

---

# **Advanced Software Engineering** **(CS6401)**

**Autumn Semester (2021-2022)**

**Dr. Judhistir Mahapatro**  
**Department of Computer Science and**  
**Engineering**  
**National Institute of Technology Rourkela**

---

# **Software Project Management**

# Topics covered in Previous Lecture:

---

- ▶ Empirical Estimation
- ▶ Heuristic estimation technique- COCOMO

# Project Estimation Techniques

---

- Estimation of various project parameters is an important planning activity
- Different parameters of a project
  - Project Size
  - Effort required
  - Project Duration
  - Cost
- Accurate estimation of these parameters is important
- Not only help in quoting an appropriate project cost to the customer, but also form the basis for resource planning and scheduling



# Main categories:

---

- Empirical Estimation Techniques
  - It is based on making an educated guess of the project parameters
  - Prior experience with development of similar products is helpful
  - Although Empirical Estimation is based on common sense and subjective decisions
  - Two such formalizations such as **Expert Judgement** and **Delphi Technique**



# Expert Judgement

---

- It is a widely used size estimation technique.
- An expert makes an educated guess about the problem size after analyzing the problem thoroughly.
- Expert estimates the cost of different components (i.e. modules or subsystems)



# Expert Judgement

---

- Shortcomings
  - Outcome subject to human error and individual bias.
  - Expert may overlook some factors inadvertently.
  - Expert may not have relevant experience and knowledge of all aspects of a project.

**Example:** expert is conversant with database but not with programming language and which leads to the size estimation far from being accurate.



# Expert Judgement

---

- Refined form is the estimation made by a group of experts
  - lack of familiarity with a particular aspect
  - Personal bias
  - Desire to win the contract through overly optimistic estimates
  - Individual oversight is minimized when a estimation is done by a group of experts.





# Expert Judgement

---

- A group of experts still may exhibit bias
  - Due to political or social considerations, entire expert team may be biased
  - Decision may be dominated by overly assertive members



# Delphi Cost Estimation

---

- Team consisting of a group of experts and a coordinator.
- Coordinator provides a copy of the SRS to each expert and a form for recording his cost estimates.
- Anonymously carry out the task and submit it to coordinator
- Estimators mention any unusual characteristics of the product which has influenced their estimates



# Delphi Cost Estimation

---

- Coordinator prepares the summary of the responses of the estimators and also includes any unusual rationale noted by any of the estimators.
- Prepared summary information is distributed to the estimators.
  - Based on this summary, estimators re-estimate the cost.



# Delphi Cost Estimation

---

- This process is iterated for several rounds.
    - **no discussion** among the estimators is allowed during the process.
    - Otherwise, many estimators will be influenced by the **rationale of a senior or experienced** estimators.
  - After several iterations, coordinator compiles the results and preparing the final estimate.
  - Though it consumes more time and effort, removes unjustly be influenced by overly **assertive and senior members**.
- 



# Heuristic Techniques

---

- It assumes that the **relationships** that exist among the different project **parameters** can be modeled using suitable **mathematical expression**.
- Heuristic estimation models can be divided into two broad categories:
  - Single variable models
  - Multivariable models



# Heuristic Techniques

---

- Single variable model **assume that** various project characteristics can be predicted based on a single previously estimated basic (independent) characteristic of the software (such as project size).

$$\text{Estimated parameter} = C_1 \times e^{d_1}$$

‘e’ represents a characteristic of the software that has already been estimated (independent variable)

- Estimated parameter is the dependent parameter (to be estimated). For example, **effort, project duration, staff size**, etc.
- 



# Heuristic Techniques

---

- A multivariable model assumes that a parameter can be predicted based on the values of more than one independent parameters.

$$\text{Estimated parameter} = C_1 \times e_1^{d1} + C_2 \times e_2^{d2} + \dots$$

*where,  $e_1$  and  $e_2$  are the basic independent variables of the software already estimated.*

The constants  $C_1$  and  $d_1$  are usually determined from the analysis of the historical data (of past projects).

- Multivariable estimation model is more accurate over single variable estimation model.
- 



# COCOMO – A Heuristic Estimation Technique

---

- COnstructive COst estimation MOdel (COCOMO) was proposed by Boehm.
  - It prescribes a three stage process for project estimation.
  - First stage, an initial estimate is arrived at.
  - Over the next two stages, the initial estimate is refined to arrive at a more accurate estimate.
  - It uses both single and multivariable estimation models at different stages of estimation.
  - Three stages of COCOMO estimation technique are – basic COCOMO, intermediate COCOMO, and complete COCOMO.
- 





# Basic COCOMO Model

---

- Boehm postulated that any software development project can be classified into one of the following three categories based on the development complexity – *organic*, *semidetached*, and *embedded*.
- Different sets of formulas for different categories to estimate the effort and duration from the size estimate.



# Basic COCOMO Model

---

- Three basic classes of software development projects
    - According to Boehm, consider **not** only the characteristics of the product **but also** those of the development team and development environment.
    - Three product development classes correspond to development of application (ex: **data processing programs**), utility (ex: **linkers and compilers** etc), and system software ( ex: **operating system**).
    - System programs interact directly with the hardware and programming complexities arise out of the requirement meeting timing constraints and concurrent processing tasks.
- 



# Basic COCOMO Model

---

- According to Brooks, Product development complexity (as difficult to write) for the three categories (application, utility, and system software) are **1:3:9**



# Basic COCOMO Model

---

## Organic:

If the project is a **well-understood** application program, the development team is **reasonably small**, and the team members are **experienced** in developing similar types of projects.

## Semidetached:

If the development team consists of **experienced** and **inexperienced** staff. Team members may have **limited experience** on related systems but may be **unfamiliar** with some aspects of the system being developed.



# Basic COCOMO Model

---

## **Embedded:**

If the project is strongly coupled to **hardware**, or if **stringent regulations** on the operational procedure exist.

Team members may have **limited experience** on related systems but may be unfamiliar with some aspects of the system.



# Basic COCOMO Model

---

- For three product categories
  - Boehm provides different sets of expressions to **predict** the effort (in units of person-months) and development time from the size estimation given in kilo lines of source code (KLSC).



# Basic COCOMO Model

---

- **One person month** is the effort an individual can typically put in a month.
  - Implicitly takes into account the productivity losses that normally occur due to time lost in holidays, weekly offs, coffee break, etc.
- **Person-month (PM)** is considered to be an appropriate unit for measuring effort, because developers are typically assigned to a project for a certain number of months.



# Basic COCOMO Model

- Effort estimation of 100 PM does not imply that 100 persons should work for 1 month. Neither does it imply that 1 should be employed for 100 months to complete the project.
- Effort estimation simply denotes **the area under the person-month curve** for the project.
  - Different personnel may work at different points in the project development.
- The number of personnel working on the project usually increases or decreases by an integral number.





# Basic COCOMO Model

- It is a single variable heuristic model
- The expression for the basic COCOMO estimation

$$\text{Effort} = a_1 \times (\text{KLOC})^{a_2} \quad \text{PM}$$

$$T_{\text{dev}} = b_1 \times (\text{Effort})^{b_2} \quad \text{months}$$

Where, KLOC is the estimated size of the software product expressed in Kilo Lines of Code.

- $a_1, a_2, b_1, b_2$  are constants.
  - $T_{\text{dev}}$  is the estimated time to develop the software, expressed in months.
  - Effort is the total effort required to develop the software product, expressed in PMs
- According to Boehm, every line of source code should be calculated as one LOC irrespective of the actual number of instructions on that line.

# Basic COCOMO Model

---

- For the three classes of software products, the formulas for estimating the effort based on the code size are as follows:

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

**Semi-detached:**        $\text{Effort} = 3.0(\text{KLOC})^{1.12}$     PM

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

- For the three classes of software products, the formulas for estimating the development time based on the effort are as follows:

**Organic:**                       $T_{\text{dev}} = 2.5(\text{Effort})^{0.38}$        months

**Semi-detached:**        $T_{\text{dev}} = 2.5(\text{Effort})^{0.35}$        months

**Embedded:**                 $T_{\text{dev}} = 2.5(\text{Effort})^{0.32}$        months

---



# Basic COCOMO Model

---

- The effort is somewhat **superlinear** (that is, slope of the curve  $> 1$ ) in the size of the software product. This is because the exponent in the effort expression is more than 1.
- The effort required to develop a product increases rapidly with project size.
- The development time is a **sublinear** function of the size of the product.
  - That is, when the size of the product increases by two times, the time to develop the product does not double but rises moderately.



# Basic COCOMO Model

---

- Duration curve does not **increase superlinearly**.
- The curve to behave similar to those in the effort-size plots.
  - This anomaly can be explained by the fact that COCOMO assumes that a project development is carried out not by a single person but a team of developers.
- For a project of any given size, the development time is roughly the same for all the three categories of product.



# Basic COCOMO Model

---

- According to COCOMO formulas, embedded programs require much higher effort than either application or utility programs.
- We can interpret it to mean that there is **more scope for parallel activities** for system programs than those in utility or application programs.



# Basic COCOMO Model

---

- Cost Estimation
    - Project cost can be obtained by multiplying the estimated effort (in man-month) by the manpower cost per month.
    - Assumption that the entire project cost is incurred on account of the manpower cost alone.
    - In addition to manpower cost, a project would incur **overhead costs** due to hardware, software required for the project, and company overheads for administration, office space, electricity, etc.
- 



# Intermediate COCOMO Model

- The basic COCOMO model assumes that effort and development time are **functions of the product size** alone.
- Other parameters affect the effort and development time
  - Example: Effort to develop a product would vary depending on the development environment
- All relevant parameter must be taken into account
- It recognizes this fact and refines the initial estimates



# Intermediate COCOMO Model

---

- **Note:** the intermediate COCOMO model **refines** the **initial estimate** obtained using the basic COCOMO expressions by **scaling the estimate up or down** based on the evaluation of a set of attributes of software development.





# Intermediate COCOMO Model

---

- It uses a set of 15 cost drivers (multipliers) that are determined based on various attributes of software development
- These cost drivers are multiplied with the initial cost and effort estimates (obtained from basic COCOMO) appropriately scale those to up or down.
- **Example:** if modern programming practices are used, then a cost driver value  $< 1$  is multiplied to downward the scale of initial estimate.



# Intermediate COCOMO Model

---

- If stringent reliability requirements on the software, initial estimates are scaled upward.
- Boehm requires the managers to rate 15 different parameters for a project on a scale of 1 to 3.
- With each grading, suggested appropriate cost drivers to refine the initial estimates.



# Intermediate COCOMO Model

---

- Cost drivers identified by Boehm can be classified as being attributes of following items:

**Product:** the characteristics of the product that are considered include the inherent complexity of the product, reliability requirements etc.

**Computer:** execution speed required, storage space required etc.

**Personnel:** Experience level of personnel, their programming capability, analysis capability

**Development environment:** the automation (CASE) tools used for software development.

---



# Complete COCOMO Model

- Basic and intermediate model considers a software product as a single homogeneous entity.
- Large systems are made up of several **sub-systems**.
- Subsystems have different characteristics.
- Some considered to be **Organic**, **semidetached** and **embedded**.
- Not only their development complexity different, but some subsystems the **reliability requirement** may be high, some of the **development team** have no prior experience of similar development, and so on.



# Complete COCOMO Model

---

- The complete COCOMO model considers these characteristics of the subsystems and estimates the effort and development time as the **sum of the estimates for the individual sub-systems**.
  - It reduces the margin of error in the final estimate.
  - To further improve the accuracy of the results, the different parameter values of the model can be fine-tuned and validated against an organizations **historical project database** to obtain more accurate estimation.
- 



# Complete COCOMO Model

---

- COCOMO are not totally accurate and lack full scientific justification.
  - It is required for an engineering approach to software project management.
  - Companies consider computed cost estimates to be satisfactory, if these are within the 80 percentage of the final cost.
  - These are gross approximations.
- 



# COCOMO2

- Present day software projects are **much larger** in size and **reuse of existing software** to develop a new
- Component-based development and service-oriented architecture
- New life cycle models and development paradigms are being developed for **web-based** and **component-based software**
- During 1980s rarely any program was interactive, and graphical user interfaces were almost non-existence.
- Effort spent on developing GUI part is often as much as the effort spent on developing the actual functionality of the software
- Boehm proposed COCOMO2 in 1995



# COCOMO2

---

- It provides three models to arrive at increasingly accurate cost estimations.
- These models can be used to estimate project costs at different phases of the software product.





## **1. Application composition model:**

- It can be used to estimate the cost for prototype development.
- a prototype is usually developed to resolve user interface issues.

## **2. Early design model:**

- estimation of cost at the architectural design stage

## **3. Post-architecture model:**

- cost estimation during detailed design and coding stages.
  - Post-architecture model can be considered as an update of the original COCOMO.
- 



- Other two models help consider the these factors:
  - interactive and GUI driven
  - GUI development constitute a significant part of the overall development effort
  - Interactive program concerns several issues that affect productivity such as the extent of reuse.

# COCOMO2

---

- **Application composition model**
  - It is based on counting the # of screens, reports and modules(components)
  - Each of these components is considered to be an object
  - These are used to compute the object points of the application
  - Effort is estimated in the application composition as follows:
    1. Estimate the number of screens, reports and modules from an analysis of the SRS document.



# COCOMO2

---

## **Application composition model**

2. Determine the complexity level of each screen and report, and rate these as a simple, medium or difficult.
  - The complexity of the screen or a report is determined by the number of tables and views it contains.



# COCOMO2

---

**3.** The weights have been designed to correspond to the amount of effort required to implement an instance of an object at the assigned complexity class.

Table1: SCREEN complexity assignments for the data tables

#of views	Tables<4	Tables<8	Tables >= 8
< 3	Simple	Simple	Medium
3 to 7	Simple	Medium	Difficult
> 8	Medium	Difficult	Difficult



# COCOMO2

---

4. Add all the assigned complexity values for the object instances together to obtain the object points.

Table2: REPORT complexity assignments for the data tables

#of sections	Tables<4	Tables<8	Tables >= 8
0 or 1	Simple	Simple	Medium
2 or 3	Simple	Medium	Difficult
4 or more	Medium	Difficult	Difficult



# COCOMO2

---

Table3: Table of Complexity Weights for each class for each object types

Object type	Simple	Medium	Difficult
Screen	1	2	3
Report	2	5	8



5. Estimate percentage of reuse expected in the system. Then, evaluate New Object-Point (NOP) count as follows,

$$NOP = \frac{(Object - points)(100 - \% of reuse)}{100}$$



# COCOMO2

---

**6.** Determine the productivity which depends on the experience of the developers as well as the maturity of the CASE environment used.

Table4: productivity table

Developers experience	Very low	low	Nomina	High	Very high
CASE maturity	very low	low	Nominal	High	very high
PRODUCTIVITY	4	7	13	25	50



# COCOMO2

---

7. Finally, the estimated effort in person-months is computed as

$$\text{Effort} = \frac{NOP}{PRODUCTIVITY}$$



# COCOMO2

---

Early design model:

- UFP are converted into Source Lines of Code
- Typical programming environment, each UFP would correspond to about 128 lines of C, 29 lines of C++, 329 lines of assembly code
- Off course, conversion depends on factors such as extent of reusable libraries supported.
- Seven cost drivers that characterise the post-architecture model are used.
- These are rated on seven point scale.

# COCOMO2

---

- Cost drivers include product reliability and complexity, the extent of reuse, platform sophistication, personal experience, CASE support, and schedule.

$$\text{Effort} = KLOC \times \prod_i \text{CostDriver}_i$$

# COCOMO2

---

- Post-architecture model
- The effort is calculated using the following formula, which is similar to the original COCOMO model.

$$\text{Effort} = a * KLOC^b \times \prod_i CostDriver_i$$

- It differs from the original COCOMO model in the choice of the set of cost drivers and the range of the values of the exponent ***b***.
- The exponent ***b*** can take values in the range of 1.01 to 1.26