Software Project Management

Durga Prasad Mohapatra

Professor

CSE Deptt.

NIT Rourkela

Project Estimation Techniques cont …

- **COCOMO II**

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.

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.

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.

COCOMO II

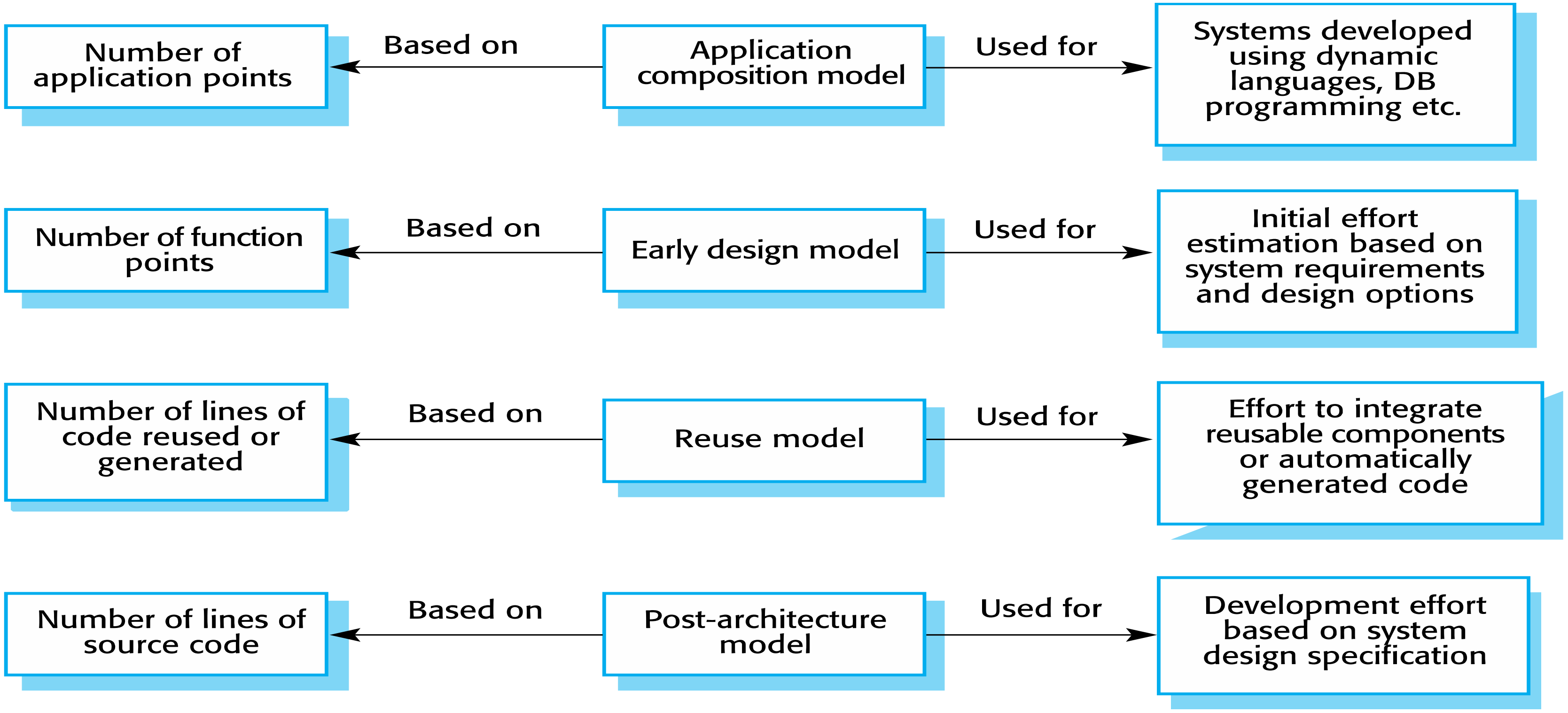
- To make COCOMO suitable in the changed scenario,
  - Boehm proposed COCOMO 2 in 1995.
- COCOMO 2 provides three models to arrive at increasingly accurate cost estimations.
- These can be used to estimate project costs at different phases of the software product.
- As the project progresses, these models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

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).**

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.


Number of application points
Based on
Application composition model
Used for
Systems developed using dynamic languages, DB programming etc.
Number of function points
Based on
Early design model
Used for
Initial effort estimation based on system requirements and design options
Number of lines of code reused or generated
Based on
Reuse model
Used for
Effort to integrate reusable components or automatically generated code
Number of lines of source code
Based on
Post-architecture model
Used for
Development effort based on system design specification

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.

COCOMO II

- The core model is:
  $pm = A \times (size)^{(sf)} \times (em_1) \times (em_2) \times (em_3)\ldots.$
  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

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.

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)

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.

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.

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

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

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

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

COCOMO II Scaling Exponent Approach

- Nominal person-months = A*(size)^B
- B = 0.91 + 0.01 ∑(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)

Project Scale Factors

Scale Factors (*Wi*)	Very Low	Low	Nominal	High	Very High	Extra High
PREC	thoroughly unprecedented	largely unprecedented	somewhat unprecedented	generally familiar	largely familiar	throughly 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					

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.

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.

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.

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.

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

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

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

Scale factor calculation

The formula for sf is

sf (B) = 0.91 + 0.01 × Σ scale factor values

i.e. sf (B) = 0.91 + 0.01

× (1.24 + 5.07 + 2.83 + 2.19 + 6.24)

= 1.0857

If system contained 10 kloc then the 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

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
PREX	Personnel experience
FCIL	Facilities available
SCED	Schedule pressure

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	

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 is regarded as exceptionally good, but do not have a lot of experience on this type of software.

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 x 1.91 x 1.81 x 0.5 x 1 = 61.9 person months

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:

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

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.

References :

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

Thank you