

Chapter 6

software project estimation



● outline

- Estimation overview
- Effort Estimation techniques
- Cost Estimation techniques
- Error of estimation



1.给需求

2.根据这些需求得到一个大概的时间3个月

3.客户和公司谈条件2个月

4.妥协2个月

(威胁论：你不做有人做)

5.签合同

市场论：打开一个新的领域

6.开工，就时间再商谈



6.1.成功：加时间

6.2.不成功：程序员加班

(软件工程、足够的测试等等都尽可能不要，只要到时候能跑起来就可以)



8.客户不满意

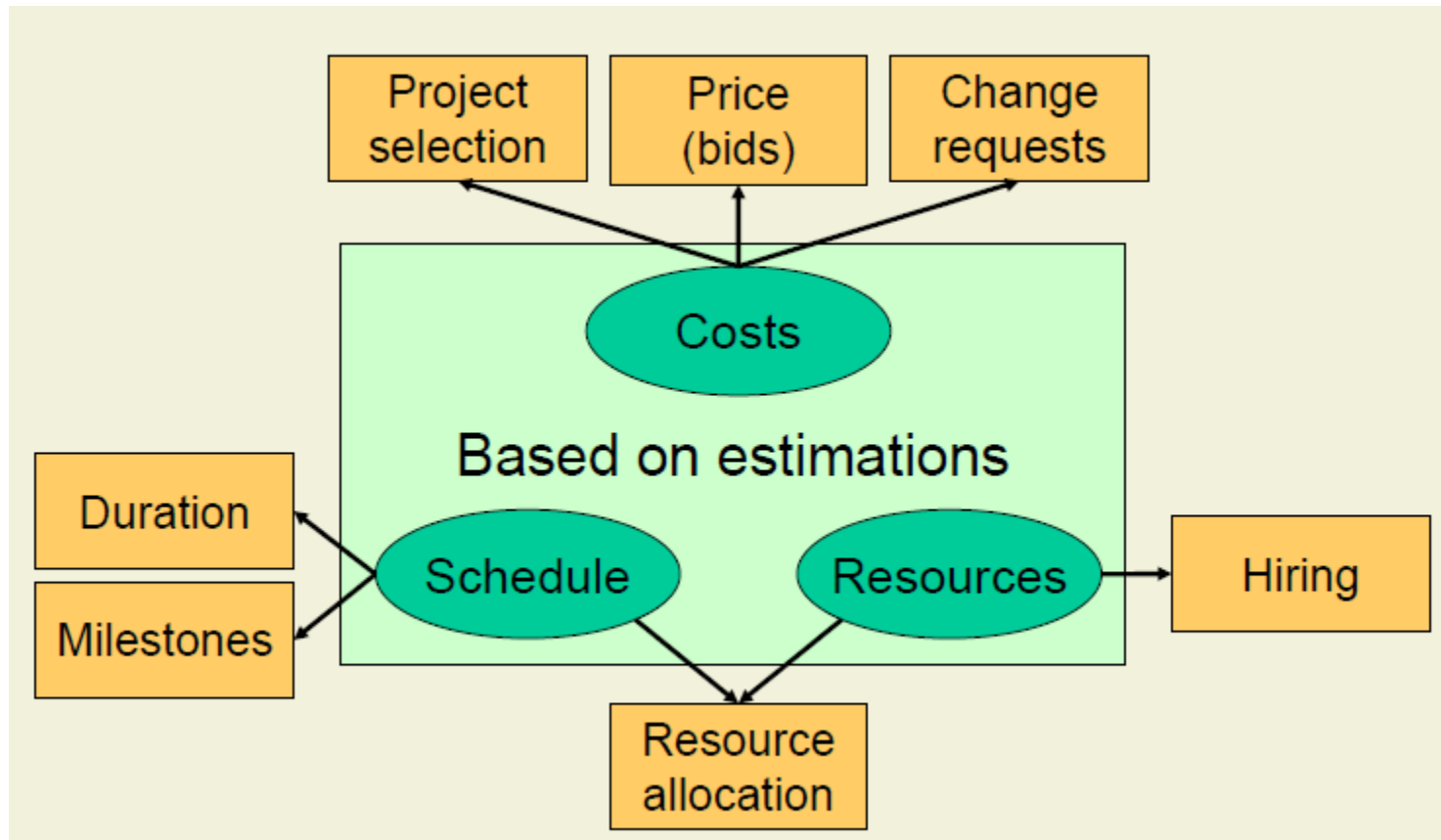
7.2个月后，提交系统

9.告诉客户以这个时间就能做到这个程度，而且有证据。项目组的程序员已经因不堪重负而病倒若干、住院若干、辞职若干。甚至于殉职若干

10.先接受（客户：没有办法，投入很多了，不能够不做）

11.为改进有新的项目

Why Estimations?

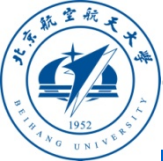




Estimation

- “The single most important task of a project: setting realistic expectations.

Unrealistic expectations based on inaccurate estimates are the single largest cause of software failure.”



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 clutch?”
 - “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 clutch?”
 - “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*



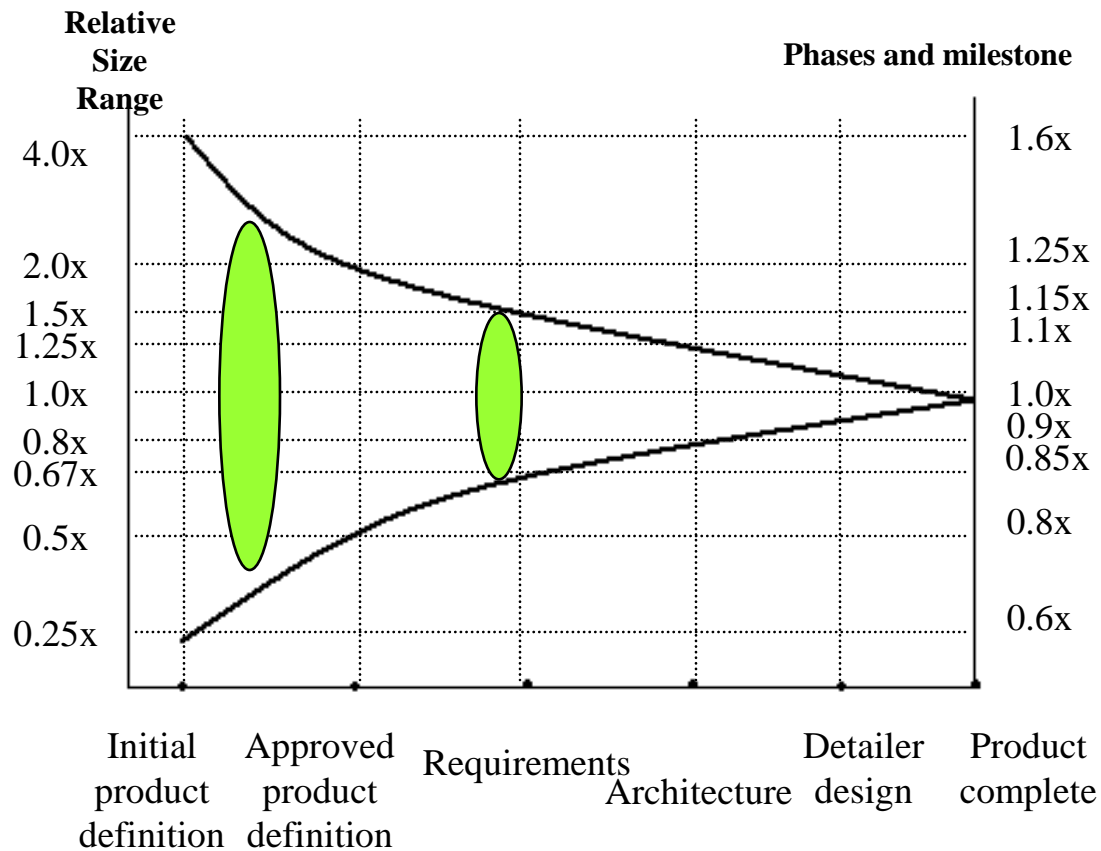
Software Estimation Woes

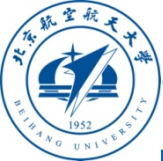
- Estimation woes 4 – **Self-fulfilling prophecies**
 - “Hmm, what do you know, the clutch seems simpler than I thought this time. This one could actually be fixed in under four hours!”
 - So that means I have six extra hours. Let’s see if I can also fix that rattling sound that’s been bugging me. And 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*

Estimate

- Very difficult to do, but needed often
- Estimate is important to plan a project, and also significant to control a project!
- Remember, an “exact estimate” is an oxymoron!
- The size of a software system can only be known accurately when it is finished
- As the development process progresses then the size estimate becomes more accurate

Estimate uncertainty





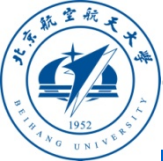
Accurate Estimation

- **Estimation is a specialized technical skill**
- **Treat estimation as a mini-project**
- **Plan to re-estimate periodically**



When to estimate

- **Created, used or refined during**
 - Feasibility study and/or SOW
 - Proposals/Requirements
 - Vendor and sub-contractor evaluation
 - Project planning (iteratively)
 - **NOTE: Not all of these steps are always explicitly performed**



Common questions during estimation

- Unrealistic estimation based on the price
- No time to estimation
- Poorly stated requirements or untrue requirement
- No history data
- Some factors not considered
- software developers are romantics at heart!



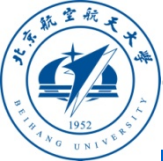
Common questions during estimation

- Some factors not considered
 - **Usability and stability of development environment**
 - **Competence and experience of team**
 - **stability of team**
 - **Mature of the development process**
 - **Reused software**
 - **The estimation of software created to reuse**
 - **The degree of automatization**
 - Huge variability in individual and team performances



| 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 16 |





Effect of Estimation Accuracy

● overestimation:

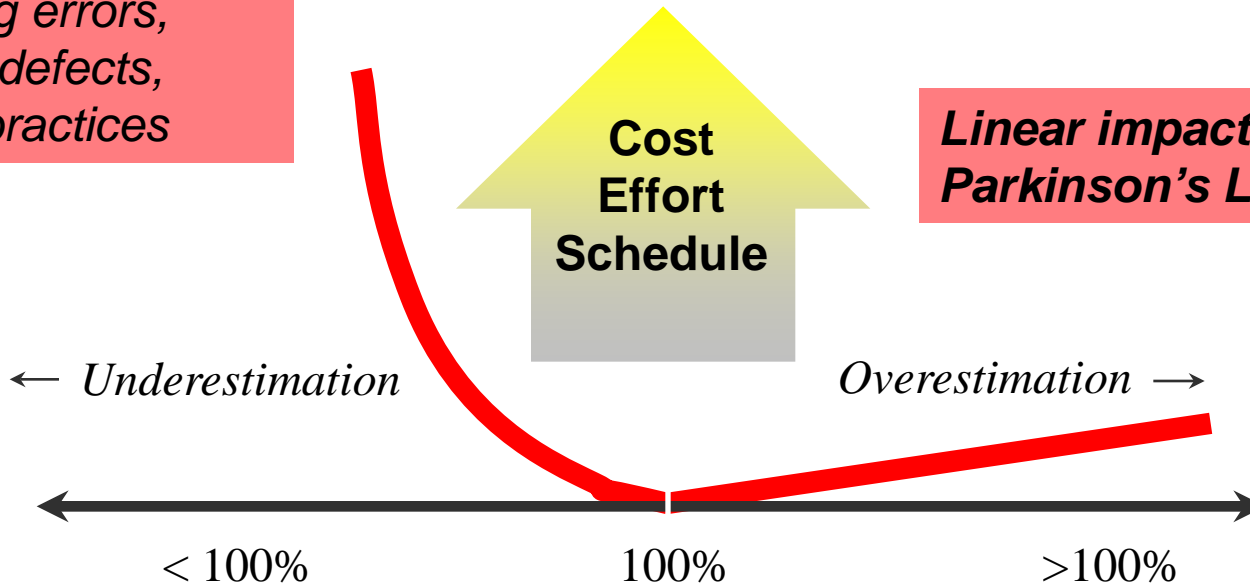
- Parkinson's Law: Work expands to fill the time available .

● underestimation: quality

- Weinberg' Law: If a system is not required to be reliable, then it can satisfy any other goals.

Effect of Estimation Accuracy

Non-linear impact due to planning errors, upstream defects, high-risk practices



Linear impact due to Parkinson's Law

Target as a Percentage of Nominal Estimate

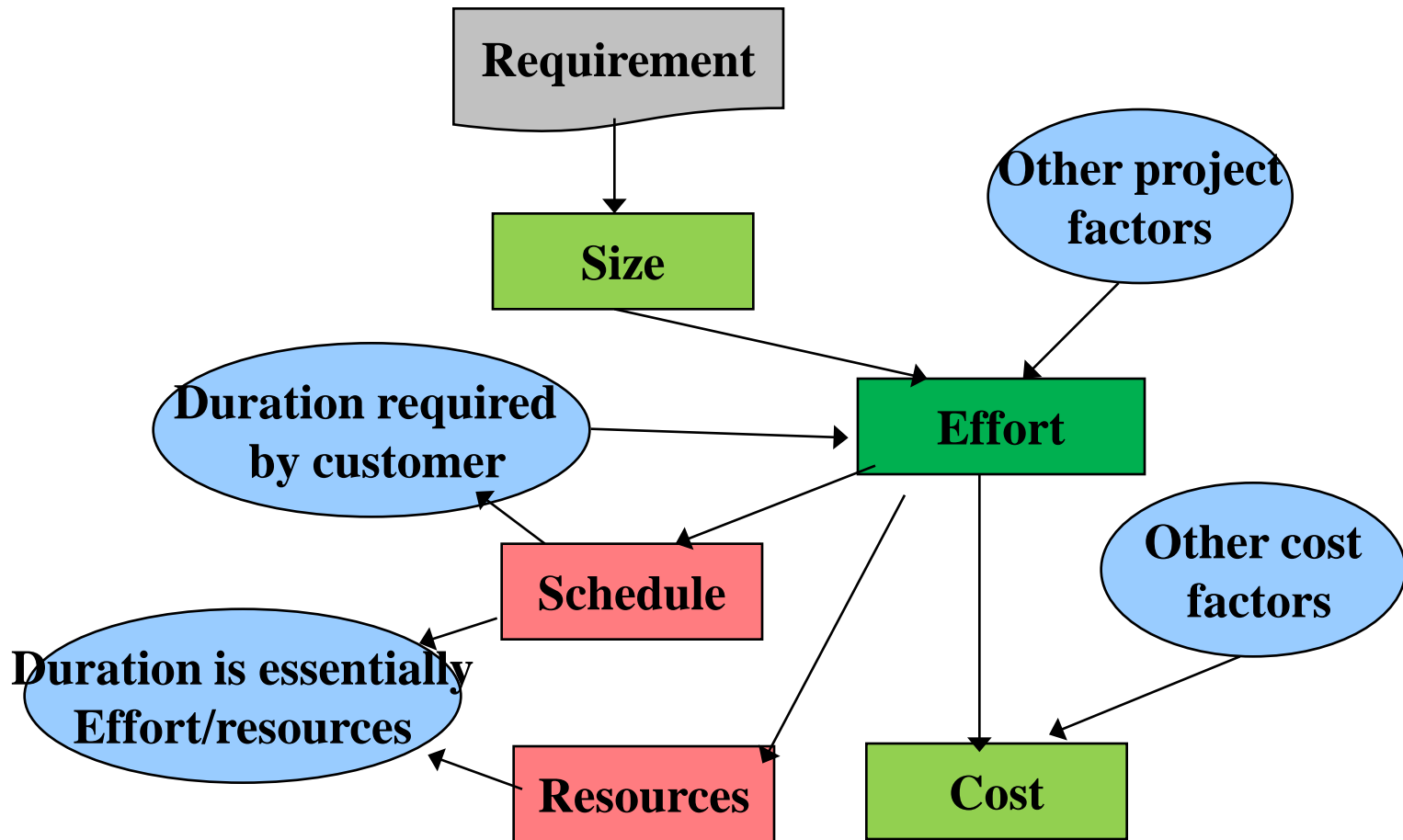


Estimate

- **Basic process**
 - Estimate the **size** of the product
 - Estimate the **effort** (man-months)
 - Estimate the **cost and schedule**
- **Unit of size: LOC、FP**
- **Unit of effort: man-months**
- **Unit of cost: yuan、Dollars**



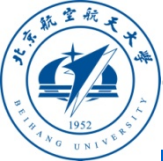
Estimations in Software Projects





Software productivity

- A measure of the rate at which individual engineers involved in software development produce software and associated documentation.
- Not quality-oriented although quality assurance is a factor in productivity assessment.
- Essentially, we want to measure useful functionality produced per time unit.

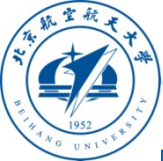


Productivity measures

- **Size related measures** based on some output from the software process. This may be lines of delivered source code, object code instructions, etc.
- **Function-related measures** based on an estimate of the functionality of the delivered software. Function-points are the best known of this type of measure.

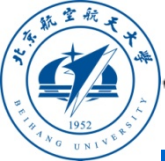
Productivity

- Productivity = **size/effort**
- The factors influenced the Productivity
 - people: the size and experience of the development organization.
 - problem: the complexity of the problem.
 - process: technique, programming language and reviewing process.
 - environment: performance and reliability of the computer system.
 - resource: tool

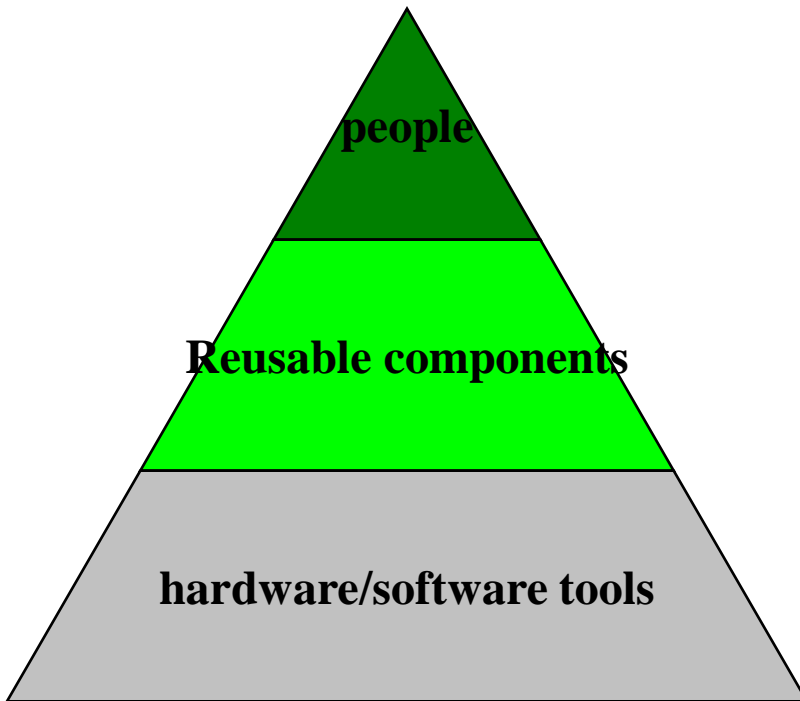


The steps of estimation

- 1.define the scope
- 2.estimate the usable resources
- 3. estimate the effort
- 4. estimate the cost



Estimate the usable resources



The describe of resource

The describe of usability

when need the resource

How long is the resource used?

● The process of estimation

● Input:

- **SOW、WBS**
- **Unit price of resource**
- **History data**
- **Learning curve**

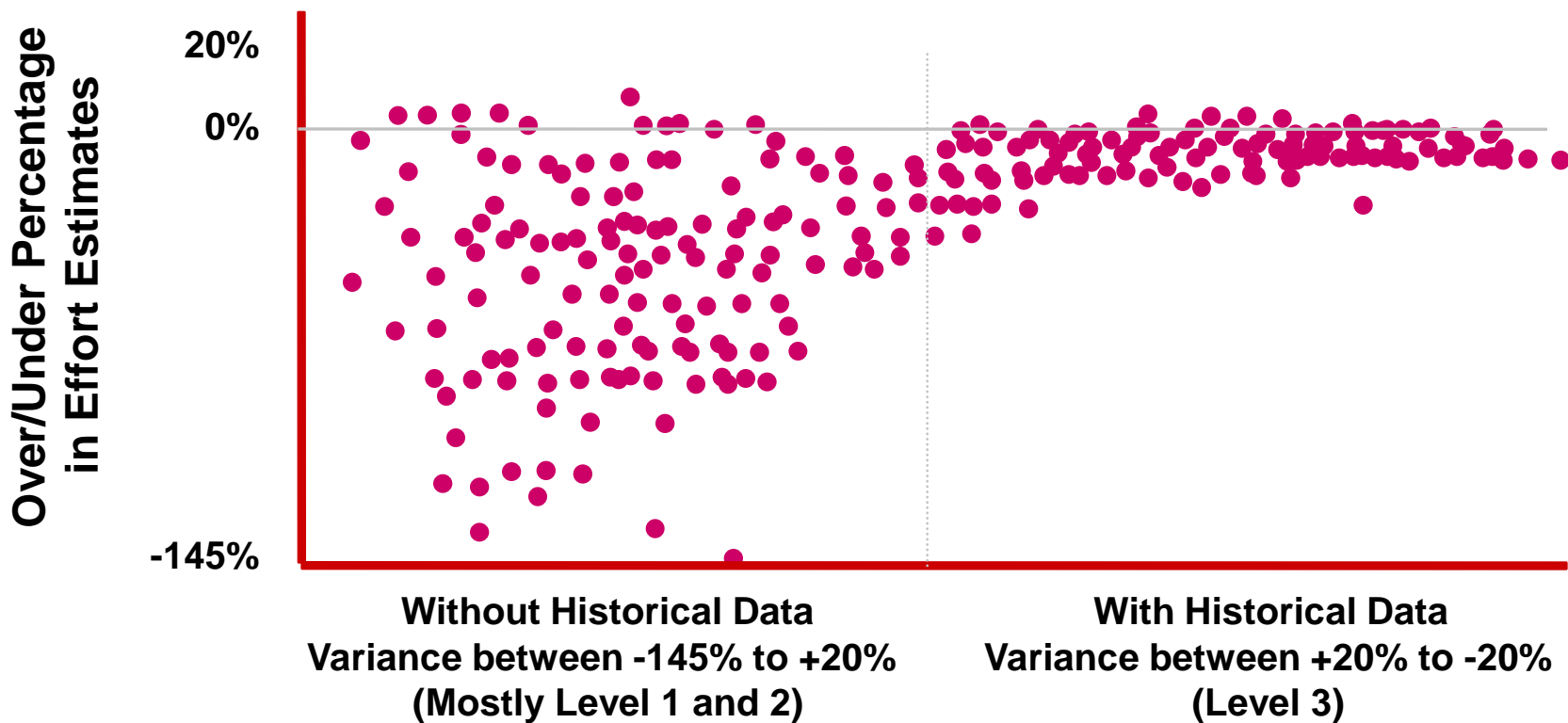
● process: estimation methods

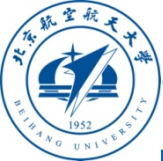
● Output:

- **The document of estimation**



历史数据





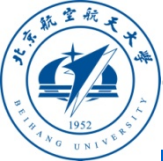
The document of estimation

- Effort and cost
- SOW, WBS
- Assumptions
- error



● outline

- Estimation overview
- Effort Estimation techniques
- Cost Estimation techniques
- Error of estimation



● Estimation Methodologies

- Lines of Code
- Function Point Analysis
- Object Points
- Bottom-Up estimation
- Analogy method
- Expert Judgment
- Parametric or Algorithmic estimation
- simple method(Wideband Delphi)



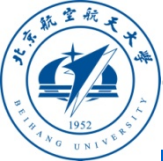
LOC方法

- **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。



LOC方法

- 例如，某软件公司统计发现该公司每一万行C语言源代码形成的源文件（.c和.h文件）约为**250K**。某项目的源文件大小为**3.75M**，则可估计该项目源代码大约为**15万行**，该项目累计投入工作量为**240人月**，每人月费用为**10000元**（包括人均工资、福利、办公费用公摊等），则该项目中**1LOC**的价值为：
 - $(240 \times 10000) / 150000 = 16 \text{元/LOC}$
- 该项目的人月均代码行数为：
 - $150000 / 240 = 625 \text{LOC/人月}$



Code-based Estimates

LOC Advantages

- Commonly understood metric
- Permits specific comparison
- Actuals easily measured

LOC Disadvantages

- Difficult to estimate early in cycle
- Counts vary by language
- Many costs not considered (ex: requirements)
- Code generators produce excess code
- stimulates programmers to write lots of code
- system-oriented, not user-oriented

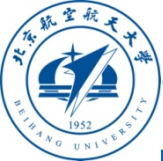


Productivity comparisons

- The lower level the language, the more productive the programmer
 - The same functionality takes more code to implement in a lower-level language than in a high-level language.
- The more verbose the programmer, the higher the productivity
 - Measures of productivity based on lines of code suggest that programmers who write verbose code are more productive than programmers who write compact code.

LOC Estimate Issues

- How do you know how many in advance?
- What about different languages?
- What about programmer style?
- avg. programmer productivity: 100-200 LOC/d



Function Points

- Software size measured by number & complexity of functions it performs
- More methodical than LOC counts
- House analogy
 - House's Square meter \sim Software LOC
 - # Bedrooms & Baths \sim Function points
 - Former is size only, latter is size & function



● Function Points

● History

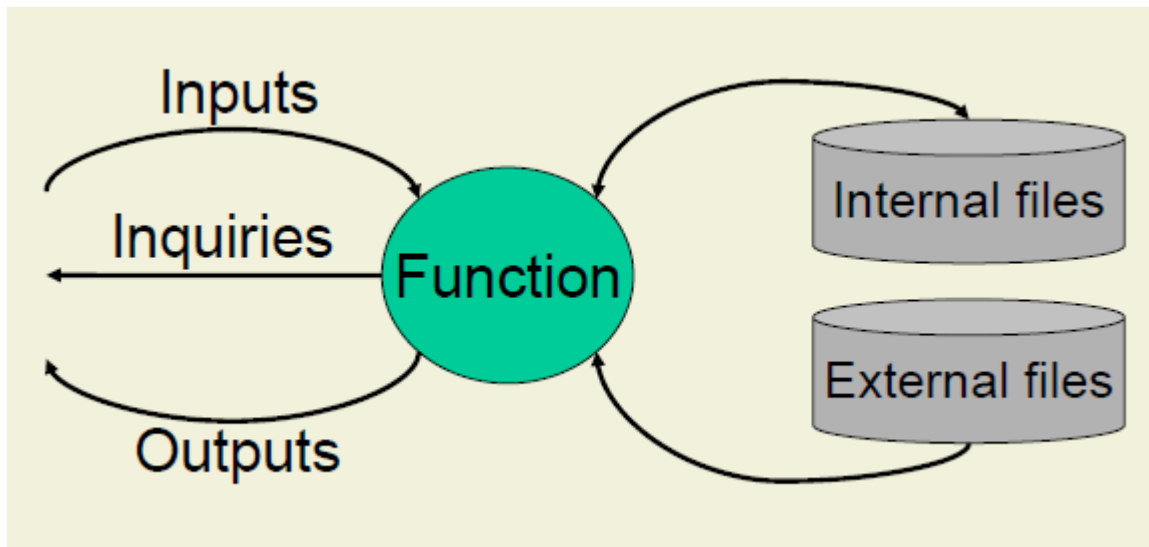
- Non-code oriented size measure
- Developed by IBM (A. Albrecht) in 1979
- Now in use by more than 500 organizations world-wide

● What are they?

- 5 weighted functionality types
- 14 complexity factors

5 functions

- Size is estimated based on **requirements**





functions

● Inputs

- Each input that provides distinct application oriented data to the software is counted. e.g. Forms, dialogs, messages, XML documents

● Outputs

- Each output that provides application oriented information to the user is counted. Individual data items within a report are not counted separately. e.g. Web pages, reports, graphs, messages, XML documents

● Inquiries (input/output combinations)

- This is an on-line input that results in the generation of some response

● Logical internal files (controlled by the program)

- Each master file is counted, e.g. Tables, views or files in database

● External files (controlled by other programs)

- Each interface that is used to transmit information to another system is counted.

Function Point Process

- 1. Count # of biz functions per category
 - Categories: outputs, inputs, db inquiries, files or data structures, and interfaces
- 2. Establish Complexity Factor for each and apply
 - Simple, Average, Complex
 - Set a weighting multiplier for each (0->15)
 - This results in the “unadjusted function-point total”(UFC)
- 3. Compute an “TCF (Technical Complexity Factor) / influence multiplier” and apply
 - It ranges from 0.65 to 1.35; is based on 14 factors
- 4. Results in “function point total”
 - $FP = UFC * TCF$
 - This can be used in comparative estimates



UFC

Weigh each function according to complexity

| | 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

Technical Complexity Factor

- Data communications
 - Distributed processing
 - Performance
 - Heavy use
 - Transaction rate
 - Online data entry
 - Complex interface
 - Online data update
 - Complex processing
 - Reusability
 - Installation ease
 - Operational ease
 - Multiple sites
 - Facilitate change
- Rate each element from 0 – 5
 - 0: no influence
 - 1: insignificant influence
 - 2: moderate influence
 - 3: average influence
 - 4: significant influence
 - 5: strong influence



TCF

- $TCF = 0.65 + 0.01 * (\text{Sum}(F_i)), i = 1, 2, 3, \dots, 14,$
- Each F_i is between 0 and 5, so TCF
Varies between 0.65 and 1.35

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：

$$\text{TCF} = 0.65 + 0.01 * (14 * 3) = 1.07$$

$$\text{则：FP} = \text{UFC} * \text{TCF} = 301 * 1.07 = 322$$



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：报告，1
- 内部逻辑文件：会计输入文件，1
- 外部接口文件：工资文件，人员文件，课程文件，会计输入文件，4₄₅



功能点的简单算法



Typical weigh:

- input 4, output 5 , inquire 4, internal file 10, external file 10

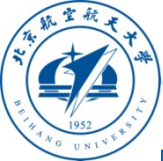


- 估计者根据对复杂度的判断，总数可以用**+25%、0、或-25%**调整



功能点转化为工作量

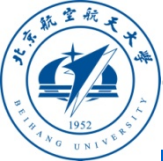
- 对于原来的项目，计算生产率：
 - $\text{生产率} = \text{功能点数目} / \text{工作量 (人日)}$
 - 则，对于新项目，功能点计算出来后，工作量为：
 - $\text{工作量} = \text{功能点数目} / \text{生产率}$



● Application Assumptions

- Requirements are known ahead of calculation
- Focus on the complete product from viewpoint of customer
- Experience FC \rightarrow PM table exists
- Actual efforts can be determined and FC \rightarrow PM table is updated accordingly
- Standardized categorization and classification procedures are followed
- Adaptable to experience in the organization





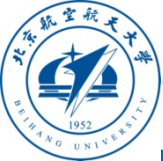
Object points

- Object points (alternatively named application points) are an alternative function-related measure to function points when 4GLs or similar languages are used for development.
- Object points are NOT the same as object classes.
- The number of object points in a program is a weighted estimate of
 - The number of separate screens that are displayed;
 - The number of reports that are produced by the system;
 - The number of program modules that must be developed to supplement the database code;



Object point estimation

- Object points are easier to estimate from a specification than function points as they are simply concerned with screens, reports and programming language modules.
- They can therefore be estimated at a fairly early point in the development process. At this stage, it is very difficult to estimate the number of lines of code in a system.
- In object points, productivity has been measured between 4 and 50 object points/month depending on tool support and developer capability.



Object point

屏幕对象点

| | 数据表的数目和来源 | | |
|--------|----------------------------|------------------------------|----------------------------|
| 包含的视图数 | 总计<4 (<2个服务器 <3个客户机) | 总计<8 (<3个服务器 <3-5个客户机) | 总计>7 (>3个服务器 >5个客户机) |
| <3 | 简单的 | 简单的 | 中等的 |
| 3-7 | 简单的 | 中等的 | 困难的 |
| >7 | 中等的 | 困难的 | 困难的 |

对象点复杂度权重

| | 简单 | 中等 | 困难 |
|-------|----|----|----|
| 屏幕 | 1 | 2 | 3 |
| 报表 | 2 | 5 | 8 |
| 3GL构件 | - | - | 10 |

报表对象点

| | 数据表的数目和来源 | | |
|--------|----------------------------|------------------------------|----------------------------|
| 包含的部分数 | 总计<4 (<2个服务器 <3个客户机) | 总计<8 (<3个服务器 <3-5个客户机) | 总计>7 (>3个服务器 >5个客户机) |
| <2 | 简单的 | 简单的 | 中等的 ₅₁ |
| 2或3 | 简单的 | 中等的 | 困难的 |
| >3 | 中等的 | 困难的 | 困难的 |

Object point--effort

- 首先考虑已经存在的对象应该排除在工作量计算内。即计算新的对象点（NOP）。
 - 比如一个应用程序包括840个对象点，但是20%可以通过使用现成的构件来提供，那么调整后新的对象点
$$NOP = 840 * 0.8 = 672$$

- 根据原来从事过的项目计算在不同情况下的项目的生产率，例如下表：

| 开发人员经验和能力 | 非常低 | 低 | 正常 | 高 | 非常高 |
|-----------|-----|---|----|----|-----|
| 生产率 | 4 | 7 | 13 | 25 | 50 |

- 假定开发者的经验和工具使用都是一般性的，则工作量为 $672 / 13 = 52$ 个月



自下而上法

- 利用**WBS**进行估计
- 结合**PERT**方法进行估算
- 优点：如果活动划分的很好就容易估算
- 缺点：活动本身很费时间，而且很多细节的活动可能没有想到，也缺少活动集成的工作量

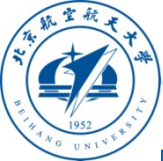
Bottom-up Estimation

- Create WBS
- Estimating the effort for **individual work packages**, Add from the bottom-up
- Advantages
 - Usable when the architecture of the system is known and components identified.
 - This can be an accurate method if the system has been designed in detail.
- Disadvantages
 - Specific activities not always known
 - More time consuming
 - Underestimation because effort does not grow linearly (due to complexity, etc.)
 - It may underestimate the costs of system level activities such as integration and documentation.



Program Evaluation and Review Technique

- Goal: Manage probabilities with simple statistics
- Approach: Ask several experts for three estimates
 - **O**ptimistic, **L**ikely (mode), and **P**essimistic
- Important formulas
 - Mean $M = (O + 4 \times L + P) / 6$
 - Deviation $V = (P - O) / 6$
- Assumptions **advise: $(P - O) / L < 40\%$**
 - Project effort is normally distributed (more than 20 work packages)
 - Work package efforts are statistically independent (ignores single underlying cause of delay)



Estimation by analogy

- Comparison with similar projects or modules
- Analysis of differences
- Advantages
 - Simple, quicker and less expensive than other methods
 - can be done early in the project, e.g. during bidding and contract
 - May be accurate if project data available and people/tools the same
- Disadvantages:
 - Impossible if no comparable project has been tackled.
 - Needs systematically maintained effort/cost database

类比法中重用代码的估算

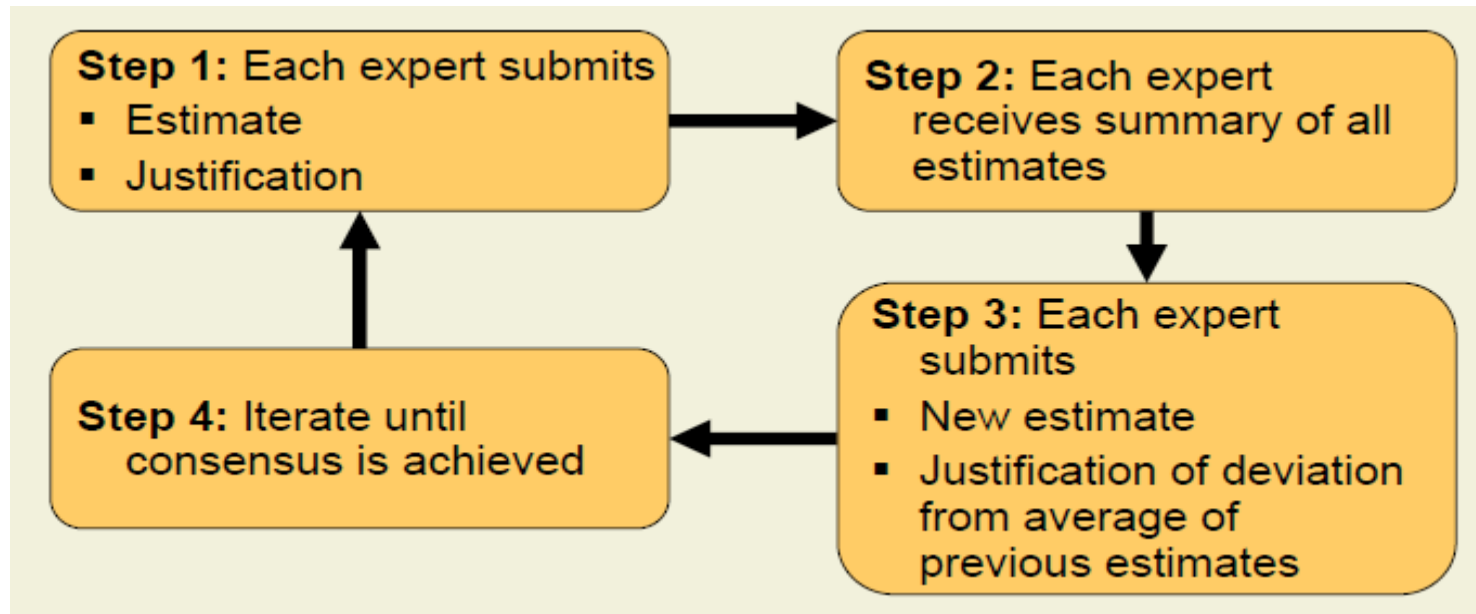
- 类比法中要解决可重用代码的估算问题。
- 一般估算出：新项目可重用的代码中需重新设计的代码百分比、需重新编码或修改的代码百分比以及需重新测试的代码百分比。则等价新代码行为：
$$\text{等价代码行} = [(\text{重新设计}\% + \text{重新编码}\% + \text{重新测试}\%)/3] \times \text{已有代码行}$$
- 比如：有10,000行代码，假定30%需要重新设计，50%需要重新编码，70%需要重新测试，那么其等价的代码行可以计算为：
- $$[(30\% + 50\% + 70\%)/3] \times 10,000 = 5,000$$
 等价代码行。
- 意即：重用这10000代码相当于编写5000代码⁵⁷行的工作量。



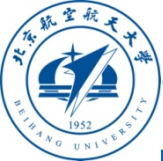
Expert judgment

- One or more experts in both software development and the application domain use their experience to predict software costs. Process iterates until some consensus is reached.
- Advantages: Relatively cheap estimation method. Can be accurate if experts have direct experience of similar systems
- Disadvantages: Very inaccurate if there are no experts!


Delphi Method



- Each expert must be independent and back-to-back.



Basic Algorithmic Form

 $\text{Effort} = A + B * (X)^C$

- where A, B, and C are empirically derived constants, E is the effort in person months, and X is the estimation variable, either in LOC or FP.

Algorithmic models

Based on LOC

- $E = 5.2 * (KLOC)^{0.91}$
- $E = 5.5 + 0.73 * (KLOC)^{1.16}$
- $E = 3.2 * (KLOC)^{1.05}$
- $E = 5.288 * (KLOC)^{1.047}$

Walston-Felix model

Bailey-Basili model

Boehm model (simple)

Doty model for $KLOC > 9$

Based on FP

- $E = -13.39 + 0.0545FP$ Albrecht and Gaffney model
- $E = 60.62 * 7.728 * 10^{(-8)} * FP^3$ Kemerer model
- $E = 585.7 + 5.12FP$ Maston、Barnett & Mellichamp

Empirical Model :COCOMO

- COCOMO: Constructive Cost Model,
- Long history from initial version published in 1981 (COCOMO-81) through various instantiations to COCOMO 2.
- Provide computational means for deriving S/W cost estimates as functions of variables (major cost drivers)
- Functions used contain constants derived from statistical analysis of data from past projects.
- Well-documented algorithmic model
- Public-domain tools exist
- Widely-used and evaluated



COCOMO Models

- Basic Model
 - Used for early rough, estimates of project cost, performance, and schedule
 - Effort or cost estimates as functions of size. Size is described LOC.
- Intermediate Model
 - computes software development effort as a function of program size and a set of fifteen "cost drivers"
 - Doesn't account for 10 - 20 % of cost (training, maintenance, etc)
- Detailed Model
 - incorporates all characteristics of the intermediate version with an assessment of the cost driver's impact on each step (analysis, design, etc.) of the software engineering process.

Basic COCOMO Model (COCOMO 81)

● Effort Equation :

● $\text{Effort} = c \times \text{size}^k$

- Effort: person-months(pm), 1pm = 19 pd = 152 ph = 1/12 py
- Size = 1000s Source Lines of Code (KSLOC)
- C and k are constants based on the developmental mode

COCOMO parameters

- C and k are given value according to the development mode.
- Three development environments (modes)
 - **Organic Mode**- -- a relatively small simple project in which small teams with good application experience work to a set of less than rigid requirements.
 - **Semi-detached Mode**- -- an intermediate project in which teams with mixed experience must meet a mix of rigid and less than rigid requirements.
 - **Embedded Mode**- -- a project that must meet tight hardware, software, and operational constraints.



COCOMO parameters

| Development Mode | Project Characteristics | | | |
|------------------|-------------------------|------------|--------------------------|--|
| | Size | Innovation | Deadline/ constraints | Dev. Environment |
| Organic | Small | Little | Not tight | Stable |
| Semi-detached | Medium | Medium | Medium | Medium |
| Embedded | Large | Greater | Tight | Complex hardware/customer interfaces |

| type of project | c | k |
|-----------------|-----|------|
| organic | 2.4 | 1.05 |
| semi-detached | 3.0 | 1.12 |
| embedded | 3.6 | 1.20 |



COCOMO系数

Table F.6 shows a comparison of the actual work-months from Table 5.1 and the COCOMO estimates. It illustrates the need for the calibration of COCOMO to suit local conditions.

Table F.6 *Comparison of COCOMO estimates and actual effort*

| <i>SLOC</i> | <i>Actual (work-months)</i> | <i>COCOMO estimates</i> | <i>Difference (work months)</i> | <i>Difference (%)</i> |
|-------------|-----------------------------|-------------------------|---------------------------------|-----------------------|
| 6050 | 16.7 | 15.9 | -0.8 | -4.9 |
| 8363 | 22.6 | 22.3 | -0.3 | -1.2 |
| 13334 | 32.2 | 36.4 | 4.2 | 13.1 |
| 5942 | 3.9 | 15.6 | 11.7 | 299.7 |
| 3315 | 17.3 | 8.4 | -8.9 | -51.2 |
| 38988 | 67.7 | 112.4 | 44.7 | 66.0 |
| 38614 | 10.1 | 111.2 | 101.1 | 1001.5 |
| 12762 | 19.3 | 34.8 | 15.5 | 80.2 |
| 26500 | 59.5 | 74.9 | 15.4 | 25.9 |



Intermediate COCOMO Model

- Takes basic COCOMO as starting point
- Identifies personnel, product, computer and project attributes which affect cost
- Multiplies basic cost by attribute multipliers which may increase or decrease costs
- $\text{Effort} = \text{basic model} * \text{effort adjustment factor}$
- $\text{Effort} = (c \times \text{size}^k) * \text{dem}$
- $(c \times \text{size}^k)$ is called nominal effort
- Dem: development effort multiplier



dem

| attributes | abbreviation | factors |
|------------|--------------|---------------------------------|
| product | RELY | Reliability requirement |
| | DATA | Database size |
| | CPLX | Product complexity |
| computer | TIME | Execution time constraints |
| | STOR | Storage constraints |
| | VIRT | Virtual machine volatility |
| | TURN | Computer turnaround time |
| personnel | ACAP | Analyst capability |
| | AEXP | Application experience |
| | PCAP | Programmer capability |
| | VEXP | Virtual machine experience |
| | LEXP | Programming language experience |
| project | MODP | Modern programming practices |
| | TOOL | Software tools |
| | SCED | Required development schedule |

| | |
|------------------|-------------|
| Very low | 1.46 |
| low | 1.19 |
| nominal | 1.00 |
| high | 0.80 |
| Very high | 0.71 |

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？



练习

Table F.7 *Calculating the development multiplier*

| <i>Factor</i> | <i>Rating</i> | <i>Multiplier</i> |
|---------------|---------------|-------------------|
| ACAP | very high | 0.71 |
| AEXP | low | 1.13 |
| PCAP | high | 0.80 |
| VEXP | high | 0.90 |
| LEXP | low | 1.07 |

$$dem = 0.71 \times 1.13 \times 0.80 \times 0.90 \times 1.07 = 0.62$$

$$\text{final estimate} = 4 \text{ person-months} \times 0.62 = 2.48 \text{ staff months}$$



COCOMO Models

- **COCOMO 2 is divided into three stage models.**
- **Application composition model** - Used during the early stages of software engineering when the following are important
 - Prototyping of user interfaces
 - Consideration of software and system interaction
 - Assessment of performance
 - Evaluation of technology maturity
 - using object points method
- **Early design stage model** – Used once requirements have been stabilized and basic software architecture has been established
 - Using FP method, but uses 7 cost drivers
- **Post-architecture stage model** – Used during the construction of the software
 - like COCOMO, with updated set of 17 cost drivers and 5 size scale factors.



COCOMO II

- equation: $P_m = A \times \text{size}^{sf} \times em_1 \times em_2 \times \dots \times em_n$
 - P_m : “person-month” effort
 - A : constant (2.55)
 - Size : LOC
 - Sf : size scale exponent
 - $Sf = 1.01 + 0.01 \times \text{SUM}(wi)$
 - Wi : size scale factor
 - The each em is counted like that of dem.
 - At different phases, the em is different.



Exponent scale factors(Wi)

| | |
|------------------------------|--|
| Precedentedness | Reflects the previous experience of the organisation with this type of project. Very low means no previous experience, Extra high means that the organisation is completely familiar with this application domain. |
| Development flexibility | Reflects the degree of flexibility in the development process. Very low means a prescribed process is used; Extra high means that the client only sets general goals. |
| Architecture/risk resolution | Reflects the extent of risk analysis carried out. Very low means little analysis, Extra high means a complete a thorough risk analysis. |
| Team cohesion | Reflects how well the development team know each other and work together. Very low means very difficult interactions, Extra high means an integrated and effective team with no communication problems. |
| Process maturity | Reflects the process maturity of the organisation. The computation of this value depends on the CMM Maturity Questionnaire but an estimate can be achieved by subtracting the CMM process maturity level from 5. |



练习

- 对于某一个软件企业，一个新的项目的新颖性一般，因而在先前经验方面给**3**分，开发灵活性方面很高，因而给以**0**分，但是需求可能会变化得比较厉害，因而风险解决指数给**4**分，团队很融洽，给**1**分，但是过程不标准，因而过程成熟性给**4**分，请计算规模指数**sf**:



练习

Table F.8 *Exponent drivers and ratings for the project*

| <i>Exponent driver</i> | <i>rating</i> |
|------------------------------|---------------|
| Precedentedness | 3 |
| Development flexibility | 0 |
| Architecture/risk resolution | 4 |
| Team cohesion | 1 |
| Process maturity | 4 |
| Total | 12 |

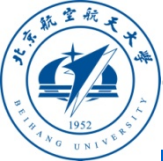
Therefore, $sf = 1.01 + 0.01 \times (3 + 0 + 4 + 1 + 4) = 1.13$



COCOMOII 中关于重用的处理

- 改编后的模块规模 = 原模块的LOC * AAF
- AAF: 调节因子
- $AAF = 0.4 * \text{设计修改的百分比} + 0.3 * \text{编程修改的百分比} + 0.3 * \text{集成修改的百分比}$





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 | | | |



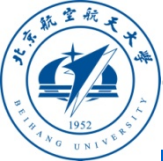
产品规模简单估算方法

- 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 \approx 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 \approx 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 \approx 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 | | | |



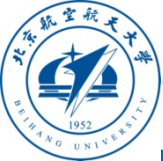
outline

- Estimation overview
- Effort Estimation techniques
- Cost Estimation techniques
- Error of estimation



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。”



Software cost components

- Hardware and software costs.
- Travel and training costs.
- Effort costs (the dominant factor in most projects)
 - The salaries of engineers involved in the project;
 - Social and insurance costs.
- Costs must take overheads into account
 - Costs of building, heating, lighting.
 - Costs of networking and communications.
 - Costs of shared facilities (e.g library, staff restaurant, etc.).



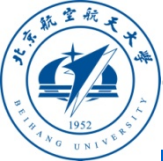
From Effort to Costs

- **Direct costs: Costs incurred for the benefit of a specific project**
 - Salaries of project staff
 - Equipment bought specifically for the project
 - Travel expenses
- **Indirect costs: Costs incurred for the joint benefit over multiple projects (“overhead”)**
 - Accounting, quality assurance department
 - Line management
 - Rooms, electricity, heating
- **Total cost=direct cost + indirect cost**



Unit Costs

- Projects have to budget for
 - Direct costs
 - A certain share of indirect costs
- Budgets are usually determined by using unit costs
 - Unit cost: Price per unit of a resource
- Examples
 - Loaded day rate for senior IT consultant: RMY 300
 - Loaded day rate for internal developer: RMY 150



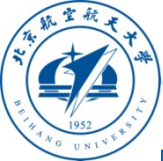
Costing and pricing

- Estimates are made to discover the cost, to the developer, of producing a software system.
- There is not a simple relationship between the development cost and the price charged to the customer.
- Broader organisational, economic, political and business considerations influence the price charged.



Software pricing factors

| | |
|---------------------------|--|
| Market opportunity | A development organisation may quote a low price because it wishes to move into a new segment of the software market. Accepting a low profit on one project may give the opportunity of more profit later. The experience gained may allow new products to be developed. |
| Cost estimate uncertainty | If an organisation is unsure of its cost estimate, it may increase its price by some contingency over and above its normal profit. |
| Contractual terms | A customer may be willing to allow the developer to retain ownership of the source code and reuse it in other projects. The price charged may then be less than if the software source code is handed over to the customer. |
| Requirements volatility | If the requirements are likely to change, an organisation may lower its price to win a contract. After the contract is awarded, high prices can be charged for changes to the requirements. |
| Financial health | Developers in financial difficulty may lower their price to gain a contract. It is better to make a smaller than normal profit or break even than to go out of business. |



From Costs to Prices

- The price is often based on the costs and a margin
- $\text{Price} = \text{Costs} / (1 - \text{Margin})$
- Example
 - Costs = RMY 1.000.000
 - Margin = 5%
 - Price = RMY 1.052.632
- Sometimes, $\text{Price} = \text{Costs} * (1 + \text{Margin})$
 - Example: RMY 1.050.000



软件成本的综合估算法

- 1. creat WBS, estimate effort for each task.
 Q_i (pm)
- 2. estimate each direct development cost: $E_i = Q_i * \text{unit person cost (unloaded rate)}$
 - direct development cost $= E_1 + \dots + E_n$
 - management cost $= \text{proportion parameter} * \text{direct development cost}$
 - proportion parameter range 20%-25%
- 3. estimate indirect cost : indirect cost = direct cost * indirect cost parameter



软件成本的综合估算法



4. estimate total cost :

- total cost = direct cost + indirect cost
= direct cost * (1+ indirect cost parameter)
= effort* unit person cost * (1+ indirect cost parameter)
= effort * cost parameter
- If there are some fixed price purchase, the price should be included in total cost.



● 软件项目快速成本估算方法1

● 只考虑人力成本的估算法

● Based on person-month:

- Cost of Project = Number of Person-Months *
Weighted average cost/per month
- Weighted average cost can be 2 to 3 times average salary.
- $C = E * \text{average salary} \times 3$ (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 $(24 \times 5000 \text{元}) \times 3 = 360000 \text{元}$



● 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



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 \times 功能块点数
- 改进公式：功能模块单价 \times 功能点数 \times 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： $(15000 \times 3) + (20000 \times 5) + (4000 \times 2) = 1530000$ 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： $(15000 \times 3) \times 2 + (20000 \times 5) \times 3 + (4000 \times 2) \times 1 = 398000$ 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



Case:校务通管理系统

| | 任务名称 | 工期 |
|----|------------|---------|
| 1 | ▢ 校务通管理系统 | 103 工作日 |
| 2 | ▢ 通用功能 | 31 工作日 |
| 3 | 电子课表 | 8 工作日 |
| 4 | 会议通知和公告 | 3 工作日 |
| 5 | 个人日记 | 5 工作日 |
| 6 | 通讯录 | 2 工作日 |
| 7 | 教师答疑 | 5 工作日 |
| 8 | 作业布置和批改 | 8 工作日 |
| 9 | ▢ 日常业务管理功能 | 72 工作日 |
| 10 | ⊕ 招生管理 | 26 工作日 |
| 15 | ⊕ 学生日常管理 | 10 工作日 |
| 20 | ⊕ 教务管理 | 31 工作日 |
| 28 | 教师备课系统(外包) | 1 工作日 |
| 29 | 资源库系统(外包) | 1 工作日 |
| 30 | 网上考试(外购) | 1 工作日 |
| 31 | 聊天室(已存在) | 1 工作日 |
| 32 | 论坛(已存在) | 1 工作日 |

| | 任务名称 | 工期 |
|----|------------|--------|
| 9 | ▢ 日常业务管理功能 | 72 工作日 |
| 10 | ▢ 招生管理 | 26 工作日 |
| 11 | 报名 | 3 工作日 |
| 12 | 招生 | 5 工作日 |
| 13 | 分班 | 10 工作日 |
| 14 | 统计查询 | 8 工作日 |
| 15 | ▢ 学生日常管理 | 10 工作日 |
| 16 | 学生档案管理 | 4 工作日 |
| 17 | 学生考勤管理 | 2 工作日 |
| 18 | 学生奖惩 | 2 工作日 |
| 19 | 学生变动 | 2 工作日 |
| 20 | ▢ 教务管理 | 31 工作日 |
| 21 | 教师日常管理 | 2 工作日 |
| 22 | 年级、班级设置 | 4 工作日 |
| 23 | 学科设置 | 2 工作日 |
| 24 | 年级、班级课程 | 5 工作日 |
| 25 | 排课表 | 9 工作日 |
| 26 | 考试管理 | 4 工作日 |
| 27 | 评价 | 5 工作日 |
| 28 | 教师备课系统(外包) | 1 工作日 |
| 29 | 资源库系统(外包) | 1 工作日 |
| 30 | 网上考试(外购) | 1 工作日 |
| 31 | 聊天室(已存在) | 1 工作日 |
| 32 | 论坛(已存在) | 1 工作日 |



case

- 已知：第3、4、5个模块外包或外购的价格分别为**5000元**、**3000元**、**3000元**。
- 对项目的功能进行**WBS**分解。因为**WBS**主要是针对开发任务的分解，管理任务和质量任务可以通过计算开发任务得出。因此根据以往的经验，管理任务和质量任务=**20%*开发任务**。



case

● 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



case

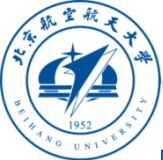
● 计算间接成本:

- 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**



case

- 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。



Estimation methods - Summary

- Each method has strengths and weaknesses
- Estimation should be based on several methods
- If these do not return approximately the same result, there is insufficient information available
- Some action should be taken to find out more in order to make more accurate estimates



● outline

- 关于估算的概念
- 软件项目估算方法
- 软件项目成本估算综述和案例
- 软件项目估算的误差



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
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

Construx Estimate



论文作业要求

- 论文题目自定
- 格式要求：按照一篇正规的paper格式撰写
- 论文要求：
 - 用自己的实践与理解，结合课程知识，阐述如何进行有效的软件项目管理，就可以就某个方面阐述
 - 必须有案例说明。
 - 个人完成，每人提交论文
 - 字数：（2500-5000）
 - 提交时间：12月30日之前,网上默认文件夹下，不要发老师邮箱