Software Engineering

CSIT- 6th Semester

Prepared by: Er. Rupak Gyawali, 2024

Unit 1: Introduction

Software and its Types; Attributes of Good Software; Software Engineering and its Importance; Fundamental Software Engineering Activities; Difference between Software Engineering and Computer Science; Difference between Software Engineering and System Engineering; Challenges and Cost of Software Engineering; Professional Software Development; Software Engineering Diversity; Internet Software Engineering; Software Engineering Ethics

Software and its types:

Software: Software is a set of instructions, data or programs used to operate computers and execute specific tasks.

- S/W is also called a collection of "Program". Software is a logical part of computer. It cannot be touched. The hardware cannot perform any task without software.
- It is a commands that tell a computer what to do. A set of programs which is designed for a specific operation is called a software. Eg: A program reads data and executes instructions.

Two types of a software:

a) System Software:

- System software is software that directly operates the computer hardware and provides the basic functionality to the users as well as to the other software to operate smoothly.
- System software basically controls a computer's internal functioning and also controls hardware devices such as monitors, printers, and storage devices, etc.
- It is like an interface between hardware and user applications, it helps them to communicate with each other because hardware understands machine language (i.e. 1 or 0) whereas user applications are work in human-readable languages like English, Hindi, German, etc. So system software converts the human-readable language into machine language and vice versa.

Example of System Software:

- **Operating System:** Run the computer and help other software work. Eg: Windows, macOS, Linux, Android.
- **Device Drivers:** Help the computer communicate with hardware like printers and monitors. Eg: Printer drivers, graphics card drivers.
- **Utility Software:** Help keep the computer fast, safe, and organized. Eg: Antivirus programs, disk cleanup tools, file compressors like WinRAR.
- **Firmware:** Low-level software embedded in hardware, providing basic control and operation of the device. Eg: BIOS/UEFI on a motherboard, Firmware for routers, Firmware for embedded systems in devices like washing machines or TVs.
- **Language Translators:** Convert programming code into instructions the computer understands. Eg: Compilers for programming languages like C++ or Python.

Software and its types:

Software:

Types of Software:

b) Application Software:

- Software designed to perform specific tasks for the user, directly supporting user activities in areas like business, media, productivity, and communication.
- It is a product or a program that is designed only to fulfill end-users' requirements.
- Helps users complete specific tasks (e.g., creating documents, managing data, communication).
- Example: MS-Word, Excel, VLC media player, MX player etc.

<u>Types of Application Software</u>

1. Packaged Software: Commercially produced and widely available software that is designed to meet the general needs of a wide range of users. Also called “off-the-shelf” software. **Examples**: Microsoft Office Suite, Adobe Creative Cloud, QuickBooks, Google Workspace.

Advantages:

i. Lower upfront cost compared to custom solutions.
ii. Quick and easy to deploy.
iii. Typically includes technical support and regular updates.

2. Customized Software: Tailor-made software specifically developed to meet the unique requirements of a particular organization or individual. **Examples**: A custom CRM for a business, specialized data analytics software, or custom software for hospital management.

Advantages:

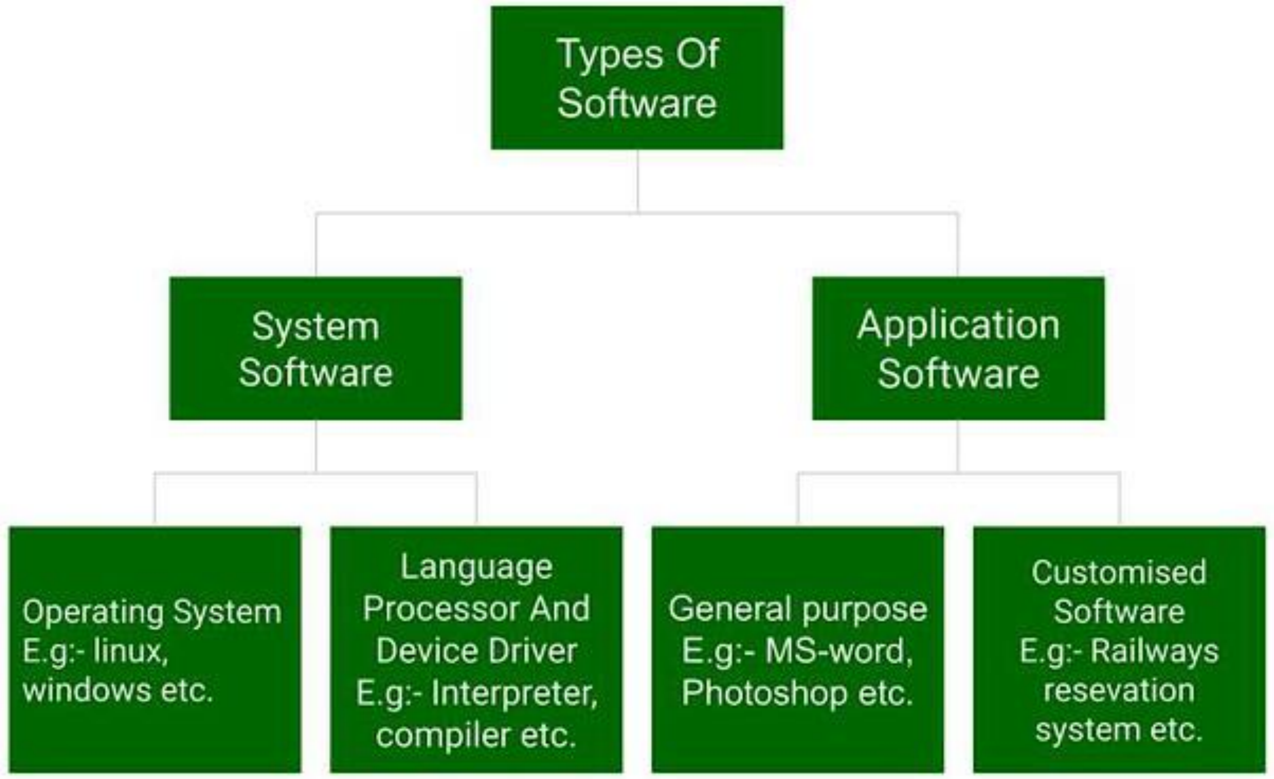
i. Highly tailored to specific needs, offering maximum flexibility and precise functionality.
ii. Adaptable and scalable to changing requirements.

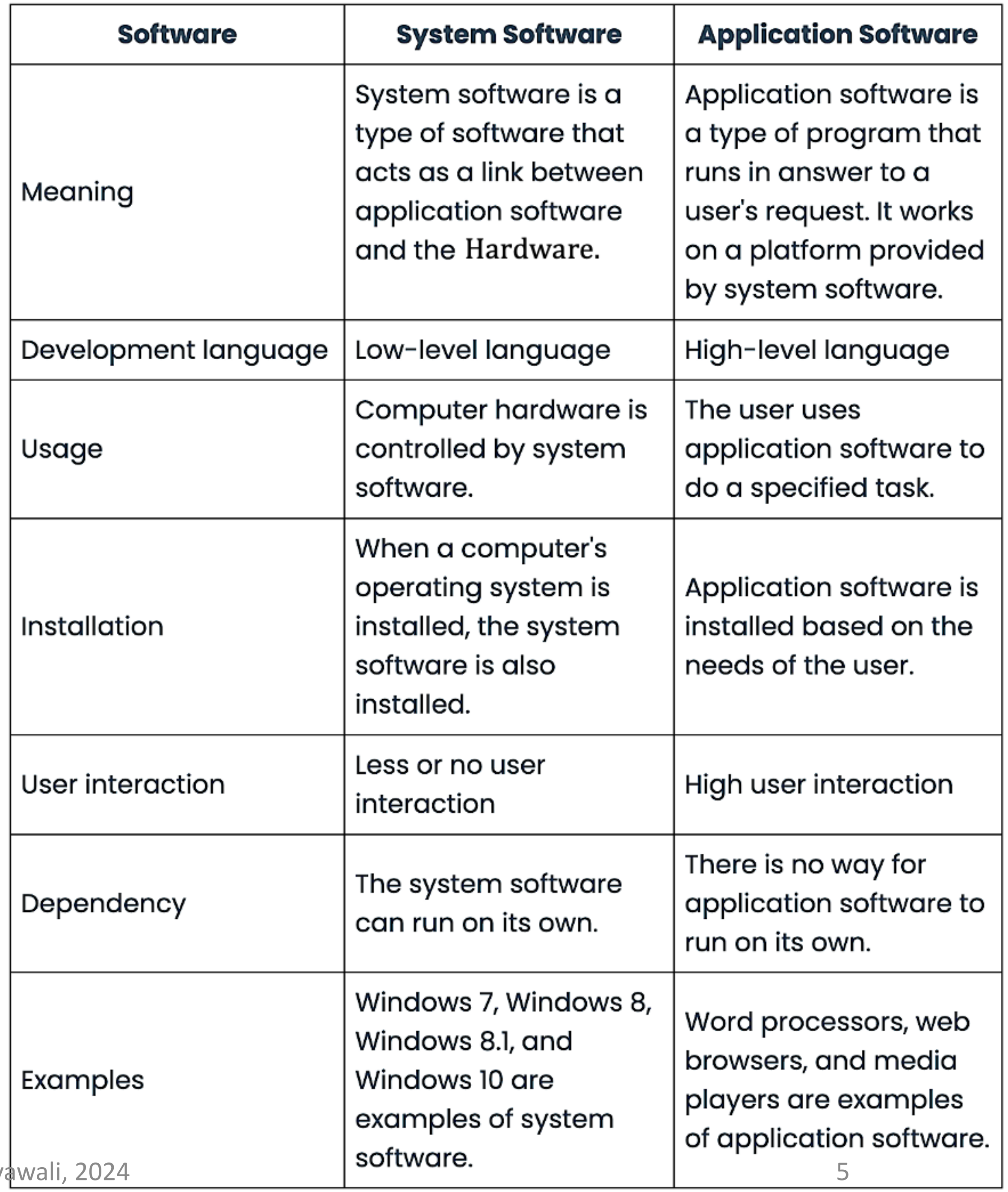
- Also called Tailored/ Customized/ Bespoke
- **Customized Software:** Usually starts with an existing solution that’s modified to fit specific needs.
- **Tailored Software:** Means software adapted to meet exact requirements, similar to how a tailor fits clothes to an individual. It may involve modifying existing software or developing new features.
- **Bespoke Software:** Refers to fully custom-built software developed from the ground up to meet unique requirements.

Software and its types:

Software:

Types of Software:


Types Of Software
System Software
Application Software
Operating System E.g:- linux, windows etc.
Language Processor And Device Driver E.g:- Interpreter, compiler etc.
General purpose E.g:- MS-word, Photoshop etc.
Customised Software E.g:- Railways resevation system etc.



Software	System Software	Application Software
Meaning	System software is a type of software that acts as a link between application software and the Hardware.	Application software is a type of program that runs in answer to a user's request. It works on a platform provided by system software.
Development language	Low-level language	High-level language
Usage	Computer hardware is controlled by system software.	The user uses application software to do a specified task.
Installation	When a computer's operating system is installed, the system software is also installed.	Application software is installed based on the needs of the user.
User interaction	Less or no user interaction	High user interaction
Dependency	The system software can run on its own.	There is no way for application software to run on its own.
Examples	Windows 7, Windows 8, Windows 8.1, and Windows 10 are examples of system software.	Word processors, web browsers, and media players are examples of application software.

Attribute of Good Software:

a) **Reliability:** Reliability is the ability of software applications to behave as expected and function under the maximum possible load.

Reliability is made up of different types

- **Availability:** The percentage of time the application is available for use. 100% availability refers to the system being always available and never going down under any circumstances.
- **Recoverability:** Recoverability refers to How quickly and easily the application can bounce back after a failure.
- **Fault Tolerance:** The ability of the application to keep working even if something goes wrong, like a hardware failure.

- **Eg:** If one server crashes on an e-commerce website, a reliable system should switch to another server automatically without any downtime.

b) **Maintainability:** Maintainability is about how easy it is for developers to update software and add new features when needed.

- The application architecture plays a critical role in maintainability.
- The well-architected software makes maintenance easier and more cost-effective.
- **Eg:** Imagine new privacy laws are introduced. If the software is based on old (legacy) code, it might be hard to make changes to meet these laws. But if the software is well-designed and the code is clear and documented, updating it to comply with the new laws will be much simpler.

Attribute of Good Software:

c) Usability: Usability refers to how easy it is for users to use a software application.

Having Good Usability relates to:

- Having Good Application performance
- Having Good User experience (UX) design
- Accessibility for all users including (Deaf, Blind)

Eg: Imagine a user who bought something on an e-commerce website and wants to return it. Good Usability means, The return option is clearly visible on the orders page, making it easy to find. Poor Usability means, If the return option is hidden on the "Contact Us" page, the user may get confused and struggle to find it.

d) Portability: Portability refers to how easily a software application can be moved or adapted to work in different environments, such as different types of hardware or operating systems. This is particularly important for applications that need to run on various devices.

- Many software developers face portability issues, especially with mobile applications. An app that works well on one platform may not function properly on another.
- **Eg:** An example of a portability issue – you have designed a web application that works perfectly fine on Android devices but when it is ported to iPhone devices (iOS), it fails to render. If the application is designed in a way that separates the user interface (UI) from the underlying business logic, it becomes much easier to fix these issues.

Attribute of Good Software:

e) **Correctness:** Correctness refers to how well software behaves according to its specified requirements.

- It means that the application should perform all functions as expected, including navigation, calculations, and form submissions.
- **Eg:** Imagine a user is signing up for an application. According to the requirements, after signing up, the user should be taken to the terms and conditions page. However, if the application instead takes the user directly to the home page without showing the terms and conditions, it has a correctness issue.

f) **Efficiency:** Efficiency is about getting tasks done in the shortest amount of time and with minimal use of system resources (like memory and processing power).

- Good efficiency is important because poor performance can lead to issues, such as making a computer freeze or slowing down other applications.

Eg: If you open one of the video editor applications on your desktop, as soon as you open your system freezes, and all the other open application start to behave in an unintended way. This is a bad application design and shows the poor efficiency of the software.

g) **Flexibility:** Flexibility refers to how quickly your application can adapt to future and current technology demands.

Eg: You are using a third-party library for styling your application. Due to some reasons, the third-party library declares the end of development. Now the question arises of how quickly your application can switch to another library. If it takes longer, then it might cost your business.

h) **Scalability:** Scalability refers to how well a system can handle increasing demands, such as more users or transactions, without slowing down or losing performance.

Types of Scalability:

- **Vertical Scalability**: Adding more power (like CPU or RAM) to existing servers.
- **Horizontal Scalability**: Adding more servers to distribute the load.

Eg: Imagine an e-commerce website running a Black Friday sale. On that day, many users visit the site at once. If the application is designed for scalability, it should automatically add more server nodes to handle the increased traffic and distribute it evenly. This prevents the website from crashing under the heavy load.

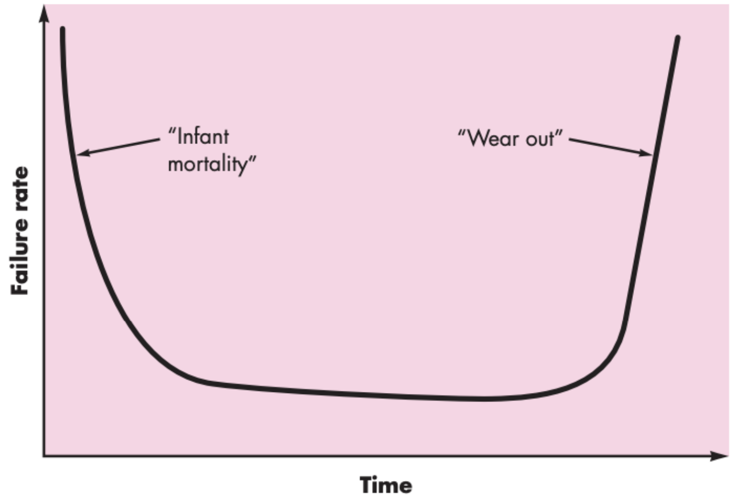
Other attributes → Compatibility, Testability, Reusability, Supportability, Interoperability, Security etc.

Characteristics of a Software:

a) **Software is developed or engineered; it is not manufactured in the classical sense:** Software is created through planning, coding, and testing rather than assembled on a factory line. ***Example:*** Unlike a car that goes through an assembly line, software like a mobile app is built by coding from scratch or using predefined modules.

b) **Software Doesn't Wear Out**: Software doesn't physically wear out over time like a machine or Hardware. However, it may need updates to stay functional. ***Example*:** A word processing software will continue working without "wearing out," though it might need occasional updates for security and new features.

Fig: Failure Curve for H/W



The "bathtub curve" figure shows two key phases for hardware:

- **Infant Mortality**: High failure rates early on due to initial defects. Once these issues are fixed, the failure rate drops.
- **Wear Out**: Over time, wear and environmental effects cause H/W to break down, increasing failure rates again.

c) **Although the industry is moving toward component-based construction, most software continues to be custom built:** Although some software uses pre-built components, most are custom-made to fit specific needs or features. ***Example:*** Many companies use customized software for unique needs, like payroll systems or inventory management, which are built specifically for them, even if they use reusable components.

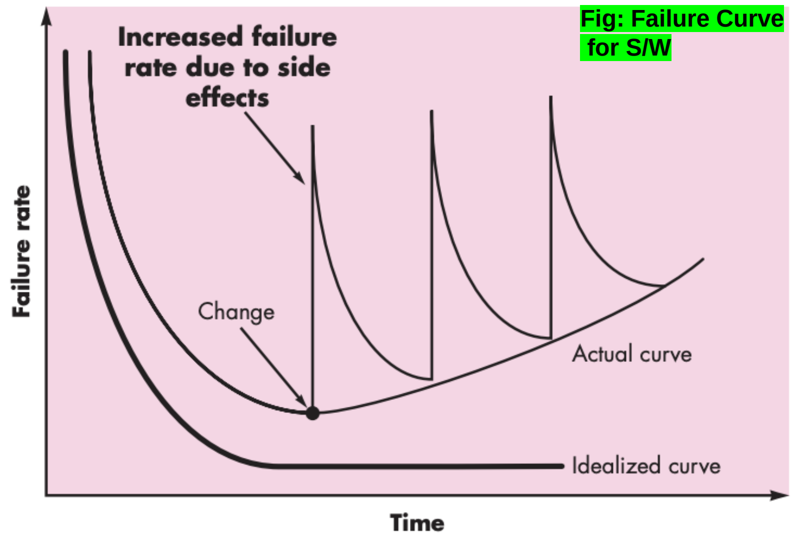


Fig: Failure Curve for S/W

As software gets updated or changed, new mistakes can happen, causing failures. Each time there's a change, failures spike in (Actual Curve), and over time, the overall failure rate increases. This means the software becomes less reliable as more changes are made.

d) Software can handle many tasks at once and needs to work well in all situations, making it complex to build. ***Example*:** An online shopping site needs to handle many users, secure payments, and product updates.

e) Software isn't limited by physical materials; it only depends on design and hardware. ***Example*:** A video game's features and graphics depend on the console's power but don't require any physical material.

Software Engineering: SE 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.

- **Well-defined scientific principles:** Modularity, Abstraction, Encapsulation etc.
- **Methods:** Agile Methodology, Waterfall Model, Spiral Model etc.
- **Procedures:** Code Review, Unit Testing, Version Control etc.

- Software Engineering is a systematic approach to designing, developing, testing and maintaining software.

Key Components of Software Engineering:

1. **Requirements Analysis**: Understand what users need from the software.
2. **Software Design**: Plan how the software will work and what it will include.
3. **Implementation**: Write the code to create the software.
4. **Software Testing**: Check that the software works correctly and fix any problems.
5. **Deployment**: Release the software to users and make sure it runs well.
6. **Maintenance**: Update and fix the software over time as needed.

Importance of Software Engineering:

1. **Reliability**: People and society depend heavily on software. It's crucial to create systems that are reliable and trustworthy.
2. **Efficiency**: Using software engineering methods helps produce software more quickly and at a lower cost. This is better than just writing code without a structured approach.
3. **Long-term Savings**: Although it may seem more expensive at first, following software engineering practices saves money in the long run. This is because most costs come from making changes to the software after it has been released.
4. **Cost of Changes**: After software is in use, changing it can be very costly. Software engineering helps manage and reduce these costs by using better planning and techniques from the start.
5. **Timely Delivery**: Proper software engineering practices help teams complete projects on time. Clear timelines and milestones make it easier to manage progress.
6. **Scalability**: Good software engineering practices ensure that software can grow and handle increased user demand without performance issues.
7. **Maintainability**: Software engineering emphasizes writing clean, organized code, making it easier to update and maintain the software over time.

Fundamental Software Engineering Activities / S/W process:

1. Software Specification: This is about figuring out what the software needs to do. It involves talking to people to understand their needs and writing down all the requirements.

- **Example**: For a shopping app, the team decides it needs a product search, shopping cart, and payment options.

2. Software Development: This step includes designing and coding the software. Developers create the architecture and write the code to turn the specifications into a working system.

- **Example**: The developers write code for the shopping app, adding features like browsing items, adding to the cart, and checking out.

3. Software Validation: In this phase, you test the software to ensure it meets the specifications and works as expected without errors.

- **Example**: The team tests the shopping app by going through different tasks, like placing an order, to ensure it works correctly and fix any issues they find.

4. Software Evolution: After release, software needs updates to fix bugs, add new features, or adapt to new technology. This phase is about maintaining and improving the software over time.

- **Example**: Over time, the shopping app gets updates for new features, like a recommendation system, and fixes for any reported bugs.

Computer Science Vs Software Engineering:

Computer Science: Computer Science focuses on the **theory** and **foundations** of computing, like algorithms, data structures, and how computers work.

- In short, computer science is about understanding how computers work.

Example: A computer scientist might study and design a new algorithm to make searching through large amounts of data faster. This research helps understand and improve the basics of how data can be processed.

Software Engineering: Software Engineering focuses on the **practical process** of creating, delivering, and maintaining software that people can use.

- Software engineering is about applying that computing knowledge to build real software.

Example: A software engineer might use that algorithm researched by a computer scientist to build a search feature in an e-commerce app, allowing users to quickly find products. The engineer focuses on making sure the search feature works well for users and is easy to maintain.

Aspect	Computer Science	Software Engineering
Focus	Theory, principles, and fundamentals of computing	Building, delivering, and maintaining software
Main Goal	Understanding how computers process information	Creating reliable and user-friendly software
Core Activities	Studying algorithms, data structures, and programming	Planning, coding, testing, and maintaining software
End Product	New knowledge, research, or theoretical insights	Practical applications, like apps or software tools
Problem Solving	Solves abstract, theoretical problems in computing	Solves real-world problems with functional software

System Engineering Vs Software Engineering:

System Engineering: (Computer based) System Engineering focuses on designing and developing entire computer-based systems, including **hardware, software, and processes**.

- System engineers work on the big picture, such as setting up system requirements, overall design, and making sure all parts work together.
- In short, system engineering covers both hardware and software.

Example of System Engineering: Designing an automated airport security system that includes hardware (like cameras, scanners, and computers) and software to manage data and track security information. The system engineer oversees the entire setup, making sure the hardware and software work together seamlessly.

Software Engineering: SE Focuses only on **software**—developing, testing, and maintaining programs within a system.

- It's just one part of the larger system engineering process.
- Software engineering focuses only on the software side.

Example of S/W Engineering: Building the software that runs the airport security system. This software might control data from the scanners, recognize faces, or monitor security footage. The software engineer focuses only on developing and maintaining the program itself.

Aspect	System Engineering	Software Engineering
Scope	Focuses on the entire system (hardware, software, and processes)	Focuses only on software development
Main Goal	Ensures all system components work together effectively	Ensures software functions correctly and efficiently
Key Activities	System specification, architecture, integration, deployment	Coding, testing, debugging, and software maintenance
Components	Involves both hardware and software	Involves only software
End Product	Complete, integrated system	Functional software within the system

Challenges faced by Software Engineering:

a. Handling Complexity and Diversity: Software systems are becoming more complicated, often combining different types of devices and programs spread across locations. For example, a web app might need to work smoothly on phones, tablets, and computers all at once. The challenge is to build reliable, adaptable software that works well with this mix.

b. **Keeping Up with Fast Business and Social Changes**: There's pressure to deliver software faster as businesses and society constantly change. The challenge is to speed up development without sacrificing quality.

c. **Ensuring Security and Trust**: Software is now essential in our daily lives, so it must be secure and trustworthy. The challenge is to make sure software is safe from threats and protects user data.

d. **Complex and Evolving Requirements:** As a software engineer, you'll need to manage changing requirements by understanding, organizing, and adjusting features. This helps avoid delays, extra costs, and disappointing users.

e. **Integration with Existing Systems:** Software needs to work well with other systems and technologies. To do this, you must handle compatibility issues by understanding requirements and limitations; failing to integrate smoothly can reduce its usefulness.

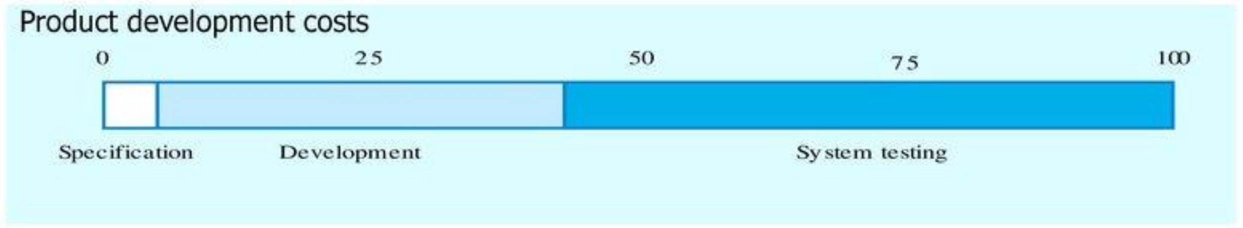
In summary:

- **1980s**: Focused on quality.
- **1990s**: Focused on speed.
- **Today**: Balancing speed, reliability, and security.

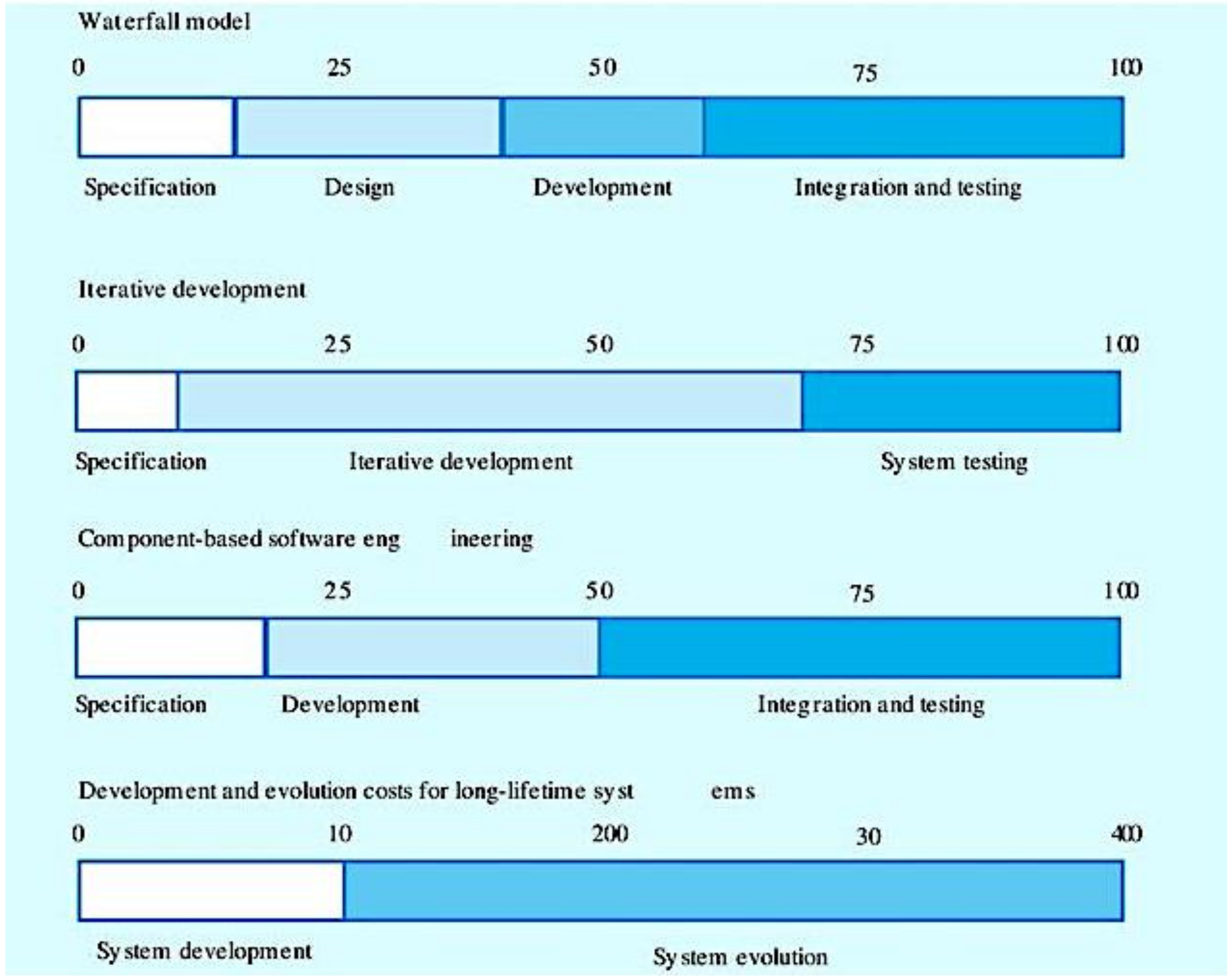
Costs of Software Engineering:

The costs of software engineering can be broken down as follows:

- Software usually costs more than the hardware in a computer system, especially on PCs. Maintaining software over time often costs more than developing it. Software engineering focuses on keeping these costs as low as possible.
- About **60% of costs** go to development (like design, coding) and **40% to testing**.
- **Most costs** come from making changes to the software after it's in use; this can be up to **60-90% of the total cost**.
- Costs also vary based on the type of system and requirements like speed and reliability.
- The cost distribution can change depending on the **development model** used (e.g., Waterfall, Extreme Programming, Cleanroom).



Around 5% cost for Spec, Dev. Covers 40% and System testing with 60% of cost.



Professional Software Development:

- Software development is a professional activity where software is developed for specific business purposes, to be used in other devices, or as software products such as information systems, CAD systems, etc.
- Unlike personal projects, professional software is developed by teams, not just one person, and it's updated and maintained over time for continuous use by others.
- Professional software development focuses on creating software for business or public use, rather than personal projects.
- Software engineering provides methods and tools for this, including ways to define, design, and update software over time. These methods are generally not needed for individual or personal programming.
    - Methods provided by SE:
        - Agile Methodology
        - Waterfall Model
        - Prototype Model
    - Tools Provided by SE:
        - Version Control tools → Git, Github
        - IDE → Visual Studio, Eclipse
        - Project Management Tools → Jira, Trello
        - Testing Tools: Junit, Postman

Why Professional S/W development is Important?

- **Creates High-Quality Software:** Using tested practices (Agile, Spiral, Prototyping) ensures the software works well and is reliable.
- **Helps Teams Work Together:** Tools (Version Controls, IDE (Integrated Development Environments), Project Management Tools, Testing Tools) and methods (Agile, Incremental, Spiral) allow team members to coordinate, share tasks, and track progress easily.
- **Makes Maintenance Easier:** Good structure and documentation make it simpler to update or fix the software when needed.
- **Meets User and Business Needs:** Clear requirements and design ensure the software does what users and businesses need.
- **Saves Time and Money:** Professional practices (Code Reviews, Automated Testing, Clear Documentation) catch problems early, making development faster and reducing future costs.

Software Engineering Diversity:

Software engineering is a structured approach to create software that meets the needs of users and developers while also considering costs, schedules, and reliability.

However, how this structured approach is implemented can vary greatly depending on:

- **The Organization**: Different companies have different methods.
  - **Example**: A big company like Google uses high-tech tools, while a small business uses simpler tools because they have less money.
- **The Type of Software**: What type of software is being developed?
  - **Example**: A fun game needs cool graphics, but a banking app needs to keep your money safe.
- **The Development Team**: The skills and experience of the people working on the software.
  - **Example:** A team of experts can handle complex tasks, while a team of beginners focuses on basic coding to learn.
- Each S/W company and software project is unique, so there isn't a one-size-fits-all approach to developing software. Different businesses have different needs, goals, and challenges, which require customized methods and processes.
- Over the last 50 years, many different software engineering techniques and tools have been developed like (Agile, Waterfall, XP)
- The type of software you are building is very important in choosing the right methods and tools to use.

<u>Some type of Software useful in choosing right methods and tools are:</u>

1. **Stand-alone Applications:** These are software programs that run on a local computer without needing an internet connection. **Example:** A word processor like Microsoft Word installed on your laptop.

2. **Interactive Transaction-based Applications:** These applications run on remote servers and are accessed by users through their PCs or terminals. **Example:** An online shopping website where users can browse and purchase items.

3. **Embedded Control Systems:** This type of software controls hardware devices and manages their functions. **Example:** The software in a microwave that controls cooking time and temperature.

Software Engineering Diversity:

Some type of Software useful in choosing right methods and tools are:

4. **Data Collection Systems:** These systems gather data from their surroundings using sensors and send it to other systems for analysis. **Example:** A smart thermostat that collects temperature data and adjusts settings accordingly.

5. **Batch Processing Systems:** These systems are designed to process large amounts of data in batches, rather than one at a time. **Example:** A phone billing system that processes all customer calls for the month to generate bills at once.

6. **Entertainment Systems:** These systems are mainly for personal use and aim to provide entertainment to the user. **Example:** A video streaming service like Netflix, where users can watch movies and shows.

7. **Systems for Modelling and Simulation:** These are developed by scientists and engineers to simulate physical processes or situations, often involving multiple interacting components. **Example:** Weather simulation software that predicts weather patterns by modeling atmospheric conditions.

8. **Systems of Systems:** This refers to complex systems made up of multiple interconnected software systems that work together. **Example:** A smart city system that integrates traffic management, public transport, and environmental monitoring systems.

Note:
Software engineering diversity means that different types of software require different development methods. For example, Agile might be best for a fast-changing project like Facebook, while an incremental approach could be more suitable for a stable product like Microsoft Office. Each project has its own needs, so choosing the right method is important for success.

Internet Software Engineering:

- **Impact of the Web:** The development of the World Wide Web has significantly changed how we live and interact with technology.
- **Evolution of Browsers:** Around the year 2000, web browsers became more advanced, allowing for more complex functions and features.
- **Web Deployment:** Instead of creating software that needs to be installed on each user's computer, developers began deploying software on web servers. This means users can access the software through their browsers.
- **Cost-Effective Updates:** It's much easier and cheaper to update software since changes can be made on the server side, eliminating the need for users to install updates on their PCs.
- **Introduction of Web Services:** The next phase in web-based systems involved creating web services, which allow different software systems to communicate and work together over the internet.
- **Cloud Computing:** The concept of "computing clouds" emerged, where services like web-based email (e.g., Gmail) operate on remote servers rather than on individual devices, making access and management easier.

Software Engineering Ethics:

Ethics: Ethics is a set of rules that help us know what is right and wrong. It guides our behavior and decisions in everyday life.

S/W Engineering Ethics: Software engineering ethics are guidelines that help software engineers make the right choices in their work.

- S/W engineering ethics ensure that engineers act honestly, create high-quality software, treat others fairly, and keep learning throughout their careers.

S/W Engineer’s Ethics: Software engineer ethics are the rules and values that guide how software engineers should behave at work. These ethics help engineers make responsible choices that are good for people, clients, and their job.

Why S/W Engineering Ethics is Important?
Software engineering ethics is important because it ensures that engineers create safe, reliable, and high-quality software that benefits society. It also builds trust between engineers and society.

Software Engineering Ethics:

Professional responsibility for software engineers that aren't strictly covered by law:

1. **Confidentiality**: Always keep your employer's or client's information private, even if there isn't a signed agreement.

2. **Competence**: Be honest about your skills. Don't take on work that you aren't qualified to do.

3. **Intellectual Property Rights**: Know the laws about copyrights and patents. Make sure to protect the intellectual property of your employer or clients.

4. **Computer Misuse**: Don't use your skills to harm others' computers. Misuse can range from playing games on work computers to spreading viruses.

S/W Engineering Code of Ethics and Professional Practices:

Eight principles outlined by the IEEE-CS/ACM Joint Task Force that software engineers should follow:

1. **Public Interest:** Always consider what is best for the public.

2. **Client and Employer:** Do what's right for your clients and employers while keeping the public in mind.

3. **Quality Products:** Make sure your software meets high standards of quality.

4. **Integrity:** Be honest and fair in your work decisions.

5. **Good Management:** If you're a manager, encourage ethical practices in your team.

6. **Professional Reputation:** Help improve how people view the software engineering profession.

7. **Support Colleagues:** Be fair and helpful to your coworkers.

8. **Lifelong Learning:** Keep learning about your field and promote ethical behavior in your work.