5.Software Effort Estimation

A successful project is one delivered 'on time, within budget and with the required quality'. This implies that targets are set which the project manager then tries to meet.

Estimation

An estimate of effort and duration has to be made at the beginning

Difficulties of estimating arise due to:

1. **Subjective nature of estimating:** underestimating the difficulty of small tasks
2. **Political Implications:** between users and development team that is due to different groups within an organization have different objectives
3. **Changing technology:** Where technologies change rapidly, it is difficult to use the experience of previous projects on new ones.
4. **Lack of homogeneity of project experience:** Even where technologies have not changed, knowledge about typical task durations may not be easily transferred from one project to another because of other differences between projects.

Exercise 5.1

- A Person Day is 8 hours
- A Person Week is 40 hours
- A Person month is approx. 20 working days=160 Person hours
- 3.9 person months or work months = 20 x 3.9 = 78 days = 624 hours
- Productivity = SLOC/wm = 6050/16.7=362(SLOC/month)
- 362/20= 18 SLOC/day(average of 10 LOC)

Where are Estimates Done?

Estimates are carried out at various stages of a software project for a variety of reasons.

- **Strategic planning**: Project portfolio management involves estimating the costs and benefits of new applications in order to allocate priorities. Such estimates may also influence the scale of development staff recruitment.
- **Feasibility study**: This confirms that the benefits of the potential system will justify the costs.
- **System specification:** The effort needed to implement different design proposals will need to be estimated. Estimates at the design stage will also confirm that the feasibility study is still valid.
- **Evaluation of suppliers' proposals:** s In the case of the IOE annual maintenance contracts subsystem, for example, IOE might consider putting development out to tender.
- **Project planning:** As the planning and implementation of the project becomes more detailed, more estimates of smaller work components will be made. These will confirm earlier broad-brush estimates, and will support more detailed planning, especially staff allocations

'Software Estimation takes place in steps 3 and 5 in Planning'

To set estimating into the context of the Step Wise framework presented in project planning, re-estimating could take place at almost any step, but specific provision is made for the production of a relatively high-level estimate at Step 3, 'Analyse project characteristics', and for each individual activity in Step 5. As Steps 5–8 are repeated at progressively lower levels, so estimates will be done at a finer degree of detail.

Problems with Over- and Under-Estimates

A project leader will need to be aware that an over-estimate may cause the project to take longer than it would otherwise. This can be explained by the application of two 'laws'.

1. **Parkinson's Law**: 'Work expands to fill the time available', that is, given an easy target staff will work less hard.
2. **Brooks' Law**: 'putting more people on a late job makes it later'. The effort of implementing a project will go up disproportionately with the number of staff assigned to the project. As the project team grows in size, so will the effort that has to go into management, coordination and communication. If there is an overestimate of the effort required, this could lead to more staff being allocated than needed and managerial overheads being increased.

Under-estimated project might not be completed on time or to cost, it might still be implemented in a shorter time than a project with a more generous estimate.

- **Weinberg's zeroth law of reliability:** 'if a system does not have to be reliable, it can meet any other objective'. The danger with the under-estimate is the effect on quality. Staff, particularly those with less experience, could respond to pressing deadlines by producing work that is substandard.

Th e Basis for Software Estimating

The need for Historical data

Most estimating methods need information about past projects. However, care is needed when applying past performance to new projects because of possible differences in factors such as programming languages and the experience of staff. If past project data is lacking, externally maintained datasets of project performance data can be accessed.

Parameters to be estimated

The project manager needs to estimate two project parameters for carrying out project planning. These two parameters are effort and duration.

Duration is usually measured in months.

- Work-month (wm) is a popular unit for effort measurement.
- The term person-month (pm) is also frequently used to mean the same as work-month.
- One person-month is the effort an individual can typically put in a month.

Person-month (pm) is considered to be an appropriate unit for measuring effort compared to person-days or person-years because developers are typically assigned to a project for a certain number of months.
The person-month estimate implicitly takes into account the productivity losses that normally occur due to time lost in holidays, weekly offs, coffee breaks, etc.

Measure of work

Measure of work involved in completing a project is also called the size of the project. Work itself can be characterized by cost in accomplishing the project and the time over which it is to be completed. Direct calculation of cost or time is difficult at the early stages of planning. It is therefore a standard practice to first estimate the project size; and by using it, the effort and time taken to develop the software can be computed.
The size of a project is obviously not the number of bytes that the source code occupies, neither is it the size of the executable code. The project size is a measure of the problem complexity in terms of the effort and time required to develop the product.
Two metrics are at present popularly being used to measure size. These are Source Lines of Code (SLOC) and Function Point (FP). The SLOC measure suffers from various types of disadvantages, which are to a great extent corrected in the FP measure.

Disadvantages of SLOC (shortcomings of the SLOC measure):

- **No precise definition**: SLOC is a very imprecise measure. The writers' view is that comment lines are excluded in determining the SLOC measure. This can be debated, but the main point is that consistency is essential.
- **Difficult to estimate at start of a project**: From the project manager's perspective, the biggest shortcoming of the SLOC metric is that it is very difficult to estimate it during project planning stage, and can be accurately computed only after the development of the software is complete. The

SLOC count can only be guessed at the beginning of a project, often leading to grossly inaccurate estimations.

- **Only a code measure:** SLOC is a measure of coding activity alone. A good problem size measure should consider the effort required for carrying out all the life cycle activities and not just coding.
- **Programmer-dependent:** SLOC gives a numerical value to the problem size that can vary widely with the coding style of individual programmers. This aspect alone renders any LOC-based size and effort estimations inaccurate.
- **Does not consider code complexity:** Two software components with the same KLOC will not necessarily take the same time to write, even if done by the same programmer in the same environment.

Software Effort Estimation Techniques

The main ways of deriving estimates of software development effort as:

- **algorithmic models**: which use 'effort drivers' representing characteristics of the target system and the implementation environment to predict effort;
- **expert judgement:** based on the advice of knowledgeable staff.
- **analogy:** where a similar, completed, project is identified and its actual effort is used as the basis of the estimate.
- **Parkinson:** where the staff effort available to do a project becomes the 'estimate'.
- **price to win**: where the 'estimate' is a figure that seems sufficiently low to win a contract.
- **top-down:** where an overall estimate for the whole project is broken down into the effort required for component tasks.
- **bottom-up:** where component tasks are identified and sized and these individual estimates are aggregated.

Expert Judgement:

This is asking for an estimate of task effort from someone who is knowledgeable about either the application or the development environment. This method is often used when estimating the effort needed to change an existing piece of software. The estimator would have to examine the existing code in order to judge the proportion of code affected and from that derive an estimate. Someone already familiar with the software would be in the best position to do this.

Estimation by Analogy: (Case-based Reasoning)

This is also called case-based reasoning. The estimator identifies completed projects (source cases) with similar characteristics to the new project (the target case). The effort recorded for the matching source case is then used as a base estimate for the target. The estimator then identifies differences between the target and the source and adjusts the base estimate to produce an estimate for the new project.

A problem is identifying the similarities and differences between applications where you have a large number of past projects to analyse. One attempt to automate this selection process is the ANGEL software tool. This identifies the source case that is nearest the target by measuring the Euclidean distance between cases. The Euclidean distance is calculated as:

distance = square-root of ((target_parameter1 – source_parameter1) 2 + . . . (target_parametern – source_parametern) 2)

Suppose that the source and the target systems are being matched on the basis of two parameters, the number of inputs to the number of outputs. The new project to be developed is known to have 7 inputs and 15 outputs. One of the past cases, project A has 8 inputs and 17 outputs. Project B has 5 inputs and 10 outputs. Which project (A or B) is more similar/close to the new project?

Identifying similarities and differences between projects where you have a large number of past projects. Selecting the most similar project will be used for estimating the effort of the new project.
We identify the source system that is nearest to the target system using Euclidean distance.

Distance = square-root-of((target parameter1 - source parameter1)2
+…… (target_parametern – source_parametern)2)

Exercise 5.7
Say that the cases are being matched on the basis of two parameters, the number of inputs to and the number of outputs from the application to be built. The new project is known to require 7 inputs and 15 outputs. One of the past cases, project A, has 8 inputs and 17 outputs.
Project B has 5 inputs and 10 outputs. What would be the Euclidean distance between this project and the target new project being considered above? Is project B a better analogy with the target than project A?

Euclidean distance between source project A and target system is =sqrt ((7-8)2 + (15-17)2) = 2.24
Euclidean distance between source project B and target system is =sqrt ((7-5)2 + (15-10)2) = 5.39
Project A is hence closer.

Bottom-up estimating:

With the bottom-up approach the estimator breaks the project into its component tasks. Each task into sub-tasks and each sub-task into sub-subtasks thus producing a work breakdown schedule (WBS). Stop when you get to what one person can do in one/two weeks Bottom-up approach -At each higher level calculate estimate by adding estimates for lower levels

A procedural code-oriented approach
The bottom-up approach described above works at the level of activities. In software development a major activity is writing code. Here we describe how a bottom-up approach can be used at the level of software components.

- Envisage the number and type of software modules in the final system.
- Estimate the SLOC of each identified module.
- Estimate the work content, taking into account complexity and technical difficulty.
- Calculate the work-days effort

The Top-down Approach and Parametric Models

The top-down approach is normally associated with parametric (or algorithmic) models. Project effort relates mainly to variables associated with characteristics of the final system. A parametric model will normally have one or more formulae in the form:

effort = (system size) * (productivity rate)

For example, system size might be in the form ‘thousands of lines of code’ (KLOC) and have the specific value of 3 KLOC while the productivity rate was 40 days per KLOC. These values will often be matters of judgement.

system size = lines of code(3KLOC)
productivity = lines of code per day (40 days per KLOC)
productivity = effort/size (based on past projects)

Two key components for forecasting effort

- Method of assessing the amount of work needed.
- Method of assessing the rate of work at which the task can be done

Let us say that a software module required 2KLOC

Kate an expert developer works at the rate of 40 days per KLOC. She can complete the work in 2 x 40 days=80 days
Ken who is less experienced takes 55 days per KLOC. She completes the work in 2 x 55days=110 days.

In this case KLOC is a size driver indicating the amount of work to be done, while developer experience is a productivity driver influencing the productivity or work rate.

If you have figures for the effort expended on past projects (in work-days for instance) and also the system sizes in KLOC, you should be able to work out a productivity rate as
productivity = effort/size

Bottom-up versus Top-down

The top-down and Bottom-up approaches are not mutually exclusive.
Bottom-up

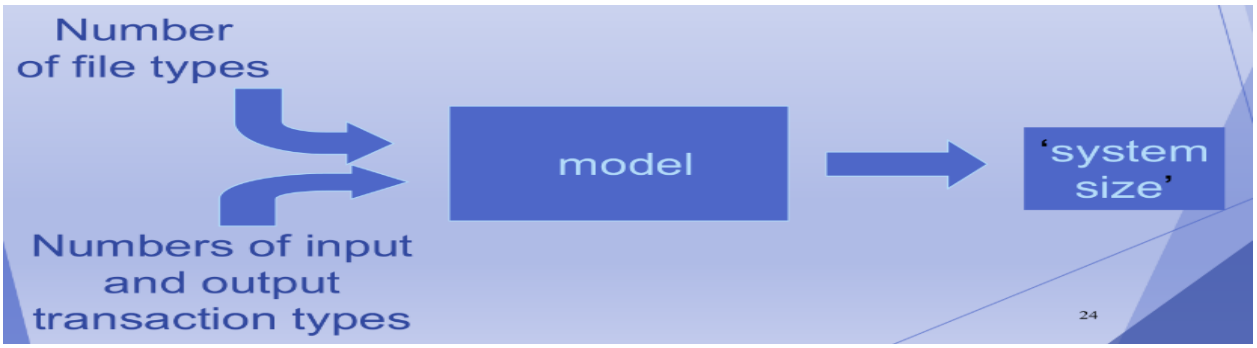
- identify all tasks that have to be done – so quite time-consuming.
- used when you have no data about similar past projects.
- to be used at later stages of planning

Top-down

- produce overall estimate based on project cost drivers, representing characteristics of the target system.
- based on past project data.
- divide overall estimate between jobs to be done.

Parametric Models

Some models focus on task or system size e.g. Function Points. FPs originally used to estimate Lines of Code, rather than effort.



Albrecht Function Point Analysis

Identify the size of programs irrespective of their programming Language used.
Information systems comprise of five major components or 'external user types'.

- **External Input types:** input transactions update internal computer files (add (), change () and delete ())
- **External Output types**: Transactions that output data to the user like reports (derived data after applying formula- information from internal logical files or external interface files)
- **External inquiry types**: transactions initiated by users which provide information without updating internal files. (data retrieval from one or more internal logical files and external interface files)
- **Logical internal file types:** files used by the system…..group of data items accessed together such as record or database.
- **External interface file types:** passing data from one application to other application.

The analyst identifies each instance of each external user type in the application. Each component is then classified as having either high, average or low complexity. The counts of each external user type in each

complexity band are multiplied by specified weights to get FP scores which are summed to obtain an overall FP count which indicates the information processing size.

Albrecht complexity mulpliers table

External user type	Multiplier		
	Low	Average	High
External input type	3	4	6
External output type	4	5	7
External inquiry type	3	4	6
Logical internal file type	7	10	15
External interface file type	5	7	10

The International FP User Group (IFPUG) has now promulgated rules on how this is assessed. For example, in the case of logical internal fi les and external interface fi les, the boundaries shown in Table 5.3 are used to decide the complexity level. Similar tables exist for external inputs and outputs.

IFPUG file type complexity table

Number of record types	Number of data types		
	<20	20–50	>50
1	Low	Low	Average
2 to 5	Low	Average	High
>5	Average	High	High

A logical internal fi le might contain data about purchase orders. These purchase orders might be organized into two separate record types: the main PURCHASE-ORDER details, namely purchase order number, supplier reference and purchase order date, and then details for each PURCHASE ORDER-ITEM specified in the order, namely the product code, the unit price and number ordered.
The number of record types for that fi le would therefore be 2 and the number of data types would be 6. According to Table IFPUG, this file type would be rated as ‘low’. This would mean that according to Albrecht complexity mulplier table , the FP count would be 7 for this file.

Function Points Mark II

Originator, Charles Symon builds on work by Albrecht, developed in parallel to IFPUG FPs. Calculate Unadjusted Function Points (UFP). For each transaction the UFP’s are calculated as

Wi 3 (number of input data element types) + We 3 (number of entity types referenced) + Wo 3 (number of output data element types)

Wi, We, and Wo are weightings derived by asking developers the proportions of effort spent in previous projects developing the code dealing respectively with inputs, accessing and modifying stored data and processing outputs.

The process for calculation weightings is time consuming and most FP counters use industry averages which are currently 0.58 for *Wi*, 1,66 for *We* and 0.26 for *Wo.*

A cash receipt transaction in the IOE maintenance accounts subsystem accesses two entity types – INVOICE and CASH-RECEIPT.
The data inputs are:
Invoice number
Date received
Cash received

If an INVOICE record is not found for the invoice number then an error message is issued. If the invoice number is found then a CASH-RECEIPT record is created. The error message is the only output of the transaction. The unadjusted function points, using the industry average weightings, for this transaction would therefore be: (0.58*3) + (1.66*2) + (0.26*1) = 5.32

Exercise 5.10
The IOE annual maintenance contracts subsystem for which Amanda is responsible will have a transaction which sets up details of new annual maintenance contract customers.
The operator will input:

Customer account number
Customer name
Address Postcode
Customer type
Renewal date

All this information will be set up in a CUSTOMER record on the system's database. If a CUSTOMER account already exists for the account number that has been input, an error message will be displayed to the operator.

Input Data Types = 6
Entities accessed = 1
Output Data Types = 1
UFP= (0.58 X 6) + (1.66 X 1) + (0.26 x 1)

COSMIC Full Function Points

The IFPUG are suitable for information systems, they are not helpful when it comes to sizing real-time or embedded applications. This has resulted in the development of another version of function points – the COSMIC FFP method.

COSMIC deals with this by decomposing the system architecture into a hierarchy of software layers. The software component to be sized can receive requests for services from layers above and can request services from those below it. At the same time there could be separate software components at the same level that engage in peer-to-peer communication. This identifies the boundary of the software component to be assessed and thus the points at which it receives inputs and transmits outputs. Inputs and outputs are aggregated into data groups, where each group brings together data items that relate to the same object of interest.

The overall FFP count is derived by simply adding up the counts for each of the four types of data movement. The resulting units are Cfsu (COSMIC functional size units).

Data groups can be moved about in four ways:

- **entries (E):** which are effected by subprocesses that move the data group into the software component in question from a 'user' outside its boundary – this could be from another layer or another separate software component in the same layer via peer-to-peer communication;

- **exits (X):** which are effected by subprocesses that move the data group from the software component to a 'user' outside its boundary;
- **reads (R):** which are data movements that move data groups from persistent storage (such as a database) into the software component;
- **writes (W):** which are data movements that transfer data groups from the software component into persistent storage.

Slid 35(diagram)

Exercise 5.11
A small computer system controls the entry of vehicles to a car park. Each time a vehicle pulls up before an entry barrier, a sensor notifies the computer system of the vehicle's presence. The system examines a count that it maintains of the number of vehicles that are currently in the car park. This count is kept on backing storage so that it will still be available if the system is temporarily shut down, for example because of a power cut. If the count does not exceed the maximum allowed then the barrier is lifted and the count is incremented. When a vehicle leaves the car park, a sensor detects the exit and reduces the count of vehicles.
There is a system administration system that can set the maximum number of cars allowed, and which can be used to adjust or replace the count of cars when the system is restarted.
Identify the entries, exits, reads and writes in this application.

Data Movement	Type
Incoming Vehicle sensed	Entry
Access vehicle count	Read
Signal barrier to be lifted	Exit
Increment vehicle count	Write
Outgoing vehicle sensed	Entry
Decrement vehicle count	Write
Record adjust vehicle count	Write
Set new maximum	Write

COCOMO II: A Parametric Productivity Model

Boehm's COnstructive COst MOdel (COCOMO) refers to a group of models for Software Effort Estimation of which the key one is COCOMO II.
The basic model was based on the equation

$$Effort = c(size)^k$$

where effort was measured in person months consisting of 152 working hours, size in *kdsi*, thousand of delivered lines of source code and c and k were constants.

The first step was to derive an estimate of the system size in terms of kdsi. The constants, c and k depended on whether the system could be classified, in Boehm's terms, as 'organic', 'semi-detached' or 'embedded'. These related to the technical nature of the system and the development environment.

TABLE 5.4 COCOMO81 constants

System type	c	k
Organic	2.4	1.05
Semi-detached	3.0	1.12
Embedded	3.6	1.20

- Organic mode This would typically be the case when relatively small teams developed software in a highly familiar in-house environment and when the system being developed was small and the interface requirements were flexible.
- Embedded mode This meant that the product being developed had to operate within very tight constraints and changes to the system were very costly.
- Semi-detached mode This combined elements of the organic and the embedded modes or had characteristics that came between the two.

The exponent value k, when it is greater than 1, means that larger projects are seen as requiring disproportionately more effort than smaller ones.

The COCOMO II approach uses various multipliers and exponents the values of which have been set initially by experts. Estimates are required at different stages in the system life cycle and COCOMO II has been designed to accommodate this by having models for three different stages.

- Application composition Here the external features of the system that the users will experience are designed. Prototyping will typically be employed to do this. With small applications that can be built using high-productivity application-building tools, development can stop at this point.
- Early design Here the fundamental software structures are designed. With larger, more demanding systems, where, for example, there will be large volumes of transactions and performance is important, careful attention will need to be paid to the architecture to be adopted.
- Post architecture Here the software structures undergo final construction, modification and tuning to create a system that will perform as required.

To estimate the effort for application composition, the counting of object points is recommended by the developers of COCOMO II. This follows the function point approach of counting externally apparent features of the software.
At the early design stage, FPs are recommended as the way of gauging a basic system size. An FP count may be converted to an LOC equivalent by multiplying the FPs by a factor for the programming language that is to be used.

The following model can then be used to calculate an estimate of person-months.

$$\mathbf{pm = A(size)\ (sf)\ *(em1)\ *\ (em2)\ *. . .\ *\ (emn)}$$

where pm is the effort in ‘person-months’, A is a constant (which was set in 2000 at 2.94), size is measured in kdsi (which may have been derived from an FP count as explained above), and sf is exponent scale factor.

The scale factor is derived thus:

$$\mathbf{sf = B + 0.01\ 3\ \sum(exponent\ driver\ ratings)}$$

where B is a constant currently set at 0.91

The qualities that govern the exponent drivers used to calculate the scale factor are listed below. Note that the less each quality is applicable, the bigger the value given to the exponent driver. The fact that these factors are used to calculate an exponent implies that the lack of these qualities increases the effort required disproportionately more on larger projects.

- **Precedentedness (PREC)** This quality is the degree to which there are precedents or similar past cases for the current project. The greater the novelty of the new system, the more uncertainty there is and the higher the value given to the exponent driver.
- **Development flexibility (FLEX**) This reflects the number of different ways there are of meeting the requirements. The less flexibility there is, the higher the value of the exponent driver.

- **Architecture/risk** resolution (RESL) This reflects the degree of uncertainty about the requirements. If they are liable to change then a high value would be given to this exponent driver.

TABLE 5.5 COCOMO II Scale factor values

Driver	Very low	Low	Nominal	High	Very high	Extra high
PREC	6.20	4.96	3.72	2.48	1.24	0.00
FLEX	5.07	4.05	3.04	2.03	1.01	0.00
RESL	7.07	5.65	4.24	2.83	1.41	0.00
TEAM	5.48	4.38	3.29	2.19	1.10	0.00
PMAT	7.80	6.24	4.68	3.12	1.56	0.00

- **Team cohesion (TEAM)** This reflects the degree to which there is a large dispersed team (perhaps in several countries) as opposed to there being a small tightly knit team.
- **Process maturity (PMAT)** Chapter 13 on software quality explains the process maturity model. The more structured and organized the way the software is produced, the lower the uncertainty and the lower the rating will be for this exponent driver.

In the COCOMO II model the effort multipliers (em) adjust the estimate to take account of productivity factors, but do not involve economies or diseconomies of scale. The multipliers relevant to early design are in Table 5.6 and those used at the post architecture stage in Table 5.7. Each of these multipliers may, for a particular application, be given a rating of very low, low, nominal, high or very high. Each rating for each effort multiplier has an associated value. A value greater than 1 increases development effort, while a value less than 1 decreases it. The nominal rating means that the multiplier has no effect.

TABLE 5.6 COCOMO II Early design effort multipliers

Code	Effort modifier	Extra low	Very low	Low	Nominal	High	Very high	Extra high
RCPX	Product reliability and complexity	0.49	0.60	0.83	1.00	1.33	1.91	2.72
RUSE	Required reusability			0.95	1.00	1.07	1.15	1.24
PDIF	Platform difficulty			0.87	1.00	1.29	1.81	2.61
PERS	Personnel capability	2.12	1.62	1.26	1.00	0.83	0.63	0.50
PREX	Personnel experience	1.59	1.33	1.12	1.00	0.87	0.74	0.62
FCIL	Facilities available	1.43	1.30	1.10	1.00	0.87	0.73	0.62
SCED	Schedule pressure		1.43	1.14	1.00	1.00	1.00	