INFORMATION SYSTEM
BCA-3RD YEAR
UNIT-6TH

Introduction to Software Markets: The **software market** refers to the ecosystem where software products and services are developed, sold, and consumed. It is a dynamic environment driven by technology trends, user demand, and business requirements. Understanding software markets is essential for planning, budgeting, and executing successful software projects.

1. Types of Software Markets

a) Custom Software Market

- Software is **built to order** for a specific client or organization.
- Designed to meet **unique business needs**.
- Examples: ERP systems for a company, banking software, hospital management systems.
- **Characteristics:** High customization, higher cost, longer development time.

b) Packaged Software Market

- Ready-made software sold to a **wide audience**.
- Examples: Microsoft Office, Adobe Photoshop, antivirus software.
- **Characteristics:** Standardized features, lower cost per user, quicker deployment.

c) Open-Source Software Market

- Software whose source code is **freely available** for use, modification, and distribution.
- Examples: Linux, Apache, WordPress.
- **Characteristics:** Community-driven development, free or low-cost, rapid innovation.

d) Software as a Service (SaaS) Market

- Software delivered **over the internet** as a service.
- Examples: Salesforce, Google Workspace, Zoom.
- **Characteristics:** Subscription-based, platform-independent, continuously updated.

2. Trends in Software Markets

- **Cloud Computing:** Shift from installed software to cloud-hosted applications.
- **Mobile and Web Applications:** Increased demand for apps on mobile devices and web platforms.
- **AI and Automation:** Growing market for AI-enabled software and automation tools.
- **Cybersecurity Software:** Rising demand due to data breaches and privacy concerns.

- **Open-Source Adoption:** Enterprises increasingly adopting open-source solutions.

3. Factors Influencing Software Markets

1. **User Demand:** Businesses and consumers drive what features and types of software are developed.
2. **Technology Advances:** Emerging technologies like AI, IoT, and blockchain create new software markets.
3. **Competition:** Presence of many software vendors influences pricing, features, and quality.
4. **Regulations:** Compliance requirements (e.g., GDPR) affect software design and marketability.
5. **Globalization:** Software can now be distributed globally, expanding market reach.

4. Implications for Software Project Management

- Market demand influences **project scope and priorities**.
- Understanding the market helps in **estimating costs and timelines**.
- Competitive markets require **high-quality, reliable software**.
- Helps managers **plan resources, select development methodology, and manage risks** effectively.

Software Market Type	Key Characteristics	Examples
Custom Software	Tailored, high cost, long development	ERP systems, banking apps
Packaged Software	Standardized, mass-market	Microsoft Office, Photoshop
Open-Source Software	Free, modifiable, community-driven	Linux, Apache
SaaS	Online service, subscription-based	Salesforce, Google Workspace

Planning of Software Projects

Software project planning is the process of defining the project's **scope, objectives, resources, schedule, and risks** to ensure successful completion. Effective planning reduces uncertainty, improves resource allocation, and helps meet deadlines and budget constraints.

1. Importance of Project Planning

- Provides a **roadmap** for the software development process.
- Helps in **estimating time, cost, and resources** accurately.
- Identifies potential **risks and dependencies**.
- Enables **progress monitoring and control** throughout the project lifecycle.
- Improves **communication among team members and stakeholders**.

2. Key Activities in Software Project Planning

a) Requirement Analysis

- Understand the **customer's needs** and expectations.
- Define the **functional and non-functional requirements**.
- Clarify scope to avoid **scope creep** during development.

b) Defining Project Scope

- Specify what is **included and excluded** from the project.
- Clearly outline **deliverables, objectives, and constraints**.
- Establish measurable **success criteria** for the project.

c) Resource Planning

- Identify required **human resources**, skills, and roles.
- Determine **hardware, software, and tools** needed.
- Allocate responsibilities and define **team structure**.

d) Estimating Effort, Time, and Cost

- Use **size metrics** like Lines of Code (LOC) or Function Points (FP).
- Apply **cost estimation models** (e.g., COCOMO).
- Estimate **project duration and effort per activity**.

e) Risk Assessment

- Identify potential risks such as **technical challenges, schedule delays, or budget overruns**.
- Plan **risk mitigation strategies** for high-priority risks.

f) Defining Milestones and Schedule

- Break the project into **tasks and subtasks**.
- Use tools like **Gantt Charts, PERT, and CPM** to schedule tasks.
- Define milestones to track progress at key points in the project.

g) Quality Planning

- Define **standards and metrics** for software quality.
- Plan for **reviews, testing, and validation activities**.

h) Documentation

- Prepare a **Project Plan Document** including scope, resources, schedule, and risk plan.
- Maintain **regular updates** as the project evolves.

3. Tools for Project Planning

- **Gantt Chart Tools:** Microsoft Project, Smartsheet, TeamGantt
- **Project Tracking Tools:** JIRA, Asana, Trello
- **Estimation Tools:** COCOMO calculators, Function Point Analysis tools

4. Benefits of Proper Planning

- Reduces **project risks** and surprises.
- Improves **team coordination and productivity**.
- Provides **realistic schedules and budgets**.

- Ensures **timely delivery of quality software**.
- Facilitates **stakeholder communication and transparency**.

Summary Table

Planning Activity	Description
Requirement Analysis	Understanding customer needs and defining requirements
Project Scope Definition	Establishing boundaries and deliverables
Resource Planning	Identifying people, skills, and tools required
Effort, Time, and Cost Estimation	Estimating project size, duration, and budget
Risk Assessment	Identifying and mitigating potential risks
Scheduling & Milestones	Sequencing tasks and defining checkpoints
Quality Planning	Setting standards, metrics, and review processes
Documentation	Maintaining a project plan and updates

Size and Cost Estimation in Software Projects

Estimating the **size and cost** of a software project is a critical step in planning. Accurate estimates help in budgeting, scheduling, resource allocation, and risk management.

1. Importance of Size and Cost Estimation

- Provides a **basis for project budgeting**.
- Helps in **scheduling tasks and allocating resources**.
- Assists in **risk management** by identifying potential overruns.
- Supports **performance tracking and control** during development.

2. Software Size Estimation

Software size indicates the **amount of work or complexity** in a software project. It is a key input for cost and effort estimation.

Common Size Metrics

Metric	Description	Advantages	Limitations
Lines of Code (LOC)	Total number of executable lines in the code	Simple to measure; widely used	Language dependent; ignores functionality complexity
Function Points (FP)	Measures software functionality from the user's perspective	Technology independent; measures functional complexity	Requires expertise to calculate; subjective in some cases
Object Points	Used in OO systems; counts screens, reports, and modules	Suitable for OO design	Less common; requires detailed design

3. Cost Estimation Techniques

a) Expert Judgment

- Relies on the experience and intuition of project managers or domain experts.
- Useful when historical data is limited.

b) Analogous Estimation

- Compares the current project with **similar past projects**.
- Quick and simple but less accurate for unique projects.

c) Algorithmic/Parametric Models

- Use mathematical formulas to estimate effort and cost.
- **COCOMO (Constructive Cost Model)** is a widely used model:
  - **Basic COCOMO:** Effort = a × (Size)^b (where Size is in KLOC)
  - **Intermediate & Detailed COCOMO:** Include factors like complexity, reliability, team experience.

d) Bottom-Up Estimation

- Estimates **cost of individual modules/tasks** and sums them.
- More accurate but **time-consuming**.

4. Steps for Size and Cost Estimation

1. **Define Scope and Requirements:** Understand what the software will do.
2. **Choose Size Metrics:** LOC, Function Points, or Object Points.
3. **Measure or Estimate Size:** Based on requirements and design documents.
4. **Select Estimation Model:** Expert judgment, COCOMO, or bottom-up.
5. **Calculate Effort and Cost:** Convert size into person-months and monetary cost.
6. **Adjust for Risk and Complexity:** Factor in uncertainty, team experience, and technical challenges.

5. Factors Affecting Cost Estimation

- **Project Size:** Larger projects require more resources and effort.
- **Complexity:** More complex algorithms or systems increase development time.
- **Team Skill:** Experienced teams can reduce effort and cost.
- **Tools and Technology:** Availability of CASE tools, libraries, and frameworks affects cost.
- **Quality Requirements:** Higher quality or reliability increases cost.

6. Benefits of Accurate Estimation

- Prevents **budget overruns** and schedule delays.
- Enables **realistic project planning**.
- Improves **stakeholder confidence**.
- Supports **risk management** and contingency planning.

Estimation Aspect	Key Points
Size Metrics	LOC, Function Points, Object Points
Cost Estimation Techniques	Expert judgment, Analogous, Algorithmic (COCOMO), Bottom-Up

Estimation Aspect	Key Points
Estimation Steps	Define scope → Choose size metric → Measure size → Select model → Calculate cost → Adjust for risk
Influencing Factors	Project size, complexity, team skill, tools, quality requirements
Benefits	Budget control, realistic planning, risk management, stakeholder confidence

Project Scheduling in Software Projects

Project scheduling is the process of planning **when and in what sequence tasks will be completed** to ensure the timely delivery of a software project. It is a key aspect of project management and helps in **resource allocation, milestone tracking, and progress monitoring**.

1. Importance of Project Scheduling

- Ensures **timely completion** of software projects.
- Helps in **allocating resources effectively**.
- Allows managers to **track progress and identify delays**.
- Facilitates **coordination among team members**.
- Provides a basis for **risk management** and contingency planning.

2. Steps in Project Scheduling

a) Task Identification

- Break down the project into **manageable tasks and sub-tasks**.
- Identify dependencies between tasks (which tasks must be done first).

b) Task Sequencing

- Arrange tasks in the **logical order of execution**.
- Identify **critical tasks** that directly impact project completion.

c) Resource Assignment

- Assign **team members, tools, and equipment** to tasks.
- Consider **skill levels, availability, and workload**.

d) Time Estimation

- Estimate **duration of each task**.
- Use historical data or expert judgment for accuracy.

e) Developing the Schedule

- Use scheduling techniques to **visualize and plan timelines**.
- Identify **start and end dates** for each task.
- Define **milestones** to track progress.

f) Monitoring and Updating

- Track **actual progress vs planned schedule**.
- Update the schedule for **delays, resource changes, or scope modifications**.

3. Project Scheduling Techniques

Technique	Description
Gantt Charts	Visual bar chart showing tasks vs time; highlights milestones and dependencies.
PERT (Program Evaluation and Review Technique)	Uses probabilistic time estimates (optimistic, pessimistic, most likely) to identify expected task duration.
CPM (Critical Path Method)	Determines the **longest path of dependent tasks** to calculate minimum project duration; identifies critical and non-critical tasks.
Milestone Charts	Focuses on **key deliverables or checkpoints** rather than all tasks.
Resource Leveling	Adjusts the schedule to **balance workloads** across team members and resources.

4. Critical Path in Scheduling

- **Critical Path:** Sequence of tasks that **directly affects project completion time**.
- Any delay in a critical path task delays the **entire project**.
- Non-critical tasks have **slack time**, allowing some flexibility.

5. Tools for Project Scheduling

- **Microsoft Project:** Comprehensive project management and Gantt charts.
- **JIRA:** Agile-friendly scheduling with sprint planning.
- **Trello / Asana:** Task boards with deadlines and progress tracking.
- **Primavera P6:** Enterprise-grade scheduling for large projects.

6. Benefits of Effective Scheduling

- Ensures **on-time project delivery**.
- Improves **team productivity** and coordination.
- Identifies **potential bottlenecks** early.
- Facilitates **budget and resource management**.
- Enhances **stakeholder communication and confidence**.

Step / Technique	Key Points
Task Identification	Break project into tasks & subtasks
Task Sequencing	Define order & dependencies
Resource Assignment	Allocate people, tools, and equipment
Time Estimation	Estimate task durations
Schedule Development	Visualize tasks, start/end dates, milestones
Monitoring & Updating	Track progress and revise schedule
Scheduling Techniques	Gantt, PERT, CPM, Milestone charts, Resource leveling

Step / Technique	Key Points
Benefits	Timely delivery, productivity, risk management, coordination

Measurement of Software Quality and Productivity

Measuring software quality and productivity is essential for ensuring that a software project meets **user expectations, business goals, and technical standards**. It also helps managers track performance, identify issues, and improve development processes.

1. Software Quality

Software quality refers to the degree to which a software product meets **specified requirements, user needs, and reliability standards**.

Key Aspects of Software Quality

Aspect	Description
Functionality	Correctness, completeness, and suitability of software functions.
Reliability	Ability to perform consistently under expected conditions (e.g., mean time to failure).
Usability	Ease of learning, use, and user satisfaction.
Efficiency	Resource utilization, response time, and throughput.
Maintainability	Ease of modifying, correcting, or upgrading the software.
Portability	Ability to run on different platforms and environments.

Common Quality Metrics

1. **Defect Density**: Number of defects per unit size (e.g., per 1000 lines of code).
2. **Mean Time to Failure (MTTF)**: Average time the software operates without failure.
3. **Mean Time to Repair (MTTR)**: Average time required to fix a defect.
4. **Customer Satisfaction Index**: Feedback from users on usability and performance.
5. **Requirements Coverage**: Percentage of requirements correctly implemented.

2. Software Productivity

Software productivity measures how efficiently software development resources are used to produce deliverables.

Common Productivity Metrics

Metric	Description
Lines of Code (LOC) per Person-Month	Amount of code produced per developer per month.
Function Points (FP) per Person-Month	Amount of functional output delivered per developer per month; language-independent.
Effort Variance	Difference between estimated and actual effort.

Metric	Description
Defect Removal Efficiency (DRE)	Percentage of defects found and fixed before release.
Schedule Adherence	Comparison of planned vs actual schedule completion.

Factors Affecting Productivity

- Team experience and skill levels.
- Complexity and size of the project.
- Tools and development environments (IDEs, CASE tools).
- Development methodology (Agile, Waterfall, OO).
- Quality of requirements and design.

3. Techniques for Measuring Quality and Productivity

1. **Code Reviews and Inspections**
   - Peer review of code to detect defects early.
2. **Testing Metrics**
   - Track **unit tests, integration tests, and system tests**.
   - Use **test coverage metrics** to ensure thorough testing.
3. **Process Metrics**
   - Measure productivity trends across the development lifecycle.
   - Examples: effort per module, defect rate per phase.
4. **Benchmarking**
   - Compare productivity and quality with **similar past projects**.

4. Importance of Measurement

- Ensures software meets **quality standards**.
- Helps in **estimating future projects** more accurately.
- Supports **process improvement** through lessons learned.
- Enables **stakeholder confidence** and accountability.
- Reduces cost of **late defect detection and correction**.

Aspect	Key Metrics / Techniques	Purpose
Quality	Defect density, MTTF, MTTR, requirements coverage	Ensure software meets specifications and user needs
Productivity	LOC/FP per person-month, DRE, schedule adherence	Measure efficiency of development resources
Techniques	Code reviews, testing metrics, process metrics, benchmarking	Improve quality and productivity continuously
Importance	Early defect detection, process improvement, cost reduction	Deliver reliable and efficient software

ISO and Capability Maturity Models (CMM) in Software Projects

Managing software projects effectively requires **standardized processes and quality assurance models**. ISO standards and CMM provide frameworks for **process improvement, quality management, and organizational growth**.

1. ISO Standards for Software Projects

ISO (International Organization for Standardization) provides globally recognized standards to ensure **software quality, process consistency, and documentation**.

a) Key ISO Standards in Software

Standard	Purpose / Description
ISO 9001	Quality management systems; ensures processes are **consistent and meet customer expectations**.
ISO/IEC 12207	Defines **software life cycle processes**, including planning, development, operation, and maintenance.
ISO/IEC 9126 / 25010	Software product quality model; includes **functionality, reliability, usability, efficiency, maintainability, portability**.
ISO/IEC 15504 (SPICE)	Software process improvement and capability determination.
ISO/IEC 27001	Information security management for software systems.

b) Importance of ISO Standards

- Standardizes processes across projects and teams.
- Ensures **high-quality software products**.
- Facilitates **audits and compliance**.
- Reduces **defects and rework**, saving cost and time.
- Supports **continuous improvement** in processes and products.

2. Capability Maturity Model (CMM)

CMM is a framework to **assess and improve the maturity of an organization's software processes**. It helps organizations achieve **predictable, high-quality software development**.

a) Purpose of CMM

- Evaluate the **current maturity of software processes**.
- Provide **guidelines for process improvement**.
- Enhance **quality, productivity, and reliability** of software projects.

b) CMM Maturity Levels

Level	Name	Description
1	**Initial**	Processes are **ad hoc and chaotic**; success depends on individual effort.
2	**Repeatable**	Basic project management practices established; projects can be **repeated** successfully.

Level	Name	Description
3	**Defined**	Processes are **standardized and documented** across the organization.
4	**Managed**	Processes are **measured, controlled, and monitored**; data-driven management.
5	**Optimizing**	Focus on **continuous process improvement** using feedback and innovation.

c) Benefits of CMM

- Increases **process consistency and predictability**.
- Improves **software quality and productivity**.
- Reduces **project risk and cost overruns**.
- Helps organizations achieve **long-term growth and competitive advantage**.

3. Relationship Between ISO and CMM

- **ISO** focuses on **standards and quality assurance**, providing a **baseline for processes**.
- **CMM** focuses on **process maturity**, helping organizations **improve processes over time**.
- Both complement each other: ISO ensures compliance and quality, while CMM drives **continuous improvement and capability development**.

Aspect	Key Points
ISO Standards	ISO 9001, ISO/IEC 12207, ISO/IEC 25010; standardize quality and processes
Purpose of ISO	Ensure process consistency, quality assurance, and compliance
CMM	Framework to assess and improve software process maturity
CMM Levels	1-Initial, 2-Repeatable, 3-Defined, 4-Managed, 5-Optimizing
Benefits	Improved quality, productivity, risk management, process consistency, and organizational growth
ISO + CMM Relationship	ISO ensures standard compliance; CMM drives continuous improvement and capability growth