BRAINWARE UNIVERSITY

[PCC-CSM701] [Software Engineering]

MODULE-1

Introduction to Software Engineering

Software is more than just a program code. A program is an executable code, which serves some computational purpose. Software is considered to be collection of executable programming code, associated libraries and documentations. Software, when made for a specific requirement is called **software product.**

Engineering on the other hand, is all about developing products, using well-defined, scientific principles and methods.

Software engineering is an engineering branch associated with development of software product using well-defined scientific principles, methods and procedures. The outcome of software engineering is an efficient and reliable software product.

Software Evolution

The process of developing a software product using software engineering principles and methods is referred to as **software evolution.** This includes the initial development of software and its maintenance and updates, till desired software product is developed, which satisfies the expected requirements.

Evolution starts from the requirement gathering process. After which developers create a prototype of the intended software and show it to the users to get their feedback at the early stage of software product development. The users suggest changes, on which several consecutive updates and maintenance keep on changing too. This process changes to the original software, till the desired software is accomplished.

Even after the user has desired software in hand, the advancing technology and the changing requirements force the software product to change accordingly. Re-creating software from scratch and to go one-on-one with requirement is not feasible. The only feasible and economical solution is to update the existing software so that it matches the latest requirements.

The Evolving Role of Software

Software is everywhere – it runs smartphones, ATMs, airplanes, and even home appliances. Initially, software was developed mainly for scientific and business applications. Today, it powers:

- Embedded systems (e.g., washing machines, smart TVs)

- Enterprise applications (e.g., ERP software in companies)
- Web apps (e.g., Gmail, Amazon)
- Mobile apps (e.g., Google Maps, WhatsApp)
- AI & ML systems (e.g., ChatGPT, recommendation engines)

Example: Tesla vehicles use complex software for autopilot features, entertainment systems, and remote updates, showing how critical software has become.

Changing Nature of Software

Software has evolved in complexity and functionality. The major changes include:

- Global reach
- Always-on availability
- Data-driven systems
- Security and privacy concerns

Example: Banking apps now offer real-time updates, fingerprint authentication, and AI chatbots.

Software Myths

Management Myths:
- Myth: Adding more people to a late project helps.
  Reality: It often causes more delays.

Customer Myths:
- Myth: Requirements can be frozen early.
  Reality: Requirements evolve over time.

Developer Myths:
- Myth: Once the program works, it's done.
  Reality: Maintenance and updates are ongoing.

Example: WhatsApp continues to be updated regularly even after years of release.

A Generic View of Process

A software process includes the set of activities to develop software:
1. Communication
2. Planning

3. Modeling
4. Construction
5. Deployment

Example: Developing a college ERP system follows all these stages.

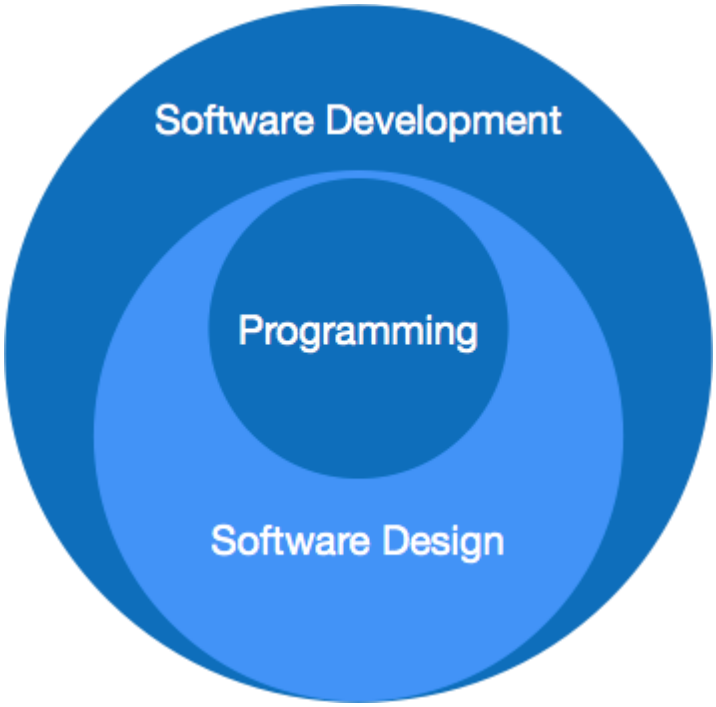
Software Engineering – A Layered Technology

Four layers of software engineering:
1. Quality Focus
2. Process
3. Methods
4. Tools

Example: A team uses Agile process, UML for design, and tools like GitHub.

Software Paradigms

Software paradigms refer to the methods and steps, which are taken while designing the software. There are many methods proposed and are in work today, but we need to see where in the software engineering these paradigms stand. These can be combined into various categories, though each of them is contained in one another:



Programming paradigm is a subset of Software design paradigm which is further a subset of Software development paradigm.

Software Development Paradigm

This Paradigm is known as software engineering paradigms where all the engineering concepts pertaining to the development of software are applied. It includes various researches and requirement gathering which helps the software product to build. It consists of –

- Requirement gathering
- Software design
- Programming

Software Design Paradigm

This paradigm is a part of Software Development and includes –

- Design

- Maintenance
- Programming

Programming Paradigm

This paradigm is related closely to programming aspect of software development. This includes

- Coding
- Testing
- Integration

Need of Software Engineering

The need of software engineering arises because of higher rate of change in user requirements and environment on which the software is working.

- **Large software -** It is easier to build a wall than to a house or building, likewise, as the size of software become large engineering has to step to give it a scientific process.
- **Scalability-** If the software process were not based on scientific and engineering concepts, it would be easier to re-create new software than to scale an existing one.
- **Cost-** As hardware industry has shown its skills and huge manufacturing has lower down the price of computer and electronic hardware. But the cost of software remains high if proper process is not adapted.
- **Dynamic Nature-** The always growing and adapting nature of software hugely depends upon the environment in which user works. If the nature of software is always changing, new enhancements need to be done in the existing one. This is where software engineering plays a good role.
- **Quality Management-** Better process of software development provides better and quality software product.

Characteristics of good software

A software product can be judged by what it offers and how well it can be used. This software must satisfy on the following grounds:

- Operational
- Transitional
- Maintenance

Well-engineered and crafted software is expected to have the following characteristics:

Operational

This tells us how well software works in operations. It can be measured on:

- Budget
- Usability
- Efficiency
- Correctness
- Functionality
- Dependability
- Security
- Safety

Transitional

This aspect is important when the software is moved from one platform to another:

- Portability
- Interoperability
- Reusability
- Adaptability

Maintenance

This aspect briefs about how well a software has the capabilities to maintain itself in the ever-changing environment:

- Modularity
- Maintainability
- Flexibility
- Scalability

In short, Software engineering is a branch of computer science, which uses well-defined engineering concepts required to produce efficient, durable, scalable, in-budget and on-time software products.

Process Framework

Generic framework activities:
- Project tracking
- Risk management
- Quality assurance
- Configuration management

Example: Developing an e-commerce site involves tracking changes, risks, and quality checks.

Capability Maturity Model Integration (CMMI)

What is CMMI?

CMMI (Capability Maturity Model Integration) is a process level improvement model developed by the Software Engineering Institute (SEI) at Carnegie Mellon University. It provides organizations with the essential elements for effective process improvement and quality assurance.

Goal: To help organizations improve performance by developing effective processes across projects and business functions.

Use Cases:
- Software development
- Systems engineering

- Product manufacturing
- Service delivery

CMMI Structure

CMMI is structured into two representations:

1. Staged Representation:
- Focuses on achieving maturity levels.
- Best for organizations aiming to improve overall performance step-by-step.

2. Continuous Representation:
- Focuses on capability levels for individual processes.
- Allows improvement in specific areas (like testing or project management).

CMMI Maturity Levels (Staged Representation)

CMMI defines five maturity levels of process maturity:

Level 1 – Initial

Characteristics: Unpredictable, poorly controlled, and reactive.
Processes are ad hoc and chaotic. Success depends on individual heroics.
Example: A startup with no defined process. Everyone codes as they like. Quality is inconsistent.

Level 2 – Managed

Processes are planned, documented, and followed at the project level.
Focus on project management and basic quality control. Requirements and commitments are tracked.
Example: Project plans and milestones are tracked in tools like Trello or Jira. Issues are documented.

Level 3 – Defined

Organization-wide standard processes are used and tailored for projects.
Focus on documentation, training, and consistency.

Processes are proactive, not reactive.
Example: All teams follow an internal SDLC standard. Every project follows the same structure.

Level 4 – Quantitatively Managed

Processes are measured and controlled using quantitative data.
Statistical techniques are used to control quality and performance.
Example: The organization tracks defect density, test coverage, and code quality metrics.

Level 5 – Optimizing

Focus on continuous process improvement.
Innovations and best practices are implemented. Lessons learned are fed back into process improvements.
Example: A company conducts regular retrospectives, applies machine learning to detect process bottlenecks.

Key Process Areas by Level

Level 2 – Managed: Requirements Management, Project Planning, Configuration Management
Level 3 – Defined: Organizational Process Focus, Training, Integrated Software Management
Level 4 – Quantitatively Managed: Quantitative Project Management, Process Performance Analysis
Level 5 – Optimizing: Organizational Innovation, Causal Analysis and Resolution

Benefits of CMMI

- Standardized processes across projects
- Predictable project outcomes (cost, schedule, quality)
- Improved product quality
- Better risk and project management
- Competitive advantage in tenders/contracts

Real-Life Example

Example: Infosys follows CMMI Level 5 practices. When they start a new client project:
- They already have templates, guidelines, and training ready (Level 3).
- They track all metrics like delivery time, bug rate (Level 4).
- They regularly update practices with new tools and client feedback (Level 5).

Comparison Summary of Levels

Level 1 – Initial: Ad hoc, unpredictable
Level 2 – Managed: Basic planning, tracking
Level 3 – Defined: Consistency across teams
Level 4 – Quantitatively Managed: Data-driven decisions
Level 5 – Optimizing: Innovation and feedback loops

Example: A startup may begin at Level 1 and progress to Level 5 by standardizing and improving processes.

Process Patterns

Types of patterns:
- Stage Patterns
- Task Patterns
- Phase Patterns

Example: 'Prototype then build' is a phase pattern used in many startups.

Process Assessment

Involves evaluating software process maturity using models like:
- CMMI
- SPICE (ISO/IEC 15504)

Example: A company undergoing ISO certification uses such assessments.

Personal and Team Process Models

PSP: Tracks individual productivity.
TSP: Helps in team planning and quality improvement.

Example: Developers track time and bugs individually (PSP), while teams conduct sprint reviews (TSP).

BRAINWARE UNIVERSITY