Software Estimation

Lecturer: Ngo Huy Bien
Software Engineering Department
Faculty of Information Technology
VNUHCM - University of Science
Ho Chi Minh City, Vietnam
nhbien@fit.hcmus.edu.vn

Objectives

- To present *what* is software estimation
- To present *why* use software estimation
- To estimate software *size* using SLOC and Function Point
- To estimate cost and effort using *algorithmic models*
- To estimate software *size* using Use Case Point and Object Point
- To estimate cost and effort using *analogy method*

References

1. Steve McConnell. Software Estimation: Demystifying the Black Art. 2006.
2. Barry W. Boehm. Software Engineering Economics. 1983.
3. Daniel D. Galorath. Software Sizing, Estimation, and Risk Management. 2006.
4. Linda M. Laird, M. Carol Brennan. Software Measurement and Estimation A Practical Approach. 2006.
5. AJ Albrecht. Measuring Application Development Productivity. 1979.
6. CR Symons. Function Point Analysis: Difficulties and Improvements. 1988.
7. David Longstreet. Function Point Training Booklet. 2004.
8. Mohammed A. Shayib. Applied Statistics. 2013.
9. N.H. Bingham and John M. Fry. Regression -- Linear Models in Statistics. 2010.
10. Robert T. Futrell et al.. Quality Software Project Management. 2002.

Some Questions

- You have relatively good *requirements* (25 use cases).

- *How long* will it take to develop system.
- What is the *cost*?
- *How many people* do you need to develop system?
- How much is your team's *productivity*?

Software Estimation

Software estimation is the act of predicting *size*, *duration* and *cost* of a project.

Who needs it?

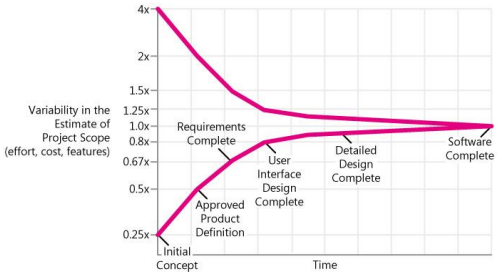
- Project Managers
- Customers
- Managers
- Architects
- Developers
- Testers
- Researchers

Why Software Estimation?

You cannot plan if you cannot measure, and if you fail to plan, you have planned to fail.

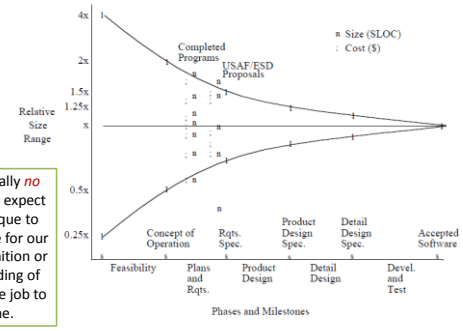
- ❖ Poor Quality, Reliability, and Capability
- ❖ Not Predictable
- ❖ Unable to Deliver
- ✓ Request for Proposals
- ✓ Contract Negotiations
- ✓ Planning and Scheduling
- ✓ Monitoring and Control

Cone of Uncertainty [1]



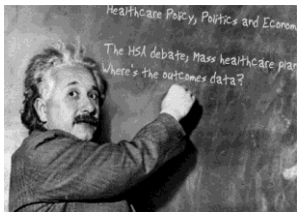
Industry Data, Historical Data, Project Data

Inputs: Specifications [2]



There is really *no way* we can expect the technique to compensate for our lack of definition or understanding of the software job to be done.

Expert Opinion



Pros & Cons of Expert Opinion

Can be used in
a new business area,
new technology, or
a brand-new kind of software

- Cannot be quantified
- Requires experts

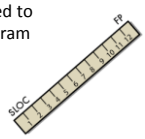
What Should I Do?

1. *Count* directly first.
2. Count something else then *compute* when you can't count directly.
3. Use *judgment* alone only as a last resort.

How Big Is Your Software? [3] [4]

How big is it? Well . . . it depends on *how you count*.

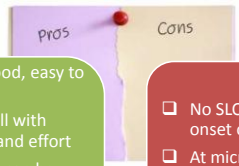
- *Source lines of code* — SLOC is a software metric used to measure the amount of code in a program.
- *Function points* — FP is a software metric used to measure the delivered functionality in a program from user perspective.
- *Effort* = Staff*Time
- *Productivity* = (Size or Function)/Time



Line of Code (LOC)

- Physical SLOC
- LOC (lines of code), SLOC (Source lines of code), *logical SLOC* — non-blank, non-comment, logical source lines.
- KLOC = 1000*LOC, KSLOC = 1000*SLOC
- ELOC (*Executable lines of code*), DSI (Delivered source instructions) — SLOC but excludes data declarations, compiler declarations, and other lines that do not generate executable instructions.
- ESLOC (*Effective source lines of code*) — SLOC that have been adjusted by the amount of rework required for portions of the system that were pre-existing at the start of the development.
- *Reused* code
- Language Productivity Factor

Pros & Cons of SLOC



- ✓ Well understood, easy to count
- ✓ Correlates well with functionality and effort
- ✓ Other metrics can be derived from the SLOC metric: productivity (SLOC/(staff*month)) , quality (defects/SLOC) measurements

- ❑ No SLOC exist at the onset of a project
- ❑ At micro-level, SLOC can be misleading

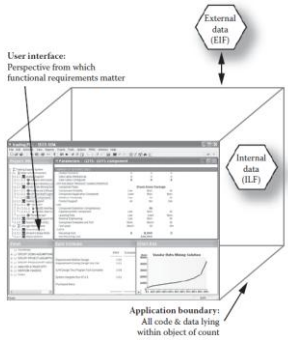
Function Point Analysis [5] [6] [7]

Function point analysis is a standard method for measuring software development from the user's point of view.

101: Determine Type of Function Point Count

- Development project function point count
- Enhancement project function point count
- Application function point count

Determine Application Boundary

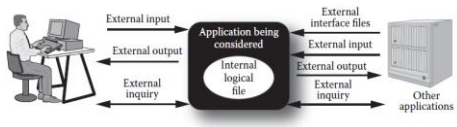


Identify Data Functions

The system's functionality is decomposed into:

Internal Logical File (ILF) – A user-identifiable group of logically related data or control information utilized and maintained by an application (tables, flat files, application control information).

External Interface File (EIF) – A user-identifiable group of logically related data or control information utilized by the application but maintained by another application (tables, flat files, application control information).

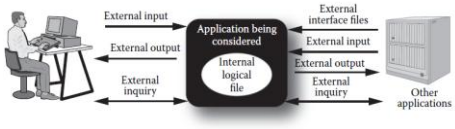


Identify Transactional Functions

External Inputs (EIs) – Any function or transaction that moves data into an application (data entry by users, data or file feeds by external applications).

External Outputs (EOs) – Any function or transaction that manipulates data and presents it to a user (reports, images).

External Inquiries (EQs) – A unique request that results in the retrieval of data (reports, search).



Identify ILF/EIF Complexity

Data Element Types (DETs) – Unique, user-recognizable, non-repeating fields or attributes, including foreign key attributes that enter the boundary of the subsystem or application (fields).

Record Element Types (RETs) – Logical sub-groupings based on the user's view of the data (0 RET = 1 RET).

ILF/EIF	1-19 DETs	20-50 DETs	51 + DETs
1 RET	Low	Low	Average
2-5 RETs	Low	Average	High
6+ RETs	Average	High	High

Example

Field	Count as a DET
UserId	No
Username	Yes
Password	Yes
AddressId	Yes
Total DETs: 3	
Total RETs: 1	
Complexity: Low	

Field	Count as a DET
AddressId	No
Address	Yes
City	Yes
State	Yes
Zip	Yes
Country	Yes
Total DETs: 5	
Total RETs: 0 (1)	
Complexity: Low	

Identify EIs/ EOs/ EQs Complexity

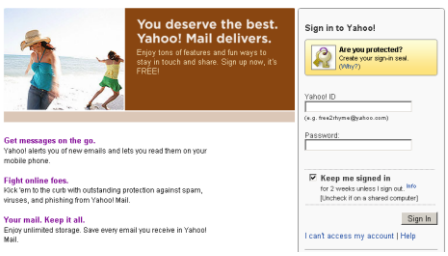
Data Element Types (DETs) – user-recognizable data elements that are maintained as ILFs by the EIs, appear in the EIs, EQs (data fields, buttons, search).

File Types Referenced (FTRs) – all ILFS and EIFs referenced or maintained during the processing of the EIs/EOs/EQs (tables, flat files).

EI	1-4 DETs	5-15 DETs	16 + DETs
0-1 FTR	Low	Low	Average
2 FTRs	Low	Average	Average
3+ FTRs	Average	High	High

EO/EQ	1-5 DETs	6-19 DETs	19+ DETs
0-1 FTR	Low	Low	Average
2-3 FTRs	Low	Average	High
4+ FTRs	Average	High	High

Example



EI DETs	EI FTRs	Complexity
4	2	Low

EO DETs	EO FTRs	Complexity
2	2	Low

Unadjusted Function Point

Program Characteristic	Low Complexity	Medium Complexity	High Complexity
External Inputs	__ × 3	__ × 4	__ × 6
External Outputs	__ × 4	__ × 5	__ × 7
External Queries	__ × 3	__ × 4	__ × 6
Internal Logical Files	__ × 4	__ × 10	__ × 15
External Interface Files	__ × 5	__ × 7	__ × 10
Unadjusted Function Point (UPFs) = Sum of EI FP, EO FP, EQ FP, ILF FP, EIF FP			

Program Characteristic	Low Complexity	Medium Complexity	High Complexity
External Inputs	1 × 3 = 3	0 × 4 = 0	0 × 6 = 0
External Outputs	1 × 4 = 4	0 × 5 = 0	0 × 7 = 0
Internal Logical Files	3 × 4 = 8	1 × 10 = 0	0 × 15 = 0
Unadjusted Function Point			**15**

General System Characteristic

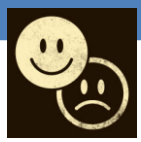
F1	Reliable back-up and recovery	0 – 5
F2	Data communications	0 – 5
F3	Distributed functions	0 – 5
F4	Performance	0 – 5
F5	Heavily used configuration	0 – 5
F6	Online data entry	0 – 5
F7	Operational ease	0 – 5
F8	Online update	0 – 5
F9	Complex interface	0 – 5
F10	Complex processing	0 – 5
F11	Reusability	0 – 5
F12	Installation ease	0 – 5
F13	Multiple sites	0 – 5
F14	Facilitate change	0 – 5
	Value adjustment factor (VAF) = 0.65 + 0.01 * Sum(F1,F14)	
	Adjusted function point (AFP) = UFPs * VAF	

Convert Function Points To SLOC

	Programming Statements per Function Point		
Language	**Minimum (Minus 1 Standard Deviation)**	**Mode (Most Common Value)**	**Maximum (Plus 1 Standard Deviation)**
Ada 83	45	80	125
Ada 95	30	50	70
C	60	128	170
C#	40	55	80
C++	40	55	140
Cobol	65	107	150
Fortran 90	45	80	125
Fortran 95	30	71	100
Java	40	55	80
Macro Assembly	130	213	300
Perl	10	20	30
Second generation default (Fortran 77, Cobol, Pascal, etc.)	65	107	160
Smalltalk	10	20	40
SQL	7	13	15
Third generation default (Fortran 90, Ada 83, etc.)	45	80	125
Microsoft Visual Basic	15	32	41

Pros & Cons of FPA

- FPA is independent of language used, development platform.
- FPA can be executed at the end of each phase/stage of a project.
- ISO standard



- Labor-intensive method
- Requires significant training and experience to be proficient
- Functional complexity weights and degrees of influence determined by trial and debate

Regression Analysis [8] [9]

- The investigator likes to check *how variables are related*.
- Am I able to *predict the value of a random variable* if I have the value of one or more variables available?
- This kind of study is what we call *regression analysis*.

Scatter Plot

Fitted Line

linear regression
$y = a + bx$

In *multiple regression*, the dependent variable is considered to depend on more than a single independent variable.

Example

Temp ^{0}C	0	10	20	30	40
Specific Heat	0.51	0.55	0.57	0.59	0.63

- *Estimate* the regression line of specific heat on temperature, and *predict* the value of the specific heat when the temperature is 25^{0}C.

- n = 5, $\sum x_i = 100$, mean of x = 20, $\sum y_i = 2.85$, mean of y = 0.57
  $\sum x_i y_i = 59.8$, $b_1 = 0.0028$, $b_0 = 0.514$
  Hence the fitted equation will be given by $\hat{y} = 0.514 + 0.0028x$.
  When the temperature x = 25^{0}C, the predicted specific heat is 0.514+0.0028*25 = 0.584.

1965 System Development Corporation Cost Model [2]

- 104 attributes of 169 software projects were collected and treated to extensive *statistical analysis*.
- 13-*parameter* linear estimation model:

```
MM = -33.63
+ 9.15 (Lack of Requirements) (0-2)
+ 10.73 (Stability of Design) (0-3)
+ 0.51 (Percent Math Instructions)
+ 0.46 (Percent Storage/Retrieval Instructions)
+ 0.40 (Number of Subprograms)
+ 7.28 (Programming Language) (0-1)
- 21.45 (Business Application) (0-1)
+ 13.53 (Stand-Alone Program) (0-1)
+ 12.35 (First Program on Computer) (0-1)
+ 58.82 (Concurrent Hardware Development) (0-1)
+ 30.61 (Random Access Device Used) (0-1)
+ 29.55 (Difference Host, Target Hardware) (0-1)
+ 0.54 (Number of Personnel Trips)
- 25.20 (Developed by Military Organization) (0-1).
```

The Putnam SLIM Model [2] [10]

- With regression modeling, the emphasis is on *constructing a formula* that best represents scattered data points.
- In mathematical modeling, the emphasis is on matching the data to the form of an *existing mathematical function*.
- Based on statistical analysis of several thousand projects, Putnam found that the relationship among the three principal elements of software estimating—size, schedule, and effort—matched the *Norden/Rayleigh function*.

$$S = C \times K^{1/3} \times t_d^{4/3}$$ where

S = software size in LOC

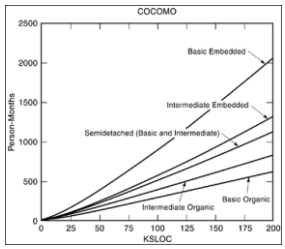
C = environmental factor (constant), dependent on the state of technology

K = total effort for the overall project

t_d = delivery time constraint (schedule) in years (development time in years)

COnstructive COst MOdel

- COCOMO is actually a hierarchy of *three* increasingly detailed *models* that range from
  - a single macro-estimation scaling model as *a function* of *product size* to
  - a micro-estimation model with a three-level work breakdown structure and a set of phase-sensitive multipliers for each *cost driver* attribute.



Project Development Modes

Mode	Product Size	Project/Team Size	Innovation	Deadline and Constraints	Development Environment
Organic	Typically 2–50 KLOC	Small project, small team—development team is familiar with the application language and tools	Little	Not Tight	Stable, In-House
Semi–detached	Typically 50–300 KLOC	Medium project, medium team—team is average in terms of abilities	Medium	Medium	Medium
Embedded	Typically over 300 KLOC	Large project requiring a large team	Greater	Severe Constraints	Complex HW/Customer Interfaces

- *The organic mode* is typified by systems such as payroll, inventory, and scientific calculation.
- *The semidetached mode* is typified by utility systems such as compilers, database systems, and editors.
- *The embedded mode* is typified by real-time systems such as those for air traffic control, ATMs, or weapon systems.

Nominal Effort and Schedule Estimation

- Boehm plotted his *observed 63 projects*.
- *Basic level* of the COnstructive COst MOdel (COCOMO) uses only mode and size to determine the effort and schedule.
- It is useful for fast, *rough* estimates of small to medium-size projects.

COCOMO NOMINAL EFFORT AND SCHEDULE EQUATIONS

DEVELOPMENT MODE	NOMINAL EFFORT	SCHEDULE
Organic	$(MM)_{NOM} = 3.2(KDSI)^{1.05}$	$TDEV = 2.5(MM_{DEV})^{0.38}$
Semidetached	$(MM)_{NOM} = 3.0(KDSI)^{1.12}$	$TDEV = 2.5(MM_{DEV})^{0.35}$
Embedded	$(MM)_{NOM} = 2.8(KDSI)^{1.20}$	$TDEV = 2.5(MM_{DEV})^{0.32}$

(KDSI = thousands of delivered source instructions)

MM: man-months
Size = KDSI = thousands of lines of code

COCOMO Example

- Suppose we are estimating the *cost* to develop the microprocessor-based communications processing software
  - for a highly ambitious new electronic funds transfer network
  - with high reliability, performance, development schedule, and interface requirements.
- We determine that these characteristics best fit the profile of an *embedded-mode* project.
- We next estimate the ~~*size*~~ of the product as 10,000 delivered source instructions, or 10 KDSI.
- We then determine that the *nominal development effort* for this embedded mode project is

$2.8(10)^{1.20} = 44$ man-months (MM).

Intermediate Level

- *Intermediate level* uses size, mode, and 15 additional variables to determine effort.
- The additional variables are called "*cost drivers*" and relate to product, personnel, computer, and project attributes that will result in more effort or less effort required for the software project.
- Effort (E) = a x $(Size)^b$ x C

  C: *effort adjustment factor* (EAF) There are two steps in determining this multiplying factor:
  - Step 1. is to assign *numerical values* to the cost drivers.
  - Step 2. is to *multiply* the cost drivers together to generate the effort adjustment factor, C.
  - EAF = C1 x C2 x . . . x Cn, [$C_{i=i}$ th cost adjustment factor]

Detailed Level

- *Detailed level* builds upon intermediate COCOMO by introducing the additional capabilities of phase-sensitive effort multipliers and a three-level product hierarchy.
- The program is *decomposed* into specific products and components of products.
- Boehm calls this the three-level *product hierarchy*:
  - system, subsystem, and module.
  - Cost drivers are analyzed separately for each component.
- The project development activities are partitioned into *phases*.
  - Boehm used four major phases: requirements (RQ), product design (PD), detailed design (DD), and coding and unit test (CUT) for development. Integration and testing (IT) and maintenance (MN) describe the entire life cycle.
  - *Phases* may be used to partition systems, subsystems, and/or modules.

COCOMO Drawbacks

- Estimation of
  - *object-oriented* software,
  - software created via spiral or *evolutionary models*, and
  - applications developed from *commercial-off-the-shelf* software.
- Project ~~*size*~~ or project *staff* information

COCOMO II

- *COCOMO II* is a revised and extended version of the model, built upon the original COCOMO.
- During the earliest conceptual stages of a project, the model uses *object point estimates* to compute effort.
- During the early design stages, when little is known about project size or project staff, *unadjusted function points* are used as an input to the model.
- After an architecture has been selected, design and development begin with *SLOC* input to the model.

COCOMO II Models

- COCOMO II – The *Early Design Model*

  Converting FPs to KLOC then use

  Effort = 2.45*KLOC*EAF
- COCOMO II – The *Post-Architecture Model*

  Converting FPs to KLOC then use

  Effort = $2.55*KLOC^B*EAF$

Model Calibration

- *Calibration* is the process of determining the deviation from a standard in order to compute the correction factors.
- *Items* which can be calibrated in a model include:
  - product types,
  - operating environments,
  - labor rates and factors,
  - various relationships between functional cost items.
- Calibration is to
  - run the model with *normal inputs* (known parameters such as software lines of code) against items for which the actual cost are known.
  - These estimates are then compared with the actual costs and the average deviation becomes a *correction factor* for the model.

Other Models [1]

- Albrect–Gaffney: Effort = 13.39 + 0.0545*FP
- Kemerer: Effort = 60.62 + 7.728*(10^{-8})*FP^3
- Matson–Barret–Meltichamp: Effort = 585.7 + 15.12*FP
- Benchmark: Effort = FP / Delivery Rate.
  where Delivery Rate - based on the most recent 600 projects.

Effort Activities Estimation

Requirement	11.30%
A&D	8.25%
Implementation	48.55%
Test	16.05%
Deployment	2.55%
Management	6.15%
Environment	2.03%
SCM	1.86%
SQA	1.80%
Training	0.63%
Defect Prev.	0.85%

Schedule Estimation

- Schedule = $3*Effort^{1/3}$
- Past schedule:
  Schedule = $PastSchedule*(EstimatedEffort/PastEffort)^{1/3}$
- Jones's First-Order estimation practice:
  Schedule = FPs^X where x

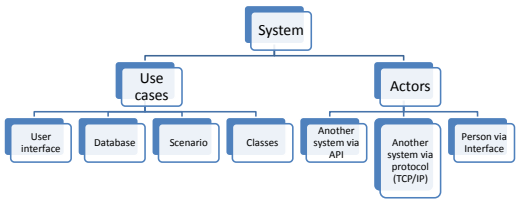
Kind of software	Better	Average	Worse
Object-oriented software	0.33	0.36	0.39
Client-server software	0.34	0.37	0.40
Business systems, internal intranet systems	0.36	0.39	0.42
Shrink-wrapped, scientific systems, engineering systems, public internet systems	0.37	0.40	0.43
Embedded systems, telecommunications, device drivers, systems software	0.38	0.41	0.44

Pros & Cons of Algorithmic Models

- ✓ Repeatable estimations
- ✓ Objectively calibrated to previous experience
- ✓ Various extensions for almost every purpose
- ✓ Tool support

- ❑ Poor sizing inputs and inaccurate cost driver rating will result in inaccurate estimation.

Use Case Points



Unadjusted Use Case Weight – UUCW

Use Case Type	Description	Weight	Number of Use Cases	Result
Simple	A simple user interface and touches only a single database entity; its success scenario has 3 steps or less; its implementation involves less than 5 classes.	5	8	40
Average	More interface design and touches 2 or more database entities; between 4 to 7 steps; its implementation involves between 5 to 10 classes.	10	12	120
Complex	Involves a complex user interface or processing and touches 3 or more database entities; over seven steps; its implementation involves more than 10 classes.	15	4	60
	Total Unadjusted Use Case Weight		**Total UUCW**	**220**

Unadjusted Actor Weight – UAW

Actor Type	Description	Weight	Number of Actors	Result
Simple	The Actor represents another system with a defined API	1	8	8
Average	The Actor represents another system interacting through a protocol, like TCP/IP	2	12	24
Complex	The Actor is a person interacting via an interface.	3	4	12
	Total Unadjusted Actor Weight		**Total UAW**	**44**

Technical Factor

Technical Factor	Description	Weight	Project Perceived Complexity	Calculated Factor (weight*perceived complexity)
T1	Distributed System	2	5	10
T2	Performance	1	4	4
T3	End User Efficiency	1	2	2
T4	Complex internal Processing	1	4	4
T5	Reusability	1	2	2
T6	Easy to install	0.5	5	2
T7	Easy to use	0.5	3	2
T8	Portable	2	3	6
T9	Easy to change	1	3	3
T10	Concurrent	1	2	2
T11	Special security features	1	2	2
T12	Provides direct access for third parties	1	5	5
T13	Special user training facilities are required	1	3	3
	TCF = 0.6 + (0.01*Total Factor).		**0 - 5**	**47 (Total Factor)**

Environment Complexity Factor – ECF

Environmental Factor	Description	Weight	Perceived Impact	Calculated Factor (weight*perceived complexity)
E1	Familiarity with UML	1.5	4	6
E2	Application Experience	0.5	2	1
E3	Object Oriented Experience	1	5	5
E4	Lead analyst capability	0.5	2	1
E5	Motivation	1	1	1
E6	Stable Requirements	2	5	10
E7	Part-time workers	-1	0	0
E8	Difficult Programming language	2	1	2
Environment Complexity Factor	**ECF = 1.4 + (-0.03*Total Factor)**		**Total Factor**	**26**

Use Case Points

- Adjusted Use Case Points

  AUCPs = UUCP *TCP * ECF

  where

  UUCP - Unadjusted Use Case Points = UUCW + UUCA

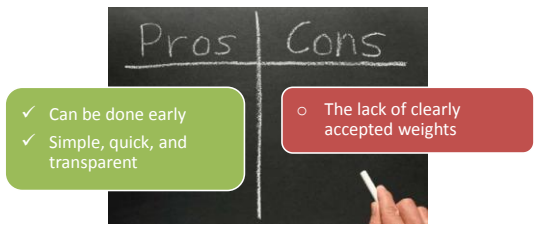
  TCF - Technical Complexity Factor.

  ECF - Environment Complexity Factor.

- Effort (person-hours) = AUCPs*PF

  where the Productivity Factor (PF) is a ratio of the number of man hours per use case point based on past projects. If no historical data has been collected, a figure between 15 and 30 is suggested by industry experts. A typical value is 20.

Pros & Cons of UC Point



Object Points

The system is decomposed into:

- Screens that are displayed
- Reports that are produced by the system
- Third-generation language (3GL) modules - the number of program modules that must be developed

Object points are NOT the same as object classes.

Determine Complexity

- Classify each element instance into simple, medium and difficult complexity levels

Screens	Number and source of data tables		
Number of views contained	**Total <4**	**Total <8**	**Total 8+**
<3	simple	simple	medium
3-7	simple	medium	difficult
8+	medium	difficult	difficult

Reports	Number and source of data tables		
Number of views contained	**Total <4**	**Total <8**	**Total 8+**
<3	simple	simple	medium
3-7	simple	medium	difficult
8+	medium	difficult	difficult

New Object Points

- Weight the number in each cell using the following table.

Object type	Simple	Medium	Difficult
Screen	1	2	3
Report	2	5	8
3GL component	-	-	10

- Add all the weighted object instances to get one number, the object points (OP)
- Compute the New Object Points to be developed, NOP=(OP) (100-%reuse)/100

Example

Hello world | Hello world | Hello world

Hello World		Weighting Simple	Sum
GUI inputs / screen	1	1	1
GUI outputs / reports	3	5	15
Number of 3 GL-Modules	1	10	10
		Object Points	26
	Reuse 50%	New Object Points	13

Effort Estimation

- COCOMO II - Application composition model:
  Effort (person-month) = NOP/PROD
  where Productivity Rate

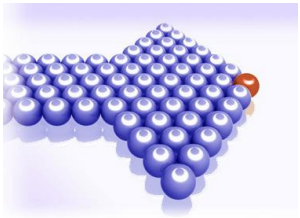
Developers experience and maturity capability	Very low	Low	Nominal	High	Very high
PROD	4	7	13	25	50

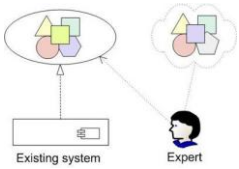
Pros & Cons of Object Point

- ✓ Can be done early
- ✓ Simple, quick, and transparent

- ○ The lack of clearly accepted weights

Analogy [1]





Get Detailed Size, Effort, and Cost Results for a Similar Previous Project

Old Project	
Database	5,000 lines of code (LOC)
User interface	14,000 LOC
Graphs and reports	9,000 LOC
Foundation classes	4,500 LOC
Business rules	11,000 LOC
TOTAL	**43,500 LOC**

Old Project	
Database	10 tables
User interface	14 Web pages
Graphs and reports	10 graphs + 8 reports
Foundation classes	15 classes
Business rules	???

Compare the Size of the New Project to a Similar Past Project

Subsystem	Actual Size of Old Project	Estimated Size of New Project	Multiplication Factor
Database	10 tables	14 tables	1.4
User interface	14 Web pages	19 Web pages	1.4
Graphs and reports	10 graphs + 8 reports	14 graphs + 16 reports	1.7
Foundation classes	15 classes	15 classes	1.0
Business rules	???	???	1.5

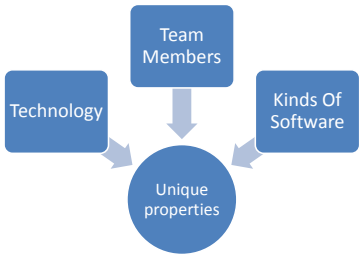
Build Up the Estimate for the New Project's Size as a Percentage of the Old Project's Size

Subsystem	Old Project	Multiplication Factor	New Project
Database	5,000	1.4	7,000
User interface	14,000	1.4	19,600
Graphs and reports	9,000	1.7	15,300
Foundation classes	4,500	1.0	4,500
Business rules	11,000	1.5	16,500
TOTAL	**43,500**	-	**62,900**

Create an Effort Estimate Based on the Size of the New Project Compared to the Previous Project

Term	Value
Size of New Project	62,900 LOC
Size of Old Project	÷ 43,500 LOC
Size ratio	= 1.45
Effort for Old Project	× 30 staff months
Estimated effort for New Project	= 44 staff months

Check for Consistent Assumptions Across the Old and New Projects



Pros & Cons of Analogy

- ✓ Simple, accurate, cheap
- ✓ Based on proven characteristics

- ○ Impossible if no comparable project has been tackled.
- ○ The need to determine the most important variables to be used for describing the solution.

Halstead's Software Science

- The measurable and countable properties are :
  1) n_1 = number of unique or distinct operators appearing in that implementation
  2) n_2 = number of unique or distinct operands appearing in that implementation
  3) N_1 = total usage of all of the operators appearing in that implementation
  4) N_2 = total usage of all of the operands appearing in that implementation

Halstead's Software Science (cont.)

- The vocabulary $n = n_1 + n_2$
- The implementation length $N = N_1 + N_2$
- Volume $V = n_1 log_2 n_1 + n_2 log_2 n_2$
- Difficulty $D = (n1/2) * (N_2 / n2)$
- Effort $E = D*V$
- Schedules $S = (n_1 N_2 (n_1 log_2 n_1 + n_2 log_2 n_2) log_2 n) / 2n_2 S$, where S from 5 to 20.

Pros & Cons of Halstead's Method

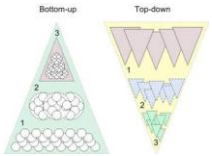
- ❑Simple to calculate
- ❑Can be used for any programming language

- ❑Depends on completed code
- ❑No predictive

Top-Down and Bottom-Up

Any of these approaches may be used top-down or bottom-up.

Start at the component level and estimate the effort required for each component. Add these efforts to reach a final estimate.



Start at the system level and assess the overall system functionality and how this is delivered through sub-systems.

Usable when the architecture of the system is known and components identified.
It may underestimate the costs of system level activities such as integration and documentation.

Usable without knowledge of the system architecture and the components that might be part of the system.
Can underestimate the cost of solving difficult low-level technical problems.

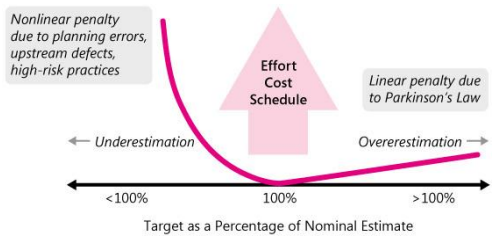
Best Practices

- Using more than one estimation technique
- Including risk impact

Question 1 [1]

- Is It Better to Overestimate or Underestimate?

Answer 1



Question 2

- In most of the companies, presales and marketing gurus will set the figures and the due date even before you start communicating with your client as proceed to the analysis phase.
- My question is when can you do the estimation?

Answer 2

- In that case you can use “Price to Win” technique.
- The estimated effort depends upon the customer's budget and not on the software functionality.
- The estimate is made as low as necessary to win the job.
- When you're asked to provide an estimate, determine whether you're supposed to be estimating or figuring out how to hit a target.
- "The price-to-win technique has won a large number of software contracts for a large number of software companies. Almost all of them are out of business today."[Boehm 81], p337.

Question3

- *EXECUTIVE: How long do you think this project will take? We need to have this software ready in 3 months for a trade show. I can't give you any more team members, so you'll have to do the work with your current staff. Here's a list of the features we'll need.*
- *PROJECT LEAD: OK, let me crunch some numbers, and get back to you.*
- Later…
- *PROJECT LEAD: We've estimated the project will take 5 months.*
- *EXECUTIVE: Five months!? Didn't you hear me? I said we needed to have this software ready in 3 months for a trade show!*
- *PROJECT LEAD:* ???

Answer 3

- Software may be priced (estimated) to gain a contract and the functionality adjusted to the price.

Tools for Estimation

- **Costar**

  This COCOMO II based product is from Softstar Systems. It estimates project duration, staffing levels, effort and cost.

  http://www.softstarsystems.com

- **KnowledgePLAN**

  SPR KnowledgePLAN® is a software tool designed to help you plan software projects. With KnowledgePLAN® you can effectively size your projects and then estimate work, resources, schedule, and defects. You can even evaluate project strengths and weaknesses to determine their impact on quality and productivity.

  http://www.spr.com/products/knowledge.shtm

Tools for Estimation

- **Construx Estimate**

  It contributes to project success by helping improve your software estimation capabilities. Estimate leverages a blend of proven estimation models to predict **effort**, **budget**, and **schedule** for your project based on size estimates.
  http://www.construx.com/Page.aspx?nid=68

- **SLIM**

  http://www.qsm.com/slim_estimate.html

- **Code Counter**

  http://www.codeproject.com/tools/codecounter.asp

More Topics

- McCabe's Cyclomatic Number.
- Fan-In Fan-Out Complexity - Henry's and Kafura's.
- Defects Estimation.
- Reliability Estimation.
- History of Software Estimation.
- Software Estimation Training.

Thinking

- Measurement vs. Rating.
  I. A *measurement* is objective and can be manipulated mathematically.
  II. A *rating* is subjective and cannot be manipulated mathematically.
  III. FPs, UCPs, OPs Are a Rating, Not a Measurement
    1) 1 Application 2000 FPs = 2 * Application 1000 FPs???
    2) If a team completes an application of 250 FP in 10 weeks, then they can complete an application 500 FPs in 20 weeks???
    3) How about complexity between 1 application 100 FPs and 1 application 50 FPs???
- Why Must Functional Size be a Single Number?

Thank You for Your Time