Learning Resource
On
Software Project Management

Unit-2
Software Measurements

Prepared By:

Kunal Anand

Assistant Professor, SCE

KIIT, DU, Bhubaneswar-24

Talkflow

- Monitoring and measurement of software development
- Introduction to Software Estimation
- Parameters to be estimated
- Taxonomy of Estimations
  - Bottom-up vs Top down
  - Parametric Model
  - Empirical Estimation
- Function Point methods
  - Albrecht FP
  - Mark-II FP
  - COSMIC-FFP method
- COCOMO
- Staffing

Monitoring and Measurement of Software Development

- **Monitoring** and **Measurement** in software development are essential for evaluating progress, ensuring quality, and achieving project objectives.
- These practices involve tracking metrics, analyzing data, and using insights to improve processes, deliverables, and team performance.
- **Key aspects of monitoring and measurements:**
  - **Defining objectives**
    - Why monitor and measure?
      - Esnure alignment with project goals
      - Track progress and identify deviations
      - Improve team productivity and product quality
    - Define metrics based on project priorities

contd..

- **Key metrics in software development**
  - **Process metrics**
    - **Velocity:** Amount of work completed in a sprint or iteration.
    - **Cycle Time:** Time taken to complete a task from start to finish.
    - **Lead Time**: Total time from task request to delivery.
    - **Work in Progress (WIP):** Tasks currently being worked on.
  - **Product metrics**
    - **Defect Density:** Number of defects per unit of code or functionality.
    - **Code Coverage:** Percentage of code tested by automated tests.

- **Technical Debt:** Amount of effort needed to fix codebase issues.
- **Mean Time to Failure (MTTF):** Average time the software operates before failing

– **Quality metrics**
  - **Customer Satisfaction (CSAT):** Feedback from end-users.
  - **Net Promoter Score (NPS):** Likelihood of users recommending the product.
  - **Defect Resolution Time:** Time taken to fix identified defects.

– **Team metrics**
  - **Team Morale:** Surveys and feedback loops to gauge team satisfaction.
  - **Collaboration Metrics:** Frequency and effectiveness of team communication.

Introduction to Software Estimation

- **What makes a successful project?**
  - Delivering
    - agreed functionality
    - on time
    - at the agreed cost with the required quality
  - Stages
    - set targets
    - Attempt to achieve targets

Difficulties in estimating due to complexity and invisibility of software.

Some problems with estimation

- **Subjective nature**
  - Underestimating the difficulties of small task and overestimating large projects.
- **Political pressures**
  - Managers may wish to reduce estimated costs in order to win support for acceptance of a project proposal (overestimate to create a comfort zone)
- **Changing technologies**
  - these bring uncertainties, especially in the early days when there is a 'learning curve'. Difficult to use the experience of previous projects.
- **Projects difference**
  - Experience on one project may not be applicable to another

Overestimation vs Underestimation

- **Overestimation** occurs when the time, effort, or resources required for a project or task are predicted to be more than what is actually needed.

- **Causes:**
  - **Uncertainty or unfamiliarity:** Lack of experience with a similar project may lead to inflated estimates.
  - **Buffering for risk:** Teams add extra time to account for potential risks or unknowns.
  - **Limited stakeholder trust:** Teams may overestimate to avoid being perceived as overly optimistic or missing deadlines.
  - **Excessive caution:** Fear of failure or penalties for delays can encourage padding estimates.

contd..

- **Consequences:**
  - **Wasted resources:** Excessive allocation of time, budget, or personnel may lead to inefficiencies.
  - **Delays in starting other projects:** Overestimation can delay subsequent initiatives in a portfolio.
  - **Decreased credibility:** If actual outcomes consistently fall short of estimates, trust between teams and stakeholders may erode.
- **Underestimation** happens when the time, effort, or resources required are predicted to be less than what is actually necessary.
- **Causes:**
  - **Optimism bias:** Overconfidence in team capabilities or project simplicity.
  - **Pressure to meet deadlines:** Stakeholders or clients push for shorter timelines, leading to overly optimistic estimates.

  - **Inexperience:** Lack of expertise or familiarity with the task can lead to an incomplete understanding of scope.
  - **Ignoring complexities:** Overlooking dependencies, risks, or potential challenges in the project.
- **Consequences:**
  - **Missed deadlines:** Unrealistic timelines lead to project delays.
  - **Budget overruns:** Insufficient planning results in unanticipated costs.
  - **Team burnout:** The pressure to meet underestimated timelines can overburden the team.
  - **Decreased quality:** Inadequate time for development and testing may compromise deliverables.
  - **Erosion of trust:** Repeated underestimations can lead to dissatisfaction among stakeholders and clients.

Basis for successful estimation

- **Use historical data:** Refer to past projects of similar scope for more accurate estimates.
- **Adopt estimation techniques** like Top-down or bottom-up approaches; PERT (Program Evaluation and Review Technique); Function point analysis: Estimate effort based on software functionalities.
- **Involve cross-functional teams:** Collaborative estimation brings diverse perspectives and mitigates biases.
- **Iterative adjustments:** Review and refine estimates as the project progresses and more information becomes available.
- **Buffer wisely:** Add realistic contingency buffers for known risks but avoid excessive padding.
- **Use tools:** Leverage project management software to model effort and resource allocation.
- **Stakeholder communication:** Align expectations early and communicate potential risks transparently.

A taxonomy of estimating methods

- **Bottom-up**
  - activity based, analytical (WBS – insert, amend, update, display, delete, print)
- **Parametric or algorithmic models**
  - Top-down approach
- **Expert opinion**
  - just guessing?
- **Analogy**
  - case-based, comparative
- **Albrecht function point analysis**

Parameters to be Estimated

- **Project Size:** It is a fundamental measure of work.
- Based on the **estimated size**, two parameters are estimated:
  - **Effort**
  - **Duration**
- **Effort:** The effort an individual can typically put in a month. It is measured in person-months.
- **Duration:** It is the measurement of the amount of time taken to develop the product. It is represented in months.

Project Size

- 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 used to measure project size:
  - Source Lines of Code (SLOC)
  - Function point (FP)
- **FP** is now-a-days favored over SLOC because of the many shortcomings of SLOC as mentioned below:
  - No precise definition (e.g., comment line, data declaration line to be included or not?)
  - Difficult to estimate at start of a project
  - Only a code measure
  - Programmer-dependent
  - Does not consider code complexity

Bottom-up versus Top-down

- **Bottom-Up Estimation** approach involves breaking the project down into smaller, manageable tasks or components.
  - Estimates are made for each task, and these are then aggregated to determine the overall project effort, cost, or duration.

- **Advantages:**
  - **Detailed and accurate:** Provides a granular view, leading to more precise estimates.
  - **Improved accountability:** Teams responsible for tasks contribute to the estimation process.
  - **Better risk identification:** Breakdowns reveal potential challenges at the task level.

contd..

- **Disadvantages:**
  - **Time-consuming:** Requires significant effort to decompose tasks and gather estimates.
  - **Complexity:** Managing a large number of small estimates can become overwhelming.
  - **Dependent on task clarity:** Accuracy depends on how well tasks are defined upfront.

- **Best for:**
  - Large, complex projects where tasks are well-defined.
  - Projects with significant historical data or prior experience in similar tasks.

contd..

- **Top-Down Estimation** approach starts with a high-level estimate based on the overall scope of the project. The total estimate is then divided among components or phases.

- **Advantages:**
  - **Faster:** Requires less initial effort, suitable for early project phases.
  - **Simpler:** Ideal when detailed requirements or tasks are not yet defined.
  - **Good for rough estimates:** Useful for budget approvals or feasibility studies.

- **Disadvantages:**
  - **Less accurate:** High-level assumptions may overlook specific complexities or risks.
  - **Prone to bias:** Relies heavily on expert judgment, which may not always be objective.
  - **Inflexible:** Hard to adjust if assumptions prove incorrect.

- **Best for:**
  - Early stages of a project when detailed information is unavailable.
  - Smaller or simpler projects with fewer variables.
  - Projects where similar historical data can guide estimates.

Empirical Size Estimation Techniques

- Based on **making an educated guess** of the project parameters. Prior experience with development of similar products is helpful.
- Two popular empirical estimation techniques are:
  - Expert judgment technique
  - Delphi cost estimation

- **Expert Judgement**
  - Here, an expert makes an educated guess of problem size after analyzing the problem thoroughly.
  - The expert estimates the cost of different component and combines them to arrive at the overall estimate.
  - Suffers from human errors and biasness. Additionally, prior experience is required. Absence of the same may lead to inadequate or wrong estimations.

Delphi Cost Estimation

- Overcomes some of the problems of expert judgement
- Team of **Experts** and a **coordinator.**
- **Experts** carry out estimation independently:
  - mention the unusual characteristic of the product which has influenced his estimation.
  - **Coordinator** notes down any extraordinary rationale and circulates among experts.
- Experts re-estimate.
- Experts never meet each other to discuss their viewpoints as many estimators may easily get Influenced.
- After several rounds of iterations, the coordinator compiles all the results and prepare the final estimates.

Parametric models

We are now looking more closely at four parametric models:

1. Albrecht/IFPUG (international function point user group) function points
2. Symons/Mark II function points
3. COSMIC (common software measurement consortium) function points
4. COCOMO81 and COCOMO II

Albrecht/IFPUG function points

- Proposed by Albrecht in 1983.
- It overcomes some of the shortcomings of the LOC metric
- It estimates the **size** of a software product directly from the **problem specification**.
- **Size** of a software product is **directly dependent** on the number of different **functions or features** it supports.
- Albrecht postulated that in addition to the number of basic functions that a software performs, the size is also dependent on the number of files and the number of interfaces.

Note: IFPUG- International FP User Group

Albrecht/IFPUG function points (contd..)

- FP metrics computes the size of a software product using three other characteristics as below:
  - UFP=(no. of inputs)*4 + (no. of outputs)*5 + (no. of inquiries)*4 + (no. of files)*10 + (no. of interfaces)*10

Where,

UFP: Unadjuisted Function Point

Number of inputs: Each data item input by the user

Number of outputs: The outputs considered refer to reports printed, screen outputs, error messages.

Number of inquiries: user commands which require specific action by the system.

Number of files: Each logical file is counted.

Number of interfaces: Used to exchange information with other systems. Examples of such interfaces are data files on tapes, disks, communication links with other systems etc.

(contd..)

- UFP is a gross indicator of the problem size as it assumes that each parameter is of average complexity, that is rarely true. Hence, UFP is refined by considering the complexity of these parameters.
- The complexities are broadly classified into simple, average and complex as shown below:

External user types	Low complexity	Medium complexity	High complexity
EI External input type	3	4	6
EO External output type	4	5	7
EQ External inquiry type	3	4	6
LIF Logical internal file type	7	10	15
EIF External interface file type	5	7	10

(contd..)

- Once UFP is calculated, Technical Complexity Factor, TCF, is calculated next. It refines the UFP measure by considering 14 other factors; assigned a value from 0 (No Influence) to 6 (Strong influence).
- The resulting numbers are summed, yielding the total Degree of Influence, DI.
  - TCF= 0.65 + 0.01*DI; where, 0<=DI<=84
  - TCF usually varies from 0.65 to 1.49.
- Finally, **FP=UFP*TCF**

Sample Problem

Consider a project with the following functional units:

i. Number of user inputs = 100 (30 simple inputs and 20 average inputs, rest complex)

ii. Number of user outputs = 80

iii. Number of user inquiries = 60 (20 simple inquiries and 15 average inquiries, rest complex)

iv. Number of user files = 06

v. Number of external interfaces = 04 (1 simple input and 3 average inputs)

Assuming all complexity adjustment factors as average. Calculate the function points for the project.

Solution

Considering the complexities and given data, Function Point can be calculated as below:

Step-1: UFP = (30*3 + 20*4 + 50*6) + 80*5 + (20*3 + 15*4 + 25*6) + 6*10 + (1*5 + 3*7)

= 470 + 400 + 270 + 60 + 26 = 1226

Step-2: Considering the complexity adjustment factors of average complexity, TCF can be computed as below:

TCF = 0.65 + (0.01*DI); where, 0<= DI<= 84

= 0.65 + (0.01*56) = 1.21

Step-3: Finally, the adjusted function point, FP = 1226 * 1.21 = 1483.46

Function Point Metric (contd..)

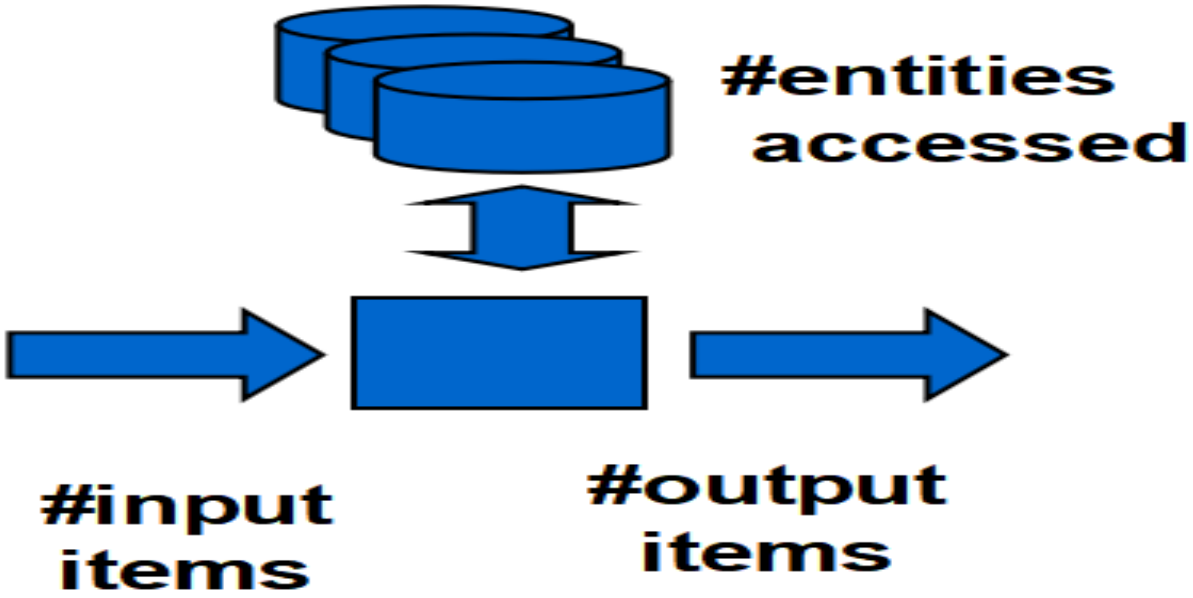
- Suffers from a major drawback as here the size of a function is independent of its complexity.
- Extend function point metric considers an extra parameter named as **Algorithm Complexity.**
- It says that greater is the effort required to develop it and therefore its size should be larger compared to simpler functions.
- Proponents claim that FP is language independent. Hence, Size can be easily derived from problem description
- Opponents claim that it is subjective which means that different people can produce different estimates for the same problem.

Function points Mark II

- Developed by Charles R. Symons
- 'Software sizing and estimating - Mk II FPA', Wiley & Sons, 1991.
- Builds on work by Albrecht
- Work originally for CCTA:
  - should be compatible with SSADM; mainly used in UK
- has developed in parallel to IFPUG FPs
- A simpler method

Function points Mk II (contd..)

- For each **transaction**, count
  - **data items input (N_i)**
  - **data items output ($N_{o)}$**
  - **entity types accessed (N_e)**



FP count = N_i * 0.58 + N_e * 1.66 + N_o * 0.26

- **UFP (Unadjusted Function Point):** a gross indicator of the problem size as it assumes that each parameter is of average complexity, that is rarely true. Hence, UFP is refined by considering the complexity of these parameters.
- **TCA (Technical Complexity Adjustment):** the assumption is, an information system comprises transactions which have the basic structures, as shown in previous slide.
- For each transaction, the UFPs are calculated:

  W_i X *(number of input data element types)* +
  W_e X *(number of entity types referenced)* +
  W_o X *(number of output data element types)*

  – W_i ,W_e ,W_o are weightings derived by asking developers the proportions of effort spent in previous projects developing the code dealing with input, accessing stored data and outputs.

  – **$W_i = 0.58$, $W_e = 1.66$, $W_o = 0.26$ (industry average)**

Exercise-1

- A cash receipt transaction in an account's 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.

  Calculate the unadjusted function points, using industry average weightings, for this transaction.

Soln: (0.58 X 3) + (1.66 X 2) + (0.26 X 1) = 5.32

Exercise-2

- In an annual maintenance contract subsystem is having a transaction which sets up details of new annual maintenance contract customers.
    1. Customer account number
    2. Customer name
    3. Address
    4. Postcode
    5. Customer type
    6. 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.
- **Calculate the number of unadjusted Mark II function points for the transaction described above using the industry average.**

Answer:

The function types are:

Input data types	6
Entities accessed	1
Output data types	1

UFP = (0.58 X 6) + (1.66 X 1) + (0.26 X 1)

= 5.4

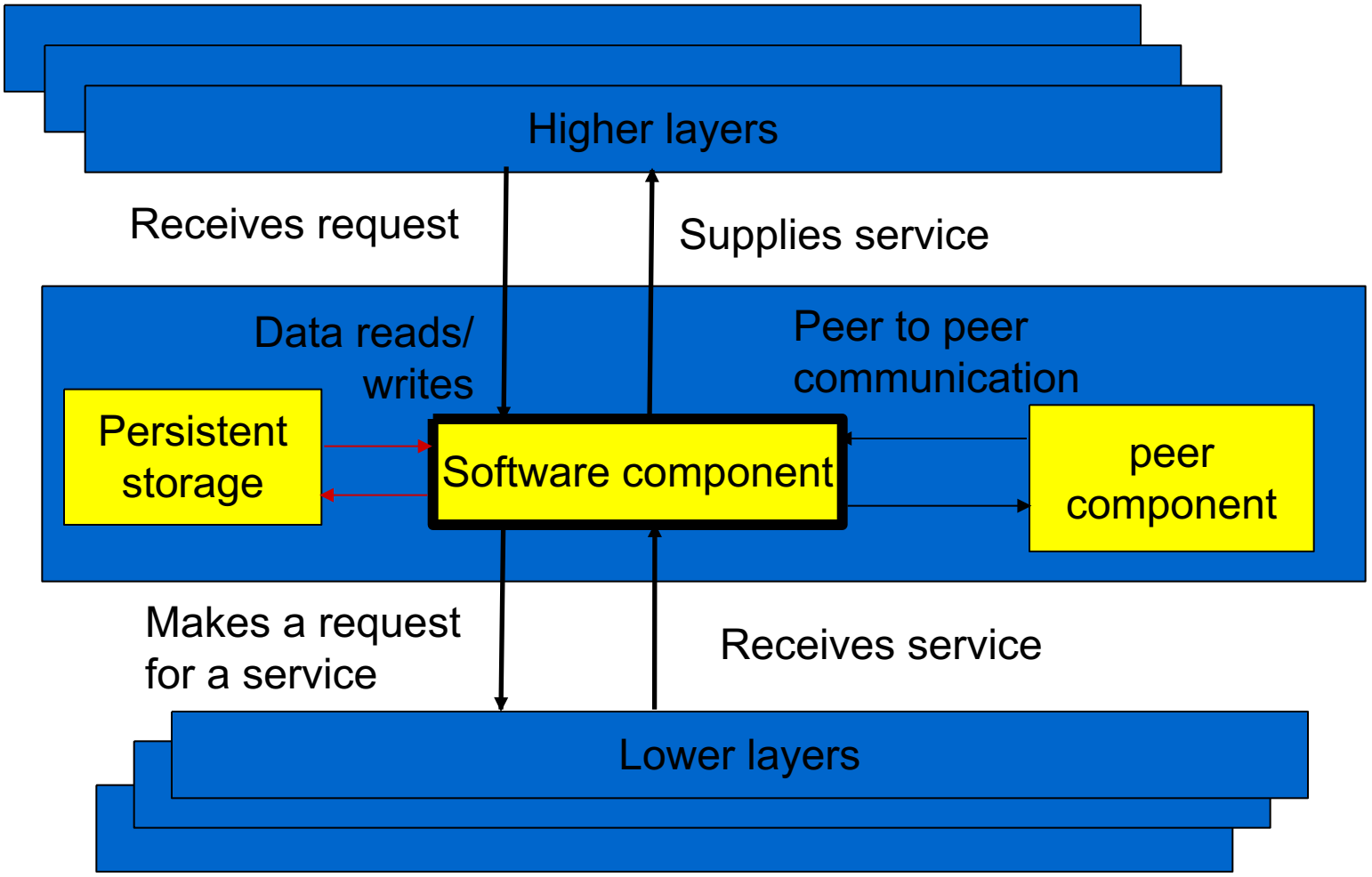
Function points for embedded systems

- Mark II function points, IFPUG function points were designed for **information systems environments**. They are not helpful for sizing **real-time or embedded systems**.
- **COSMIC-FFPs** (common software measurement consortium-full function point) attempt to extend concept to **embedded systems or real-time systems**.
- FFP method origins the work of two interlinked groups in Quebec, Canada.
- Embedded software seen as being in a particular ‘layer’ in the system that communicates with other layers and also other components at same level.

contd..

- COSMIC deals with by decomposing the system architecture into a hierarchy of software layers.
- The software component to be sized can receive requests the service from layers above and can request services from those below.
- There may be separate software components engage in peer-to-peer communication.
- Inputs and outputs are aggregated into data groups, where each data group brings together data items related to the same objects.

Layered software



COSMIC-FFPs

The following are counted: (Data groups can be moved in four ways)

- **Entries (E):** movement of data into software component from a higher layer or a peer component
- **Exits (X):** movements of data out to a user outside its boundary
- **Reads (R):** data movement from persistent storage
- **Writes (W):** data movement to persistent storage

- Each counts as **1 'COSMIC functional size unit' (Cfsu)**.
- The overall FFP count is derived by simply adding up the counts for each of the four types of data movement.

Exercise

- 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 the number of vehicles currently in the car park.
  - This count is kept on the 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 count is incremented. When the vehicle leaves the car park, a sensor detects the exit and reduce the count of vehicles.
- **Identify the entries, exits, reads and writes in this application.**

Exercise

Data movement	Type
Incoming vehicles sensed	E
Access vehicle count	R
Signal barrier to be lifted	X
Increment vehicle count	W
Outgoing vehicle sensed	E
Decrement vehicle count	W
New maximum input	E
Set new maximum	W
Adjust current vehicle count	E
Record adjusted vehicle count	W

Note: different interpretations of the requirements could lead to different counts. The description in the exercise does not specify to give a message that the car park is full or has spaces.

COCOMO81

- Based on industry productivity standards - database is constantly updated
- Allows an organization to benchmark its software development productivity
- **Basic model**

$$\textbf{effort} = \textbf{c} \times \textbf{size}^{\textbf{k}}$$

- c and k depend on the type of system: organic, semi-detached, embedded
- Size is measured in 'kloc' ie. Thousands of lines of code

Boehm in 1981, on a study of 63 projects, made this model. Of these only seven were business systems and so the model was based on other applications (non-information systems).

The COCOMO constants

System type	c	k
Organic (broadly, information systems, small team, highly familiar in-house environment)	2.4	1.05
Semi-detached (combined characteristics between organic and embedded modes)	3.0	1.12
Embedded (broadly, real-time, products developed has to operate within very tight constraints and changes to system is very costly)	3.6	1.20

- k exponentiation – 'to the power of…' adds disproportionately more effort to the larger projects takes account of bigger management overheads

- **Effort**

$$\mathbf{c \times size^{k}}$$

Organic : Effort = $2.4(KLOC)^{1.05}$ PM

Semidetached : Effort = $3.0(KLOC)^{1.12}$ PM

Embedded : Effort = $3.6(KLOC)^{1.20}$ PM

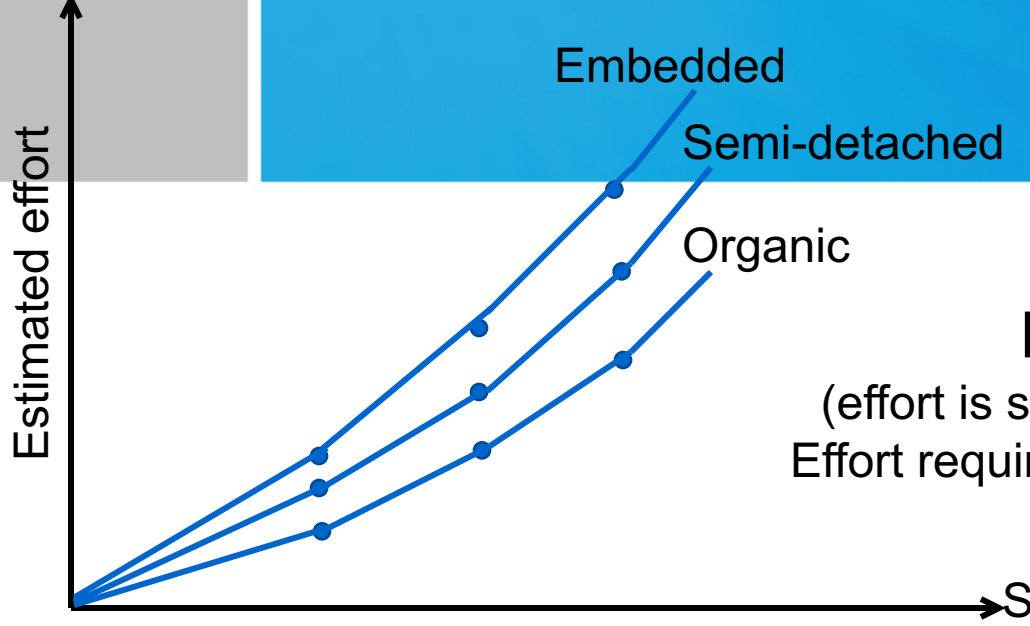
- **Development Time**

$$\mathbf{T_{dev} = a \times (Effort)^{b}}$$

Organic : $2.5(Effort)^{0.38}$

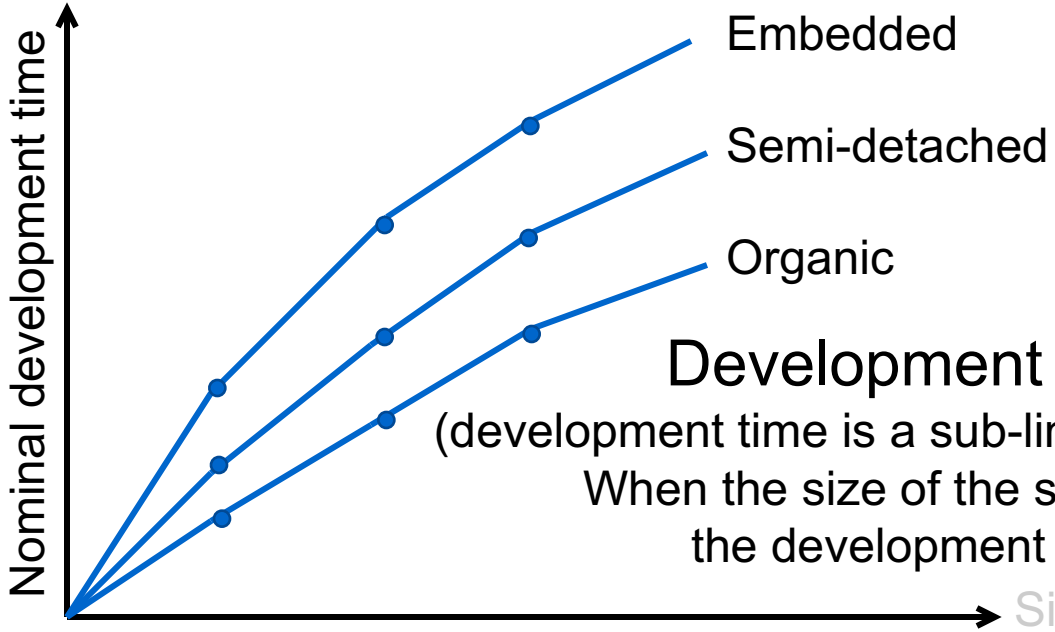
Semidetached : $2.5(Effort)^{0.35}$

Embedded : $2.5(Effort)^{0.32}$



Effort vs. product size

(effort is super-linear in the size of the software)
Effort required to develop a product increases very rapidly with project size.



Development time vs. product size

(development time is a sub-linear function of the size of the product)
When the size of the software increases by two times, the development time increases moderately

Exercise-1

- Assume that the size of an organic type software product is estimated to be 32,000 lines of source code. Assume that the average salary of a software developer is Rs.50,000 per month. Determine the effort required to develop the software product, the nominal development time, and the staff cost to develop the product.

Soln:

Effort = $2.4 \text{ X } 32^{1.05} = 91$ pm

Nominal development time = $2.5 \text{ X } 91^{0.38} = 14$ months

Staff cost required to develop the product

91 X Rs. 50,000 = Rs. 45,50,000

Exercise-2

- Two software managers separately estimated a given product to be of 10,000 and 15,000 lines of code respectively. Bring out the effort and schedule time implications of their estimation using COCOMO. For the effort estimation, use a coefficient value of 3.2 and exponent value of 1.05. For the schedule time estimation, the similar values are 2.5 and 0.38 respectively. Assume all adjustment multipliers to be equal to unity.

Soln:

For 10,000 LOC

$$\text{Effort} = 3.2 \text{ X } 10^{1.05} = 35.90 \text{ PM}$$

$$\text{Schedule Time} = T_{dev} = 2.5 \text{ X } 35.90^{0.38} = 9.75 \text{ months}$$

For 15,000 LOC

$$\text{Effort} = 3.2 \text{ X } 15^{1.05} = 54.96 \text{ PM}$$

$$\text{Schedule Time} = T_{dev} = 2.5 \text{ X } 54.96^{0.38} = 11.46 \text{ months}$$

NB: Increase in size drastic increase in effort but moderate change in time.

COCOMO II

- An updated version of COCOMO.
- There are different COCOMO II models for estimating at the 'early design' stage and the 'post architecture' stage when the final system is implemented. We'll look specifically at the first.
- The core model is:

$$\mathbf{pm = A(size)^{(sf)} \times (em_1) \times (em_2) \times (em_3) \ldots.}$$

where,

  - **pm** = person-months,
  - **A** is 2.94,
  - **size** is number of thousands of lines of code,
  - **sf** is the scale factor; **sf = B + 0.01 X Σ** (*exponent driver ratings*)
  - **em** is an effort multiplier

COCOMO II Scale factor

- Boehm et al. have refined a family of cost estimation models.
- The key one is COCOMO II. It uses multipliers and exponent values.
- Based on five factors which appear to be particularly sensitive to system size.

1. **Precedentedness (PREC)**. Degree to which there are past examples that can be consulted, else uncertainty
2. **Development flexibility (FLEX)**. Degree of flexibility that exists when implementing the project
3. **Architecture/risk resolution (RESL)**. Degree of uncertainty about requirements, liable to change
4. **Team cohesion (TEAM)**. Large dispersed
5. **Process maturity (PMAT)** could be assessed by CMMI, more structured less uncertainty
6. – see Section 13.8

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

Example of scale factor

- A software development team is developing an application which is very similar to previous ones it has developed. A very precise software engineering document lays down very strict requirements. PREC is very high (score 1.24). FLEX is very low (score 5.07). The good news is that these tight requirements are unlikely to change (RESL is high with a score 2.83). The team is tightly knit (TEAM has high score of 2.19), but processes are informal (so PMAT is low and scores 6.24).

Soln:

sf = B + 0.01 × Σ scale factor values

= 0.91 + 0.01 × (1.24 + 5.07 + 2.83 + 2.19 + 6.24) = **1.0857**

- If system contained 10 kloc then estimate would be

***effort* = *c* (*size*)^{k} = 2.94 x 10$^{1.0857}$ = 35.8 person-months**

- Using exponentiation ('to the power of') adds disproportionately more to the estimates for larger applications

B = 0.91(constant), c = 2.94 (average)

Example-2

- A new project has ‘average’ novelty for the software supplier that is going to execute it and thus given a nominal rating on this account for precedentedness. Development flexibility is high, requirements may change radically and so risk resolution exponent is rated very low. The development team are all located in the same office, and this leads to team cohesion being rated as vey high, but the software house as a whole tends to be very informal in its standards and procedures and the process maturity driver has therefore been given a rating of ‘low’.

  (i) What would be the scale factor (*sf*) in this case?

  (ii) What would the estimate effort if the size of the application was estimated as in the region of 2000 lines of code?

Assessing the scale factors

Factor	Rating	Value
PREC	nominal	3.72
FLEX	high	2.03
RESL	very low	7.07
TEAM	very high	1.10
PMAT	low	6.24

(i) The overall scale factor = sf = B + 0.01 X Σ (exponent factors)

= 0.91 + 0.01 X (3.72 + 2.03 + 7.07 + 1.10 + 6.24

= 0.91 + 0.01 X 20.16 = 1.112

(ii) The estimated effort = c $(\text{size})^{k}$ = 2.94 X $2^{1.112}$ = 6.35 staff-months

Effort multipliers (COCOMO II - early design)

- As well as the scale factor effort multipliers are also assessed:

RCPX	Product reliability and complexity
RUSE	Reuse required
PDIF	Platform difficulty
PERS	Personnel capability
PREX	Personnel experience
FCIL	Facilities available
SCED	Schedule pressure

Effort multipliers
(COCOMO II - early design)

Table-3

	Extra low	Very low	Low	Nom-inal	High	Very high	Extra high
RCPX	0.49	0.60	0.83	1.00	1.33	1.91	2.72
RUSE			0.95	1.00	1.07	1.15	1.24
PDIF			0.87	1.00	1.29	1.81	2.61
PERS	2.12	1.62	1.26	1.00	0.83	0.63	0.50
PREX	1.59	1.33	1.12	1.00	0.87	0.74	0.62
FCIL	1.43	1.30	1.10	1.00	0.87	0.73	0.62
SCED		1.43	1.14	1.00	1.00	1.00	

Example-1

- Say that a new project is similar in most characteristics to those that an organization has been dealing for some time
- **except**
  - the software to be produced is exceptionally complex and will be used in a safety critical system.
  - The software will interface with a new operating system that is currently in beta status.
  - To deal with this the team allocated to the job are regarded as exceptionally good, but do not have a lot of experience on this type of software.

Example -continued

Refer Table-3

- RCPX very high 1.91
- PDIF very high 1.81
- PERS extra high 0.50
- PREX nominal 1.00
- All other factors are nominal
- Say estimate is 35.8 person months
- With effort multipliers this becomes 35.8 x 1.91 x 1.81 x 0.5 = 61.9 person months

Example-2

- A software supplier has to produce an application that controls a piece of equipment in a factory. A high degree of reliability is needed as a malfunction could injure the operators. The algorithms to control the equipment are also complex. The product reliability and complexity are therefore rates as very high. The company would lie to take opportunity to exploit fully the investment that they made in the project by reusing the control system, with suitable modifications, on future contracts. The reusability requirement is therefore rate as very high. Developers are familiar with the platform and the possibility of potential problems in that respect is regarded as low. The current staff are generally very capable and are rated as very high, but the project is in a somewhat novel application domain for them so experience s rated as nominal. The toolsets available to the developers are judged to be typical for the size of company and are rated nominal, as it is the degree of schedule pressure to meet a deadline.

Given the data table-3

(i) What would be the value for each of the effort multipliers?

(ii) What would be the effort of the said project if the size of the project is 100KLOC? (Assume Scale Factor as 1.112)

Soln:

Factor	Description	Rating	Effort multiplier
RCPX	Product reliability and complexity	Very high	1.91
RUSE	Reuse	Very high	1.15
PDIF	Platform difficulty	Low	0.87
PERS	Personnel capability	Very high	0.63
PREX	Personnel experience	Nominal	1.00
FCIL	Facilities available	Nominal	1.00
SCED	Required development schedule	nominal	1.00

New development effort multipliers (dem)

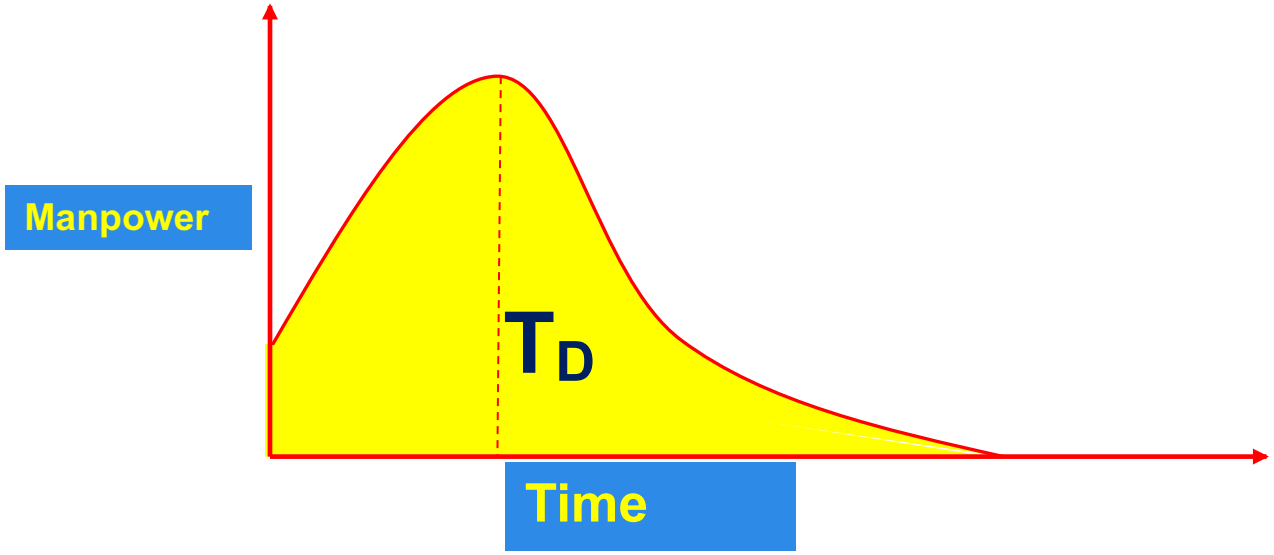
- According to COCOMO, the major productivity drivers includes:
  - **Product attributes:** required reliability, database size, product complexity
  - **Computer attributes:** execution time constraints, storage constraints, virtual machine (VM) volatility
  - **Personnel attributes:** analyst capability, application experience, VM experience, programming language experience
  - **Project attributes:** modern programming practices, software tools, schedule constraints

COCOMO II Post architecture effort multipliers

Modifier type	Code	Effort multiplier
Product attributes	RELY	Required software reliability
	DATA	Database size
	DOCU	Documentation match to life-cycle needs
	CPLX	Product complexity
	REUSE	Required reusability
Platform attributes	TIME	Execution time constraint
	STOR	Main storage constraint
	PVOL	Platform volatility
Personnel attributes	ACAP	Analyst capability
	AEXP	Application experience
	PCAP	Programmer capabilities
	PEXP	Platform experience
	LEXP	Programming language experience
	PCON	Personnel continuity
Project attributes	TOOL	Use of software tools
	SITE	Multisite development
	SCED	Schedule pressure

Staffing

- Norden was one of the first to investigate staffing pattern:
  - Considered general research and development (R&D) type of projects for efficient utilization of manpower.
- Norden concluded:
  - Staffing pattern for any R&D project can be approximated by the Rayleigh distribution curve



Rayleigh-Norden Curve

Putnam's Work

- Putnam adapted the Rayleigh-Norden curve:
  - Related the number of delivered lines of code to the effort and the time required to develop the product.
  - Studied the effect of schedule compression:

$$pm_{new} = pm_{org} \times \left(\frac{td_{org}}{td_{new}} \right)^4$$

  - Where:
    - pm = effort in person-month
    - td = time to develop
    - org: original estimate
    - new: new estimate

contd..

- If the estimated development time using COCOMO formulas is 1 year:
  - Then to develop the product in 6 months, the total effort required (and hence the project cost) increases by 16 times. **Why?**
- The extra effort can be attributed to the increased communication requirements and the free time of the developers waiting for work.
- The project manager recruits many developers hoping to complete the project early but becomes very difficult to keep these additional developers continuously occupied with work.
- Implicit in the schedule and duration estimated arrived at using COCOMO model, is the fact that all developers can continuously be assigned work.
- However, when many developers are hired to decrease the duration significantly, it becomes difficult to keep all developers busy all the time. The simultaneous work is getting restricted.

Example

- The nominal effort and duration of a project is estimated to be 1000 pm and 15 months. This project is negotiated to be £200,000. This needs the product to be developed and delivered in 12 months time. What is the new effort and the new cost that needs to be negotiated.

Soln: The project can be classified as a large project. Therefore, the new cost to be negotiated can be given by Putnam's formula as

- New effort = 1,000 X $(15/12)^4$ = 2441.40 PM
- New Cost = new effort X cost = 2441.40 X £200000
  = £488281

Boehm's Result

- There is a limit beyond which a software project cannot reduce its schedule by buying any more personnel or equipment.
  - This limit occurs roughly at 75% of the nominal time estimate for small and medium sized projects
  - If a project manager accepts a customer demand to compress the development schedule of a project (small or medium) by more than 25% , he is very unlikely to succeed.
  - The reason is, every project has a limited number of activities which can be carried out in parallel, and the sequential work can not be speeded up by hiring a greater number of additional developers.

Capers Jones' Estimating Rules of Thumb

- Empirical rules: (IEEE journal – 1996)
  - Formulated based on observations
  - No scientific basis
- Because of their simplicity:
  - These rules are handy to use for making off-hand estimates.
  - Not expected to yield very accurate estimates.
  - Give an insight into many aspects of a project for which no formal methodologies exist yet.

Capers Jones' Rules

- ***Rule 1: SLOC-function point equivalence:***
  - One function point = 125 SLOC for C programs.
- **Rule 2: Project duration estimation:**
  - Function points raised to the power 0.4 predicts the approximate development time in calendar months.
- **Rule 3: Rate of requirements creep:**
  - User requirements creep in at an average rate of 2% per month from the design through coding phases.
- **Rule 4: Defect removal efficiency:**
  - Each software review, inspection, or test step will find and remove 30% of the bugs that are present. (Companies use a series of defect removal steps like requirement review, code inspection, code walk-through followed by unit, integration and system testing.
  - A series of ten consecutive defect removal operations must be utilized to achieve good product reliability.)

Capers Jones' Rules

- **Rule 5: Project manpower estimation:**
  - The size of the software (in function points) divided by 150 predicts the approximate number of personnel required for developing the application.
  - (For a project size of 500 FPs the number of development personnel will be 500/150 = 4, without considering other aspects like use of CASE tools, project complexity and programming languages.)
- **Rule 6: Software development effort estimation:**
  - The approximate number of staff months of effort required to develop a software is given by the software development time multiplied with the number of personnel required. (using rule 2 and 5 the effort estimation for the project size of 150 FPs is 8 X 1 = 8 person-months)
- **Rule 7: Number of personnel for maintenance**
  - *Function points divided by 500 predicts the approximate number of personnel required for regular maintenance activities.* (as per Rule-1, 500 function points is equivalent to about 62,500 SLOC of C program, the maintenance personnel would be required to carry out minor fixing, functionality adaptation **ONE**.)

Illustration:

Size of a project is estimated to be 150 function points.

Rule-1: 150 X 125 = 18,750 SLOC

Rule-2: Development time = $150^{0.4} = 7.42 \approx 8$ months

Rule-3: The original requirement will grow by 2% per month i.e., 2% of 150 is 3 FPs per month.

If the duration of requirements specification and testing is 5 months out of total development time of 8 months, the total requirements creep will be roughly 3 X 5 = 15 function points.

The total size of the project considering the creep will be 150 + 15 = 165 function points and the manager need to plan on 165 function points.