# CHAPTER 3: (MANEGING SOFTWARE PROJECT)

## Q. Explain Software Metrices.

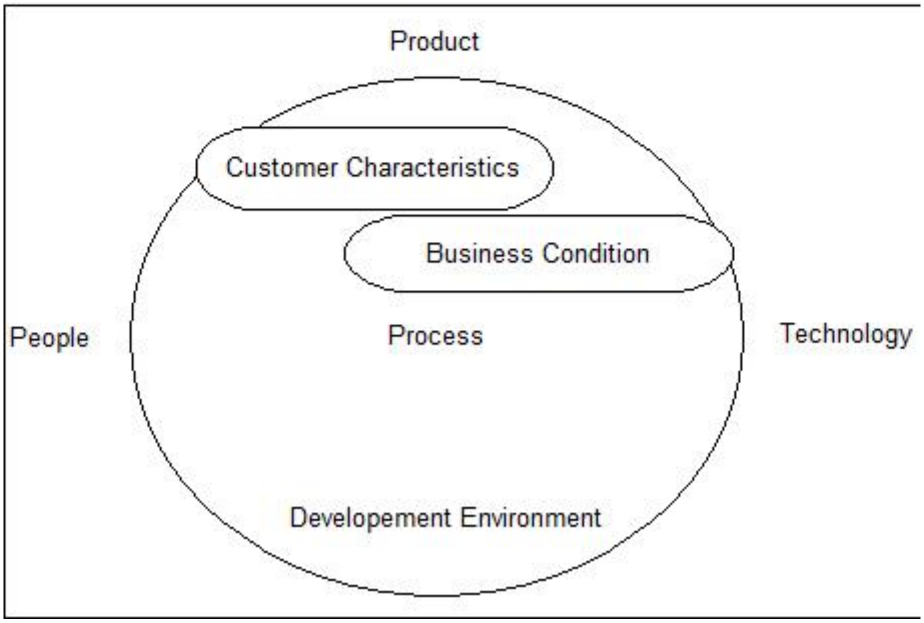
The success of a software project depends largely on the quality and effectiveness of the software design. Hence, it is important to develop software metrics from which meaningful indicators can be derived. With the help of these indicators, necessary steps are taken to design the software according to the user requirements. Various design metrics such as architectural design metrics, component-level design metrics, user-interface design metrics, and metrics for object-oriented design are used to indicate the complexity, quality, and so on of the software design.

Measurement is done by metrics. Three parameters are measured: process measurement through process metrics, product measurement through product metrics, and project measurement through project metrics. effectiveness and quality of software process, determine maturity of the process, effort required in the process, effectiveness of defect removal during development, and so on. Product metrics is the measurement of work product produced during different phases of software development. Project metrics illustrate the project characteristics and their execution.

## Process Metrics

To improve any process, it is necessary to measure its specified attributes, develop a set of meaningful metrics based on these attributes, and then use these metrics to obtain indicators in order to derive a strategy for process improvement.

Using software process metrics, software engineers are able to assess the efficiency of the software process that is performed using the process as a framework. Process is placed at the centre of the triangle connecting three factors (product, people, and technology), which have an important influence on software quality and organization performance. The skill and motivation of the people, the complexity of the product and the level of technology used in the software development have an important influence on the quality and team performance. The process triangle exists within the circle of environmental conditions, which includes development environment, business conditions, and customer /user characteristics.



To measure the efficiency and effectiveness of the software process, a set of metrics is formulated based on the outcomes derived from the process. These outcomes are listed below.

- Number of errors found before the software release
- Defect detected and reported by the user after delivery of the software
- Time spent in fixing errors
- Work products delivered
- Human effort used
- Time expended
- Conformity to schedule
- Wait time
- Number of contract modifications
- Estimated cost compared to actual cost.

Note that process metrics can also be derived using the characteristics of a particular software engineering activity. For example, an organization may measure the effort and time spent by considering the user interface design.

It is observed that process metrics are of two types, namely, private and public. Private Metrics are private to the individual and serve as an indicator only for the specified individual(s). Defect rates by a software module and defect errors by an individual are examples of private process metrics. Note that some process metrics are public to all team members but private to the project. These include errors detected while performing formal technical reviews and defects reported about various functions included in the software.

Public metrics include information that was private to both individuals and teams. Project-level defect rates, effort and related data are collected, analyzed and assessed in order to obtain indicators that help in improving the organizational process performance.

## Product Metrics

In software development process, a working product is developed at the end of each successful phase. Each product can be measured at any stage of its development. Metrics are developed for these products so that they can indicate whether a product is developed according to the user requirements. If a product does not meet user requirements, then the necessary actions are taken in the respective phase.

Product metrics help software engineer to detect and correct potential problems before they result in catastrophic defects. In addition, product metrics assess the internal product attributes in order to know the efficiency of the following.

- Analysis, design, and code model
- Potency of test cases
- Overall quality of the software under development.

Various metrics formulated for products in the development process are listed below.

- **Metrics for analysis model:** These address various aspects of the analysis model such as system functionality, system size, and so on.
- **Metrics for design model:** These allow software engineers to assess the quality of design and include architectural design metrics, component-level design metrics, and so on.
- **Metrics for source code:** These assess source code complexity, maintainability, and other characteristics.
- **Metrics for testing:** These help to design efficient and effective test cases and also evaluate the effectiveness of testing.
- **Metrics for maintenance:** These assess the stability of the software product.

Metrics for the Analysis Model

There are only a few metrics that have been proposed for the analysis model. However, it is possible to use metrics for project estimation in the context of the analysis model. These metrics are used to examine the analysis model with the objective of predicting the size of the resultant system. Size acts as an indicator of increased coding, integration, and testing effort; sometimes it also acts as an indicator of complexity involved in the software design. Function point and lines of code are the commonly used methods for size estimation.

### Function Point (FP) Metric

The function point metric, which was proposed by A.J Albrecht, is used to measure the functionality delivered by the system, estimate the effort, predict the number of errors, and estimate the number of components in the system. Function point is derived by using a relationship between the complexity of software and the information domain value. Information domain values used in function point include the number of external inputs, external outputs, external inquires, internal logical files, and the number of external interface files.

**Lines of Code (LOC)**

Lines of code (LOC) is one of the most widely used methods for size estimation. LOC can be defined as the number of delivered lines of code, excluding comments and blank lines. It is highly dependent on the programming language used as code writing varies from one programming language to another. Fur example, lines of code written (for a large program) in assembly language are more than lines of code written in C++.

From LOC, simple size-oriented metrics can be derived such as errors per KLOC (thousand lines of code), defects per KLOC, cost per KLOC, and so on. LOC has also been used to predict program complexity, development effort, programmer performance, and so on. For example, Hasltead proposed a number of metrics, which are used to calculate program length, program volume, program difficulty, and development effort.

## Q. Explain Software Project planning in details.

- Software project planning encompasses five major activities
  - Estimation, scheduling, risk analysis, quality management planning, and change management planning
- Estimation determines how much money, effort, resources, and time it will take to build a specific system or product
- The software team first estimates
  - The work to be done
  - The resources required
  - The time that will elapse from start to finish
- Then they establish a project schedule that
  - Defines tasks and milestones
  - Identifies who is responsible for conducting each task
  - Specifies the inter-task dependencies

## **Observations on Estimation**:

- Planning requires technical managers and the software team to make an initial commitment
- Process and project metrics can provide a historical perspective and valuable input for generation of quantitative estimates
- Past experience can aid greatly
- Estimation carries inherent risk, and this risk leads to uncertainty
- The availability of historical information has a strong influence on estimation risk
- When software metrics are available from past projects
  - Estimates can be made with greater assurance
  - Schedules can be established to avoid past difficulties
  - Overall risk is reduced
- Estimation risk is measured by the degree of uncertainty in the quantitative estimates for cost, schedule, and resources
- Nevertheless, a project manager should not become obsessive about estimation

- Plans should be iterative and allow adjustments as time passes and more is made certain

**Task Set for Project Planning:**

1) Establish project scope
2) Determine feasibility
3) Analyze risks
4) Define required resources
   a) Determine human resources required
   b) Define reusable software resources
   c) Identify environmental resources
5) Estimate cost and effort
   a) Decompose the problem
   b) Develop two or more estimates using different approaches
   c) Reconcile the estimates
6) Develop a project schedule
   a) Establish a meaningful task set
   b) Define a task network
   c) Use scheduling tools to develop a timeline chart
   d) Define schedule tracking mechanisms

**Software Scope**:

- Software scope describes
  - The functions and features that are to be delivered to end users
  - The data that are input to and output from the system
  - The "content" that is presented to users as a consequence of using the software
  - The performance, constraints, interfaces, and reliability that bound the system
- Scope can be define using two techniques
  - A narrative description of software scope is developed after communication with all stakeholders

A set of use cases is developed by end users

- After the scope has been identified, two questions are asked
  - Can we build software to meet this scope?
  - Is the project feasible?
- Software engineers too often rush (or are pushed) past these questions
- Later they become mired in a project that is doomed from the onset

**Feasibility:**

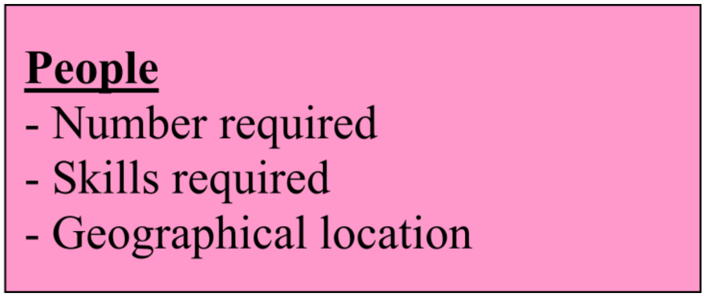
- After the scope is resolved, feasibility is addressed
- Software feasibility has four dimensions
  - Technology – Is the project technically feasible? Is it within the state of the art? Can defects be reduced to a level matching the application's needs?
  - Finance – Is is financially feasible? Can development be completed at a cost that the software organization, its client, or the market can afford?
  - Time – Will the project's time-to-market beat the competition?
  - Resources – Does the software organization have the resources needed to succeed in doing the project?

**Project Resources**:

**Resource Estimation**

- Three major categories of software engineering resources
  - People
  - Development environment
  - Reusable software components
    - Often neglected during planning but become a paramount concern during the construction phase of the software process
- Each resource is specified with
  - A description of the resource
  - A statement of availability
  - The time when the resource will be required
  - The duration of time that the resource will be applied

**Categories of Resources**:



**People**
- Number required
- Skills required
- Geographical location

**Development Environment**
- Software tools
- Computer hardware
- Network resources

**Reusable Software Components**
- Off-the-shelf components
- Full-experience components
- Partial-experience components
- New components

### Human Resources:

- Planners need to select the number and the kind of people skills needed to complete the project
- They need to specify the organizational position and job specialty for each person
- Small projects of a few person-months may only need one individual
- Large projects spanning many person-months or years require the location of the person to be specified also
- The number of people required can be determined only after an estimate of the development effort

### Development Environment Resources

- A software engineering environment (SEE) incorporates hardware, software, and network resources that provide platforms and tools to develop and test software work products
- Most software organizations have many projects that require access to the SEE provided by the organization
- Planners must identify the time window required for hardware and software and verify that these resources will be available

### Reusable Software Resources:

- Off-the-shelf components
  - Components are from a third party or were developed for a previous project
  - Ready to use; fully validated and documented; virtually no risk
- Full-experience components
  - Components are similar to the software that needs to be built

  - Software team has full experience in the application area of these components
  - Modification of components will incur relatively low risk
- Partial-experience components
  - Components are related somehow to the software that needs to be built but will require substantial modification
  - Software team has only limited experience in the application area of these components
  - Modifications that are required have a fair degree of risk
- New components
  - Components must be built from scratch by the software team specifically for the needs of the current project
  - Software team has no practical experience in the application area

Software development of components has a high degree of risk

## Q.3. Estimation of Project Cost and Effort.

### Factors Affecting Project Estimation:

- The accuracy of a software project estimate is predicated on
  - The degree to which the planner has properly estimated the size (e.g., KLOC) of the product to be built
  - The ability to translate the size estimate into human effort, calendar time, and money
  - The degree to which the project plan reflects the abilities of the software team
  - The stability of both the product requirements and the environment that supports the software engineering effort.
- Options for achieving reliable cost and effort estimates
  - Delay estimation until late in the project (we should be able to achieve 100% accurate estimates after the project is complete)
  - Base estimates on similar projects that have already been completed
  - Use relatively simple decomposition techniques to generate project cost and effort estimates
  - Use one or more empirical estimation models for software cost and effort estimation
- Option #1 is not practical, but results in good numbers
- Option #2 can work reasonably well, but it also relies on other project influences being roughly equivalent
- Options #3 and #4 can be done in tandem to cross check each other
- Decomposition techniques
  - These take a "divide and conquer" approach
  - Cost and effort estimation are performed in a stepwise fashion by breaking down a project into major functions and related software engineering activities
- Empirical estimation models

    - Offer a potentially valuable estimation approach if the historical data used to seed the estimate is good

**Decomposition Techniques:**

- Before an estimate can be made and decomposition techniques applied, the planner must
    - Understand the scope of the software to be built
    - Generate an estimate of the software's size
- Then one of two approaches are used
    - Problem-based estimation
        - Based on either source lines of code or function point estimates
    - Process-based estimation
        - Based on the effort required to accomplish each task

**Approaches to Software Sizing:**

- Function point sizing
    - Develop estimates of the information domain characteristics ( Product Metrics for Software)
- Standard component sizing
    - Estimate the number of occurrences of each standard component
    - Use historical project data to determine the delivered LOC size per standard component
- Change sizing
    - Used when changes are being made to existing software
    - Estimate the number and type of modifications that must be accomplished
    - Types of modifications include reuse, adding code, changing code, and deleting code
    - An effort ratio is then used to estimate each type of change and the size of the change

**Problem-Based Estimation:**

1) Start with a bounded statement of scope
2) Decompose the software into problem functions that can each be estimated individually
3) Compute an LOC or FP value for each function
4) Derive cost or effort estimates by applying the LOC or FP values to your baseline productivity metrics (e.g., LOC/person-month or FP/person-month)
5) Combine function estimates to produce an overall estimate for the entire project
6) In general, the LOC/pm and FP/pm metrics should be computed by project domain
    1) Important factors are team size, application area, and complexity
7) LOC and FP estimation differ in the level of detail required for decomposition with each value

1) For LOC, decomposition of functions is essential and should go into considerable detail (the more detail, the more accurate the estimate)
2) For FP, decomposition occurs for the five information domain characteristics and the 14 adjustment factors
    1) External inputs, external outputs, external inquiries, internal logical files, external interface files

**Reconciling Estimates**:

- The results gathered from the various estimation techniques must be reconciled to produce a single estimate of effort, project duration, and cost
- If widely divergent estimates occur, investigate the following causes
    - The scope of the project is not adequately understood or has been misinterpreted by the planner
    - Productivity data used for problem-based estimation techniques is inappropriate for the application, obsolete (i.e., outdated for the current organization), or has been misapplied
- The planner must determine the cause of divergence and then reconcile the estimates

**Q.4. Explain Empirical Estimation Models or COCOMO Model in details.**

- Estimation models for computer software use empirically derived formulas to predict effort as a function of LOC or FP
- Resultant values computed for LOC or FP are entered into an estimation model
- The empirical data for these models are derived from a limited sample of projects
    - Consequently, the models should be calibrated to reflect local software development conditions
- Stands for COnstructive COst MOdel
- Introduced by Barry Boehm in 1981 in his book “Software Engineering Economics”
- Became one of the well-known and widely-used estimation models in the industry
- It has evolved into a more comprehensive estimation model called COCOMO II
- COCOMO II is actually a hierarchy of three estimation models
- As with all estimation models, it requires sizing information and accepts it in three forms: object points, function points, and lines of source code
- **Application composition model** - Used during the early stages of software engineering when the following are important
    - Prototyping of user interfaces
    - Consideration of software and system interaction
    - Assessment of performance
    - Evaluation of technology maturity

- **Early design stage model** – Used once requirements have been stabilized and basic software architecture has been established
- **Post-architecture stage model** – Used during the construction of the software

**COCOMO Cost Drivers:**

- Personnel Factors
    - Applications experience
    - Programming language experience
    - Virtual machine experience
    - Personnel capability
    - Personnel experience
    - Personnel continuity
    - Platform experience
    - Language and tool experience
- Product Factors
    - Required software reliability
    - Database size
    - Software product complexity
    - Required reusability
    - Documentation match to life cycle needs
    - Product reliability and complexity
- Platform Factors
    - Execution time constraint
    - Main storage constraint
    - Computer turn-around time
    - Virtual machine volatility
    - Platform volatility
    - Platform difficulty
- Project Factors
    - Use of software tools
    - Use of modern programming practices
    - Required development schedule
    - Classified security application
    - Multi-site development
    - Requirements volatility
- It is often more cost effective to acquire rather than develop software
- Managers have many acquisition options
    - Software may be purchased (or licensed) off the shelf
    - "Full-experience" or "partial-experience" software components may be acquired and integrated to meet specific needs
    - Software may be custom built by an outside contractor to meet the purchaser's specifications
- The make/buy decision can be made based on the following conditions

  - Will the software product be available sooner than internally developed software?
  - Will the cost of acquisition plus the cost of customization be less than the cost of developing the software internally?
  - Will the cost of outside support (e.g., a maintenance contract) be less than the cost of internal support?

The COstructive COst Model (COCOMO) is the most widely used software estimation model in the world. IThe COCOMO model predicts the effort and duration of a project based on inputs relating to the size of the resulting systems and a number of "cost drives" that affect productivity.

**COCOMO Models:**

- COCOMO is defined in terms of three different models:
  - the Basic model,
  - the Intermediate model, and
  - the Detailed model.
- The more complex models account for more factors that influence software projects, and make more accurate estimates.

The Development mode:

- the most important factors contributing to a project's duration and cost is the Development Mode
  - **Organic Mode:** The project is developed in a familiar, stable environment, and the product is similar to previously developed products. The product is relatively small, and requires little innovation.
  - **Semidetached Mode:** The project's characteristics are intermediate between Organic and Embedded.
- the most important factors contributing to a project's duration and cost is the Development Mode:
  - **Embedded Mode:** The project is characterized by tight, inflexible constraints and interface requirements. An embedded mode project will require a great deal of innovation.

**LIMITATIONS OF COCOMO**

- COCOMO is used to estimate the cost and schedule of the project, starting from the design phase and till the end of integration phase. For the remaining phases a separate estimation model should be used.
- COCOMO is not a perfect realistic model. Assumptions made at the beginning may vary as time progresses in developing the project.
- When need arises to revise the cost of the project. A new estimate may show over budget or under budget for the project. This may lead to a partial development of the system, excluding certain requirements.
- COCOMO assumes that the requirements are constant throughout the development of the project; any changes in the requirements are not accommodated for calculation of cost of the project.
- There is not much difference between basic and intermediate COCOMO, except during the maintenance and development of the software project.
- COCOMO is not suitable for non-sequential ,rapid development, reengineering ,reuse cases models.

## Q.5. What is risk?

- Risks are potential problems that may affect successful completion of a software project.
- Risks involve uncertainty and potential losses.
- Risk analysis and management are intended to help a software team understand and manage uncertainty during the development process.

**Risk Strategies**:

- Reactive strategies
  - very common, also known as fire fighting
  - project team sets resources aside to deal with problems
  - team 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
  - team builds a plan to avoid risks if they can or to minimize risks if they turn into problems

Software Risks – Types:

- 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 from past project experience
- Unknown risks
    - some problems will simply occur without warning

**Risk Analysis**:

- Risk identification
- Risk projection
    - impact of risks/likelihood of risk actually happening
- Risk assessment
    - what will change if risk becomes problem
- Risk management

**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
        - customer characteristics
        - development environment
        - technology to be built

**Risk Projection**:

- The risk drivers affecting each risk component are
    - classified according to their impact category
    - potential consequences of each undetected software fault or unachieved project outcome are described

**Risk Estimation**:

1. Establish a scale indicating perceived likelihood of risk occurring
2. Determine consequences.
3. Estimate impact of consequences on project (for each risk).
4. Note overall accuracy of risk projection (to avoid misunderstandings).

**Risk Assessment:**

- Define referent levels for each project risk that can cause project termination

  - performance degradation
  - cost overrun
  - support difficulty
  - schedule slippage
- Attempt to develop a relationship between each risk triple (risk, probability, impact) and each of the reference levels**.**
- Predict the set of referent points that define a region of termination, bounded by a curve or areas of uncertainty.
- 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>).

**Q.6. Explain RMMM. (Risk Management, Risk Mitigation and Risk Monitoring).**

- 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
  - determining which risks caused which problems
- Risk Management
  - contingency planning
  - actions to be taken in the event that mitigation steps have failed and the risk has become a live problem

Risk Mitigation Example:

Risk: loss of key team members

- Determine causes of job turnover.
- Eliminate causes before project starts.
- After project starts, assume turnover is going to occur and work to ensure continuity.
- Make sure teams are organized and distribute information widely.
- Define documentation standards and be sure documents are produced in a timely manner.
- Conduct peer review of all work.
- Define backup staff.

- Alternative to RMMM 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

**The risk management process:**

Risk identification

- Identify project, product and business risks

Risk analysis

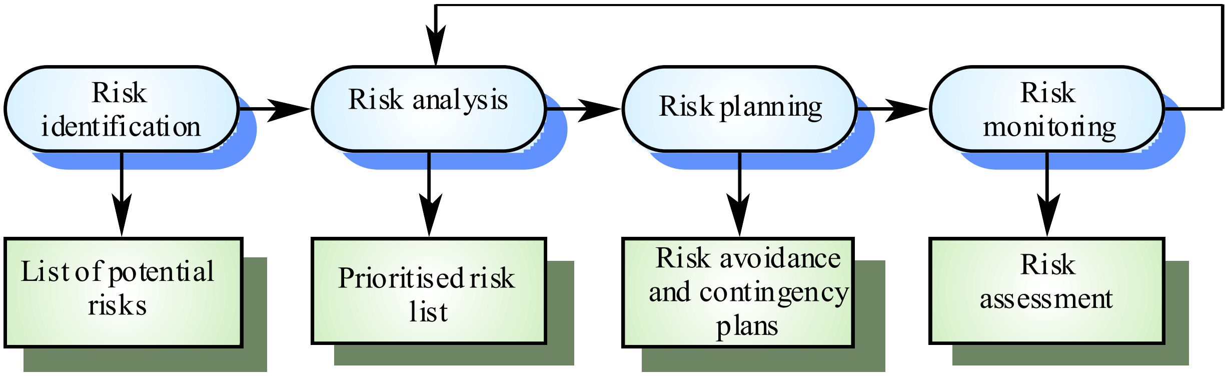
- Assess the likelihood and consequences of these risks

Risk planning

- Draw up plans to avoid or minimise the effects of the risk

Risk monitoring

- Monitor the risks throughout the project



**Risk identification:**

Technology risks

People risks

Organisational risks

Requirements risks

Estimation risks

| Risk type | Possible risks |
|---|---|
| Technology | The database used in the system cannot process as many transactions per second as expected.<br>Software components which should be reused contain defects which limit their functionality. |
| People | It is impossible to recruit staff with the skills required.<br>Key staff are ill and unavailable at critical times.<br>Required training for staff is not available. |
| Organisational | The organisation is restructured so that different management are responsible for the project.<br>Organisational financial problems force reductions in the project budget. |
| Tools | The code generated by CASE tools is inefficient.<br>CASE tools cannot be integrated. |
| Requirements | Changes to requirements which require major design rework are proposed.<br>Customers fail to understand the impact of requirements changes. |
| Estimation | The time required to develop the software is underestimated.<br>The rate of defect repair is underestimated.<br>The size of the software is underestimated. |

**Risk analysis:**

Assess probability and seriousness of each risk

Probability may be very low, low, moderate, high or very high

Risk effects might be catastrophic, serious, tolerable or insignificant

**Risk planning:**

Consider each risk and develop a strategy to manage that risk

Avoidance strategies

- The probability that the risk will arise is reduced

Minimisation strategies

- The impact of the risk on the project or product will be reduced

Contingency plans

- If the risk arises, contingency plans are plans to deal with that risk

**Risk monitoring:**

Assess each identified risks regularly to decide whether or not it is becoming less or more probable

Also assess whether the effects of the risk have changed

Each key risk should be discussed at management progress meetings.