

UNIT 1

Introduction

(4 Hours) — Software Engineering (CACS253)

1.1 Definition of Software

Software is a collection of computer programs together with the associated documentation and configuration data needed to make those programs operate correctly. It is more than just code: requirement documents, design models, test plans, user manuals and data files are all part of a software product. Software is **logical** rather than physical – it is developed or engineered, not manufactured in the classical sense.

Formula View

Software = Programs + Data + Documentation + Configuration

Example: A banking app = source code + database schema + user manual + installation and configuration files.

1.2 Types of Software

Type	Description	Examples
System software	Provides services to other programs and manages hardware	Operating systems, compilers, device drivers
Application software	Stand-alone programs solving a specific business or user need	MS Word, accounting software, hospital system
Engineering / Scientific software	Number-crunching and simulation programs	CAD tools, MATLAB, weather simulation
Embedded software	Resides inside a hardware product and controls it	Washing machine controller, car ABS
Web / Mobile applications	Network-centric software accessed via browser or phone	E-commerce site, eSewa, Facebook
Artificial Intelligence software	Uses non-numerical algorithms for complex problems	Expert systems, pattern recognition, chatbots
Product-line software	Reusable core sold to many users	Spreadsheets, games, multimedia

From the viewpoint of development, software is also divided into two broad categories:

- **Generic products** – developed by an organisation and sold to any customer (e.g., MS Office, Photoshop). The developer owns and controls the specification.
- **Customised (bespoke) products** – developed for a particular customer according to that customer's specification (e.g., an air-traffic control system, a college management system). The customer owns and controls the specification.

1.3 Characteristics of Software

- **Software is developed or engineered, not manufactured.** Cost lies in design and development; copying is almost free. Quality problems are removed by fixing the design, not by rejecting physical units.
- **Software does not wear out.** Hardware follows the “bathtub curve” (high failure early, low in the middle, high at end of life). Software is not affected by environmental wear, but it **deteriorates** because of changes made during maintenance, which introduce new defects.
- **Most software is custom built.** Although component-based reuse is growing, most software is still constructed to order rather than assembled from off-the-shelf parts.
- **Software is complex and intangible.** It cannot be touched or seen, which makes progress and quality hard to measure.
- **Software is flexible – and therefore constantly changing.** It is easy to modify, which invites continuous change requests.

Note: Hardware fails from wear; software fails because of hidden design or coding defects and because of changes made to it later.

1.4 Attributes of Good Software

Good software delivers the required functionality and performance to the user and is maintainable, dependable and usable. These are the essential attributes:

Attribute	Meaning
Maintainability	Software should be written so that it can evolve to meet changing customer needs. Change is inevitable in a changing business environment.
Dependability and security	Includes reliability, security and safety. Dependable software should not cause physical or economic damage in the event of failure and must not be exploitable by malicious users.
Efficiency	Software should not waste system resources such as memory and processor cycles. Includes responsiveness, processing time and memory use.
Acceptability (usability)	Software must be acceptable to the users for whom it is designed: understandable, usable and compatible with other systems they use.

1.5 Definition of Software Engineering

Software Engineering is an engineering discipline concerned with **all aspects of software production**, from the early stages of system specification to maintaining the system after it has gone into use. The IEEE definition states that it is the application of a systematic, disciplined, quantifiable approach to the development, operation and maintenance of software – that is, the application of engineering to software.

Two key phrases are important:

- **Engineering discipline** – engineers make things work by applying theories, methods and tools appropriately, while working within organisational and financial constraints.
- **All aspects of software production** – not just technical process of development, but also project management and the development of tools, methods and theories to support production.

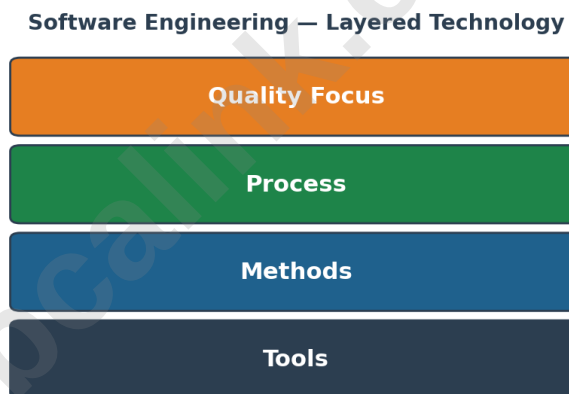


Figure 1.1: Software engineering as layered technology (quality focus is the foundation)

Why is software engineering important? Individual and society rely on advanced software systems. Without engineering, large projects overrun budget, miss deadlines and fail to satisfy users. Using software engineering methods is usually cheaper in the long run than writing programs as personal projects, because the cost of changes and failures is much higher.

Software Engineering vs. Computer Science

Computer Science	Software Engineering
Concerned with theory and fundamentals of computing	Concerned with practicalities of developing and delivering useful software
Studies algorithms, computability, data structures	Studies process, requirements, design, testing, management

Computer Science	Software Engineering
Focus on <i>what is possible</i>	Focus on <i>what is practical and economical</i>

1.6 Software Engineering Costs

Software costs often dominate computer-system costs; the cost of software on a PC is often greater than the hardware cost. Software costs more to **maintain** than it does to develop – for systems with a long life, maintenance costs may be several times development costs.

- Roughly **60%** of costs are development costs and **40%** are testing costs for a typical project.
- For custom software, evolution (change after delivery) costs often exceed development costs.
- Costs depend on the process used, the type of system and the required attributes such as performance and dependability.
- Distribution of costs varies by process model, but in general specification, design, implementation and integration/testing each take a substantial share.

Example – Cost Distribution (Waterfall)

Specification $\approx$ 15%, Design $\approx$ 25%, Development $\approx$ 20%, Integration and testing $\approx$ 40%.

Observation: **Integration and testing** is the most expensive phase; finding defects late is costly.

Note: The earlier a defect is found, the cheaper it is to fix. A defect found in maintenance can cost 50–100 times more than one found during requirements.

1.7 Key Challenges Facing Software Engineering

Challenge	Explanation
Heterogeneity	Systems must operate as distributed systems across networks with different types of computers and mobile devices, and integrate with legacy systems written in older languages. Techniques for building dependable software must cope with this diversity.
Business and social change	Business and society change rapidly with emerging economies and new technologies. Existing software must be adapted and new software delivered quickly, so traditional methods that take too long are unsuitable.
Security and trust	As software is intertwined with all aspects of life, it is essential that it be trusted. This is especially critical for remote web-based systems where attacks can be launched from anywhere.

Challenge	Explanation
Scale	Software must be developed across a very wide range of scales, from tiny embedded systems in wearable devices to enormous, cloud-based systems serving a global community.
Delivery time	Customers demand faster delivery without loss of quality. Agile and rapid approaches try to address this.

1.8 System Engineering and Software Engineering

A **system** is a purposeful collection of interrelated components working together to achieve some common objective. A system may include software, mechanical, electrical and electronic hardware, and be operated by people. **System engineering** is concerned with all aspects of the development and evolution of complex systems where software plays a major role. It therefore covers hardware development, policy and process design, system deployment and software engineering.

Aspect	System Engineering	Software Engineering
Scope	Whole system: hardware, software, people, processes	Only the software component of the system
Concern	System properties (emergent), integration, operation	Program structure, code quality, software process
Emergent properties	Reliability, performance, safety, security of the whole system	Contributes to but cannot alone guarantee them
Example	Designing an ATM network (machine, software, bank, staff)	Developing the ATM control program

System engineering stages: requirements definition → system design → subsystem development → system integration → system installation → system evolution → system decommissioning. Software engineers work within this larger context, especially during requirements and system design, where decisions about which functions are done by software and which by hardware are made.

Note: Emergent properties (e.g., reliability, safety) belong to the whole system and cannot be attributed to a single component.

1.9 Professional Practice

Software engineering is carried out within a legal and social framework that limits the freedom of engineers. Engineers must behave in an ethical and morally responsible way if they are to be respected as professionals. Ethical behaviour is more than simply following the law.

Issues of Professional Responsibility

- **Confidentiality** – respect the confidentiality of employers and clients, whether or not a formal agreement exists.
- **Competence** – do not misrepresent your level of competence or knowingly accept work outside your competence.
- **Intellectual property rights** – know local laws on patents and copyright and protect the IP of employers and clients.
- **Computer misuse** – do not use technical skills to misuse other people's computers (from games to viruses).

ACM / IEEE Code of Ethics – Key Principles

Principle	Meaning
PUBLIC	Act consistently with the public interest.
CLIENT AND EMPLOYER	Act in the best interests of client and employer, consistent with the public interest.
PRODUCT	Ensure products meet the highest professional standards possible.
JUDGMENT	Maintain integrity and independence in professional judgment.
MANAGEMENT	Managers and leaders subscribe to and promote an ethical approach to development and maintenance.
PROFESSION	Advance the integrity and reputation of the profession.
COLLEAGUES	Be fair to and supportive of colleagues.
SELF	Participate in lifelong learning and promote an ethical approach to the practice of the profession.

Ethical Dilemma – Example

You discover that the safety-critical software you shipped has a defect that may rarely cause failure, but your manager asks you to stay silent to avoid delays.

Guidance: Public interest comes first. Report the problem through proper channels, document the issue and refuse to conceal it.

Key Takeaways

- ✓ Software = programs + data + documentation; it is engineered, does not wear out but deteriorates with change.
- ✓ Key attributes: maintainability, dependability and security, efficiency, acceptability.
- ✓ Software engineering covers **all aspects** of production, including management and maintenance – not just programming.
- ✓ Maintenance cost usually exceeds development cost; late defects are the most expensive.
- ✓ Major challenges: heterogeneity, business/social change, security and trust, scale, delivery time.
- ✓ System engineering has a wider scope (hardware + software + people) than software engineering.
- ✓ Professionals must follow ethical codes: confidentiality, competence, IP rights, no computer misuse.

— *End of Unit 1* —