Chapter 4.

THE DEFINITION PHASE

PART 2_1

Summary

5. Software Cost Estimation

5.1. Cost Estimation Objectives

Cost estimation for a Software Project has as essential scope the determination in a *predictive manner* of the resources necessary for the developing process of the product.

OBJECTIVES

- (1)To establish a *budget* for the *project development.*
- (2) To supply the *means* for the control of the *project costs.*
- (3) To *monitor* the project development based on planned budget by comparing the planned and the actual *costs.*
- (4) To establish a *database* referring to costs, to be used in future evaluations.
- (5) To *correlate* costs estimation with the planning *activities.*

5.2 The Resources of a Software Project

- To develop a SW project is necessary to invest resources.
- There are three *types* of resources: *software, hardware* and *human*.
- **(1) Software Resources** consist in:
    - (1) *Programs.*
    - (2) *Environments.*
    - (3) *Developing tools* together with the corresponding ***using licenses***.
    - (4) Different categories of *software support resources* direct implied in a software project development:
        - (4.1) Operating systems
        - (4.2) Different types of servers
        - (4.3) Project management tools
        - (4.4) Support tools (text editors, electronic mail, grupware systems, network software)
        - (4.5) Programming tools (compilers, SGBD's)
        - (4.6) Developing tools (Case tools)
        - (4.7) Testing tools

        - (4.8) Debugging tools
        - (4.9) Simulation tools
        - (4.10) Documentation tools
            - Some of the *tools* can have a very high degree of specificity and they are **not available** on the market.
            - This tools cold be developed as parts of the project, eventually, depending on the real situation, before the project development as separate *contracts*.
            - In this case they are not considerate as resources but *auxiliary parts* of the project.
- **(2) Hardware resources** can be classified by type and by destination.
    - By *type* there are:
        - (2.1) Computer systems with their configurations
        - (2.2) Computer networks
        - (2.3) Communication lines and devices
        - (2.4) Printers
        - (2.5) Scanners
        - (2.6) Special back-up units
        - (2.7) Multimedia equipment
        - (2.8) Special equipment
    - By *destination* there are:
        - (2.9) Development resources
        - (2.10) Testing resources
        - (2.11) Prototyping resources
        - (2.12) Pilot implementation resources
- **(3) Human resources** consist in *specialists* with certain competencies and experience in the domain. Human resources can de classified by profile and specialties.
    - By *profile*:
        - (3.1) Project manager
        - (3.2) Analyst
        - (3.3) Programmer
        - (3.4) Designer
        - (3.5) Developer

        - (3.6) Tester
        - (3.7) Responsible with documentation
        - (3.8) Responsible with quality assurance
    - By *specialty*:
        - (3.9) Specialist in languages
        - (3.10) Specialist in systems
        - (3.11) Specialist in quality assurance
- From the three categories of resources, usually the costs related to *human resources* are the most substantial.

5.3 Costs Elements of a Software Project

- **Cost elements** of a software project can be classified in the following groups:
    - (1) **Hardware** and **software resources** costs.
    - (2) **Personal mobility** and **training** costs.
    - (3) **Effort** costs *(usually the most significant)*. They consist in:
        - (3.1) Costs with **salaries** of the engineers and implied personal achieving different activities.
            - The activities are measured as *working time* multiplied by the **cost** corresponding to the qualification level of the person in charge with.
            - Some *costs elements* included in this category:
                - (3.1.1) Problem study
                - (3.1.2) Documentation
                - (3.1.3) Personal training
                - (3.1.4) Analyze
                - (3.1.5) Inception
                - (3.1.6) Design
                - (3.1.7) Coding
                - (3.1.8) Test
                - (3.1.9) Documentation elaboration
                - (3.1.10) Implementation
                - (3.1.11) User training
                - (3.1.12) Warranty

            - (3.1.13) Project management
        - (3.2) Costs with **space**, **electricity**, **heat.**
        - (3.3) Costs for **computer networks** and **communications.**
        - (3.4) Costs for **shared facilities** (library, cafeteria).
        - (3.5) Costs for **retirements**, **social assurance**, **taxes**, **health assurance.**
        - (3.6) Costs related to **currency fluctuations.**
        - (3.7) Other *costs*, measured by specific intrinsic characteristics: **redeems** (amortizari), **overheads**, **consumables**
- The costs are corrected with a ***correction factor*** reflecting the project risks.
- Usually the *estimated cost* is not the *effective cost* of the product.
- The **final cost** depends on many other factors as manner of estimation, price policy of the developing organization, etc.

5.4 Software Costs Estimating Techniques

- **Boehm** discusses several different types of models and methods for cost estimation including:
    - **(1) Analogy**
    - **(2) Expert judgment**
    - **(3) Bottoms-up**
    - **(4) Top-down**
    - **(5) Combined Method**
    - **(6) Parkinson**
    - **(7) Price-to-win.**
    - **(8) Parametric models**

- The Figure 5.4.a provides a *brief summary* of these models along with a listing of their advantages and disadvantages.

    - These techniques are discussed in further detail latter.

Model Category	Description	Advantages	Limitations
Analogy	Compare project with past similar projects	Estimates are based on actual experience	Truly similar projects must exist
Expert Judgment	Consult with one or more experts	Little or no historical data is needed; good for new or unique projects	Experts tend to be biased; knowledge level is sometimes questionable
Bottoms-Up	Individuals assess each component and then component estimates are summed to calculate the total estimate	Accurate estimates are possible because of detailed basis of estimate (BOE); promotes individual responsibility	Methods are time-consuming; detailed data may not be available, especially early in a program; integration costs are sometimes disregarded
Top-Down	The project is partitioned into lower-level components and life cycle phases beginning at the highest level. This method is more applicable to early cost estimates when only global properties are known.	The system-level activities are considered. It is usually faster, easier to implement and requires minimal project detail.	It can be less accurate and tends to overlook lower-level components and possible technical problems and provides very little detail for justifying decisions or estimates.
Parametric Models	Perform overall estimate using design parameters and mathematical algorithms	Models are usually fast and easy to use, and useful early in a program; they are also	Models can be inaccurate if not properly calibrated and validated; it is possible that historical data used for calibration may not be

		objective and repeatable	relevant to new programs

Fig. 5.4.a. Categories of Software Cost Models

5.4.1 Analogy Techniques

- **Analogy techniques** are the simplest form of estimating.
- Analogy techniques are used to estimate costs by comparing *proposed programs* with *similar* previously completed programs, for which historical information is available.
    - Actual data from the completed projects are *extrapolated* to estimate the proposed project.
- Estimating by analogy can be done either at the system level or at the component level.
    - *For **example**, if an agency wanted to develop a new payroll system serving 5,000 people and containing 100,000 lines of COBOL code, and a second agency had developed a similar 100,000 line program for $2 million, it could be expected that the first agency's program, ignoring inflation and other similar factors, would also cost $2 million.*
- An *advantage* of the analogy method is that it is based on actual experience.
- However, analogy estimating has *limited use* because in many instances, no truly similar programs exist.
    - *For **example**, the cost of 100,000 lines of Ada code for a bomber-terrain program would not be the same as that of 100,000 lines of COBOL code for payroll software.*
    - Furthermore, for most modern systems, software programs have no true historical precedents.
    - As a result, it is important that differences between **completed projects** and **proposed projects** to be identified and their impacts estimated when analogy techniques are used.
- Analogy techniques are generally **not** *used alone* to estimate software costs
    - They are more often used to check *other estimates* for reasonability (i.e., sanity checks).

Advantages:

- (1) The estimates are based on *actual project data* and *past experience*.

- (2) Differences between completed projects and the proposed project can be identified and impacts estimated.

Disadvantages:

- (1) It is not easy to identify those *differences*.
- (2) This method is also limited because similar projects may not exist, or the accuracy of available historical data is suspect.

5.4.2 Expert Judgment Techniques

- **Expert Judgment Techniques** involves consulting one or more experts to *benchmark off* their *experience* and *understanding* of a **proposed project** to provide an **estimate** for the **cost** of the project.
    - Examples of popular expert judgment techniques are the **Delphi** and **Wideband Delphi** methods.
- Expert judgment techniques are useful in assessing *differences* between *past and future* programs and *new or unique* programs for which **no** *historical precedent* exists.
    - However, an expert's *biases* and possible *lack of knowledge* concerning the differences between past projects and the requirements of the proposed project, often create difficulties.
    - Although Delphi techniques can help alleviate bias problems, experts are usually hard-pressed to accurately estimate the cost of a new software program.
- Therefore, while **expert judgment models** are useful in determining inputs to other models, they are **not** frequently used alone as a **BOE** (Base of Estimation), particularly for software estimates that will be submitted to the Government.
- Expert judgment involves *consulting* with human experts to use their experience and understanding of a proposed project.

Advantages:

- **(1)** The expert can *factor in* **differences** between past project experiences and requirements of the proposed project.
- **(2)** The expert can also *factor in* **project impacts** caused by new technologies, applications, and languages.
- **(3)** Expert judgment always compliments other estimation methodologies.

Disadvantage:

- **(1)** It is hard to document the factors used by the expert.

5.4.3 Bottoms-Up Techniques

- **Bottoms-up estimating** is generally a *very detailed* and *time-consuming process* for estimating costs.
  - (1) It may be used to estimate software costs by performing *detailed analyses* of costs specific to each unit (i.e., SU (Software Units)). (Fig. 5.4.3.a.)

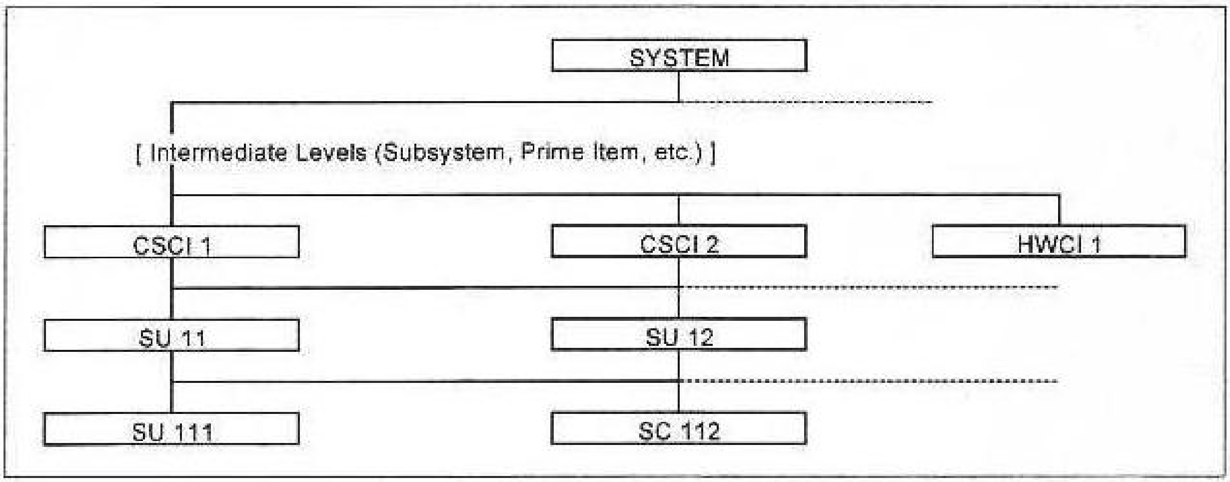

Fig. 5.4.3.a. ISO/EIA 12207 Software Hierarchy

  - (2) Unit costs are totaled *to determine the cost* (or effort) for each CSCI (Computer Software Configuration Items);
  - (3) Then is possible to determine the software cost for the **overall system**.
- **Bottoms-up estimating** is often used (1) during *proposal preparation* and subsequently, (2) for *tracking* software project costs.
- **Advantages** of this method:
  - **(1)** It provides a *detailed basis* for cost estimating.
  - **(2)** Can be more accurate than other methods.
  - **(3)** Since personnel responsible for developing a specific unit assess the associated costs, this type of estimating can promote *individual responsibility* for controlling costs.
  - **(4)** Bottoms-up estimating can also be useful for *cost tracking,* since separate estimates are usually established for the activities that will be performed during each software development phase.
- Bottoms-up estimating also has several **disadvantages**.
  - **(1)** Since detailed information is required, it tends to be time and cost intensive.

- **(2)** In addition, *historical data* are not always available to support these estimates, and there is a tendency to rely extensively on *judgment*.
- **(3)** Bottoms-up techniques may **not** capture costs associated with software integration activities, which can be significant cost drivers.

Advantages:

- **(1)** Bottom-up technique provides a *detailed basis* for cost estimating.
- **(2)** Can be *more accurate* than other methods.
- **(3)** Can promote *individual responsibility* for controlling costs because personnel responsible for developing a specific unit assess the associated costs.

Disadvantages:

- **(1)** Detailed information is required, as result tends to be *time and cost intensive*.
- **(2)** *Historical data* are not always available to support these estimates.
- **(3)** Does not reveal costs associated with *software integration activities*.

5.4.4 Top-Down Techniques

- The **top-down method** of estimation is based on *overall characteristics* of the software project.
    - The project is *partitioned* into *lower-level components* and *life cycle phases* beginning at the highest level.
- This method is more applicable to ***early*** *cost estimates* when only global properties are known.

Advantages:

- **(1)** The *system-level activities* (integration, documentation, project control, configuration management, etc.) are considered.
- **(2)** It is usually *faster* and *easier* to implement.
- **(3)** It requires *minimal* project detail.

Disadvantages:

- **(1)** It can be *less* accurate.
- **(2)** It tends to *overlook* lower-level *components* and possible *technical problems*.
- **(3)** It provides very *little detail* for justifying decisions or estimates.

5.4.5 Combined Method

- The **Top-Down** method starts with a *global estimation* which is detailed through a *stepwise refinement process*
- The **Bottom-Up** method estimates software costs by performing *detailed analyses* of costs specific to each unit (i.e., SU (Software Units)).
    - **Unit costs** are totaled to determine the cost (or effort) for each CSCI (Computer Software Configuration Items).
    - **CSCI costs** are totaled to determine the software cost for the subsystems**.**
    - **Subsystems costs** are totaled to determine finally possibly the cost of the **overall system**.
- The two methods can be **combined** in the following manner which benefits of the advantages of the both methods:
    - **(1)** The project is *partitioned* into *lower-level components* in a top-down manner as far as reasonable.
    - **(2)** The current *smallest unit* are estimated.
    - **(3)** The *costs* are determined in a bottom-up manner starting from the current smallest units.
    - **(4)** The *total sums* are adjusted by re-distributing in a *top-down* manner the *costs* of the elements.
    - **(5)** The steps (2),(3) and (4) are *repeated* until the desired level of accuracy is reached.

5.4.6 Parkinson's Law

- Parkinson's Law says: *"The project costs whatever resources are available"*.
    - That means the development of a project can be extended in time, until disposable resources exist.
- In other words the same project can be finalized in *3 man*months* or in *6 man*months*.
    - The *explanation* is that a project which is finalized in a short time:
        - **(1)** It will be *less elaborated*.
        - **(2)** It will include *less functionalities*.
        - **(3)** It will be *less tested.*
    - While a project finalized in a longer time *won't have* these problems.

- In fact, a complex software project is never finished.
    - It can be any time improved and optimized, the performances increased and new functionalities added.
    - That is the reason the software products has usually several *versions* and are in a continuous *evolution*.

Advantage:

- **(1)** The method doesn't produce budget overflows.

Disadvantage:

- **(1)** The obtained products are usually unfinished.

5.4.7 Pricing to Win

- The project costs as much as the customer *can afford to spend* on it.
    - In this case, the project estimation must take into account the *customer's payment options*.
- Starting with an imposed price, the estimation is done in such a way that the *available sum* to be not overflowed, of course if this is possible.
    - In fact the customer receives as much as he *can afford to pay*.

Advantage:

- **(1)** After evaluation, the contract is easy to sign.

Disadvantages:

- **(1)** The probability that the customer to get what he wants is **very low**.
- **(2)** On the other side, the cost doesn't reflect with accuracy the necessary effort.

5.5 Software Costs Evaluation Models

- Software Cost Evaluation models and techniques are partitioned in two big categories:
    - **(1) Decomposition models.**
    - **(2) Parametric models.**

5.5.1 Decomposition Models

- Decomposition models *partition* the project in *small tasks* (elements) which are separately evaluated.
- In this category are included evaluation models as:
    - (1) **Effort Estimation** Model (**EE** Model).

- (2) **Line of Code** Model (**LOC** Model).
- (3) **Functional Point** Model (**FP** Model).

5.5.1.1 **Effort Estimation** Model (**EE Model**)

- **EE Model** (Effort Estimation Model) is a *decomposition model*.
- EE Model construction consists in the following steps:
  - **(1)** The project is decomposed in *functional tasks* (features).
    - The functional tasks are those corresponding to project features.
  - **(2)** Each functional task is then decomposed in *subtasks* corresponding to project development life cycle *phases*.
    - The subtasks corresponding to life cycle phases are: *analyze, preliminary design, detailed design, codification, test.* (Fig. 5.5.1.1.a)
  - **(3)** The costs for each *subtask* of each *feature* are estimated (usually in **hours**).
  - **(4)** Then are determined *features' costs, phases' costs* and *total cost*.

Nr. Crt	**Task Feature**	**Analyze**	**Preliminary Design**	**DetailedDesign**	**Codification**	**Test**	**Total**
1.	Feature 1						
2.	Feature 2						
3	Feature 3						
4.	Feature 4						

	…						
	…						
	Total						
	Unitary Cost						
	Total Cost						

Fig. 5.5.1.1.a. EE Evaluation Model

Advantages:

- **(1)** The *phase costs* of *each feature* can be clearly emphasized.
  - This thing is very important because in many situations the costs differ from a phase to another.
  - For **example**, usually, *cost_analyze>cost_designer>cost_coder>cost_test>cost_delivery* (in money man*day)
- **(2)** The evaluation method is *simple*, very *efficient* and achieves to results which are affected by a *reduced* error coefficient.

Disadvantages:

- **(1)** The method requires evaluators with *significant experience* and a high degree of *professionalism*.

5.5.1.2 LOC (Line of Code) and FP (Functional Point) Models

- LOC and FP are also *decomposition models*.
  - At general level structure they are similar.
- The *difference* appears in ***estimation manner***:
  - **LOC** (Lines of Code) for **LOC Model**
  - **FP** (Functional Points) for **FP Model**
- *For **example**,* **LOC Model** requires the following steps:
  - **(1)** The project is decomposed in *functional tasks* (features).

- **(2)** The functional tasks are those corresponding to *project features*.
- **(3)** Each *feature* is estimated in LOCs.
- **(4)** Knowing the *unitary costs* (Cost/LOC), the cost for *each feature* is determined.
- **(5)** Knowing the productivity (LOC/day) the duration for *each feature* is determined.
- **(6)** By summing, the *total cost* and *total duration* can be determined.
- **(7)** The total duration can be reduced if more people will work in *parallel*.(Fig.5.5.1.2.a.)

Nr.crt	**Feature**	**LOC**	**Cost/LOC**	**LOC/day**	**Cost (col 1)*(col 2)**	**Duration (col 1)/(col 3)**
	0	1	2	3	5	6
1.	Feature 1					
2	Feature 2					
3.	Feature 3					
…						
…						
	Total					

Fig.5.5.1.2.a. LOC Model for Cost Evaluation

- LOC Model utilization must take into consideration that *software productivity* (Cost/LOC) depends on the used programming language.
  - The software productivity is higher for high level languages and lower for assembling languages
- LOC Model is difficult to apply if *visual developing tools* are used.
  - In these cases the FP Model is more suitable.

5.5.2 Parametric Models

- **Parametric models** generate estimates based on *statistical relationships*.
- **Parametric models** relate **dependent** variables (i.e., cost and/or schedule) to one or more **independent** variables (i.e., parameters).
- Parametric estimating techniques for software projects generally estimate *overall system* or CSCI (Computer Software Configuration Items) costs based on a software program's design characteristics.
- These overall costs can be partitioned among:
  - The *lower-level SUs* (Software Units – lowest level contain between 100 and 200 LOCs).
  - The *life cycle phases*.
  - The advantages of parametric models are:
    - **(1)** Are *fast* and *easy to use*.
    - **(2)** Require *little detailed information* (after they are calibrated).
    - **(3)** Capture *total system* or CSCI-*level costs* (including costs for *integration activities*).
    - **(4)** Can be as *accurate* (if not more accurate) as other estimating techniques, if they are properly *calibrated* and *validated*.
  - Because of these advantages, parametric models are generally ***DOD's** software estimating technique* of choice.
  - There are several commercial parametric models that are widely used by both industry and government.
  - There are also many sophisticated parametric software estimating models that use *multiple parameters* to compute software costs and effort.
  - In this section will be discussed three common *software parametric models* and will be provided a basic understanding of some of their common attributes.
  - The three models are:

- **Constructive Cost Model** (COCOMO),
- **PRICE Software Model (PRICE S®),**
- Galorath **Software Evaluation and Estimation of Resources - Software Estimating Model** (SEER-SEM®).

- For each of these models the following *information* is discussed:
  - (1) Background.
  - (2) Principal inputs (i.e., parameters).
  - (3) Processing.
  - (4) Principal outputs.
  - (5) Cost estimating capabilities for software support. These are addressed because of the growing importance of this area.

5.5.2.1 COCOMO 81 Model

- Dr. Barry Boehm, formerly of the TRW Corporation, developed the **COCOMO (COnstructive COst MOdel)** *parametric software model* and published its first edition in 1981.
- COCOMO is **not** a **proprietary model.**
  - It is completely described in Dr. Boehm's 1981 book, "Software Engineering Economics".
- In actuality, only a *pocket calculator* with an exponential key is required to execute COCOMO.
- Many computerized versions of COCOMO are available in the marketplace.
  - One model, the Revised Enhanced Version of Intermediate COCOMO **(REVIC),** developed by Ray Kile, was the Air Force's *"standard edition"* of COCOMO in the late 1980s and early 1990s.
- During the last several years, Dr. Boehm made substantial revisions to COCOMO.
- COCOMO81 was upgraded to **COCOMO II**.

COCOMO 81 General Overview

- **COCOMO 81** (i.e., the first version of COCOMO) is a *regression-based model* that considers *programs* in three ***modes*** (Fig. 5.5.2.1.a.)
  - *(1) Organic*
  - *(2) Semi-detached*
  - *(3) Embedded*

Modelul COCOMO - Tipuri de proiecte

Tip de proiect	Caracteristici			
	Dimensiune	Noutate	Constrângeri de timp	Mediu de lucru
Organic	Foarte mic	Mic	Nu foarte strânse	Stabil
Embedded	Mare	Mare	Foarte strânse	Interfețe utilizator sau hardware complexe
Semidetached	Mediu	Mediu	Mediu	Mediu

Modelul COCOMO de bază - Constante de calcul

Tip de proiect	a	b	c	d
Organic	2.4	1.05	2.5	0.38
Embedded	3.0	1.12	2.5	0.35
Semidetached	3.6	1.20	2.5	0.32

COCOMO de baza: Ecuații fundamentale

Effort = a *dimensiune^b
Durata = c* Effort^d
Numar persoane = effort/Durata

Fig. 5.5.2.1.a. COCOMO Project modes

- Separate equations relating *level of effort* (**LOE**) in *man-months* (**MM**) to *program size* in *thousands of delivered source instructions* (**KDSI**) are established for each mode.
- There are **three** levels within COCOMO 81:
  - **(1) Basic**
  - **(2) Nominal Intermediate**
  - **(3) Detailed**
- **(1)** The **Basic** level consists of *three simple **models*** (one for each **mode**). Each model consists in two single parameter equations: one for **effort** and one for **schedule**.
  - ***Effort** equations* relate **MM** (Man*Month) to **KDSI**
  - The ***schedule** equations* relate months of time needed (**M**) to **MM,** computed from the effort equations.

- The **Basic level** of COCOMO is often used to develop *rough order of magnitude* ***(ROM)*** estimates for software costs.
- Figures 5.5.2.1.a and 5.5.2.1.b. (the first two columns) lists ***Effort*** and ***Schedule*** *equations* for the three modes of **Basic** COCOMO 81.

Level / Mode	**Basic** *Effort*	**Basic** and **Nominal Intermediate** ***Schedule***	**Nominal Intermediate** *Effort*
Organic	$MM = 2.4\ (KDSI)^{1.05}$	$M = 2.5\ (MM)^{0.38}$	$MM = 3.2\ (KDSI)^{1.05}$
Semi-Detached	$MM = 3.0\ (KDSI)^{1.12}$	$M = 2.5\ (MM)^{0.35}$	$MM = 3.0\ (KDSI)^{1.12}$
Embedded	$MM = 3.6\ (KDSI)^{1.20}$	$M = 2.5\ (MM)^{0.32}$	$MM = 2.8\ (KDSI)^{1.20}$

Fig. 5.5.2.1.b. Elementary COCOMO 81 Equations

- **(2)** The **Nominal Intermediate** COCOMO 81
  - Contains also *nominal equations* (fig. 5.5.2.1.b. - columns 2-3)
    - The **schedule** equations are similar to the *basic equations* for each mode.
    - The **effort** coefficients are specific.
  - In **addition**, the Intermediate model introduces ***15 multipliers.***
    - These **multipliers** adjust the result of the *nominal intermediate effort equation* to reflect a *program's unique attributes* (fig. 5.5.2.1.c.).
    - Each **multiplier** has **predetermined values** related from very low, to extremely high presented in the table 5.5.2.1.b.
    - The **nominal** value for each attribute is 1.00 (no influence).

		Rating					
Nr.	**Attributes**	**VL**	**LO**	**NM**	**HI**	**VH**	**XH**
1	Required Reliability (RELY)	0.75	0.88	1.00	1.15	1.40	
2	Database Size (DATA)		0.94	1.00	1.08	1.16	
3	Product Complexity (CPLX)	0.70	0.85	1.00	1.15	1.30	1.65
4	Execution Time Constraints (TIME)			1.00	1.11	1.30	1.66
5	Main Storage Constraint (STOR)			1.00	1.06	1.21	1.56
6	Virtual Machine Volatility (VIRT)		0.87	1.00	1.15	1.30	
7	Computer Turnaround Time (TURN)		0.87	1.00	1.07	1.15	
8	Analyst Capability (ACAP)	1.46	1.19	1.00	0.86	0.71	
9	Applications Experience (AEXP)	1.29	1.13	1.00	0.91	0.82	
10	Programmer Capability (PCAP)	1.42	1.17	1.00	0.86	0.70	
11	Virtual Machine Experience (VEXP)	1.21	1.10	1.00	0.90		

12	Programming Language Experience (LEXP)	1.14	1.07	1.00	0.95		
13	Use of Modern Programming Practices (MODP)	1.24	1.10	1.00	0.91	0.82	
14	Use of Software Tools (TOOL)	1.24	1.10	1.00	0.91	0.83	
15	Required Development Schedule (SCED)	1.23	1.08	1.00	1.04	1.10	

Fig. 5.5.2.1.c. Intermediate COCOMO 81 Software Development Effort Multipliers

- **(3)** The **Detailed** COCOMO 81
  - Adjusts the ***multipliers*** for *each phase* of the *software life cycle*

For ***example*** **CSCI phases** in **Software Waterfall Model** (Fig.5.5.2.1.d.).

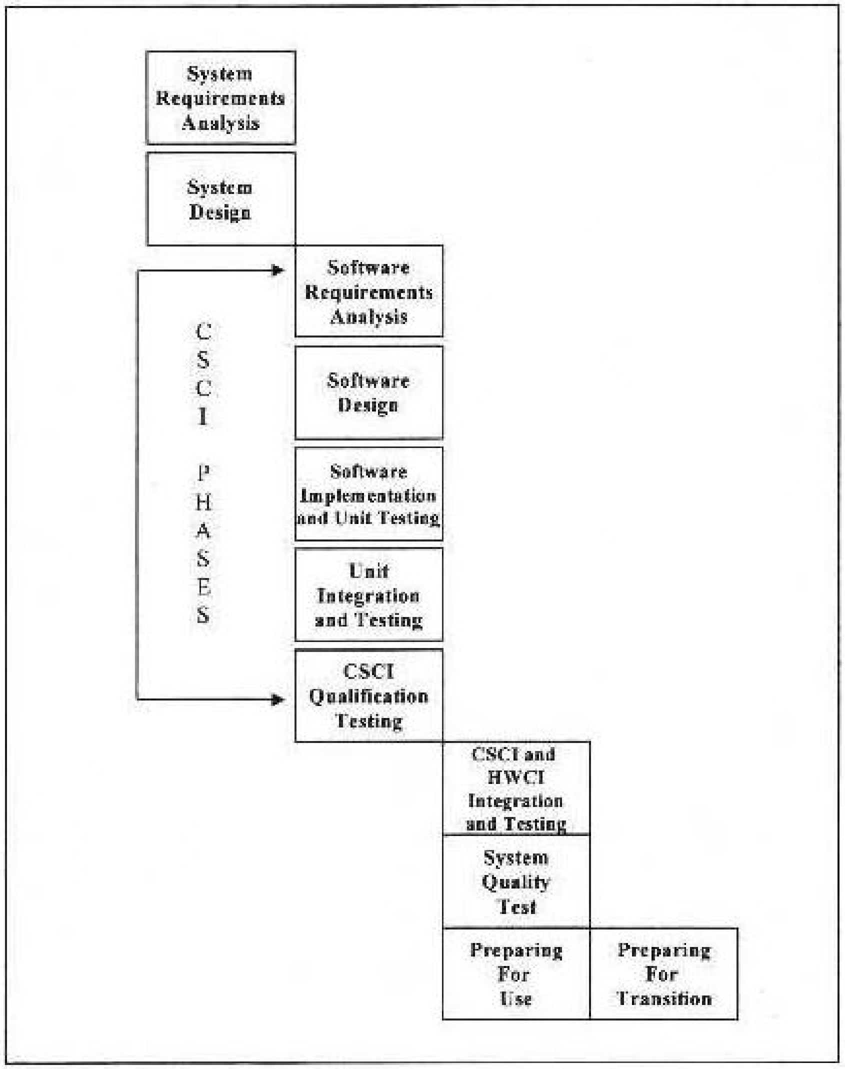

Fig. 5.5.2.1.d. IEEE/EIA 12207 Software Waterfall Model Development Phases

- **Remarks:**
  - All COCOMO levels *are designed* for use on any program, although the model is probably *best* used for **CSCI**-level estimates.
  - All levels consider the CSCI *phases.*
    - e.g., software development cycle from design through CSCI testing, although the design phase is partitioned into the "*product design*" and "*detailed design*" phases.
  - All three COCOMO 81 levels allow an *adjustment* to KDSI for reused or modified software.

- COCOMO 81 computes “effective KDSI” based on the percentages of *redesign*, *recoding*, and *retesting* required.

COCOMO 81 Inputs

- The following discussion focuses on the **Intermediate level** of COCOMO 81.
- The *primary* **Intermediate COCOMO** *input* (i.e., cost driver) is *program size*, in **KDSI**.
- However, there are *15 additional attributes* that must be assessed.
- These **attributes**, shown in figure 5.5.2.1.c, are classified into the following **four categories:**
- ***(1) Product attributes:*** describe the *environment* the program operates in. The three attributes in this category are:
  - (1) Reliability Requirements (RELY),
  - (2) Database Size (DATA),
  - (3) Product Complexity (CPLX).
- ***(2) Computer attributes:*** describe the relationship between a program and its *host or developmental computer*. The four attributes in this category are:
  - (4) Execution Time Constraints (TIME),
  - (5) Main Storage Constraints (STOR),
  - (6) Virtual Machine Volatility (VIRT),
  - (7) Computer Turnaround Time (TURN).
- ***(3) Personnel attributes:*** describe the *capability* and *experience of personnel* assigned to the program. The five attributes in this category include:
  - (8) Analyst Capability (ACAP),
  - (9) Applications Experience (AEXP),
  - (10) Programmer Capability (PCAP),
  - (11) Virtual Machine Experience (VEXP).
  - (12) Programming Language Experience (LEXP),
- ***(4) Project attributes:*** describe selected *project management facets* of a program. The three attributes in this category include:
  - (13) Use of Modern Programming Practices (MODP),
  - (14) Use of Software Tools (TOOL),
  - (15) Required Development Schedule (SCED).

- Tables developed by Dr. Boehm describe ratings from "*very low*" to "*extremely high*," that must be assessed for each of the 15 attributes identified above.
  - For ***example***, the reliability factor (i.e., RELY) would be rated:
    - "*very low*" if an error in a specific software system caused only slight inconvenience;
    - "*nominal*" if an error could result in moderate, recoverable losses;
    - "*very high*" if potential loss to human life is at stake.
- Figure 5.5.2.1.c, extracted from Dr. Boehm's book, shows the numerical values assigned to specific ratings of **Very Low** (VL), **Low** (LO), **Nominal** (NM), **High** (HI), **Very High** (VH), and **Extra High** (XH) for each of the 15 attributes.
  - Note that all "*nominal*" attributes would have **no** effect on the effort required.
- Dr. Boehm's book provides detailed guidance on these attributes.

COCOMO 81 Processing

- Using the **Intermediate** COCOMO 81, the following steps are performed:
- (1) A **nominal assessment** of **MM** based on *size* alone, is made for the program being estimated.
  - The *nominal equations* shown in fig. 5.5.2.1.b. (columns 2-3) are used for this purpose.
- (2) Assess **ratings** for all the *15 effort multipliers* (fig. 5.5.2.1.c).
- (3) The **ratings** assessed for all 15 attributes are multiplied to obtain an *overall product for the attributes* (hereafter referred to as "*P*").
- (4) The **MM** figure from the nominal equation is multiplied by *P* to compute the **required MM** of effort for the program in question.
  - An ***example***:
    - Suppose an **embedded program** was 20 KDSI in size.
    - The nominal embedded equation would be: $MM=2.8(20)^{1.20}$, which would yield 102 MM.
    - Next, suppose all attributes are assessed at the "nominal" level, except reliability (i.e., RELY), which is rated "*very high*" and assigned a value of 1.40; *P* is now 1.40.
    - The level of **effort** computed by the Intermediate COCOMO would be (102 x 1.4) = 143 man-months.

        - The **schedule** (also based on the equations shown in fig.2.6-5) would be $M=2.5(143)^{0.32}$ = 12.2 months.
- From a *review* of the model, the basic concept behind the Intermediate COCOMO 81 is rather straightforward.
- The **primary challenges** in the use of Intermediate COCOMO 81 relates:
    - (1) To properly estimating software size.
    - (2) Assigning proper ratings to the 15 attributes.
- COCOMO 81 also provides facilities for **reuse of existing software** by adapting the size (KDSI) of the software to be estimated.
    - The equation for this size revision is presented in fig. 5.5.2.1.e.

COCOMO 81: The equation for this size revision

EDSI = ADSI (AAF/100),

where

- EDSI = equivalent number of **source instructions**.
- ADSI = # of instructions to be **adapted** (modified or reused) from existing software for use in the new software.
- AAF = Adaptation Adjustment Factor.
    - It is computed as **AAF = 0.40 (DM) + 0.30 (CM) +0.30 (IM)**
    - It is possible for DM, CM, or IM to exceed 100 percent, indicating effort to reuse software might be greater than that needed to develop a new program.
        - **DM** = percentage of the adapted software design that will be modified.
        - **CM** = percentage of adapted software code modified.
        - **IM** = percentage of effort needed to integrate the adapted software into the overall product and test the resulting product.

Fig. 5.5.2.1.e. Intermediate COCOMO 81. Size revision equation

COCOMO 81 Outputs

- The output of the Intermediate COCOMO model is simply:
    - (1) The **LOE** in **MM**s for the project being estimated.
    - (2) A **schedule** in **months**.

- The effort output can easily be converted to a ***monetary value*** if the cost per MM is known.
- It is also possible to determine the allocation of the overall effort to various ***phases*** of the CSCI life cycle, or time periods, using information presented in Dr. Boehm's book.

COCOMO 81 Support Cost Considerations

- The COCOMO 81 Maintenance model (**COCOMO-M**) can be used to estimate **annual MMs** required to support a software program.
- For the **Intermediate level** of COCOMO-M, *all* COCOMO *inputs* are used, except different numerical values are assigned to two attributes:
  - Reliability (RELY).
  - Use of modern design practices (MODP).
- A product of **maintenance multipliers** (***PM***) is computed, similar to *P* for development effort equations.
- An assessment of **annual change traffic** (**ACT**), reflecting the fraction of code to be changed per year, is also required.
- **Annual man-months** (**MMA**) are computed from the equation shown below.
- **Annual support costs** can be computed by multiplying the number of MMs by the cost per MM (Fig. 5.5.2.1.f.)

COCOMO 81: Annual support costs

MMA = (MMNOM) (ACT) (PM), where

where

- MMNOM = nominal intermediate MMs.
- ACT = Annual Change Traffic.
- PM = Person-Months.

Fig. 5.5.2.1.f. Intermediate COCOMO 81. Annual support costs

summary

5.5.2.2 COCOMO II Model

- **COCOMO II** was developed during the mid 1990s by a consortium of organizations.
  - The author was Dr. Barry Boehm and several Graduate students from the University of Southern California (USC).
  - The *first version* of the COCOMO II model was released in late 1996.
- The purpose of COCOMO II was "*to develop a software cost and schedule estimation model tuned to the life cycle practices of the 1990s and 2000s*."
  - While COCOMO 81 was largely based on projects developed using the *waterfall methodology*, COCOMO II is more compatible with new design methodologies (e.g., *object-oriented techniques*).
  - The updated model also includes a *new database* consisting of data submitted by consortium members.
- Similar to COCOMO 81, the model is *non-proprietary*, and *computerized editions* are available.
- **COCOMO II** contains *three stages* of estimation.
  - **(1) Stage 1** called **Applications Composition**
    - Supports prototyping or application composition efforts.
    - Uses *object points* as its size measure.
  - **(2) Stage 2** called **Early Design**
    - Supports estimation during the program's early design phases.
    - Uses *function points* or *thousands of source lines of code* (KSLOC), as the size measure.
    - In addition, Stage 2 uses *7 attributes*, which are based on combinations of the 17 attributes used in Stage 3.
  - **(3) Stage 3** called **Post Architecture**
    - Supports estimation after early design.
    - Stage 3 uses *17 attributes* as effort multipliers.
    - Stage 3 is actually a *modification* to COCOMO 81, so the descriptions discussed in the following paragraphs represent the *differences* between COCOMO 81 and COCOMO II.
  - These stages are described in further detail, in reverse order.

General Description of COCOMO II (Stage 3 - Post Architecture)

- COCOMO II equations are *revisions* of the COCOMO 81 equations.

- These revisions contain ***variable exponents*** and treat *software reuse* differently.
- The Effort Equation for Stage 3 is presented in fig. 5.5.2.2.a.

COCOMO II: The Effort Equation for Stage 3

$PM = [A\ (Size')^{B}\ (EM)] + PM_{AT}$, where

PM is *person-months* of estimated effort.

A is a coefficient that is provisionally set to 2.5 (i.e., default value), but should be calibrated to reflect the specific organization's costs and culture.

Size' = Size (1 + BRAK/100) where BRAK is breakage, or the percentage of *code thrown away* due to requirements volatility. Size itself is the sum of *new* and *adapted KSLOC*.

B is a variable exponent with the equation: **B = 0.91 + 0.01 (SF),**

- **SF** is the **sum** of five **scale factors** (fig. 5.5.2.2.b).
- The **five scale factors**, listed in figure 5.5.2.2.b are:
  - Precedentedness (PREC);
  - Development flexibility (FLEX);
  - Degree of risk resolution (RESL);
  - Degree of team cohesion (TEAM);
  - Process maturity (PMAT).
- Like the effort multipliers in COCOMO 81 and COCOMO II, they are rated from **VL** to **XH** (very low to extra high).
  - These values are shown in figure 5.5.2.2.b. with more favorable ratings resulting in lower values and therefore, a lower effort exponent.
  - For PREC for ***example***, VL corresponds to "thoroughly unprecedented," while XH corresponds to "thoroughly familiar." Figure 5.5.2.2.b shows the numerical value for VL (4.05) is higher than the value for XH (0.00).

EM is the product of **17 effort multipliers** shown later in fig. 5.5.2.2.e. These are similar to those used in Intermediate COCOMO 81.
PM_{AT} is the effort for components automatically translated.

Fig. 5.5.2.2.a. COCOMO II. Stage 3. Effort equation

		Rating					
Nr	**Scale Factors**	VL	LO	NM	HI	VH	XH
1	Precedentedness (PREC)	4.05	3.24	2.43	1.62	0.81	0.00
2	Development Flexibility (FLEX)	6.07	4.86	3.64	2.43	1.21	0.00
3	Architecture / Risk Resolution (RESL)	4.22	3.38	2.53	1.69	0.84	0.00
4	Team Cohesion (TEAM)	4.94	3.95	2.97	1.98	0.99	0.00
5	Process Maturity (PMAT)	4.54	3.64	2.73	1.82	0.91	0.00

Fig. 5.5.2.2.b. COCOMO II. Stage 3. Scale factors

- The Schedule Equation for COCOMO II has a ***variable exponent***, and is calculated using the following equation (Fig. 5.5.2.2.c).

COCOMO II: The Schedule Equation for Stage 3

M = [3.0 x PM $^{(0.33 + 0.2 (B-0.91)}$] (SCED%/100), where

PM = *person-months* of estimated effort.
SCED% = *the percentage of schedule compression or expansion*. This is the same parameter used in COCOMO 81, although the numerical ratings differ.

Fig. 5.5.2.2.c. COCOMO II. Stage 3. Schedule equation

- As referenced, COCOMO II treats *reused code* differently.
- The new **reuse equation** takes into account that a certain amount of effort is needed for reuse no matter how much code is actually being modified.
- For this purpose an **Adaptation Adjustment Multiplier** factor is used.
- The **Adaptation Adjustment Multiplier** (**AAM**) is non-linear and is computed by the equation presented in fig. 5.5.2.2.d.

COCOMO II: The Adaptation Adjustment Multiplier (AAM) (Stage 3)

AAM = (AAF + AA + SU + UNFM)/100, where

AAF = Adaptation Adjustment Factor. Computed as AAF = 0.40 (DM) + 0.30 (CM) + 0.30 (IM). It is possible for DM, CM, or IM to **exceed** 100 percent, indicating effort to reuse software might be greater than that needed to develop a new program.

DM = percentage of the adapted software **design** that will be modified.

CM = percentage of adapted software **code** modified.

IM = percentage of effort needed to **integrate** the adapted software into the overall product and test the resulting product.

AA = the degree of **assessment and assimilation** to determine reuse suitability and to integrate the description of the reused software into the product description.

SU = amount of **software understanding** needed.

UNFM = degree of programmer unfamiliarity with the software.

Fig. 5.5.2.2.d. COCOMO II. Stage 3. Adaptation Adjustment Multiplier

COCOMO II Inputs (Stage 3)

- For Stage 3, as in COCOMO 81, *size* in KSLOC is the primary input.

- In addition, there are **17 effort multipliers**. Most are similar to those used in Intermediate COCOMO 81, with the exception of **new multipliers** (fig. 5.5.2.2.e.)
    - Reusability Requirements (**RUSE**)
    - Documentation Requirements (**DOCU**)
    - Personnel Continuity (**PCON**)
    - Multiple Site Development (**SITE**)
- Platform Volatility (**PVOL**) replaces VIRT (Virtual Machine Volatility) in COCOMO 81
- Platform Experience (**PEXP**) and Language and Tool Experience (**LTEX**) replace LEXP (Programming Language Experience) and VEXP (Virtual Machine Experience).
- COCOMO II does not use two of the original COCOMO 81 multipliers: MDOP (Use of Modern Programming Practices) and TURN (Computer Turnaround Time).
- Figure 5.5.2.2.e shows the 17 attributes for Stage 3 along with their **ratings** and **numerical values**.

		Rating					
Nr	Attributes	VL	LO	NM	HI	VH	XH
1	Required Reliability (RELY)	0.75	0.88	1.00	1.15	1.39	
2	Database Size (DATA)		0.93	1.00	1.09	1.19	
3	Product Complexity (CPLX)	0.75	0.88	1.00	1.15	1.30	1.66
4	Required Reusability (RUSE)		0.91	1.00	1.14	1.29	1.49
5	Documentation Required (DOCU)	0.89	0.95	1.00	1.06	1.13	

6	Execution Time Constraints (TIME)			1.00	1.11	1.31	1.67
7	Main Storage Constraint (STOR)			1.00	1.06	1.21	1.57
8	Platform Volatility (PVOL)		0.87	1.00	1.15	1.30	
9	Analyst Capability (ACAP)	1.50	1.22	1.00	0.83	0.67	
10	Applications Experience (AEXP)	1.22	1.10	1.00	0.89	0.81	
11	Programmer Capability (PCAP)	1.37	1.16	1.00	0.87	0.74	
12	Personnel Continuity (PCON)	1.24	1.10	1.00	0.92	0.84	
13	Platform Experience (PEXP)	1.25	1.12	1.00	0.88	0.81	
14	Language and Tools Experience (LTEX)	1.22	1.10	1.00	0.91	0.84	
15	Use of Software Tools (TOOL)	1.24	1.12	1.00	0.86	0.72	
16	Multiple Site Development (SITE)	1.25	1.10	1.00	0.92	0.84	0.78
17	Required Development Schedule (SCED)	1.29	1.10	1.00	1.00	1.00	

Fig. 5.5.2.2.e. COCOMO II. Stage 3. Software Development Effort Multipliers

COCOMO II Inputs (Stage 2- Early Design)

- For Stage 2, size in *function points* is the primary input
  - Although the model converts *function points* to *KSLOC* using a *language table* as described in the USC COCOMO II Model Definition Manual.
- The equation for **Effort** and **Schedule** are quite similar to those belonging to the Stage 3 (fig.5.5.2.2.a, 5.5.2.2.c)
- As a difference, the **EM** (Effort Multiplier) factor includes **7 attributes** that are either replicas or combinations of certain **17 multipliers** of Stage 3.
- The **seven attributes** include the following:
  - *(1) Product Reliability and Complexity (***RCPX***)*: The RCPX attribute is a combination of RELY, CPLX, DATA, and DOCU from Stage 3.
  - *(2) Required Reuse (***RUSE***)*: The RUSE attribute is the same as in Stage 3.
  - *(3) Platform Difficulty (***PDIF***)*: The PDIF attribute is a combination of TIME, STOR, and PVOL from Stage 3.
  - *(4) Personnel Capability (***PERS***)*: The PERS attribute is a combination of ACAP, PCAP, and PCON from Stage 3.
  - *(5) Personnel Experience (***PREX***)*: The PREX attribute is a combination of AEXP, PEXP, and LTEX from Stage 3.
  - *(6) Facilities (***FCIL***)*: The FCIL attribute is a combination of TOOL and SITE from Stage 3.
  - *(7) Required Development Schedule (***SCED***)*: The SCED attribute is the same as in Stage 3.

COCOMO II Inputs (Stage 1- Application Composition)

- Stage 1 inputs include *new object points* (**NOP**) and a *productivity rating* (**PROD**).
- **PROD** is a measure of the *developer's capability* and *experience* with CASE tools
  - **PROD** assumes a value between 4 and 50 with 4 being the highest levels of experience and 50 reflecting the lowest levels of experience.

COCOMO II Processing

- Equations for Stages 2 and 3 were discussed previously.
- As in COCOMO 81, the primary challenges relate to determining *size* and in assessing appropriate *ratings for the attributes*.
- For Stage 1, the model uses a simple equation (fig. 5.5.2.2.f.).

COCOMO II: (Stage 1)

PM = NOP / PROD, where

NOP = New objects point.
PROD = Productivity Rating. Determining both the NOP and PROD can be challenging.

Fig. 5.5.2.2.f. COCOMO II. Stage 1. Effort Equation

- At this moment **only** Stage 3 has been *automated* in the USC COCOMO II computerized model.

COCOMO II Outputs

- The **output** for Stages 2 and 3 of COCOMO II is simply:
  - (1) The **LOE** (Level of Effort) in **person-months** for the project being estimated and a **schedule** in **months**.
  - (2) A **schedule** in **months.**
  - (3) The **effort** output can be easily converted to a *monetary value* if the cost per PM is known.
- For Stage 1, the only output provided is **effort** (i.e., schedule is not provided).

COCOMO II Support Cost Considerations

- There is no support equation for Stage 1.
- Stages 2 and 3 of COCOMO II use a variant of the **reuse algorithms** to compute **maintenance** or **support effort**.
- As of 1998, the support effort equations were not included in the USC computerized edition of COCOMO II.
- The **support effort equation** is presented in fig. 5.5.2.2.g.

COCOMO II: Equation for Support Effort (Stage 2,3):

$PM_m = A(Size_m)^B(EM)$, where

PM_m is maintenance person-months of estimated effort.

A is a coefficient that is provisionally set to 2.5 (i.e., default value), but should be calibrated to reflect the specific organization's costs and culture.

Size = $Size_M$ =(Size Added + Size Modified)(MAF),

- **MAF** represents the **maintenance adjustment factor** computed from the formula MAF = 1 + (UNFM) (SU/100), where
- **UNFM** = degree of **programmer unfamiliarity** with the software
- **SU** = amount of **understanding** needed.

B is a variable exponent with the equation: **B = 0.91 + 0.01 (SF), where**

- **SF** is **the sum** of **5 scale factors** that vary between 0 and 5.
- The 5 scale factors, listed in figure 2.6-7 are: PREC, FLEX, RESL, TEAM, and PMAT.
- Like the effort multipliers in COCOMO 81 and COCOMO II, they are rated from VL to XH (very low to extra high).
- These values are shown in fig.5.5.2.2.b, with more favorable ratings resulting in lower values and therefore, a lower effort exponent.
- For PREC for example, VL corresponds to "thoroughly unprecedented," while XH corresponds to "thoroughly familiar."
- Figure 5.5.2.2.b. shows the numerical value for VL (4.05) is higher than the value for XH (0.00).

EM is the product of 17 effort multipliers as shown in figure 5.5.2.2.e. These are similar to those used in the **Intermediate** COCOMO 81.

Fig. 5.5.2.2.g. COCOMO II. Stage 2,3. Support Effort Equation

summary

5.5.2.3 PRICE S Model

- The PRICE S® commercial model was developed by PRICE Systems, LLC to support software cost estimation.
- PRICE Systems, LLC also developed:
  - (1) A *hardware model*, PRICE H®
  - (2) A *hardware operations and support cost estimating model*, PRICE HL®
  - (3) A *microcircuit and electronic module model*, PRICE M®.
- The PRICE S® model is partly proprietary in that all equations are not published, though most are described in the PRICE S® Reference Manual.
  - This model is applicable to **all** types of software projects.
  - The model considers **all** software life cycle phases.
  - In addition to the software life cycle phases, it also considers system concept and operational testing phases.

PRICE S® Inputs

- The principal inputs for PRICE S® are grouped into the following nine categories:
  - **(1) *Project Magnitude* (SIZE):**
    - Reflects the size of the software to be developed or supported.
    - Size can be input as either*:*
      - SLOC,
      - *Function points*,
      - *Predictive object points*.
  - **(2) *Program Application*** (**APPL**):
    - Provides a measure of the type (or types) of software, described by one of *seven categories*:
      - (1) *Mathematical,*
      - (2) *String Manipulation,*
      - *Data Storage and Retrieval,*
      - *On-Line,*
      - *Real-Time,*
      - *Interactive,*

            - *Operating System.*

    - **(3) *Productivity Factor* (PROFAC)**:
        - PROFAC is a calibration parameter that relates the software program to the *productivity and efficiency of personnel* and *management practices*.
    - **(4) *Design Inventory:***
        - Provides for the amount of software inventory available for use (i.e., reuse).
        - Two parameters indicate the amount of newness for each software type.
            - **New Design** *(***NEWD**)
            - **New Code** (**NEWC**)
    - **(5) *Utilization* (UTIL)**:
        - Reflects the extent of processor loading relative to speed and memory capacity.
        - Values above 50 percent usually increase effort.
    - **(6) *Customer Specifications and Reliability Requirements:***
        - The platform *(***PLTFM**) parameter provides a measure of the level of testing and documentation that will be needed.
    - **(7) *Development Environment:***
        - Three complexity parameters *(***CPLX1**, **CPLX2**, and **CPLXM**) measure unique project conditions such as multiple site development, requirements volatility, use of tools (e.g., CASE tools), and other factors.
    - **(8) *Difficulty ratings for internal* (INTEGI) and *external* (INTEGE) *integration***.
    - **(9) *Development Process:***
        - Reflects the process being used. Choices include waterfall, spiral, evolutionary, and incremental development.
- In addition, there are inputs specific to software support activities. These are discussed in further detail in the following paragraphs.

PRICE S® Processing

- **(1)** PRICE S® computes a **volume** (VOL) **of software** based on the size of the product and APPL (*Program Application*) factor.
- **(2)** It then uses VOL and PROFAC (*Productivity Factor*) to determine an *initial estimate* of **development effort** in **labor hours** (LH).

- As discussed in the PRICE S® Reference Manual, this equation is:

$$LH = [e^{PROFAC}] [VOL^{f(PROFAC)}] / 1000$$

- **(3)** The model then adjusts this *initial estimate* by the PLATFM (Platform) parameter, which has a linear effect on LH obtaing the *core estimate*.
- **(4)** The other inputs are used to make further adjustments to the *core estimate* of LH, and to compute *development schedule*.

PRICE S® Outputs

- (1) PRICE S® computes an ***effort estimate*** in person-months that may be converted to cost in dollars or other currency units.
  - The *effort* is allocated among *three* ***stages*** of software development:
    - (1) Design.
    - (2) Code.
    - (3) Test.
  - The *effort* is also subdivided into *five* ***activities***:
    - (1) Systems Engineering.
    - (2) Programming.
    - (3) Configuration and Quality Control.
    - (4) Documentation.
    - (5) Program Management.
- (2) PRICE S® also computes a ***development schedule*** in months.
  - PRICE S® provides a *schedule effects option* that compares an **input schedule** with **that computed by the model**.
  - This option also shows *penalties for compressing* the user's schedule compared to the model's predicted schedule.
- (3) PRICE S® provides several ***optional outputs*** including:
  - Resources-complexity.
  - Instructions-application sensitivity matrices.
  - Resource expenditure profiles.
- (4) PRICE S® also provides an "*at-a-glance*" output option to rapidly view the effects of changing selected input parameters.
  - This option is useful for performing "*what-if*" type trade studies.

PRICE S® Support Cost Considerations

- The PRICE S® **life cycle model** estimates:
    - *Software maintenance*
    - *Enhancements*
    - *Growth*
    - *Modification costs using acquisition and deployment data*
- The **deployment** inputs, which are support-unique inputs, include the following:
    - Support schedule (start and end dates).
    - Number of installations.
    - Expected growth and enhancement levels.
    - Software quality level.
    - Calibration productivity factors for maintenance, enhancements, and growth.
- The PRICE S® model provides *cost outputs* in three support categories:
    - (1) Maintenance.
    - (2) Enhancements.
    - (3) Growth.
- It also calculates the *number of delivered defects* in the program to be supported.
- The model allocates *effort or cost* among the five activities (e.g., systems engineering) previously described.

summary

5.5.2.4 SEER SEM Model

- SEER-SEM® **(Software Evaluation and Estimation of Resources - Software Estimating Model)** is one of a *family of tools* offered by Galorath Associates.
- The SEER family also includes:
    - (1) A hardware cost estimating model - SEER-H®
    - (2) A hardware-life cycle model - SEER-HLC®
    - (3) A software-sizing model - SEER-SSM®
    - (4) An integrated-circuit model - SEER-IC®
    - (5) A design-for-manufacturability tool - SEER-DFM®.
- SEER-SEM® is *partly proprietary* in that not all equations are published.

- However, some relationships are described in the SEER-SEM® User's Manual.
- SEER-SEM® is applicable to *all program types*, as well as *most phases of the software development life cycle*.

SEER-SEM® Inputs

- SEER-SEM® **inputs** can be divided into three categories:
  - (1) Size
  - (2) Knowledge-Base Inputs
  - (3) Input Parameters
- The **SEER-SEM** inputs are described in further detail below.
  - ***(1) Size:***
    - Size can be input in one of *three formats*:
      - SLOC
      - Function Points
      - Proxies - proxies allow the user to specify his or her own size measure, which the model later coverts to SLOC
    - In addition, all software is categorized as:
      - "New"
      - "Preexists Designed for Reuse"
      - "Preexists **not** Designed for Reuse"
    - For *pre-existing software*, users must specify the amount of software deleted, plus the percentages of redesign, reimplementation, and retest required to **modify** or **reuse** the program for the current application.
    - Because the model uses the **Program Evaluation and Review Technique** (**PERT**), users must input a "minimum," "most likely," and "maximum" value for **all** size inputs.
  - ***(2) Knowledge-Base Inputs:***
    - *SEER-SEM® contains knowledge bases* for different types of software.
    - Knowledge bases assign default values to the input parameters described below, based on the type of software selected.

- Users must address these inputs to specify the knowledge base to be used by the model:
    - ***Platform***: *The* *operating environment* of the program (e.g., avionics, ground-based, or manned space).
    - ***Application***: *The* *overall software function* (e.g., command and control, mission planning, or testing).
    - ***Acquisition method***: *The method in which the software is to be* *acquired* (e.g., development, modification, or re-engineering).
    - ***Development method***: *The* *method* used for development (e.g., waterfall, evolutionary, or spiral).
    - ***Development standard***: *The* *standard* used in development and the degree of tailoring (e.g., MIL-STD-498 weapons, ANSI J-STD 016 full, ANSI J-STD 016 nominal, or commercial).
    - ***Class***: *This input is primarily for user-defined knowledge bases.*
    - ***COTS** component type: The type of COTS program, if any, such as class library, database, or applications*.
        - COTS relates to activities associated with *incorporating* commercial software components into development activities.
        - That means that software systems are built using commercial **off-the-shelf** SW packages purchased from vendors and integrating them in the application "**buy-and-integrate**" approach.

- ***(3) Input Parameters:***
    - *SEER-SEM® contains over 30 input parameters*, from which users can refine their estimates.
    - Similar to COCOMO 81 and COCOMO II, the input values generally range from "*very low*" to "*extra high*."
    - As in size, users must specify a "*least*," "*greatest*," and "*most likely*," value for each input.
    - The selected knowledge base computes **default values** for most input parameters.
    - Therefore, if users are unfamiliar with a particular parameter, they can use the knowledge-base **default values**.

- The primary categories of input parameters and a brief description of each follows:
    - ***(1) Personnel capability and experience**: The 7 parameters* in this category, similar to the "*personnel attributes*" of COCOMO 81, measure the **caliber of personnel** used on the project. These inputs are:
        - (1.1) Analyst Capability
        - (1.2) Applications Experience
        - (1.3) Programmer Capability
        - (1.4) Language Experience
        - (1.5) Host-Development System Experience
        - (1.6) Target System Experience
        - (1.7) Practices and Methods Experience
    - ***(2) Development support environment:** The 9 parameters* in this category are similar to the project attributes and some of the computer attributes in COCOMO 81. They include:
        - (2.1) Usage of modern development practices
        - (2.2) Usage of automated tools
        - (2.3) Turnaround time
        - (2.4) Terminal response time
        - (2.5) Multiple site development
        - (2.6) Resource dedication
        - (2.7) Resource and support location
        - (2.8) Host system volatility
        - (2.9) Target system volatility
    - ***(3) Product development requirements**: The 5 parameters* in this category include:
        - (3.1) Requirements volatility
        - (3.2) Re-hosting from development to target computer
        - (3.3) Specification level
        - (3.4) Test level
        - (3.5) Quality assurance level
- The last three parameters are identical to the *reliability attributes* described above for COCOMO 81.

- ***(4) Reusability requirements:*** *The 2 parameters* in this category measure:
  - (4.1) The degree of reuse needed for future programs
  - (4.2) The percentage of software affected by reusability requirements
- ***(5) Development environment complexity:*** *The 4 parameters* in this category measure the:
  - (5.1) Language complexity
  - (5.2) Host development system complexity
  - (5.3) Application class complexity
  - (5.4) The impact of process improvements
- ***(6) Target environment:*** *The 7 parameters* in this category are similar to some of the computer attributes of COCOMO 81, but focus on the target computer. These include:
  - (6.1) Special display requirements
  - (6.2) Memory constraints
  - (6.3) Time constraints
  - (6.4) Real-time code
  - (6.5) Target-system complexity
  - (6.6) Target-system volatility
  - (6.7) Security (this is the most sensitive input parameter in the model)
- ***(7) Other input parameters****: There are also* special inputs for:
  - (7.1) Schedule constraints
  - (7.2) Labor rates
  - (7.3) Integration requirements
  - (7.4) Personnel costs
  - (7.5) Metrics
  - (7.6) Software support

SEER-SEM® Processing

- Although the model is proprietary, some of the equations of the SEER-SEM® model can be found in the User's Manual, as well as in other published articles.
    - (1) **Estimated effort** is proportional to *size* raised to an *entropy factor*, which is nominally 1.2 (as in embedded COCOMO 81), but can vary based on user selected options.
    - (2) **Schedule** is estimated in a similar manner, but is less sensitive to size.
- In SEER-SEM® estimated effort spread over the estimated schedule provides a staffing profile.
    - The model can accommodate many different **staffing profiles:**
        - Including the traditional Rayleigh-Norden profile
        - Fixed staffing
        - Mixed profiles
- *Staffing constraints* are evaluated against a project's ability to absorb staff
- Productivity adjustments can be made for **over-staffing** and **under-staffing** situations.
- Before *effort*, *schedule*, and *defects* are computed, SEER-SEM® makes several intermediate calculations.
- **Effective size** is computed from *new* and *reused code*.
- An *effective technology rating* (ETR) is computed using technology and environment parameters, and *staffing rates* are determined by the application's complexity and selected staffing profile.

SEER-SEM® Outputs

- SEER-SEM® allows the users to select a *variety* of outputs.
- (1) The model provides **labor estimates** in the categories of:
    - *Management*
    - *Systems engineering*
    - *Design*
    - *Code*
    - *Data*
    - *Test*
    - *Configuration management*
    - *Quality assurance*
- (2) A **quick estimate** provides a snapshot of:

- Size
- Effort
- Schedule
- ETR (Effective Technology Rating)
- Other selected outputs
  - The quick estimate can be provided anytime during the estimating process.

- (3) **Optional outputs** include:
  - A basic estimate
  - Staffing by month
  - Cost by month
  - Cost by activity
  - Person-months by activity
  - Delivered defects
  - A SEI maturity rating

SEER-SEM® Support Cost Considerations

- SEER-SEM® contains an optional "*maintenance*" output report that provides annual costs and person-months for each year of a user-specified schedule in four categories:
  - (1) Corrective
  - (2) Adaptive
  - (3) Perfective
  - (4) Enhancements
- The user can specify the *support time period desired*, along with several other support-unique parameters. These include:
  - Annual change rate
  - Number of support sites
  - Expected program growth
  - Differences between development and support personnel and environment
  - Minimum or maximum staffing constraints
  - Percent of code to be maintained
  - Degree of rigor (level of support)

5.5.3 Cost Model Calibration

- By definition, calibration adjusts a *commercial parametric software estimating model* to a *specific user's environment*.
- At minimum, calibration *should be performed* if the model will be used to develop estimates for proposals that will be submitted to the Government or a higher tier contractor.
- A 1981 study by Thibodeau disclosed that calibration can improve model accuracy by as much as *400 percent*.
    - In addition, other studies have shown accuracy improvements of varying degrees when models are calibrated.
- Furthermore, most model developers *encourage model calibration* and they usually provide instructions for performing this process in their user manuals.
- An overview of the calibration processes for the models previously discussed is provided below.

COCOMO Calibration

- **Intermediate COCOMO 81** users can either calibrate:
    - Only the *coefficient* (e.g., 2.8 for the embedded category),
    - Or both the *coefficient* and *exponent* (e.g., 2.8 and 1.20 for the embedded category), if sufficient historical data are available.
- In addition, **COCOMO II** has an *on-line calibration capability*, as discussed in the COCOMO II Reference Manual.
    - This allows users to calibrate the *coefficients* of the *effort* and *schedule estimating equations*.
    - The on-line capability also allows users to calibrate both the *effort and schedule coefficient* and *exponent terms* simultaneously.
    - However, this capability *should be used* carefully since the COCOMO II equation exponents are variable and depend on several factors that may be difficult to assess from historical data.

PRICE S® Calibration

- The PRICE S® model should be calibrated to adjust selected parameters so the outputs generated by the model are reflective of the user's environment.
- The most common calibration is that of the PROFAC (Productivity Factor).

  - According to the PRICE S® Reference Manual, PROFAC tends to remain constant for a given organization, particularly in the short term.
  - To calibrate PROFAC for a CSCI, the user must input actual cost or effort, development schedule, PLTFM, INTEGI, UTIL, and management complexity.
- It is also possible to calibrate platform, application, and selected internal parameters.

SEER-SEM® Calibration

- SEER-SEM® contains a "*calibration mode*" which compares actual to estimated effort and schedule figures, and then computes four suggested calibration figures.
- Users must enter the *knowledge-base inputs*, *actual effort, actual schedule, size*, and whether *requirements analysis* or *system test activities* are included in the actual effort or schedule.
- It is also desirable, but not mandatory, that users include *labor categories* and *ratings* for as many parameters (e.g., applications experience) as are known.
- Using the entered information, the model computes four calibration adjustment factors:
  - (1) Effort Adjustment
  - (2) Schedule Adjustment
  - (3) Technology Adjustment
  - (4) Complexity Adjustment
  - The Effort and Schedule Adjustment factors are *linear* multipliers that do not change any inputs, but scale the output effort and schedule.
    - For **example**, an effort adjustment factor of 1.19 will add 19 percent to computer effort.
  - The Technology and Complexity multipliers adjust intermediate factors that, in turn, adjust selected model inputs.
    - Their effect on effort and schedule is non-linear.
  - It may be best for inexperienced SEER-SEM® users to use only effort and schedule multipliers.

5.5.4 Cost Model Selection

- With the multitude of software cost and sizing models available, selecting the *appropriate tool* can be difficult.
- A *four-step approach* can be used in the selection process. The four steps are:
  - (1) Determine user needs;
  - (2) Select candidate models;
  - (3) Choose the most appropriate model or models;
  - (4) Reevaluate the choice.

(1) Step 1: Determine User Needs

- This first step is the most crucial. *Different models* are best for *different applications*, and the user should understand the unique requirements of the program.
  - (a) The user should first write a *general statement of the organization's needs*,
  - (b) Then attempt to expand the information in more detail.
  - (c) A *weighted factors approach* such as that illustrated in figure 5.5.4.a, can be used to clearly define each unique situation.
- The list of factors and weightings shown in figure 5.5.4.a reflects the importance of these *factors* to the user's organization (columns 1-2).
  - They are only presented as an example, the factors and weightings for other organizations may be quite *different*.
  - Also, the listing of factors and assignment of weightings can be subjective.
- However, such an approach can provide a *framework* for considering qualitative evaluation factors.

(2) Step 2: Select Candidate Models

- The second step is to select a *set of candidate models* that meet the needs identified through Step 1.
  - An examination of needs can point out the most suitable models.
- For size estimation, various categories of models (e.g., analogy, bottom-up, expert judgment, or parametric) can be selected.
- However, for cost models, the choices will probably focus on *parametric models*.
- Once the category or categories are identified, candidate models can be selected.

Factor	Importance Rating	Model Ratings			Sub-Factor Products		
		A	B	C	A	B	C
Input Data Availability	10	10	9	7	100	90	70
Design Evaluation Criteria	9	10	6	7	90	54	63
Ease of Use	8	8	9	6	64	72	48
Ease of Calibration	6	2	5	5	12	30	30
Database Validity	5	7	7	4	35	35	20
Currentness	5	3	5	5	15	25	25
Accessibility	4	6	9	4	24	36	16
Range of Applicability	2	1	7	10	2	14	20
Ease of Modification	1	3	4	2	3	4	2
Weighted Totals					**345**	**360**	**294**

Fig. 5.5.4.a. The Weighted Factors Approach

(3) Step 3: Choose the Most Appropriate Model or Models

- The user should perform both qualitative and quantitative (accuracy) assessments of the candidate models selected in Step 2, and choose the best model or models for their organization.
- For software estimates, it is recommended that two models be selected for routine use: one as the *primary model* and one for *cross-checking* the results of the primary model.
    - A study by Coggins and Russell showed that software cost models, even given "*equivalent*" *inputs*, produce significantly different *cost and schedule estimates*.
    - Their conclusion was that a user should learn one or two models well, instead of trying to use several different models.
    - Nevertheless, other models can still be used, even if only for consideration for future use as discussed below, in Step 4.

- In order to choose a model effectively, users must become *familiar* with each of the candidates. This often involves attending a training course and using the model for several months.
- Once the user becomes sufficiently familiar with the models, the selection process can begin.
    - It is highly desirable that the users perform their own studies, and not rely solely on outside information.
    - Nevertheless, *validation studies* performed by outside agencies certainly can help the user in the model selection process.
- An excellent ***example*** is a study by Institute for Defense Analysis.
    - This study compared and evaluated features of most of the cost models currently in use by Industry and Government.
    - While outside studies can provide valuable information, they should be used as supplementary material since they may not reflect the unique facets of the user's environment.
- For qualitative assessments of candidate models, the Weighted-Factors Approach shown in fig. 5.5.4.a. can help.
    - (1)The user first assigns a weight to each factor (in Step 1);
    - (2) Then assigns a rating between "1" and "10" to each model on how well it addresses each factor; then multiplies the model and importance ratings;
    - (3) Then sums the results.
    - (4) The *highest total* can indicate the best model alternative (e.g., Model B in fig. 5.5.4.a.).
- However, models that are close in score to the highest (e.g., Model A in Figure fig. 5.5.4.a.), should possibly be scrutinized further.
- Since there is some subjectivity in this process, small differences may be negligible.
- Again, while the Weighted-Factors Approach is somewhat subjective, it can help a user consider what is important in model selection and in quantifying the rating process.
- For quantitative assessments, or in determining whether the models meet accuracy requirements, users *should calibrate* the models, and then *run* the models against projects for which the user has historical data that was not used during the calibration process.

(4) Step 4: Reevaluate the Choice

- User needs and models can change over time.

- Many commercial models such as PRICE S® and SEER-SEM® are *updated* every year, and major refinements occur every few years.
    - New models occasionally appear that *could be more suitable* than the current models being used by the organization.
- Therefore, a user should reevaluate his or her selection *every few years*.
- There is no reason to be "married" to a particular model or models for life unless they continue to be the best available.

Conclusion

- The four-step approach previously presented can help a user in model selection.
- The most crucial step in this process is the first: needs determination.
- The remaining steps hinge on the success of the first step.
- The four-step approach is sometimes laborious, but the benefits of improved estimating can make it worthwhile.

5.6 Software Activity Productivity

- The productivity of Software Activity is *a measure of the speed* the implied engineers in the development activity of a SW project produce individual software and associated documentation.
- Usually **Software productivity** is expressed in:
    - (1) **LOC/time_unit**
    - (2) **Functionalities/time_unit.**
- **SW productivity** can be:
    - Not quality oriented
    - Quality oriented
- Usually the productivity's metrics based on volume/time_unit, do not take into account the quality (QA).
- SW productivity depends mainly on:
    - (1) The *level of the programming language* used for software development (assembly language, high level language, specialized language)
    - (2) The *nature of the developed project* (Real-Time embedded systems, system applications, commercial applications)
    - (3) The *nature and the facilities of the development environment*.

- **Factors** affecting productivity:
  - (1) Application domain experience.
    - Knowledge of the application domain is essential for the effective software development.
    - Engineers who already understand a domain are more likely to be the most productive
  - (2) Process quality.
    - The development process can have a significant effect on productivity.
  - (3) Project size.
    - The larger a project, the more time required for team communications. Less time is available for development so individual productivity is reduced
  - (4) Technology support.
    - Good support technology as CASE tools, supportive configuration management systems can improve productivity.
  - (5) Working environment.
    - A quiet working environment with private work areas contributes to improved productivity.
- ***Examples*** of software productivity:
    - **Embedded RT Systems:** 40 - 160 LOC/month
    - **System Programs:** 150 - 200 LOC/month
    - **Commercial application:** 200 - 800 LOC/month.
- **Remark**: at this time it is **not** known a relation connecting **productivity** and **quality**