用例，另一方面充分考虑有影响的技术和环境因素。
  - 如果企业建立了组织自身的每个用例点的平均实现时间，这对预测未来的时间表非常有用。
- 缺点：
  - 在编写完所有用例之前无法进行评估。编写用户目标用例是一项重要的工作，可能代表项目总体工作的10-20%。这样就延迟了团队创建发布计划的时间点。
  - 这种方法依赖于用例的描述，用例描述的详细程度和用例的结构都对估算有一定的影响。
  - 用例点估算方法并不特别适合使用敏捷软件开发过程，因为创建一个完整的用例模型的需要与敏捷的价值观是不相容的。

自底向上评估

- 创建WBS
- 对WBS中单个工作包进行详细的规模、成本估算，然后按照WBS的层次结构将结果累加起来得出项目总规模或成本
  - 结合PERT方法进行估算
- 优点
  - 如果WBS分解的足够准确，则这是一种准确的估算方法
- 缺点
  - 不适用于项目初期的估算
  - 采用这种方法进行估算本身也是很花费时间和精力的
  - 估算结果容易低估，因为工作量不是线性增长的
  - 可能低估了系统级活动(如集成和文档编制)的成本。

类比法

- 自顶向下类推法:将估算项目的总体参数与类似项目进行直接相比。
- 自底向上类推法:比较类似的工作单元
- 优点：简单易行，适合项目初期信息不足时候使用（比如合同期和市场招标）
- 缺点：
  - 准确性差
  - 如果没有类似的项目，这种方法是不可行的

类比法中重用代码的估算

- 类比法中要解决可重用代码的估算问题。
- 一般估算出：新项目可重用的代码中需重新设计的代码百分比、需重新编码或修改的代码百分比以及需重新测试的代码百分比。则等价新代码行为：等价代码行 = [(重新设计% +重新编码% +重新测试%)/3]×已有代码行
- 比如：有10,000行代码，假定30%需要重新设计，50%需要重新编码，70%需要重新测试，那么其等价的代码行可以计算为：
- [(30% + 50% + 70%)/3]× 10,000 = 5,000 等价代码行。
- 意即：重用这10000代码相当于编写5000代码行的工作量。

专家判断法(Deiphi法)

- 专家判断法是通过专家对项目成本做出估算。这种方法的精确性取决于专家对估算项目的定性参数的了解和经验。
- 现在比较常用的专家判断法是德尔菲法(Delphi technique)。

德尔菲法的具体步骤

- (1) 协调人向各专家提供项目规格和估算表格，并且告知各专家与项目规模相关的因素；
- (2) 各专家匿名填写估算表格；
- (3) 协调人根据各个专家估算的结果整理出一个估算总结，如果对同一个工作单元估算结果在可以接受的误差范围之内，那么此估算结束，否则进入下一步；
- (4) 协调人再次就估算差异较大的项制定估算表格反馈给专家，并且把其他专家的估算结果和原因作为可选项列在表格中供专家参考；
- (5) 专家复查估算结果并提交另一个匿名估算；
- (7) 重复3～5，直到最终的估算结果是项目小组可以接受的误差范围内的数据。

参数估算法

- 参数估算法是基于一种建模统计技术，如回归分析和学习曲线等。用于估算的算式通常由理论或经验导出。
- 估算模型的结构：$E=A+B*(X)^C$
  - A、B、C是由经验导出的常数，
  - E是以人月为单位的工作量
  - X 是估算变量

参数模型

- 基于代码行的
  - $E=5.2*(KLOC)^{0.91}$ Walston-Felix model
  - $E=5.5+0.73*(KLOC)^{1.16}$ Bailey-Basili model
  - $E=3.2*(KLOC)^{1.05}$ Boehm model (simple)
  - $E=5.288*(KLOC)^{1.047}$ Doty model for KLOC > 9
- 基于功能点的
  - $E=-13.39+0.0545FP$ Albrecht and Gaffney model
  - $E=60.62*7.728*10^{(-8)}*FP^3$ Kemerer model
  - $E=585.7+5.12FP$ Maston、Barnett &Mellichamp

COCOMO：参数化模型

- COCOMO：Constructive Cost Model构造性成本模型
- 构造性成本模型是一种精确、易于使用的基于规模的成本估算方法。
- 它是由Barry Boehm在1981年提出的，基于对63个项目的研究结果。
- COCOMO模型是一个综合经验模型，模型中的参数取值来自于经验值，并且综合了诸多的因素，是比较全面的估算模型，用于对软件开发项目的规模、成本、进度等方面进行估算。
- 无论最初的COCOMO 81模型，还是后来提出的COCOMO II模型，都是文档完备的模型，并且存在以此种算法为基础的估算工具，得到了广泛使用和评价

成本估算模型

- COCOMO 模型（Constructive Cost MOdel）
  - 软件估算模型的层次体系

模型1：基本COCOMO模型，将软件开发工作量及成本作为程序规模的函数进行计算，程序规模由已估算的代码来表示。

模型2：中级COCOMO模型，将软件开发工作量及成本作为程序规模及一组“成本驱动因子”的函数来进行计算，其中“成本驱动因子”包括对产品、硬件、人员、及项目属性的主观评估。

模型3：高级COCOMO模型，包含了中级模型能够的所有特性，并结合了成本驱动因子对软件工程过程中每一步骤的影响评估。

基本COCOMO模型

- 基本COCOMO模型的公式为：
- $Effort=c \times size^{k}$
  - 其中effort采用“人月”pm来度量，size采用kdsi即千行交付源代码指令(thousands of delivered source code instructions)
  - DSI-源指令条数。不包括注释。1KDSI = 1000DSI
  - 开发工作量（以人月计） 1pm = 19 人日 = 152 人时 =1/12 人年
  - C和k的值基于开发模式而取值

COCOMO系数

- C,k的取值根据系统的分类而定：
  - 根据系统的技术特性和开发环境可以分为：
  - 有机模式（organic mode)：相对小的团队在一个高度熟悉的内部环境中开发规模较小，接口需求较灵活的系统。(规模小于5万行)
  - 嵌入式模式（Embedded Mode)开发的产品在高度约束的条件下进行，通常与某种复杂的硬件设备紧密结合在一起。对接口、数据结构、算法要求都比较高，对系统改变的成本很高。软件规模任意。
  - 半分离模式（Semi-detached Mode):两者之间,规模和复杂度都属于中等，最大可达30万行。

COCOMO系数

- 系数表：

系统类型	c	k	a	b
有机模式	2.4	1.05	2.5	0.38
嵌入型	3.6	1.20	2.5	0.32
半分离型	3.0	1.12	2.5	0.35

- 工作量= c*(规模)^{k}
- 进度 = a*(工作量)^{b}
- K的值反映了项目越大越复杂，则工作量成指数增加，因为大项目需要更多的协调和安排。

中等COCOMO模型

- 中级COCOMO模型将软件开发工作量及成本作为程序规模及一组“成本驱动因子”的函数来进行计算，其中“成本驱动因子”包括对产品、硬件、人员、及项目属性的主观评估，共15个成本驱动因子。
- $Effort=(c\times size^{k})*dem$
  - 公式中$(c\times size^{k})$结果称为名义工作量，中等模型中c，k取值有变化，也是三种类型下取值，但是取值比初等模型略有变化（c增大）
  - 开发工作量乘子：dem (development effort multiplier)，是每个成本驱动因子所选择的系数相乘来产生每个组合的系数

dem

驱动因子类型	编码	驱动因子
产品属性	RELY	重要的软件可靠性
	DATA	数据库规模
	CPLX	产品复杂度
计算机属性	TIME	执行时间限制
	STOR	主存限制
	VIRT	虚拟机易变性：操作系统变更的程度
	TURN	计算机往返时间
人员属性	ACAP	分析员能力
	AEXP	应用经验
	PCAP	程序员能力
	VEXP	虚拟机（OS）经验
	LEXP	编程语言经验
项目属性	MODP	现代编程实践的使用
	TOOL	软件工具的使用
	SCED	需要的开发进度

非常低	1.46
低	1.19
正常	1.00
高	0.80
非常高	0.71

高级COCOMO模型

- 高级COCOMO模型包括了中级COCOMO模型的所有特性，其工作量以及进度的估算公式和中级COCOMO模型一致，但是高级COCOMO模型中引入了两种主要功能：
  - 阶段敏感工作系数。高级COCOMO模型还要考虑软件工程过程中分析、设计等各步骤的影响，认为某些阶段（设计、编码和调试）比其他阶段有关因素的影响可能更大。高级COCOMO模型为每个因素提供了一个“阶段敏感工作权数”。
  - 三层产品分级结构。三个产品层次是模块、子系统和系统。
- 所以高级COCOMO模型工作量因素分层、分阶段给出。针对每一个影响因素，按模块层、子系统层、系统层，有三张工作量因素分级表，供不同层次的估算使用，每一张表中工作量因素又按开发各个不同阶段给出。

练习

- 在某企业中，绝大多数系统技术上，产品，计算机和项目等属性都是类似的,因此可以给出1.0的系数。只有人员的属性有所差异。该企业制定了下表：

Personnel attributes	ACAP	analyst capability
	AEXP	application experience
	PCAP	programmer capability
	VEXP	virtual machine (i.e. operating system) experience
	LEXP	programming language experience

Attribute	*Very low*	*Low*	*Nominal*	*High*	*Very high*
ACAP	1.46	1.19	1.00	0.86	0.71
AEXP	1.29	1.13	1.00	0.91	0.82
PCAP	1.42	1.17	1.00	0.80.	0.70
VEXP	1.21	1.10	1.00	0.90	—
LEXP	1.14	1.07	1.00	0.95	—

- 分析员非常优秀，编程人员能力是高的但是对该项目面向的领域不熟悉并准备用新的编程语言。他们对操作系统很熟悉。请计算dem。如果名义工作量是4人月，则估算的工作量是多少？
- 每个因子取值：ACAP：0.71，AEXP：1.13，PCAP：0.8，VEXP：0.9，LEXP：1.07
- 则dem=0.71*1.13*0.8*0.9*1.07=0.62
- 最终估算结果：4*0.62=2.48人月

COCOMOII

- COCOMO 81是在使用瀑布过程的假设下开发的，所有的软件都是从头开始开发的
- COCOMOII的产生是为了适应不同的软件开发方法。COCOMOII是以原始COCOMO为基础，它更容易允许对面向对象的软件、通过螺旋或进化模型创建的软件以及从商业现成软件开发的应用程序进行评估。
- 该方法利用多种乘法算子和指数。
- 此模型适合于各种类型的软件项目。
- 一个明显的特点是该模型适应了项目在执行过程中变得越来越确定的状态，因而是一种渐进性评估。

COCOMOII

- COCOMOII被分为三个阶段性模型：
  - 应用构成阶段（Application Composition)：用于软件开发早期，支持对原型开发项目所需工作量的估算，同时也支持基于已有构件进行开发的软件项目。在这个阶段设计了用户将体验的系统的外部特征。
  - 早期设计（Early Design)：基本的软件结构被设计。
  - 后架构阶段（Post architecture)：软件构造经历了最后的构造、修改，并在需要时候开始创建要执行的系统。

简单估算方法

- 简单估算方法，实际是“专家判断法”的变形，采用项目组成员作为项目的“专家”来进行估计。
- 因为项目组的成员对项目的了解程度是比项目组外任何人都强的，而且项目组对于项目的各种功能是达成一致共识的，因此能够让他们作为项目的“专家”来进行估算。

产品规模简单估算方法(Wideband Delphi)

- 1.项目规划小组先分解产品的功能，制定“产品功能分解与规模估计表” 。软件规模的度量单位主要有：代码行、对象个数等等。

模块名称	模块的主要功能	新开发的软件规模（度量单位如对象个数）	复用的软件规模（度量单位如对象个数）
模块 A	A.1		
	A.2		
	A.3		
模块 B	B.1		
	B.2		
	B.3		
汇总			

产品规模简单估算方法

- 2. 规划小组各成员独立填写表格。
- 3. 汇总每个成员的表格，进行对比分析。如果各人估计的差额小于20%，则取平均值。如果差额大于20%，则转向第（2）步，让各成员重新估计产品的规模，直到各人估计的差额小于20%为止。

工作量简单估算方法

- 步骤与规模估计相似
- 1. 先估算开发工作量。一般地可以把开发过程划分为需求开发、系统设计、实现、测试四个阶段，分别估计每个阶段的工作量，然后累计得出总的工作量
- 2. 再估算管理工作量。一般地，项目的80%以上的工作量用于开发，20%以下的工作量用于管理。

工作量的度量单位	1 人年 = 12 人月，1 人月 ≈ 22 人天，1 人天 = 8 人小时		
开发工作量估算公式	项目开发工作量 ≈ 新开发的软件规模 / 人均生产率		
开发阶段	人均生产率	软件规模	工作量
需求开发			
系统设计			
实现			
测试			
…			
开发工作量汇总			
管理工作量估算公式	项目管理工作量 ≈ 开发工作量 * 比例系数		
项目管理的主要事务	项目规划、项目监控、需求管理、配置管理、质量管理…		
比例系数	例如 20%		
管理工作量汇总			

简单估算法实际案例

- 案例

敏捷中宽带德尔菲法体现

- 扑克牌法：
  - 每个团队成员拿到有编号的卡片
  - 产品负责人朗读每个故事卡片并回答团队问题；
  - 每个团队成员评估故事工作量并给故事分配点数；
  - 当Scrum Master要求时，每个人必须同时举起写有他们估算的数字卡；
  - 如果有差异，团队成员要解释估算偏高或偏低的原因；
  - 讨论后，团队成员重新估算直到达成一致。

三点估算法（PERT法）

- 三点估算法，也叫计划评估和审查技术(Program Evaluation and Review Technique，PERT)。
- 估算时考虑三种情况：
  - 最乐观**O**ptimistic——好，最悲观**P**essimistic---坏，最可能**L**ikely———一般　　建议：(P -O)/L< 40%
- 估算公式：
  - 均值 M = (O + 4 × L + P) / 6
  - 标准差： σ = (P - O) / 6
- 完成概率：
  - 1σ：68.3%，2σ：95.5%，3σ：99.7%
- 此方法往往和前面估算方法结合使用，例如类比法时给出三点估算值。
- 假设条件：
  - 项目工作量是正态分布的

三点估算法（PERT法）

- 例如，假设A任务持续时间悲观估计为36天，最大可能估计为21天，乐观估计为6天。那么估计A任务在16到26天之间完成的概率有多大？
- 首先计算出标准差为（36-6）/6=5，由最大可能21天的一个标准差范围是：21+5=26，21-5=16
- 所以16到26之间为1个标准差，完成概率是68.3%。

基于人工智能方法的估算方法

- AI +SE

- 参考文献：
- “Machine Learning-based Software Effort Estimation : An Analysis” , 2019, 11th Edition Electronics, Computers and Artificial Intelligence
  - The dataset PROMISE
- Systematic literature review of machine learning based software development effort estimation models, 2012,Information and software technology

敏捷项目估算

- 故事点
  - 用于表示对实现产品待办事项所需的总体工作量的估计的度量单位
  - 不是花费的时间(pm)
  - 扑克牌法估算
  - T恤尺寸:许多团队使用t恤尺寸(S, M, L, XL)来描述每个属性或故事。如果需要，可以在估算完成后给出尺寸的数值。

估算方法总结

- 每种方法都有优点和缺点评估
- 应该基于几种方法进行估算
- 如果它们没有返回近似相同的结果，则说明可用信息不足以做出准确的估计，应该采取一些行动来了解更多情况

内容

- 估算概述
- 工作量估算方法
- 成本估算方法
- 估算表示

关于成本的故事

曾经有一位项目经理解释他的项目计划，其中有几个成员在某几天里面是没有工作安排的，他说：“就让他们待在公司里面休整一下吧，反正也不花钱。”

软件项目成本构成

- 国家标准《软件工程 软件开发成本度量规范》（GB/T36964-2018） 中
- 软件开发成本（SDC）为从项目立项开始到项目完成验收之间所涉及的需求分析、概要设计、编码实现、集成测试、验收交付及相关的项目管理支持活动。
- 软件开发成本仅包括软件开发过程中的所有人力成本和非人力成本之和，不包括数据迁移和软件维护等成本。
  - 人力成本包括直接人力成本（DHC）和间接人力成本（IHC）
  - 非人力成本包括直接非人力成本（DNC）和间接非人力成本（INC）。
  - 直接人力成本和直接非人力成本统称为直接成本，间接人力成本和间接非人力成本统称为间接成本

人力成本

- **1. 直接人力成本（DHC）**
  - 包括开发方项目组成员的工资、奖金和福利等人力资源费用。项目成员包括参与该项目开发过程的所有开发或支持人员。
  - 对于非全职投入该项目开发工作的人员，按照项目工作量所占其总工作量比例折算其人力资源费用。
- **2. 间接人力成本（IHC）**
  - 指开发方服务于开发管理整体需求的非项目组人员的人力资源费用分摊。包括开发部门经理、项目管理办公室人员等人员的工资、奖金和福利等的分摊。

非人力成本

- **3. 直接非人力成本（DNC）**
  - 办公费
  - 差旅费
  - 培训费
  - 业务费：如招待费、评审费和验收费等；
  - 采购费
  - 其他：未在以上项目列出但却是开发方为开发此项目所需花费的费用。
- **4. 间接非人力成本（INC）**
  - 指开发方不为开发某个特定项目而产生，但服务于整体开发活动的非人力成本分摊。例如开发方开发场地房租、水电和物业等。

- 需要注意的是，在编制软件项目预算、报价或结算时，除软件开发成本外，考虑开发方合理的毛利润水平是必要的。对于需要提供其他支持服务的项目或产品，还需要考虑支持活动所需的各种成本，如数据迁移费和维护费等

1）基于成本构成的成本估算方法

- 1. 直接人力成本估算
  - 得到工作量估算结果后，根据工作量估算结果和项目人员直接人力成本费率估算直接人力成本。
  - 直接人力成本费率是指每人月的直接人力成本金额，单位通常为元每人月。
- 直接人力成本的计算可以采用以下两种方式之一：
  - （1）根据不同类别人员的直接人力成本费率和估算工作量分别计算每类人员的直接人力成本，将各类人员的直接人力成本相加得到该项目的直接人力成本；对应估算公式如下：
    - $\mathrm{DHC} = \sum_1^n (E_i \times IF_i)$
  - 其中，DHC是直接人力成本，单位元；n是人员类别数量，取值为不小于1的自然数；E_i是第i类人员的工作量，单位人月；IF_i是第i类人月的直接人力成本费率，单位是元/人月。

基于成本构成的成本估算方法

- 直接人力成本的计算可以采用以下两种方式之一：
  - （2）根据项目平均直接人力成本费率和估算的总工作量直接计算该项目的直接人力成本。
    - $DHC = E \times IF$
  - 其中，DHC是直接人力成本，单位元；E是项目总工作量，单位人月；IF是平均直接人力成本费率，单位是元/人月

基于成本构成的成本估算方法

- 2. 间接人力成本宜按照工作量比例进行分摊。
  - 例如质量保证部门的质量保证人员甲负责组织级质量保证工作和3个项目（A、B、C）的项目级质量保证工作。
  - 其中用于项目A、B、C的工作量各占总工作量的1/4，用于组织级质量保证工作和其他工作的工作量占其总工作量的1/4；同时项目A的开发总工作量占该组织所有开发项目总工作量的1/3
  - 则质量保证人员甲的人力资源费用中1/4计入项目A的直接人力成本，1/12计入项目A的间接人力成本（因为：占质量保证工程师甲1/4的组织级质量保证工作和其他工作中，只有1/3计入项目A的成本）。

基于成本构成的成本估算方法

- 3. 直接非人力成本估算
  - 可以按照直接非人力成本分项估算，也可依据基准数据或经验估算。
  - 例如：项目组封闭开发租用会议室而产生的费用宜计入直接非人力成本中的办公费；为项目采购专用测试软件的成本宜计入直接非人力成本中的采购费。
- 4. 间接非人力成本宜按照工作量比例进行分摊。
  - 例如，某公司有员工200人，1年的房屋租赁费为人民币120万元，则每人每月的房租分摊为500元（1200000/（200*12））。如果项目A的总工作量为100人月，则分摊到项目A的房屋租赁费为人民币5万元（即100人月×500元/人月）。

基于成本构成的软件开发成本估算

- 求出软件开发成本每个分项后，则软件开发成本就是四项之和：
  - 开发成本 = 直接人力成本 + 直接非人力成本 + 间接人力成本 + 间接非人力成本。

- 间接成本简单估算方法：
  - 间接人力成本和间接非人力成本都是通过分摊求得，所以可以通过组织的间接成本系数求间接成本。
  - 人力成本费率=直接人力成本费率*（1+间接成本系数）
- 则开发成本 = 工作量 * 人力成本费率 + 直接非人力成本

2）基于规模的软件开发成本估算

- 对于委托方，如果已经确定了规模综合单价，则可以根据规模综合单价和估算出的规模直接计算出直接人力成本和间接成本的总和，然后计算软件开发成本：

$$SDC = P \times S + DNC$$

  - 其中，P为规模综合单价，单位为元每功能点（元/FP），S为软件规模，单位为功能点（FP）
- DNC：直接非人力成本

3）基于参数的简化软件开发成本估算

- 直接非人力成本估算有时也不容易估算，因此，更简单的方法是用组织的参数来由直接人力成本估算项目总成本
  - 开发成本=开发工作量*人力成本参数，人力成本参数单位元/人月
  - 管理成本=比例系数*开发成本，其中比例系数约为20%-25%
  - 直接成本=开发成本+管理成本
  - 间接成本＝直接成本*间接成本系数=（开发成本+管理成本）*间接成本系数，此处的间接成本系数也是需要根据公司历来项目情况的数据得出的。一般也可以取20%。
  - 总成本＝（开发成本+管理成本）+间接成本=（开发成本+管理成本）*（1+间接成本系数）= （开发成本*（1+管理成本系数））*（1+间接成本系数）
- 如果项目合同中包含一些固定价格的采购，比如硬件设备，这些在计算总成本时候需要加入

从成本到报价

- 基于成本估算可以得到项目报价，一般两种方法：
- 方法1：项目报价=项目总成本/（1-利润系数）
- 例如
  - 成本 = RMY 1.000.000
  - 利润系数 = 5%
  - 价格 = RMY 1.052.632
- 方法2：项目报价=项目总成本+风险利润＝项目总成本*（1+利润系数）
  - 例子中：RMY 1.050.000

4）软件项目快速成本估算方法1

- 只考虑人力成本的估算法
- 基于人月工作量：
  - 估算的基本公式是：成本＝工作量*单位人员平均工资×（成本系数）

例如：项目共需要24个人月，若人员平均工资每月5000元，则简单的成本估算为（24×5000元）×3＝360000元

以人月为基础估算成本

- 成本系数的确定根据历史经验：
  - 人员规模越大，成本系数越高。
  - 技术水平越高，成本系数越高。
  - 开发周期越长，成本系数越高。
  - 一般系数为：1.5~3.0之间。
- 这种方法的特点：
  - 简单，容易估算
  - 需要建立在工作量计算的基础上进行估算
  - 不够准确，弹性大

5）软件项目快速成本估算方法1

- 以功能为基础的成本估算
- 功能估算办法：
  - 1、整理出项目功能列表；
  - 2、将功能列表进行归类，整理成模块；
  - 3、按照模块估算代码量和工作量；
  - 4、估算出功能点的成本；
  - 5 、根据用户的需求和实现方式，估算开发系数。
- 基本计算公式：功能模块单价×功能块点数
- 改进公式：功能模块单价×功能点数×开发系数

以功能为基础的成本估算

- 计算公式：　（功能模块×单价）×功能块点数
- 例如：某个系统可分为10个模块，每个模块按照历史的经验计算，其中3个为15000元，5个为20000元，2个为4000元，则系统的成本为：（15000 × 3)+(20000 ×5）＋（4000 ×2)=1530000元
- 改进公式：功能模块单价×功能点数×开发系数
- 例如：某个系统可分为10个模块，每个模块按照历史的经验计算，其中：3个为15000元，开发难度系数为2；5个为20000元，开发难度系数为3；2个为4000元，开发难度系数为1
- 则系统的成本为：（15000 × 3) ×2 +(20000 ×5）×3 ＋（4000 ×2) ×1 =398000元

以功能为基础的成本估算

- 历史经验：
  - 系统越复杂，开发难度系数越高
  - 开发架构与语言越高级，开发难度越高
  - 功能点越精细，准确度越高
  - 团队开发历史越久，准确度越高
- 功能点单价除了根据历史经验外可参考同等规模的同行报价。
- 特点：
  - 需要参照历史经验或者同类产品
  - 需要进行需求分析与概要设计
  - 准确度相对比较高

估算示例

- 已知：系统共5个模块，第1和第2模块自研，按照WBS预计103人日。但是项目WBS中只包括开发任务。
- 第3、4、5个模块外包或外购的价格分别为5000元、3000元、3000元。
- 因此根据以往的经验，管理任务和质量任务＝20%*开发任务。
- 估计项目成本是多少？

估算示例

- 计算开发成本：
  - 根据经验，开发人员成本参数＝480元/天，则内部开发成本＝480元/天*103天＝49440元
  - 外购、外包成本＝5000+3000+3000＝11000元
  - 所以开发成本＝11000+49440＝60440元
- 计算管理成本：根据经验，管理成本系数为20%，所以
  - 管理成本＝60440*20%＝12088元
- 计算直接成本＝60440+12088＝72528元

估算示例

- 计算间接成本：
  - 根据经验，间接成本系数为25%，则间接成本＝25%*直接成本＝72528*0.25＝18132元
- 计算总估算成本：总估算成本＝72528+18132＝90660
- 计算报价：
  - 假设项目利润是30%，则项目总报价＝90660*1.3＝117858元

估算示例

- 简便算法进行估算，企业的报价可以直接从开发规模的估算直接得出：
  - 如果含利润成本系数是2.5万元/人月
  - 则总报价＝25000*103/22＝117045元

其他人建议公式：

- 总结11个行业8300个系统得出以下公式：
  项目成本 = 开发 + 测试 + 项目管理 + 质量管理 + 返工 + 意外
  = x + 32%*x +32%*x +16%*x + 41%*x + 15%*x
  = 236%*x

无效的项目估计

- 在某种情况下，任何的项目估计方法都没有实际价值，例如：
  - 项目的人员已经被上级领导限定死了，再多的活也是那几个人干；
  - 除了办公计算机和工资外，这个项目没有其它经费，项目经理只有干活的权利没有用钱的权利；
  - 项目的结束日期早就被领导和客户指定了，不管合理不合理，反正时间一到就要交付软件。
  - 如果人员、资金、时间都已经被毫无道理地指定了，你进行科学地估计还有啥用？这样的项目在国内并不少见，如果你碰上了，那么就自认倒霉吧。

内容

- 估算概述
- 工作量估算方法
- 成本估算方法
- 估算表示

成本估算的准确度

类型	准确度	说明
量级估算：合同前	-25%--+75%	概念和启动阶段，决策
预算估算：合同期	-10%--+25%	编制初步计划
确定性估算：WBS后	-5%--+10%	任务分解后的详细计划

估算的表达方式

- 1. 加减限定表示，例如6个人月规模可以表示成为6+3人月，6-1人月
- 2. 范围表示：例如6个人月表示成为5-9人月
- 3. 风险量化：
- 4. 分情况阐述:最佳情况3个月，计划情况6个月，最差情况12个月。

风险量化

估算：6个月，+3个月，-2个月			
+1个月：	延迟交付转换子系统	-1个月：	新成员的工作效率高
+1个月：	新开发工具没有希望的好用	-1个月：	新开发工具比预期的好用
+0.5个月：	员工病假		
+0.5个月：	低估规模		

Software of estimation

- SystemStar (http://www.softstarsystems.com/)

总结

- 估算顺序
- 工作量估算方法
- 成本估算方法