

Software Engineering SE

SE is layered technology because it accomplishes a software development in three successive layers or steps which are processes, methods and tools,



Cont.

Process

In the layered technology, the fundamental layer is the process layer that sets a groundwork for software engineering. Software engineering process layer facilitates reasonable and timely development of computer software. Software process has been substantiated as a framework for effective delivery of software engineering technology. The process framework lays the foundation for software project management and provides technical methods and tools to the software engineer for yielding products such as data, reports, documents, and models. Moreover, the process framework sets the milestones and guidelines to cope with project deadlines, quality standards, and sudden changes, such as changes in user requirement.

Methods

The layer just above the process layer is the layer that defines the set of methods for software development. The regime of method layer varies from requirement analysis, design modeling, code development, software testing, software product implementation, and software maintenance. Software engineering methods follow the predefined fundamental rules of product design and modeling.

Tools

The topmost layer in the layered technology defines a set of tools for software development. The tools may be automated or semi-automated for speeding up the development cycle. Computer-Aided Software Engineering (CASE) is an integration of a number of tools including software, hardware, and database.

Software Process



it a group of activities, actions, and tasks defined as follows:

- ❑ An *activity* is a general objective that is defined irrespective of the application domain, project size, and effort complexity
- ❑ An *action* includes a set of tasks that deliver a major work product
- ❑ A *task* is a trivial but thoroughly stated objective

A standard *process framework* contains the following activities:

- ❑ **Communication:** Refers to the exchange of views between the customer and software engineer. The customer states the requirement that works as a basis for software development.
- ❑ **Planning:** Refers to the development of software project plan that broadly defines the scheduled tasks, projected risks, and required resources.
- ❑ **Modeling:** Refers to designing prototypes or rough drafts that a software engineer creates to better understand the user requirements.
- ❑ **Construction:** Refers to code generation and testing to overcome the errors in the code.
- ❑ **Deployment:** Refers to configuring the software at the user end, where the software is thoroughly evaluated by the user and feedback is provided to the software engineer.

Graphical Representation Of Software Engineering Concepts

Project = A software system we have to develop (Goal)



Activities (Phases) plus Actions



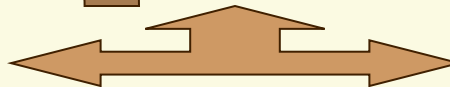
Produces

Tasks

Consumes

Work Product

Resources



System
Model
Document



Participant
Time
Equipment

Software Engineering Concepts (Cont.)



System: refers to the underlying reality

- E.g. a ticket distributor for an underground train



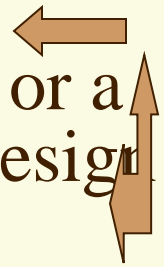
Model: refers to any abstraction of the reality, or a pattern of something to be made, or a type or design of product (such as a car)

- E.g. schematics of its electrical wiring
- Schedule, budget



Work product: an artifact that is produced during the development

- Internal work product
 - (for project's internal consumption)
- Deliverable
 - (for client)





Why do we model?

- Models are representations that **can aid in defining, analyzing, and communicating a set of concepts.**
- System models are specifically developed to support analysis, specification, design, verification, and validation of a system, as well as to communicate certain information
- An example of a model is a woman who wears a designer's clothes to show them to potential buyers at a fashion show.

Software Engineering Concepts (Cont.)

 Recourses: assets that are used to accomplish work

 Task: represent an atomic unit of work that can be managed

 Activities: a set of tasks that is performed toward a specific purpose

— E.g. Delivery (its purpose is to install the system)

Management (its purpose is to monitor and control the project such that it meets its goal [budget, deadline, quality])

Notations, Methods and Methodologies



Method: is a repeatable technique for solving a specific problem

- Sorting algorithm is a method for sorting elements



Methodology: is a collection of methods for solving a class of problem





- A seafood cookbook is a methodology for preparing seafood



Notation: is a graphical or textual set of rules **for representing a model**

- E.g. Roman alphabet is a notation for representing words
- UML (Unified Modeling Language) is an object-oriented notation for representing models

Reasons to use methodology

-  Helps produce better quality product
-  Better documented software
-  More maintainable and consistent software
-  More acceptable to user

Choosing a methodology:

Issues and Costs



Issues

- **Type of project** (large, small, etc.)
- **Application domains** (real-time, interactive distributed, security)



Costs

- **Staff training** (techniques, structure)
- **Documentation**
- **Software licenses**

Software Development (SD) Levels of Abstraction

Increasing level of abstraction in SD

- Task
- Technique (e.g. UML Use case diagram)
- Method
- Methodology

Software Engineering Blueprints

- Specifying software problems and solutions is like cartoon strip writing
- Unfortunately, most of us are not artists, so we will use something less exciting:
UML symbols
- 📄 UML is a graphical language used in object-oriented development that includes several types of system models that provide different views of a system

Remark

System to be developed consists of:


Actors

- Agents external to the system that interact with it

Concepts/ Objects

- Agents working inside the system to make it function

Use Cases: Scenarios for using the system

-  It identifies the actors involved in an interaction and names the type of interaction.


System Modeling

- ❏ System modeling is the process of developing abstract models of a system, with each model offering a distinctive viewpoint
- ❏ Systems are represented in system modeling using the Unified Modeling Language (UML), a sort of graphical notation


UML (Unified Modeling Language)


- For teams to work effectively they need a language to communicate (UML).
- In previous courses we learned languages, such as Java or C++, and how to turn ideas into code. But these ideas are independent of the language.
- With UML we will see a way to describe code independently of language, and more importantly, we learn to think in one higher level of abstraction.
- UML can be an invaluable communication and documentation tool..أداة اتصال وتوثيق لا تقدر بثمن

UML Types

 There are several diagram types in the UML, which are divided into two categories:

- **structure and behaviour diagrams.**

 As opposed to behaviour diagrams, which depict the dynamic behaviour of the objects in a system and may be thought of as a sequence of changes to the system over time, structure diagrams such as class and package diagrams represent the static structure of the system.

 UML Use case, state, activity, and sequence diagrams are common types of behaviour diagrams

UML Types (Remark)



You may develop different models to represent the system from different perspectives


1. An external perspective, where you model the context or environment of the system.
2. An interaction perspective where you model the interactions between a system and its environment or between the components of a system.
3. A structural perspective, where you model the organization of a system or the structure of the data that is processed by the system.
4. A behavioral perspective, where you model the dynamic behavior of the system and how it responds to events.

UML Types


1. Activity diagrams, which show the activities involved in a process or in data processing.
2. Use case diagrams, which show the interactions between a system and its environment.
3. Sequence diagrams, which show interactions between actors and the system and between system components.
4. Class diagrams, which show the object classes in the system and the associations between these classes.
5. State diagrams, which show how the system reacts to internal and external events.


Use case diagrams and sequence diagrams are used to describe the interactions between user the system being designed and users/other systems. Use cases describe interactions between a system and external actors; sequence diagrams add more information to these by showing interactions between system objects

UML Types (Cont.)

 A **use-case diagram** describes a system's functional requirements in terms of use cases. It is a simulation of the system's intended functionality (use cases) and its environment (actors), illustrating how a system interacts with its environment

 Interactions between actors, the system, and system components are depicted in the **sequence diagram**.

 **Activity diagrams** are visual representations of processes that include sequential activities and actions and include choice, iteration, and concurrency.

 The system's response to both internal and external events is depicted in **state diagrams**

Use case Diagram



Systems (what you are going to develop, e.g. Website, application, games, etc.).

- Is represented by a rectangle and the name of the system at the top



Actors (are external objects and are going to be someone or something that uses a system to achieve a goal)

- E.g. person, organization, external device, etc. (represented with a stick figure)
- Two types: A **primary actor** (located on the left side of the system) initiates the use of the system while a **secondary actor** (located on the right side of the system) is more reactionary



Use cases (represented by oval shape and it represents an action that accomplishes some sort of task within a system)

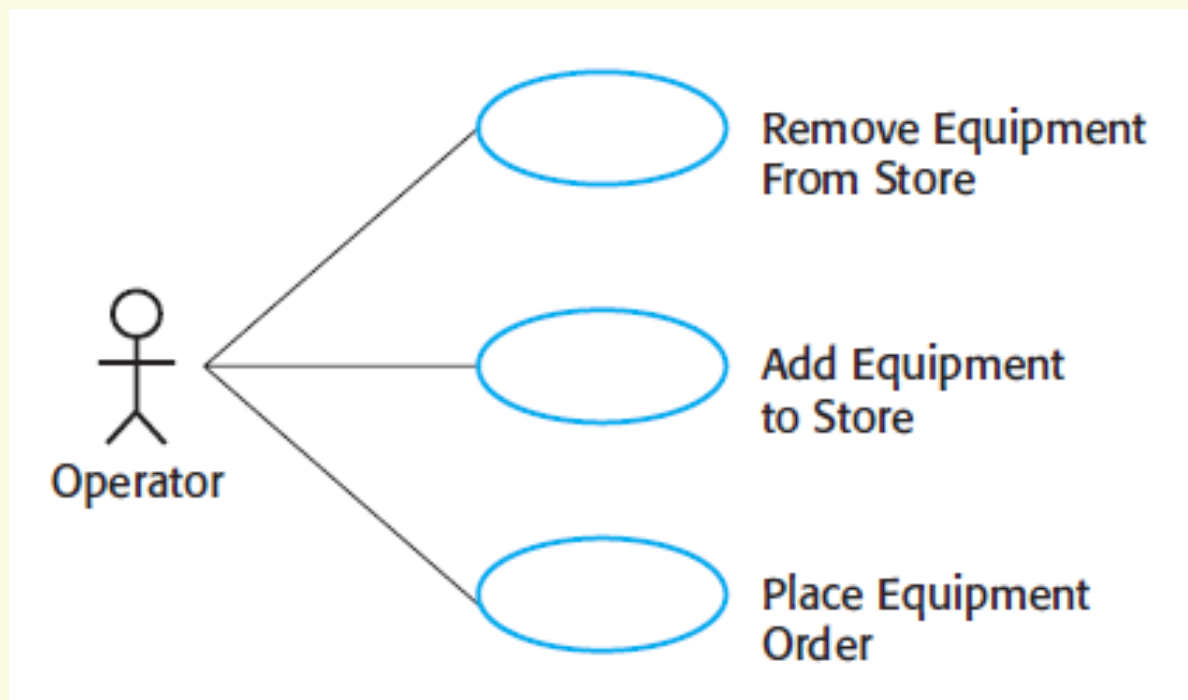
- **Put your use cases in a logical order as possible**



Relationships (each actor has to interact with at least one of the use cases within a system and we use solid line a symbol for representation)

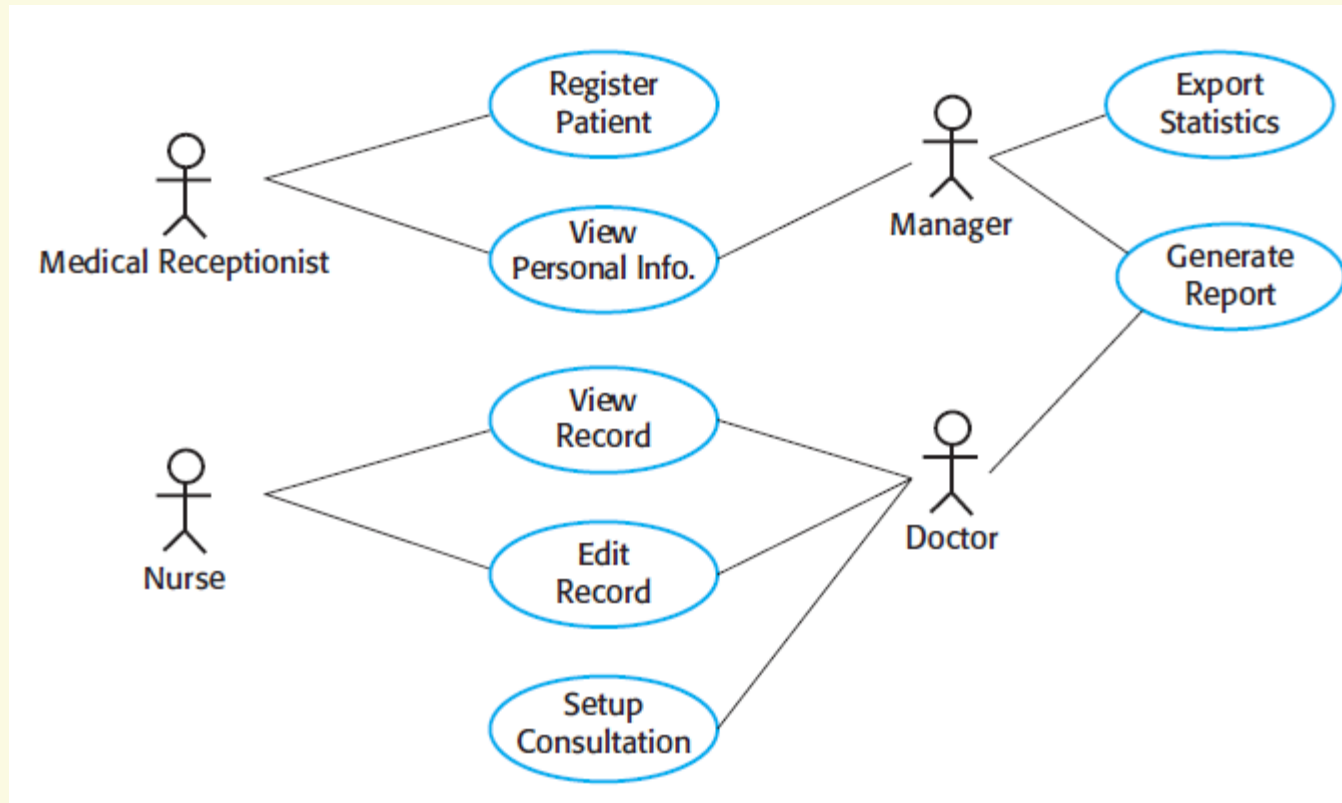
Use Case Example

An example of three use cases that might be part of the inventory management system.



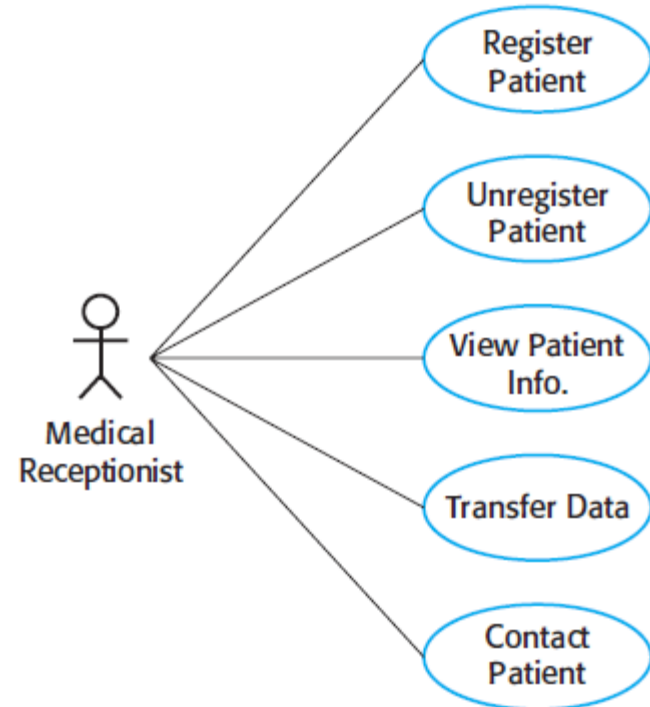
Example

Example shows some of the use cases for the patient information system.



Example

Example shows all of the use cases in the patient information system in which the actor 'Medical Receptionist' is involved



Weather Control station use case

Report weather—send weather data to the weather information system

Report status—send status information to the weather information system

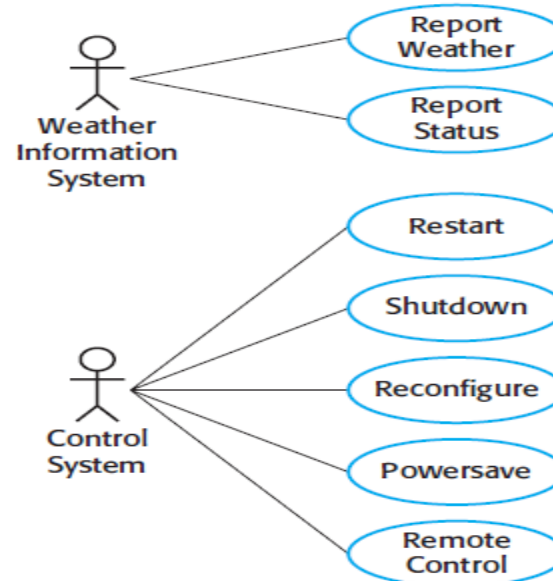
Restart—if the weather station is shut down, restart the system

Shutdown—shut down the weather station

Reconfigure—reconfigure the weather station software

Powersave—put the weather station into power-saving mode

Remote control—send control commands to any weather station subsystem



Relationship types



Include

- Shows dependency between a base use case and an included use case
- Every time the base use case is executed the included use case is executed as well
- The base use case requires an included use case in order to be completed.
- Use a dashed line from a base to the included use case



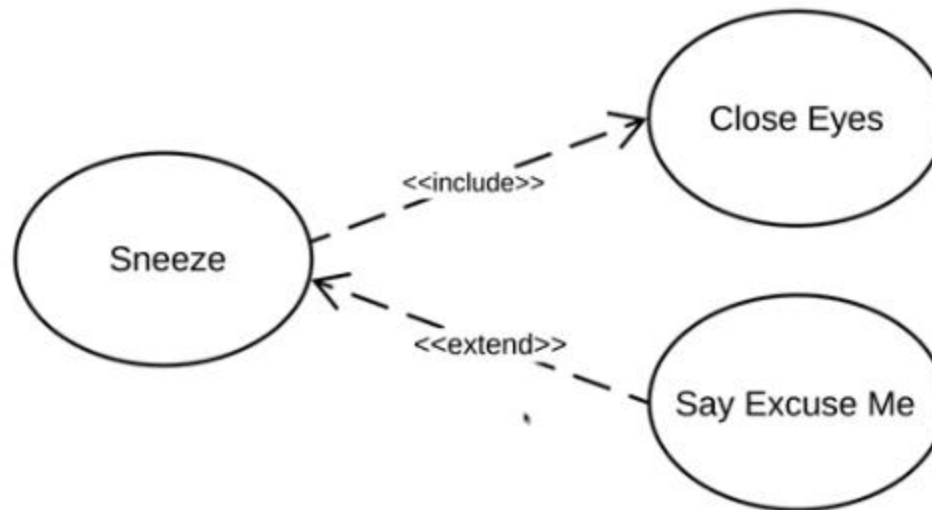
Extends

- When the base use case is executed the included use case is executed the extend use case will happen sometimes but not every time
- The extend use case will only happen if certain criteria are met.
- Use a dashed line from the extend use case to the base use case
- Note that: Multiple base use cases can point to the same included or extended use case



Generalization or inheritance: Relation between general use case and specialized use case E.g. make a payment from a bank can be either from checking account or saving account

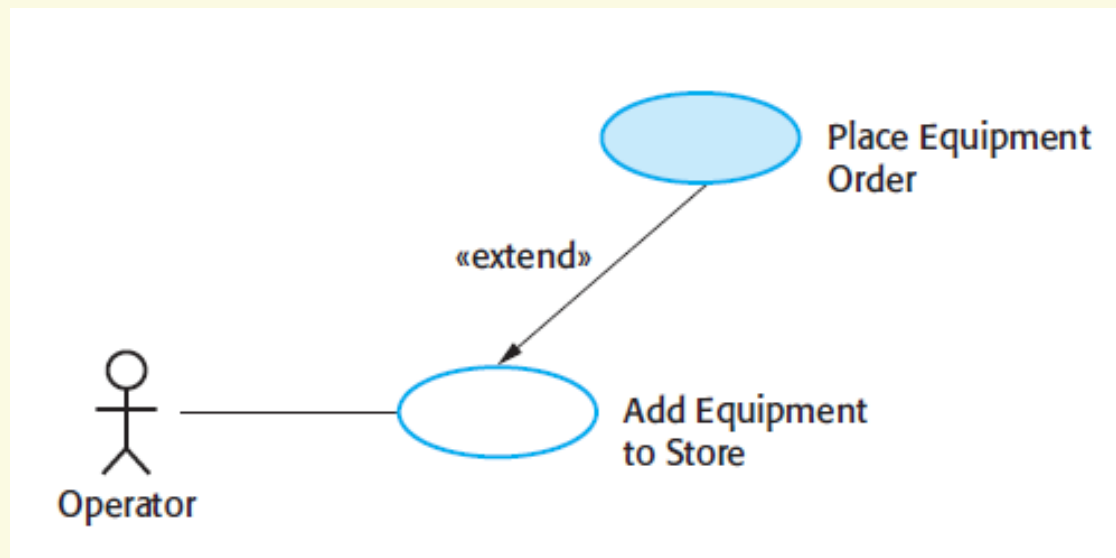
Illustrated Example



Include happens every time
Extend happen just sometimes

Example (Extend)

Example shows how the placing of an equipment order extends the core use case for adding equipment to a specific store. If the equipment to be added does not exist, it can be ordered and added to the store when the equipment is delivered

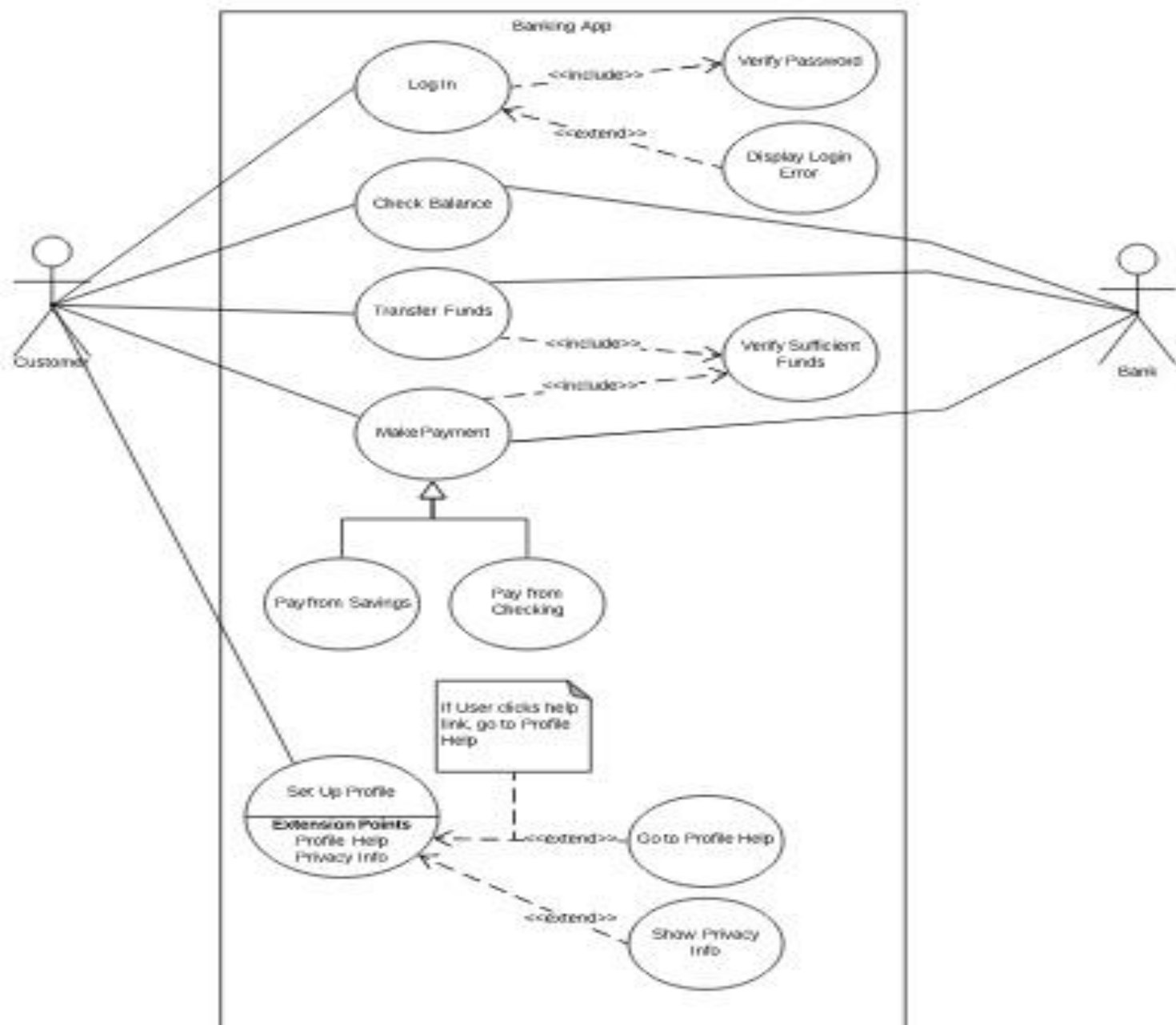


Banking system Use Case Diagram



Actors:

- Customer: download and use banking app.
- Bank : is going to provide information that feeds into the system such as transactions, and account balances
- Is only going to act once the customer does something



Software Engineering Development Methodologies

Development activities **Management activities**

- Deal with the system complexity by constructing models of the system

- Focus on planning the project, monitoring its status, tracking changes and coordinating resources such that a product is delivered on time and within budget

Software Engineering Development Activities



Specification [Requirements analysis]

- The user and developers define the purpose of the system (What is the user wants)
 - The product of this activity is a requirements specification
- ❖ Requirements: are features that the system must have
 - **Functional requirements:** is an area of functionality that the system must support
 - **Non-Functional requirements:** is a constraint on the operation of the system
e.g. (specific hardware or programming language)

Software Engineering Development Activities (Cont.)

Architectural (system) design [Analysis]

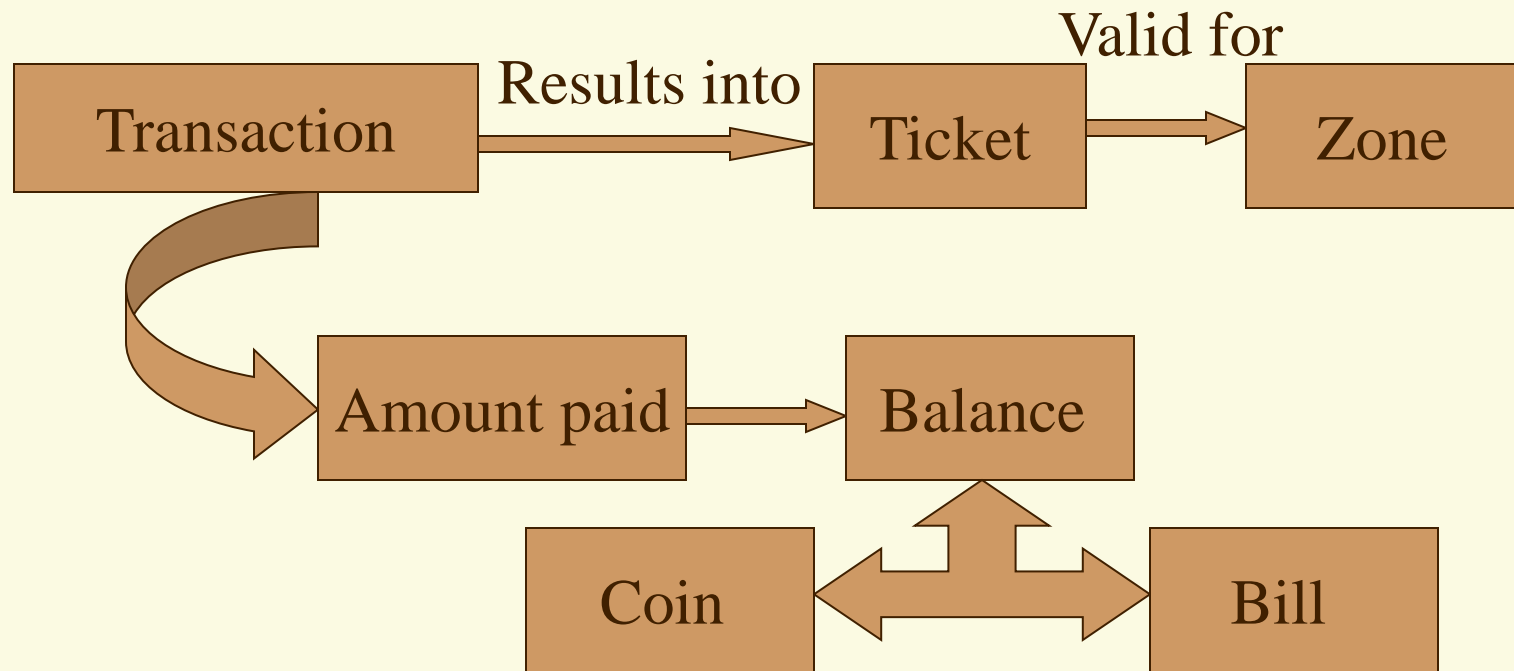
(breaks the overall system down into a number of simpler modules or components)

- The products of this activity are an architectural design and module specification





Detailed design [Object design] (the design of each module or component is carried out)

- The result of this activity is a detailed object model annotated with constraints and precise description of each element


Architectural design Example for Ticket Distributor




Software Engineering Development Activities (Cont.)

-  Coding [Implementation] (The developers translate the object model into source code)
 - The product of this activity is the code
-  System integration (The individual components of the software are combined together)
 - the product is the complete system
-  Verification and Validation
-  Maintenance (Fixing faults)

Verification and Validation


 Verification : it refers to the set of tasks that ensure that software correctly implements a specific function (or The job of removing bugs and trying to ensure reliability)

- Are we building the product right?

 Validation : is a collection of techniques that try to ensure that the software does meet the requirements

- Are we building the right product?

Self test question

 Which stages of software development, if any, can be omitted if the software being developed is only a small program?

Software Engineering Management Activities



Communication

- Includes the exchange of models and documents about the system and its application domain, reporting the status of work products, providing feedback on the quality of work products and communicating decisions
- Communication Tools
 - UML (Unified Modeling Language) diagrams: is an object-oriented notation for representing models
 - CASE (Computer-Aided Software Engineering): It is the implementation of computer facilitated tools and methods in software development.
 - CASE is used to ensure a high-quality and defect-free software. (CASE) describes a broad set of labor-saving tools used in software development. They create a framework for managing projects and are intended to help users stay organized and improve productivity

What is CASE ?

(Computer-Aided Software Engineering)

Software systems which are intended to provide automated support for software process activities, such as requirements analysis, system modelling, debugging and testing

One of the major advantages of using CASE is the delivery of the final product, which is more likely to meet real-world requirements

Upper-CASE (front end CASE)







- Tools to support the early process activities of requirements and design

Lower-CASE (back end CASE)


- Tools to support later activities such as programming, debugging and testing




Types of CASE Tools:

-  Diagramming Tools
-  Computer Display and Report Generators
-  Analysis Tools
-  Central Repository
-  Documentation Generators
-  Code Generators

Types of CASE Tools:

 **Diagramming Tools:** It helps in diagrammatic and graphical representations of the data and system processes. It represents system elements, control flow and data flow among different software components and system structure in a pictorial form.

 For example, Flow Chart Maker tool for making state-of-the-art flowcharts.

Cont.



Computer Display and Report Generators:

It helps in understanding the data requirements and the relationships involved.



Analysis Tools: It focuses on inconsistent, incorrect specifications involved in the diagram and data flow. It helps in collecting requirements, automatically check for any irregularity, imprecision in the diagrams, data redundancies or erroneous omissions.

For example, (i) Accept 360, Accompa, CaseComplete for requirement analysis.



(ii) Visible Analyst for total analysis.

Cont.



Central Repository:

It provides the single point of storage for data diagrams, reports and documents related to project management.



Documentation Generators: It helps in generating user and technical documentation as per standards. It creates documents for technical users and end users.



For example, Doxygen, DrExplain, Adobe RoboHelp for documentation.




Cont.






Code Generators:

It aids in the auto generation of code, including definitions, with the help of the designs, documents and diagrams.

Advantages of CASE


-  The overall quality of the product is improved as an organized approach is undertaken during the process of development.
-  Chances to meet real-world requirements are more likely and easier with a computer-aided software engineering approach.
-  CASE indirectly provides an organization with a competitive advantage by helping ensure the development of high-quality products.

Disadvantages

-  **Cost:** Using case tool is a very costly. Mostly firms engaged in software development on a small scale do not invest in CASE tools because they think that the benefit of CASE are justifiable only in the development of large systems.
-  **Learning Curve:** In most cases, programmers productivity may fall in the initial phase of implementation , because user need time to learn the technology. Many consultants offer training and on-site services that can be important to accelerate the learning curve and to the development and use of the CASE tools.
-  **Tool Mix:** It is important to build an appropriate selection tool mix to urge cost advantage CASE integration and data integration across all platforms is extremely important.


Question

 Upper CASE tools are used in _____ stages of SDLC?

-  A: planning
- B. analysis
- C. design
- D. All of the above

Question

 Which of the following is not a type of CASE tool?

-  A: Diagram tools
- B. Process Modeling Tools
- C. Documentation Tools
- D. Testing tool

Question



What are the **Differences between Use Case and Test Case?**

- A **Use Case** is used to define the system that how to use the system for performing a specific task.
- A use case is not a part of execution it is only a diagrammatic presentation of a document that specifies how to perform a certain task.
- A **Test Case** is defined as a group of test inputs, execution condition, and expected results which further lead to developing a particular test objective.
- If we talk about test case it is used to validate the software which is developed by testers for validating that the software is in working as per requirement or not.

Use Case vs. Test Case



Use Case:

- A sequential actions which is use to describe the interaction among the role and system to maintain a specified objective
- in terms of
 - interaction: user
 - Working: it is working as following the step by step function ability of the software.



Test Case:

- A Group of test inputs, conditions and variables by which the characteristics of the software is defined.
- in terms of
 - Interaction: results
 - Working: it is working with the help of testers to validate the software

Software Engineering Management Activities (Cont.)

Rationale management

- Is the justification of decisions

Testing

- Its goal is to discover any system faults and repairing it before delivering

Software configuration management

- Is the process that monitors and controls changes in work product. It enables developers to track changes

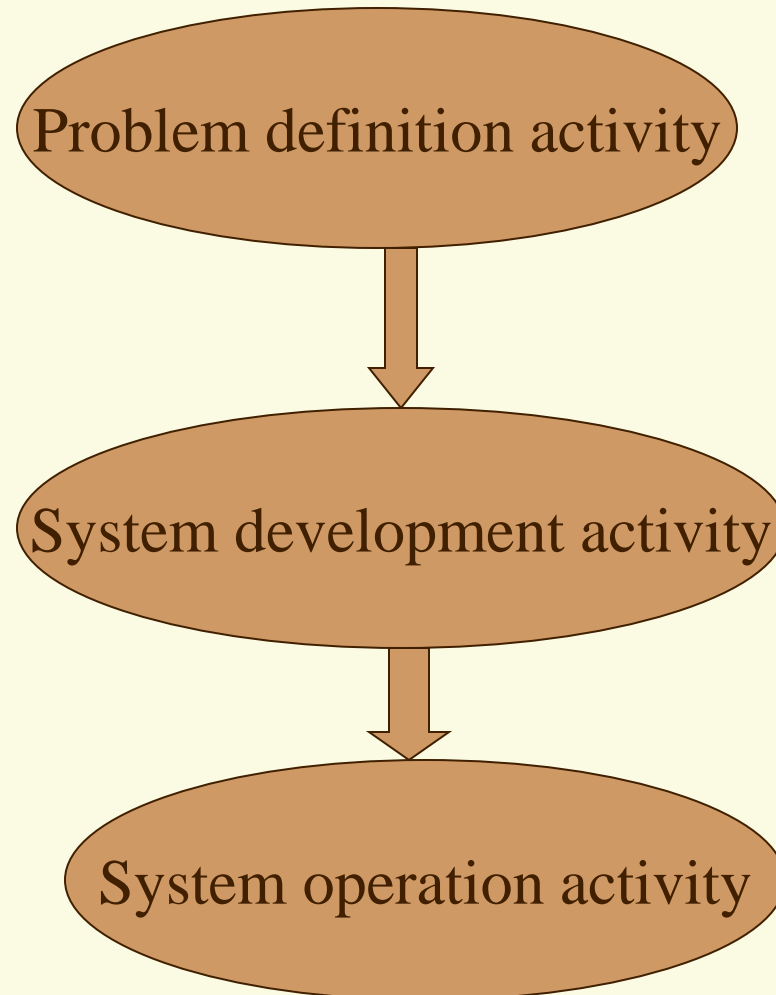
Project management

- Includes the oversight activities that ensure the delivery of a high quality system on time and within budget

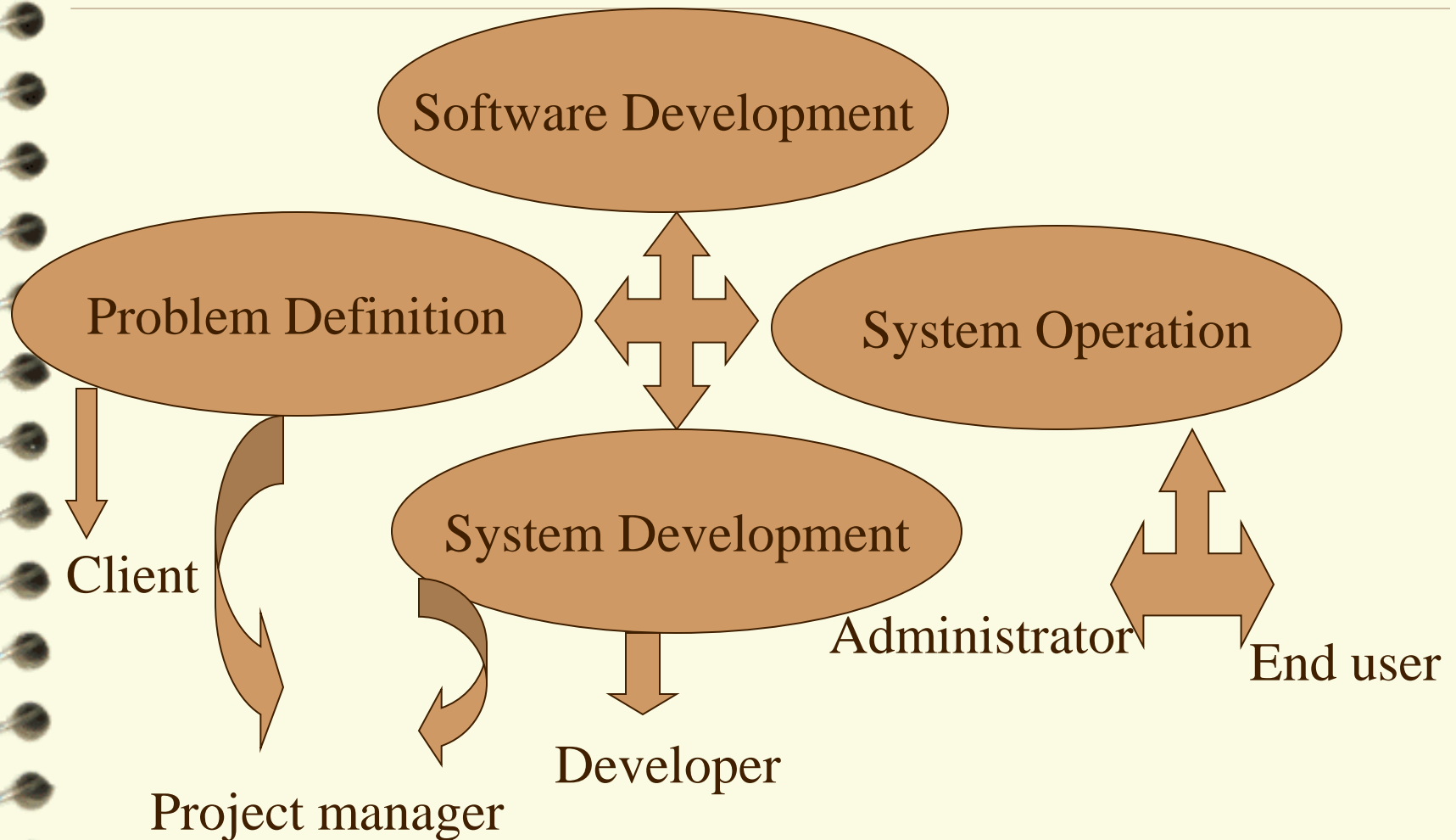
Software life cycle (process models)

- The general model of the software development process


Software life cycle development




Software life cycle development (Cont.)




What are the **costs of software engineering**?

 **Roughly 60% of costs are development costs, 40% are testing costs.** For custom software, evolution costs often exceed development costs

 **Costs vary depending on the type of system** being developed **and the requirements** of system attributes such as performance and system reliability








 **Distribution of costs depends on the development model that is used**

How is the cost made up?








 Which parts of a software development project cost most money

- Testing (1/2)
- Analysis and design (1/3)
- Coding (1/6)

Software Development Components

-  Specification (Requirements analysis)
-  Architectural design (breaks the overall system down into a number of simpler modules or components)
-  Detailed design (the design of each module or component is carried out)
-  coding (The product is the code)
-  System integration (the product is the complete system)
-  Verification and validation
-  Maintenance (Fixing faults)

Relative Costs of the stages of software development

-  Specification 3%
-  Requirements 3%
-  Design 5%
-  Coding 7%
-  Unit test 8%
-  System test 7%
-  Maintenance 67%