CSE-327
Software Engineering
Week-1

Md Shakil Ahmed

Lecturer

CSE, BUBT

Office – B1/602

Reference book

- Software Engineering – Ian Sommerville (10th Edition)
- Software Engineering: A Practitioner’s Approach – Roger S. Pressman (9th Edition)

Software

Software is:

1. A program or a set of instructions that tells the computer what to do and how to do.

2. It is the opposite of hardware, which describes the physical aspects of a computer.

Software Engineering

Software Engineering is:

- Software engineering is an engineering discipline that is concerned with all aspects of software production from the early stages of system specification through to maintaining the system after it has gone into use.

Software Engineering

- **IEEE Definition:**

Software engineering is a systematic, disciplined and cost-effective approach to develop a software; that is, the application of engineering to software.

Software Process

- The systematic approach that is used in software engineering is sometimes called a software process.
- A software process is a sequence of activities that leads to the production of a software product.

Fundamental Activities of Software Process

- There are four fundamental activities that are common to all software processes. These activities are:

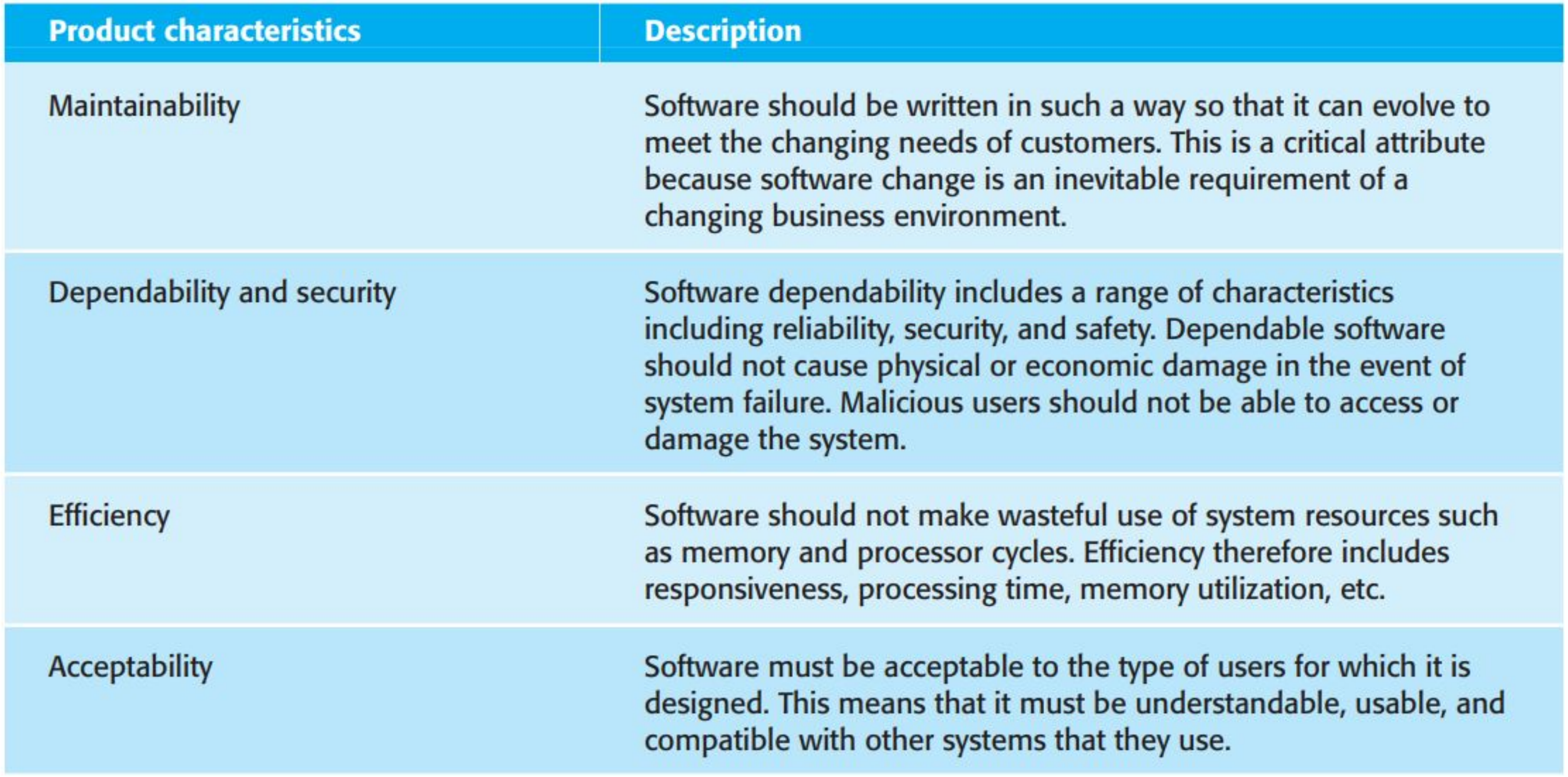
1. **Software specification**, where customers and engineers define the software that is to be produced and the constraints on its operation.
2. **Software development**, where the software is designed and programmed.
3. **Software validation**, where the software is checked to ensure that it is what the customer requires.
4. **Software evolution**, where the software is modified to reflect changing customer and market requirements.

Importance of Software Engineering

Software engineering is important for two reasons:

1. More and more, individuals and society rely on advanced software systems. We need to be able to produce reliable and trustworthy systems economically and quickly.
2. It is usually cheaper, in the long run, to use software engineering methods and techniques for software systems rather than just write the programs as if it was a personal programming project. For most types of systems, the majority of costs are the costs of changing the software after it has gone into use.

Some Essential attributes of good software



Product characteristics	Description
Maintainability	Software should be written in such a way so that it can evolve to meet the changing needs of customers. This is a critical attribute because software change is an inevitable requirement of a changing business environment.
Dependability and security	Software dependability includes a range of characteristics including reliability, security, and safety. Dependable software should not cause physical or economic damage in the event of system failure. Malicious users should not be able to access or damage the system.
Efficiency	Software should not make wasteful use of system resources such as memory and processor cycles. Efficiency therefore includes responsiveness, processing time, memory utilization, etc.
Acceptability	Software must be acceptable to the type of users for which it is designed. This means that it must be understandable, usable, and compatible with other systems that they use.

Legacy Software

- Legacy software is an older version of a program or application that still functions even after newer updates or versions are available.
- Companies typically use legacy software when they have older technology systems. Older computers are often only compatible with software of a similar age, so it's easier for companies to use legacy software instead of purchasing new equipment.

Who uses Legacy Software?

Here are some companies that may use legacy software:

1. **Background checking organizations**: Industries that perform background checks, like law enforcement and human resources (HR), may use legacy software due to the high volume of information and the complexity of the background checking system. Transferring the information to a newer system is challenging for organizations since data loss is risky.
2. **Banks**: It's common for banks to use legacy software since they handle accounts and transactions over long periods. They may use outdated software to ensure their accounts and transactions remain unchanged.
3. **Retail**: Companies in the retail industry may use legacy software so that they don't have to update all of their sales terminals. For example, if a grocery store uses older cash registers, it may cost less to repair them than to buy new ones for the entire store.

Software Engineering Ethics

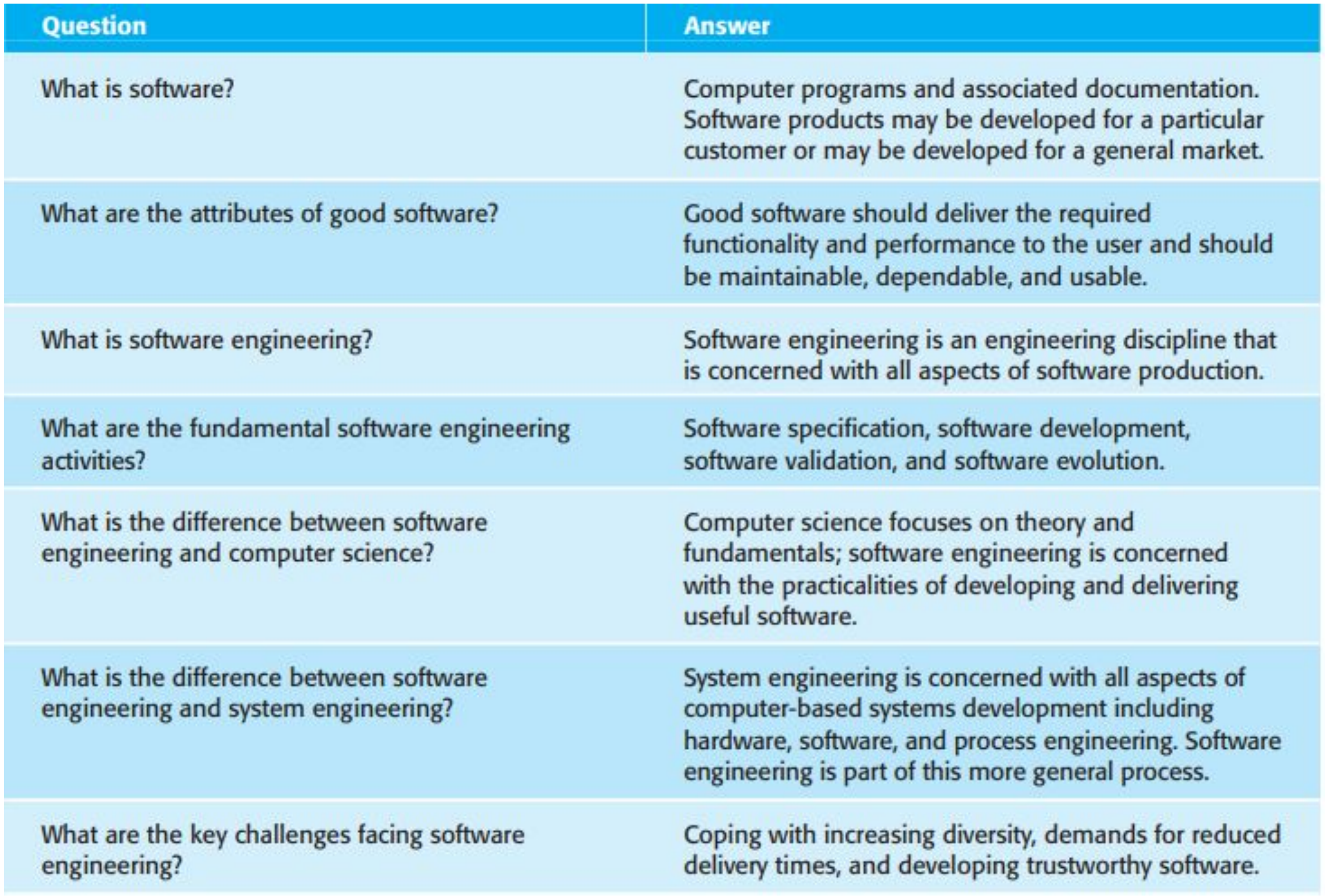
- Like other engineering disciplines, software engineering is carried out within a social and legal framework that limits the freedom of people working in that area.
- As a software engineer, you must accept that your job involves wider responsibilities than simply the application of technical skills.
- You must also behave in an ethical and morally responsible way if you are to be respected as a professional engineer.
- It goes without saying that you should uphold normal standards of honesty and integrity. You should not use your skills and abilities to behave in a dishonest way or in a way that will bring disrepute to the software engineering profession.

Software Engineering Ethics

- However, there are areas where standards of acceptable behavior are not bound by laws but by the more tenuous notion of professional responsibility. Some of these are:

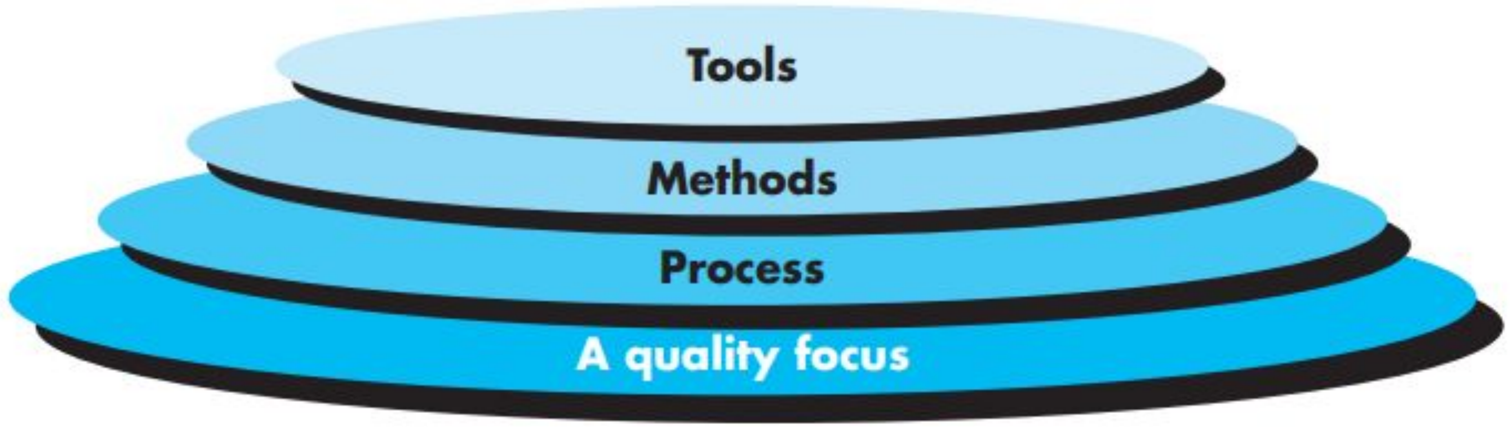
1. **Confidentiality:** You should normally respect the confidentiality of your employers or clients irrespective of whether or not a formal confidentiality agreement has been signed.
2. **Competence:** You should not misrepresent your level of competence. You should not knowingly accept work that is outside your competence.
3. **Intellectual property rights:** You should be aware of local laws governing the use of intellectual property such as patents and copyright. You should be careful to ensure that the intellectual property of employers and clients is protected.
4. **Computer misuse:** You should not use your technical skills to misuse other people's computers. Computer misuse ranges from relatively trivial (game playing on an employer's machine) to extremely serious (dissemination of viruses or other malware).

Frequently Asked Questions About Software



Question	Answer
What is software?	Computer programs and associated documentation. Software products may be developed for a particular customer or may be developed for a general market.
What are the attributes of good software?	Good software should deliver the required functionality and performance to the user and should be maintainable, dependable, and usable.
What is software engineering?	Software engineering is an engineering discipline that is concerned with all aspects of software production.
What are the fundamental software engineering activities?	Software specification, software development, software validation, and software evolution.
What is the difference between software engineering and computer science?	Computer science focuses on theory and fundamentals; software engineering is concerned with the practicalities of developing and delivering useful software.
What is the difference between software engineering and system engineering?	System engineering is concerned with all aspects of computer-based systems development including hardware, software, and process engineering. Software engineering is part of this more general process.
What are the key challenges facing software engineering?	Coping with increasing diversity, demands for reduced delivery times, and developing trustworthy software.

Software Engineering Layers



Software Engineering Layers

- Software engineering is a layered technology.
- Referring to the figure, any engineering approach (including software engineering) must rest on an organizational commitment to quality.
- Total quality management foster a continuous process improvement culture, and it is this culture that ultimately leads to the development of increasingly more effective approaches to software engineering.
- The bedrock that supports software engineering is **a quality focus**.

Software Engineering Layers

- The foundation for software engineering is the **process** layer.
- The software engineering process is the glue that holds the technology layers together and enables rational and timely development of computer software.
- Process defines a framework that must be established for effective delivery of software engineering technology.

Software Engineering Layers

- Software engineering **methods** provide the technical how-to's for building software.
- Methods encompass a broad array of tasks that include communication, requirements analysis, design modeling, program construction, testing, and support.
- Software engineering methods rely on a set of basic principles that govern each area of the technology and include modeling activities and other descriptive techniques.

Software Engineering Layers

- Software engineering **tools** provide automated or semi-automated support for the process and the methods.
- When tools are integrated so that information created by one tool can be used by another, a system for the support of software development, called computer-aided software engineering, is established.

Process Framework

- A process framework establishes the foundation for a complete software engineering process by identifying a small number of framework activities that are applicable to all software projects, regardless of their size or complexity.
- In addition, the process framework encompasses a set of umbrella activities that are applicable across the entire software process.
- Two types of process framework:

    (1) Framework Activities

    (2) Umbrella Activities

Framework Activities

- Communication
- Planning
- Modeling
  - Analysis of requirements
  - Design
- Construction
  - Code generation
  - Testing
- Deployment

Umbrella Activities

1. Software project tracking and control
2. Risk management
3. Software quality assurance
4. Technical reviews
5. Measurement
6. Software configuration management
7. Reusability management
8. Work product preparation and production

Polya’s Principle

- Generic framework activities— communication, planning, modeling, construction, and deployment—and umbrella activities establish a skeleton architecture for software engineering work. But how does the practice of software engineering fit in?
- George Polya outlined the essence of software engineering practice:

1. *Understand the problem* (communication and analysis).
2. *Plan a solution* (modeling and software design)
3. *Carry out the plan* (code generation).
4. *Examine the result for accuracy* (testing and quality assurance).

1. Understand the problem

It’s sometimes difficult to admit, but most of us suffer from hubris when we’re presented with a problem. We listen for a few seconds and then think, Oh yeah, I understand, let’s get on with solving this thing.

Unfortunately, understanding isn’t always that easy. It’s worth spending a little time answering a few simple questions:

- Who are the stakeholders?
- What data, functions, and features are required to properly solve the problem?
- Is it possible to represent smaller problems that may be easier to understand?
- Can the problem be represented graphically? Can an analysis model be created?

2. Plan the solution

Now you understand the problem (or so you think), and you can’t wait to begin coding. Before you do, slow down just a bit and do a little design:

- Have you seen similar problems before?
- Is there existing software that implements the data, functions, and features that are required?
- Has a similar problem been solved? If so, are elements of the solution reusable?
- Can subproblems be defined? If so, are solutions readily apparent for the subproblems?
- Can you represent a solution in a manner that leads to effective implementation?
- Can a design model be created?

3. Carry out the plan

The design you’ve created serves as a road map for the system you want to build. There may be unexpected detours, and it’s possible that you’ll discover an even better route as you go, but the “plan” will allow you to proceed without getting lost.

- Does the solution conform to the plan? Is source code traceable to the design model?
- Is each component part of the solution correct?
- Has the design and code been reviewed, or have correctness proofs been applied to the algorithm?

4. Examine the result

You can't be sure that your solution is perfect, but you can be sure that you've designed a sufficient number of tests to uncover as many errors as possible.

- Is it possible to test each component part of the solution?
- Has a reasonable testing strategy been implemented?
- Has the software been validated against all stakeholder requirements?

Hooker’s General Principles

David Hooker has proposed seven principles that focus on software engineering practice as a whole.

1: The Reason It All Exists
2: KISS (Keep It Simple, Stupid!)
3: Maintain the Vision
4: What You Produce, Others Will Consume
5: Be Open to the Future
6: Plan Ahead for Reuse
7: Think!

Hooker’s General Principles

- The Third Principle: *Maintain the Vision*

  A clear vision is essential to the success of a software project.

- The Fourth Principle: *What You Produce, Others Will Consume*

  In some way or other, someone else will use, maintain, document, or otherwise depend on being able to understand your system. So, always specify, design, and implement knowing someone else will have to understand what you are doing.

Hooker’s General Principles

- The Fifth Principle: *Be Open to the Future*

  Prepare for all possible answers by creating systems that solve the general problem, not just the specific one. This could very possibly lead to the reuse of an entire system.

- The Sixth Principle: *Plan Ahead for Reuse*

  Planning ahead for reuse reduces the cost and increases the value of both the reusable components and the systems into which they are incorporated.

Hooker's General Principles

- The Seventh Principle: *Think!*

  This last Principle is probably the most overlooked.

  When you think about something, you are more likely to do it right. You also gain knowledge about how to do it right again.

  If you do think about something and still do it wrong, it becomes a valuable experience.

Software Development Myths

- Erroneous beliefs about software and the process that is used to build it.

  - Management myths
  - Customer myths
  - Practitioner's myths

Management myths

- Myth: *If we get behind schedule, we can add more programmers and catch up (“Mongolian horde” concept).*

- Reality:
    - Software development is not a mechanistic process like manufacturing.
    - In the words of Brooks “adding people to a late software project makes it later.” At first, this statement may seem counterintuitive.
    - However, as new people are added, people who were working must spend time educating the newcomers, thereby reducing the amount of time spent on productive development effort.
    - People can be added but only in a planned and well-coordinated manner.

Customer myths

- Myth: *A general statement of objectives is sufficient to begin writing programs—we can fill in the details later.*

- Reality:
    - Although a comprehensive and stable statement of requirements is not always possible, an ambiguous “statement of objectives” is a recipe for disaster.
    - Unambiguous requirements (usually derived iteratively) are developed only through effective and continuous communication between customer and developer.

Practitioner’s myths

- Myth: *Once we write the program and get it to work, our job is done.*
- Reality: Industry data indicate that between 60 and 80 percent of all effort expended on software will be expended after it is delivered to the customer for the first time.