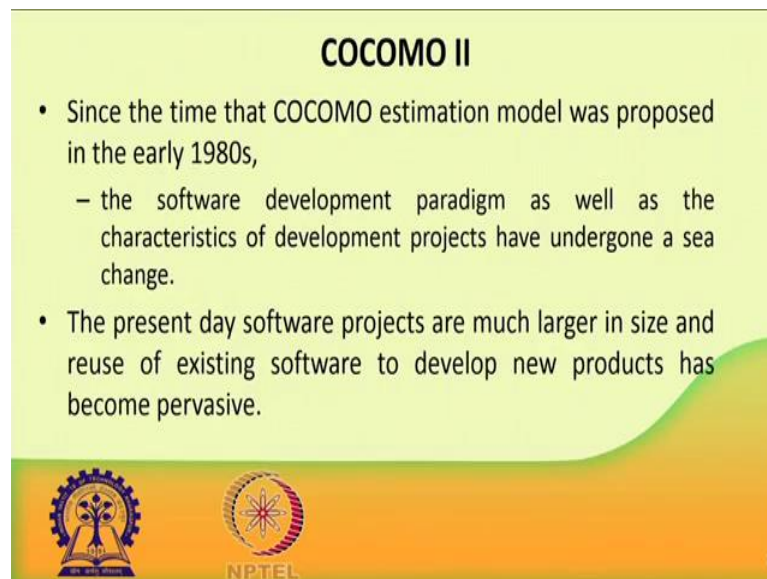


Software Project Management
Prof. Durga Prasad Mohapatra
Department of Computer Science and Engineering
National Institute of Technology, Rourkela

Lecture - 23
Project Estimation Techniques (Contd.)



Good morning, so now let us see another important type of cost estimation technique another variation of COCOMO, we call it as COCOMO II. So, somewhere you will find it as COCOMO roman letters II somewhere in some books they have written COCOMO that digit as COCOMO 2 . So, that will that will be no problem.

(Refer Side Time: 00:37)



COCOMO II

- Since the time that COCOMO estimation model was proposed in the early 1980s,
 - the software development paradigm as well as the characteristics of development projects have undergone a sea change.
- The present day software projects are much larger in size and reuse of existing software to develop new products has become pervasive.

So, COCOMO II you see that since the time that COCOMO estimation model was proposed in early 1980s; then the software development paradigm as well as the characteristics of develop projects have undergone a large change. The present day software projects are much larger in size and they reuse the existing software components to develop new products that has also become pervasive.

So, similarly other new techniques such as reverse engineering, reengineering all those things have use of automated tools etcetera case tools of come up. So, they need to be addressed while estimating the cost. So, COCOMO the initial version of COCOMO that has been extended, and that is known as the COCOMO II.

(Refer Side Time: 01:32)

COCOMO II

- For example, component-based development and service-oriented architectures (SOA) have become very popular.
- New life cycle models and development paradigms are being deployed for web-based and component-based software.
- During the 1980s rarely any program was interactive, and graphical user interfaces were almost non-existent.

The slide features a video inset of a man in a light-colored jacket speaking. At the bottom left, there are logos for IIT Bombay and NPTEL.

For example, in case of component-based software development and service oriented architectures they have also become popular. How we can estimate the effort or the cost of or developing component based software or for developing SOA based applications, new life cycle models such as your RAD, scrum or extreme programming etcetera.

They have come up also new development paradigms are being deployed for web-based applications for component-based software. So, they need to be also taken into account while estimating the effort and cost. During the 1900 1980s rarely any program was interactive but now you see due to because graphical users were GUIs Graphical User Interfaces were also almost non-existent. But now did you see any program without having GUI it is of no use nobody will use that one.

So, how to taken to account the development of GUIs while estimating the effort, cost etcetera that is also very much important.

(Refer Side Time: 02:37)

COCOMO II

- On the other hand, the present day software products are highly interactive and support elaborate graphical user interface.
- Effort spent on developing the GUI part is often as much as the effort spent on developing the actual functionality of the software.



On the other hand, the present day software products are highly interactive, they support elaborated graphical user interfaces, so they have to be considered while estimating the effort and cost. Effort spent on developing the GUI part is often as much as the effort spent on developing the actual functionality of the software, see almost 50 percent of the effort is spent on the developing the GUI part, so that cannot be neglected.

So, since effort spent on developing the GUI part is often as much as effort spent on developing the actual functionality of the software. So, we must have to taken to account taken to consideration this developing GUI part while estimating the effort and cost.

(Refer Side Time: 03:20)

COCOMO II

- To make COCOMO suitable in the changed scenario,
 - Boehm proposed COCOMO 2 in 1995.
- COCOMO 2 provides three models to arrive at increasingly accurate cost estimations.
- These can be used to estimate project costs at different phases of the software product.
- As the project progresses, these models can be applied at the different stages of the same project.

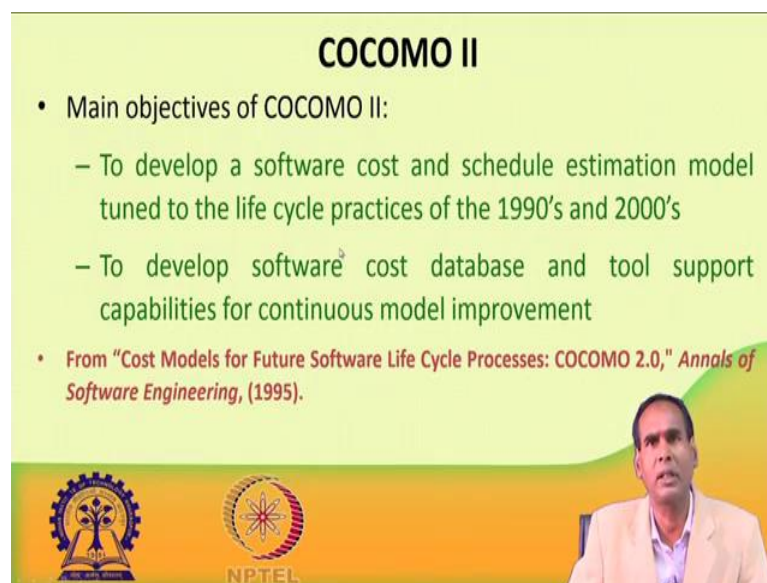


In order to make COCOMO suitable in this changed scenario where this modern programming paradigms are there, GUI is there and this is say other technique reverse engineering reuse etcetera there Boehm proposed COCOMO 2 in 1995. So, in as I have already told you some books they are using this COCOMO II roman letter some are just this COCOMO 2 in this digit, so please do not confuse.

COCOMO 2 provides three models to arrive at the increasingly accurate cost estimations taking into account on those all those new things modern techniques. This can be used to estimate the project cost at a different phases of the software product. As the project progresses these models of they can be applied at different stages of the same project.

Actually here it is written three models, but now a days we have found recently this recent material they are saying four model, I will see one more model that is added that is this for a reuse. So, you can correct it as four models.

(Refer Side Time: 04:27)



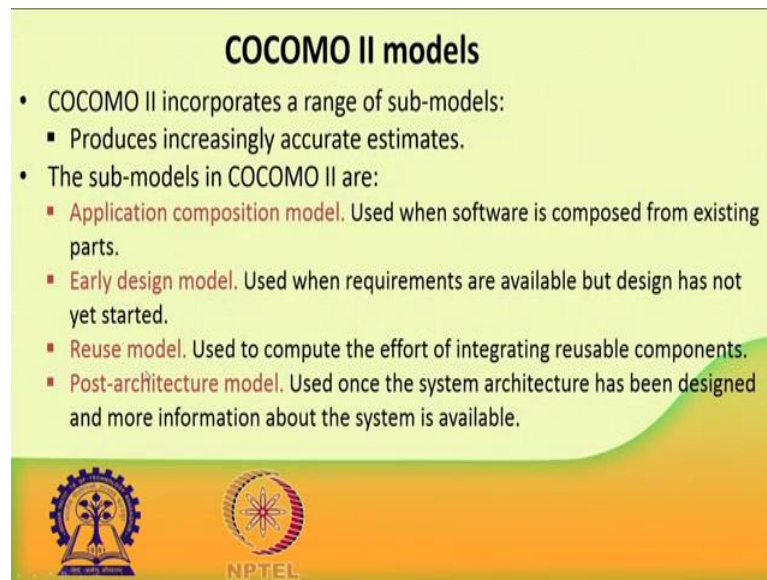
COCOMO II

- Main objectives of COCOMO II:
 - To develop a software cost and schedule estimation model tuned to the life cycle practices of the 1990's and 2000's
 - To develop software cost database and tool support capabilities for continuous model improvement
- From "Cost Models for Future Software Life Cycle Processes: COCOMO 2.0," *Annals of Software Engineering*, (1995).

The slide includes a logo on the left, the NPTEL logo in the center, and a small video inset of a man speaking on the right.

So, now, let us see what are the main objectives of COCOMO II. The main objective of COCOMO II is to develop a software cost and schedule estimation model which is tuned to the life cycle practices of 1990's and 2000's. Another objective is to develop software cost database and tool support capabilities for continuous model improvement. So, this has been taken from the Cost Models for Future Software Life Cycle Processes: COCOMO 2.0 in the published in the Annals of Software Engineering, 1995.

(Refer Side Time: 05:01)



The slide features a light green background with a yellow-to-orange gradient at the bottom. At the top center, the title "COCOMO II models" is written in bold black font. Below the title, there are two main bullet points. The first bullet point states "COCOMO II incorporates a range of sub-models:" followed by a sub-bullet "Produces increasingly accurate estimates." The second bullet point states "The sub-models in COCOMO II are:" followed by four sub-bullets: "Application composition model. Used when software is composed from existing parts.", "Early design model. Used when requirements are available but design has not yet started.", "Reuse model. Used to compute the effort of integrating reusable components.", and "Post-architecture model. Used once the system architecture has been designed and more information about the system is available." At the bottom of the slide, there are two logos: on the left, the logo of Anna University, and on the right, the NPTEL logo.

COCOMO II models

- COCOMO II incorporates a range of sub-models:
 - Produces increasingly accurate estimates.
- The sub-models in COCOMO II are:
 - **Application composition model.** Used when software is composed from existing parts.
 - **Early design model.** Used when requirements are available but design has not yet started.
 - **Reuse model.** Used to compute the effort of integrating reusable components.
 - **Post-architecture model.** Used once the system architecture has been designed and more information about the system is available.

COCOMO II models are, let us see the COCOMO II incorporates a range of sub-models, it produces increasingly accurate estimates. So, here the estimates are more accurate than the previous COCOMO models. The sub models in COCOMO II are I have already told you there will be four sub-models, those are application composition model, early design model, reuse model, and post-architecture model.

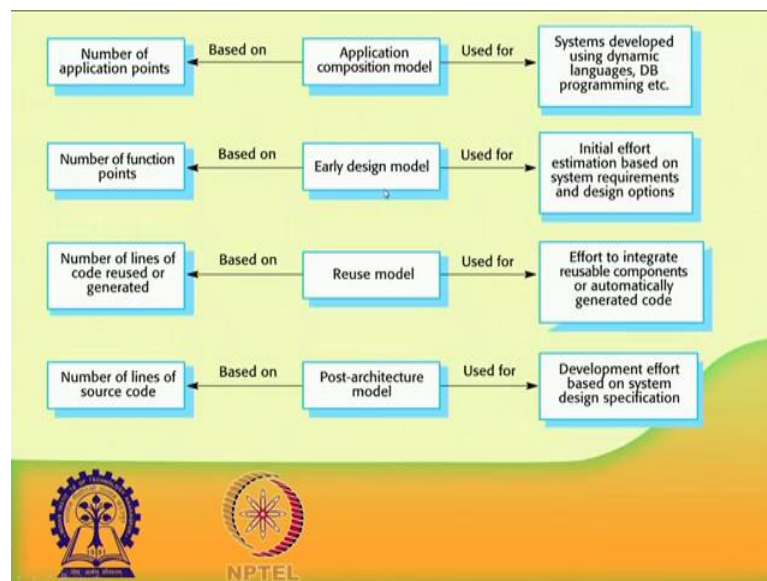
So, application composition model is used when the software is composed from the existing parts. Some parts are there you are composing what these existing parts to develop a new one, so when, so then you have to use application composition model. So, this model is used when software is composed from existing parts.

Next one is early design model. So, this -sub model is used when the requirements are available, but design has not yet started only requirements are known then you can go for early design model; that is why the name early is used. Reuse model, so it is used to compute the effort of integrating reusable components.

So, there are some component, some code, or some database they are already existing; they can be reusable in another application then you can use reuse model. So, reuse model is used to compute the effort of integrating some reusable components. And next one is post-architectural model, this is used once the system architecture has been designed and no information about the system is available.

So, when the system architecture has already been designed but more information about the system and more information about the system is available then you can get use this post-architecture model that is why the name is post, post- architecture Architecture is already designed and more information is already available, that is why we are we have to now develop the model and estimates the cost etcetera. So, since after this architecture is prepared this is known as post-architecture model.

(Refer Side Time: 06:57)



So, this is what summarized here, application composition model is based on the number of application points and it is used for systems developed using dynamic languages, data bases, programming data base programming etcetera and early design model is based on number of function points which is used for the initial effort estimation based on system requirements and design options.

Reuse model is based on number of lines of a code that is reused or generated and it is used for effort to integrate reusable components or automatically generated code. Post-architecture model is based on numbers of lines of source code which are used and this is used for development effort which is based on system design specification.

(Refer Side Time: 07:42)

COCOMO II

COCOMO 81 DSI - "Delivered Source Instructions"
COCOMO II SLOC - "Source Lines Of Code"

- The original COCOMO 81 model was defined in terms of Delivered Source Instructions, which are very similar to SLOC.
- The major difference between DSI and SLOC is that a single Source Line of Code may be several physical lines.
- For example, an "if-then-else" statement would be counted as one SLOC, but might be counted as several DSI.



So, in COCOMO 81, we are using the term DSI which stands for the Delivered Source Instructions, but in COCOMO II, we are using term SLOC Source Lines of Code. So, that we have which is, so this Delivered Source Instructions is very much similar to SLOC, but there is a one difference.

The major difference is that, a single Source Line of Code it may be several physical lines. For example, if there is a if-then-else statement, then this if-then-else statement would be counted as only one source lines of code. But it may be counted as several Delivered Source Instructions this is the important difference between DSI and SLOC.

(Refer Side Time: 08:28)

COCOMO II

- The core model is:
$$pm = A \times (size)^{sf} \times (em_1) \times (em_2) \times (em_3) \dots$$
where,
 - pm = person months,
 - A = 2.94,
 - size is number of thousands of lines of code,
 - sf is the scale factor
 - em is an effort multiplier




Now, let us see what is the core model how can effort, how can estimate the effort for or by using COCOMO II for any model. The core model says that

$$pm = A \times (\text{size})^{(sf)} \times (em_1) \times (em_2) \times (em_3) \dots$$

where pm is the effort in a person months, A is a constant whose value is 2.94, size is the number of thousands of lines of code, sf is known as scale factor which is taken as the exponent here and em this em i's they are what; they are the effort multipliers already effort multiplier we have seen in case of intermediate COCOMO.

(Refer Side Time: 09:12)



Application composition model

- Applicable to prototyping projects and projects where there is extensive reuse.
- Based on standard estimates of developer productivity in terms of application (object) points/month. Application points are computed using objects such as screens, reports, modules (components) etc.
- Takes CASE tool use into account.
- Formula is
 - $PM = (NAP (1 - \%reuse/100)) / PROD$
 - where, PM is the effort in person-months, NAP is the no. of application points and PROD is the productivity.

Now, let us see the first sub-model, that is a application composition model. I have already told you this is applicable to prototyping projects and projects where there is extensive use. So, this is applicable to prototyping projects, I hope you have already known about prototype models. So, this is applicable to prototyping projects and projects where is extensive scope of reuse. This is based on the standard estimates of developer productivity in terms of application points or object points.

So, this application composition model that is why the name is application. Application composition model it is based on the standard estimates of the developer productivity in terms of what; in terms of application points or object points per month. So, this what is this application points we will see, the application points are normally computed using

the objects such as the number of screens, number of reports, number of modules or components present in the your application.

Please do not confuse here this objects is nothing related to the object oriented programming ok. Here objects maybe treated the examples of objects be screens, reports, modules. So, the number of screens, number of reports, number of modules or components etcetera they can be treated as the objects and by you have to count those objects.

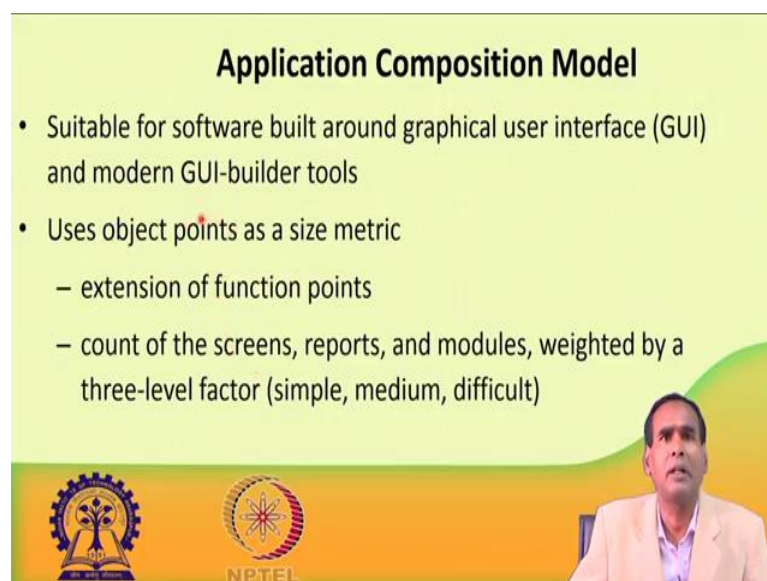
So, based on these objects you can compute the value for the application point or number of application points. So, application composition model takes the case tools used into account. So, now, let us see the formula for estimating effort using application composition model.

So, the formula for computing for estimating effort using application composition model is given by

$$PM = (NAP (1 - \%reuse/100)) / PROD$$

where PM is the effort in terms of person-months, NAP stands for number of application points which can be found out which can be computed by what using the different objects such as numbers of screens, number of reports, numbers of modules etcetera and the PROD is PROD stands for the productivity.

(Refer Side Time: 11:30)



Application Composition Model

- Suitable for software built around graphical user interface (GUI) and modern GUI-builder tools
- Uses object points as a size metric
 - extension of function points
 - count of the screens, reports, and modules, weighted by a three-level factor (simple, medium, difficult)

The slide features a light green background with a yellow-to-orange gradient at the bottom. On the left side of the bottom gradient, there are two circular logos: the first is the logo of Anna University, and the second is the NPTEL logo. On the right side of the bottom gradient, there is a small inset video frame showing a man in a light-colored jacket speaking.

So, this model is suitable for softwares built around Graphical User Interfaces and the modern GUI builder tools. This model uses object points as a size metric. So, I have already told you this model uses application points or object points as the what size metric, this is a extension of these object point or application point these are extensions of function points.

Here we have to count these object use or the object point or application point it is a count of the screens, reports, and modules; this I have already told you. This is based on counting the number of screens, number of reports, number of modules etcetera which are weighted by a three-level factor, either they could be simple, medium, or difficult.

So, the application points or objects points they are size metrics and they are based on counting the number of screens, reports, and modules which are weighted by a three-level factor may be simple, medium, and difficult.

(Refer Side Time: 12:38)



Application Points

- Used with languages such as database programming languages or scripting languages.
- Number of application points is a weighted estimate of:
 - **Number of separate screens that are displayed:** Simple screens count as 1 object point, moderately complex screens count as 2 and very complex screens count as 3 object points.

The slide features a green and yellow background. At the bottom left, there are two logos: the Indian Institute of Technology (IIT) logo and the NPTEL logo. On the bottom right, there is a small video inset showing a man in a light-colored jacket speaking.

So, now let us see quickly about application points. These are reused with languages such as database programming languages, or scripting languages. Number of application points is a weighted estimate of three things, I have already told here that these number of application points they can be used or they can be found out by counting the number of screens, reports, and modules weighted by a three-level factor.

(Refer Side Time: 13:14)

Application Points cont ...

- **Number of reports that are produced:** For simple reports, count 2 object points, for moderately complex reports, count 5 and for reports which are likely to be difficult to produce, count 8 object points.
- **Number of modules** in languages such as Java or C++ that must be developed to supplement the database programming code. Each of these modules counts as 10 object points.



You can see that the number of application points is weighted estimate of three things, what; the number of separate screens that are displayed, the number of reports that are produced, and the number of modules.

So, let us see the first one, number of separate screens that are displayed. So, here simple screens they are counted as 1 object point, moderately complex screens they are count as 2 and very complex screens they are counted as 3 object points.

Similarly, number of reports that are produced, how they are categorised or rated in the scale of say 1, 2, 3 or I think something more than that. Here it is rated as the screens are rated as in between 1, 2, 3. Let us see how the number of reports they can be rated; number of report that are produced they can be rated as follows.

For simple reports count 2 object points, for moderately complex reports you count to 5, and for reports which are likely to be very much difficult to produce for count 8 object points. Similarly, number of modules in languages the number of modules how we can count? So, the number of modules in languages such as Java or C plus plus etcetera that must be developed to supplement the database programming code. Each of these modules counts as 10 object points.

So, if you are using languages such as Java or C plus plus etcetera to develop or they must be developed to supplement the database programming code then each of these modules they counts as 10 object points.

(Refer Side Time: 14:44)

Application-point productivity

Developer's experience and capability	Very low	Low	Nominal	High	Very high
ICASE maturity and capability	Very low	Low	Nominal	High	Very high
PROD (NAP/month)	4	7	13	25	50



See in the this is a table showing the application-point productivity. So, the developer's experience and capability they are rated as like on a 5 point scale; very low, low, nominal, high, very high. Similarly the ICASE maturity capability, it can be very low, low, nominal, high, very high; these are the values of productivity.

That means if developer's experience and capability is very low and maturity and capability is very low; use a factor of 4. This is the number of, so here productivity is counted as number of application points divided by month. Similarly if developer's experience and capability is low, maturity and capability is low; then the productivity is counted as 7.

So, this have been what Boehm has already produced. These values have been has have already been produced by conducting so many research works by taking different what data.

(Refer Side Time: 15:39)

The Scale Drivers (Exponents)

- An important factor contributing to a project's duration and cost is the Scale Driver.
- The Scale Drivers determine the exponent used in the Effort Equation.
- Scale Drivers have replaced the Development Modes of COCOMO 81.
- The 5 Scale Drivers are:
 - Precedentedness
 - Development Flexibility
 - Architecture / Risk Resolution
 - Team Cohesion
 - Process Maturity



Now, let us see about The Scale Drivers, I have already showed you in the what in the initial function there was a what factor called as this scale factor. Now, let us see how this scale factor is defined. The scale drivers or the exponent parts are computed like this.

This is an important factor contributing to a project's duration and cost. The scale drivers they determine the exponent that has to be used in the Effort equation. Scale drivers are replaced; they have replaced the development modes of COCOMO 81, see in COCOMO 81 we are having three development modes.

One for your organic, semidetached and embedded but in COCOMO II they have used scale drivers and they have replaced all those three development modes of COCOMO 81. What are the possible 5 scale drivers used in COCOMO II; they are as follows. precededness, development flexibility, architecture or risk resolution, team cohesion, and process maturity.

(Refer Side Time: 16:52)

Early Design and Post-Architecture Model

$$\text{Effort} = (\text{Environment Multipliers}) * [\text{Size}]^{(\text{Process Scale Factors})}$$

Environment: Product, Platform, People, Project Factors
Size: Reuse and volatility effects
Process: Constraint, Risk/Architecture, Team, Maturity Factors

$$\text{Schedule} = (\text{Multiplier}) * [\text{Effort}]^{(\text{Process Scale Factors})}$$


Then we will come to the next model, that early design model and the post-architecture model. So, here effort is calculated as follows,

$$\text{Effort} = (\text{Environment multipliers}) * [\text{size}]^{(\text{process scale factors})}$$

where environment multipliers could be product, platform, people, and project factors; size is what that normally that the basic size that is estimated and these here it takes into account reuse and volatility effects, process scale factors are what, the constraints, risk architecture, team, maturity factors etcetera.

The schedule can be calculated in early design and post-architecture model as follows,


$$\text{Schedule} = (\text{multiplier}) * [\text{Effort}]^{(\text{process scale factors})}$$

where these are the process scale factors, constraints, risk architecture, team and maturity factors.

(Refer Side Time: 17:50)

COCOMO II Scaling Exponent Approach

- Nominal person-months = $A * (\text{size})^B$
- $B = 0.91 + 0.01 \sum(\text{scale factor ratings})$
 - B ranges from 0.91 to 1.23
 - 5 scale factors; 6 rating levels each
- Scale factors:
 - Precedentedness (PREC)
 - Development flexibility (FLEX)
 - Architecture/ risk resolution (RESL)
 - Team cohesion (TEAM)
 - Process maturity (PMAT, derived from SEI CMM)



Now, let us see another approach COCOMO II scaling exponent approach. So, here the nominal person-months is calculated as follows,

$$\text{Nominal person-months} = A * (\text{size})^B$$

where B is computed as using this formula


$$B = 0.91 + 0.01 \sum(\text{scale factor ratings})$$

B value may range from 0.91 to 1.23, there are five scale factors; and six rating levels we will see. The five scale factors are those things precededntedness, development flexibility, architecture or risk resolution, team cohesion, and process maturity and the six ratings we will; so what are these six; so, process maturity it is a derived from SEI CMM model that Capability Maturity Model. These are the codes they are referred they will be referred while doing these problems. So, now let us see what are the six rating levels for these five scale factors.

(Refer Side Time: 18:48)

Project Scale Factors

Scale Factors (W)	Very Low	Low	Nominal	High	Very High	Extra High
PREC	thoroughly unprecedented	largely unprecedented	somewhat unprecedented	generally familiar	largely familiar	thoroughly familiar
FLEX	rigorous	occasional relaxation	some relaxation	general conformity	some conformity	general goals
RESL	little (20%)	some (40%)	often (60%)	generally (75%)	mostly (90%)	full (100%)
TEAM	very difficult interactions	some difficult interactions	basically cooperative interactions	largely cooperative	highly cooperative	seamless interactions
PMAT	weighted sum of 18 KPA achievement levels					




So, these are the six rating levels; like very low, low, nominal, high, very high, and extra high. The scale factors are the codes we have used here like PREC, FLEX, RESL, TEAM, PMAT etcetera. So, those are shown here these are the scale factors and they are rated on what a 6 scale; 6 point scale and this PMAT, it is a weighted sum of 18 KPA achievement levels in where in the CMM or yes, in the CMMI.

(Refer Side Time: 19:22)

The reuse model

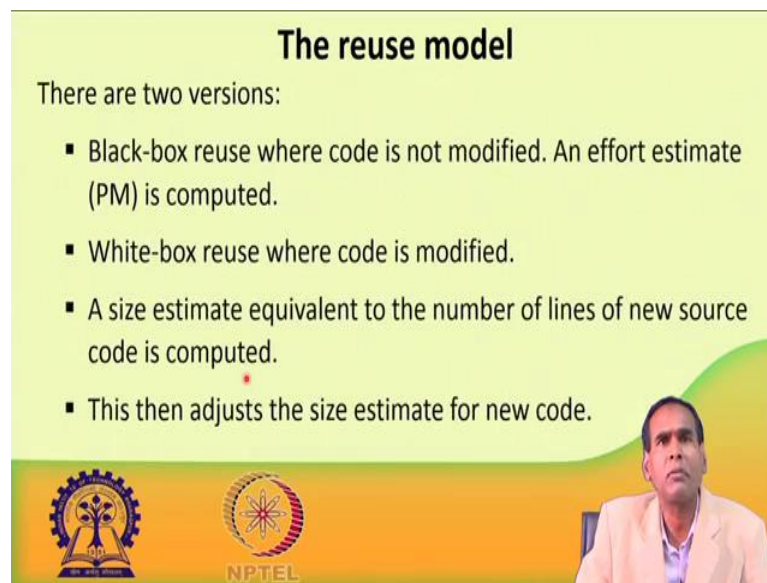
- Reuse costs:
 - overhead for assessing, selecting and assimilating component
 - small modifications generate disproportional large costs
- Takes into account:
 - black-box code that is reused without change
 - code that has to be adapted to integrate it with new code.



Then we will see the next model that the reuse model. So, reuse costs normally reuse cost consider overhead for assessing, selecting and assimilating component. So, even small modifications they generate disproportional large costs, so they are also what taken care prior.

This reuse model takes into account a black-box code that is reused without any change. The code that has to be adapted to integrate it with new code. So, this reuse model takes into account two things, the black-box code that is reused without change, and the code that has to be adapted to integrate it with the new code.

(Refer Side Time: 20:12)



The reuse model

There are two versions:

- Black-box reuse where code is not modified. An effort estimate (PM) is computed.
- White-box reuse where code is modified.
- A size estimate equivalent to the number of lines of new source code is computed.
- This then adjusts the size estimate for new code.


The slide includes the IIT Bombay logo on the left, the NPTEL logo in the center, and a small video inset of a man in a white shirt on the right.

There are two versions for this reuse model; one is the black-box reuse model, the black-box reuse model where code is not modified. An effort estimate is computed then. In white-box reuse model the code is modified then you may estimate the effort. A size estimate is equivalent to the number of lines of new source code is computed. This then adjust the size estimate for the new code, this is happening in the reuse model.

(Refer Side Time: 20:46)

Reuse model estimates

1. For generated (reused) code:
 - $PM = (ASLOC * AT/100)/ATPROD$
 - ASLOC is the number of lines of generated code
 - AT is the percentage of code automatically generated.
 - ATPROD is the productivity of engineers in integrating this code.
- 2 When code has to be understood and integrated:
 - $ESLOC = ASLOC * (1-AT/100) * AAM$.
 - ASLOC and AT as before.
 - AAM is the adaptation adjustment multiplier computed from the costs of changing the reused code, the costs of understanding how to integrate the code and the costs of reuse decision making.



The reuse estimates are like this, we will see two possibilities we will see. For the generated or reused code. Person month means the effort in person month is

$$PM = (ASLOC * AT/100)/ATPROD$$

Here ASLOC is the number of lines of generated code, AT is the percentage of code automatically change generated, ATPROD is the productivity of the engineers in integrating this code and when the code has to be understood and integrated then the formula changes. So, here ESLOC is equal to

$$ESLOC = ASLOC * (1-AT/100) * AAM.$$


Where ASLOC already we have taken earlier, AT also we have already taken earlier, ASLOC means number of lines of generated code, AT is the percentage of code automatically generated, they are as usual before they have used, and AAM this is a new term here.

So, this is the adaptation adjustment multiplier which is computed from the cost of the changing the reused code, the cost of understanding how to integrate the code and cost of reuse during decision making. So, this how when the code has to be understood and integrated you can compute this E estimated you can compute the what ESLOC.

(Refer Side Time: 22:18)

Post-architecture level

- Uses the same formula as the early design model:
 - but with 17 rather than 7 associated multipliers.
- The code size is estimated as:
 - Number of lines of new code to be developed;
 - Estimate of equivalent number of lines of new code computed using the reuse model;
 - An estimate of the number of lines of code that have to be modified according to requirements changes.



Now, we will see the last model that is post-architecture level. It uses the same formula as the early design model, but with 17 rather than 7 associated multipliers. See in the earlier one was that e or this earlier model sorry in this early design model actually there are 7 associated multipliers.

In earlier design model 7 associated multipliers are were used, but in post-architecture model 17 multipliers are used. The code size is estimated as follows, first the number of lines of new code to be developed is found out, then estimate the equivalent number of lines of new code which is computed using the reuse model earlier reuse model we have already shown.

Then an estimate of the number of lines of the code that have to be modified according to the requirements changes, whenever the requirement changes you may have to modify some of the lines or some of the code. So, then an estimate of the number of lines of code that have to be modified according to the requirements changes that has also to be estimated.

(Refer Side Time: 23:32)

COCOMO II Scale factor

Based on five factors which appear to be particularly sensitive to system size

1. **Precedentedness (PREC)**. Degree to which there are past examples that can be consulted
2. **Development flexibility (FLEX)**. Degree of flexibility that exists when implementing the project
3. **Architecture/risk resolution (RESL)**. Degree of uncertainty about requirements
4. **Team cohesion (TEAM)**.
5. **Process maturity (PMAT)** could be assessed by CMMI



Now, let us see about quickly about the COCOMO II scale factors. I have already told you there are five scale factors, this already we have seen earlier. So, these five factors appear to be particularly sensitive to system size. So, precedednedness represent the degree to which there are past examples that can be consulted already similar cases you have already handled earlier. So, what is the degree to which these past action examples can be consulted.




Development flexibility represents the degree of a flexibility that exist when implementing the project. So, during the implementation of whether many constraints are there or you are completely flexible you can take your decision, you can do these implementation yourself with there are no constraints you are free. So, development flexibility represents the degree of flexibility that exist when the when implement the project.

Architecture risk resolution it says the degree of uncertainty about requirements. So, these requirements are not clear cut to you, the requirements are very much uncertain they are not certain. So, what is the degree of uncertainty about their requirements that is represented by RESL and similar team cohesion among the team members, and process maturity this could be assessed by the CMMI Capability Maturity Model Integration.

(Refer Side Time: 24:47)

COCOMO II Scale factor values

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








So, these are the different factors for this what COCOMO II scale factors. So, when these are the drivers, these are the 1, 2, 3, 5, 6 you see rated as 6 scale and these are the values these have been given.

(Refer Side Time: 25:00)

Example Usage of scale factor

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

Then now let us quickly take some examples for using of scale factor. For example, a software development team is developing an application. It is very similar to previous ones it has already developed. So already they have developed similar types of application. So, now, you are developing a similar application right now.

It is a very precise software engineering document lays down very strict requirements ok, a very precise software engineering document lays down very strict requirement. So, it is say that precedent is very high is PREC is the precedent I think; yes, So, now precedent is very high and for high value it is 1.24 perhaps high value it is precedent high value is see very high is 1.24, so that you have used and similarly, it is say that flexibility is very low that value is 5.07 you can see flexibility is low is very low it is 5.07 and the good news is that requirements are unlikely to change requirements are it will not change, so RESL is high with a score of 2.83. RESL you can see RESL here it is high it is 2.83. So, RESL is 2.83 but the team is tightly knit.

So, team score is high, team see high is how much? High is 2.19. So, that we have shown here 2.19, but processes but processes are very informal. So, PMAT is low and for PMAT low value is 6.24; PMAT low is 6.24 now you can find out the scale factor.

(Refer Side Time: 26:41)

Scale factor calculation


The formula for sf is

$$sf = B + 0.01 \times \Sigma \text{ scale factor values}$$

i.e. $sf = 0.91 + 0.01$
 $\times (1.24 + 5.07 + 2.83 + 2.19 + 6.24)$
 $= 1.0857$

If system contained 10 kloc then estimate would be $2.94 \times 10^{1.0857} = 35.8$ person months

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



The scale the formula for scale factor sf is given like this,

$$sf = B + 0.01 \times \Sigma \text{ scale factor values}$$

that is equal to 0.91 plus 0.01 into what are the four values you have got here 1.24, 5.07, 2.83, 2.19 and 6.24. So, those values will be multiplied, so on simplification you will get 1.0857.

If a system contained suppose the system or if a system contained 10 kloc then estimate would be how much, the basic estimate is what suppose the system contained 10 kloc. So, the estimate would be how much? The obtained value of 2 point value 2.94 the value of a into size is 10 to the power this, so the exponent factor is 1.0857 this. So, on simplification you will get 35.8 person months. So, using exponentiation that means to the power of it adds disproportionately more to the estimates for larger applications.

(Refer Side Time: 27:57)

Effort multipliers

In addition to the scale factors,

- effort multipliers are also assessed. Followings are the effort multipliers

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

The slide also features logos for an institution and NPTEL, and a small video inset of a speaker in the bottom right corner.

These are the, so what besides these scale factors that we have already seen earlier five scale factors. Effort multipliers they are also assessed. Followings are the effort multipliers that you will using in COCOMO II like RCPX means product reliability and complexity, RUSE means reuse required, PDIF platform difficulty, PERS personnel capability, FCIL facilities available, SCED schedule pressure. So, these are the different effort multipliers they are also to be assessed.

(Refer Side Time: 28:27)

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








Their values are given also in a how much; 7 scale perhaps; 1, 2, 3, 4, 5, 6, 7. Yes, the values of the effort multipliers are given in 7 point scale you can use them.

(Refer Side Time: 28:39)

Example

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

We will take one more example quickly.

A new project has to be developed in ok, a new project to be developed is similar in most of the characteristics to those that an organization has been dealing for some time. Before they have already developed similar type of project, except the followings; what are the exceptions; the software to be produced is exceptionally complex and will be used in a safety critical system.


Software will interface with a new operating system that is currently in beta status and to deal with this the team allocated to the job are regarded as exceptionally good but do not have a lot of experience on this type of software, we have to now estimate.

(Refer Side Time: 29:21)

Example – cont...

RCPX	very high	1.91
PDIF	very high	1.81
PERS	extra high	0.50
PREX	nominal	1.00

All other factors are nominal
Say estimate is 35.8 person months
With effort multipliers this becomes $35.8 \times 1.91 \times 1.81 \times 0.5 \times 1 =$
61.9 person months



So, here RCPX is very high so 1.91, PDIF is very high 1.81, PERS extra high 0.50, and PREX is nominal 1.0. So, these values you can get from that table; all other factors are nominal and suppose the estimate say estimate is 35.8 person month. For example, suppose it is estimated as suppose the effort is estimated as 35.8 person months.

Then with these effort multipliers the revised estimate becomes how much; 35.8 multiplication of these multipliers, so this gives into 61.9 person months. So, in this way you can use the effort multipliers and the scaling factors to revise to refine the initial estimates using COCOMO II.

(Refer Side Time: 30:07)

COCOMO II Effort Equation

Example: A project with all Nominal Cost Drivers and Scale Drivers would have an EAF (Effort Adjustment Factor) of 1.00 and exponent, E, of 1.0997.

Assuming that the project is projected to consist of 8,000 source lines of code, COCOMO II estimates that 28.9 Person-Months of effort is required to complete it:

$$\text{Effort} = 2.94 * (1.0) * (8)^{1.0997} = 28.9 \text{ Person-Months}$$



So, another equation I have we have taken another what example, a project with all nominal cost drivers and scale drivers would have an EAF. EAF we have already discussed it earlier Effort Adjustment Factor of 1.0 and exponent, E, is 1.0997. So, these things they have already calculated EAF is calculated as 1.0 and exponent E is calculated 1.0999.

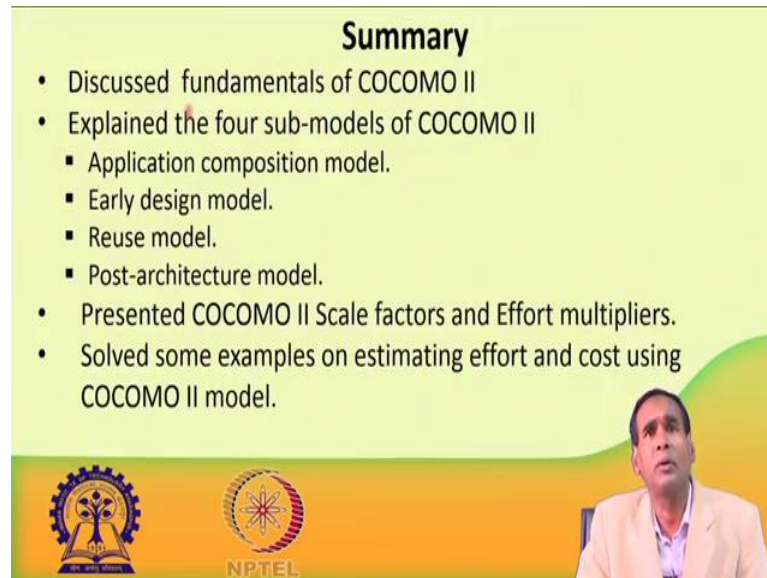
So, a project with all nominal cost drivers and scale drivers, for this project it is already calculated EAF value is 1.0 and exponent values E is 1.0997. Assuming that the project is projected to consist 8,000 source lines of code, then we have to estimate the effort. So, effort will be COCOMO II estimates 28.9 person-months using the previous equations ok, now we have to complete it.

So, how we can complete it? So, COCOMO II estimates the effort as 28.9 person-month. COCOMO II estimates that the effort is of 28.9 person months of effort is required to complete it, so now we can revise it. So, effort is equal to how much; that is value of constraint is 2.94 into what into the EAF Effort Adjustment Factor that is 1 into what is the source code, see what is this source code; 8,000 source lines.

So, this is, so in this way see how COCOMO II estimates this 28.9 it is shown here. For this data that it is a project with all nominal cost drivers, EAF is 1, exponent value is 1.0997 and the source lines is 8,000. So that means, project size is 8,000 source line, we have to estimate the effort using COCOMO II. So, that will be based on the simplest formula a into what the EAF into source lines to the power of the exponent.

So, value of a is 2.90 and EAF is equal to 1.0, size is 8,000 source line means 8 kloc and to the power of the exponent part, exponent part is 1.0997 put the values in this equation you will get the effort in person-months. So, on solving we will get approximately 28.9 person-months. So, COCOMO II estimates roughly 28.9 person-months will be required as the effort for developing this project.

(Refer Side Time: 32:44)



The image shows a presentation slide titled "Summary" with a bulleted list of topics. The slide has a light green background with a yellow-to-green gradient at the bottom. On the right side of the slide, there is a small inset video of a man in a light-colored jacket. At the bottom left, there are two logos: the Indian Institute of Technology (IIT) logo and the NPTEL logo.

Summary

- Discussed fundamentals of COCOMO II
- Explained the four sub-models of COCOMO II
 - Application composition model.
 - Early design model.
 - Reuse model.
 - Post-architecture model.
- Presented COCOMO II Scale factors and Effort multipliers.
- Solved some examples on estimating effort and cost using COCOMO II model.

So, finally, we have discussed the fundamentals of COCOMO II, explained the various sub-models of COCOMO II, presented the COCOMO II scale factors and effort multipliers, solved some examples on estimating effort and cost using COCOMO II model.

(Refer Side Time: 32:59)

References :

1. B. Hughes, M. Cotterell, R. Mall, *Software Project Management*, Fifth Edition, McGraw Hill Education (India) Pvt. Ltd., 2018.
2. R. Mall, *Fundamentals of Software Engineering*, Fifth Edition, PHI Learning Pvt. Ltd., 2018.



We have taken the references from these books.

Thank you very much.