Process Framework

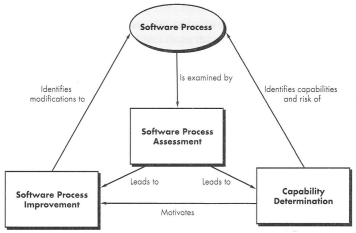
Communication
- Planning
- modeling

construction
- deployment

Process Assessment

the existence of a software process is no guarantee that software will be delivered on time

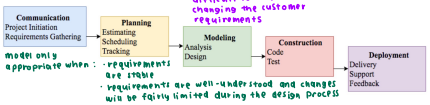
- depends on: meet the customer's needs
  exhibit the technical characteristics that will lead to long-term quality



Software Process Model

waterfall model

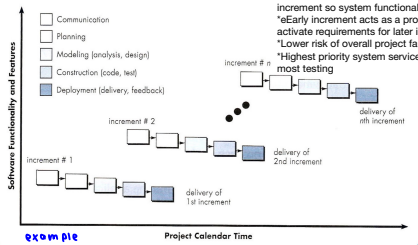
Problem : - inflexible
- difficult to respond of changing the customer requirements



model only appropriate when :
- requirements are stable
- requirements are well-understood and changes will be fairly limited during the design process

Incremental Process model

advantages :
*Custom values can be delivered with each increment so system functionality is available earlier
*eEarly increment acts as a prototype to help activate requirements for later increment s
*Lower risk of overall project failure
*Highest priority system services tend to receive the most testing



example

1st increment: Menu display, Checkout process, Addresses

2nd increment: Login, User's information, Pizza add-on, Estimated time arrive, Promotion, Feedback

3rd increment: Page layout, Grammar and spelling, Tracking

Rapid Application Development (RAD) model

An incremental software process model that emphasizes a short development cycle

Evolutionary Process model

Iterative model and they are characterized in a manner that enables software engineers to develop increasingly more complete versions of software

Objective: *To work with customers
*To evolve a final system from an initial outline specification
-Start with well-understood requirements and add new features as proposed by customers

Software Process Model

1. Waterfall Model

- Linear and sequential
- Each phase must be completed before moving to the next.
- **Best for**: Stable, well-defined requirements.
- **Problem**: Inflexible to changes.

2. Incremental Model

- Delivers software in small, functional parts (increments).
- Early increments provide basic functionality, later ones add more.
- **Advantages**:
  - Early delivery of value.
  - Feedback-based refinement.
  - Reduced risk.

3. Rapid Application Development (RAD)

- An accelerated version of the waterfall model.
- Emphasizes component-based, quick development.
- **Suitable for**: Time-critical projects with modular systems.

4. Evolutionary Process Models

- Software evolves over iterations.
- Useful when requirements are not fully understood from the start.
  - **Prototyping**: Quick-and-dirty models to gather user feedback.
  - **Spiral Model**: Combines prototyping and waterfall with risk analysis at each iteration.

5. Component-Based Software Engineering (CBSE)

- Develops systems by integrating pre-existing components.
- **Stages**:
  - Component analysis
  - Requirement adaptation
  - System design with reuse
  - Development and integration

🔷 Agile Models

✅ Agile Principles

- Customer satisfaction via early and continuous delivery
- Embrace change
- Frequent delivery
- Close, daily cooperation between business and developers
- Simplicity and self-organizing teams

🔸 Extreme Programming (XP)

- Driven by **User Stories**
- Practices: Pair programming, simple design, continuous integration
- Testing: Automated unit and acceptance tests

🔸 Scrum

- Iterative development using time-boxed sprints
- Emphasizes team roles (Scrum Master, Product Owner), ceremonies (Daily Scrum), and artifacts (Product Backlog)

Analysis Model

✅ Use-Case Diagrams

- Describe **functions** of a system from the user's point of view
- Elements:
  - **Actor**: User/system interacting with the system
  - **Use Case**: What the actor does
  - **Relationships**:
    - **Association**
    - **Include** (common step reused)
    - **Extend** (optional/alternative behavior)
    - **Generalization**

✅ Use-Case Description Includes:

- Overview info
- Flow of events (normal/subflow/alternate)
- Optional characteristics

🧩 4. Activity Diagrams

- Describe **workflow** or sequence of actions
- Key elements:
  - **Action/Activity**
  - **Decision/Merge Node**
  - **Fork/Join Node**
  - **Initial/Final Node**
  - **Swimlanes** (responsibilities by actor/group)

🔁 2. State Machine Diagrams

- **Purpose**: Show how a **single object changes states** in response to events over its lifecycle.

✅ Key Elements

Element	Description
Initial State	Small filled circle (start point)
State	Rounded rectangle with state name
Event	Triggers transition to another state
Transition	Arrow labeled by the event name
Final State	Circle with smaller black circle inside (end point)

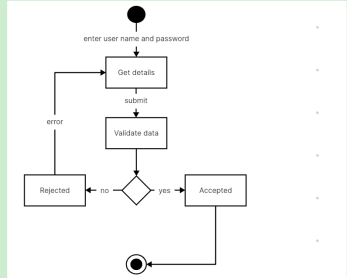
📍 1. Sequence Diagrams

- **Purpose**: Model dynamic behavior for a specific use case by showing how objects interact **over time**.

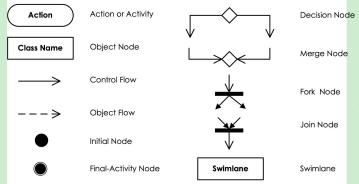
✅ Key Elements

Element	Description
Actor	External user or system interacting with the system
Object	Class instances that send/receive messages
Lifeline	Vertical dashed line showing object lifetime
Execution Occurrence	Long narrow rectangle showing when an object performs actions
Message	Communication between objects (solid = call, dashed = return)

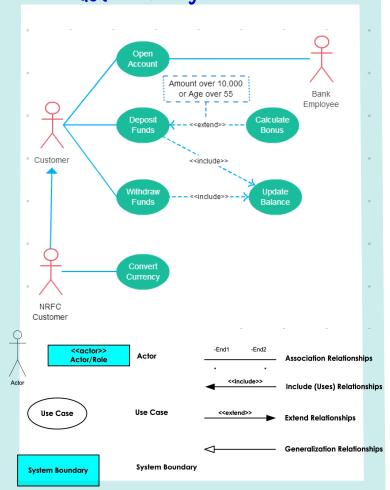
Activity diagram



Elements of an activity diagram



use case diagram



UX

💡 What is UX?

User Experience (UX) = How a person **feels** when using a product
→ Focus: satisfaction, usability, accessibility, and emotional connection

👤 User-Centric Design (UCD)

Focus: User is the center of the design process
Phases:

- **Research**: Discover (pain points) → Define (needs)
- **Design**: Develop (prototype) → Deliver (final version)

🧭 User Flow

Shows steps a user takes to complete a task **Includes**:

- **Users**
- **User goals/needs**
- **Steps (screens/actions/decisions)**

🔧 UX Foundations (7 Principles)

Principle	Meaning
Useful	Solves the user's problem
Usable	Easy and pleasant to use
Desirable	Attractive and emotionally engaging
Learnable	Easy to learn by doing
Memorable	Easy to return to and reuse
Accessible	Usable by people with disabilities
Credible	Trusted and believable information

🔍 UX Research Methods

Phase	Methods
Discover	Interviews, field study, diary study
Explore	Persona, journey mapping, prototype testing
Test	Usability testing, accessibility evaluation
Listen	Surveys, analytics, bug review

🎯 5 Elements of UX (Top → Bottom)

1. **Strategy** – User needs + business goals
2. **Scope** – Features + content
3. **Structure** – Info architecture + interactions
4. **Skeleton** – Wireframes, UI layout
5. **Surface** – Visual design, colors, fonts

🗺️ User Journey

- **Visual map** of user's experience across all touchpoints
- Includes: **Persona**, **Goals**, **Emotions**, **Steps**, **Suggestions**

💗 Empathy Mapping

Helps understand user thoughts & feelings
Quadrants:

- **Says** – Verbal feedback
- **Thinks** – Internal thoughts
- **Does** – Observable actions
- **Feels** – Emotional state

Software requirment

🔷 What is a Requirement?

- A **requirement** is a high-level statement of a service or system constraint

🔷 Types of Requirements

1. User Requirements

- Written in natural language with diagrams for non-technical users.
- Describe functional and non-functional expectations.

2. System Requirements

- Detailed description of system functions, services, and constraints.
- Used by developers and become part of the system contract.

🔷 Common Issues with Requirements

- **Imprecision** – ambiguous language leads to misinterpretation.
- **Incompleteness & Inconsistency** – missing or conflicting details.
- **Requirements Interaction** – conflicting non-functional goals (e.g. performance vs. power consumption).

🔷 Functional vs Non-Functional Requirements

✅ Functional Requirements

- Define **what** the system should do.
- Examples: search functionality, data storage, report generation.

✅ Non-Functional Requirements

- Define **how** the system performs (constraints).
- Types:
  - **Product Requirements** – e.g. reliability, speed
  - **Organizational Requirements** – e.g. development standards
  - **External Requirements** – e.g. legal, interoperability

🔷 Writing Good Requirements (IEEE Guidelines)

- Use "**shall**" for mandatory and "**should**" for optional.
- Avoid jargon, be precise and verifiable.
- Highlight keywords and use consistent language.

🔷 Requirement Specification Approaches

1. **Natural Language (NL)** – Easy but prone to ambiguity.
2. **Structured Language** – Uses templates for clarity and consistency.
3. **Form-Based Specification** – Inputs, outputs, pre/post-conditions, side effects.
4. **Tabular Specifications** – Tabular format to describe behavior.
5. **Graphical Models** – e.g. sequence diagrams for user interactions.

Viewpoint Type	Description	Example (ATM System)
Interactor Viewpoints	Entities (people or systems) that **interact directly** with the system.	- Customer using the ATM - Bank account database
Indirect Viewpoints	Stakeholders who **don't use the system directly**, but **influence requirements**.	- Bank managers - Security officers
Domain Viewpoints	Define **rules, standards, and constraints** of the application domain.	- Inter-bank communication standards - Financial regulations

🔄 Requirements Engineering Process

1. **Elicitation** – Discovering what is needed
2. **Analysis** – Clarifying and prioritizing
3. **Specification** – Documenting
4. **Validation** – Checking correctness
5. **Management** – Handling changes

Software Process Model

1. Waterfall Model

- Linear and sequential — oldest model
- Each phase must be completed before moving to the next.
- **Best for**: Stable, well-defined requirements.
- **Problem**: Inflexible to changes.

2. Incremental Model

- Delivers software in small, functional parts (increments).
- Early increments provide basic functionality, later ones add more.
- **Advantages**:
  - Early delivery of value.
  - Feedback-based refinement.
  - Reduced risk.

3. Rapid Application Development (RAD)

- An accelerated version of the waterfall model.
- Emphasizes component-based, quick development.
- **Suitable for**: Time-critical projects with modular systems.

4. Evolutionary Process Models

- Software evolves over iterations.
- Useful when requirements are not fully understood from the start.
  - **Prototyping**: Quick-and-dirty models to gather user feedback.
  - **Spiral Model**: Combines prototyping and waterfall with risk analysis at each iteration.

5. Component-Based Software Engineering (CBSE)

- Develops systems by integrating pre-existing components.
- **Stages**:
  - Component analysis
  - Requirement adaptation
  - System design with reuse
  - Development and integration

Agile Models

Agile Principles

- Customer satisfaction via early and continuous delivery
- Embrace change
- Frequent delivery
- Close, daily cooperation between business and developers
- Simplicity and self-organizing teams

Extreme Programming (XP)

- Driven by **User Stories**
- Practices: Pair programming, simple design, continuous integration
- Testing: Automated unit and acceptance tests

Scrum

- Iterative development using time-boxed sprints
- Emphasizes team roles (Scrum Master, Product Owner), ceremonies (Daily Scrum), and artifacts (Product Backlog)

Software requirment

What is a Requirement?

- A **requirement** is a high-level statement of a service or system constraint

Types of Requirements

1. User Requirements

- Written in natural language with diagrams for non-technical users.
- Describe functional and non-functional expectations.

2. System Requirements

- Detailed description of system functions, services, and constraints.
- Used by developers and become part of the system contract.

Common Issues with Requirements

- **Imprecision** – ambiguous language leads to misinterpretation.
- **Incompleteness & Inconsistency** – missing or conflicting details.
- **Requirements Interaction** – conflicting non-functional goals (e.g. performance vs. power consumption).

Functional vs Non-Functional Requirements

Functional Requirements

- Define **what** the system should do.
- Examples: search functionality, data storage, report generation.

Non-Functional Requirements

- Define **how** the system performs (constraints).
- Types:
  - **Product Requirements** – e.g. reliability, speed
  - **Organizational Requirements** – e.g. development standards
  - **External Requirements** – e.g. legal, interoperability

Writing Good Requirements (IEEE Guidelines)

- Use "**shall**" for mandatory and "**should**" for optional.
- Avoid jargon, be precise and verifiable.
- Highlight keywords and use consistent language.

Requirement Specification Approaches

1. **Natural Language (NL)** – Easy but prone to ambiguity.
2. **Structured Language** – Uses templates for clarity and consistency.
3. **Form-Based Specification** – Inputs, outputs, pre/post-conditions, side effects.
4. **Tabular Specifications** – Tabular format to describe behavior.
5. **Graphical Models** – e.g. sequence diagrams for user interactions.

Requirements Engineering Process

1. **Elicitation** – Discovering what is needed
2. **Analysis** – Clarifying and prioritizing
3. **Specification** – Documenting
4. **Validation** – Checking correctness
5. **Management** – Handling changes

If you have to develop software for a movie ticket kiosk system, what is the most suitable software process model?

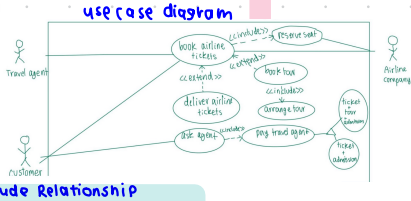
- Using a **waterfall** as a kiosk is a simple and very clear requirement and doesn't require much updating.
- **Spiral or Agile** model suits if its a movie ticket kiosk system, too complex requirements, membership, coupon, Need frequent updates.

If the question is "Never been developed before" use a prototyping model.

Viewpoints

Viewpoint Type	Description	Example (ATM System)
Interactor Viewpoints	Entities (people or systems) that **interact directly** with the system.	- Customer using the ATM - Bank account database
Indirect Viewpoints	Stakeholders who **don't use the system directly**, but **influence requirements**.	- Bank managers - Security officers
Domain Viewpoints	Define **rules, standards, and constraints** of the application domain.	- Inter-bank communicatio standards - Financial regulations

use case diagram

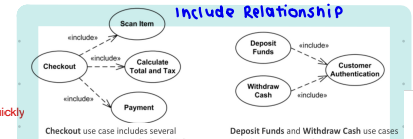


A Legacy system is an old method, technology, or application that is still in use because it continues to function effectively and users still rely on it.
Why keeping legacy system?
High availability requirement: system must always be available and cannot be taken offline (e.g. for upgrade or replacement)
High cost to replace: rebuilding a system with similar reliability is expensive
Common examples: bank customer account systems, airline reservation systems, air traffic control, military defense systems.

What is the software process model for the software development situation below:

- In the ACM-computer programming competition, each group will have to solve 5 programming problems within 3 hours.
  - Answer: Code and Fix: not enough time for formal process and focused purely on writing and testing code quickly
- If you have to do the software project "Online meeting schedule planning" with the features that multiusers can collaborate to schedule several courses.
  - Answer: Spiral model: support iterative development and handles evolving requirements well and include risk management and prototypes.
- Mobile application for the social network communication i.e. Line
  - Answer: Incremental model: features added in increments(e.g. chat, call, video, sticker), and enables fasters releases and continuous updates.
- Basic unit converter application
  - Answer: Waterfall model: simple and stable requirements, and best for small, predictable projects

Include Relationship



Checkout use case includes several use cases - Scan Item, Calculate Total and Tax, and Payment

Deposit Funds and Withdraw Cash use cases include Customer Authentication use case.

Extend diagram

Extending use case is optional, supplementary

Registration use case is complete and meaningful on its own. It could be extended with optional Get Help On Registration use case.

If you have to develop software for a movie ticket kiosk system, what is the most suitable software process model?

- Using a **waterfall** as a kiosk is a simple and very clear requirement and doesn't require much updating.
- **Spiral or Agile** model suits if its a movie ticket kiosk system, too complex requirements, membership, coupon, Need frequent updates.

If the question is "Never been developed before" use a prototyping model.

Functional vs Non-Functional Properties

- **Functional**: What the system should do
  → e.g., login, data processing, report generation
- **Non-Functional**: How the system performs
  → e.g., performance, security, usability

What are CASE Tools?

Computer-**A**ided **S**oftware **E**ngineering tools that support software development activities (e.g., modeling, testing, coding).

Attributes of Good Software

1. **Maintainability** – Easy to fix or update
2. **Dependability** – Reliable and secure
3. **Efficiency** – Uses resources wisely
4. **Acceptability** – Meets user needs and is easy to adopt

Four Activities of Software Process

1. **Specification** – Define what the software should do
2. **Development** – Build the software
3. **Validation** – Check if it meets the requirements
4. **Evolution** – Modify as user needs change

1) What kind of project is suitable for the Prototyping Model?

Answer:
Projects where **requirements are unclear or evolving**, or when the **user is unsure of what they want**.

Suitable for exploratory or user interface–focused systems where feedback is critical early on.

2) Which model cannot be used for complex or large-size projects?

Answer:
Rapid Application Development (RAD) Model

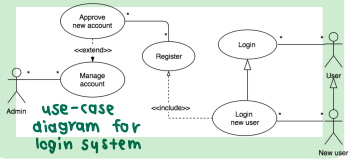
Because it relies on quick development with small teams and is **not suitable for complex, large-scale systems** that need deep integration and long-term planning.

Types of CASE Tools

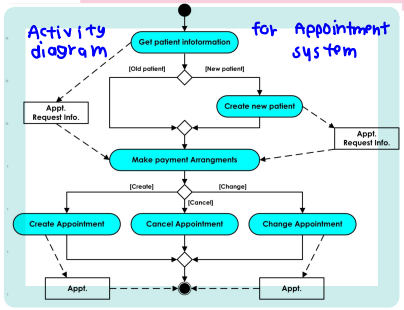
Computer-Aided Software Engineering tool

Type	Function
Upper CASE	Support **early SDLC stages** (planning, analysis, design)
Lower CASE	Support **later stages** (implementation, testing, maintenance)
Integrated CASE (I-CASE)	Support the **entire SDLC**

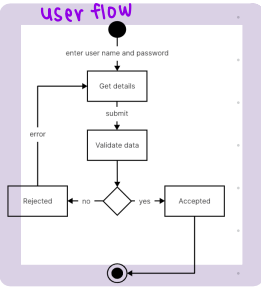
use-case diagram for login system



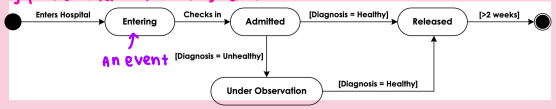
Activity diagram for Appointment system



User flow



state machine diagram



UX

What is UX?

User Experience (UX) = How a person **feels** when using a product
→ Focus: satisfaction, usability, accessibility, and emotional connection

UX Foundations (7 Principles)

Principle	Meaning
Useful	Solves the user's problem
Usable	Easy and pleasant to use
Desirable	Attractive and emotionally engaging
Learnable	Easy to learn by doing
Memorable	Easy to return to and reuse
Accessible	Usable by people with disabilities
Credible	Trusted and believable information

User-Centric Design (UCD)

Focus: User is the center of the design process
Phases:

- **Research**: Discover (pain points) → Define (needs)
- **Design**: Develop (prototype) → Deliver (final version)

User Flow

Shows steps a user takes to complete a task **Includes**:

- **Users**
- **User goals/needs**
- **Steps (screens/actions/decisions)**

UX Research Methods

Phase	Methods
Discover	Interviews, field study, diary study
Explore	Persona, journey mapping, prototype testing
Test	Usability testing, accessibility evaluation
Listen	Surveys, analytics, bug review

5 Elements of UX (Top → Bottom)

1. **Strategy** – User needs + business goals
2. **Scope** – Features + content
3. **Structure** – Info architecture + interactions
4. **Skeleton** – Wireframes, UI layout
5. **Surface** – Visual design, colors, fonts

Empathy Mapping

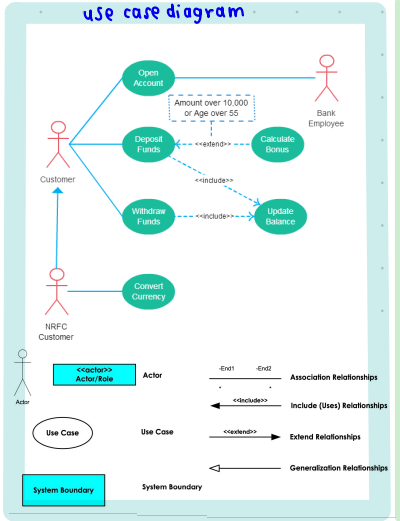
Helps understand user thoughts & feelings
Quadrants:

- **Says** – Verbal feedback
- **Thinks** – Internal thoughts
- **Does** – Observable actions
- **Feels** – Emotional state

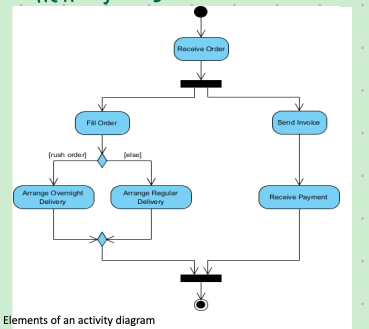
User Journey

- **Visual map** of user's experience across all touchpoints
- Includes: **Persona, Goals, Emotions, Steps, Suggestions**

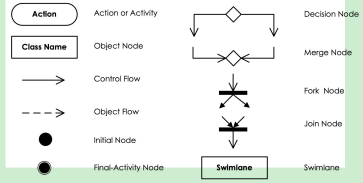
use case diagram



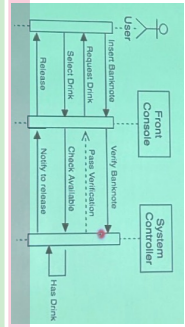
Activity diagram



Elements of an activity diagram



sequence diagram



Draw the sequence diagram according to the description below.
Description:
1) The customer inserts 20 Baht banknote into the slot in the front of the machine and makes a selection of their preferred drink.
2) In the system, banknote travels to the computing part, which can verify the banknote.
3) The system will check availability of selected drink in the stock, and the system orders the dispenser to release the selected drink to the front of the machine.

- sequence diagram
- use case diagram ✓
- activity diagram ✓
- state machine diagram ✓
- user journey