CHAPTER 7: Managing object oriented software engineering (5 Hrs)

7.1 Project selection and preparation,
7.2 Project development, organization and management,
7.3 Software project planning and scheduling and techniques,
7.4 COCOMO model,
7.5 Risk management process,
7.6 Software quality assurance,
7.7 Software metrics

Project Selection and Preparation

1.1.1 Introducing a new development process
1.1.2 Selecting the first process
1.1.3 Education and training
1.1.4 Risk Analysis

Product Development Organization

- When developing a product, the basic organization of a project should be built around the product and the activities associated with the development of the product.
- Since the product (one hopes) will be developed in several versions, we must have the product life cycle in mind when discussing this topic.
- This means that all changes in the requirements specification should first be analyzed in the requirements model and inserted there.
- All these models must be maintained during the entire product life cycle; every further development should be accomplished by modifying these models.
- Product development with OOSE is built around different models that are developed in sequence.
- Hence all of these models must be up to date at all times during the product's life cycle.

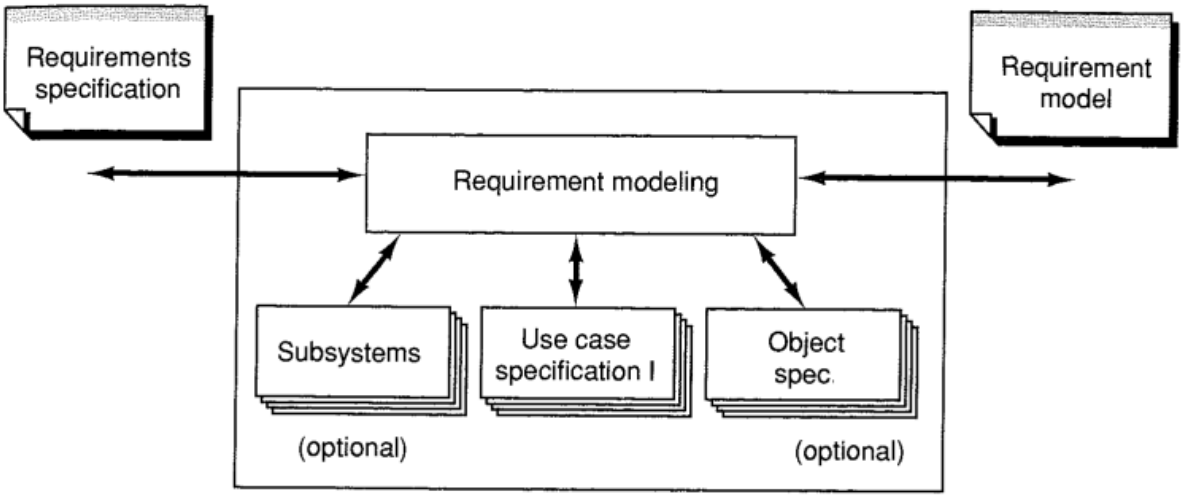


Fig: The process of requirement Analysis

- The first model to be developed in OOSE is requirement model.
- The requirements analysis process delivers a well-defined result: the requirement model with the use case specification.
- The identification of domain objects, actors and use cases is done in the coordination process while the specification of the use cases, objects and subsystems and their interfaces is done in the specification processes.

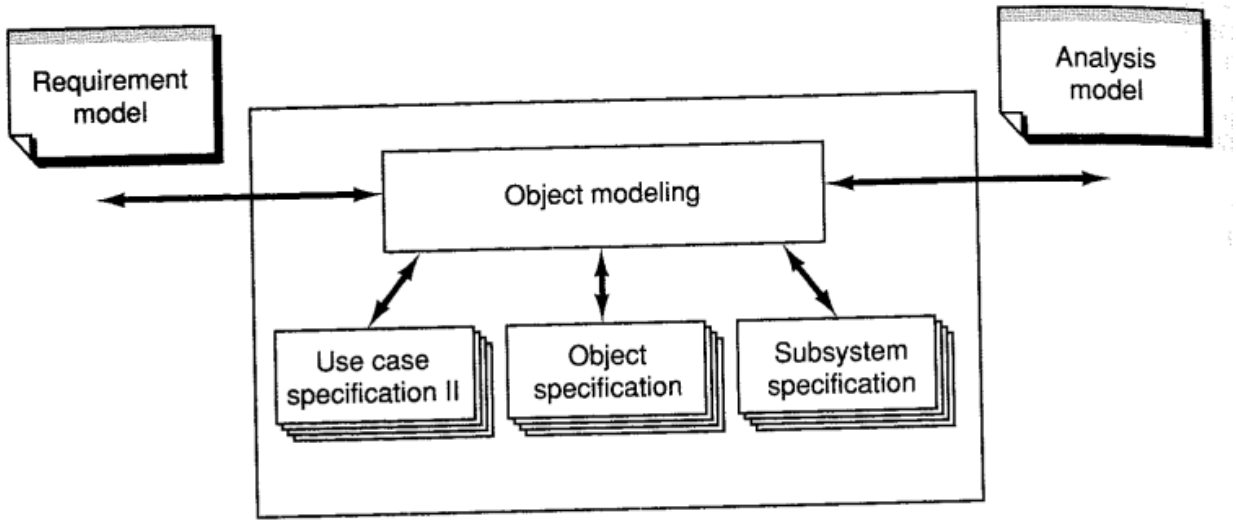


Fig: The process of robustness Analysis

- The next is analysis process which forms the well-defined result process model. Outcome of requirement analysis process acts as input to the object modeling for analysis process.
- The main process here coordinates three different kinds of activities: the identification of analysis objects from the use cases; the specification of each object; and the specification of each subsystem.

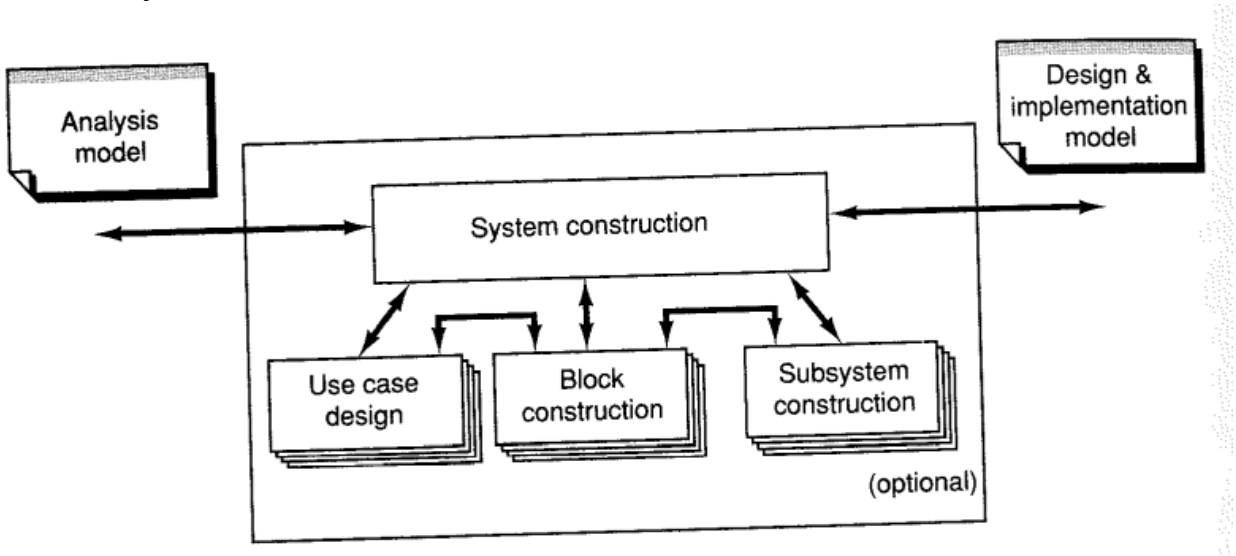


Fig: The process of Construction

- Analysis model is the input for the construction process. The construction process has three sub-process. They are use case design, block construction and subsystem construction.
- The first is the design of the use cases, where each use case is designed over the blocks.
- The second class of process is the block construction process where each block is designed, implemented and unit tested.
- The third is an optional subsystem construction process for a top-down approach.
- The well-defined result delivered by the construction process is the design model and the source code for the unit tested blocks.

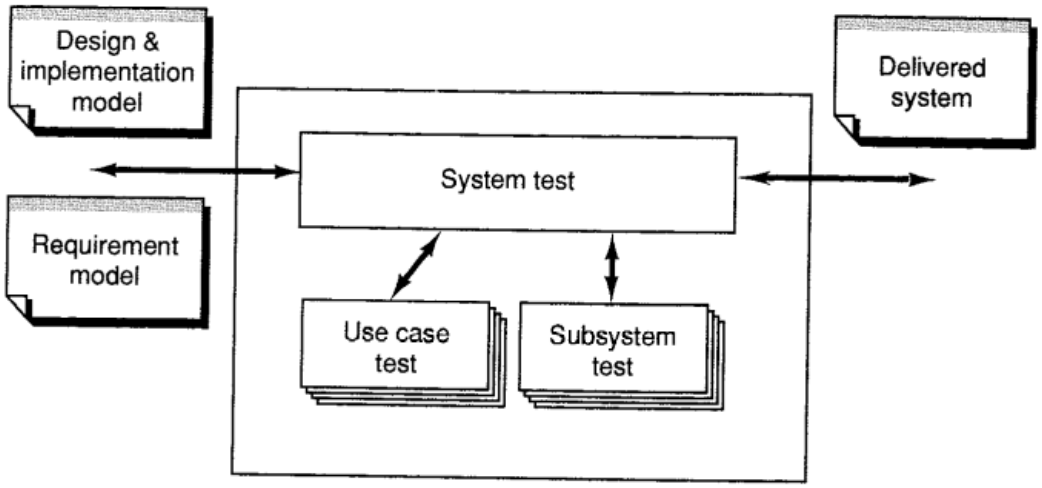


Fig: The Testing Process

- Next is testing process.
- In the testing process the system is integration tested and system block tested. Here the coordinating process is the system test itself.
- The integration testing is performed by two sub processes, use case tests and subsystem tests. In addition to these three main processes we also have the component process.

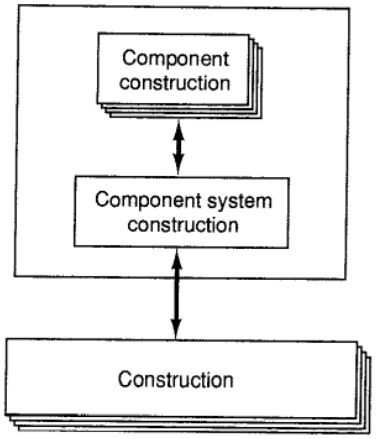


Fig: Component process interacts with several construction processes, one per product being developed

- The component system construction process includes the actual design, implementation and unit testing one for each component.

Project Organization and Management

- A necessary but not sufficient, condition for successful software development is good project management.
- A project is divided into numbers of milestones and the managerial and technical aspects must fit together to achieve this milestones.
- Milestones are concrete, objectively defined deliverable.
- Milestones are often combined with reviews with audits of the work done so far. This division aims to give better control of the projects.

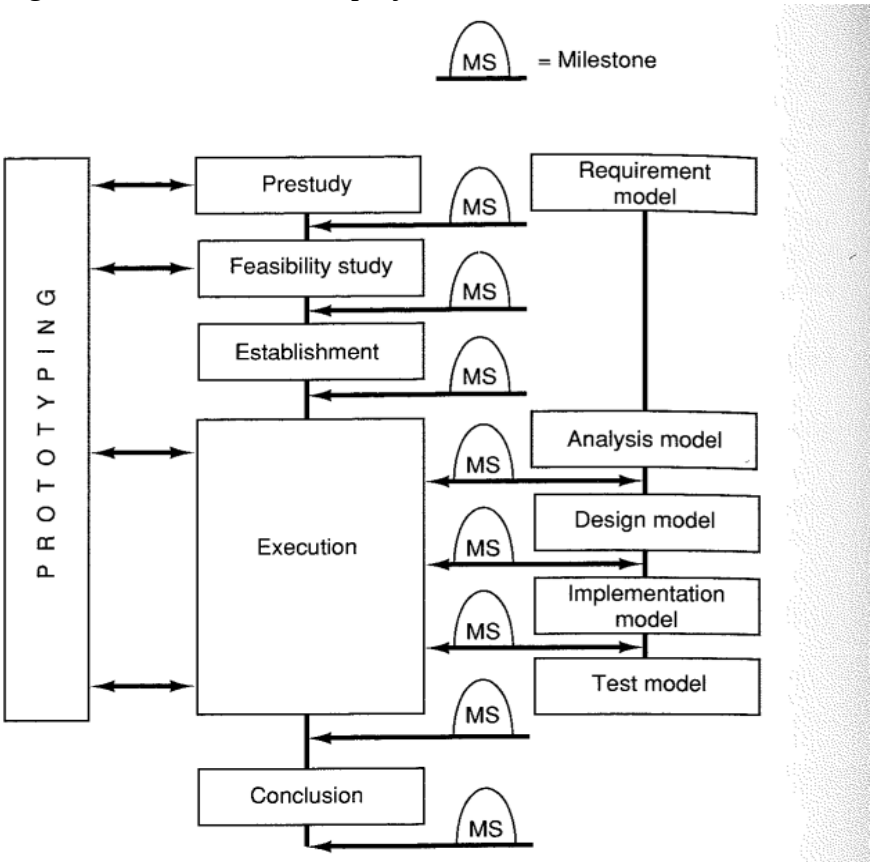


Figure 15.8 Prototyping may be used during many of the phases in the project The models in OOSE may be used as milestones that should be achieved.

- The models developed in OOSE are good for supporting the steering the project. All models have well defined result and it is appropriate to use them in combination with milestones. Hence the models can be matched onto project model.
- This ideal mapping gives a sense of an early waterfall model where the entire analysis model should be developed, reviewed and frozen before starting work on the design model.
- It is essential to realize, though, that the models will be modified when work is started on subsequent models.

Project management phases are:

i. **Pre-study:** It studies about if the project is practicable or not. It is done by defining and evaluating different kinds of requirements to judge projects technically and practically.

ii. **Feasibility Study:** It studies different technical alternatives and their consequences.

iii. **Establishment:** Detailed plans and resource plans are developed.

iv. **Execution:** The projects is developed in accordance with the plans previously prepared.

v. **Conclusion:** The project is completed and proposals to improve the project and development methods are summarized.

Project Staffing:

- One of the difficulties in software development lies in the staffing problems. Software development is an interdependent group task. A group of people with different knowledge and skills, which we call a software project team, work together to develop software.
- Accordingly, the project team influences the outcome of software development. Therefore, project staffing, that is, how to form software project teams, has persistently been a key question of software organizations.

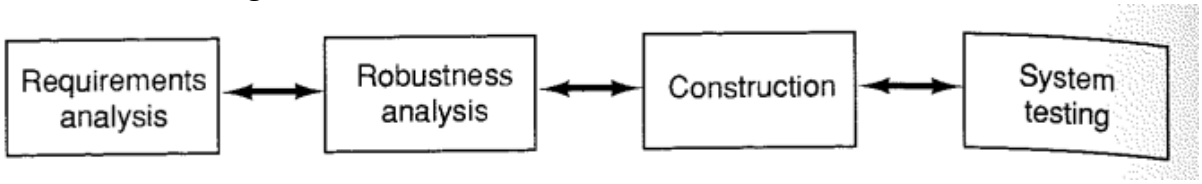


Fig: Coordination process in OOSE

Different development group can form for a projects:

System architecture group: These people are responsible for making system architecture and delivering coherent idea to the projects. They are core of development and are same for at least first three processes. i.e., requirement analysis, robustness analysis, construction. Project manager belongs to this groups.

Requirement analysis group: Initial requirement analysis is done by a small group with interaction with end users.

Development personnel group: More detail work should be done by development personnel who are skilled for their activity. It is good to have same person for the same group of objects in all activities. For example: a person specifying a particular use case should also offer object, use case design and implementation block for that use case. During construction phase more people can add to existing group or add new group to manage more people needed in this phase

Testing group: Testing is done in a separate phase often by separate group. Besides the actual development groups, there are other roles and groups in the projects:

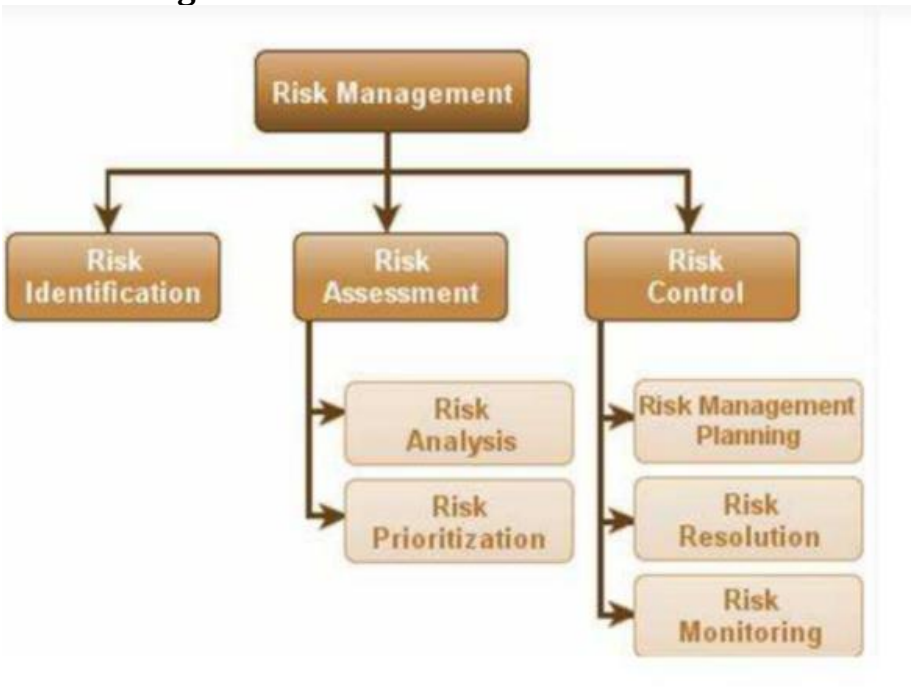
- **Methodologist:** The person responsible for the method used, language, operating system, product structure and development organization of the team where the new method should be introduced.

- **Quality Assurance:** People responsible for product and process to develop the product so that it is of high quality.
- **Documentation, manuals and training:** Developers (with special skills) are responsible for documentation of the system both for the maintenance and users.
- **Reuse coordinator:** Person or group responsible for encouraging and evaluating how much project is reusing and also investigating reuse potential of code and designs developed.
- **Staff:** Person responsible for helping project manager for following up cost and time schedules.
- **Prototyper:** This role is necessary to investigate different solutions at early stage to prepare for later development.
- **Support Environment:** This is to function as a service to the project as a whole. Typically system managers play this role.

Risk Management:

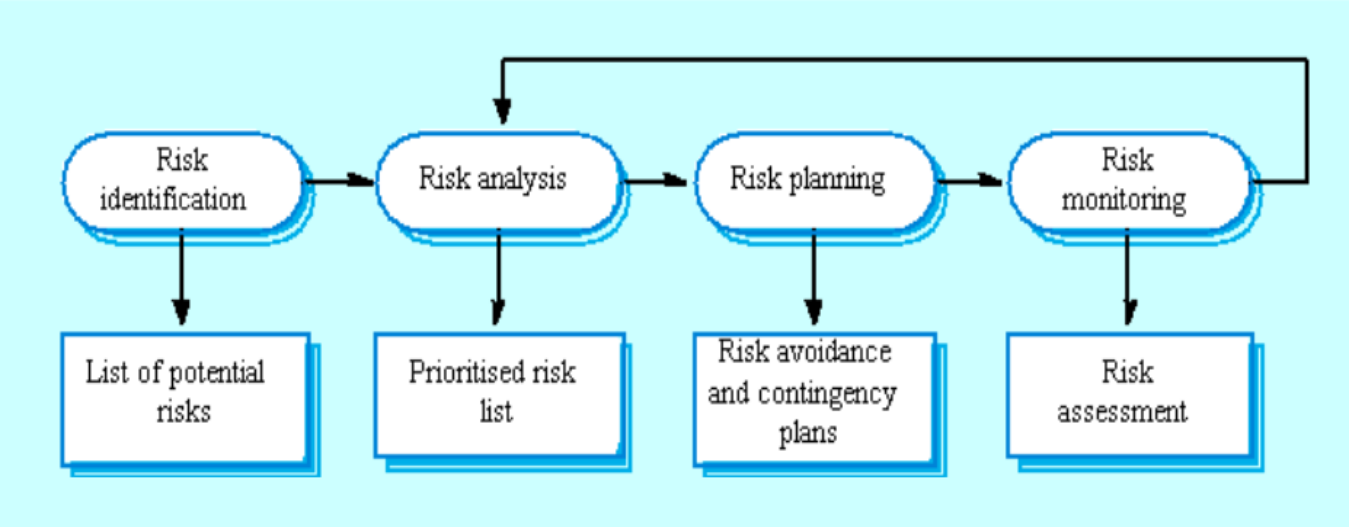
Risk management is concerned with identifying risks before and after development process and drawing up plans to minimize their effect on a project.

Risk Management Activities:



i. **Risk Identification**
ii. **Risk Analysis**
iii. **Risk Control**

Risk Management Process



- **Risk identification:** Identify project, product and business risks.
- **Risk analysis:** Assess the likelihood and consequences of these risks.
- **Risk planning**: Draw up plans to avoid or minimize the effects of the risks.
- **Risk monitoring:** Monitor the risks throughout the project.

Types of Risks

1. **Schedule Risk:**

- Project schedules get slipped when project tasks and schedule release risks are not addressed properly.
- Schedule risks mainly affect a project and finally on the company's economy and may lead to project failure.
- **Schedules often slip due to the following reasons:**
  - Wrong time estimation.
  - Resources are not tracked properly. All resources like staff, systems, skills of individuals, etc.
  - Failure to identify complex functionalities and time required to develop those functionalities.
  - Unexpected project scope expansions.

2. **Budget Risk:**

- Budget risk includes the following:
  - Wrong budget estimation.
  - Cost overruns
  - Project scope expansion

3. **Operational Risks**

- Risk of loss due to improper process implementation, failed system or some external event risks.

Causes of Operational Risks:

- Failure to address priority conflicts.
- Failure to resolve responsibilities.
- Insufficient resources
- No proper subject training.
- No resource planning
- No communication within the team.

4. **Technical Risks**

- Technical risks generally lead to failure of functionality and performance.

Causes of Technical Risks are:

- Continuously changing requirements
- No advanced technology is available or the existing technology is in the initial stages.
- The product is complex to implement.
- Difficult project module integration.

5. **Programmatic Risks**

- These are external risks beyond the operational limits. These are all uncertain risks that are outside the control of the program.

External events can be:

- Running out of funds.
- Market development
- Changing customer product strategy and priorities.
- Government rule changes.

Risk Management Principles

i. **Global Perspective:** In this, we review the bigger system description, design, and implementation. We look at the chance and the impact the risk is going to have.

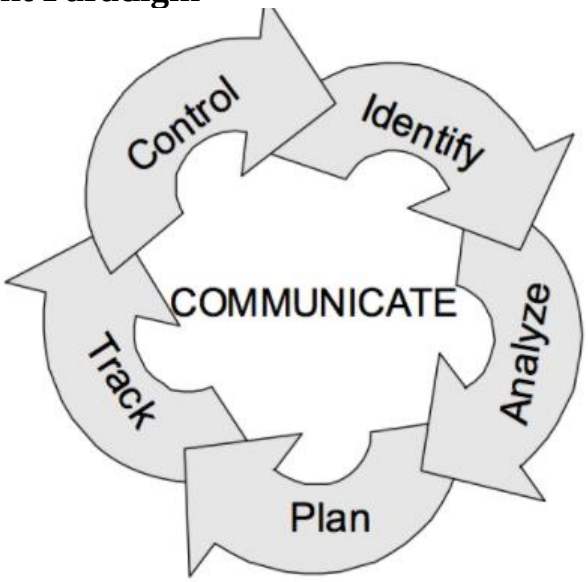
ii. **Take a forward-looking view:** Consider the threat which may appear in the future and create future plans for directing the next events.

iii. **Open Communication:** This is to allow the free flow of communications between the client and the team members so that they have certainty about the risks.

iv. **Integrated management:** In this method risk management is made an integral part of project management.

v. **Continuous process:** In this phase, the risks are tracked continuously throughout the risk management paradigm.

Risk Management Paradigm



i. **Identify:** In this phase, identification of risks is done to avoid future problems. Controlling risk is a far easier process than solving the problem.
ii. **Analyze:** In this phase, we need to analyze the risk to extract information on nature, behavior and type of risk. It extracts knowledge about the risk to extract better solutions.
iii. **Plan:** In this phase, the actual implementation of the plans. Here, we take actions and implementation of planning and plans are made and executed.
iv. **Track:** In this phase, we track actions that are required for removal and minimization of the risk.
v. **Control:** In this phase, we will check the risk management techniques and make necessary actions to make the risk management better.
vi. **Communicate:** the last phase of the risk management paradigm is to discuss all the risk management processes with the development and testing team.

Software Quality Assurance:

- Software quality assurance (SQA) aims at ensuring that the final product will have an acceptable quality.
- Software quality assurance is a planned and systematic plan of all actions necessary to provide adequate confidence that an item or product conforms to establish technical requirements.
- It is mainly a management activity to identify quality problems early in the development.
- Cost and time schedules are often tracked in the early stages, as opposed to quality, but quality problems appearing late in development involve large risks for any project. Therefore it is just as important as tracking cost and time to track the quality.
- Quality assurance focuses on both the product and the process.
- The product-oriented part of SQA (Software Quality Control) should strive to ensure that the software delivered has a minimum number of faults and satisfies the users' need.

- The process-oriented part (often called Software Quality Engineering) should institute and implement procedures, techniques and tools that promote the fault-free and efficient development of software products.
- The material to work with when doing SQA is mainly the documentation produced during development. No new documents should be needed for SQA. Therefore it is essential that everything important that is done should also be documented.
- The main tools for quality assurance are development process, reviews and audits, testing and metrics.
- To achieve good quality disciplines, and high quality awareness, an independent quality group responsible for quality assurance in the development department may be needed. This group can do reviews of how well the project members follow the given process of development and can also make an assessment of the project's possibilities of achieving its goals and illustrate potential risks.
- QA group should not function as policemen, but rather, together with the development team, increase the quality of what is being done.
- Some quality advice:
  • Follow the development process thoroughly, and note what goes wrong,
  • Eliminate the faults as quickly as possible by reviewing all specifications thoroughly,
  • See that the review groups have the right composition of people,
  • Note in the review protocols the number of pages reviewed and the number of faults of different types found,
  • Follow up the review protocols and identify, and possibly rewrite, extremely error prone objects. Try also to identify any individuals who seem to produce many defects in specifications or code,
  • Have an independent testing group testing the system and also writing the test report,
  • It is always cheapest to do it right the first time.

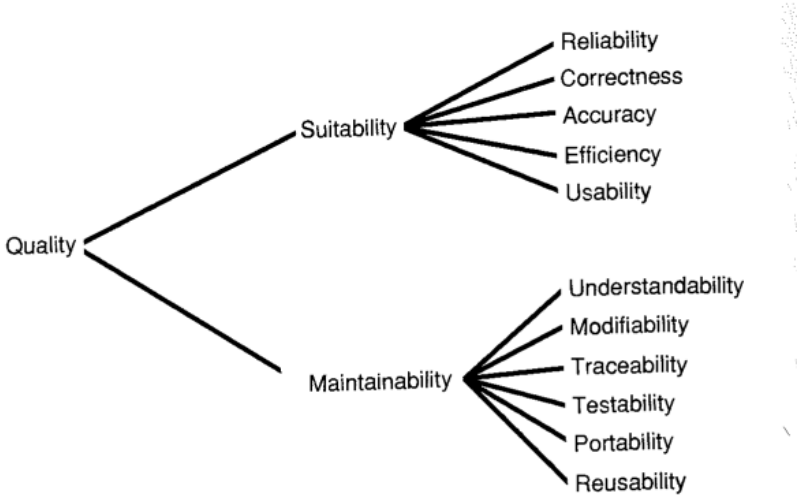


Fig: Characteristics of software product quality

Software metrics:

- The standard of measure for the estimation of quality, progress and health of the software testing effort is called software metrics.
- It can be divided into three groups:

i. Product metrics:

- The product characteristics like size, features of the design, complexity, performance, level of quality, etc., is described using product metrics.
- Product related metrics has not been demonstrated to be a useful quality predictor.

ii. Process metrics:

- Process-related metrics measures things like total development time, schedule time and no .of faults during testing.
- Example of process related metrics are:
  - Total development time,
  - Development time in each process and sub process
  - Cost for quality assurance.
- Process related metrics may form a basis for future planning.
- Software development and maintenance are improved using process metrics.

iii. Project metrics

- The project's characteristics and execution are described by project metrics whose examples include the count of software developers, cost, etc.

- Software metrics is a necessary method for controlling a development, which can measure either the process of development or various aspects of the products.
- Traditional metrics on products (including code) may to some extent be used also in object-oriented software. However the most common metric, lines of code, is actually even less interesting to measure for object-oriented software The less code you have written the more you have reused and that often (but not always!) gives your product a higher quality. The actual code metrics that are more appropriate for OOSE are:
  - Total no. of classes
  - Number of classes reused and the number newly developed.
  - Total number of operations.
  - Number of operation reused and the number newly developed etc.
- Some statistical metrics interesting to measure are:
  - Average number of operation in a class.
  - Length of operations.
  - Stimuli sent from each operation.
  - Average number of inherited operation.
- For instance, the McCabe cyclomatic complexity measures the complexity of graph such that it draws the sequence of program as a graph. **N= Connection – Nodes + 2**
- The complexity calculated as Connection-Nodes+2 will give a number denoting how complex a program (sequence) is.

- Too high McCabe number should be avoided since complexity will increase the possibility of errors. No module should have McCabe number higher than 10.

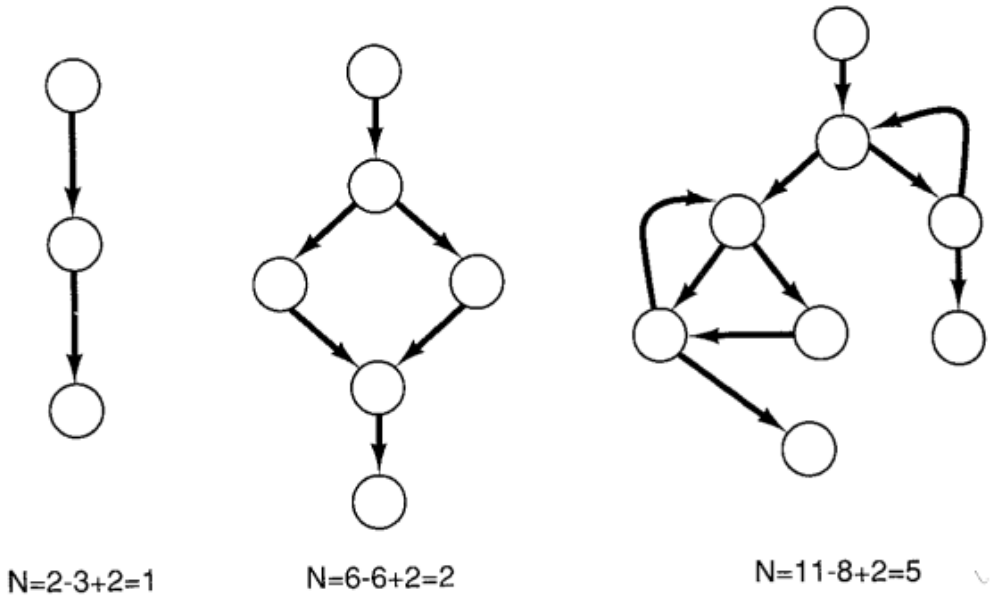


Fig: McCabe Cyclomatic Complexity N=Connection-Nodes+2

COCOMO Model:

- Constructive Cost Model
- B.W. Boehm Introduced COCOMO model in 1981.
- Based on a cost database of more than 60 different projects
- It estimates or predicts the effort required for the project, total project cost and scheduled time for the project.
- It estimates the cost for software product development in terms of effort (resources required to complete the project work) and schedule (time required to complete the project work) based on the size of the software product.
- It estimates the required number of Man-Months (MM) for the full development of software products.
- COCOMO is a hierarchy of cost estimation models.
- It includes three forms of COCOMO: basic, intermediate and detailed sub model.

DEVELOPMENT MODES:

i. Organic Mode:

- Relatively Small, Simple Software projects.
- Small teams with good application experience work to a set of less than rigid requirements.
- Similar to previously developed projects.
- Relatively small and require little innovation.
- Example: simple business systems, simple inventory management systems, and data processing systems.

ii. Semidetached Mode:

- Intermediate (in size and complexity) software projects in which teams with mixed experience levels must meet a mix of rigid and less than rigid requirements.
- Example: developing a new operating system (OS), a Database Management System (DBMS), and complex inventory management system.

iii. Embedded Mode:

- Software projects that must be developed within set of tight hardware, software and operational Constraints.
- Example : ATM, Air Traffic Control

Comparison of three COCOMO Modes

Mode	Project Size	Nature of Project	Innovation	Deadline of the Project	Development Environment
Organic	Typically 2 – 50 KLOC	Small Size Projects, experienced developers.	Little	Not tight	Familiar And In house
Semi-Detached	Typically 50 – 300 KLOC	Medium size project, average previous experience on similar projects.	Medium	Medium	Medium
Embedded	Typically over 300 KLOC	Large projects, complex interfaces, very little previous experience.	Significant	Tight	Complex Hardware / Customer interfaces required

According to Boehm, software cost estimation should be done through three stages:

1. Basic Model:

- The basic model aims at estimating, in a quick and rough fashion, most of the small to medium sized software projects.
- Depending on the problem at hand, the team might include a mixture of experienced and less experienced people with only a recent history of working together.
- It mainly deals with the number of lines of code and the level of estimation accuracy is less as we don't consider the all parameters belongs to the project.
- The estimated effort and scheduled time for the project are given by the relation:

Effort (E) = a*(KLOC)^b MM
Scheduled Time (D) = c*(E)^d Months(M)
SS = E/D persons
P = KLOC/E
TDEV = cE^d

E= effort applied in terms of person months
D = scheduled time
SS = staff size
P = productivity
a, b, c, d = Coefficients
TDEV= Development Time
KLOC= Kilo Lines of Code

Basic COCOMO Coefficients

Project	a_b	b_b	c_b	d_b
Organic mode	2.4	1.05	2.5	0.38
Semidetached mode	3.0	1.12	2.5	0.35
Embedded mode	3.6	1.20	2.5	0.32

Example 1: Suppose that a project was estimated to be 400 KLOC. Calculate the effort and development time for each of the three modes i.e. organic, semidetached and embedded.

Solution:

The basic COCOMO equations take the form: $E = a\,(KLOC)^b$

$D = c\,(E)^d$

Estimated size of the project = 400 KLOC

1. **Organic Mode**

$E = 2.4\,(400)^{1.05} = 1295.31$ PM

$D = 2.5\,(1295.31)^{0.38} = 38.07$ M

2. **Semi-detached Mode**

$E = 3.0\,(400)^{1.12} = 2462.79$ PM

$D = 2.5\,(2462.79)^{0.35} = 38.45$ M

3. **Embedded Mode**

$E = 3.6\,(400)^{1.20} = 4772.81$ PM

$D = 2.5\,(4772.81)^{0.32} = 37.59$ M

Example 2: Consider a software project using semi-detached mode with 30000 lines of code. We will obtain estimation for this project as follows:

Solution:

1. Effort estimation

$E = a (KLOC)^{b}$ person-months

$E=3.0(30)^{1.12}$ where lines of code=30000=30 KLOC

E=135 person-month

2. Duration estimation

$D = c (E)^{d}$ months$=2.5(135)^{0.35}$

D=14 months

3. Person estimation

SS=E/D=135/14

SS=10 persons approx.

Example 3: We have determined our project fits the characteristics of Semi-Detached mode & we estimate our project will have 32,000 Delivered Source Instructions.

Solution:

Effort = $3.0*(32)^{1.12}$ = 146 man-months

Duration = $2.5*(146)^{0.35}$ = 14 months

Productivity = 32,000 DSI / 146 MM = 219 DSI/MM

Person estimation = 146 MM /14 months = 10 FSP

Example 4: Assume that the size of an organic type software product has been estimated to be 32,000 lines of source code. Assume that the average salary of software engineers be Rs. 15,000/- per month. Determine the effort required to develop the software product and the nominal development time.

Solution: As per the basic COCOMO estimation formula for organic software:

Effort = $2.4 * (32)^{1.05}$ = 91 PM

Nominal development time = $2.5 * (91)^{0.38}$ = 14 months

Cost required to develop the product = 14 * 15,000= Rs. 210,000/-

Merits of Basic COCOMO model:

- Basic COCOMO model is good for quick, early, rough order of magnitude estimates of software project.

Limitations:

- The accuracy of this model is limited because it does not consider certain factors for cost estimation of software. These factors are hardware constraints, personal quality and experiences, modern techniques and tools.
- The estimates of COCOMO model are within a factor of 1.3 only 29% of the time and within the factor of 2 only 60% of time.

2. Intermediate Model:

- In the Intermediate model Boehm introduced an additional set of 15 predictors called cost drivers in the intermediate model to take account of the software development

environment. Cost drivers are used to adjust the nominal cost of a project to the actual project environment to increase the accuracy of the estimate.

- The cost drivers are grouped into 4 categories:-
  1. Product attributes
     a. Required software reliability (RELY)
     b. Database size (DATA)
     c. Product complexity (CPLX)
  2. Computer attributes
     a. Execution time constraint (TIME)
     b. Main store constraint (STOR)
     c. Virtual machine volatility (VIRT)
     d. Computer turnaround time (TURN)
  3. Personnel attributes
     a. Analyst capability (ACAP)
     b. Application experience (AEXP)
     c. Programmer capability (PCAP)
     d. Virtual machine experience (VEXP)
     e. Programming Language experience (LEXP)
  4. Project attributes
     a. Modern programming practices (MODP)
     b. Use of software tool (TOOL)
     c. Required development schedule (SCED)
- It produces better results than the Basic model because the user supplies settings for cost drivers that determine the effort and duration of the software projects.
- The intermediate COCOMO equation takes the form:

  $E = a\,(KLOC)^b * EAF$

  $D = c(E)^d$

  SS = E/D persons, P = KLOC/E

EAF = EffortAdjustment factor
E = effort
D = Deployment time
SS = staff size
P = productivity
a, b, c, d = Coefficients

Co-efficients for Intermediate COCOMO

Project	a_i	b_i	c_i	d_i
Organic mode	3.2	1.05	2.5	0.38
Semidetached mode	3.0	1.12	2.5	0.35
Embedded mode	2.8	1.20	2.5	0.32

Multiplier values for Effort Calculations

Cost Drivers	Ratings					
	Very Low	Low	Nominal	High	Very High	Extra High
Product attributes						
RELY	0.75	0.88	1.00	1.15	1.40	-
DATA	-	0.94	1.00	1.08	1.16	-
CPLX	0.70	0.85	1.00	1.15	1.30	1.65
Computer attributes						
TIME	-	-	1.00	1.11	1.30	1.66
STOR	-	-	1.00	1.06	1.21	1.56
VIRT	-	0.87	1.00	1.15	1.30	-
TURN	-	0.87	1.00	1.07	1.15	-
Personnel attributes						
ACAP	1.46	1.19	1.00	0.86	0.71	-
AEXP	1.29	1.13	1.00	0.91	0.82	-
PCAP	1.42	1.17	1.00	0.86	0.70	-
VEXP	1.21	1.10	1.00	0.90	-	-
LEXP	1.14	1.07	1.00	0.95	-	-
Project attributes						
MODP	1.24	1.10	1.00	0.91	0.82	-
TOOL	1.24	1.10	1.00	0.91	0.83	-
SCED	1.23	1.08	1.00	1.04	1.10	-

Example 1: A new project with estimated 400 KLOC embedded system has to be developed. Project manager has a choice of hiring from two pools of developers : with very high application experience and very little experience in the programming language being used or developers of very low application experience but a lot of experience with the programming language. What is the impact of hiring all developers from one or the other pool?

Solution:

This is the case of embedded mode:

Hence $E = a(KLOC)^b * EAF$

$D = c(E)^d$

Case 1: Developers are with very high application experience and very little experience in the programming language being used.

$EAF = 0.82 * 1.14 = 0.9348$

$E = 2.8(400)^{1.20} * 0.9348 = 3470$ PM

$D = 2.5(3470)^{0.32} = 33.9$ M

Case 2: developers of very low application experience but a lot of experience with the programming language.

EAF = 1.29*0.95 = 1.22
E = 2.8 $(400)^{1.20}$ *1.22 = 4528PM
D = 2.5 $(4528)^{0.32}$ = 36.9 M
Case 2 requires more effort and time. Hence, low quality application experience but a lot of programming language experience could not match with the very high application experience and very little programming language experience.

Example 2: For a given project was estimated with a size of 300 KLOC. Calculate the Effort, Scheduled time for development by considering developer having high application experience and very low experience in programming.

Solution:

Given the estimated size of the project is: 300 KLOC
Developer having highly application experience: 0.82 (as per above table)
Developer having very low experience in programming: 1.14(as per above table)
EAF = 0.82*1.14 = 0.9348
Effort (E) = a*$(KLOC)^b$ *EAF = 3.0*$(300)^{1.12}$ *0.9348 = 1668.07 MM
Scheduled Time (D) = c*$(E)^d$ = 2.5*$(1668.07)^{0.35}$ = 33.55 Months(M)

3. Detailed Model:

- It is the advanced model that estimates the software development effort like Intermediate COCOMO in each stage of the software development life cycle process.
- It incorporates all qualities of the standard version with an assessment of the cost drivers effect on each method of the software engineering process.
- In it, the whole software is differentiated into multiple modules, and then we apply COCOMO in various modules to estimate effort and then sum the effort.

Advantages and Disadvantages of COCOMO Model

Advantages

- Easy to estimate the total cost of the project.
- Easy to implement with various factors.
- Provide ideas about historical projects.

Disadvantages

- It ignores requirements, customer skills, and hardware issues.
- It limits the accuracy of the software costs.
- It mostly depends on time factors.