MSS Module 4 Important topics

Software Project Management

Risk Managment

- Risk management involves anticipating risks that might affect the project schedule or the quality of the software being developed, and then taking action to avoid these risks

Types of Risks

- Project Risks
  - Affect the project schedule or resources. An example of a project risk is the loss of an experienced system architect.
  - Finding a replacement architect with appropriate skills and experience may take a long time; consequently, it will take longer to develop the software design than originally planned.
- Product risks
  - Affect the quality or performance of the software being developed
  - An example of a product risk is the failure of a purchased component to perform as expected.
- Business risks
  - Affect the organization developing or procuring the software.
  - For example, a competitor introducing a new product is a business risk.

Risk Management Process

- Risk identification
  - Risk identification is the first stage of the risk management process.
  - It is concerned with identifying the risks that could pose a major threat to the software engineering process.
- Risk analysis
  - During the risk analysis process, you have to consider each identified risk and make a judgment aboutthe probability and seriousness of that risk.
- Risk planning
  - The risk planning process develops strategies to manage the key risks that threaten the project.
- Risk monitoring
  - Risk monitoring is the process of checking that your assumptions about the product, process, and business risks have not changed

Managing people

- Consistency:
  - All the people in a project team should be treated in a comparable way.
- Respect:
  - Different people have different skills, and managers should respect these differences.
  - All members of the team should be given an opportunity to make a contribution.
- Inclusion:

    - People contribute effectively when they feel that others listen to them and take account of their proposals.
- Honesty:
    - As a manager, you should always be honest about what is going well and what is going badly in the team.
    - You should also be honest about your level of technical knowledge and be willing to defer to staff with more knowledge when necessary.

Team Work

- Teamwork in software development means that a group of people, ranging from a few to a lot, work together to create professional software.
- Because it's hard for everyone to work on one big problem at the same time, these large teams are divided into smaller groups. Each of these smaller groups has its own job in building the whole system.
- Being part of a cohesive group has some advantages:
    - **Establishing Quality Standards:**
    - **Learning and Support:**
    - **Sharing Knowledge:**
    - **Encouraging Improvement:**

Project Planning

- Project planning involves break down the work into parts and assign them to project team members, anticipate problems that might arise, and prepare tentative solutions to those problems
- **Stages of Planning:**
    - **Proposal Stage:**
    - **Project Startup Phase:**
    - **Periodic Updates Throughout the Project:**
- **Significance of Project Planning:**
    - **Anticipation of Issues:** Helps anticipate potential problems and prepare tentative solutions in advance.
    - **Resource Allocation:** Ensures efficient allocation of resources, both in terms of personnel and materials.
    - **Progress Assessment:** Serves as a tool to assess progress on the project and make informed decisions.

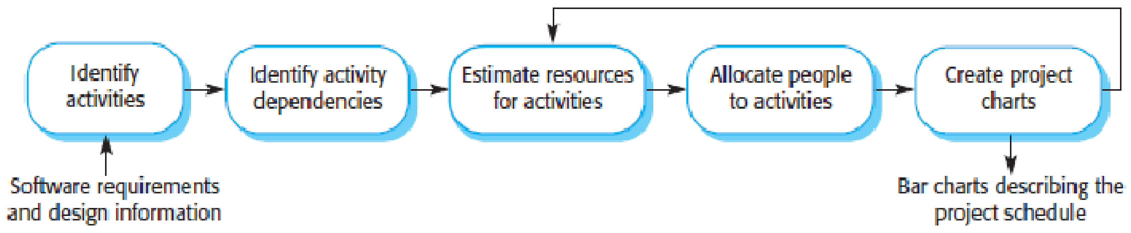
- **Communication:** Provides a clear roadmap for team members, stakeholders, and clients, enhancing communication and understanding.

Project scheduling, Agile planning

Project Scheduling

- Project scheduling is the process of deciding how the work in a project will be organized as separate tasks, and when and how these tasks will be executed.
- You estimate the calendar time needed to complete each task and the effort required, and you suggest who will work on the tasks that have been identified.
- You also have to estimate the hardware and software resources that are needed to complete each task

Project Scheduling Activities

- Scheduling in plan-driven projects involves breaking down the total work involved in a project into separate tasks and estimating the time required to complete each task.
- 
Identify activities
Identify activity dependencies
Estimate resources for activities
Allocate people to activities
Create project charts
Software requirements and design information
Bar charts describing the project schedule

Schedule Presentation

- Project schedules may simply be documented in a table or spreadsheet showing the tasks, estimated effort, duration, and task dependencies
- Two types of visualization
  - Calendar-based bar charts
  - Activity networks show the dependencies between the different activities

Milestones and deliverables:

- Milestone is a logical end to a stage of the project where the progress of the work can be reviewed.

- Each milestone should be documented by a brief report (often simply an email) that summarizes the work done and whether or not the work has been completed as planned.
- Milestones may be associated with a single task or with groups of related activities.

Agile Planning

- Agile methods of software development are iterative approaches where the software is developed and delivered to customers in increments.

Stages

- Release planning
    - Looks ahead for several months and decides on the features that should be included in a release of a system
- Iteration planning
    - shorter term outlook and focuses on planning the next increment of a system. This usually represents 2 to 4 weeks of work for the team

Advantages

- This approach to planning has the advantage that a software increment is always delivered at the end of each project iteration.
- If the features to be included in the increment cannot be completed in the time allowed, the scope of the work is reduced.
- The delivery schedule is never extended.

Disadvantages

- A major difficulty in agile planning is that it relies on customer involvement and availability.
- This involvement can be difficult to arrange, as customer representatives sometimes have to prioritize other work and are not available for the planning game.
- Furthermore, some customers may be more familiar with traditional project plans and may find it difficult to engage in an agile planning process.

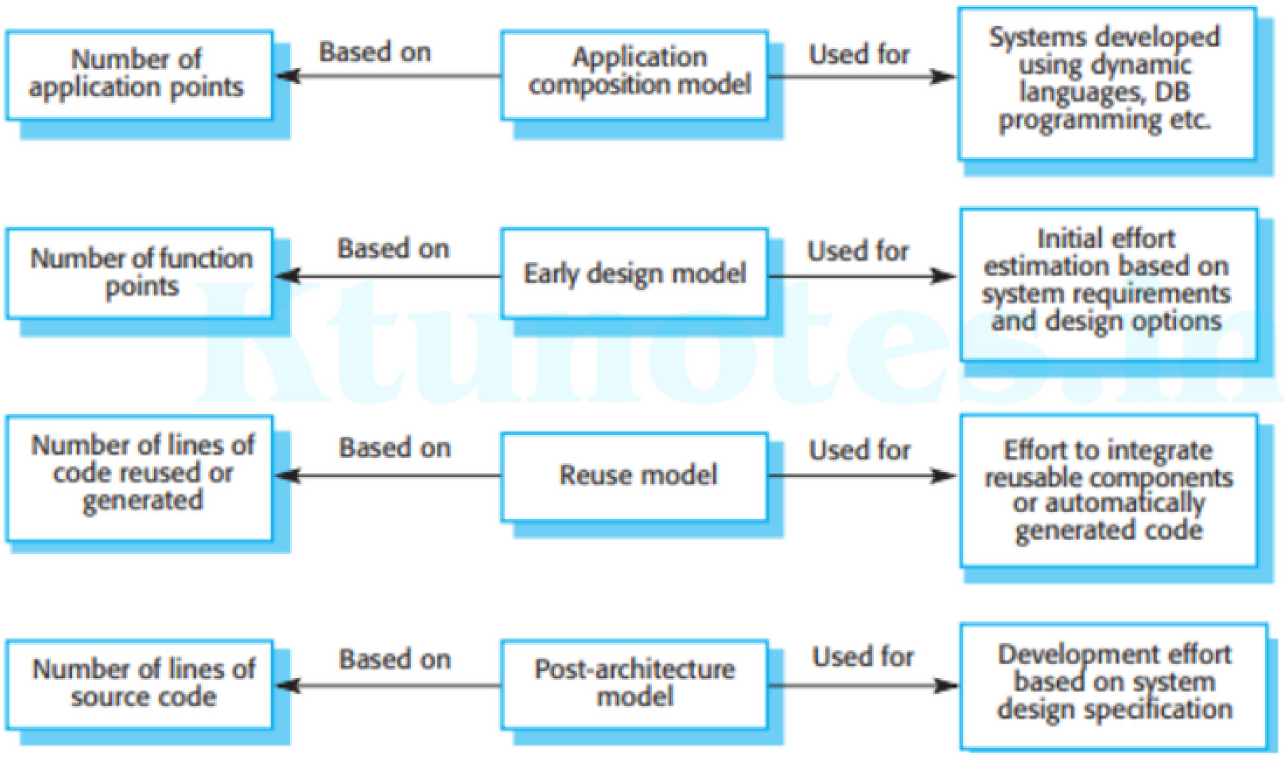
COCOMO cost modeling

- The best known algorithmic cost modeling technique and tool is the COCOMO II model.
- This empirical model was derived by collecting data from a large number of software projects of different sizes.

- These data were analyzed to discover the formulas that were the best fit to the observations.
- These formulas linked the size of the system and product, project, and team factors to the effort to develop the system.
- COCOMO II is a freely available model that is supported with open-source tools.

COCOMO Submodels

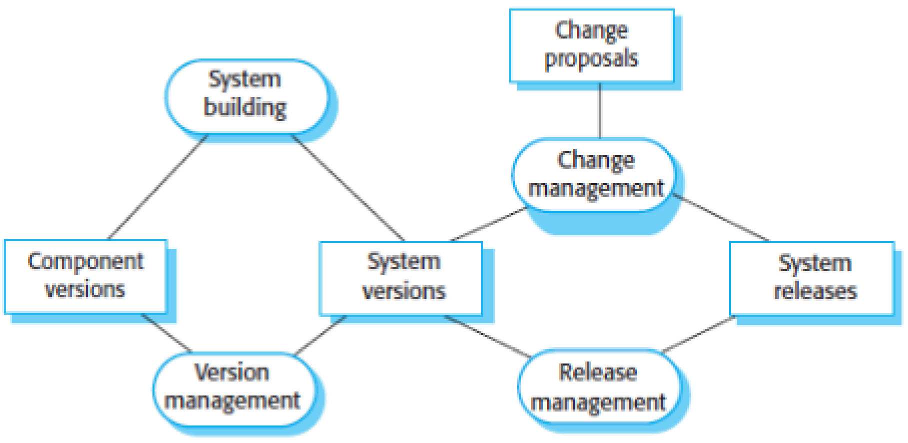
- An application composition model:
  - This models the effort required to develop systems that are created from reusable components, scripting, or database programming.
  - Software size estimates are based on application points, and a simple size/productivity formula is used to estimate the effort required.
- An early design model:
  - This model is used during early stages of the system design after the requirements have been established.
  - The estimate is based on the standard estimation formula
- A reuse model
  - This model is used to compute the effort required to integrate reusable components and/or automatically generated program code. It is normally used in conjunction with the post-architecture model.
- A post-architecture model
  - Once the system architecture has been designed, a more accurate estimate of the software size can be made.
  - Again, this model uses the standard formula for cost estimation discussed above.


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

COCOMO Estimation Models

Configuration management

- Configuration management (CM) is concerned with the policies, processes, and tools for managing changing software systems.
- The configuration management of a software system product involves four closely related activities:



- [illegible]
  - Version Control
    - This involves keeping track of the multiple versions of system components and ensuring that changes made to components by different developers do not interfere with each other
  - System building
    - This is the process of assembling program components, data, and libraries, then compiling and linking these to create an executable system
  - Change management
    - This involves keeping track of requests for changes to delivered software from customers and developers,
  - Release management
    - This involves preparing software for external release and keeping track of the system versions that have been released for customer use.

Version management

- Version management is the process of keeping track of different versions of software components and the systems in which these components are used
- It also involves ensuring that changes made by different developers to these versions do not interfere with each other.
- Version control (VC) systems identify, store, and control access to the different versions of components.
- There are 2 types of modern version control system
  - Centralized systems,
    - Where a single master repository maintains all versions of the software components that are being developed.Subversion is a widely used example of a

    centralized VC system.
  - Distributed systems,
    - where multiple versions of the component repository exist at the same time.

Storage Management

- When version control systems were first developed, storage management was one of their most important functions.
- Disk space was expensive, and it was important to minimize the disk space used by the different copies of components.
- Instead of keeping a complete copy of each version, the system stores a list of differences (deltas) between one version and another.
- When a new version is created, the system simply stores a delta, a list of differences, between the new version and the older version used to create that new version.
- Deltas are usually stored as lists of changed lines, and, by applying these automatically, one version of a component can be created from another.

Agile software management

SCRUM

- Scrum is an agile software development
- Within each process activities task work within a process pattern called sprint
- Scrums emphasize the use of set of software patterns that have been proven effective for projects with tight timelines, changing requirements and business criticality
- Each of these process patterns define a set of development activities:
  - Backlog
    - a prioritized list of project requirements or features that provide business value for the customer. The project manager access backlogs and updates are made
  - Sprints:
    - consists of work units that are required to achieve a requirement defined in the backlog that must be fit into a predefined time box
  - Scrum meetings:
    - Short meetings held daily by the scrum team.
    - Three questions are asked and answered by team members: what did you do since last meeting? what obstacles are you encountering? What do you plan to accomplish by next meeting?

    - A team leader called Scrum Master leads the meeting and assess each responses.
    - This meeting helps to uncover potential problems as early as possible.
  - Demos
    - Deliver the software increment to the customer so that functionality that has been implemented can be demonstrated and evaluated by the customer.

Kanban methodology and lean approaches

- 1. Eliminate waste:
  - Lean philosophy regards everything not adding value to the customer as waste. Such waste may include: Partially done work, Extra features, Relearning, Task switching, Waiting, Handoffs, Defects, Management activities.
- 2. Amplify Learning
  - Software development is a continuous learning process based on iterations when writing code.
  - Software value is measured in fitness for use and not in conformance to requirements. Instead of adding more documentation or detailed planning, different ideas could be tried by writing code and building
- 3. Decide as late as possible:
  - delaying decisions as much as possible until they can be made based on facts and not on uncertain assumptions and predictions.
- 4. Deliver as fast as possible
  - In the era of rapid technology evolution, it is not the biggest that survives, but the fastest.
  - The sooner the end product is delivered without major defects, the sooner feedback can be received, and incorporated into the next iteration.
- 5. Empower the team
  - The lean approach follows the agile principle build projects around motivated individuals and trust them to get the job done
- 6. Build integrity in
  - The customer needs to have an overall experience of the system. This is the so-called perceived integrity:
- 7. Optimize the whole