UNIT I Introduction to Software Engineering and Process Models

Introduction to Software Engineering

Instructions (Computer Programs)

- Definition:
  - Instructions that provide desired features, function, and performance when executed.
- Supporting Elements:
  - Data structures enabling adequate manipulation of information.
  - Documents describing the operation and use of the programs.

Characteristics of Software

- Software is developed or engineered; it is not manufactured in the classical sense.
- Software does not "wear out."
- Industry is moving toward component-based construction, but most software is custom-built.

Software Engineering

- Definition:
  - The systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software.
- Components:
  - Application of engineering to software.
  - Study of various approaches.

Evolving Role of Software

- Dual Role:
  - As a product: Delivers computing potential embodied by computer hardware or a network of computers.
  - As a vehicle: Information transformer producing, managing, acquiring, modifying, displaying, or transmitting information.
- Transformations by Software:
  - Personal data transformation.

    - Business information management for competitiveness.
    - Gateway to worldwide information networks.
    - Means for acquiring information.
- Significant Changes Over Time:
    - Dramatic improvements in hardware performance.
    - Vast increases in memory and storage capacity.
    - Exotic input and output options.

Historical Perspectives

1970s and 1980s

- Recognition of a "new industrial revolution."
- Microelectronics as part of "the third wave of change" in human history.
- Prediction of a shift to an "information society."
- Information and knowledge focal point for power.
- Importance of the "electronic community" for global knowledge interchange.

1990s

- Power shift and democratization of knowledge due to computers and software.
- Concerns about the decline of the American programmer.
- Information technologies pivotal in the "reengineering of the corporation."

Mid-1990s

- Pervasiveness of computers and software led to concerns and discussions by neo-luddites.

Later 1990s

- Reevaluation of the prospects of the software professional.
- Impact of the Y2K "time bomb" at the end of the 20th century.

2000s

- Discussion on the power of "emergence" in system self-organization.
- Revisiting tragic events of 9/11 and the continuing impact of global terrorism on the IT community.
- Treatise on a "new kind of science" based on sophisticated software simulations.
- Evolution of "the semantic web."

Present Day

- A huge software industry has become a dominant factor in the economies of the industrialized world.

The Changing Nature of Software

Software engineers face ongoing challenges with the evolution of computer software, which can be broadly categorized into seven main types:

1. System Software

- Definition:
  - Collection of programs written to service other programs.
- Characteristics:
  - Heavy interaction with computer hardware.
  - Heavy usage by multiple users.
  - Concurrent operation requiring scheduling, resource sharing, and sophisticated process management.
  - Complex data structures and multiple external interfaces.
- Examples:
  - Compilers, editors, and file management utilities.

2. Application Software

- Definition:
  - Standalone programs solving specific business needs.
- Characteristics:
  - Facilitates business operations or management/technical decision making.
  - Used to control business functions in real-time.
- Examples:
  - Point-of-sale transaction processing, real-time manufacturing process control.

3. Engineering/Scientific Software

- Definition:
  - Applications ranging from astronomy to volcanology.
- Examples:
  - Computer-aided design, system simulation, and other interactive applications.

4. Embedded Software

- **Definition:**
  - Resides within a product or system, used to implement and control features and functions.
- **Examples:**
  - Digital functions in automobiles, dashboard displays, braking systems, etc.

5. Product-line Software

- **Definition:**
  - Designed to provide a specific capability for use by many different customers.
- **Examples:**
  - Word processing, spreadsheets, computer graphics, multimedia, entertainment, database management, personal and business financial applications.

6. Web-Applications

- **Definition:**
  - Evolving into sophisticated computing environments integrated with corporate databases and business applications.

7. Artificial Intelligence Software

- **Definition:**
  - Makes use of nonnumerical algorithms to solve complex problems.
- **Applications:**
  - Robotics, expert systems, pattern recognition, artificial neural networks, theorem proving, and game playing.

New Challenges on the Horizon

1. **Ubiquitous Computing:**
   - Develop systems and application software for communication across vast networks involving small devices, personal computers, and enterprise systems.
2. **Netsourcing:**
   - Architect simple and sophisticated applications providing benefits to targeted end-user markets worldwide.
3. **Open Source:**
   - Build self-descriptive source code and develop techniques for customers and developers to track changes within the software.

4. The "New Economy":
    - Build applications facilitating mass communication and mass product distribution.

Software Myths

Beliefs about software and the process used to build it can be traced to the earliest days of computing. These myths have several attributes that make them insidious.

Management Myths

Myth: We already have a book that's full of standards and procedures for building software - Won't that provide my people with everything they need to know?

- Reality:
    - The book of standards may exist, but its effectiveness depends on its use and awareness.
    - It needs to reflect modern software engineering practices to be beneficial.

Myth: If we get behind schedule, we can add more programmers and catch up.

- Reality:
    - Software development is not a mechanistic process like manufacturing.
    - Adding more people requires time for education, reducing productive development effort.
    - People can be added, but in a planned and well-coordinated manner.

Myth: If I decide to outsource the software project to a third party, I can just relax and let that firm build it.

- Reality:
    - If an organization lacks internal software project management capabilities, outsourcing can lead to struggles.

Customer Myths

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

- Reality:
    - Ambiguous objectives are a recipe for disaster.

- While comprehensive requirements may not always be possible, stability is crucial.

Myth: Project requirements continually change, but change can be easily accommodated because software is flexible.

- Reality:
  - Change impact varies, and it can require additional resources and major design modifications.

Practitioner's Myths

Myth: Once we write the program and get it to work, our jobs are done.

- Reality:
  - Effort after the software is delivered is substantial (60-80%).
  - Completion of the working program is just one part of the software configuration.

Myth: The only deliverable work product for a successful project is the working program.

- Reality:
  - A working program is part of a software configuration, which includes documentation for software support.

Myth: Software engineering will make us create voluminous and unnecessary documentation and will invariably slow us down.

- Reality:
  - Software engineering is about creating quality, not just documents.
  - Better quality reduces rework and leads to faster delivery times.

A Generic View of Process

Software Engineering - A Layered Technology

> Software Engineering - A Layered Technology
>
> ### Software Engineering Layers
>
> Software engineering is structured as a layered technology, with each layer playing a crucial role in the development process.
>
> 1. Tools

Software Engineering is a Layered Technology

Software engineering is built on an organizational commitment to quality, making quality a fundamental focus of the entire process.

1. Process Layer

The process layer is the foundation for software engineering. It serves as the glue that holds the technology layers together. The software engineering process is a framework established for the effective delivery of software engineering technology.

2. Software Layer

The software layer forms the basis for management control of software projects. It establishes the context in which:

- Technical methods are applied.
- Work products are produced.
- Milestones are established.
- Quality is ensured.
- Change is properly managed.

3. Methods Layer

Software engineering methods rely on a set of basic principles governing various areas of technology, including modeling activities. Methods encompass a broad array of tasks, such as:

- Communication.
- Requirements analysis.
- Design modeling.
- Program construction.
- Testing and support.

4. Tools Layer

Software engineering tools provide automated or semi-automated support for the process and methods. When these tools are integrated to allow information

interchange, it establishes a system for supporting software development known as Computer-Aided Software Engineering (CASE).

The integration of tools ensures that information created by one tool can be seamlessly used by another, enhancing efficiency and collaboration in software development.

Software engineering is structured as a layered technology, and any engineering approach within it must be built upon an organizational commitment to quality. The bedrock supporting software engineering is a focus on quality.

Layers of Software Engineering

1. **Quality Focus:**
    - The foundational layer emphasizing a commitment to quality as an organizational principle.
2. **Process Layer:**
    - Serves as the foundation for software engineering, acting as the glue that holds the technology layers together.
    - Defines a framework essential for the effective delivery of software engineering technology.
    - Establishes the context in which:
        - Technical methods are applied.
        - Work products are produced.
        - Milestones are established.
        - Quality is ensured.
        - Change is properly managed.
3. **Software Layer:**
    - Forms the basis for management control of software projects.
    - Provides the context for applying technical methods, producing work products, establishing milestones, ensuring quality, and managing change.
4. **Methods Layer:**
    - Relies on a set of basic principles governing various areas of technology, including modeling activities.
    - Encompasses a broad array of tasks, such as:
        - Communication.
        - Requirements analysis.
        - Design modeling.
        - Program construction.

- Testing and support.

5. Tools Layer:
   - Provides automated or semi-automated support for both the process and the methods.
   - Integration of tools enables information created by one tool to be used by another.
   - Establishes a system for supporting software development, known as Computer-Aided Software Engineering (CASE).

A Process Framework

A Process Framework

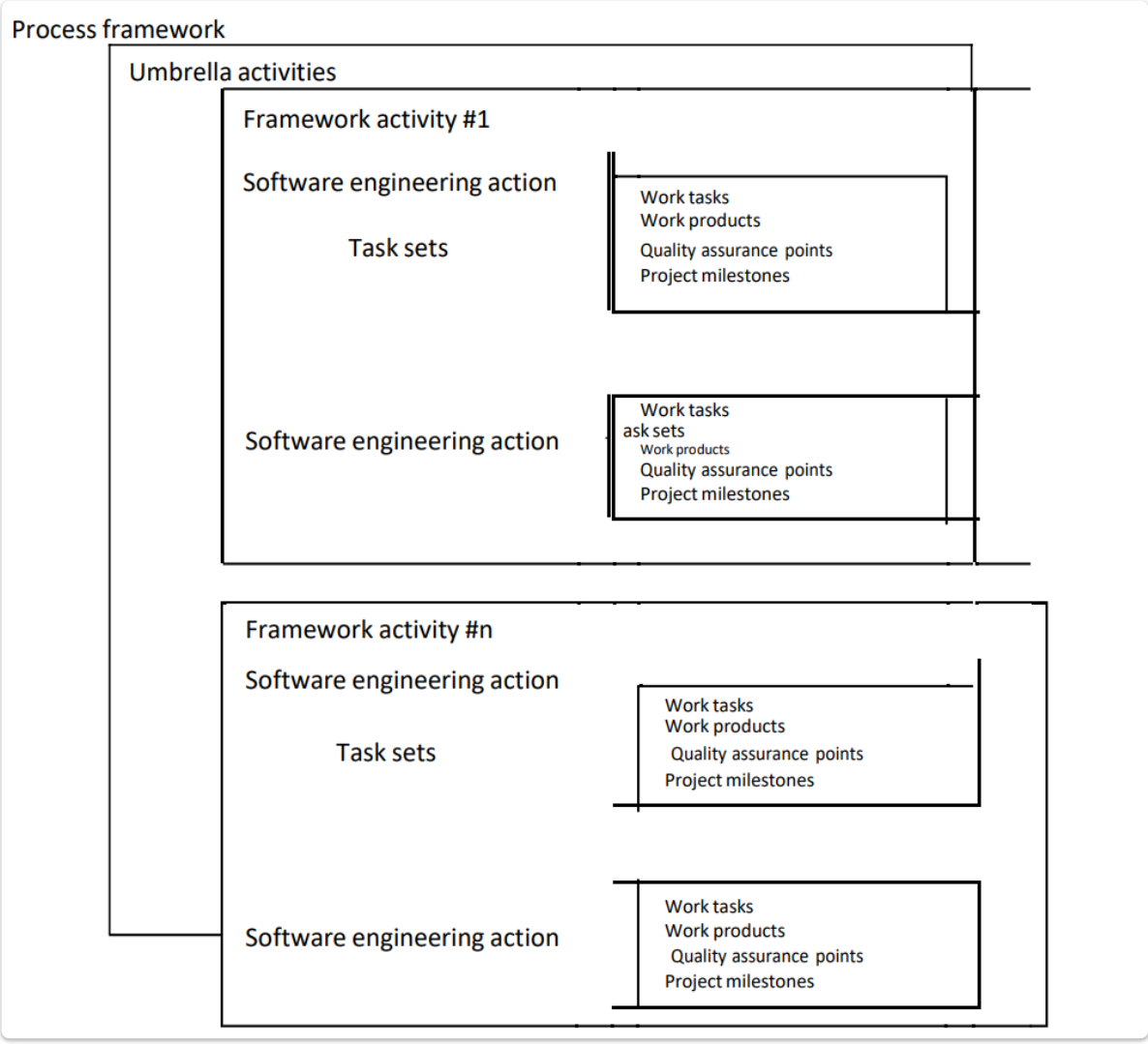
A PROCESS FRAMEWORK

A software process must be established for the effective delivery of software engineering technology. The process framework lays the foundation for a complete software process, identifying a small number of framework activities applicable to all software projects, regardless of size or complexity.

Process Framework Overview

- The process framework encompasses:
  - A set of umbrella activities applicable across the entire software process.
  - Framework activities populated by software engineering actions.
  - Each software engineering action represented by different task sets.

- a process defines "who is doing what, when, and how to reach a certain goal."



Generic Process Framework Components

- Communication Activity
- Planning Activity
- Modeling Activity
  - Analysis Action
    - Requirements Gathering Work Task
    - Elaboration Work Task
    - Negotiation Work Task
    - Specification Work Task
    - Validation Work Task
  - Design Action
    - Data Design Work Task
    - Architectural Design Work Task
    - Interface Design Work Task
    - Component-Level Design Work Task
- Construction Activity

- Deployment Activity

Framework Activities in Software Engineering

Communication

- Overview:
  - Involves heavy communication and collaboration with the customer.
  - Encompasses requirements gathering and other related activities.

Planning

- Overview:
  - Establishes a plan for the software engineering work.
  - Describes:
    - Technical tasks to be conducted.
    - Risks likely to be encountered.
    - Resources required.
    - Work products to be produced.
    - Work schedule.

Modeling

- Overview:
  - Encompasses the creation of models for better understanding of software requirements and design.
  - Composed of two software engineering actions:
    - Analysis:
      - Involves a set of work tasks focused on understanding requirements.
    - Design:
      - Involves work tasks to create a design model.

Construction

- Overview:
  - Combines core generation and testing.
  - Core generation involves creating the actual code.
  - Testing is required to uncover errors in the code.

Deployment

- **Overview:**
  - Software is delivered to the customer.
  - Customer evaluates the delivered product.
  - Provides feedback based on the product's evolution.

Umbrella Activities

A set of umbrella activities plays a crucial role in the software process, providing overarching support:

1. **Software Project Tracking and Control**
   - Assess progress against the project plan and take necessary action to maintain the schedule.
2. **Risk Management**
   - Assess risks that may affect the project outcome or product quality.
3. **Software Quality Assurance**
   - Define and conduct activities to ensure software quality.
4. **Formal Technical Reviews**
   - Assess software engineering work products to uncover and remove errors before propagation.
5. **Measurement**
   - Define and collect process, project, and product measures to assist the team in delivering software that meets customer needs.
6. **Software Configuration Management**
   - Manage the effects of change throughout the software process.
7. **Reusability Management**
   - Define criteria for work product reuse and establish mechanisms to achieve reusable components.
8. **Work Product Preparation and Production**
   - Encompass activities required to create work products such as models, documents, logs, forms, and lists.

Adaptation of Software Process Models

- Intelligent application recognizes that adaptation is essential for success.
- Fundamental differences exist in various process models, including:
  - The overall flow of activities and tasks and their interdependencies.
  - The degree to which work tasks are defined within each framework activity.
  - The identification and requirement of work products.

- The application of quality assurance activities.
- The application of project tracking and control activities.
- The detailed and rigorous description of the process.
- Involvement of the customer and other stakeholders.
- Autonomy given to the software project team.
- The prescription of team organization and roles.

Capability Maturity Model Integration (CMMI)

Capability Maturity Model Integration (CMMI)

THE CAPABILITY MATURITY MODEL INTEGRATION (CMMI)

The CMMI represents a process meta-model in two different ways: As a continuous model and as a staged model. Each process area is formally assessed against specific goals and practices, and is rated according to the following capability levels.

Capability Levels:

Level 0: Incomplete

- The process area is either not performed or does not achieve all goals and objectives defined by CMMI for level 1 capability.

Level 1: Performed

- All specific goals of the process area have been satisfied.
- Work tasks required to produce defined work products are being conducted.

Level 2: Managed

- All level 1 criteria have been satisfied.
- Additionally, all work associated with the process area conforms to an organizationally defined policy.
- All people doing the work have access to adequate resources to get the job done.
- Stakeholders are actively involved in the process area as required.
- All work tasks and work products are "monitored, controlled, and reviewed."

Level 3: Defined

- All level 2 criteria have been achieved.

- The process is "tailored from the organization's set of standard processes according to the organization's tailoring guidelines."
- Contributes work products, measures, and other process-improvement information to the organizational process assets.

Level 4: Quantitatively Managed

- All level 3 criteria have been achieved.
- The process area is controlled and improved using measurement and quantitative assessment.
- Quantitative objectives for quality and process performance are established and used as criteria in managing the process.

Level 5: Optimized

- All level 4 criteria have been achieved.
- Additionally, the process area is adapted and optimized using quantitative means to meet changing customer needs and to continually improve the efficacy of the process area under consideration.

Note: Each level represents a maturity stage, and achieving higher levels indicates a more mature and optimized organizational process.

The CMMI defines each process area in terms of "specific goals" and the "specific practices" required to achieve these goals. Specific practices refine a goal into a set of process-related activities.

Project Planning Process Area

Specific Goals (SG) and Associated Specific Practices (SP):

- SG 1 Establish estimates:
  - SP 1.1 Estimate the scope of the project.
  - SP 1.2 Establish estimates of work product and task attributes.
  - SP 1.3 Define project life cycle.
  - SP 1.4 Determine estimates of effort and cost.
- SG 2 Develop a Project Plan:
  - SP 2.1 Establish the budget and schedule.
  - SP 2.2 Identify project risks.
  - SP 2.3 Plan for data management.
  - SP 2.4 Plan for needed knowledge and skills.
  - SP 2.5 Plan stakeholder involvement.

    - SP 2.6 Establish the project plan.
- SG 3 Obtain commitment to the plan:
    - SP 3.1 Review plans that affect the project.
    - SP 3.2 Reconcile work and resource levels.
    - SP 3.3 Obtain plan commitment.

Generic Goals (GG) and Practices (GP):

- GG 1 Achieve specific goals:
    - GP 1.1 Perform base practices.
- GG 2 Institutionalize a managed process:
    - GP 2.1 Establish an organizational policy.
    - GP 2.2 Plan the process.
    - GP 2.3 Provide resources.
    - GP 2.4 Assign responsibility.
    - GP 2.5 Train people.
    - GP 2.6 Manage configurations.
    - GP 2.7 Identify and involve relevant stakeholders.
    - GP 2.8 Monitor and control the process.
    - GP 2.9 Objectively evaluate adherence.
    - GP 2.10 Review status with higher-level management.
- GG 3 Institutionalize a defined process:
    - GP 3.1 Establish a defined process.
    - GP 3.2 Collect improvement information.
- GG 4 Institutionalize a quantitatively managed process:
    - GP 4.1 Establish quantitative objectives for the process.
    - GP 4.2 Stabilize sub-process performance.
- GG 5 Institutionalize and optimizing process:
    - GP 5.1 Ensure continuous process improvement.
    - GP 5.2 Correct root causes of problems.

Note: Achieving these generic goals is essential for progressing through the capability levels.

PROCESS PATTERNS

The software process can be defined as a collection of patterns that define a set of activities, actions, work tasks, work products, and/or related behaviors required to develop computer software.

Process Pattern Components:

1. **Pattern Name:**
   - Given a meaningful name describing its function within the software process.
2. **Intent:**
   - Briefly describes the objective of the pattern.
3. **Type:**
   - Specifies the pattern type (Task, Stage, Phase).
4. **Initial Context:**
   - Describes conditions under which the pattern applies.
5. **Problem:**
   - Describes the problem to be solved by the pattern.
6. **Solution:**
   - Describes the implementation of the pattern and how it modifies the initial state of the process.
7. **Resulting Context:**
   - Describes the conditions that result once the pattern has been successfully implemented.
8. **Known Uses:**
   - Indicates specific instances in which the pattern is applicable.

Process patterns provide an effective mechanism for describing any software process. They enable a software engineering organization to develop a hierarchical process description starting at a high level of abstraction. Once developed, patterns can be reused for defining process variants, allowing teams to use them as building blocks for customized process models.

PROCESS ASSESSMENT

The existence of a software process is no guarantee that software will be delivered on time, meet customer needs, or exhibit the technical characteristics leading to long-term quality. Additionally, assessing the process itself is essential to ensure it meets basic criteria for successful software engineering.

Software Capability

- **Motivation:** Identifies capabilities and risks.
- **Lead:** Software Lead.

Several approaches to software process assessment have been proposed:

1. **Standards CMMI Assessment Method for Process Improvement (SCAMPI):**
    - Five-step process assessment model (Initiating, Diagnosing, Establishing, Acting, Learning).
    - Uses SEI CMMI as the basis for assessment.
2. **CMM Based Appraisal for Internal Process Improvement (CBA IPI):**
    - Diagnostic technique for assessing the relative maturity of a software organization.
    - Uses SEI CMM as the basis for assessment.
3. **SPICE (ISO/IEC15504) Standard:**
    - Defines requirements for software process assessments.
    - Aims to assist organizations in objectively evaluating the efficacy of any defined software process.
4. **ISO 9001:2000 for Software:**
    - Generic standard applicable to organizations seeking to improve overall product, system, or service quality.
    - Directly applicable to software organizations and companies.

PERSONAL AND TEAM PROCESS MODELS

The best software process is one that is close to the people doing the work. Each software engineer or team would create a process that fits their needs while meeting broader organizational needs.

Personal Software Process (PSP)

PSP emphasizes personal measurement of work product and resultant quality. The process model includes:

1. **Planning:**
    - Isolates requirements, develops size and resource estimates, and makes defect estimates.
    - Identifies development tasks and creates a project schedule.
2. **High-Level Design:**
    - Develops external specifications for components, creates component designs.
    - Prototypes built when uncertainty exists.
3. **High-Level Design Review:**
    - Applies formal verification methods to uncover errors in the design.
    - Maintains metrics for all important tasks and work results.
4. **Development:**

- Refines and reviews component-level design, generates, reviews, compiles, and tests code.
- Maintains metrics for all important tasks and work results.

5. **Postmortem:**
   - Determines the effectiveness of the process using collected measures and metrics.
   - Provides guidance for modifying the process to improve effectiveness.

PSP stresses early error identification and a metrics-based approach to software engineering.

Team Software Process (TSP)

TSP aims to build self-directed project teams that organize themselves to produce high-quality software. Objectives include:

- Building self-directed teams that plan and track their work, establish goals, and own their processes and plans.
- Coaching and motivating teams to sustain peak performance.
- Accelerating software process improvement, making CMM level 5 behavior normal and expected.

TSP defines the following framework activities: launch, high-level design, implementation, integration and test, and postmortem.

A self-directed team in TSP:

- Defines roles and responsibilities for each team member.
- Tracks quantitative project data.
- Identifies a team process appropriate for the project.
- Develops a strategy for implementing the process.
- Defines local standards applicable to the team's software engineering work.
- Continually assesses and reacts to risk.
- Tracks, manages, and reports project status.

TSP utilizes scripts, forms, and standards to guide team members. Scripts define specific process activities and detailed work functions.

Each project is "launched" using a sequence of tasks, including reviewing project objectives, establishing team roles, defining the development process, making a quality plan, and setting quality targets.

> This emphasis on both personal and team-oriented processes aims to improve software development outcomes.

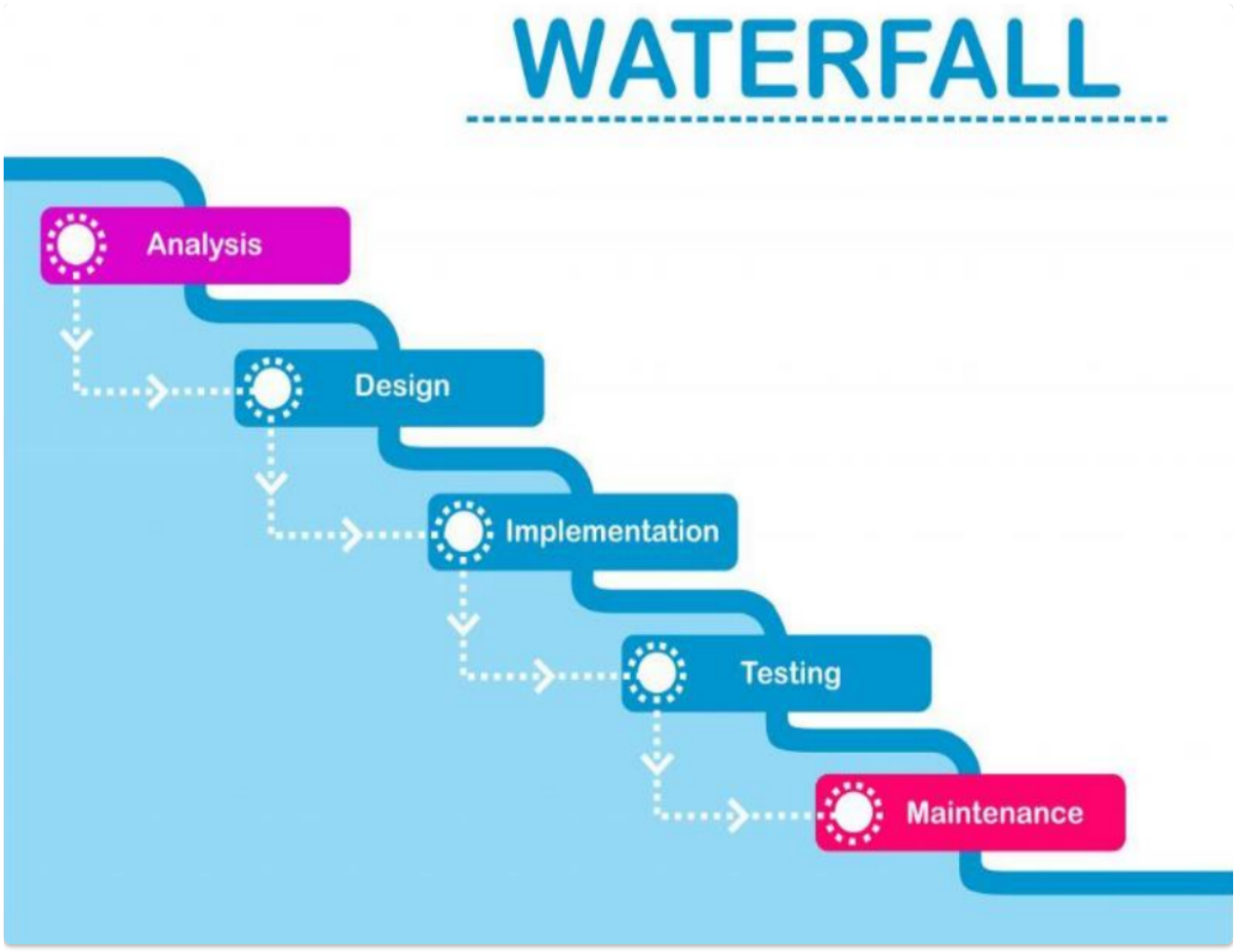
Process Models

The Waterfall Model Process Models > THE WATERFALL MODEL

Spiral Model Process Models > THE SPIRAL MODEL

Agile Methodology Process Models > Agile Process Model

> Process Models
>
> # PROCESS MODELS
>
> Prescriptive process models define a set of activities, actions, tasks, milestones, and work products required for engineering high-quality software. Although not perfect, these models offer a useful roadmap for software engineering work. A prescriptive process model populates a process framework with explicit task sets for software engineering actions.
>
> ## THE WATERFALL MODEL
>
> The waterfall model, also known as the classic life cycle, proposes a systematic sequential approach to software development. It begins with customer specification of requirements and progresses through planning, modeling, construction, and deployment.



Context:

- Used when requirements are reasonably well understood.

Advantage:

- Useful in situations where requirements are fixed, and work proceeds to completion in a linear manner.

Challenges:

1. **Sequential Flow:** Real projects rarely follow the sequential flow proposed by the model. While it can accommodate iteration, it does so indirectly, leading to potential confusion as changes occur.
2. **Requirement Understanding:** Difficulty in explicitly stating all requirements. The model struggles to accommodate the natural uncertainty present at the beginning of many projects.
3. **Customer Patience:** The customer needs patience as a working version of the software is not available until late in the project time-span. If a major error is undetected, it can be disastrous until the program is reviewed.

The waterfall model's rigidity in handling changes and the need for explicit requirements make it less suitable for dynamic and uncertain project environments.

While it provides a clear structure, its applicability is limited in scenarios where flexibility and adaptability are crucial.

Evolutionary Process Model

Evolutionary process models, characterized by iterative development, aim to produce increasingly more complete versions of software with each iteration. Two notable evolutionary models are Prototyping and the Spiral Model.

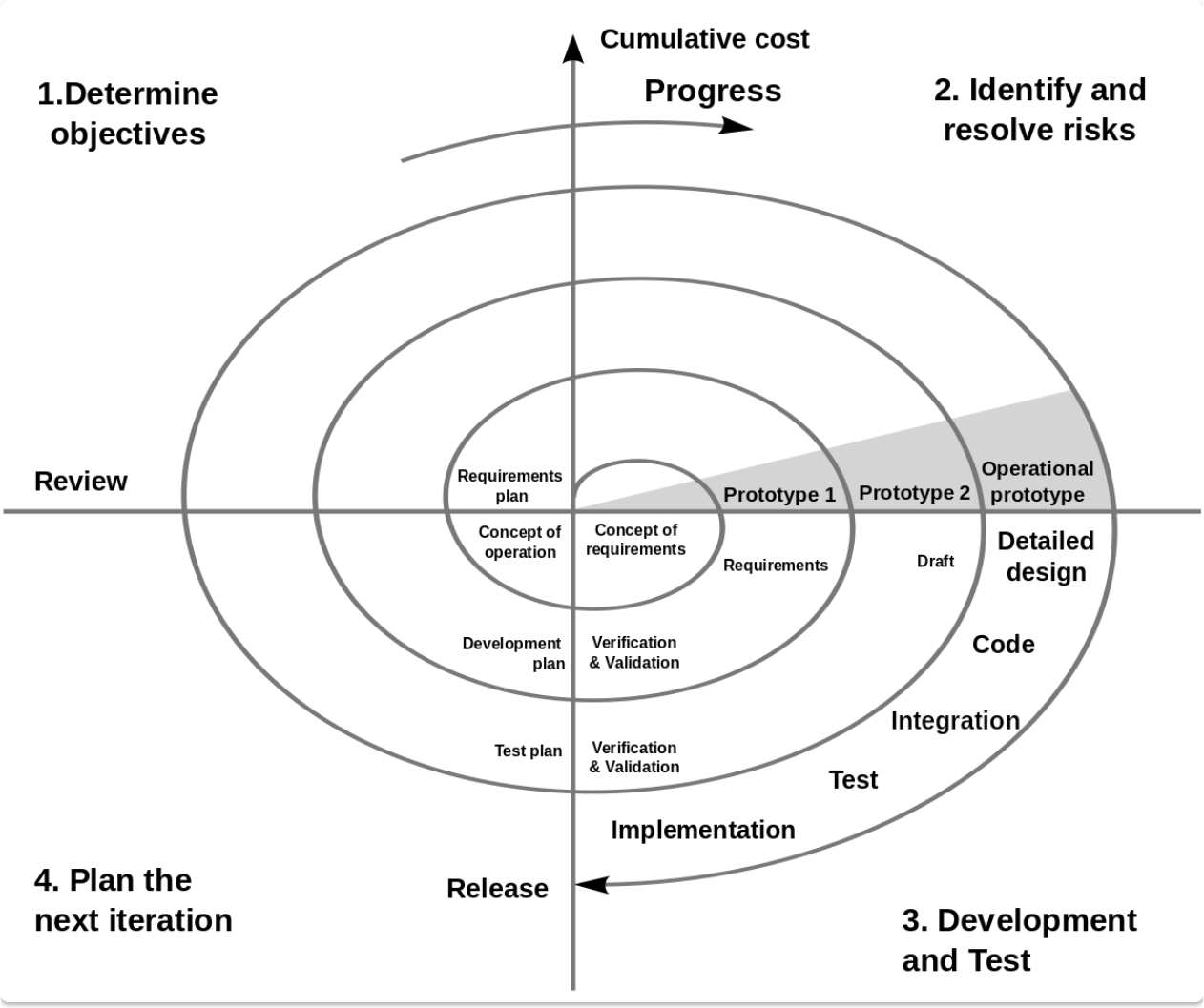
- Software evolves over time.
- Business and product requirements often change during development.
- Straight-line paths to an end product are unrealistic.
- Evolutionary models are iterative and applicable to modern applications.

Types of Evolutionary Models

1. Prototyping
2. Spiral Model
3. Concurrent Development Model

THE SPIRAL MODEL

The Spiral Model, proposed by Boehm, combines the iterative nature of prototyping with the systematic aspects of the waterfall model. It facilitates the development of software in evolutionary releases through multiple iterations. Each iteration results in increasingly more complete versions of the software.



Key Features:

- **Iterative Evolution:** Early iterations may involve a paper model or prototype, while later iterations produce more sophisticated software versions.
- **Anchor Points:** Milestones marked along the spiral, representing work products and conditions attained for each evolutionary pass.
- **Adjustable Planning:** Adjustments to project plans, costs, and schedules are made based on customer feedback after each iteration.

Context:

- Applicable throughout the entire life cycle of an application, from concept development to maintenance.

Advantages:

- Potential for rapid development of increasingly complete software versions.
- Realistic approach for large-scale systems.
- Utilizes prototyping for risk reduction at any project stage.

Drawbacks:

- Difficult to convince customers of controllability.
- Requires significant risk assessment expertise.
- Dependence on expertise for risk management; major risks not uncovered may lead to problems.

Agile Process Model

- The meaning of Agile is swift or versatile. "Agile process model" refers to a software development approach based on iterative development.
- Agile methods break tasks into smaller iterations, or parts do not directly involve long-term planning.
- The project scope and requirements are laid down at the beginning of the development process.
- Plans regarding the number of iterations, the duration, and the scope of each iteration are clearly defined in advance.
- Each iteration is considered as a short time "frame" in the Agile process model, which typically lasts from one to four weeks.
- The division of the entire project into smaller parts helps to minimize the project risk and to reduce the overall project delivery time requirements.
- Each iteration involves a team working through a full software development life cycle including planning, requirements analysis, design, coding, and testing

before a working product is demonstrated to the client.

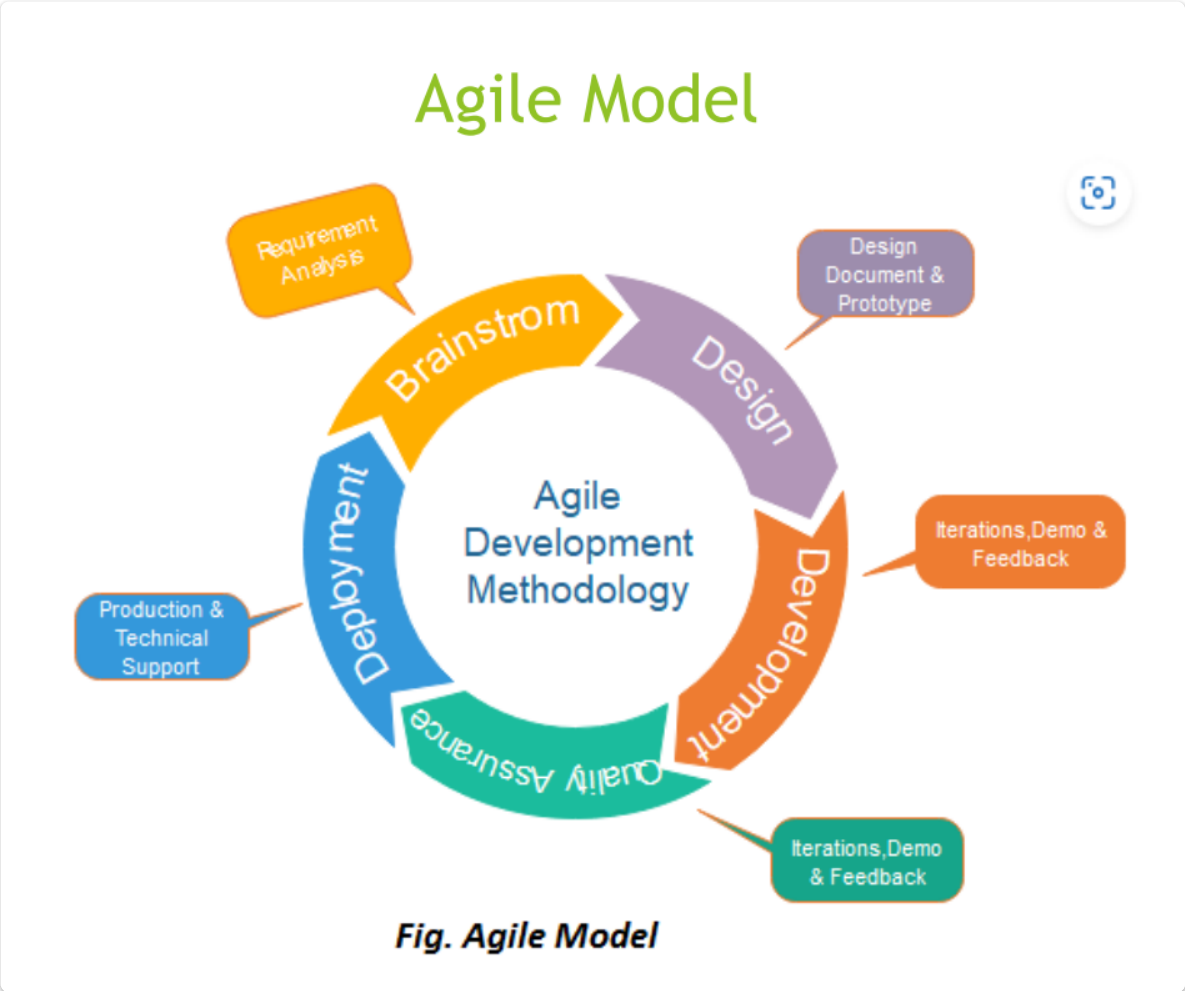


Fig. Agile Model

Phases of Agile Model:

1. Requirements Gathering
2. Design the Requirements
3. Construction/Iteration
4. Testing/Quality Assurance
5. Deployment
6. Feedback

Requirements Gathering:

In this phase, you must define the requirements. You should explain business opportunities and plan the time and effort needed to build the project. Based on this information, you can evaluate technical and economic feasibility.

Design the Requirements:

When you have identified the project, work with stakeholders to define requirements. You can use the user flow diagram or the high-level UML diagram to show the work of new features and show how it will apply to your existing system.

Construction/Iteration:

When the team defines the requirements, the work begins. Designers and developers start working on their project, which aims to deploy a working product. The product will undergo various stages of improvement, so it includes simple, minimal functionality.

Testing:

In this phase, the Quality Assurance team examines the product's performance and looks for bugs.

Deployment:

In this phase, the team issues a product for the user's work environment.

Feedback:

After releasing the product, the last step is feedback. In this, the team receives feedback about the product and works through the feedback.

Advantages (Pros) of Agile Method:

- Frequent Delivery
- Face-to-Face Communication with clients.
- Efficient design and fulfills the business requirement.
- Anytime changes are acceptable.
- It reduces total development time.

Disadvantages (Cons) of Agile Model:

- Due to the shortage of formal documents, it creates confusion and crucial decisions taken throughout various phases can be misinterpreted at any time by different team members.
- Due to the lack of proper documentation, once the project completes and the developers allotted to another project, maintenance of the finished project can become a difficulty.