

High performance. Delivered.

**TU München
Applied Software Engineering
Prof. Bernd Brügge
Software Engineering II, SS 2006**

Guest Lecture: Estimation
Wolfgang Behr, Accenture

Miscellaneous: Exercise on Estimation

- **Advanced Developer Telecommuting Project**
 - Problem Statement from client (*last week*)
 - Project Agreement from client (*last week*)
- **Today's Handouts**
 - Estimation Handout
 - Estimating Factors
 - Sample Solution of WBS exercise
- **Preparation for Tomorrow's Exercise:**
 - Read the "Estimation Handout"
 - Prepare a draft solution **based on the WBS Sample Solution**
 - Required Hardware/Software: Laptop with Microsoft Excel
- **Tomorrow's Exercise:**
 - Consolidate your draft solution within your team
 - Present your team solution
 - Send e-mail to aljadiri@in.tum.de with: (*deadline May 25, 2006*)
 - *Team solution, Team members' names*



Objectives for Today

Build an understanding of...

- Importance of estimations
- Different estimation approaches (initial situation, expectations, top-down versus bottom-up...)
- Advantages and disadvantages of different approaches
- Common pitfalls

Importance of Estimations

- During the planning phase of a project, a first guess about cost and time is necessary
- Estimations are often the basis for the decision to start a project
- Estimations are the foundation for project planning and for further actions

→ Estimating is one of the core tasks of project management !

Challenges

- Incomplete knowledge about:
 - Project scope and changes
 - Prospective resources and staffing
 - Technical and organizational environment
 - Infrastructure
 - Feasibility of functional requirements
- Comparability of projects in case of new or changing technologies, staff, methodologies
- Learning curve problem
- Different expectations towards project manager.

Problems with Estimations

- Estimation results (effort and time) are almost always too high (for political / human reasons) and have to be adjusted in a structured and careful manner
- Reviews by experts always necessary
- New technologies can make new parameters necessary
- Depending on the situation, multiple methods are to be used in combination.

Guiding Principles

- Documentation of assumptions about
 - Estimation methodology
 - Project scope, staffing, technology
- Definition of estimation accuracy
- Increasing accuracy with project phases
 - Example: Better estimation for implementation phase after object design is finished
- Reviews by experienced colleagues.

Components of an Estimation

This lecture

- **Cost**
 - Personnel (in person days or valued in personnel cost)
 - **Person day**: Effort of one person per working day
 - Material (PCs, software, tools etc.)
 - Extra costs (travel expenses etc.)
- **Development Time**
 - Project duration
 - Dependencies
- **Infrastructure**
 - Rooms, technical infrastructure, especially in offshore scenarios.

Lecture on Scheduling.

Estimating Development Time

Development time often estimated by formula

$$\text{Duration} = \text{Effort} / \text{People}$$

Problem with formula, because:

- A larger project team increases communication complexity which usually reduces productivity
- Therefore it is not possible to reduce **duration** arbitrarily by adding more **people** to a project
- In the lectures on organization and scheduling we take a more detailed look at this issue.

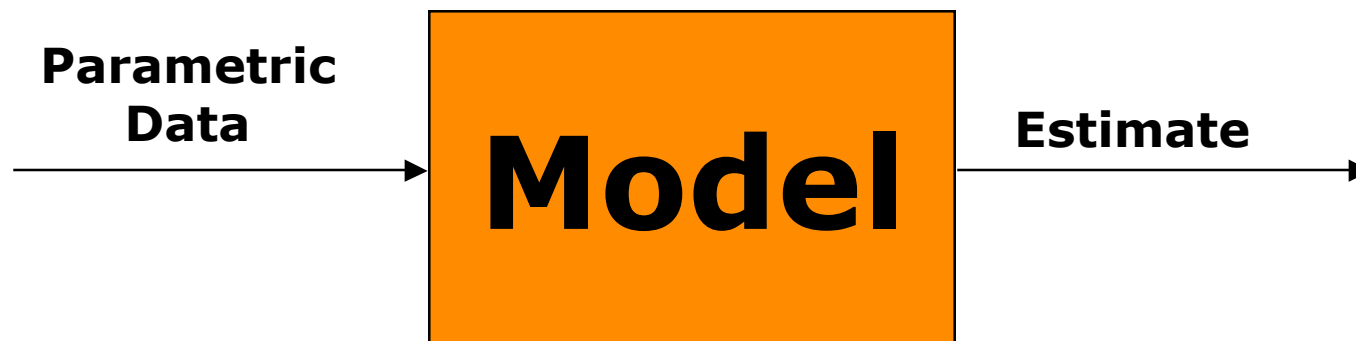
Estimating Personnel Cost

- Personnel type: Team leader, application domain expert, analyst, designer, programmer, tester...
- **Cost rate:** Cost per person per day
 - 2 alternatives for cost rate:
 - Single cost rate for all types (no differentiation necessary)
 - Assign different cost rates to different personnel types based on experience, qualification and skills
- **Personnel cost:** person days x cost rate.

Estimating Effort

- Most difficult part during project planning
 - Many planning tasks (especially project schedule) depend on determination of effort
- Basic principle:
 - Select an estimation model (or build one first)
 - Evaluate known information: size and project data, resources, software process, system components
 - Feed this information as parametric input data into the model
 - Model converts the input into estimates: effort, schedule, performance, cycle time.

Basic Use of Estimation Models



Examples:

Data Input

Size & Project Data

System Model

Software Process

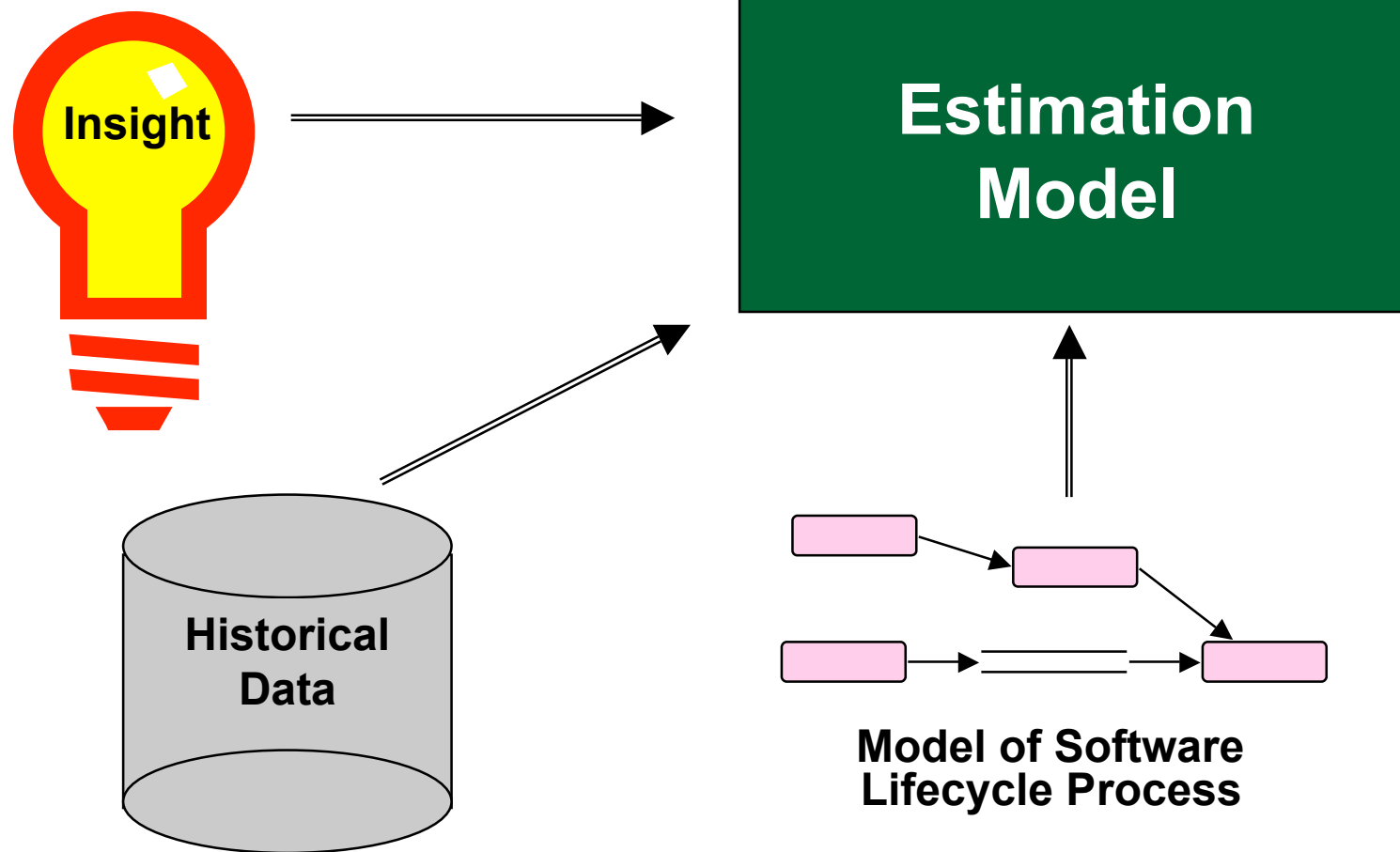
Estimate

Effort & Schedule

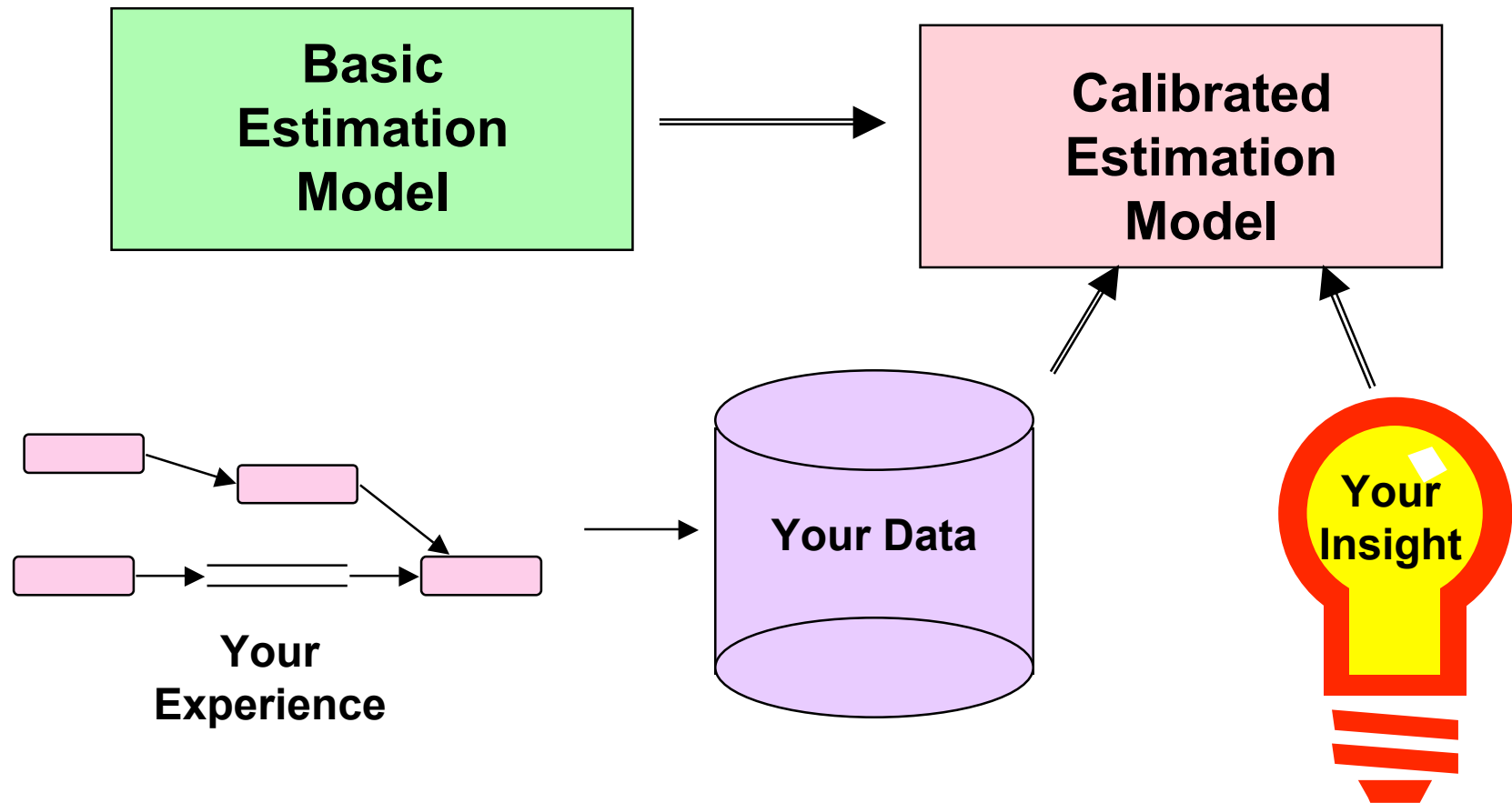
Performance

Cycle Time.

How do you Build an Estimating Model?



Calibrating an Estimation Model



Top-Down and Bottom-Up Estimation

- Two common approaches for estimations
 - Top-Down Approach
 - Estimate effort for the whole project
 - Breakdown to different project phases and work products
 - Bottom-Up Approach
 - Start with effort estimates for tasks on the lowest possible level
 - Aggregate the estimates until top activities are reached.

Top-Down versus Bottom-Up (cont'd)

- Top-Down Approach
 - Normally used in the planning phase when little information is available how to solve the problem
 - Based on experiences from similar projects
 - Not appropriate for project controlling (too high-level)
 - Risk add-ons usual
- Bottom-Up Approach
 - Normally used after activities are broken down the task level and estimates for the tasks are available
 - Result can be used for project controlling (detailed level)
 - Smaller risk add-ons
- Often a mixed approach with recurring estimation cycles is used.

Estimation Techniques

- Expert estimations
- Lines of code
- Function point analysis
- COCOMO
- Estimation Technique used by Accenture.

Expert Estimations



Expert Estimations

= Guess from experienced people

- No better than the participants
- Suitable for atypical projects
- Result justification difficult
- Important when no detailed estimation can be done (due to lacking information about scope).

Lines of Code

Lines of Code

- Traditional way for estimating application size
- Advantage: Easy to do
- Disadvantages:
 - Focus on developer's point of view
 - No standard definition for "Line of Code"
 - "You get what you measure": If the number of lines of code is the primary measure of productivity, programmers ignore opportunities of reuse
 - Multi-language environments: Hard to compare mixed language projects with single language projects

"The use of lines of code metrics for productivity should be regarded as professional malpractice" (Caspers Jones).

Function Point Analysis



Function Point Analysis

- Developed by Allen Albrecht, IBM Research, 1979
- Technique to determine size of software projects
 - Size is measured from a functional point of view
 - Estimates are based on functional requirements
- Albrecht originally used the technique to predict effort
 - Size is usually the primary driver of development effort
- Independent of
 - Implementation language and technology
 - Development methodology
 - Capability of the project team
- A top-down approach based on function types
 - Three steps: Plan the count, perform the count, estimate the effort.

Steps in Function Point Analysis

- Plan the count
 - Type of count: development, enhancement, application
 - Identify the counting boundary
 - Identify sources for counting information: software, documentation and/or expert
- Perform the count
 - Count data access functions
 - Count transaction functions
- Estimate the effort
 - Compute the unadjusted function points (UFP)
 - Compute the Value Added Factor (VAF)
 - Compute the adjusted Function Points (FA)
 - Compute the performance factor
 - Calculate the effort in person days.

Function Types

Data function types

of internal logical files (ILF)

of external interface files (EIF)

Transaction function types

of external input (EI)

of external output (EO)

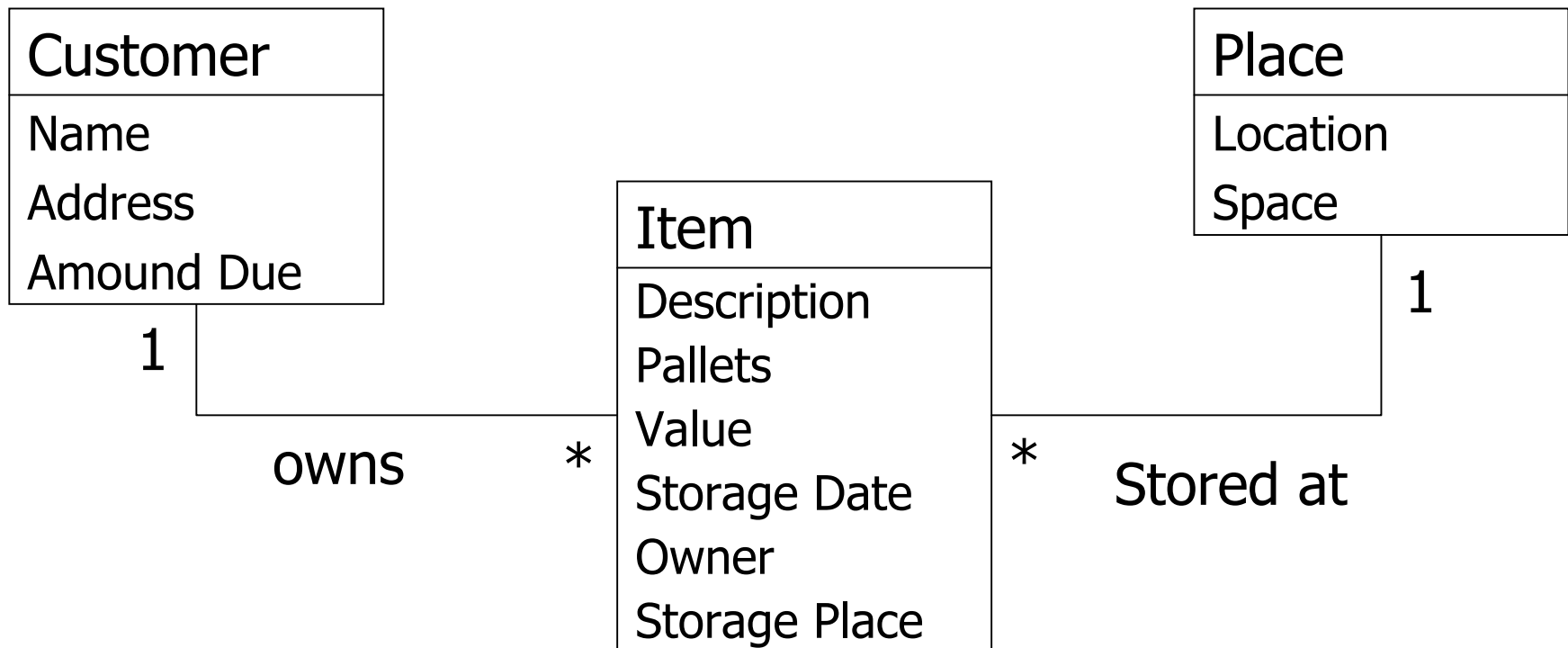
of external queries (EQ)

Calculate the UFP (unadjusted function points):

$$\text{UFP} = a \cdot \text{EI} + b \cdot \text{EO} + c \cdot \text{EQ} + d \cdot \text{ILF} + e \cdot \text{EIF}$$

a-f are so-called weight factors (see slide 28)

Object Model Example



Mapping Functions to Transaction Types

Add Customer

Change Customer

Delete Customer

Receive payment

Deposit Item

Retrieve Item

Add Place

Change Place Data

Delete Place

Print Customer item list

Print Bill

Print Item List

Query Customer

Query Customer's items

Query Places

Query Stored Items

External Inputs

External Outputs

External Inquiries

Calculate the Unadjusted Function Points

Function Type	Number	Weight Factors			=		
		simple	average	complex			
External Input (EI)	<input type="text"/>	x	3	4	6	=	<input type="text"/>
External Output (EO)	<input type="text"/>	x	4	5	7	=	<input type="text"/>
External Queries (EQ)	<input type="text"/>	x	3	4	6	=	<input type="text"/>
Internal Datasets (ILF)	<input type="text"/>	x	7	10	15	=	<input type="text"/>
Interfaces (EIF)	<input type="text"/>	x	5	7	10	=	<input type="text"/>
Unadjusted Function Points (UFP)						=	<input type="text"/>

14 General System Complexity Factors

- The unadjusted function points are adjusted with general system complexity (GSC) factors

GSC1: Reliable Backup & Recovery

GSC2: Use of Data Communication

GSC3: Use of Distributed Computing

GSC4: Performance

GSC5: Realization in heavily used configuration

GSC6: On-line data entry

GSC7: User Friendliness

GSC8: On-line data change

GSC9: Complex user interface

GSC10: Complex procedures

GSC11: Reuse

GSC12: Ease of installation

GSC13: Use at multiple sites

GSC14: Adaptability and flexibility

- Each of the GSC factors gets a value from 0 to 5.

Calculate the Effort

- After the GSC factors are determined, compute the Value Added Factor (VAF):

$$\text{VAF} = 0.65 + 0.01 * \sum_{i=1}^{14} \text{GSC}_i \quad \text{GSC}_i = 0,1,\dots,5$$

- Function Points =
Unadjusted Function Points * Value Added Factor
 - $\text{FP} = \text{UFP} \cdot \text{VAF}$
- **Performance factor**
 - PF = Number of function points that can be completed per day
- Effort = FP / PF

Advantages of Function Point Analysis

- Independent of implementation language and technology
- Estimates are based on design specification
 - Usually known before implementation tasks are known
- Users without technical knowledge can be integrated into the estimation process
 - Incorporation of experiences from different organizations
- Easy to learn
 - Limited time effort.

Disadvantages of Function Point Analysis

- Complete description of functions necessary
 - Often not the case in early project stages -> especially in iterative software processes
- Only complexity of specification is estimated
 - Implementation is often more relevant for estimation
- High uncertainty in calculating function points:
 - Weight factors are usually deducted from past experiences (environment, used technology and tools may be out-of-date in the current project)
- Does not measure the performance of people.

COCOMO

COCOMO (COnstructive COst MOdel)

- Developed by Barry Boehm in 1981
- Also called COCOMO I or Basic COCOMO
- Top-down approach to estimate cost, effort and schedule of software projects, based on size and complexity of projects
- Assumptions:
 - Derivability of effort by comparing finished projects (“COCOMO database”)
 - System requirements do not change during development
 - Exclusion of many efforts (for example administration, training, rollout, integration).

Advantages of COCOMO

- Appropriate for a quick, high-level estimation of project costs
- Fair results with smaller projects in a well known development environment
 - Assumes comparison with past projects is possible
- Covers all development activities (from analysis to testing)
- Intermediate COCOMO yields good results for projects on which the model is based.

Problems with COCOMO

- Model derived from the time when central batch processing was the standard
- Expert judgment required to determine the influencing factors and their values
- Experience shows that estimation results can deviate from actual effort by a factor of 4!
- Important project factors are not considered:
 - Skills of team members, travel, environmental factors, user interface quality, overhead cost.
- Out of date!

Estimation Technique used by Accenture

Estimation Technique used by Accenture

Uses both top-down and bottom-up elements

Consists of 9 steps:

1. Determine essential project characteristics
 - Infrastructure, technology, team skills, experience...
2. Use factors for fixed efforts and phases:
 - Often derived from already finished phases (step-by-step detailing of estimations)
 - Example:
 - 10% for project management
 - 10 % for infrastructure
 - 50% for testing efforts.

Estimation Technique used by Accenture (2)

3. Determine work products for the system to be developed (WBS)
5. Determine work product types (use case, user interface, subsystem, ...)
4. Assign a complexity factor to each of these work products
6. Define all necessary activities or tasks that need to be done to produce these work products
7. Assign effort estimates (in person days) to these tasks by using past experience
8. Aggregate the estimates to compute the overall project effort
9. Use add-ons (contingency and risk factors).

Example of Complexity and Multipliers (Non-exhaustive)

	Complexity	Type	Multiplier / Factor	Person Days
Requirements Elicitation				29
Function A	Low	Use Case	1	5
Function B	Medium	Use Case	1	8
Function C	High	Use Case	2	16
Implementation				39
Screen A	High	User interface	1	18
Screen B	Low	User interface	2	8
Batch Job A	Medium	Batch	1	8
Batch Job B	Low	Batch	1	5
Software Architecture			10 %	3,9

Prerequisites for Accenture's Technique

- Identical estimation approach for different projects necessary
- Lots of experience with estimating projects necessary in order to develop good parameters
- Multiple checks of top-down with bottom-up results and vice versa
- Post calculation after end of project important for improving estimation parameters.

Estimation Technique used by Accenture (3)

There are different estimating models available for different situations:

- Top-Down Model (initial estimate for early project phases)
- Bottom-Up Model (detailed estimating model)
- Custom development
- Packaged development (implementation of application software packages like SAP, Siebel, PeopleSoft, Oracle and any other packages)

Summary

- Estimation is often the basis for the decision to start, plan and manage a project
- Estimating software projects is an extremely difficult project management function
- All approaches depend very much on personal experiences
- If used properly, estimates can be a transparent way to discuss project effort and scope
- However,
 - Few software organizations have established formal estimation processes
 - Existing estimation techniques have lots of possibilities to influence the results - must be used with care.

Summary ctd

- Estimation results (effort and time) are almost always too high (for political / human reasons) and have to be adjusted in a structured and careful manner
- Review by experts (top-down versus bottom-up) are always necessary
- New technologies can make new parameters necessary
- Depending on the situation, multiple methods are to be used in combination.

Further Readings

- B. Boehm, Software Engineering Economics, Prentice-Hall, 1981
- B. Boehm, Software Cost Estimation With COCOMO II, Prentice Hall, 2000
- D. Garmus, D. Herron, Function Point Analysis: Measurement Practices for Successful Software Projects, Addison-Wesley, 2000
- International Function Point Users Group
 - <http://www.ifpug.org/publications/case.htm>
- C. Jones, Estimating Software Costs, 1998
- S. Whitemire, Object-Oriented Design Measurement, John Wiley, 1997

Online Availability of Estimation Tools

- Basic and Intermediate COCOMO I (JavaScript)
 - <http://www1.jsc.nasa.gov/bu2/COCOMO.html>
 - <http://ivs.cs.uni-magdeburg.de/sw-eng/us/java/COCOMO/index.shtml>
- COCOMO II (Unix, Windows and Java)
 - http://sunset.usc.edu/available_tools/index.html
- Function Point Calculator (Java)
 - <http://ivs.cs.uni-magdeburg.de/sw-eng/us/java/fp/>

Additional Slides



Function Point Examples

- UFP = 18
 - Sum of GSC factors = 0.22
 - VAF = 0.87
 - Adjusted FP = VAF * UFP = $0.87 * 18 \sim 16$
 - PF = 2
 - Effort = $16/2 = 8$ person days
-
- UFP = 18
 - Sum of GSC factors = .70
 - VAF = 1.35
 - Adjusted FP = VAF * UFP = $1.35 * 18 \sim 25$
 - PF = 1
 - Effort = $25/1 = 25$ person days

GSC Factors in Function Point Analysis

1. Data communications: How many communication facilities aid in the transfer or exchange of information with the system?
2. Distributed data processing: How are distributed data and processing functions handled?
3. Performance: Does the user require a specific response time or throughput?
4. Platform usage: How heavily used is the platform where the application will run?
5. Transaction rate: How frequently are transactions executed (daily, weekly, monthly)?
6. On-line data entry: What percentage of the information is entered On-Line?
7. End-user efficiency: Is the application designed for end-user efficiency?

GSC Factors in Function Point Analysis (cont'd)

8. On-line update: How many ILF's are updated on-line?
9. Complex processing: Does the application have extensive logical or mathematical processing?
10. Reusability: Will the application meet one or many user's needs?
11. Installation ease: How difficult is the conversion and installation?
12. Operational ease: How automated are start-up, backup and recovery procedures?
13. Multiple sites: Will the application be installed at multiple sites for multiple organizations?
14. Adaptability and flexibility: Is the application specifically designed to facilitate change?

Function Points: Example of a GSC Rating

GSC	Value(0-5)
Data communications	1
Distributed data processing	1
Performance	4
Heavily used configuration	0
Transaction rate	1
On-Line data entry	0
End-user efficiency	4
On-Line update	0
Complex processing	0
Reusability	3
Installation ease	4
Operational ease	4
Multiple sites	0
Adaptability and Flexibility	0
Total	22

Calculation of Effort in COCOMO

- Estimate number of instructions
 - KDSI = "Kilo Delivered Source Instructions"
- Determine project complexity parameters: A, B
 - Regression analysis, matching project data to equation
- 3 levels of difficulty that characterize projects
 - Simple project ("organic mode")
 - Semi-complex project ("semidetached mode")
 - Complex project ("embedded mode")
- Calculate effort
 - $\text{Effort} = A * \text{KDSI}^B$
- Also called Basic COCOMO

Calculation of Effort in Basic COCOMO

Formula: $\text{Effort} = A * KDSI^B$

- Effort is counted in person months: 152 productive hours (8 hours per day, 19 days/month, less weekends, holidays, etc.)
- A, B are constants based on the complexity of the project

Project Complexity	A	B
Simple	2.4	1.05
Semi-Complex	3.0	1.12
Complex	3.6	1.20

Calculation of Development Time

Basic formula: $T = C * \text{Effort}^D$

- T = Time to develop in months
- C, D = constants based on the complexity of the project
- Effort = Effort in person months (see slide before)

Project Complexity	C	D
Simple	2.5	0.38
Semi-Complex	2.5	0.35
Complex	2.5	0.32

Basic COCOMO Example

Volume = 30000 LOC = 30KLOC

Project type = Simple

Effort = $2.4 * (30)^{1.05} = 85$ PM

Development Time = $2.5 * (85)^{0.38} = 13.5$ months

=> Avg. staffing: $85/13.5 = 6.3$ persons

=> Avg. productivity: $30000/85 = 353$ LOC/PM

Compare: Semi-detached: 135 PM 13.9 M 9.7 persons

Embedded: 213 PM 13.9 M 15.3 persons

Cocomo: Example of Cost Driver Rating

Cost Driver	Very Low	Low	Nominal	High	Very High	Extra High
Required software reliability	0.75	0.88	1.00	1.15	1.40	-
Database size	-	0.94	1.00	1.08	1.16	-
Product Complexity	0.70	0.85	1.00	1.15	1.30	1.65
Execution Time Constraint	-	-	1.00	1.11	1.30	1.66
Main storage constraint	-	-	1.00	1.06	1.21	1.56
Virtual Storage volatility	-	0.87	1.00	1.15	1.30	-
Computer turn around time	-	0.87	1.00	1.07	1.15	-
Analyst capability	1.46	1.19	1.00	0.86	0.71	-
Applications experience	1.29	1.13	1.00	0.91	0.82	-
Programmer Capability	1.42	1.17	1.00	0.86	0.70	-
Virtual machine experience	1.21	1.10	1.00	0.90	-	-
Prog. language experience	1.14	1.07	1.00	0.95	-	-
Use of modern Practices	1.24	1.10	1.00	0.91	0.82	-
Use of software tools	1.24	1.10	1.00	0.91	0.83	-
Required schedule	1.23	1.08	1.00	1.04	1.10	-

Other COCOMO Models

- Intermediate COCOMO
 - 15 cost drivers yielding a multiplicative correction factor
 - Basic COCOMO is based on value of 1.00 for each of the cost drivers
- Detailed COCOMO
 - Multipliers depend on phase: Requirements; System Design; Detailed Design; Code and Unit Test; Integrate & Test; Maintenance

Steps in Intermediate COCOMO

- Basic COCOMO steps:
 - Estimate number of instructions
 - Determine project complexity parameters: A, B
 - Determine level of difficulty that characterizes the project
- New step:
 - Determine cost drivers
 - 15 cost drivers c_1, c_2, \dots, c_{15}
- Calculate effort
 - $\text{Effort} = A * KDSIB * c_1 * c_2 \dots * c_{15}$

Calculation of Effort in Intermediate COCOMO

Basic formula:

$$\text{Effort} = A * KDSI^B * C_1 * C_2 * \dots * C_{15}$$

Effort is measured in PM (person months, 152 productive hours (8 hours per day, 19 days/month, less weekends, holidays, etc.)

A, B are constants based on the complexity of the project

Project Complexity	A	B
Simple	2.4	1.05
Semi-Complex	3.0	1.12
Complex	3.6	1.20

Intermediate COCOMO: 15 Cost drivers

- Product Attributes
 - Required reliability
 - Database size
 - Product complexity
 - Computer Attributes
 - Execution Time constraint
 - Main storage constraint
 - Virtual Storage volatility
 - Turnaround time
 - Personnel Attributes
 - Analyst capability
 - Applications experience
 - Programmer capability
 - Virtual machine experience
 - Language experience
 - Project Attributes
 - Use of modern programming practices
 - Use of software tools
 - Required development schedule
- Rated on a qualitative scale between "very low" and "extra high"
 - ▶ Associated values are multiplied with each other.

COCOMO II

- Revision of COCOMO I in 1997
- Provides three models of increasing detail
 - Application Composition Model
 - Estimates for prototypes based on GUI builder tools and existing components
 - Early Design Model
 - Estimates before software architecture is defined
 - For system design phase, closest to original COCOMO, uses function points as size estimation
 - Post Architecture Model
 - Estimates once architecture is defined
 - For actual development phase and maintenance; Uses FPs or SLOC as size measure
- Estimator selects one of the three models based on current state of the project.

COCOMO II (cont'd)

- Targeted for iterative software lifecycle models
 - Boehm's spiral model
 - COCOMO I assumed a waterfall model
 - 30% design; 30% coding; 40% integration and test
- COCOMO II includes new costs drivers to deal with
 - Team experience
 - Developer skills
 - Distributed development
- COCOMO II includes new equations for reuse
 - Enables build vs. buy trade-offs

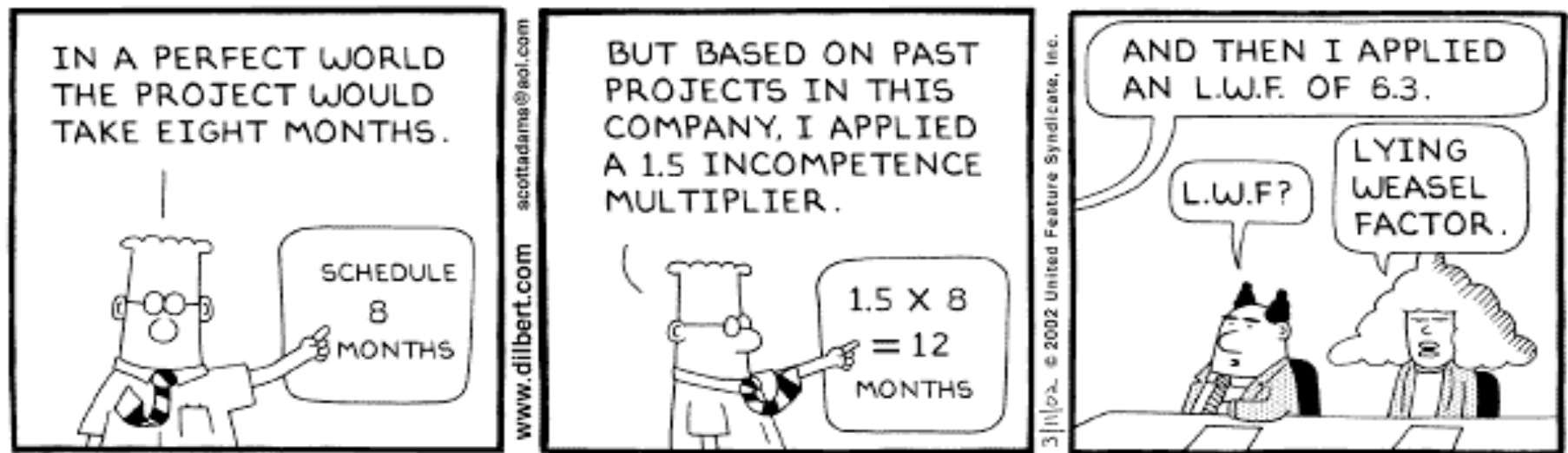
COCOMO II: Added Cost drivers

- Development flexibility
- Team cohesion
- Developed for reuse
- Precedent
- Architecture & risk resolution
- Personnel continuity
- Documentation match life cycle needs
- Multi-Site development.

How would you reply to this post?

- Post on www.ifpug.org:
 - “Our organization has just started to use function point analysis for estimation.
We have no internal metrics from the past we are not sure what productivity (hours/FP) to use for Cobol projects and for Java projects, in the financial industry.
Can anyone tell me their experiences with hours/FP for this platform or a place to go where to find this industry metrics?”

L.W.F Factor



Copyright © 2002 United Feature Syndicate, Inc.