UNIT-V

➢ Overview of Product Metrics

Software engineers use product metrics to help them assess the quality of the design and construction of the software product being built. Product metrics provide software engineers with a basis to conduct analysis, design, coding, and testing more objectively. Qualitative criteria for assessing software quality are not always sufficient by themselves. The process of using product metrics begins by deriving the software measures and metrics that are appropriate for the software representation under consideration. Thendata are collected and metrics are computed. The metrics are computed and compared to pre-established guidelines and historical data. The results of these comparisons are used to guide modifications made to work products arising from analysis, design, coding, or testing.

➢ Software Quality

Software Quality Principles - a Qualitative View

Conformance to software requirements is the foundation from which quality is measured.

Specified standards define a set of development criteria that guide the manner in which software is engineered.

Software quality is suspect when a software product conforms to its explicitly stated requirementsand fails to conform to the customer's implicit requirements (e.g., ease of use).

McCall's Quality Factors

The factors that affect software quality can be categorized in two broad groups.

Factors that can be directly measured (e.g., defects uncovered during testing)

Factors that can be measured only indirectly (e.g., usability or maintainability)The factors that affect software quality are shown below.

- Correctness: The extent to which a program satisfies its specification and fulfills the customer'smission objectives.
- Reliability: The extent to which a program can be expected to perform its intended function with required precision.
- Efficiency: The amount of computing resources and code required by a program to perform itsfunction.
- Integrity: The extent to which access to software or data by unauthorized persons can be controlled.
- Usability: The effort required to learn, operate, prepare input for, and interpret output of a program.
- Maintainability: The effort required to locate and fix an error in a program.
- Flexibility: The effort required to modify an operational program.
- Testability: The effort required to test a program to ensure that it performs its intended function.
- Portability: The effort required to transfer the program from one hardware and/or software system environment to another.
- Reusability: The extent to which a program can be reused in other applications-related to thepackaging and scope of the functions that the program performs.
- Interoperability: The effort required to couple one system to another.

ISO 9126 Quality Factors

- Functionality
- Reliability
- Usability
- Efficiency
- Maintainability
- Portability

➢ A Framework for Product Metrics

Benefits of Product Metrics

1. Assist in the evaluation of the analysis and evaluation model
2. Provide indication of procedural design complexity and source code complexity
3. Facilitate design of more effective testing

Measurement Principles

Measurement Process Activities

- Formulation - derivation of software measures and metrics appropriate for software representation being considered
- Collection - mechanism used to accumulate the date used to derive the software metrics
- Analysis - computation of metrics
- Interpretation - evaluation of metrics that results in gaining insight into quality of the work product
- Feedback - recommendations derived from interpretation of the metrics is transmitted to the software development team

Measurement Principles

- The objectives of measurement should be established before collecting any data.
- Each product metric should be defined in an unambiguous manner.
- Metrics should be derived based on a theory that is valid for the application domain.
- Metrics should be tailored to accommodate specific products and processes.

Metrics Characterization and Validation Principles

- A metric should have desirable mathematical properties
- The value of a metric should increase when positive software traits occur or decrease whenundesirable software traits are encountered
- Each metric should be validated empirically in several contexts before it is used to make decisions

Measurement Collection and Analysis Principles

1. Automate data collection and analysis whenever possible
2. Use valid statistical techniques to establish relationships between internal product attributes andexternal quality characteristics
3. Establish interpretive guidelines and recommendations for each metric

Goal-Oriented Software Measurement (GQM)

- GQM emphasizes the need

1. establish explicit measurement goal specific to the process activity or product characteristic being assessed
2. define a set of questions that must be answered in order to achieve the goal
3. identify well-formulated metrics that help to answer these questions

- A goal definition template can be used to define each measurement goal

Attributes of Effective Software Metrics

- Simple and computable
- Empirically and intuitively persuasive
- Consistent and objective
- Consistent in use of units and measures
- Programming language independent
- Provides an effective mechanism for quality feedback

Important Metrics Areas

- Metrics for the Analysis Model
  - Functionality delivered-provides an indirect measure of the functionality that is packaged within the software.
  - System size –measures of the overall size of the system defined in terms of information available as part of the analysis model.
  - Specification quality-provides an indication of the specificity and completeness of a requirements specification.
- Metrics for the Design Model
  - Architecture metrics –provide an indication of the quality of the architectural design.
  - Component-level metrics –measure the complexity of software components and other characteristics that have a bearing on quality.
  - Specialized OO Design Metrics-measure characteristics of classes and their communication and collaboration characteristics.

- Metrics for Source Code
  - Halstead metrics –controversial but nonetheless fascinating, these metrics provide uniquemeasures of a computer program.
  - Complexity metrics –measure the logical complexity of source code.
  - Length metrics-provide an indication of the size of the software.
- Metrics for Testing
  - Statement and branch coverage metrics –lead to the design of test cases that provide programcoverage.
  - Defect-related metrics –focus on bugs found, rather than on the tests themselves.
  - Testing effectiveness –provide a real-time indication of the effectiveness of tests that have been conducted.
  - In-process metrics-process related metrics that can be determined as testing is conducted.

➢ Metrics for the Analysis Model

- Function-based metrics

  The function point metric (FP) can be used effectively as a means for measuring the functionality delivered by a system. Using historical data, the FP can then be used to

  ❖ Estimate the cost or effort required to design, code and test the software.
  ❖ Predict the number of errors that will be encountered during testing.
  ❖ Forecast the number of components and/or the number of projected source lines in the implemented system.

  Functions points are derived from the following information domain values.

  ▪ Number of external inputs (EIs)
  ▪ Number of external outputs (EOs)
  ▪ Number of external inquiries (EQs)
  ▪ Number of internal logical files (ILFs)
  ▪ Number of external interface files (EIFs)

  To compute function points (FP), the following relationship is used: FP=count total* [0.65+0.01+∑ (Fi)]

  Where count total is the sum of all FP entries and the Fi (i=1 to 14) are value adjustment factors(VAF) based on responses to the following questions.

4. Does the system require reliable backup and recovery?
5. Are specialized data communications required to transfer information to or from the application?
6. Are there distributed processing functions?
7. Is performance critical?
8. Will the system run in an existing, heavily utilized operational environment?
9. Does the system require on-line data entry?
10. Does the on-line data entry require the input transaction to be built over multiple screens oroperations?
11. Are the ILFs updated on-line?
12. Are the inputs, outputs, files, or inquiries complex?
13. Is the internal processing complex?
14. Is the code designed to be reusable?
15. Are conversion and installation included in the design?
16. Is the system designed for multiple installations in different organizations?
17. Is the application designed to facilitate change and for ease of use by the user?

Each of these questions is answered using a scale that ranges from 0 (not important or applicable) to 5 (absolutely essential)

- Specification quality metrics (Davis)

  Davis and his colleagues propose a list of characteristics that can be used to assess the quality of the analysis model and the corresponding requirements specification: specificity (lack of ambiguity), completeness, correctness, understandability, verifiability, internal and external consistency, achievability, concision, traceability, modifiability, precision, and reusability.

➢ Metrics for the Design Model

- Architectural design metrics

- Structural complexity (based on module fanout): structural complexity of a module i is defined in the following manner S(i)=f
- Where fis the fan-out of module i. Fan-out is defined as the number f modulesimmediately subordinate to the module i.
  - Data complexity (based on module interface inputs and outputs): provides an indication of the complexity in the internal interface for a module I and is defined as D(i)=v(i)/[f(i)+1].
  - Where v(i) is the number of input and output variables that are passed to and from module i.
- System complexity (sum of structural and data complexity)C(i)=S(i)+D(i)
- Morphology (number of nodes and arcs in program graph): defined assize =n+a
  where n is the number of nodes and a is the number of arcs.
- Design structure quality index (DSQI): the DSQI is computed in the following manner
  DSQI=∑wiDi
  Where i=1 to 6 and Wi is the relative weighting of the importance of each of the intermediate values, and ∑Wi=1.

OO design metrics

- Size (defined in terms of four views: population, volume, length, functionality. Population ismeasured by taking a static count of OO entities such as classes or operations)
- Complexity (how classes interrelate to one another)
- Coupling (physical connections between design elements of the OO design)
- Sufficiency (how well design components reflect all properties of the problem domain)
- Completeness (coverage of all parts of problem domain)
- Cohesion (manner in which all operations work together)
- Primitiveness (is the degree to which attributes and operations are atomic)
- Similarity (degree to which two or more classes are alike in terms of their structure, function,behavior or purpose.)
- Volatility (likelihood a design component will change)

➢ Metrics for Coding

Halstead proposed the first analytic laws for computer science by using a set of primitive measures, which can be derived once the design phase is complete and code is generated. These measures are listed below.

n_1 = number of distinct operators in a program
n_2 = number of distinct operands in a program
N_1 = total number of operators
N_2= total number of operands.

By using these measures, Halstead developed an expression for overall program length, program volume, program difficulty, development effort, and so on.

Program length (N) can be calculated by using the following equation.

$N = n_1\log_2 n_1 + n_2 \log_2 n_2$.

Program volume (V) can be calculated by using the following equation.

$V = N \log_2 (n_1+n_2)$.

Note that program volume depends on the programming language used and represents the volume of information (in bits) required to specify a program. Volume ratio (L)can be calculated by using the following equation.

$$L = \frac{\text{Volume of the most compact form of a program}}{\text{Volume of the actual program}}$$

Where, value of L must be less than 1. Volume ratio can also be calculated by using the following equation.

$L = (2/n_1)* (n_2/N_2)$.

Program difficulty level (D) and effort (E)can be calculated by using the following equations.

$D = (n_1/2)*(N_2/n_2)$.

$E = D * V$.

➢ Metrics for Testing

Majority of the metrics used for testing focus on testing process rather than the technical characteristics of test. Generally, testers use metrics for analysis, design, and coding to guide them in design and execution of test cases.

Function point can be effectively used to estimate testing effort. Various characteristics like errors discovered, number of test cases needed, testing effort, and so on can be determined by estimating the number of function points in the current project and comparing them with any previous project.

Metrics used for architectural design can be used to indicate how integration testing can be carried out. In addition, cyclomatic complexity can be used effectively as a metric in the basis-path testing to determine the number of test cases needed.

Halstead Metrics Applied to Testing:

Halstead measures can be used to derive metrics for testing effort. By using program volume (V) and program level (PL), Halstead effort (e) can be calculated by the following equations.

e = V/ PL

Where

$PL = 1/ [(n_1/2) * (N_2/n_2)]$ … (1)

For a particular module (z), the percentage of overall testing effort allocated can be calculated by the following equation.

Percentage of testing effort (z) = $e(z)/\sum e(i)$

Where, e(z) is calculated for module z with the help of equation (1). Summation in the denominator is the sum of Halstead effort (e) in all the modules of the system.

Metrices for Object –Oriented Testing:

For developing metrics for object-oriented (OO) testing, different types of design metrics that have a direct impact on the testability of object-oriented system are considered. While developing metrics for OO testing, inheritance and encapsulation are also considered. A set of metrics proposed for OO testing is listed below.

- **Lack of cohesion in methods (LCOM):** This indicates the number of states to be tested. LCOM indicates the number of methods that access one or more same attributes. The value of LCOM is 0, if no methods access the same attributes. As the value of LCOM increases, more states need to be tested.
- **Percent public and protected (PAP):** This shows the number of class attributes, which are public or protected. Probability of adverse effects among classes increases with increase in value of PAP as public and protected attributes lead to potentially higher coupling.
- **Public access to data members (PAD):** This shows the number of classes that can access attributes of another class. Adverse effects among classes increase as the value of PAD increases.
- **Number of root classes (NOR):** This specifies the number of different class hierarchies, which are described in the design model. Testing effort increases with increase in NOR.
- **Fan-in (FIN):** This indicates multiple inheritances. If value of FIN is greater than 1, it indicates that the class inherits its attributes and operations from many root classes. Note that this situation (where FIN> 1) should be avoided.

➢ Overview Of Managing Software Projects

- Project management involves the planning, monitoring, and control of people, process, and events that occur during software development.
- Everyone manages, but the scope of each person's management activities varies according his or her role in the project.
- Software needs to be managed because it is a complex, long duration undertaking.
- Managers must focus on the fours P's to be successful (people, product, process, and project).
- A project plan is a document that defines the four P's in such a way as to ensure a cost effective, high quality software product.
- The only way to be sure that a project plan worked correctly is by observing that a high quality product was delivered on time and under budget.

Management Spectrum

- People: The people management maturity model defines the following key practice areas for software people: recruiting, selection, performance management, training, compensation, career development, organization, work design, team/culture development.
- Product: Before a project can be planned, product objectives and scope should be established, alternative solutions should be considered and technical and management constraints should be identified.
- Process: A software process provides the framework activities populated with tasks, milestones, work products, and quality assurance points.
- Project (planning, monitoring, controlling)

People

The software process is populated by stakeholders who can be categorized into one of five constituencies:

1. Senior managers: who define business issues that often have significant influence on the project.
2. Project (technical) managers: who must plan, motivate, organize, and control practitioners who do software work.
3. Practitioners: who deliver the technical skills that are necessary to engineer a product or application.
4. Customers: who specify the requirements for the software to be engineered and other stakeholders who have a peripheral interest in the outcome
5. End-users: who interact with the software once it is released for production use.

Team leadership model: Project management is a people-intensive activity.

- Motivation: The ability to encourage (by "push or pull") technical people to produce to their best ability.
- Organization: The ability to mold existing processes that will enable the initial concept to be translated into a final product.
- Ideas or innovation: The ability to encourage people to create and feel creative even when they must work within bounds established for a particular software product or application.

Characteristics of effective project managers (problem solving, managerial identity, achievement, influenceand team building)

Factors Affecting Team Organization

- Difficulty of problem to be solved
- Size of resulting program
- Team lifetime
- Degree to which problem can be modularized
- Required quality and reliability of the system to be built
- Rigidity of the delivery date
- Degree of communication required for the project

Team Organizational Paradigms

- Closed paradigm (structures a team along a traditional hierarchy of authority. Top level problem solving and internal coordination managed by team leader, good for projects that repeat past efforts)
- Random paradigm (structures a team loosely and success depends on initiative of individual team members, paradigm excels when innovation and technical breakthroughs required)
- Open paradigm (work is performed collaboratively. Rotating task coordinators and group consensus, good for solving complex problems - not always efficient as other paradigms)
- Synchronous paradigm (relies on natural problem compartmentalization and team organized to require little active communication with each other)

Toxic Team Environment Characteristics

1. Frenzied work atmosphere where team members waste energy and lose focus on work objectives
2. High frustration and group friction caused by personal, business, or technological problems
3. Fragmented or poorly coordinated procedures or improperly chosen process model blocks accomplishments

4. Unclear role definition that results in lack of accountability or finger pointing
5. Repeated exposure to failure that leads to loss of confidence and lower morale

Agile Teams

- Teams have significant autonomy to make their own project management and technical decisions
- Planning kept to minimum and is constrained only by business requirements and organizational standards
- Team self-organizes as project proceeds to maximize contributions of each individual's talents
- May conduct daily (10 - 20 minute) meeting to synchronized and coordinate each day's work
  - What has been accomplished since the last meeting?
  - What needs to be accomplished by the next meeting?
  - How will each team member contribute to accomplishing what needs to be done?
  - What roadblocks exist that have to be overcome?

Coordination and Communication Issues

- Formal, impersonal approaches (e.g., documents, milestones, memos)
- Formal interpersonal approaches (e.g., review meetings, inspections)
- Informal interpersonal approaches (e.g., information meetings, problem solving)
- Electronic communication (e.g., e-mail, bulletin boards, video conferencing)
- Interpersonal networking (e.g., informal discussion with people other than project team members)

The Product

- Software scope (context, information objectives, function, and performance)
- Problem decomposition (partitioning or problem elaboration - focus is on functionality to bedelivered and the process used to deliver it)

The Process

- Process model chosen must be appropriate for the: customers and developers, characteristics of the product, and project development environment
- Project planning begins with melding the product and the process
- Each function to be engineered must pass through the set of framework activities defined for asoftware organization
- Work tasks may vary but the common process framework (CPF) is invariant (project size does not change the CPF)
- The job of the software engineer is to estimate the resources required to move each function throughthe framework activities to produce each work product
- Project decomposition begins when the project manager tries to determine how to accomplish eachCPF activity

The Project

Signs of Potential Project Failure

1. Developers do not understand customer's needs
2. Product scope poorly defined
3. Changes poorly managed
4. Chosen technology changes
5. Business needs change or ill-defined
6. Deadlines unrealistic
7. Users are resistant
8. Sponsorship lost or never obtained
9. Project team members lack appropriate skills
10. Managers and practitioners avoid best practices and lessons learned

Avoiding Project Failure

1. Start on the right foot: This is accomplished by working hard to understand the problem that is to be solved and then setting realistic objectives and expectations for everyone who will be involved in the project.
2. Maintain momentum: Many projects get off to a good start and then slowly designate. To maintain

momentum, the project manager must provide incentives to keep turnover of personnel to an absolute minimum; the team should emphasize quality in every task it performs.

3. Track progress: For software project, progress is tracked as work products (e.g., models, source code, sets of test cases) are produced and approved as part of a quality assurance activity.
4. Make smart decisions: In essence, the decisions of the project manager and the software team should be to "keep it simple".
5. Conduct a postmortem analysis: Evaluate the planned and actual schedules, collect and analyze software project metrics, get feedback from team members and customers, and record findings in written form.

W5HH Principle

- Why is the system being developed?: The answer to this question enables all parties to assess the validity of business reasons for the software work.
- What will be done?: The answer to this question establishes the task set that will be required for the project.
- When will it be accomplished?: The answer to this question helps the team to establish a project schedule by identifying when project tasks are to be conducted and when milestones are to be reached.
- Who is responsible for a function?: The role and responsibility of each member of the software team must be defined.
- Where they are organizationally located?: Not all roles and responsibilities reside within the software team itself. The customer, users, and other stakeholders also have responsibilities.
- How will the job be done technically and managerially?: Once the product scope is established, a management and technical strategy for the project must be defined.
- How much of each resource is needed?: The answer to this question is derived by developingestimates based on answers to earlier questions.

Critical Practices

- Formal risk management
- Empirical cost and schedule estimation
- Metric-based project management
- Earned value tracking
- Defect tracking against quality targets
- People-aware program management

➢ Overview of Metrics in process

Software process and project metrics are quantitative measures that enable software engineers to gain insight into the efficiency of the software process and the projects conducted using the process framework. In software project management, we are primarily concerned with productivity and quality metrics. There are four reasons for measuring software processes, products, and resources (to characterize, to evaluate, to predict, and to improve).

Measures and Metrics

- Measure - provides a quantitative indication of the size of some product or process attribute
- Measurement - is the act of obtaining a measure
- Metric - is a quantitative measure of the degree to which a system, component, or process possesses a given attribute

Process Indicators

- Metrics should be collected so that process and product indicators can be ascertained
- Process indicators enable software project managers to: assess project status, track potential risks,detect problem area early, adjust workflow or tasks, and evaluate team ability to control product quality

Process Metrics

- Private process metrics (e.g., defect rates by individual or module) are only known to by theindividual or team concerned.
- Public process metrics enable organizations to make strategic changes to improve the softwareprocess.

- Metrics should not be used to evaluate the performance of individuals.
- Statistical software process improvement helps and organization to discover where they are strongand where they are weak.

Statistical Process Control

1. Errors are categorized by their origin
2. Record cost to correct each error and defect
3. Count number of errors and defects in each category
4. Overall cost of errors and defects computed for each category
5. Identify category with greatest cost to organization
6. Develop plans to eliminate the most costly class of errors and defects or at least reduce theirfrequency

Project Metrics

- A software team can use software project metrics to adapt project workflow and technical activities.
- Project metrics are used to avoid development schedule delays, to mitigate potential risks, and toassess product quality on an on-going basis.
- Every project should measure its inputs (resources), outputs (deliverables), and results (effectivenessof deliverables).

➢ Software Measurement

- Direct measures of a software engineering process include cost and effort.
- Direct measures of the product include lines of code (LOC), execution speed, memory size, defects reported over some time period.
- Indirect product measures examine the quality of the software product itself (e.g., functionality, complexity, efficiency, reliability, maintainability).

Size-Oriented Metrics

- Derived by normalizing (dividing) any direct measure (e.g., defects or human effort) associated withthe product or project by LOC.
- Size-oriented metrics are widely used but their validity and applicability is a matter of some debate.

Function-Oriented Metrics

- Function points are computed from direct measures of the information domain of a business software application and assessment of its complexity.
- Once computed function points are used like LOC to normalize measures for software productivity, quality, and other attributes.
- The relationship of LOC and function points depends on the language used to implement thesoftware.

Object-Oriented Metrics

- Number of scenario scripts (NSS): A scenario script is a detailed sequence of steps that describes the interaction between the user and the application. The number of scenario scripts is directly correlatedto the size of the application and to the number of test cases.
- Number of key classes (NKC): Key classes are the "highly independent components". Because key classes are central to the problem domain, the number of such classes is an indication of the amountof effort required to develop the software.
- Number of support classes: Support classes are required to implement the system but are not immediately related to the problem domain. (e.g., UI classes, database access classes, computations classes, etc.)
- Average number of support classes per key class: In general, key classes are known early in the project. Support classes are defined throughout if the average number of support classes per key class were known for a given problem domain, estimating would be much simplified.
- Number of subsystems (NSUB): A subsystem is an aggregation of classes that support a functionthat is visible to the end-user of a system.

Web Engineering Project Metrics

- Number of static Web pages (Nsp): Web pages with static content(i.e., the end-user has no controlover the content displayed on the page) are the most common of all webapp features.
- Number of dynamic Web pages (Ndp): Web pages with dynamic content (i.e., end-user actions resultin customized content displayed on the page) are essential in all e-commerce applications, search

engines, financial applications, and many other webapp categories

- Customization index: . C = Nsp / (Ndp + Nsp)
- Number of internal page links within the webapp: Internal page links are pointers that provide a hyperlink to some other webpage
- Number of persistent data objects: One or more persistent data objects (e.g., a database or data file) may be accessed by a webapp. As the number of persistent data objects grows, the complexity of the webapp also grows.
- Number of external systems interfaced: webapps must often interface with "backroom" business applications. As the requirement for interacting grows, system complexity and development effortalso increase.
- Number of static content objects: Static content objects encompass static text-based, graphical, video, animation, and audio information that are incorporated within the webapp.
- Number of dynamic content objects: Dynamic content objects are generated based on end-user actions and encompasses internally generated text-based, graphical, video, animation, and audio information that are incorporated within the webapp.
- Number of executable functions: An executable function (e.g., a script or applet) provides some computational service to the end-user.

Software Quality Metrics

- Factors assessing software quality come from three distinct points of view (product operation, product revision, product modification).
- Software quality factors requiring measures include
  - Correctness: Correctness is the degree of which the software performs its required function. The most common measure for correctness is defects per KLOC, where a defect is defined as a verified lack of conformance to requirements.
  - Maintainability: Maintainability is the ease with which a program can be corrected if an error is encountered, adapted if its environment changes, or enhanced if the customer desires a change in requirements. A simple time-oriented metric is mean time to change.
  - Integrity: This attribute measures a system's ability to withstand attacks(both accidental and intentional) to its security. To measure integrity, two additional attributes must be defined: threat and security. Threat is the probability that an attack of a specific type will occur within a given time. Security is the probability that the attack of a specific type will be repelled.
    Integrity=$\sum$ [1- (threat * (1 – security))]
  - Usability: Usability is an attempt to quantify easy to learn, easy to use, productivity increase, user attitude.
- Defect removal efficiency (DRE) is a measure of the filtering ability of the quality assurance and control activities as they are applied throughout the process framework
  DRE = E / (E + D)
  E = number of errors found before delivery of work productD = number of defects found after work product delivery

Integrating Metrics with Software Process

- Many software developers do not collect measures.
- Without measurement it is impossible to determine whether a process is improving or not.
- Baseline metrics data should be collected from a large, representative sampling of past software projects.
- Getting this historic project data is very difficult, if the previous developers did not collect data in anon-going manner.

Metrics for Small Organizations

- Most software organizations have fewer than 20 software engineers.
- Best advice is to choose simple metrics that provide value to the organization and don't require a lotof effort to collect.
- Even small groups can expect a significant return on the investment required to collect metrics, ifthis

activity leads to process improvement.

The following are the set of easily collected metrics:

- Time (hours or days) elapsed from the time a request is made until evaluation is complete.
- Effort (person-hours) to perform the evaluation
- Time (hours or days) elapsed from completion of evaluation to assignment of change order to personnel
- Effort (person-hours) required to make the change
- Time required (hours-days) to make the change
- Errors uncovered during work to make the change
- Defects uncovered after change is released to the customer base.

Establishing a Software Metrics Program

1. Identify business goal
2. Identify what you want to know
3. Identify subgoals
4. Identify subgoal entities and attributes
5. Formalize measurement goals
6. Identify quantifiable questions and indicators related to subgoals
7. Identify data elements needed to be collected to construct the indicators
8. Define measures to be used and create operational definitions for them
9. Identify actions needed to implement the measures
10. Prepare a plan to implement the measures

➢ Overview of Estimation

Software planning involves estimating how much time, effort, money, and resources will be required to build a specific software system. After the project scope is determined and the problem is decomposed intosmaller problems, software managers use historical project data (as well as personal experience and intuition) to determine estimates for each. The final estimates are typically adjusted by taking project complexity and risk into account. The resulting work product is called a project management plan.

Estimation is the process of finding an estimate, or approximation, which is a value that can be used for some purpose even if input data may be incomplete, uncertain, or unstable.

Estimation determines how much money, effort, resources, and time it will take to build a specific system or product. Estimation is based on −

- Past Data/Past Experience
- Available Documents/Knowledge
- Assumptions
- Identified Risks

The four basic steps in Software Project Estimation are −

- Estimate the size of the development product.
- Estimate the effort in person-months or person-hours.
- Estimate the schedule in calendar months.
- Estimate the project cost in agreed currency.

Observations on Estimation

- Estimation need not be a one-time task in a project. It can take place during −
  - Acquiring a Project.
  - Planning the Project.
  - Execution of the Project as the need arises.
- Project scope must be understood before the estimation process begins. It will be helpful to have historical Project Data.
- Project metrics can provide a historical perspective and valuable input for generation of quantitative estimates.

- Planning requires technical managers and the software team to make an initial commitment as it leads to responsibility and accountability.
- Past experience can aid greatly.
- Use at least two estimation techniques to arrive at the estimates and reconcile the resulting values. Refer Decomposition Techniques in the next section to learn about reconciling estimates.
- Plans should be iterative and allow adjustments as time passes and more details are known.

Project Planning Objectives

- To provide a framework that enables a software manager to make a reasonable estimate of resources, cost, and schedule.
- 'Best case' and 'worst case' scenarios should be used to bound project outcomes.
- Estimates should be updated as the project progresses.

Estimation Reliability Factors

- Project complexity
- Project size
- Degree of structural uncertainty (degree to which requirements have solidified, the ease with which functions can be compartmentalized, and the hierarchical nature of the information processed)
- Availability of historical information

Project Planning Process

1. Establish project scope
2. Determine feasibility
3. Analyze risks
4. Determine required resources

a. Determine required human resources

b. Define reusable software resources

c. Identify environmental resources

5. Estimate cost and effort

a. Decompose the problem

b. Develop two or more estimates

c. Reconcile the estimates

6. Develop project schedule

a. Establish a meaningful task set

b. Define task network

c. Use scheduling tools to develop timeline chart

d. Define schedule tracking mechanisms

Software Scope

- Describes the data to be processed and produced, control parameters, function, performance, constraints, external interfaces, and reliability.
- Often functions described in the software scope statement are refined to allow for better estimates of cost and schedule.

Customer Communication and Scope

- Determine the customer's overall goals for the proposed system and any expected benefits.
- Determine the customer's perceptions concerning the nature if a good solution to the problem.
- Evaluate the effectiveness of the customer meeting.

Feasibility

- Technical feasibility is not a good enough reason to build a product.
- The product must meet the customer's needs and not be available as an off-the-shelf purchase.

Estimation of Resources

- Human Resources (number of people required and skills needed to complete the developmentproject)
- Reusable Software Resources
  - Off-the-shelf components: Existing software can be acquired from a third party or has beendeveloped internally for a past project. COTS (commercial off-the-shelf) components are purchased from a third

party, are ready for use on the current project, and have been fully validated.

- Full-experience components: Existing specifications, design, code, or test data developed for past projects are similar to the software to be built for the current project. Therefore, modifications required for full-experience components will be relatively low-risk.
- Partial-experience components: Existing specifications, design, code, or test data developed for past projects are related to the software to be built for the current project but will require substantial modification. Therefore, modifications required for partial-experience components have a fair degree of risk.
- New components: Software components must be built by the software team specifically for theneeds of the current project.

- Environment Resources (hardware and software required to be accessible by software team during the development process)

➢ Empirical Estimation Models

COCOMO Model

Boehm proposed COCOMO (Constructive Cost Estimation Model) in 1981.COCOMO is one of the most generally used software estimation models in the world. COCOMO predicts the efforts and schedule of a software product based on the size of the software.

The necessary steps in this model are:

1.Get an initial estimate of the development effort from evaluation of thousands of delivered lines of source code (KDLOC).

2.Determine a set of 15 multiplying factors from various attributes of the project.

3.Calculate the effort estimate by multiplying the initial estimate with all the multiplying factors i.e., multiply the values in step1 and step2.

The initial estimate (also called nominal estimate) is determined by an equation of the form used in the static single variable models, using KDLOC as the measure of the size. To determine the initial effort E_i in person-months the equation used is of the type is shown below

$$E_i=a*(KDLOC)\ b$$

The value of the constant a and b are depends on the project type.

In COCOMO, projects are categorized into three types:

1.Organic

2.Semidetached

3.Embedded

1.Organic: A development project can be treated of the organic type, if the project deals with developing a well-understood application program, the size of the development team is reasonably small, and the team members are experienced in developing similar methods of projects. Examples of this type of projects are simple business systems, simple inventory management systems, and data processing systems.

2. Semidetached: A development project can be treated with semidetached type if the development consists of a mixture of experienced and inexperienced staff. Team members may have finite experience in related systems but may be unfamiliar with some aspects of the order being developed. Example of Semidetached system includes developing a new operating system (OS), a Database Management System (DBMS), and complex inventory management system.

3. Embedded: A development project is treated to be of an embedded type, if the software being developed is strongly coupled to complex hardware, or if the stringent regulations on the operational method exist. For Example: ATM, Air Traffic control.

For three product categories, Bohem provides a different set of expression to predict effort (in a unit of person month)and development time from the size of estimation in KLOC(Kilo Line of code) efforts estimation takes into account the productivity loss due to holidays, weekly off, coffee breaks, etc.

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

1.Basic Model

2.Intermediate Model

3.Detailed Model

1. Basic COCOMO Model: The basic COCOMO model provide an accurate size of the project parameters. The following expressions give the basic COCOMO estimation model:

Effort=a_1*(KLOC) a_2 PM

Tdev=b_1*(efforts) b2 Months

Where

KLOC is the estimated size of the software product indicate in Kilo Lines of Code,

a_1, a2, b1, b2 are constants for each group of software products,

Tdev is the estimated time to develop the software, expressed in months,

Effort is the total effort required to develop the software product, expressed in person months (PMs).

Estimation of development effort

For the three classes of software products, the formulas for estimating the effort based on the code size are shown below:

Organic: Effort = 2.4(KLOC) 1.05 PM

Semi-detached: Effort = 3.0(KLOC) 1.12 PM

Embedded: Effort = 3.6(KLOC) 1.20 PM

Estimation of development time

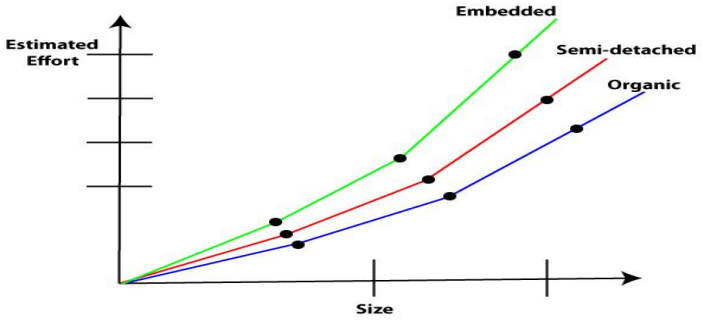
For the three classes of software products, the formulas for estimating the development time based on the effort are given below:

Organic: Tdev = 2.5(Effort) 0.38 Months

Semi-detached: Tdev = 2.5(Effort) 0.35 Months

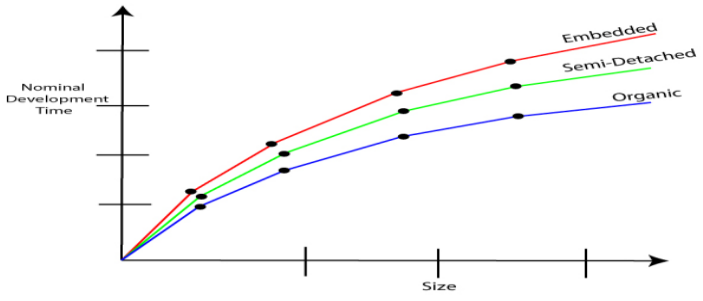
Embedded: Tdev = 2.5(Effort) 0.32 Months

Some insight into the basic COCOMO model can be obtained by plotting the estimated characteristics for different software sizes. Fig shows a plot of estimated effort versus product size. From fig, we can observe that the effort is somewhat superliner in the size of the software product. Thus, the effort required to develop a product increases very rapidly with project size.



Effort versus product size

The development time versus the product size in KLOC is plotted in fig. From fig it can be observed that the development time is a sub linear function of the size of the product, i.e. when the size of the product increases by two times, the time to develop the product does not double but rises moderately. This can be explained by the fact that for larger products, a larger number of activities which can be carried out concurrently can be identified. The parallel activities can be carried out simultaneously by the engineers. This reduces the time to complete the project. Further, from fig, it can be observed that the development time is roughly the same for all three categories of products. For example, a 60 KLOC program can be developed in approximately 18 months, regardless of whether it is of organic, semidetached, or embedded type.



Development time versus size

From the effort estimation, the project cost can be obtained by multiplying the required effort by the manpower cost per month. But, implicit in this project cost computation is the assumption that the entire project cost is incurred on account of the manpower cost alone. In addition to manpower cost, a project would incur costs due to hardware and software required for the project and the company overheads for administration, office space, etc.

It is important to note that the effort and the duration estimations obtained using the COCOMO model

are called a nominal effort estimate and nominal duration estimate. The term nominal implies that if anyone tries to complete the project in a time shorter than the estimated duration, then the cost will increase drastically. But, if anyone completes the project over a longer period of time than the estimated, then there is almost no decrease in the estimated cost value.

Example1: Suppose a project was estimated to be 400 KLOC. Calculate the effort and development time for each of the three model i.e., organic, semi-detached & embedded.

Solution: The basic COCOMO equation takes the form:

$Effort=a_1*(KLOC)\ a_2\ PM$

$Tdev=b_1*(efforts)b_2$ Months

Estimated Size of project= 400 KLOC

(i)Organic Mode

$E = 2.4 * (400)1.05 = 1295.31$ PM

$D = 2.5 * (1295.31)0.38=38.07$ PM

(ii)Semidetached Mode

$E = 3.0 * (400)1.12=2462.79$ PM

$D = 2.5 * (2462.79)0.35=38.45$ PM

(iii) Embedded Mode

$E = 3.6 * (400)1.20 = 4772.81$ PM

$D = 2.5 * (4772.8)0.32 = 38$ PM

Example2: A project size of 200 KLOC is to be developed. Software development team has average experience on similar type of projects. The project schedule is not very tight. Calculate the Effort, development time, average staff size, and productivity of the project.

Solution: The semidetached mode is the most appropriate mode, keeping in view the size, schedule and experience of development time.

Hence $E=3.0(200)1.12=1133.12$PM

$D=2.5(1133.12)0.35=29.3$PM

$$\text{Average Staff Size (SS)} = \frac{E}{D} \text{ Persons}$$

$$= \frac{1133.12}{29.3} = 38.67 \text{ Persons}$$

$$\text{Productivity} = \frac{KLOC}{E} = \frac{200}{1133.12} = 0.1765 \text{ KLOC/PM}$$

P = 176 LOC/PM

2. Intermediate Model: The basic Cocomo model considers that the effort is only a function of the number of lines of code and some constants calculated according to the various software systems. The intermediate COCOMO model recognizes these facts and refines the initial estimates obtained through the basic COCOMO model by using a set of 15 cost drivers based on various attributes of software engineering.

Classification of Cost Drivers and their attributes:

(i) Product attributes -

- Required software reliability extent
- Size of the application database
- The complexity of the product

Hardware attributes -

- Run-time performance constraints
- Memory constraints
- The volatility of the virtual machine environment
- Required turnabout time

Personnel attributes -

- Analyst capability
- Software engineering capability

- Applications experience
- Virtual machine experience
- Programming language experience

Project attributes -

- Use of software tools
- Application of software engineering methods
- Required development schedule

Intermediate COCOMO equation:

$E=a_i (KLOC)b_i*EAF$

$D=c_i (E)d_i$

Coefficients for intermediate COCOMO

Project	a_i	b_i	c_i	d_i
Organic	2.4	1.05	2.5	0.38
Semidetached	3.0	1.12	2.5	0.35
Embedded	3.6	1.20	2.5	0.32

3. Detailed COCOMO Model: Detailed COCOMO incorporates all qualities of the standard version with an assessment of the cost driver?s effect on each method of the software engineering process. The detailed model uses various effort multipliers for each cost driver property. In detailed cocomo, the whole software is differentiated into multiple modules, and then we apply COCOMO in various modules to estimate effort and then sum the effort.

The Six phases of detailed COCOMO are:

1.Planning and requirements
2.System structure
3.Complete structure
4.Module code and test
5.Integration and test
6.Cost Constructive model

The effort is determined as a function of program estimate, and a set of cost drivers are given according to every phase of the software lifecycle.

➢ Decomposition Techniques

- Software sizing: The "size" of software to be built can be estimated using a direct measure, LOC (lines of code), or on indirect measure, FP (function points). Four different approaches to the sizingproblem are:
  - "Fuzzy logic" sizing: To apply this approach, the planner must identify the type of application, establish its magnitude on a quantitative scale, and then refine the magnitude within the originalrange.
  - Function point sizing: The planner develops estimates of the information domain characteristics.
  - Standard component sizing: Software is composed of number of different "standard components"that are generic to a particular application area. The project planner estimates the number of occurrences of each standard component and then uses historical project data to determine the delivered size per standard component.
  - Change sizing: This approach is used when a project encompasses the use of existing software that must be modified in some way as part of a project. The planner estimates the number and type (e.g., reuse, adding code, changing code, and deleting code) of modifications that must beaccomplished.
- Problem-based estimation (using LOC decomposition focuses on software functions, using FP decomposition focuses on information domain characteristics)
- Process-based estimation (decomposition based on tasks required to complete the software process framework. That is, the process is decomposed into a relatively small set of tasks and the effort required to accomplish each task is estimated)
- Use-case estimation (promising, but controversial due to lack of standardization of use-cases)

Causes of Estimation Reconciliation Problems

- Project scope is not adequately understood or misinterpreted by planner
- Productivity data used for problem-based estimation techniques is inappropriate or obsolete for the

application

➢ Empirical Estimation Models

- Typically derived from regression analysis of historical software project data with estimated person-months as the dependent variable and KLOC, FP, or object points as independent variables.
- Constructive COst MOdel (COCOMO) is an example of a static estimation model.
- COCOMO II is a hierarchy of estimation models that address the following areas

- Application composition model. Used during the early stages of software engineering, when prototyping of user interfaces, consideration of software and system interaction, assessment of performance, and evaluation technology maturity are paramount.
- Early design stage model. Used once requirements have been stabilized and basic softwarearchitecture has been established.
- Post-architecture stage model. Used during the construction of the software.

The COCOMO II model requires sizing information. Three different sizing options are available as part of the model hierarchy: object points, function points, and lines of source code.

The COCOMO II application composition model uses object points-an indirect software measure thatis computed using counts of the number of (1) screens (2) reports and (3) components likely to be required to build the application. The object point count is determined by multiplying the original number of object instances by the weighting factor. When component-based development or general software reuse is to be applied, the percent of reuse (%reuse) is estimated and the object point count is adjusted:

NOP=(object points) * [(100 - %reuse)/100]Where NOP is defined as new object points.

PROD (productivity rate)=NOP/person-monthEstimated effort=NOP/PROD

Estimation for Agile Development

1. Each user scenario is considered separately
2. The scenario is decomposed into a set of engineering tasks
3. Each task is estimated separately

a. May use historical data, empirical model, or experience

4. Scenario volume can be estimated (LOC, FP, use-case count, etc.)Total scenario estimate computed

a. Sum estimates for each task

b. Translate volume estimate to effort using historical data

5. The effort estimates for all scenarios in the increment are summed to get an increment estimate

➢ Estimation for Object-Oriented Projects

- Develop estimates using effort decomposition ,FP analysis ,and any other method that is applicable for conventional applications
- Using object-oriented analysis modeling, develop use-cases and determine a count. Recognize that the number of use-cases may change as the project progressed.
- From the analysis model, determine the number of key classes
- Categorize the type of interface for the application ,and develop a multiplier for support classes:

Interface	Multiplier
No GUI	2.0
Text-based user interface	2.25
GUI	2.5
Complex GUI	3.0

Multiply the number of key classes by the multiplier to obtain an estimate for the number of support classes.

- Multiply the total number of classes by the average number of work-units per class.
- Cross-check the class-based estimate by multiplying the average number of work-units per use-case.

➢ Other estimation Techniques

Software team encounters an extremely short project duration that is likely to have a continuing stream

of changes, project planning in general and estimation in particular should be abbreviated.

Estimation for Agile Development:

Estimation for agile projects uses a decomposition approach that encompasses the following steps:

- Each user scenario is considered separately for estimation purposes.
- The scenario is decomposed into the set of functions and the software engineering tasks that will be require to develop them.
- Each task is estimated separately.Note: estimation can be based on historical data, an empirical model, or "experience".
- Alternatively the "volume" (size) of the scenario can be estimated in LOC, FP, or some other volume-oriented measure (e.g., object points).
- Estimates for each task are summed to create an estimate for the scenario.
- Alternatively, the volume estimate for the scenario is translated into effort using historical data.
- The effort estimates for all scenarios that are to be implemented for a given software increment are summed to develop the effort estimate for the increment.

This estimation approach serves two purposes:

(1) To ensure that the number of scenarios to be included in the increment conforms to the available resources and (2 to establish a basis for allocating effort as the increment is developed.

Estimation for Web Engineering Projects:

The following information domain values when adapting function points for WebApp estimation**:**

- Inputs are each input screen or form (for example, CGI or java), each maintenance screen, and if u use a tab notebook metaphor anywhere, each tab).
- Outputs are each static Web page, each dynamic Webpage script (for example, ASP, ISAPI,or other DHTML script),and each report(whether Web based or administrative in nature).
- Tables are each logical table in the database plus, if u are using XML to store data in a file, each XML object (or collection of XML attributes).
- Interfaces retain their definition as logical files (for example, unique record formats)into our out-of-the-system boundaries.
- Queries are each externally published or use a message-oriented interface.A typical example is DCOM or COM external references.

  Function points are a reasonable indicator of volume for a WebApp.

➢ Overview of Project Scheduling

The chapter describes the process of building and monitoring schedules for software development projects. To build complex software systems, many engineering tasks need to occur in parallelwith one another to complete the project on time. The output from one task often determines when anothermay begin. It is difficult to ensure that a team is working on the most appropriate tasks without building a detailed schedule and sticking to it.

Root Causes for Late Software

- Unrealistic deadline established outside the team
- Changing customer requirements not reflected in schedule changes
- Underestimating the resources required to complete the project
- Risks that were not considered when project began
- Technical difficulties that have not been predicted in advance
- Human difficulties that have not been predicted in advance
- Miscommunication among project staff resulting in delays
- Failure by project management to recognize project falling behind schedule and failure to take corrective action to correct problems

How to Deal With Unrealistic Schedule Demands

1. Perform a detailed project estimate for project effort and duration using historical data.
2. Use an incremental process model that will deliver critical functionality imposed by deadline, butdelay other requested functionality.
3. Meet with the customer and explain why the deadline is unrealistic using your estimates based onprior

team performance.

4. Offer an incremental development and delivery strategy as an alternative to increasing resources or allowing the schedule to slip beyond the deadline.

Project Scheduling Perspectives

- One view is that the end-date for the software release is set externally and that the softwareorganization is constrained to distribute effort in the prescribed time frame.
- Another view is that the rough chronological bounds have been discussed by the developers and customers, but the end-date is best set by the developer after carefully considering how to best usethe resources needed to meet the customer's needs.

Software Project Scheduling Principles

- Compartmentalization - the product and process must be decomposed into a manageable number of activities and tasks
- Interdependency - tasks that can be completed in parallel must be separated from those that must completed serially
- Time allocation - every task has start and completion dates that take the task interdependencies into account
- Effort validation - project manager must ensure that on any given day there are enough staffmembers assigned to complete the tasks within the time estimated in the project plan.
- Defined Responsibilities - every scheduled task needs to be assigned to a specific team member
- Defined outcomes - every task in the schedule needs to have a defined outcome (usually a workproduct or deliverable)
- Defined milestones - a milestone is accomplished when one or more work products from anengineering task have passed quality review

Relationship Between People and Effort

- Adding people to a project after it is behind schedule often causes the schedule to slip further
- The relationship between the number of people on a project and overall productivity is not linear (e.g., 3 people do not produce 3 times the work of 1 person, if the people have to work in cooperationwith one another)
- The main reasons for using more than 1 person on a project are to get the job done more rapidly andto improve software quality.

Project Effort Distribution

- The 40-20-40 rule (a rule of thumb):
  - 40% front-end analysis and design
  - 20% coding
  - 40% back-end testing
- Generally accepted guidelines are:
  - 02-03 % planning
  - 10-25 % requirements analysis
  - 20-25 % design
  - 15-20 % coding
  - 30-40 % testing and debugging

Software Project Types

1. Concept development - initiated to explore new business concept or new application of technology
2. New application development - new product requested by customer
3. Application enhancement - major modifications to function, performance, or interfaces (observableto user)
4. Application maintenance - correcting, adapting, or extending existing software (not immediately obvious to user)
5. Reengineering - rebuilding all (or part) of a legacy system

Factors Affecting Task Set

- Size of project

- Number of potential users
- Mission criticality
- Application longevity
- Requirement stability
- Ease of customer/developer communication
- Maturity of applicable technology
- Performance constraints
- Embedded/non-embedded characteristics
- Project staffing
- Reengineering factors

Concept Development Tasks

- Concept scoping - determine overall project scope
- Preliminary concept planning - establishes development team's ability to undertake the proposedwork
- Technology risk assessment - evaluates the risk associated with the technology implied by thesoftware scope
- Proof of concept - demonstrates the feasibility of the technology in the software context
- Concept implementation - concept represented in a form that can be used to sell it to the customer
- Customer reaction to concept - solicits feedback on new technology from customer

Scheduling

- Task networks (activity networks) are graphic representations that can be of the taskinterdependencies and can help define a rough schedule for particular project
- Scheduling tools should be used to schedule any non-trivial project.
- PERT (program evaluation and review technique) and CPM (critical path method) are quantitative techniques that allow software planners to identify the chain of dependent tasks in the project work breakdown structure that determine the project duration time.
- Timeline (Gantt) charts enable software planners to determine what tasks will be need to be conducted at a given point in time (based on estimates for effort, start time, and duration for eachtask).
- The best indicator of progress is the completion and successful review of a defined software work product.
- Time-boxing is the practice of deciding a priori the fixed amount of time that can be spent on eachtask. When the task's time limit is exceeded, development moves on to the next task (with the hope that a majority of the critical work was completed before time ran out).

Tracking Project Schedules

- Periodic status meetings
- Evaluation of results of all work product reviews
- Comparing actual milestone completion dates to scheduled dates
- Comparing actual project task start-dates to scheduled start-dates
- Informal meeting with practitioners to have them assess subjectively progress to date and future problems
- Use earned value analysis to assess progress quantitatively

Tracking Increment Progress for OO Projects

- Technical milestone: OO analysis complete
  - All hierarchy classes defined and reviewed
  - Class attributes and operations are defined and reviewed
  - Class relationships defined and reviewed
  - Behavioral model defined and reviewed
  - Reusable classed identified
- Technical milestone: OO design complete
  - Subsystems defined and reviewed
  - Classes allocated to subsystems and reviewed
  - Task allocation has been established and reviewed

- Responsibilities and collaborations have been identified
- Attributes and operations have been designed and reviewed
- Communication model has been created and reviewed

- Technical milestone: OO programming complete
- Each new design model class has been implemented
- Classes extracted from the reuse library have been implemented
- Prototype or increment has been built

- Technical milestone: OO testing complete
- The correctness and completeness of the OOA and OOD models has been reviewed
- Class-responsibility-collaboration network has been developed and reviewed
- Test cases are designed and class-level tests have been conducted for each class
- Test cases are designed, cluster testing is completed, and classes have been integrated
- System level tests are complete

Earned Value Analysis

- Earned value is a quantitative measure given to each task as a percent of project completed so far.

1. The total hours to complete each project task are estimated
2. The effort to complete the project is computed by summing the effort to complete each task
3. Each task is given an earned value based on its estimated percentage contribution to the total.

➢ Overview of Risk Management

Risks are potential problems that might affect the successful completion of a software project. Risks involve uncertainty and potential losses. Risk analysis and management is intended to help a software team understand and manage uncertainty during the development process. The important thing is to remember that things can go wrong and to make plans to minimize their impact when they do. The work product is called a Risk Mitigation, Monitoring, and Management Plan (RMMM).

Risk Strategies

- Reactive strategies - very common, also known as fire fighting, project team sets resources aside todeal with problems and does nothing until a risk becomes a problem
- Proactive strategies - risk management begins long before technical work starts, risks are identified and prioritized by importance, then team builds a plan to avoid risks if they can or minimize them if the risks turn into problems

Software Risks

- Project risks - threaten the project plan
- Technical risks - threaten product quality and the timeliness of the schedule
- Business risks - threaten the viability of the software to be built (market risks, strategic risks, management risks, budget risks).Known risks - predictable from careful evaluation of current project plan and those extrapolated frompast project experience
- Unknown risks - some problems simply occur without warning

Risk Identification

- Product-specific risks - the project plan and software statement of scope are examined to identify any special characteristics of the product that may threaten the project plan
- Generic risks - are potential threats to every software product (product size, business impact, customer characteristics, process definition, development environment, technology to be built, staff size and experience)

Risk Checklist Items

- Product size
- Business impact
- Customer characteristics
- Process definition
- Development environment
- Technology to be built
- Staff size and experience

Project-Related Risk Assessment Questions

1. Have top software and customer managers formally committed to support the project?
2. Are end-users enthusiastically committed to the project?
3. Are requirements fully understood by developers and customers?
4. Were customers fully involved in requirements definition?
5. Do end-users have realistic expectations?
6. Is project scope stable?
7. Does software team have the right skill set?
8. Are project requirements stable?
9. Does the project team have experience with technology to be implemented?
10. Is the number of people on project team adequate to do the job?
11. Do all stakeholders agree on the importance of the project the requirements for the systems beingbuilt?

Risk Impact

- Risk components - performance, cost, support, schedule
- Risk impact - negligible, marginal, critical, catastrophic
- The risk drivers affecting each risk component are classified according to their impact category andthe potential consequences of each undetected software fault or unachieved project outcome are described

Risk Projection (Estimation)

1. Establish a scale that reflects the perceived likelihood of each risk
2. Delineate the consequences of the risk
3. Estimate the impact of the risk on the project and product
4. Note the overall accuracy of the risk projection to avoid misunderstandings

Risk Table Construction

- List all risks in the first column of the table
- Classify each risk and enter the category label in column two
- Determine a probability for each risk and enter it into column three
- Enter the severity of each risk (negligible, marginal, critical, catastrophic) in column four
- Sort the table by probability and impact value
- Determine the criteria for deciding where the sorted table will be divided into the first priorityconcerns and the second priority concerns
- First priority concerns must be managed (a fifth column can be added to contain a pointer into the RMMM)

Assessing Risk Impact

- Factors affecting risk consequences - nature (types of problems arising), scope (combines severitywith extent of project affected), timing (when and how long impact is felt)
- If costs are associated with each risk table entry a risk exposure metric can be computed (RE = Probability * Cost) and added to the risk table.

Risk Assessment

1. Define referent levels for each project risk that can cause project termination (performancedegradation, cost overrun, support difficulty, schedule slippage).
2. Attempt to develop a relationship between each risk triple (risk, probability, impact) and each of the reference levels.
3. Predict the set of referent points that define a region of termination, bounded by a curve or areas of uncertainty.
4. Try to predict how combinations of risks will affect a referent level.

Risk Refinement

- Process of restating the risks as a set of more detailed risks that will be easier to mitigate, monitor,and manage.
- CTC (condition-transition-consequence) format may be a good representation for the detailed risks(e.g., given that <condition> then there is a concern that (possibly) <consequence>).
  This general condition can be refined in the following manner:

- Sub condition 1. Certain reusable components were developed by a third party with no knowledge of internal design standards.
- Sub condition 2. The design standard for component interfaces has not been solidified and may not conform to certain existing reusable components.
- Sub condition 3. Certain reusable components have been implemented in a language that is not supported on the target environment.

Risk Mitigation, Monitoring, and Management

- Risk mitigation (proactive planning for risk avoidance)
- Risk monitoring (assessing whether predicted risks occur or not, ensuring risk aversion steps are being properly applied, collect information for future risk analysis, attempt to determine which risks caused which problems)
- Risk management and contingency planning (actions to be taken in the event that mitigation steps have failed and the risk has become a live problem)

Safety Risks and Hazards

- Risks are also associated with software failures that occur in the field after the development project has ended.
- Computers control many mission critical applications in modern times (weapons systems, flight control, industrial processes, etc.).
- Software safety and hazard analysis are quality assurance activities that are of particular concern for these types of applications

Risk Information Sheets

- Alternative to RMMM (Risk Mitigation, Monitoring and Management plan) in which each risk is documented individually.
- Often risk information sheets (RIS) are maintained using a database system.
- RIS components - risk id, date, probability, impact, description, refinement, mitigation/monitoring, management/contingency/trigger, status, originator, assigned staff member.

➢ Overview of Reengineering

This chapter defines reengineering as the process of rebuilding legacy software products. The rebuilt software products often have increased functionality, better performance, greater reliability, and are easier to maintain than their predecessors. Business process reengineering (BPR) defines business goals, evaluates existing business processes, and creates revised business processes that better meet current goals. Software reengineering involves inventory analysis, document restructuring, reverse engineering, program and data restructuring, and forward engineering. Many reengineering work products are the same as those generated for any software engineering process (analysis models, design models, test procedures). The final product for any reengineering process is a reengineered business process and/or the reengineered software to support it. The same SQA practices are applied to software reengineering as they would to any other software development process. Testing is used to uncover errors in content, functionality, and interoperability.

Business Process Reengineering Principles

- Organize around outcomes, not tasks.
- Have those people who use the output of a process, perform the process.
- Incorporate information processing work into the real work that produces the raw information.
- Treat geographically dispersed resources as though they were centralized.
- Link parallel activities instead of integrating their results.
- Put the decision point where the work is performed and build control into the process.
- Capture the data once, at its source.

Business Process Reengineering Model

- Business definition - business goals are identified in the context of four key drivers (cost reduction, time reduction, quality improvement, empowerment)
- Process identification - processes critical to achieving business goals are identified and prioritized
- Process evaluation - existing processes are analyzed and measured, costs and time consumed by

processes are noted, quality/performance problems are isolated

- Process specification and design - use-cases are prepared for each process to be redesigned, these use-case scenarios deliver some outcome to a customer, new tasks are designed for each process
- Prototyping - used to test processes before integrating them into the business
- Refinement and instantiation - based on feedback from the prototype, business processes are refined and then instantiated within a business system

Software Reengineering

Software Maintenance

- Corrective maintenance (fixing errors)
- Adaptive maintenance (accommodating changes in the environment or user needs)
- Perfective maintenance (reengineering the application to improve performance or make the software product easier to maintain)
- Preventative maintenance (modifying software to avoid anticipated future problems)

Software Reengineering Process Model

- Inventory analysis - sorting active software applications by business criticality, longevity, current maintainability, and other local criteria helps to identify reengineering candidates
- Document restructuring - need to decide to live with weak documentation, update poor documents if they are used, or fully rewrite the documentation for critical systems focusing on the "essential minimum"
- Reverse engineering - process of design recovery - analyzing a program in an effort to create a representation of the program at some abstraction level higher than source code
- Code restructuring - source code is analyzed and violations of structured programming practices are noted and repaired, the revised code also needs to be reviewed and tested
- Data restructuring - usually requires full reverse engineering, current data architecture is issected and data models are defined, existing data structures are reviewed for quality
- Forward engineering - also called reclamation or renovation, recovers design information from existing source code and uses this information to reconstitute the existing system to improve its overall quality and/or performance

Reverse Engineering Concepts

- Abstraction level - ideally want to be able to derive design information at the highest level possible
- Completeness - level of detail provided at a given abstraction level
- Interactivity - degree to which humans are integrated with automated reverse engineering tools
- Directionality - one-way means the software engineer doing the maintenance activity is given all information extracted from source code, two-way means the information is fed to a reengineering tool that attempts to regenerate the old program
- Extract abstractions - meaningful specification of processing performed is derived from old source code

Reverse Engineering Activities

- Understanding processing - source code is analyzed at varying levels of detail (system, program, component, pattern, statement) to understand procedural abstractions and overall functionality
- Understanding data
  - internal data structures - program code is examined with the intention of grouping related program variables.
  - database structure - often done prior to moving from one database paradigm to another (e.g., flat file to relational)

Reverse Engineering User interfaces

- What are the basic actions (e.g., key strokes or mouse operations) processed by the interface?
- What is a compact description of the system's behavioral response to these actions?
- What concept of equivalence of interfaces is relevant here?

Restructuring Benefits

- Improved program and documentation quality
- Makes programs easier to learn, improves productivity, reduces developer frustration

- Reduces effort required to maintain software
- Software is easier to test and debug

Types of Restructuring

- Code restructuring
  - program logic modeled using Boolean algebra and series of transformation rules are applied to yield restructured logic
  - create resource exchange diagram showing data types, procedure and variables shared between modules, restructure program architecture to minimize module coupling
- Data restructuring
  - analysis of source code
  - data redesign
  - data record standardization
  - data name rationalization
  - file or database translation

Identifying Forward Engineering Candidates

1. Program will continue to be used for several more years
2. Program is currently being used successfully
3. Program is likely to undergo major modification or enhancement in the future

Forward Engineering Client/Server Architectures

- application functionality migrates to each client computer
- new GUI interfaces implemented at client sites
- database functions allocated to servers
- specialized functionality may remain at server site
- new communications, security, archiving, and control requirements must be established at both client and server sites

Forward Engineering Object-oriented Architectures

- existing software is reverse engineered so that appropriate data, functional, and behavioral models can be created
- use-cases are created if reengineered system extends functionality of application
- data models created during reverse engineering are used with CRC modeling as a basis to defineclasses
- create class hierarchies, object-relationship models, object-behavior models and begin object-oriented design
- a component-based process model may be used if a robust component library already exists
- where components must be built from scratch, it may be possible to reuse algorithms and datastructures from the original application

Forward Engineering User interfaces

1. understand the original user interface and how the data moves between the user interface and the remainder of the application
2. remodel the behavior implied by the existing user interface into a series of abstractions that have meaning in the context of a GUI
3. introduce improvements that make the mode of interaction more efficient
4. build and integrate the new GUI

Economics of Reengineering

- Cost of maintenance = cost annual of operation and maintenance over application lifetime
- Cost of reengineering = predicted return on investment reduced by cost of implementing changes and engineering risk factors
- Cost benefit = Cost of reengineering - Cost of maintenance