Intro to SE

WEEK 2

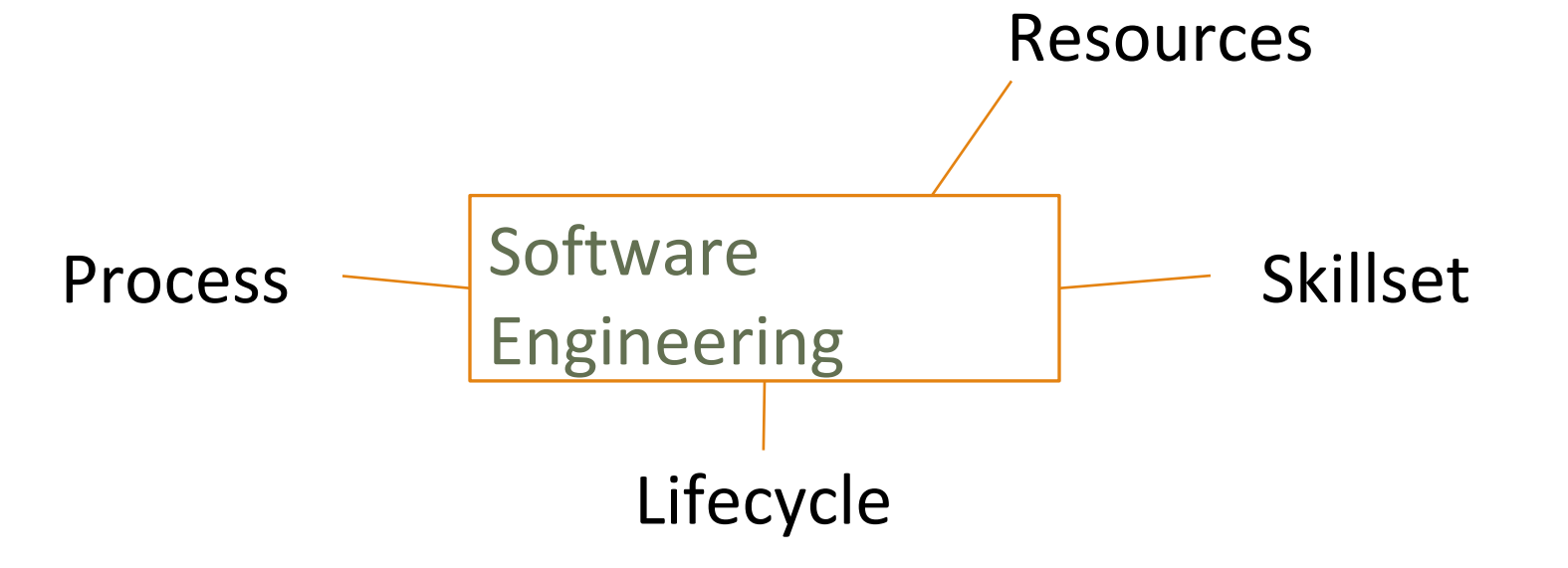
Lehman’s Laws

(1974) "Continuing Change" — A system must be continually adapted or it becomes progressively less satisfactory. It happens so until it becomes economical to replace it by a new or a restructured version

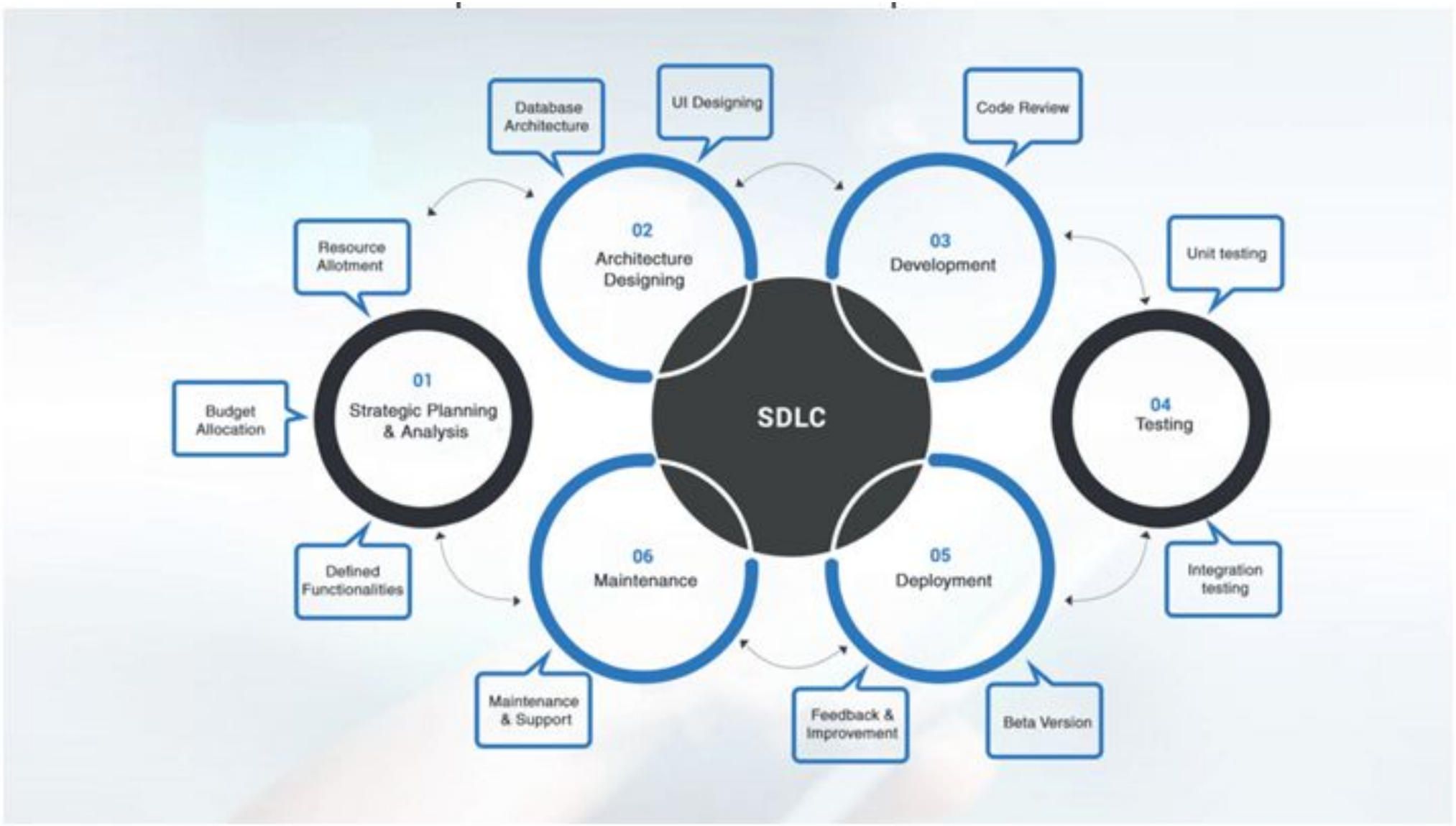
(1974) "Increasing Complexity/Entropy" —Complexity/entropy of a system increases with time, unless work is done to maintain or reduce it

(1991) "Continuing Growth" — the functional content of a system must be continually increased to maintain user satisfaction over its lifetime

(1996) "Declining Quality" — the quality of a system will appear to be declining unless it is rigorously maintained and adapted to operational environment changes



Typical phases in lifecycle of software


SDLC
01
Strategic Planning & Analysis
Budget Allocation
Resource Allotment
Defined Functionalities
02
Architecture Designing
Database Architecture
UI Designing
03
Development
Code Review
Unit testing
04
Testing
Integration testing
05
Deployment
Beta Version
Feedback & Improvement
06
Maintenance
Maintenance & Support

Software Lifecycle

Phases

- Requirements analysis and definition
- System (architecture) design /Program (detailed/procedural) design
- Writing programs (coding/implementation)
- Testing: unit, integration, system
- System delivery (deployment)

- -

- Maintenance

SDLC

1. Requirement Phase

This is the first and fundamental step in the Life Cycle of Software Development. It starts with gathering the requirements from customers or clients. In most organizations, this role is taken care of by Business Analysts. A Business Analysts interacts with the customer/clients, set up daily meetings, document the requirements in Business Requirement Specifications (or Simple Business Specification), and hand over the final documented requirement to the development team. It is the responsibility of Business Analysts that every detail is captured and documented and also to make sure that everyone clearly understands the client requirements.

2. Analysis Phase

Once the Requirement Gathering phase is completed, the next task is to analyze the requirements and get them approved by the customer/clients. This is achieved through Software Requirement Specification (SRS), which consists of all the requirements gathered and developed during the Requirements Gathering phase. This phase is mainly done by Project Managers, Business Analysts, and Consultants.

SDLC Cont.

3. Design Phase

Once the Analysis Phase is over, next comes the need to come up with the most accurate, robust, efficient and cost-effective architecture of the product that needs to be developed. Usually, more than one design is proposed in this phase, and the best one is selected based on different parameters such as robustness, durability, timeline, cost-effectiveness, and many more! The different design architecture is generally documented in Design Document Specification or DDS.

This phase consists of 2 design approaches:

Low-Level Design: This task is performed by the Senior Developers where they specify the function of each module of the product architecture that has to be developed.

High-Level Design: This task is performed by Architects/Senior Architects where they design different possible architectures of the product that has to be developed.

SDLC Cont.

4. Development Phase

This phase is where the actual implementation of programming languages and different frameworks are being utilized for the development of the product. In this phase, all developers are involved. Developers are expected to follow certain predefined coding standards and guidelines; they are expected to complete the project modules within the defined deadline for the project. This phase is also the longest and one of the most critical phases in the Software Development Life Cycle. This phase is documented as a Source Code Document (SCD).

5. Testing Phase

Once the Development phase is completed, the next step is to test the developed software. The developed software is sent to the testing team, where they conduct different types of testing thoroughly on the software and look for defects. If any defect is found, the testing team records and document which is again sent back to the development team for error removal. This role is taken care of by Software Testers and Quality Analysts of the company. The testing team has to make sure that each component of the software is error-free and it works as expected.

SDLC Cont.

6. Deployment and Maintenance Phase

After the testing phase is over, the first version of the software is deployed and delivered to the customer for their use. Once the customer starts using the developed software, there is the scope of bug fixing that was not detected during the testing phase as when a large group of end-users starts using the software; there could be some probability that few boundary cases might have been missed. There is also scope for upgrading the software with newer versions and the latest security patches and technologies. And finally, there is also scope for enhancement of the software by adding more features into the existing software.

Costs in Software Engineering

Cost to change software

Cost to overcome a mistake (defect)

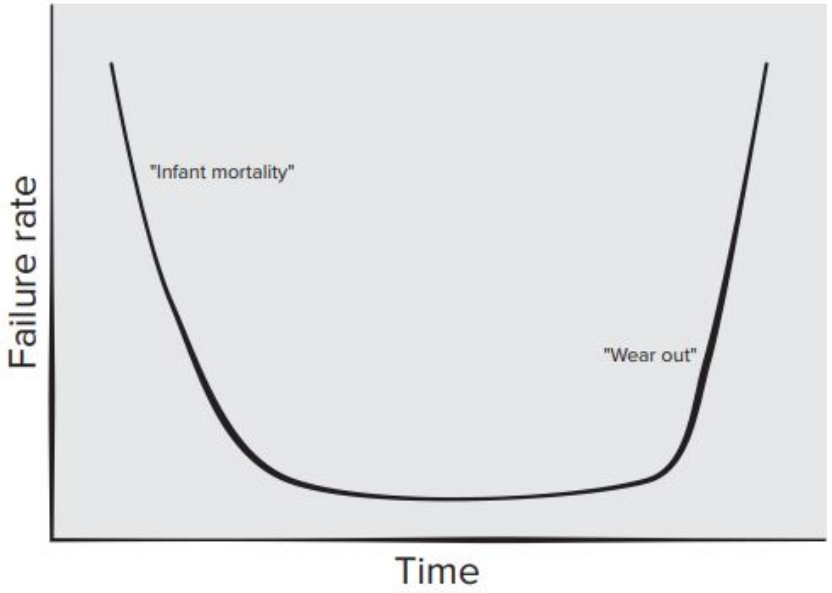
Operational cost

Deployment cost

Software failure vs. hardware failure

Software is a logical rather than a physical system element. Therefore, software has one fundamental characteristic that makes it considerably different from hardware: Software doesn't "wear out."

Picture depicts failure rate as a function of time for hardware. The relationship, often called the "bathtub curve," indicates that hardware exhibits relatively high failure rates early in its life (these failures are often attributable to design or manufacturing defects); defects are corrected, and the failure rate drops to a steady-state level (hopefully, quite low) for some period of time

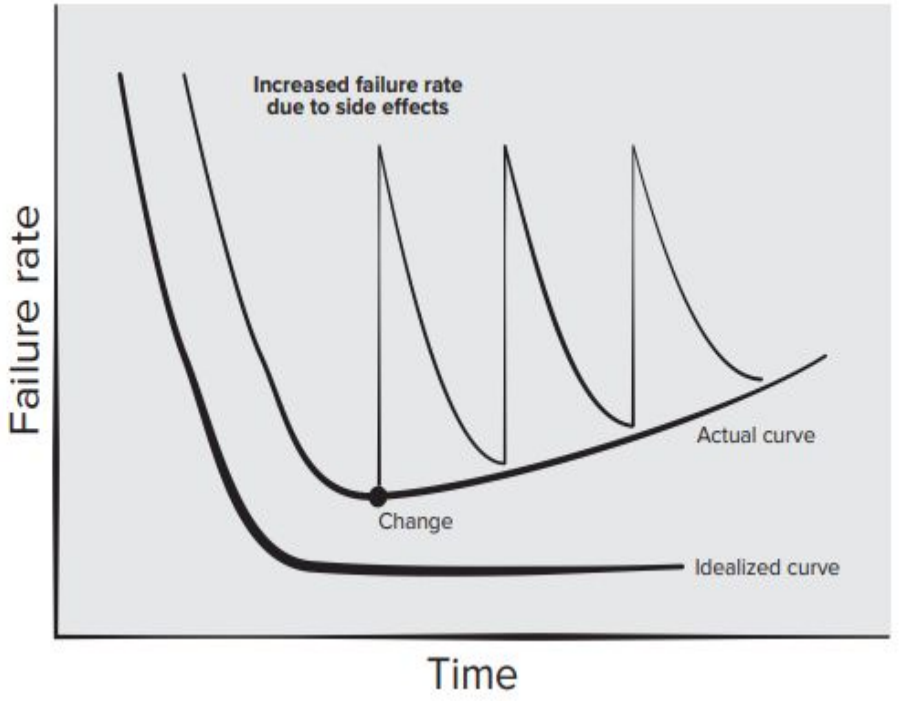


Software failure vs. hardware failure

Software is not susceptible to the environmental maladies that cause hardware to wear out. In theory, therefore, the failure rate curve for software should take the form of the “idealized curve”

Undiscovered defects will cause high failure rates early in the life of a program. However, these are corrected and the curve flattens as shown. The idealized curve is a gross oversimplification of actual failure models for software. However, the implication is clear—software doesn’t wear out. But it does deteriorate!

When a hardware component wears out, it is replaced by a spare part. There are no software spare parts. Every software failure indicates an error in design or in the process through which design was translated into machine executable code. Therefore, the software maintenance tasks that accommodate requests for change involve considerably more complexity than hardware maintenance



FAQ about software engineering

Question	Answer
What is software?	Computer programs, data structures 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 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.

Essential attributes of good software

Product characteristic	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 utilisation, 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.

Software Engineering

Disciplined, consistent, and systematic effort to construct (design + build) and maintain software in timely and cost-effective manner

Software engineering layers

The main objective of software engineering layers is to help software developers obtain high-quality software. There are four types of layers in Software Engineering such as – **Tools, methods, process, A quality focus**.

A quality focus: It is a culture that ultimately leads to the development of software engineering. The bedrock that supports software engineering, says **A quality focus**.

Process: A process defines who is doing what, when and how to reac a certain goal. The software process forms the basis for managemen control of software projects. It establishes the context in which technical methods are applied for work products are produced.

Example: Models, Documents, Data, Reports, Forms, etc.

Software engineering layers

Methods: 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 modelling, program construction, testing and support.

iv. **Tools:** 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 is called Computer-Aided Software Engineering

Software Development Process

Software Process Framework is an abstraction of the software development process. It details the steps and chronological order of a process. Since it serves as a foundation for them, it is utilized in most applications. Task sets, umbrella activities, and process framework activities all define the characteristics of the software development process.

A **process** is a collection of Activities, actions, tasks that are performed when some work product is to be created.

Activity: An activity strives to achieve a broad objective (e.g., communication with stakeholders)

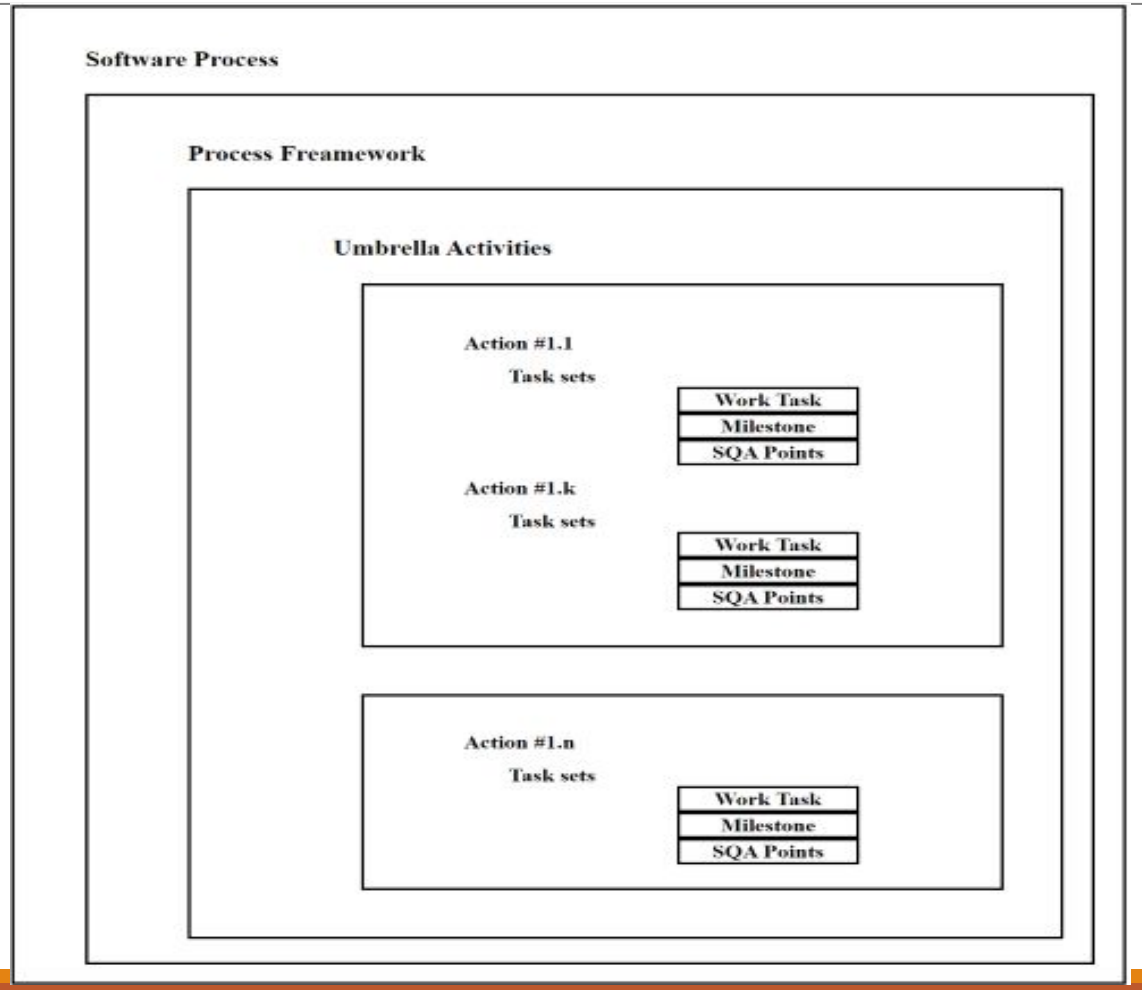
Action: An action (e.g., architectural design) encompasses a set of tasks that produce a major work product (e.g., an architectural design model).

Task: A task focuses on a small, but well-defined objective (e.g., conducting a unit test) that produces a tangible outcome.

In the context of software engineering, a process is not a rigid prescription for how to build computer software. Rather, it is an adaptable approach that enables the people doing the work (the software team) to pick and choose the appropriate set of work actions and tasks. The intent is always to deliver software in a timely manner and with sufficient quality

Process Framework Activities:

The process framework is required for representing common process activities. Five framework activities are described in a process framework for software engineering. Communication, planning, modeling, construction, and deployment. In addition, the process framework encompasses a set of umbrella activities that are applicable across the entire software process.



Framework Activities:

Communication: By communication, customer requirement gathering is done. Communication with consumers and stakeholders to determine the system's objectives and the software's requirements.

Planning: Establish engineering work plan, describes technical risk, lists resources requirements, work produced and defines work schedule.

Modeling: Architectural models and design to better understand the problem and for work towards the best solution. The software model is prepared by:

- o Analysis of requirements
- o Design

Construction: Creating code, testing the system, fixing bugs, and confirming that all criteria are met. The software design is mapped into a code by:

- o Code generation
- o Testing

Deployment: In this activity, a complete or non-complete product or software is represented to the customers to evaluate and give feedback. On the basis of their feedback, we modify the product for the supply of better products.

Umbrella activities:

Umbrella Activities are that take place during a software development process for improved project management and tracking.

Software project tracking and control: This is an activity in which the team can assess progress and take corrective action to maintain the schedule. Take action to keep the project on time by comparing the project's progress against the plan.

Risk management: The risks that may affect project outcomes or quality can be analyzed. Analyze potential risks that may have an impact on the software product's quality and outcome.

Software quality assurance: These are activities required to maintain software quality. Perform actions to ensure the product's quality.

Formal technical reviews: It is required to assess engineering work products to uncover and remove errors before they propagate to the next activity. At each level of the process, errors are evaluated and fixed.

Umbrella activities:

Software configuration management: Managing of configuration process when any change in the software occurs.

Work product preparation and production: The activities to create models, documents, logs, forms, and lists are carried out.

Reusability management: It defines criteria for work product reuse. Reusable work items should be backed up, and reusable software components should be achieved.

Measurement: In this activity, the process can be defined and collected. Also, project and product measures are used to assist the software team in delivering the required software.

The Essence of Practice

How does the practice of software engineering fit in the process activities mentioned above? Namely, communication, planning, modeling, construction and deployment.

George Polya outlines the essence of problem solving, suggests:

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).

Understand the Problem

Who has a stake in the solution to the problem? That is, who are the stakeholders?

What are the unknowns? What data, functions, and features are required to properly solve the problem?

Can the problem be compartmentalized? 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?

Plan the Solution

Have you seen similar problems before? Are there patterns that are recognizable in a potential solution? 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?

Carry Out the Plan

Does the solutions conform to the plan? Is source code traceable to the design model?

Is each component part of the solution provably correct? Has the design and code been reviewed, or better, have correctness proofs been applied to algorithm?

Examine the Result

Is it possible to test each component part of the solution? Has a reasonable testing strategy been implemented?

Does the solution produce results that conform to the data, functions, and features that are required? Has the software been validated against all stakeholder requirements?