Tribhuvan University

Institute of Engineering

Pulchowk Campus

Department of Electronics and Computer Engineering

OBJECT ORIENTED SOFTWARE ENGINEERING

Chapter Seven

Managing Object Oriented Software engineering

by

Santosh Giri

Lecturer, IOE, Pulchowk Campus.

Overview

- ✓*Project selection and preparation*
- ✓*Project development, organization and management*
- ✓*Software project planning, scheduling and techniques*
- ✓*COCOMO model*
- ✓*Risk Management process*
- ✓*Software quality assurance*
- ✓*software metrics*

Project selection and preparation

1. Selecting the first project

- When selecting the first project there are certain things to consider. Generally, it should have the optimal conditions possible, so that the method can be evaluated without any disturbances such as unnecessary shortage of staff or problems in defining the system responsibility.
- We summarize this in the following recommendations:

(1) Select a real project that is important, but not with a tight time schedule or any other hard constraints.

(2) Select a problem domain that is well known and well defined.

(3) Select people experienced in system development who have a positive view of changes. The management should .have confidence in them.

(4) Select a project manager with a high degree of interest in the task.

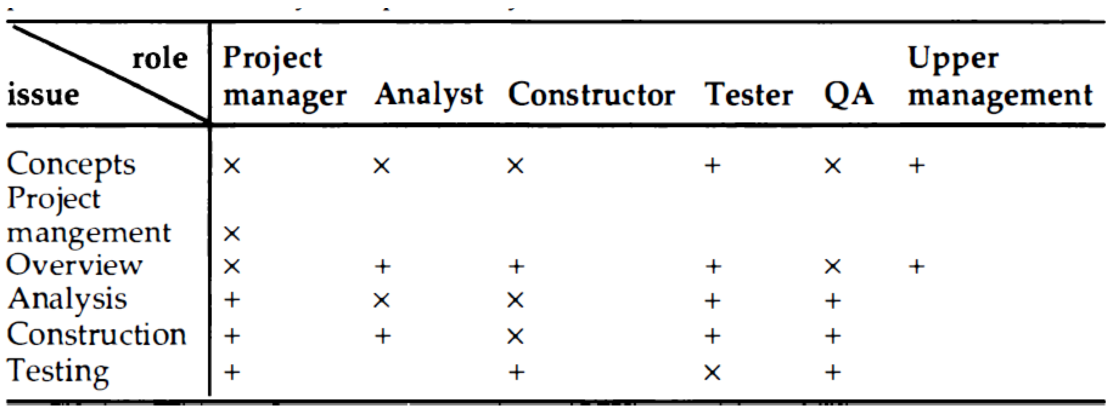
(5) The staff should work full time within the project and not be distracted by other projects.

(6) Base your work on a detailed plan developed in advance. Perform evaluation at all stages with criteria established in advance.

2. Education and Training

- All personnel involved in the new order of work need education and training. Given a strict method and process definition, more emphasis can be put on formal education and training.
- Therefore more material can be taught, and staff will need fewer projects on which to learn how to work in an orderly way. In a less formalized method, staff need several learning projects before getting familiar and highly productive with the method.
- The scope and amount of education needed varies according to each person's role in the project. Everyone should have a basic education, so that there is a common basis of concepts and way of working.
- More specialized education and training is needed for individual roles within the project according to their roles.
- We have summarized our experiences of the need for education on the table (next page).

2. Education and Training



role / issue	Project manager	Analyst	Constructor	Tester	QA	Upper management
Concepts	×	×	×	+	×	+
Project mangement	×					
Overview	×	+	+	+	×	+
Analysis	+	×	×	+	+	
Construction	+	+	×	+	+	
Testing	+		+	×	+	

Table: The need for education when introducing a new development process:

x *=necessary* *+= preferably*

Concepts

- ✓ *Basic concepts of object orientation and the fundamental concepts of the method.*
- ✓ *Duration: 1-2 days*

Project management

- ✓ *Specific characteristics for the new method. Appropriate metrics for the management within the new method.*
- ✓ *Duration: 1 day*

2. Education and Training

Overview

- ✓ *An overview of the method using examples to introduce the entire method in the system life cycle and the underlying ideas. This entire book may form a basis for this course.*
- ✓ *Attendees should be familiar with object orientation.*
- ✓ *Duration: 3- 4 days.*

Analysis

- ✓ *A thorough and detailed study of the entire analysis process.*
- ✓ *Emphasis should be on applying the method and process to a larger example, possibly on the current system.*
- ✓ *Duration: 3- 6 days.*

Construction

- ✓ *A thorough and detailed study of the entire design and implementation process. Emphasis should be on applying the method and process to a larger example, preferably the system to be built.*
- ✓ *Attending should be familiar with the programming language used.*
- ✓ *Duration: 3-6*

Testing

- ✓ *A thorough and detailed study of the testing activities and principles.*
- ✓ *Duration: 2-3 days.*

3. Risk Analysis

- Introducing new technologies always brings potential risks. The number of new technologies introduced simultaneously increases risk exponentially.
- We will here discuss a simple technique for detecting and managing risks in OOSE. The method is divided into three steps.
  (1) Risk identification
  (2) Risk valuation
  (3) Managing the risks

In ***risk identification***, we define the potential and foreseeable risks of the project that, if they occur, may seriously injure the project. The starting point should be the goal of the project. What risks may occur that will make us fail to reach the goal? These risk areas could involve OOSE-specific risks, but also other risks. There are many risks that must be identified for each project. Few examples of potencial risk areas involved in OOSE projects are:

(1) Paradigm shift

-Are we mature enough to adopt a new development strategy?

-Will the team be mature enough not to start coding too early?

-Is the project manager familiar with the new paradigm?

-Are the team members familiar with the new paradigm?

-Are we familiar with the programming language to be used?

3.1. Risk Identification

(2) Process

-Is our process well defined and well documented, and can we work through it?

-Is the process mature?

-Does the documentation produced fit our purpose?

-Do we have sufficient training capabilities?

(3) Tools

-Are we familiar with the tools to be used, and with the learning threshold?

-Are the tools to be used mature and stable?

-Will we have the tools when they are needed?

-Are the tools compatible? Are they integrated?

(4) The system

-Are we familiar with the application domain?

-Are the requirements clear, consistent and stable?

-Can or must we integrate existing systems? Is it possible?

(5) Organization

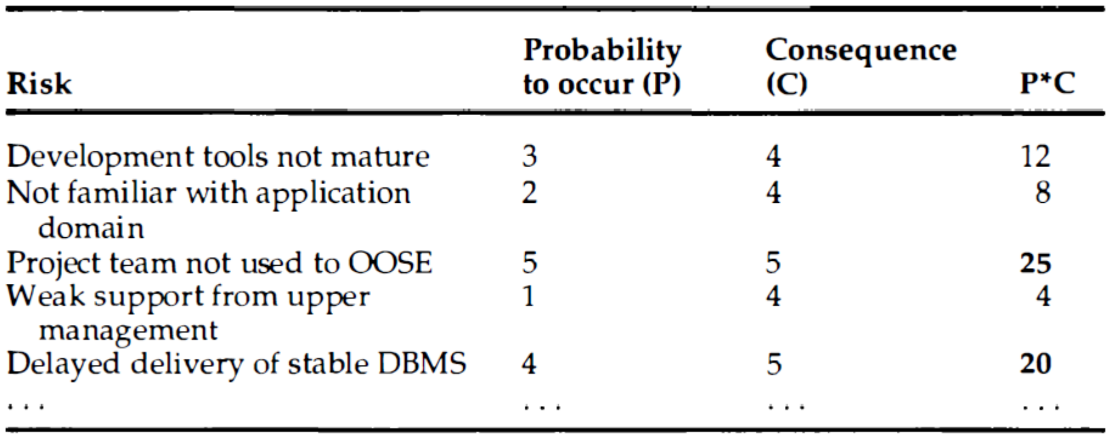
-Do we have a tight schedule?

-Will time-to-market be critical? Will the pressure from the market change?

-Does the organization have realistic expectations for the project?

3.2. Risk Valuation

- The next step is risk valuation. Now the potential risks should be evaluated to assess the consequences (disturbance or damage) if they occur.
- We then initially judge the probability that the risk will occur. A scale from 1 (very improbable) to 5 (very probable) could be used.
- Then we assess the consequences of each risk. A scale from 1 (negligible) to 5 (catastrophic) could be used.
- After that, we multiply the probability and the consequence factor for each risk (see Table). So, we will now have a relative measure of the size and potential threats of the risks.



Risk	**Probability to occur (P)**	**Consequence (C)**	**P*C**
Development tools not mature	3	4	12
Not familiar with application domain	2	4	8
Project team not used to OOSE	5	5	**25**
Weak support from upper management	1	4	4
Delayed delivery of stable DBMS	4	5	**20**
. . .	. . .	. . .	. . .

3.3. Managing the Risk

- The third step is managing the risks. From the first two steps, we establish a prioritized table of the most serious threats to the project.
- For each of these serious threats we propose active actions to prevent them from occurring or to reduce their consequences. This could be done by decreasing the probability of the risk occurring or decreasing the consequences, or both.
- In the table (previous table), we see that the major threats are that the project team are not used to OOSE and we are not sure about the delivery of the DBMSs.
- The measures to avoid these threats could be to put in place an education program for the team members and not to use the DBMS proposed.

Software Quality Assurance

Software Quality Assurance?

- Software Quality Assurance is a planned and systematic way of creating an environment to assure that the software product being developed meets the quality requirements.

- This process is controlled and determined at managerial level.

SQA Activities & Tasks

1. Prepare a SQA plan for a product

- ✓ The plan is developed during project planning and reviewed by all interested parties.
- ✓ SQA activities, performed by the s/w engineering team & SQA team are governed by the plan.

2. Participate in the development of the project's s/w process descriptions

✓The s/w engineering team select a process for work to be performed.

✓The SQA groups reviews the process descriptions for the agreements with the organizations policy, standards etc.

SQA Activities & Tasks

3. Review s/w engineering activities to verify compliance (agreement) with the defined s/w process

✓The SQA group identifies, documents and tracks deviations from the process and verify that and corrections have been made.

4. Audits designed s/w work product to verify compliance with those defined as part of the s/w process

✓The SQA groups reviews selected work products, identifies documents, tracks deviations and verify that correctness have been made and periodically reports the results of its work to the project manager.

SQA Activities & Tasks

5.Ensure that deviation in s/w work and work product are documented and handled according to the documented procedure

✓Deviation occurred in s/w product development process has been documented according to documentation procedure.

6.Periodically reports any non-compliance(agreement) and reports to the senior management

✓Non agreement items are tracked and reports to the senior management.

Software Metrics

Software metrics

Software metric

✓Any type of measurement which relates to a software system, process or related documentation.

→Lines of code in a program, the Fog index-readability test, number of person-days required to develop a component.

✓Allow the software and the software process to be quantified.

✓May be used to predict product attributes or to control the software process.

✓Product metrics can be used for general predictions or to identify anomalous components.

Software product metrics

Software Metrics	Description
Fan-in/ Fan-out	Fan-in is a measure of the number of functions or methods that call some other function or method (say X). Fan-out is the number of functions that are called by function X. A high value for fan-in means that X is tightly coupled to the rest of the design and changes to X will have extensive knock-on effects. A high value for fan-out suggests that the overall complexity of X may be high because of the complexity of the control logic needed to coordinate the called components.
Length of Code	This is a measure of the size of a program. Generally, the larger the size of the code of a component, the more complex and error-prone that component is likely to be. Length of code has been shown to be one of the most reliable metrics for predicting error proneness in components.
Length of identifiers	This is a measure of the average length of distinct identifiers in a program. The longer the identifiers, the more likely they are to be meaningful and hence the more understandable the program.
Depth of conditional Nesting	This is a measure of the depth of nesting of if-statements in a program. Deeply nested if statements are hard to understand and are potentially error-prone.
Fog index	This is a measure of the average length of words and sentences in documents. The higher the value for the Fog index, the more difficult the document is to understand.

Object oriented metrics

OO Metrics	Description
Depth of inheritance tree	This represents the number of discrete levels in the inheritance tree where subclasses inherit attributes and operations (methods) from super-classes. The deeper the inheritance tree, the more complex the design. Many different object classes may have to be understood to understand the object classes at the leaves of the tree.
Method fan-in/fan-out	This is directly related to fan-in and fan-out as described above and means essentially the same thing. However, it may be appropriate to make a distinction between calls from other methods within the object and calls from external methods.
Weighted methods per class	This is the number of methods that are included in a class, weighted by the complexity of each method. Therefore, a simple method may have a complexity of 1 and a large and complex method a much higher value. The larger the value for this metric, the more complex the object class. Complex objects are more likely to be more difficult to understand. They may not be logically cohesive so cannot be reused effectively as super-classes in an inheritance tree.
Number of overriding operations	This is the number of operations in a super-class that are over-ridden in a subclass. A high value for this metric indicates that the super-class used may not be an appropriate parent for the sub-class.

COCOMO model

Constructive Cost Model(COCOMO)

- ❖COCOMO is one of the most widely used software estimation models in the world.
- ❖In this model, size is measured in terms of thousand of delivered lines of code (KDLOC).
- ❖In order to estimate effort accurately, COCOMO model divides projects into three categories:

1. Organic projects:

- ✓These projects are small in size (not more than 50 KDLOC.)
- ✓Example of organic project are, business system, inventory management system, payroll management system, and library management system.

2. Semi-detached projects:

✓The size of semi-detached project is not more than 300 KDLOC.

✓Examples of semi-detached projects include operating system, compiler design, and database design.

3. Embedded projects:

✓These projects are complex in nature (size is more than 300 KDLOC).

✓Example of embedded projects are software system used in avionics and military hardware.

Constructive cost model is based on the hierarchy of three models

- ✓ basic model
- ✓ intermediate model and
- ✓ advance model.

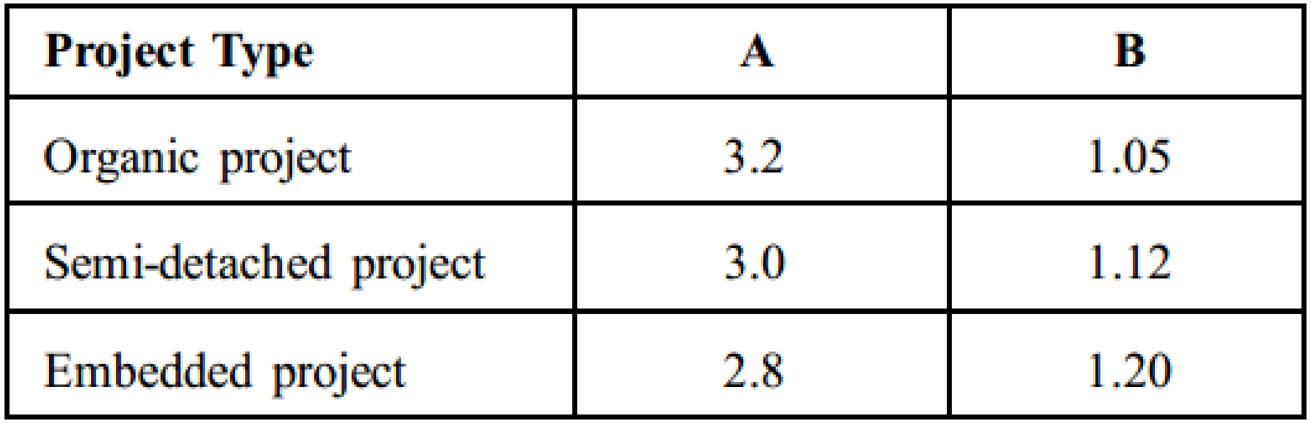
1. Basic Model:

- In basic model, only the size of project is considered while calculating effort.
- To calculate effort, use the following equation (known as effort equation):

E = A × (size)^B ....(i)

where E is the effort in person-months and size is measured in terms of KDLOC.

The values of constants ‘A’ and ‘B’ depend on the type of the software project. In this model, values of constants (‘A’ and ‘B’) for three different types of projects are listed in Table.



Project Type	A	B
Organic project	3.2	1.05
Semi-detached project	3.0	1.12
Embedded project	2.8	1.20

Example:

if the project is an organic project having a size of 30 KDLOC, then effort is calculated using equation,

E = 3.2× (30)^1.05

E = 114 Person-Month

2. Intermediate Model:

- ❖In intermediate model, parameters like software reliability and software complexity are also considered along with the size, while estimating effort.

- ❖To estimate total effort in this model, a number of steps are followed, which are listed below:
  - ✓Calculate an initial estimate of development effort by considering the size in terms of KDLOC.
  - ✓Identify a set of 15 parameters, which are derived from attributes of the current project. All these parameters are rated against a numeric value, called **multiplying factor**.
  - ✓Effort adjustment factor (EAF) is derived by multiplying all the multiplying factors with each other.

The COCOMO II Effort Equation:

Effort(Person-Month**) = 2.94(Initial calibration) * EAF * (KDLOC)^E**

Where,

EAF: **Effort Adjustment Factor** derived from the 15 Cost Drivers or multiplying factors (make assumption if value are not given in exam)

E: **Exponent derived from the five Scale Drivers** (make Assumption if value not given)

Example:

A project with all **Nominal Cost Drivers** and **Scale Drivers** would have an "EAF" of 1.00 and exponent "E" of 1.0997. Assuming that the project is projected to consist of 8,000 source lines of code.

Then by Using COCOMO II estimation

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

In the same example if effort multipliers are given then

If your project is rated Very **High for Complexity** (effort multiplier of 1.34), and **Low for Language & Tools Experience** (effort multiplier of 1.09), and **all of the other cost drivers are rated to be Nominal** (effort multiplier of 1.00) then,

Effort Adjustment Factor (EAF) = 1.34 * 1.09* 1 = 1.46
Effort = 2.94 * (1.46) * $(8)^{1.0997}$ = 42.3 Person-Months

COCOMO II Schedule Equation:

The COCOMO II schedule equation predicts the number of months required to complete your software project. It is predicted as:

Duration=3.67(Initial calibration) *(Effort)^{SE}

Where,

Effort: Effort from the COCOMO II effort equation.

SE: Schedule equation exponent derived from the five Scale Drivers

Example:

Continuing previous example and assuming the schedule equation exponent of 0.3179 that is calculated from the five scale drivers.

Duration=3.67*(42.3)$^{0.3179}$=12.1months

Average staffing = Effort/Duration

=(42.3 Person-Months) / (12.1 Months) = 3.5 people

Example:

If your project is rated Very High for Complexity (effort multiplier of 1.34), and Nominal for Language & Tools Experience (effort multiplier of 1.00), and all of the other cost drivers are rated to be Nominal (effort multiplier of 1.09) assuming that exponent “E” of 1.0997 and project is projected to consist of 10500 source lines of code and the schedule equation exponent of 0.419 that is calculated from the five scale drivers. Find the duration and average staffing.