SOFTWARE ENGINEERING & PROJECT MANAGEMENT

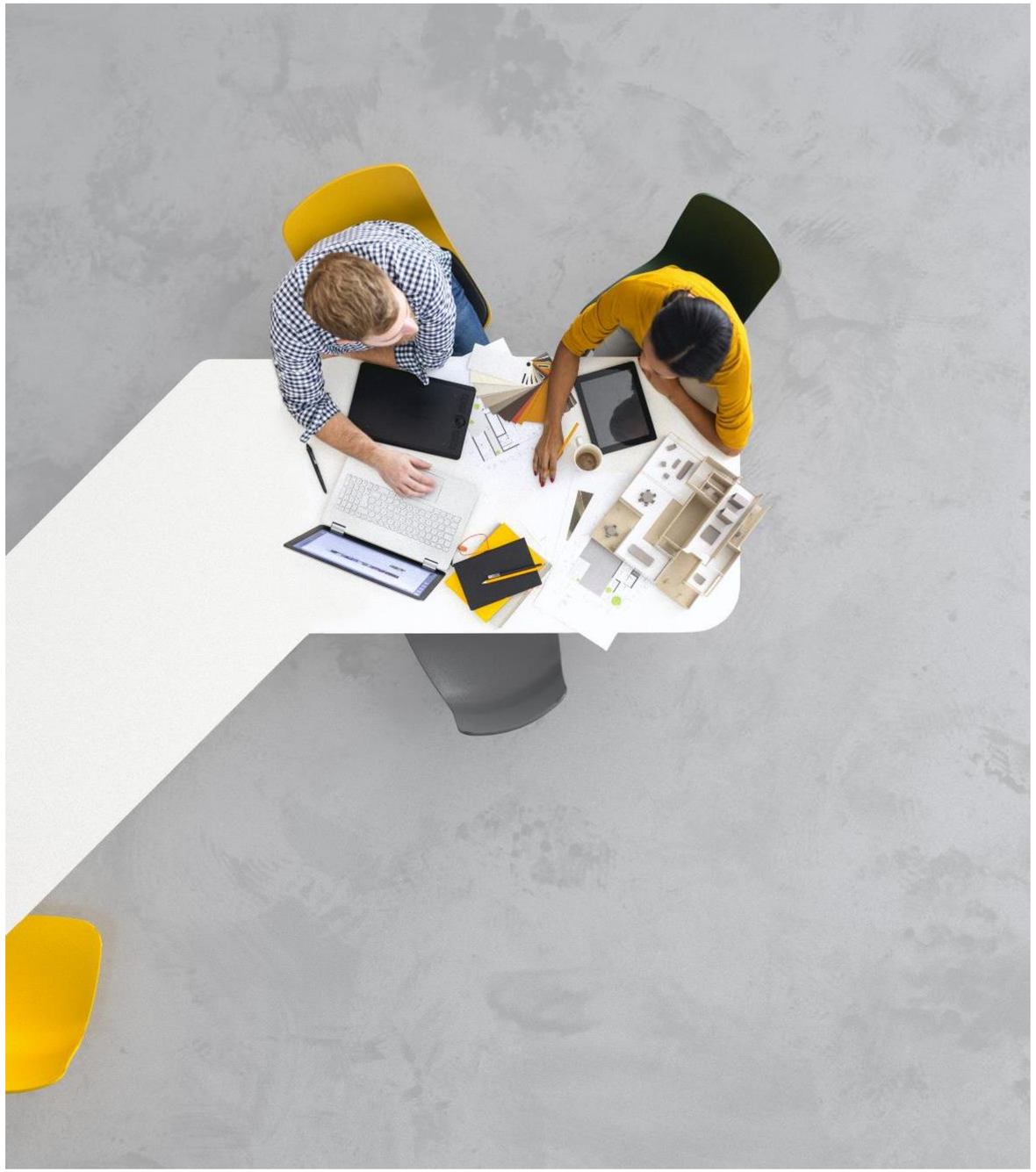
Presented By: Dr. Prakhyath Rai

Module - I

Introduction: The evolving role of software, The changing nature of software, Software engineering, A Process Framework, Process Patterns, Process Assessment, Personal and Team Process Models, Process Technology, Product and Process.

Process Models: Prescriptive models, Waterfall model, Incremental process models, Evolutionary process models, Specialized process models.

Requirements Engineering: Requirements Engineering Task, Initiating the Requirement Engineering process, Eliciting Requirements, developing use cases, Building the analysis model, Negotiating Requirements, Validating Requirements, Software Requirement Document (Sec 4.2).Case Tools(Textbook 5).



Software

Software is a computer programs along with the associated documents and the configuration data that make these programs operate correctly.

A **program** is a set of instructions (written in form of human-readable code) that performs a specific task

Software is:

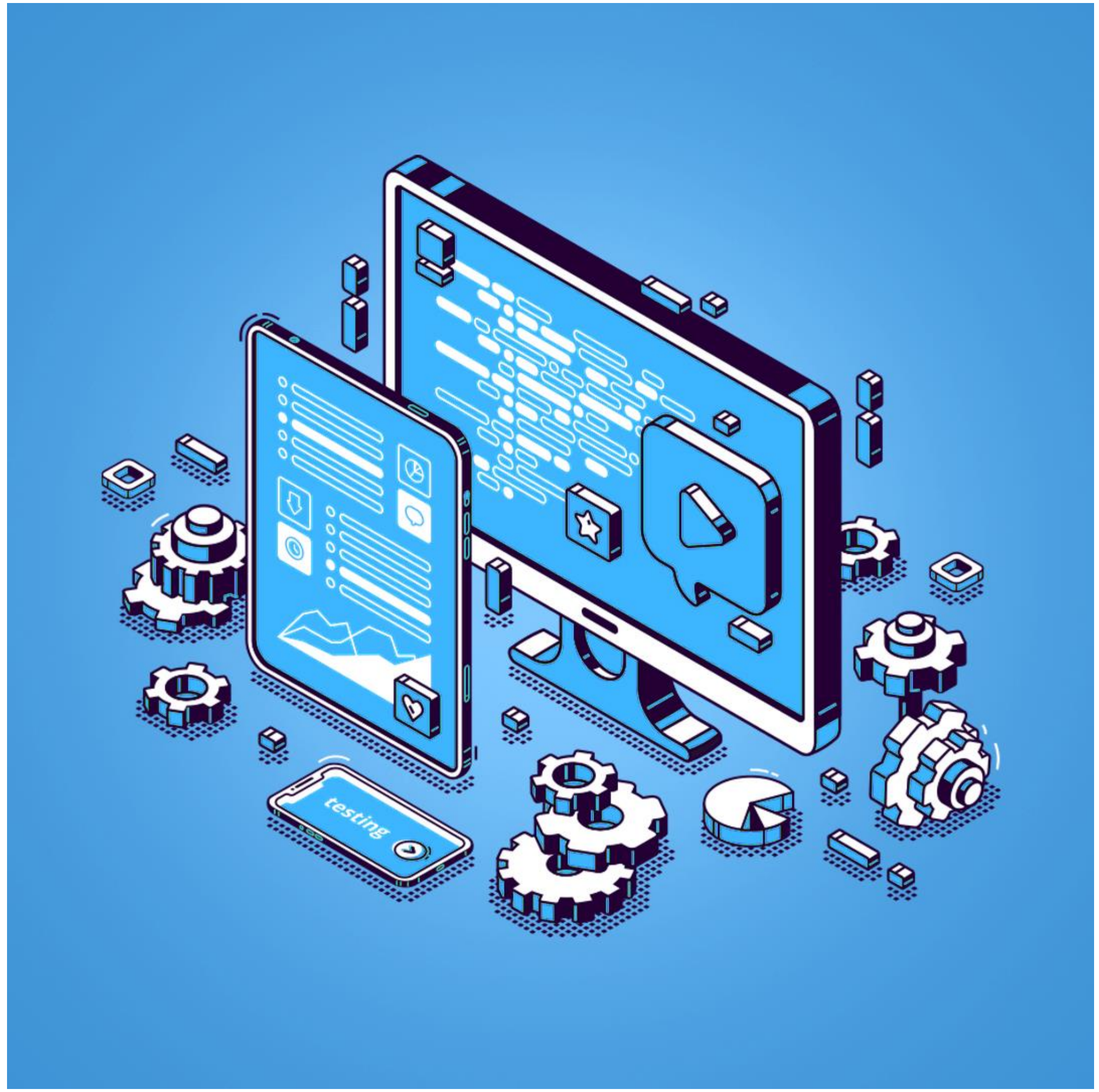
(1) instructions (computer programs) that when executed provide desired features, function, and performance;

(2) data structures that enable the programs to adequately manipulate information, and

(3) descriptive information in both hard copy and virtual forms that describes the operation and use of the programs

Software Types - Evolving Role

- **As a Product:** It delivers the computing potential embodied by computer Hardware or by a network of computers.
- **As a Vehicle for Delivering Product:** It is information transformer-producing, managing, acquiring, modifying, displaying, or transmitting information that can be as simple as single bit or as complex as a multimedia presentation.



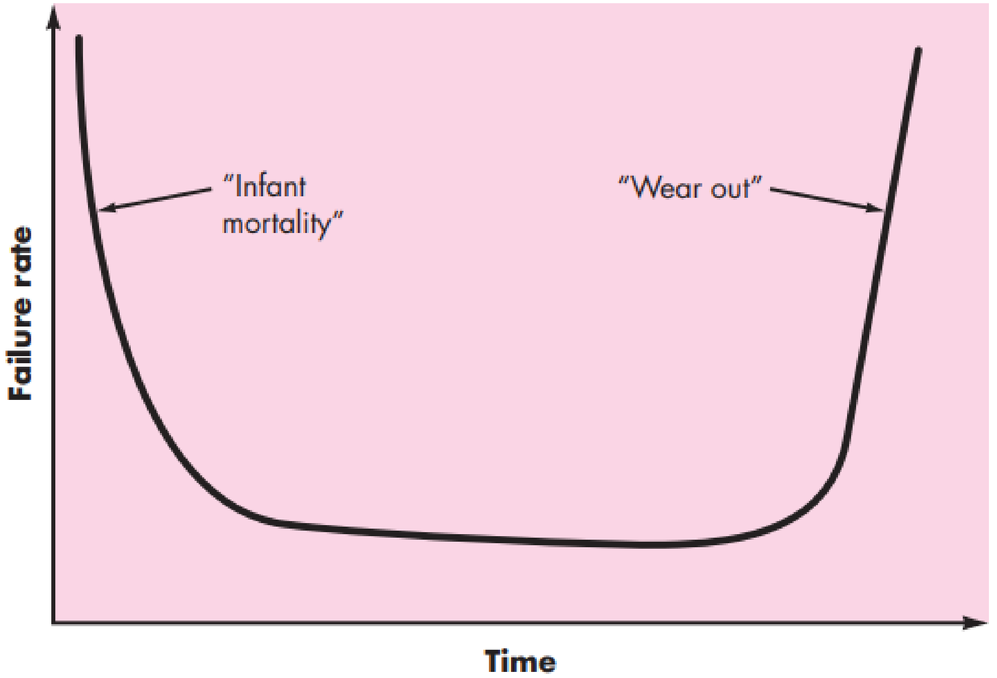
- **System Software:** Designed to provide a platform for running application software and to manage computer hardware resources efficiently. Example: Operating Systems, Device Drivers, Firmware etc.
- **Application Software:** Application software is a type of software designed to perform specific tasks or functions for end-users. Example: Word Processors, Web Apps, Multimedia Players etc.

Characteristics of Software

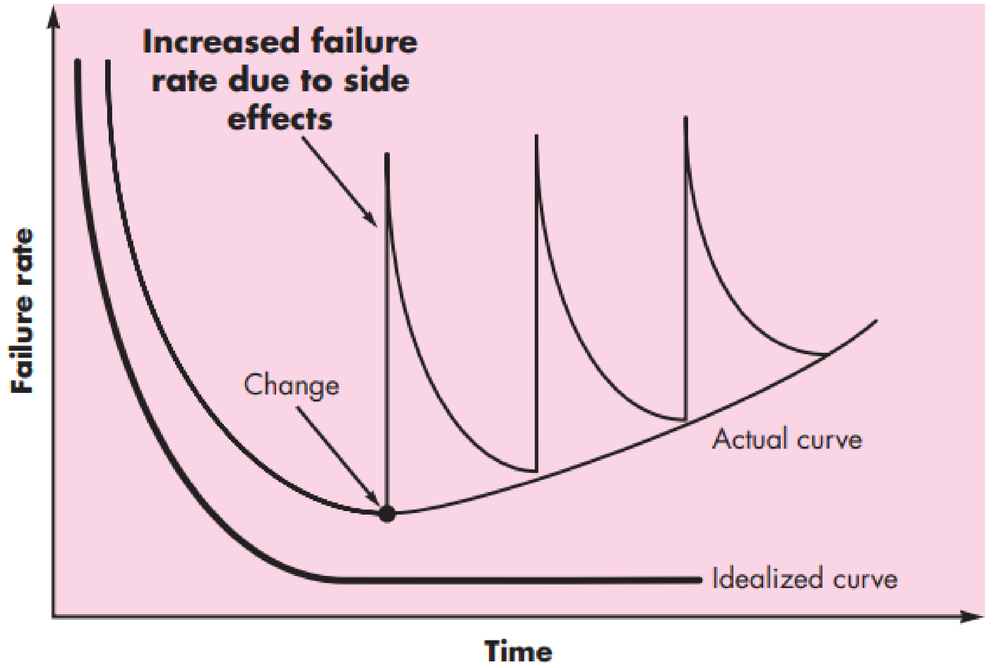
1. Software is developed or engineered; it is not manufactured in the classical sense.
2. Software doesn't "wear out"
3. Although the industry is moving toward component-based construction, most software continues to be custom built.

Nature of Software

1. System software
2. Application software
3. Engineering/scientific software
4. Embedded software
5. Product-line software
6. Web applications
7. Artificial intelligence software



Hardware



Software

Challenges

1. **Open-world computing** – with wireless networking design of software to communicate across devices

2. **Net sourcing** – aid of www to target end-users' market worldwide

3. **Open source** - distribution of source code for systems applications to contribute to its development (Challenge to collate the new changes to customers and developers)

Legacy Software

Challenges

Legacy software refers to applications or systems that have been in use for a long time and may be outdated in terms of technology or design but are still relied upon by organizations due to their critical functions or historical significance.

- The software must be adapted to meet the needs of new computing environments or technology.
- The software must be enhanced to implement new business requirements.
- The software must be extended to make it interoperable with other more modern systems or databases.
- The software must be re-architected to make it viable within a network environment

Software Engineering

Need

- understand the problem before a software solution is developed
- Changing Demands - design becomes a pivotal activity
- Software should exhibit high quality
- software should be maintainable

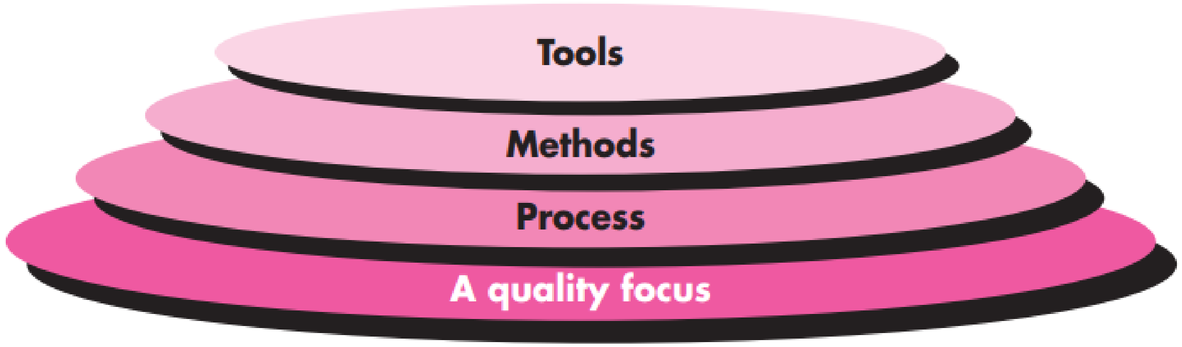
realities lead to

[Software Engineering is] the establishment and use of sound engineering principles to obtain economical software that is reliable and works efficiently on real machines

Software Engineering:

(1)The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software.

(2) The study of approaches as in (1).



Software Engineering Layers

Tools

- provide automated or semiautomated support for the process and the methods – Computer aided Software Engineering

Methods

- provide the technical how-to's for building software
- communication, requirements analysis, design modelling, program construction, testing, and support

Process

- Defines a framework that must be established for effective delivery of software engineering technology

A Quality Focus

- Emphasis on ensuring the quality of software delivered

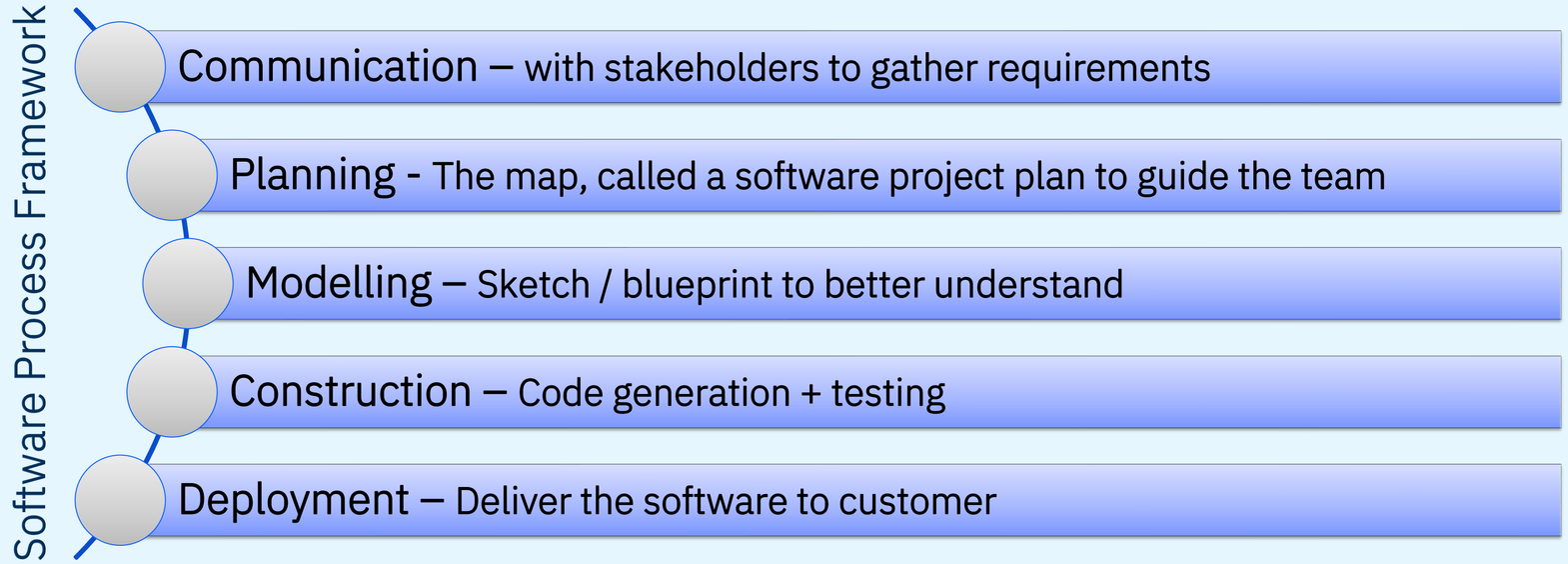
The Software Process

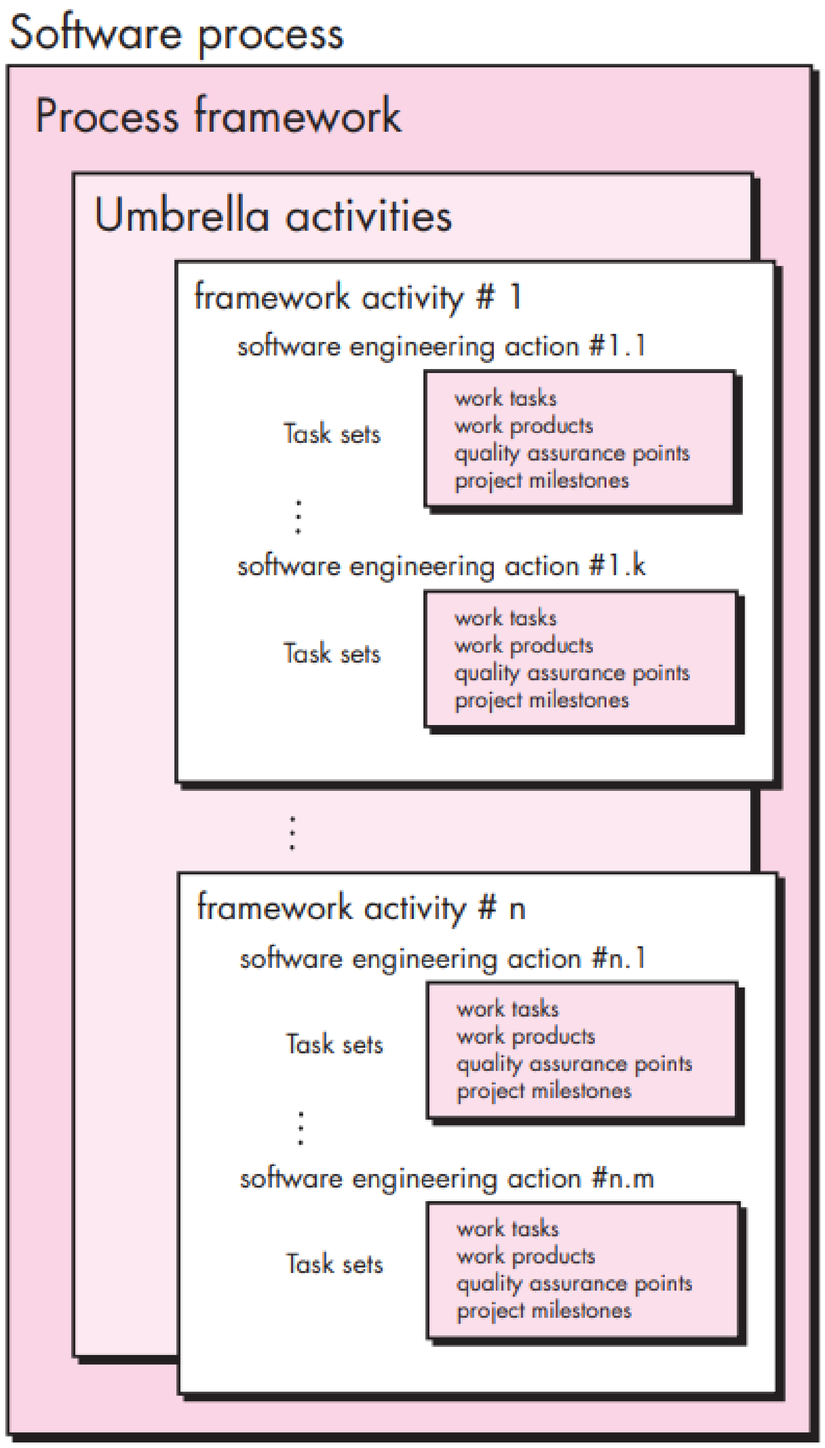
A process is a collection of activities, actions, and tasks that are performed when some work product is to be created.

Activities: *Activity strives to achieve a* ***broad objective*** *(e.g., communication with stakeholders) and is applied regardless of the application domain, size of the project, complexity of the effort, or degree of rigor with which software engineering is to be applied – Adaptable approach followed by software team not a rigid prescription*

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

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





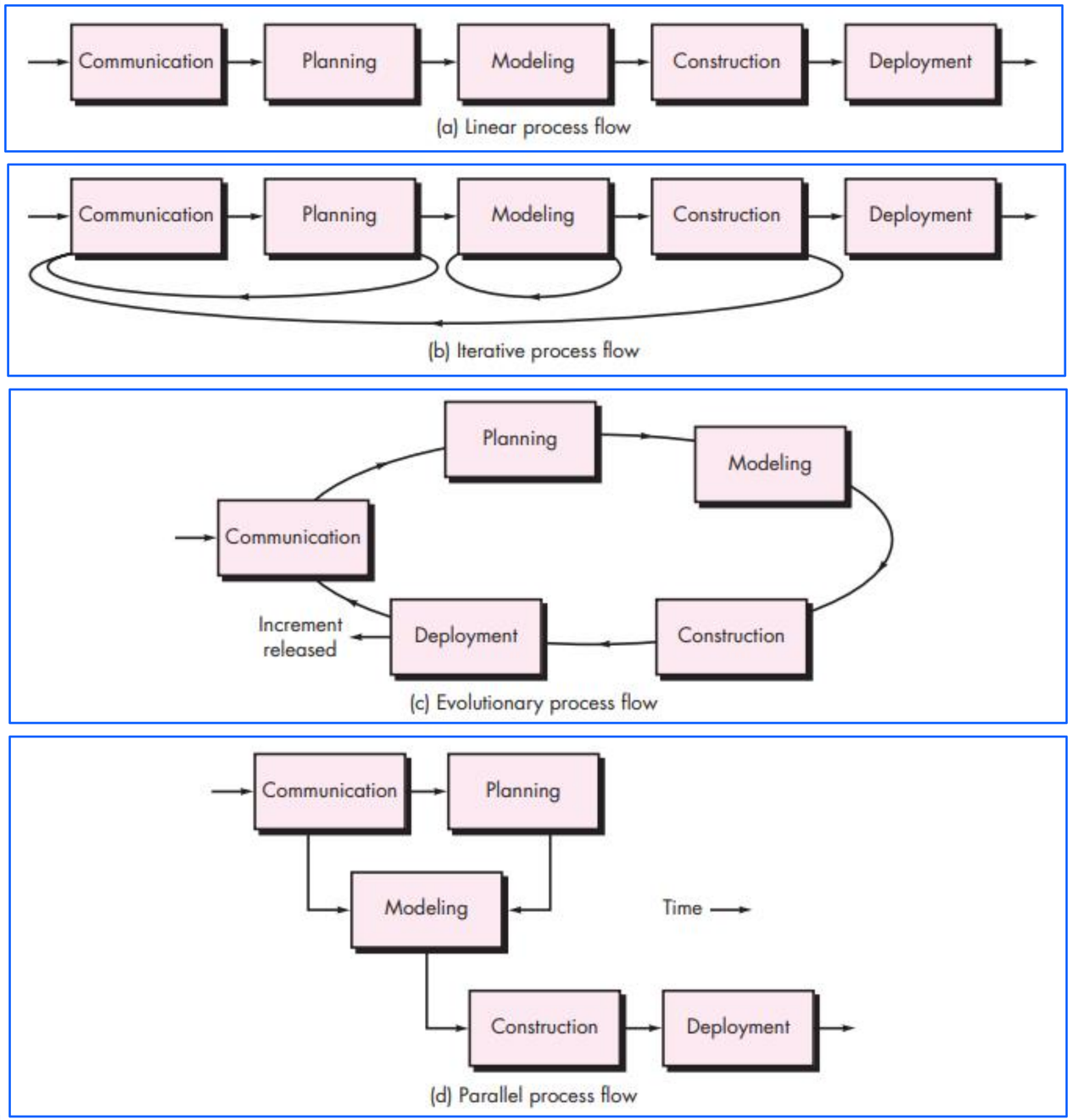
Software Process Framework

- Umbrella Activities

1. Software project tracking and control – Assess project plan + Action to maintain schedule
2. Risk management – Assess risks which may impact quality or outcome
3. Software quality assurance – Activities required to ensure software quality
4. Technical reviews - assesses software engineering work products
5. Measurement - defines and collects process, project, and product measures to check deliverables
6. Software configuration management - manages the effects of change
7. Reusability management - defines criteria for work product reuse
8. Work product preparation and production - encompasses the activities required to create work products – models + logs etc.

Software Process Flows

- A linear process flow executes each of the five framework activities in sequence, beginning with communication and culminating with deployment.
- An iterative process flow repeats one or more of the activities before proceeding to the Next.
- An evolutionary process flow executes the activities in a “circular” manner. Each circuit through the five activities leads to a more complete version of the software.
- A parallel process flow executes one or more activities in parallel with other activities (e.g., modelling for one aspect of the software might be executed in parallel with construction of another aspect of the software).



Process Flows

Process Pattern

- Solutions to common problems encountered during software development, which facilitate quick resolution of issues within software teams.
- A process pattern describes a process-related problem that is encountered during software engineering work, identifies the environment in which the problem has been encountered, and suggests one or more proven solutions to the problem.

Types of Process Patterns

Stage Patterns: Problems associated with framework activities – E.g.: EstablishingCommunication

Task Patterns: Problems related to specific software engineering tasks – E.g.: RequirementsGathering

Phase Patterns: Sequences of framework activities within the process – E.g.: SpiralModel or Prototyping

Pattern Template Overview

Pattern Name: Meaningful name describing the pattern.

Forces: Environment and issues influencing the problem.

Initial Context: Conditions before the pattern initiation.

Problem: Specific issue addressed by the pattern.

Solution: How to implement the pattern successfully.

Resulting Context: Conditions after pattern implementation.

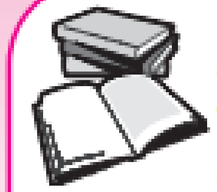
Related Patterns: Patterns directly associated with this one.

Known Uses and Examples: Instances where the pattern is applicable.

Benefits of Process Patterns

- Effective mechanism for addressing process-related problems.
- Enables hierarchical process description from high-level to detailed tasks.
- Facilitates reusability and customization of process models.

INFO



An Example Process Pattern

The following abbreviated process pattern describes an approach that may be applicable when stakeholders have a general idea of what must be done but are unsure of specific software requirements.

Pattern name. **RequirementsUnclear**

Intent. This pattern describes an approach for building a model (a prototype) that can be assessed iteratively by stakeholders in an effort to identify or solidify software requirements.

Type. Phase pattern.

Initial context. The following conditions must be met prior to the initiation of this pattern: (1) stakeholders have been identified; (2) a mode of communication between stakeholders and the software team has been established; (3) the overriding software problem to be solved has been identified by stakeholders; (4) an initial understanding of project scope, basic business requirements, and project constraints has been developed.

Problem. Requirements are hazy or nonexistent, yet there is clear recognition that there is a problem to be solved, and the problem must be addressed with a software solution. Stakeholders are unsure of what they want; that is, they cannot describe software requirements in any detail.

Solution. A description of the prototyping process would be presented here and is described later in Section 2.3.3.

Resulting context. A software prototype that identifies basic requirements (e.g., modes of interaction, computational features, processing functions) is approved by stakeholders. Following this, (1) the prototype may evolve through a series of increments to become the production software or (2) the prototype may be discarded and the production software built using some other process pattern.

Related patterns. The following patterns are related to this pattern: **CustomerCommunication, IterativeDesign, IterativeDevelopment, CustomerAssessment, RequirementExtraction.**

Known uses and examples. Prototyping is recommended when requirements are uncertain.

Process Assessments

Challenges:

- No guarantee of timely delivery, meeting customer needs, or ensuring long-term quality.
- Coupling process patterns with solid software engineering practice is essential.

Importance:

- Ensuring adherence to basic process criteria for successful software engineering.
- Integrating process patterns with solid engineering practices is crucial for success.

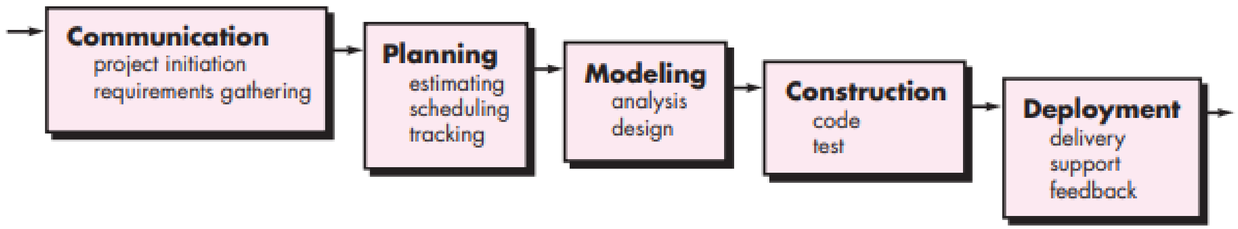
Techniques:

- SCAMPI (Standard CMMI Assessment Method for Process Improvement): Five-step assessment model based on SEI CMMI. **Five phases:** initiating, diagnosing, establishing, acting, and learning.
- CBA IPI (CMM-Based Appraisal for Internal Process Improvement): Diagnostic technique for assessing organizational maturity.
- SPICE (ISO/IEC15504): Standard defining requirements for software process assessment.
- ISO 9001:2000 for Software: Generic standard applicable to improving overall quality in software organizations.

Prescriptive Process Models

- Prescriptive process models were originally proposed to bring order to the chaos of software development.
- *"prescriptive" because they prescribe a set of process elements—framework activities, software engineering actions, tasks, work products, quality assurance, and change control mechanisms for each project*

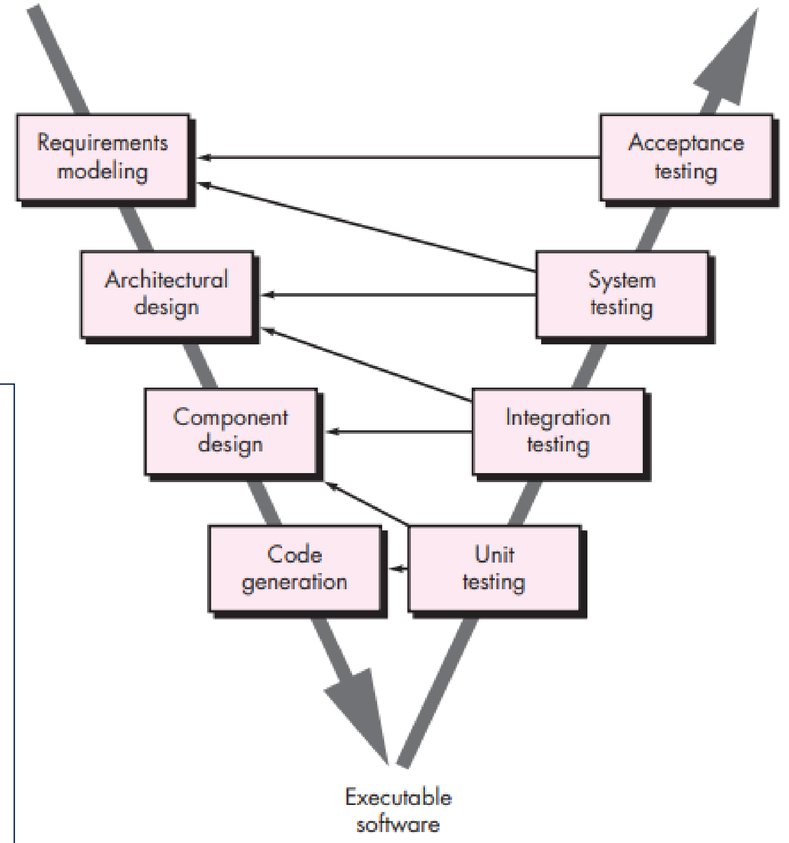
1. The Waterfall Model



The Waterfall model is a sequential software development process, where progress is seen as flowing steadily downwards through several phases. Each phase must be completed before the next one begins.

Problems Encountered:

- Rarely follows sequential flow in real projects, causing confusion with changes.
- Difficulty accommodating natural uncertainty in initial requirements.
- Customer patience required, as working versions are not available until late in the project.



The V Model

2. Incremental Process Models

Situation: Initial requirements are well-defined, but overall scope requires a non-linear process.

Need: Quick delivery of limited software functionality followed by refinement and expansion in later releases.

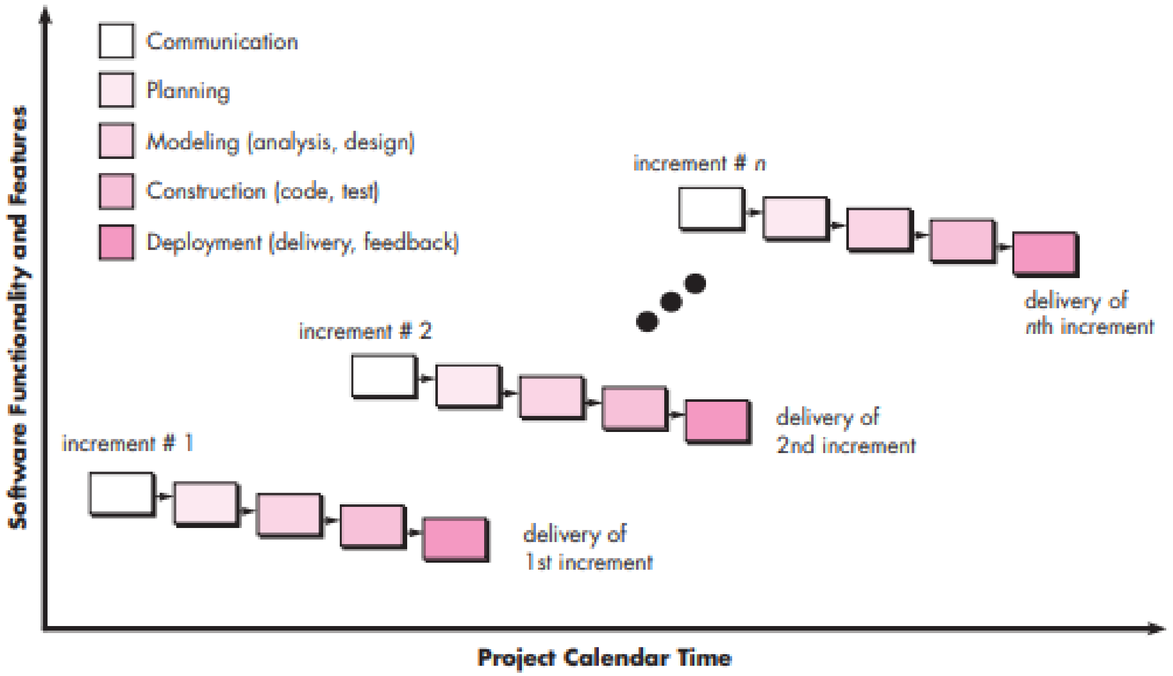
Focus of Incremental Process

- Delivery of operational product with each increment.
- Early Increments: Stripped-down versions providing capability and serving user needs.

Benefits of Incremental Development

Resource Management: Useful when staffing is insufficient for a complete implementation by the business deadline.

Risk Management: Increments can be planned to manage technical risks, ensuring timely delivery of partial functionality.



Example of Incremental Development

Word-processing software development:

First Increment: Basic file management, editing, and document production functions.

Subsequent Increments: More sophisticated capabilities added gradually.

Use of Prototyping: Incorporation into process flow for any increment.

3. Evolutionary Process Models

3.1: Prototyping

Iterative model for developing increasingly complete versions of software.

Process:

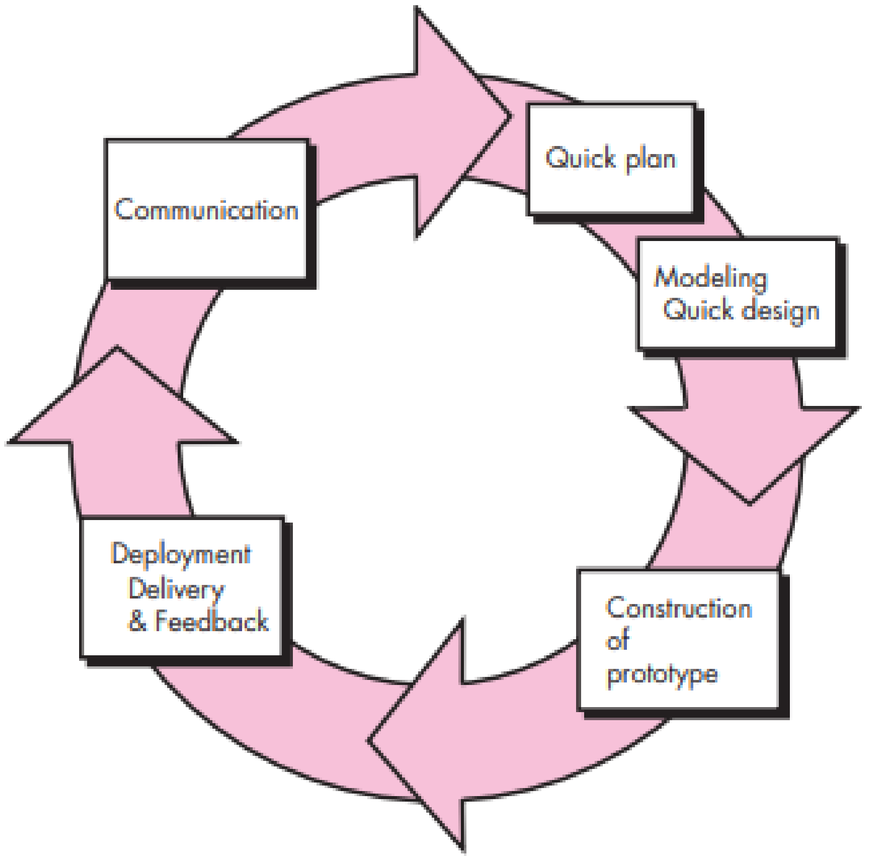
- Begins with communication to define objectives and requirements.
- Quick design and construction of prototype.
- Deployment and evaluation by stakeholders, leading to requirement refinement.

Benefits:

- Provides stakeholders a feel for the actual system.
- Immediate development for developers.

Challenges:

- Stakeholders may perceive prototype as a working product.
- Implementation compromises may occur for quick prototype development.



The prototyping paradigm

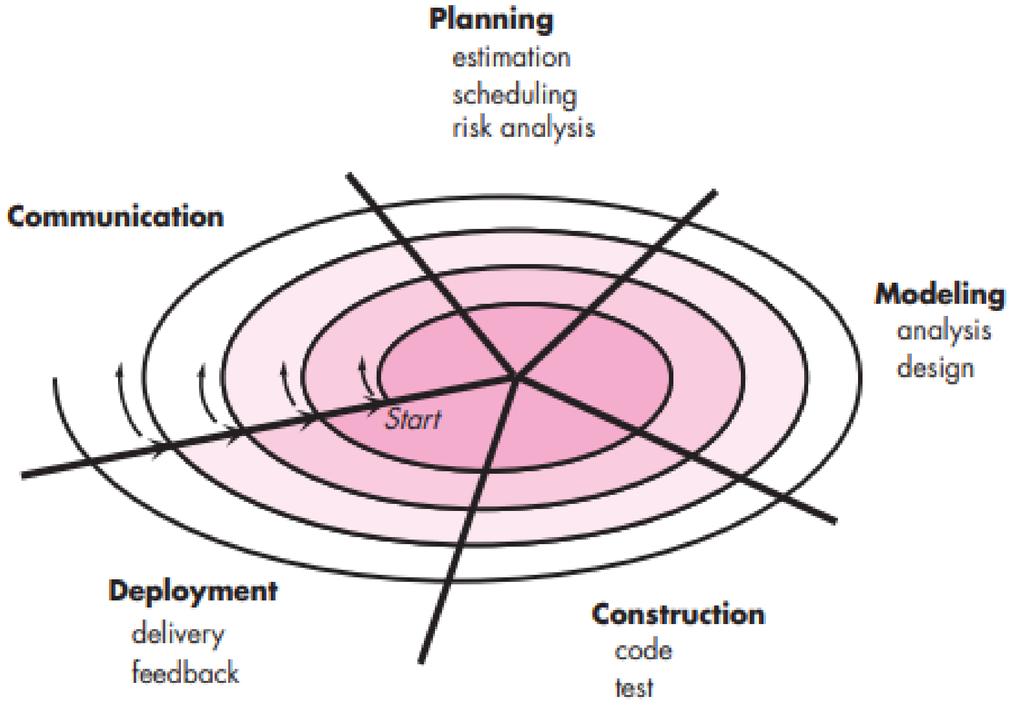
3. Evolutionary Process Models

3.2: Spiral Model

Proposed by Barry Boehm, combines iterative nature of prototyping with systematic aspects of the waterfall model.

Process:

- Divided into framework activities, each representing a segment of the spiral path.
- Evolutionary releases developed in iterative passes.
- Risk considered at each revolution around the spiral.



Spiral Model

Benefits:

Realistic approach for large-scale system development.

Provides a systematic yet iterative framework.

Addresses technical risks at all stages, reducing potential problems.

Challenges:

Requires convincing customers of the controllability of the evolutionary approach.

Demands expertise in risk assessment for successful implementation.

4. Concurrent Models

- Allows representation of iterative and concurrent elements from various process models.
- Provides flexibility in accommodating different software engineering actions.

Schematic Representation:

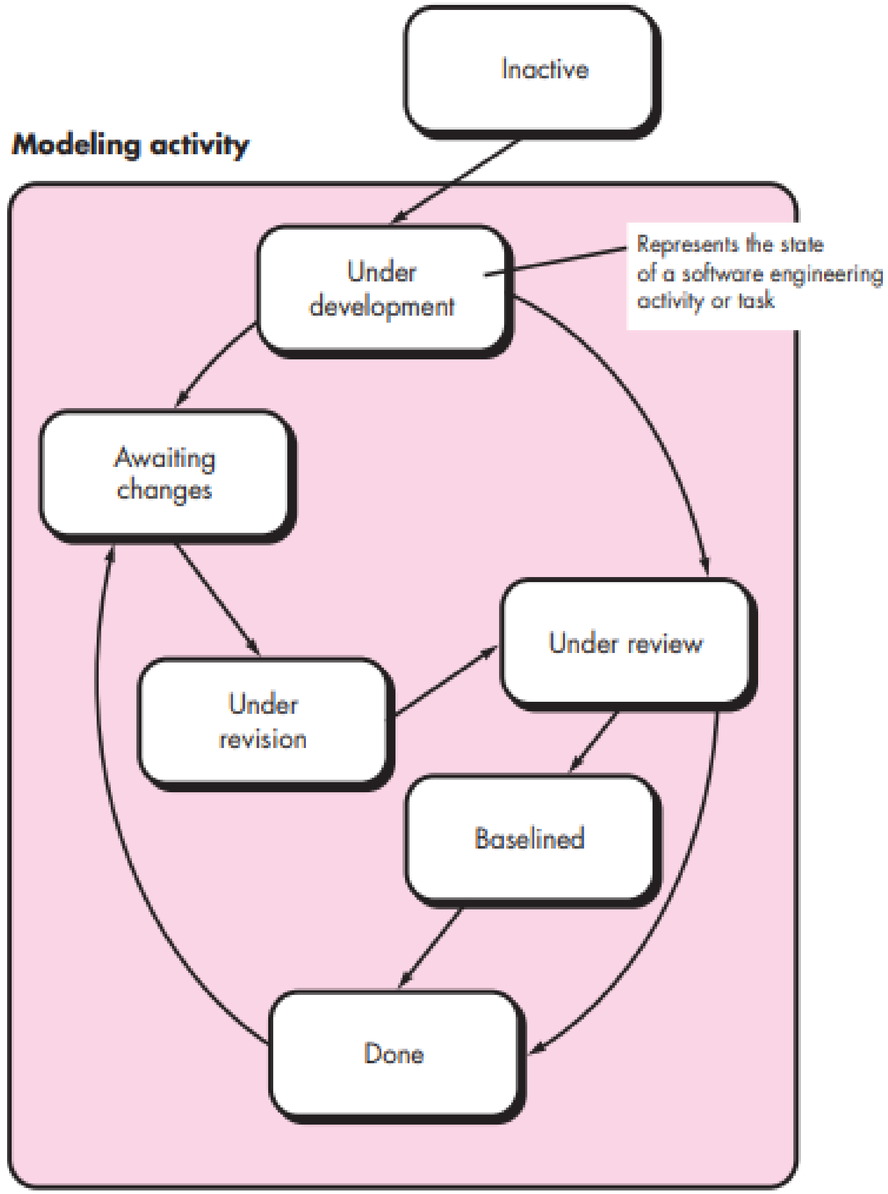
- Activities exist concurrently but in different states.
- Each activity, action, or task transitions through states triggered by events.
- Transitions between states triggered by events like changes in requirements.

Applicability:

- Relevant to all types of software development.
- Provides an accurate depiction of the project's current state.

Benefits:

- Avoids confining software engineering activities to a strict sequence.
- Defines a process network for simultaneous execution of activities.



Specialized Process Models

➢ 1. Component-Based Development

Overview:

- Utilizes commercial off-the-shelf (COTS) software components.
- Characteristics like the spiral model with an evolutionary approach.

Process Steps:

- Research and evaluate available COTS components.
- Consider component integration issues.
- Design software architecture accommodating components.
- Integrate components into the architecture.
- Conduct comprehensive testing for functionality.

Benefits:

- Leads to software reuse.
- Reduces development cycle time and project cost.

➢ 2. Formal Methods Model

Overview:

- Involves a set of activities leading to formal mathematical specification of computer software.
- Enables rigorous specification, development, and verification using mathematical notation.

Variation: Cleanroom Software Engineering

- Offers a defect-free software promise.
- Requires extensive training due to its complexity.

Concerns:

- Time-consuming and expensive development.
- Requires extensive training for developers.
- Difficulty in communication with technically unsophisticated customers.

Applicability:

Preferred in safety-critical software development (e.g., aircraft avionics, medical devices).

Specialized Process Models

➢ 3. Aspect-Oriented Software Development

Overview:

- Provides a process and methodological approach for defining, specifying, designing, and constructing aspects.
- Offers mechanisms beyond subroutines and inheritance for localizing the expression of crosscutting concerns.

Aspectual Requirements:

Define crosscutting concerns impacting the entire software architecture.

Key Elements:

- Horizontal slices through vertically-decomposed software components.
- Address cross-cutting functional and non-functional properties.

Integration:

- Integrates evolutionary and concurrent process models.
- Aspects are engineered independently but impact localized software components.

Personal Software Process (PSP)

Overview:

- Emphasizes personal measurement of work product and quality.
- Requires practitioners to control project planning and quality.

Framework Activities:

i. Planning: Isolates requirements, develops estimates, schedules tasks.
ii. High-level design: Develops external specifications and component designs.
iii. High-level design review: Applies formal verification methods.
iv. Development: Refines designs, generates code, conducts testing.
v. Postmortem: Analyses effectiveness of the process using collected metrics.

Benefits:

- Identifies errors early through rigorous assessment.
- Leads to significant improvement in productivity and software quality.

Challenges:

- Requires commitment and thorough training.
- Culturally difficult for many practitioners.
- Relatively lengthy training process.

Team Software Process (TSP)

Overview:

- Extends lessons from PSP to team-based software development.
- Aims to build self-directed project teams for high-quality software production.

Objectives:

- Build self-directed teams that plan, track, and own their processes.
- Accelerate software process improvement.
- Facilitate university teaching of industrial-grade team skills.

Framework Activities:

- Project launch, high-level design, implementation, integration and test, postmortem.
- Guides team members in their work through scripts, forms, and standards.

Benefits:

- Enables disciplined planning, design, and construction of software.
- Provides quantifiable benefits in productivity and quality.

Challenges:

- Requires full commitment and thorough training from the team.
- Rigorous approach may face resistance.
- Ensuring proper application of the approach is essential.

Process Technology

- Process technology tools play a crucial role in adapting process models for software teams.
- They help analyse current processes, organize tasks, monitor progress, and manage quality.

Modelling the Process:

- Process technology tools allow building an automated model of the process framework, task sets, and umbrella activities.
- The model, often represented as a network, facilitates analysing workflow and exploring alternative process structures.

Creating an Acceptable Process:

- After creating a model, tools aid in allocating, monitoring, and controlling software engineering activities.
- Each team member can develop checklists for tasks, work products, and quality assurance activities.

Coordination and Integration:

- Process technology tools coordinate the use of other software engineering tools required for specific tasks.
- This integration ensures seamless workflow and enhances productivity.

Product and Process

Understanding the Dichotomy:

- The software community often oscillates between focusing on product and process.
- Margaret Davis highlights the shifting focus, from structured programming languages to the Software Engineering Institute's Capability Maturity Model to agile software development.

Challenges of Pendulum Swings:

- Constant shifts in focus confuse practitioners and fail to address the core issues.
- Treating product and process as opposing forces hampers progress and understanding.

Embracing Duality:

- Analogous to scientific notions of duality, software development exhibits a fundamental duality between product and process.
- Viewing artifacts solely as products or processes limits understanding of their context, use, and worth.

Significance of Reuse:

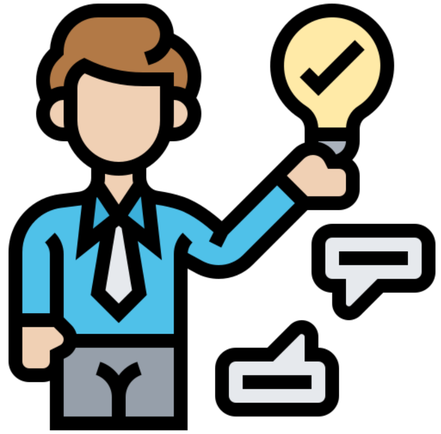
- Reuse enhances job satisfaction but necessitates acceptance of the duality of product and process.
- Failure to recognize this duality restricts opportunities for reuse and diminishes job satisfaction.

Embracing Creativity:

- Just as artists derive satisfaction from both the creative process and the product, software professionals should value both aspects.
- Recognizing the duality of product and process is crucial for keeping creative individuals engaged in software engineering.

Requirements Engineering

The broad spectrum of tasks and techniques that lead to an understanding of requirements is called Requirements Engineering.



Challenges in Requirements Engineering:

- Difficulty in eliciting requirements from customers.
- Trouble in understanding acquired information.
- Recording requirements in a disorganized manner.
- Spending too little time verifying recorded requirements.
- Allowing change to control the process.

Requirements engineering provides mechanisms for:

- Understanding customer needs.
- Analysing feasibility.
- Negotiating solutions.
- Specifying solutions clearly.
- Validating specifications.
- Managing requirements throughout the project lifecycle.

Requirements Engineering Tasks

1) Inception

- Introduction to project inception and its catalysts.
- Identification of business needs or new market opportunities as common starting points.
- Involvement of stakeholders from the business community in defining project scope and feasibility.

2) Elicitation

- What the objectives for the system or product are + what is to be accomplished + How the system or product fits into business needs
- *Problems:* Problems of scope + Problems of understanding + Problems of volatility.
- Solution: organized approach to requirements gathering.

3) Elaboration

- The information obtained from the customer during inception and elicitation is expanded and refined during elaboration.
- Focus on developing a refined requirements model.
- Use of user scenarios to identify software function, behaviour, and information.

4) Negotiation

- Negotiation to resolve conflicting requirements.
- Prioritization of requirements and discussion of conflicts.
- Iterative approach to achieve satisfaction for all stakeholders - requirements are eliminated, combined, and/or modified

5) Specification

- Written document, a set of graphical models, a formal mathematical model, a collection of usage scenarios, a prototype, or any combination of these

6) Validation

- To ensure quality and conformity to standards – requirements engineering results are assessed during validation (technical review) .

7) Requirements Management

- Activities that help the project team identify, control, and track requirements and changes to requirements at any time as the project proceeds

Initiating the Requirements Engineering Process - *Groundwork*

1. Identifying Stakeholders

   "anyone who benefits in a direct or indirect way from the system which is being developed."

   - Business operations managers, product managers, marketing people, internal and external customers, end users, consultants, product engineers, software engineers, support and maintenance engineers, and others.
   - The initial list will grow as stakeholders are contacted because every stakeholder will be asked: *"Whom else do you think I should talk to?"*

2. Recognizing Multiple Viewpoints
   - Categorize all stakeholder information (including inconsistent and conflicting requirements) – to facilitate decision makers to choose an internally consistent set of requirements for the system

3. Working toward Collaboration
   - Multiple stakeholders can have varied opinions – have a conclusive decision on the opinions
   - Analyse opinions – find commonalities and inconsistencies

4. Asking the First Questions
   - Who the solution is for ? + Benefits ? + How to decide best solution ? + Multiple Solutions ? + Envionment ? + Right Stakeholder to communicate / discuss / derive conclusion ?

Eliciting Requirements – *Requirement Gathering*

1. Collaborative Requirements Gathering
   - *Objective:* Identify the problem + propose elements of the solution + negotiate different approaches + specify a preliminary set of solution requirements in an atmosphere that is conducive to the accomplishment of the goal
2. Quality Function Deployment
   - Quality function deployment (QFD) is a quality management technique that translates the needs of the customer into technical requirements for software.
   - QFD identifies three types of requirements
     - Normal requirements - Objectives / goals that are stated for a product or system during meetings with the customer
     - Expected requirements - Not stated explicitly by the customers as it may be standard to have
     - Exciting requirements - Beyond customer expectation (Vow Factor)
3. Usage Scenarios
   - Developers and users create a set of scenarios that identify a thread of usage for the system to be constructed. The scenarios, often called use cases provide a description of how the system will be used
4. Elicitation Work Products

Eliciting Requirements

– Requirement Gathering

4. Elicitation Work Products

- Statement of Need and Feasibility
- Bounded Scope Statement
- List of Participants
- Technical Environment Description
- Requirements List
- Usage Scenarios
- Prototypes (if applicable)

Review Process:

- All participants in elicitation review each work product.
- Ensures alignment and understanding across stakeholders.

Importance:

- Provides clarity on system objectives and feasibility.
- Defines scope and stakeholders.
- Details technical aspects and constraints.
- Specifies functional requirements and domain constraints.
- Offers insights into system usage through scenarios.
- Refines requirements through prototype development.

Alignment with Elicitation:

- Each work product reflects insights gained during elicitation process.
- Ensures comprehensive understanding and documentation.

SafeHome Case Study

SafeHome

Conducting a Requirements Gathering Meeting

The scene: A meeting room. The first requirements gathering meeting is in progress.

The players: Jamie Lazar, software team member; Vinod Raman, software team member; Ed Robbins, software team member; Doug Miller, software engineering manager; three members of marketing; a product engineering representative; and a facilitator.

The conversation:

Facilitator (pointing at whiteboard): So that's the current list of objects and services for the home security function.

Marketing person: That about covers it from our point of view.

Vinod: Didn't someone mention that they wanted all *SafeHome* functionality to be accessible via the Internet? That would include the home security function, no?

Marketing person: Yes, that's right . . . we'll have to add that functionality and the appropriate objects.

Facilitator: Does that also add some constraints?

Jamie: It does, both technical and legal.

Production rep: Meaning?

Jamie: We better make sure an outsider can't hack into the system, disarm it, and rob the place or worse. Heavy liability on our part.

Doug: Very true.

Marketing: But we still need that . . . just be sure to stop an outsider from getting in.

Ed: That's easier said than done and . . .

Facilitator (interrupting): I don't want to debate this issue now. Let's note it as an action item and proceed.

(Doug, serving as the recorder for the meeting, makes an appropriate note.)

Facilitator: I have a feeling there's still more to consider here.

(The group spends the next 20 minutes refining and expanding the details of the home security function.)

Negotiating Requirements

Ideal Requirements Engineering Context:

- Inception, elicitation, and elaboration tasks determine customer requirements for subsequent activities.

In Reality:

- Negotiation often required due to various constraints.
- Stakeholders balance functionality, performance, cost, and time-to-market.

Negotiation Goals:

- Develop a project plan meeting stakeholder needs.
- Reflect real-world constraints (time, people, budget).

Boehm's Negotiation Activities:

- Identify key stakeholders.
- Determine stakeholders' "win conditions."
- Negotiate win conditions into a set of win-win conditions for all.

Importance of Negotiation:

- Ensures alignment between stakeholder needs and project constraints.
- Sets the stage for successful project execution.

Validating Requirements

Purpose of Review:

- Identify inconsistencies, omissions, and ambiguity.
- Prioritize requirements.
- Group requirements into implementation packages.

Key Questions for Review:

- Consistency with system objectives?
- Proper level of abstraction?
- Necessity vs. add-on feature?
- Boundedness and unambiguity?
- Attribution of requirements?
- Conflict resolution among requirements?
- Achievability in technical environment?
- Testability once implemented?
- Reflective of system information, function, and behaviour?
- Proper partitioning and detailing?
- Use of requirements patterns and validation?

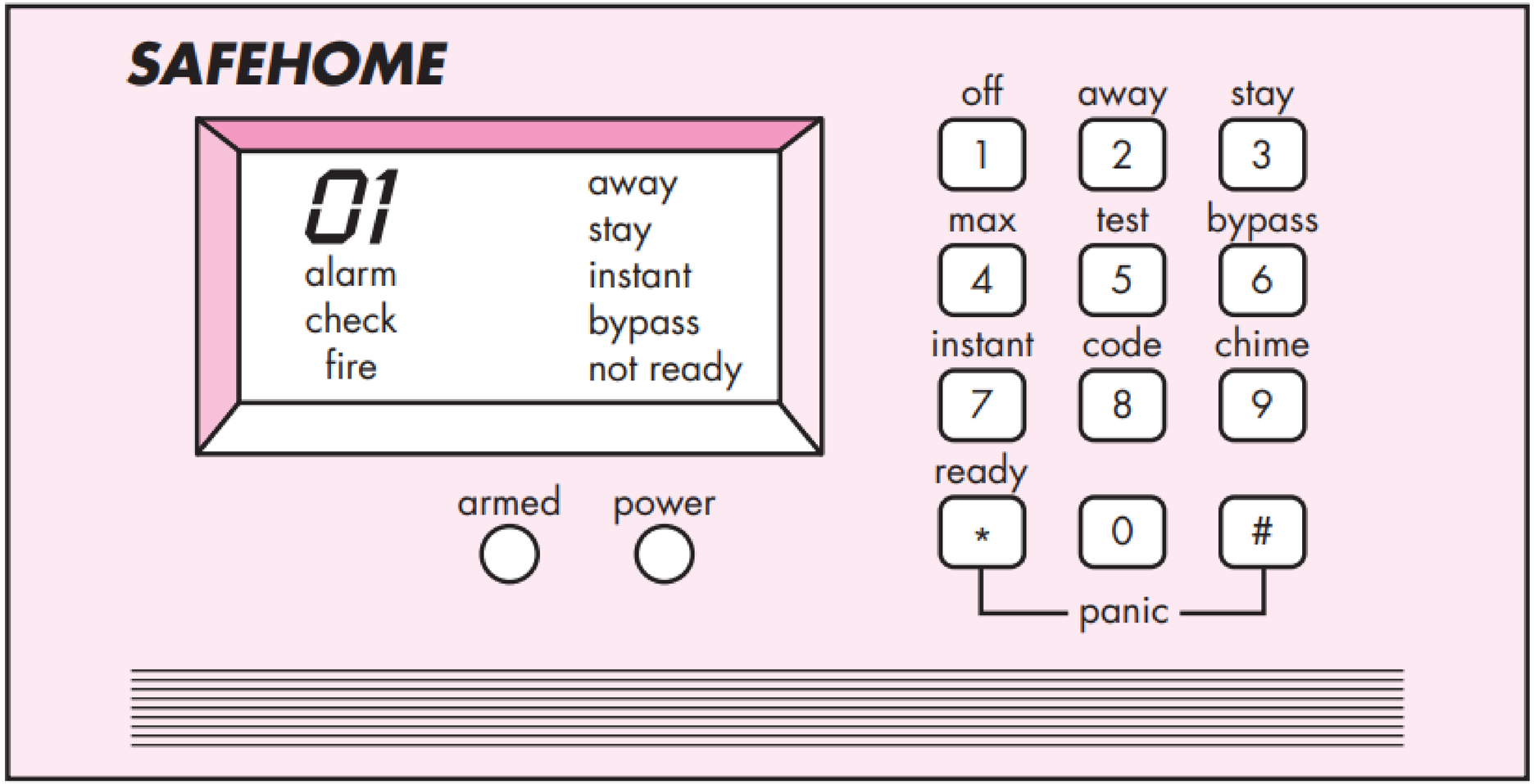
Mediation Suggestions:

- Find compromises that prioritize critical functionalities.
- Consider phased releases to accommodate resource limitations.
- Ensure alignment with overall system objectives.

Importance of Review:

- Ensures accuracy in reflecting stakeholder needs.
- Provides a solid foundation for design and implementation.

Developing Use Cases



Developing Use Cases

A contract describing the system's behaviour under various conditions.

Tells a stylized story of how an end user interacts with the system.

Forms of Use Cases:

- Narrative text, task outlines, templates, or diagrams.
- Depicts the software/system from the end user's perspective.

Key Elements of Use Case:

Actors: Involved individuals or devices interacting with the system.

Goals: Objectives actors aim to achieve through system interaction.

Preconditions: Conditions required before the use case starts.

Main Tasks: Actions performed by the actor.

Exceptions: Alternate scenarios or error conditions.

Variations: Different ways the actor can interact.

System Information: Data exchanged with the system.

Communication Channels: Methods of interaction.

Open Issues: Unresolved questions or concerns.

Questions to be answered by usecase:

- Who is the primary actor, the secondary actor(s)?
- What are the actor’s goals?
- What preconditions should exist before the story begins?
- What main tasks or functions are performed by the actor?
- What exceptions might be considered as the story is described?
- What variations in the actor’s interaction are possible?
- What system information will the actor acquire, produce, or change?
- Will the actor have to inform the system about changes in the external environment?
- What information does the actor desire from the system?
- Does the actor wish to be informed about unexpected changes?

Developing Use Cases

Use case: InitiateMonitoring

Primary actor: Homeowner.

Goal in context: To set the system to monitor sensors when the homeowner leaves the house or remains inside.

Preconditions: System has been programmed for a password and to recognize various sensors.

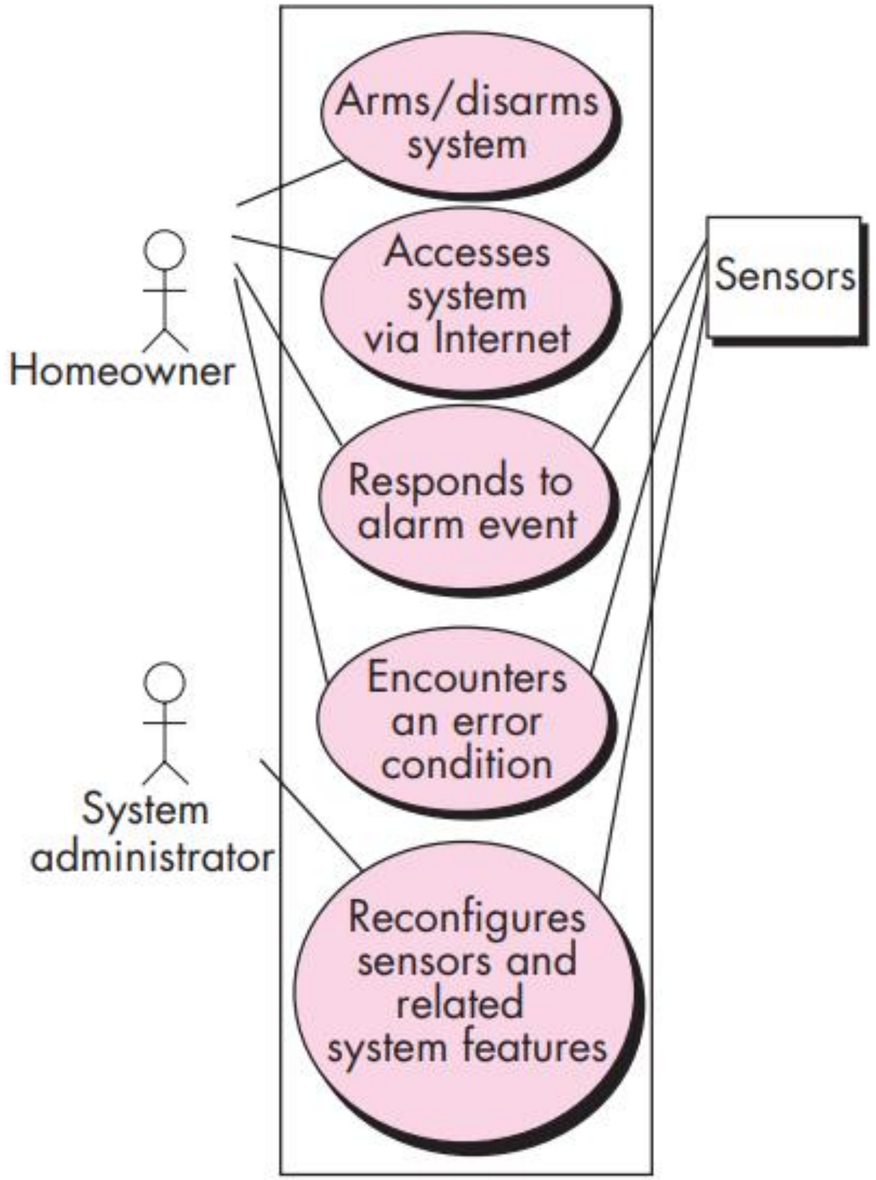
Trigger: The homeowner decides to “set” the system, i.e., to turn on the alarm functions.

Scenario:

1. Homeowner: observes control panel
2. Homeowner: enters password
3. Homeowner: selects “stay” or “away”
4. Homeowner: observes read alarm light to indicate that SafeHome has been armed

Exceptions:

1. Control panel is not ready: homeowner checks all sensors to determine which are open; closes them.
2. Password is incorrect (control panel beeps once): homeowner reenters correct password.
3. Password not recognized: monitoring and response subsystem must be contacted to reprogram password.
4. Stay is selected: control panel beeps twice and a stay light is lit; perimeter sensors are activated.
5. Away is selected: control panel beeps three times and an away light is lit; all sensors are activated.



UML use case diagram for SafeHome home security function

Developing Use Cases

Priority: Essential, must be implemented

When available: First increment

Frequency of use: Many times per day

Channel to actor: Via control panel interface

Secondary actors: Support technician, sensors

Channels to secondary actors:

Support technician: phone line

Sensors: hardwired and radio frequency interfaces

Open issues:

1. Should there be a way to activate the system without the use of a password or with an abbreviated password?
2. Should the control panel display additional text messages?
3. How much time does the homeowner have to enter the password from the time the first key is pressed?
4. Is there a way to deactivate the system before it actually activates?

Review and Elaboration:

- Each use case should be carefully reviewed for clarity and completeness.
- Ambiguities indicate areas for further refinement in the requirements.

Building the Requirements Model

Intent of Analysis Model:

- Description of required informational, functional, and behavioural domains.
- Changes dynamically as understanding of system evolves.
- Snapshot of requirements at any given time; expected to change.

Elements of Requirements Model:

1. *Scenario-based Elements:*
    - Described from user's viewpoint using scenarios.
    - Basic use cases evolve into more elaborate templates.
    - Input for creation of other modelling elements.
2. *Class-based Elements:*
    - Objects categorized into classes based on usage scenarios.
    - UML class diagrams depict attributes, operations, and relationships.
3. *Behavioural Elements:*
    - Represent system behaviour through state diagrams.
    - Depict states, events, and actions.
    - Individual class behaviour can also be modelled.
4. *Flow-oriented Elements:*
    - Model information flow through the system.
    - Transform input to output via functions.
    - Can create flow model for any computer-based system.

Analysis Patterns:

Reoccurring problems across projects in specific application domains.

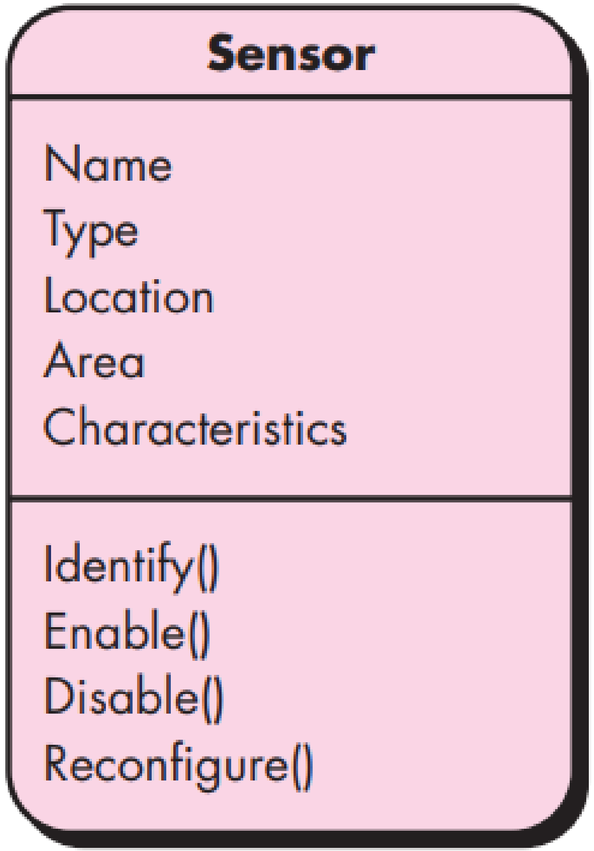
Provide reusable solutions within the application domain.

Speed up development of abstract analysis models.

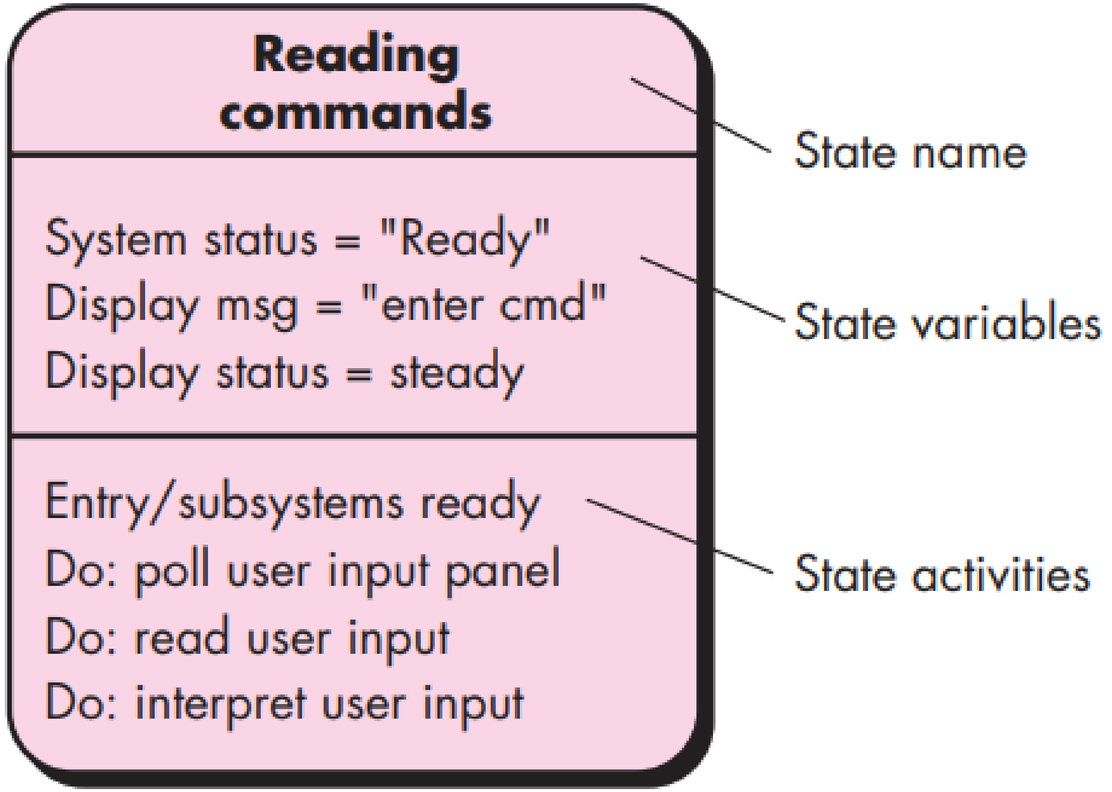
Facilitate transformation of analysis model into design model.

Integrated into analysis model by reference and stored in a repository.

Building the Requirements Model

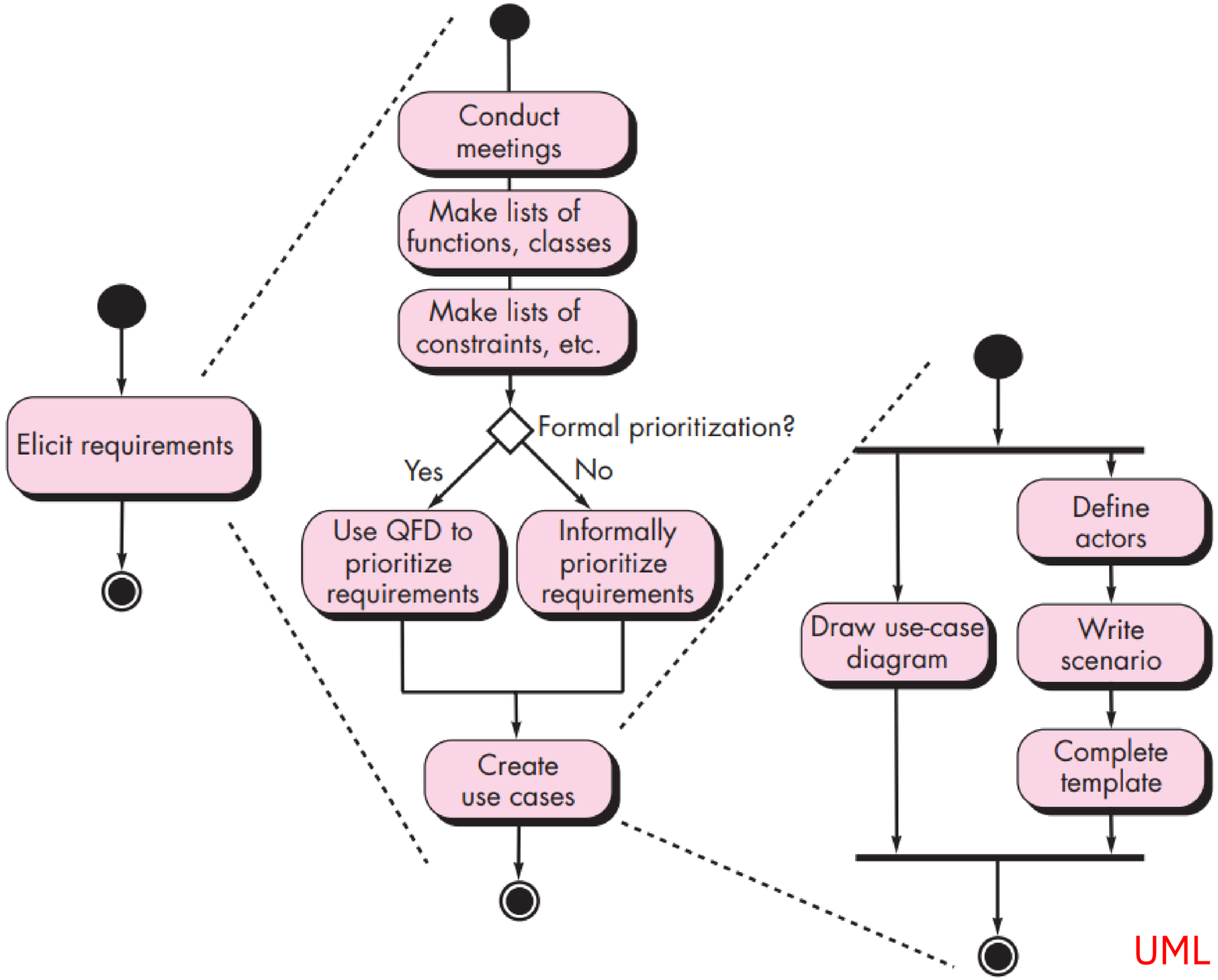


Class diagram for sensor



UML state diagram notation

Building the Requirements Model



UML activity diagrams for eliciting requirements

Software Requirement Document

- SRS Document

1. Introduction
    - 1.1 Purpose
    - 1.2 Scope
    - 1.3 Definitions, Acronyms, and Abbreviations
    - 1.4 References
    - 1.5 Overview
2. Overall Description
    - 2.1 Product Perspective
    - 2.2 Product Functions
    - 2.3 User Characteristics
    - 2.4 General Constraints
    - 2.5 Assumptions and Dependencies
3. Specific Requirements

General structure of an SRS

Detailed Requirements Section

3. Detailed Requirements
    - 3.1 External Interface Requirements
        - 3.1.1 User Interfaces
        - 3.1.2 Hardware Interfaces
        - 3.1.3 Software Interfaces
        - 3.1.4 Communication Interfaces
    - 3.2. Functional Requirements
        - 3.2.1 Mode 1
        - 3.2.1.1 Functional Requirement 1.1
        - :
        - 3.2.1.n Functional Requirement 1.n
        - :
        - 3.2.m Mode m
        - 3.2.m.1 Functional Requirement m.1
        - :
        - 3.2.m.n Functional Requirement m.n
    - 3.3 Performance Requirements
    - 3.4 Design Constraints
    - 3.5 Attributes
    - 3.6 Other Requirements


THANK
YOU