What is software (SW)?

- SW is
  - not only *programs*
  - but also all *associated documentation*, and
  - *configuration data*

  that make these programs operate correctly.
- More specifically, a SW system consists of
  - *separate programs*
  - *configuration files* setting up these programs
  - *system documentation* describing the structure of the system in good detail
  - *user documentation* explaining how to use and operate the system.

Two Classes of SW Products

- *Generic* products
  - *stand-alone systems*
  - *sold* on open market *to any customer* such as
    - *word processors,*
    - *databases,*
    - *drawing packages*
- *Custom* or *bespoke* products
  - systems developed *specifically for a customer*

The Pioneering Era (1955-1965)

- New computers were coming out almost every year or two, rendering existing ones obsolete.
- Software people had to rewrite all their programs to run on these new machines.
- Programmers did not have computers on their desks and had to go to the "machine room".

The Pioneering Era - 2

- Jobs were run by
  - signing up for machine time or by operational staff.
  - putting punched cards for input into the machine's card reader and waiting for results to come back on the printer.
- The field was so new that the idea of management by schedule was non-existent.
- Making predictions of a project's completion date was almost impossible.

The Pioneering Era - 3

- Computer hardware was *application-specific*. Scientific and business tasks needed different machines.
- Due to the *need to frequently translate old software to meet the needs of new machines*, high-order languages like FORTRAN, COBOL, and ALGOL were developed.

The Pioneering Era - 4

- Hardware vendors gave away *systems software for free* as hardware could not be sold without software. A few companies sold the service of building custom software but no software companies were selling packaged software.

The Pioneering Era - 5

- The notion of *reuse* flourished. As software was free, user organizations commonly gave it away. Groups like IBM's scientific user group SHARE *offered catalogs of reusable components*.
- Academia did not yet teach the *principles of computer science*.
- *Modular programming and data abstraction* were already being used in programming.

The Stabilizing Era (1965-1980)

- The whole *job-queue system* had been *institutionalized* and so programmers no longer ran their jobs except for peculiar applications like on-board computers. To handle the jobs, an *enormous bureaucracy* had grown up around the central computer center.
- The major problem as a result of this bureaucracy was *turnaround time*, the time between job submission and completion. At worst it was *measured in days*.

The Stabilizing Era - 2

- *IBM 360* signaled the beginning of the *stabilizing era*.
- Largest software project to date ending the era of a faster and cheaper computer emerging every year or two.
- Software people could finally spend time writing new software instead of rewriting the old.
- The 360 also combined scientific and business applications onto one machine. It offered both binary and decimal arithmetic. With the advent of the 360, the organizational separation between scientific and business application people came to diminish and this had a massive impact on the sociology of the field. Scientific programmers who usually had bachelor degrees felt superior to business programmers who usually held only associate degrees.
- One scientific programmer remarked: "I don't mind working with business programmers, but I wouldn't want my daughter to marry one!"

The Stabilizing Era - 3

- The massive O/S, still coming largely free with the computer, controlled most of the services that a running program needed.
- The *job control language JCL* raised a whole new class of problems. The programmer had to write the program in a whole new language to tell the computer and OS what to do. JCL was the least popular feature of the 360.
- PL/I, introduced by IBM to merge all programming languages into one, failed.
- The demand for programmers exceeded the supply.

The Stabilizing Era - 4

- The notion of *timesharing*, using terminals at which jobs could be directly submitted to queues of various kinds was beginning to emerge, meeting with some resistance from traditionalists.
- As the software field stabilized, software became a corporate asset and its value became huge.
- *Stability* lead to the emergence of *academic computing disciplines* in the late 60's. However the *software engineering discipline did not yet exist*.

The Stabilizing Era - 5

- Many *“high-hype”* disciplines like *Artificial Intelligence* came into existence. As these new concepts could not be converted into predicted benefits, the credibility of the computing field began to diminish.
- "*Structured Programming*" burst on the scene *in the middle of this era*.
- *Standards organizations became control battle grounds*. The vendor who defined the standards could gain significant competitive advantage by making the standards match their own technology.

The Stabilizing Era - 6

- Although hardware vendors tried to put a brake on the software industry by keeping their prices low, software vendors emerged a few at a time.
- Most *customized applications* continued to be *done in-house*.
- Programmers still had to go to the "*machine room*" and did not have computers on their desks.

The Micro Era (1980-Present)

- The price of computing has dropped dramatically making ubiquitous computing (i.e., computing everywhere) possible. Now every programmer can have *a computer on his desk*.
- The old *JCL* has been *replaced by* the user-friendly *GUI*.

The Micro Era - 2

- The field still has its problems. The software part of the hardware architecture that the programmer must know about, such as the instruction set, has not changed much since the advent of the IBM mainframe and the first Intel chip. The most-used programming languages today are between 15 and 40 years old. The *Fourth Generation Languages* never achieved the *dream of "programming without programmers"* and the *idea is pretty much limited to report generation from databases*. There is an increasing clamor though for more and better software research.

Software Development Life Cycle

- Software Development
- Traditional Software Development Models
- Waterfall Model
- Classical Waterfall Model

SOFTWARE DEVELOPMENT

- It is a process to create computer software using one or more specific programming languages that provides functionality to address particular business or personal objectives.
- The development of software is usually a planned initiative with a number of steps and stages that result in the creation of operational software

- a process with a set of activities that create computer software products, including their design, development, testing, and deployment
- Software itself is a set of instructions or programs that tell a computer what to do. It is independent of hardware and makes computers programmable.
- Software development is the process programmers use to build computer programs. The process, also known as the Software Development Life Cycle (SDLC), includes several phases that provide a method for building products that meet technical specifications and user requirements.

- The SDLC provides an international standard that software companies can use to build and improve their computer programs. It offers a defined structure for development teams to follow in the design, creation and maintenance of high-quality software. The aim of the IT software development process is to build effective products within a defined budget and timeline.
-

Key steps in the software development process

1. Needs identification
2. Requirement analysis
3. Design
4. Development and implementation
5. Testing
6. Deployment and maintenance

1. Needs identification

- Needs identification is a market research and brainstorming stage of the process. Before a firm builds software, it needs to perform extensive market research to determine the product's viability.
- Developers must identify the functions and services the software should provide so that its target consumers get the most out of it and find it necessary and useful.
- There are several ways to get this information, including feedback from potential and existing customers and surveys.

- The IT teams and other divisions in the company must also discuss the strengths, weaknesses and opportunities of the product. Software development processes start only if the product satisfies every parameter necessarily for its success.

2. Requirement analysis

- Requirement analysis is the second phase in the software development life cycle. Here, stakeholders agree on the technical and user requirements and specifications of the proposed product to achieve its goals.
- This phase provides a detailed outline of every component, the scope, the tasks of developers and testing parameters to deliver a quality product.

- The requirement analysis stage involves developers, users, testers, project managers and quality assurance.
- This is also the stage where programmers choose the software development approach such as the waterfall or V model.
- The team records the outcome of this stage in a Software Requirement Specification document which teams can always consult during the project implementation.

3. Design

- Design is the third stage of the software development process. Here, architects and developers draw up advanced technical specifications they need to create the software to requirements.
- Stakeholders will discuss factors such as risk levels, team composition, applicable technologies, time, budget, project limitations, method and architectural design.

- The Design Specification Document (DSD) specifies the architectural design, components, communication, front-end representation and user flows of the product.
- This step provides a template for developers and testers and reduces the chances of flaws and delays in the finished product.

4. Development and implementation

- The next stage is the development and implementation of the design parameters. Developers code based on the product specifications and requirements agreed upon in the previous stages.
- Following company procedures and guidelines, front-end developers build interfaces and back-ends while database administrators create relevant data in the database. The programmers also test and review each other's code.
- Once the coding is complete, developers deploy the product to an environment in the implementation stage. This allows them to test a pilot version of the program to make performance match the requirements.

5. Testing

- The testing phase checks the software for bugs and verifies its performance before delivery to users. In this stage, expert testers verify the product's functions to make sure it performs according to the requirements analysis document.
- Testers use exploratory testing if they have experience with that software or a test script to validate the performance of individual components of the software. They notify developers of defects in the code. If developers confirm the flaws are valid, they improve the program, and the testers repeat the process until the software is free of bugs and behaves according to requirements.

6. Deployment and maintenance

- Once the software is defect-free, the developers can deliver it to customers. After the release of a software's production version, the IT software development company creates a maintenance team to manage issues clients encounter while using the product.
- Maintenance can be a hot-fix if it is a minor issue but severe software failures require an update.