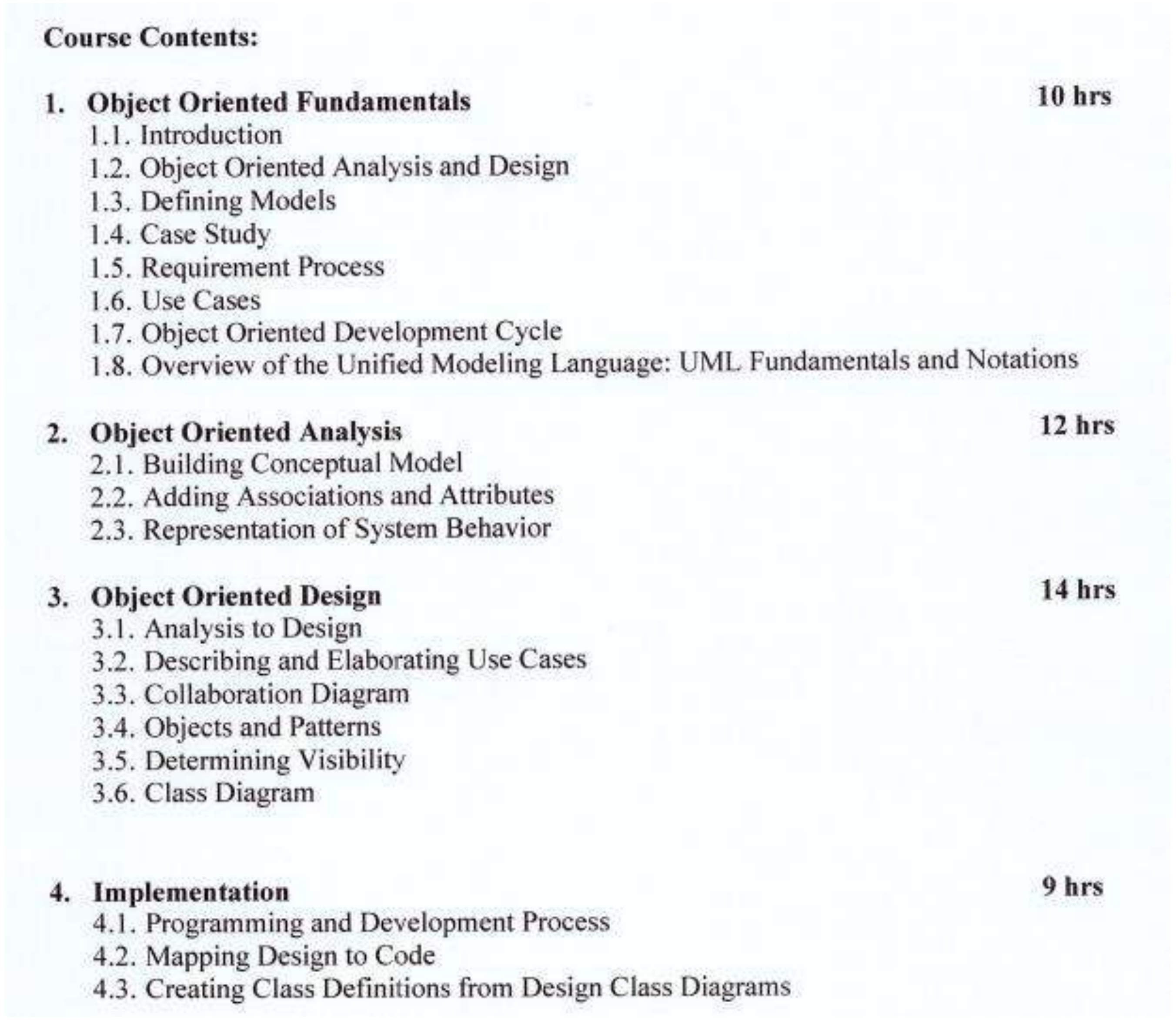
OODM
chp1:Object oriented Fundamentals

Prepared by: Er. Simanta Kasaju

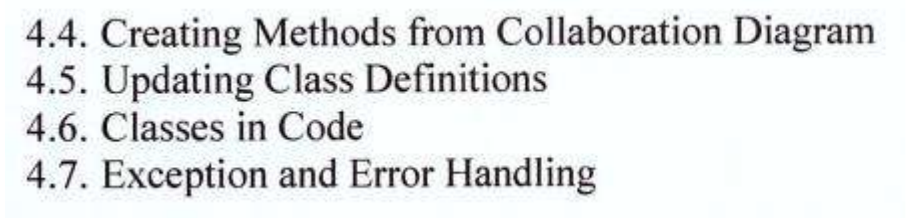

Course Contents:

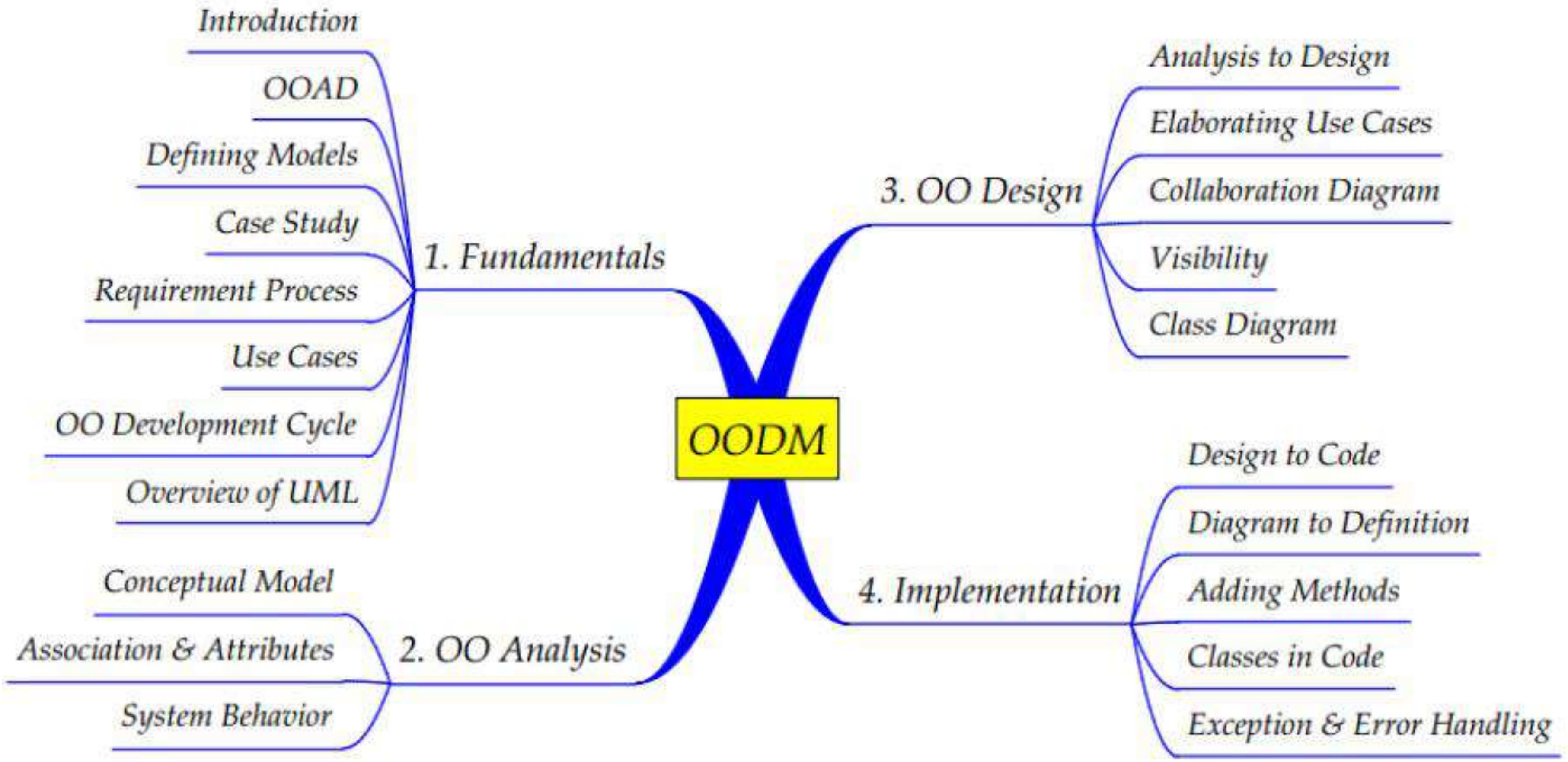
1. **Object Oriented Fundamentals** **10 hrs**
   1.1. Introduction
   1.2. Object Oriented Analysis and Design
   1.3. Defining Models
   1.4. Case Study
   1.5. Requirement Process
   1.6. Use Cases
   1.7. Object Oriented Development Cycle
   1.8. Overview of the Unified Modeling Language: UML Fundamentals and Notations

2. **Object Oriented Analysis** **12 hrs**
   2.1. Building Conceptual Model
   2.2. Adding Associations and Attributes
   2.3. Representation of System Behavior

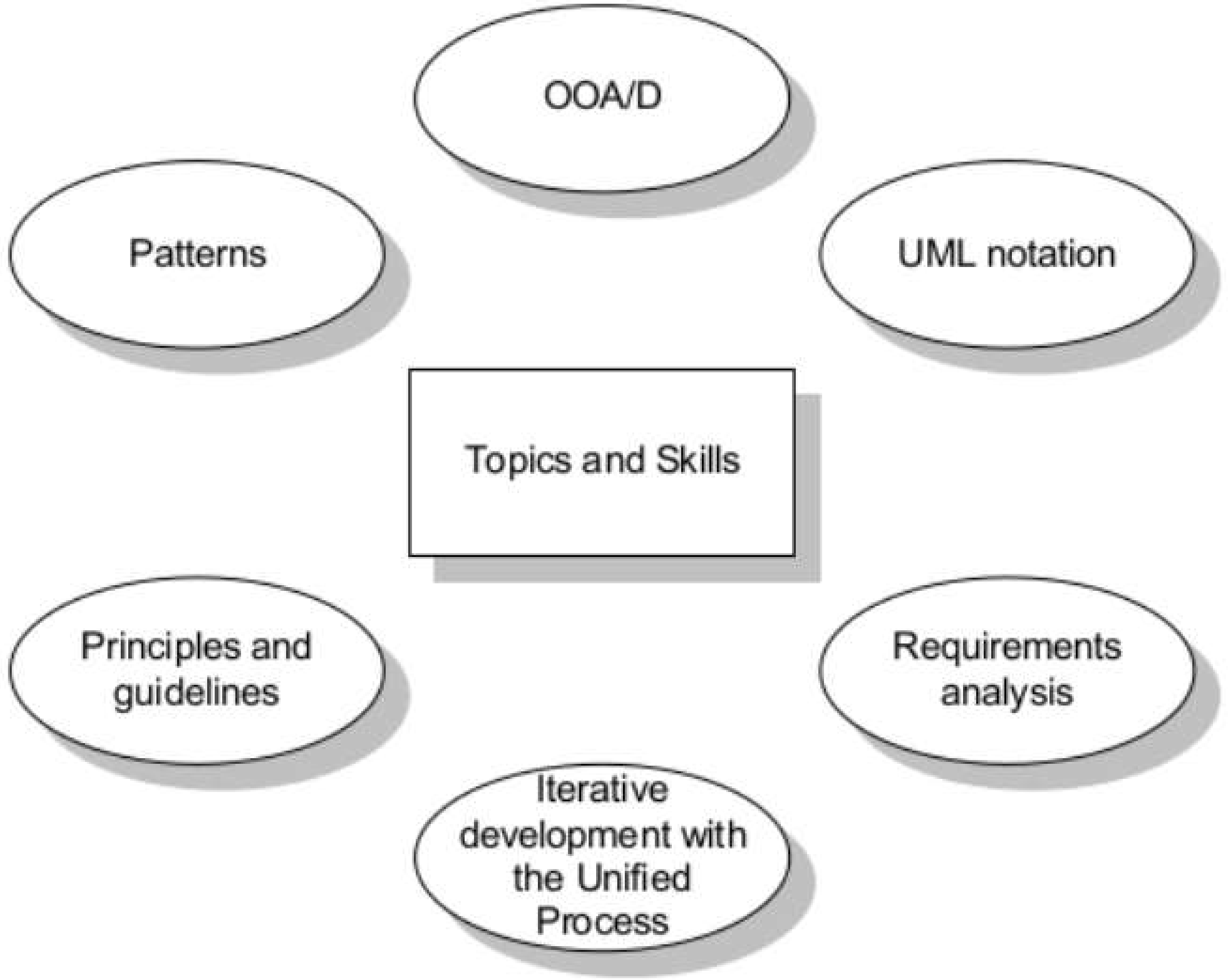
3. **Object Oriented Design** **14 hrs**
   3.1. Analysis to Design
   3.2. Describing and Elaborating Use Cases
   3.3. Collaboration Diagram
   3.4. Objects and Patterns
   3.5. Determining Visibility
   3.6. Class Diagram

4. **Implementation** **9 hrs**
   4.1. Programming and Development Process
   4.2. Mapping Design to Code
   4.3. Creating Class Definitions from Design Class Diagrams


4.4. Creating Methods from Collaboration Diagram
4.5. Updating Class Definitions
4.6. Classes in Code
4.7. Exception and Error Handling


OODM
1. Fundamentals
Introduction
OOAD
Defining Models
Case Study
Requirement Process
Use Cases
OO Development Cycle
Overview of UML
2. OO Analysis
Conceptual Model
Association & Attributes
System Behavior
3. OO Design
Analysis to Design
Elaborating Use Cases
Collaboration Diagram
Visibility
Class Diagram
4. Implementation
Design to Code
Diagram to Definition
Adding Methods
Classes in Code
Exception & Error Handling

Course objectives


OOA/D
Patterns
UML notation
Topics and Skills
Principles and guidelines
Requirements analysis
Iterative development with the Unified Process

OBJECT ORIENTED CONCEPTS

→In case of an Object-Oriented System, the overall architecture of the system is perceived as a **collection of objects that are assigned specific responsibilities**. These objects exhibit their **behavior based on the type of responsibility assigned to them**. Thus, the main **goal of the system is achieved through the collaboration of objects** that exist in the system.

The steps in designing an Object-Oriented System comprises of mainly

i. **Identification of objects** that exist in a domain

ii. **Assigning responsibilities** to these objects so that each object can exhibit the desired behavior

iii. **Seek collaboration** between these objects for fulfilling the goal of the system (computation as a simulation)

Responsibility Driven Design

- Just like in our daily lives we get to see different entities fulfilling their goals independently, in case of an object-oriented system, objects exhibit behavior based on the kind of responsibility that has been assigned to it.
- We expect a lamp to glow whenever we pass it a message to glow by switching the lamp on and we do not expect a lamp to rotate or make sound.

Object oriented concepts

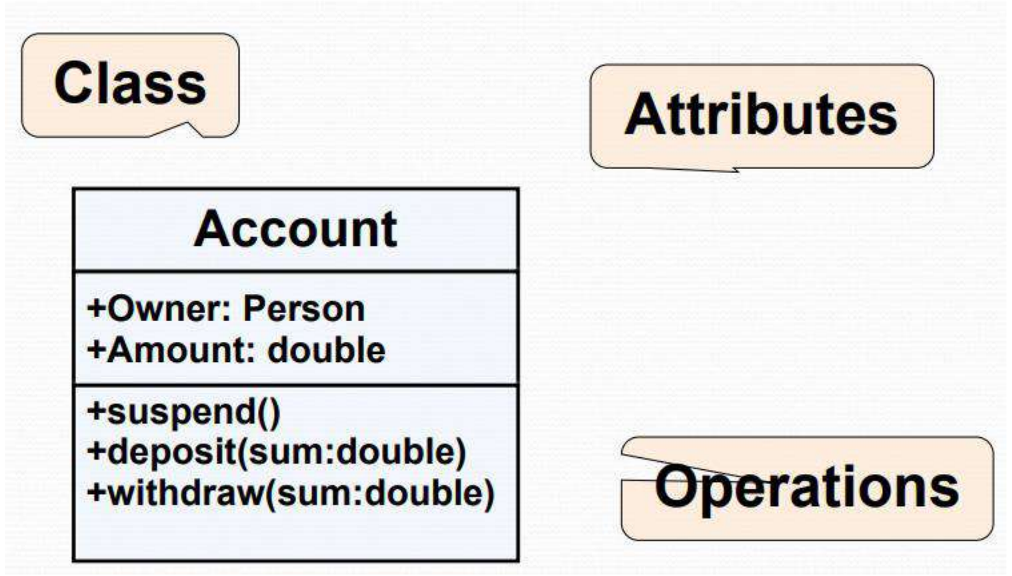
- Software is organized as a collection of discrete objects that incorporate both State and behavior.
- Four aspects (characteristics) required by an OO approach are - Identity Classification Polymorphism Inheritance

Identity

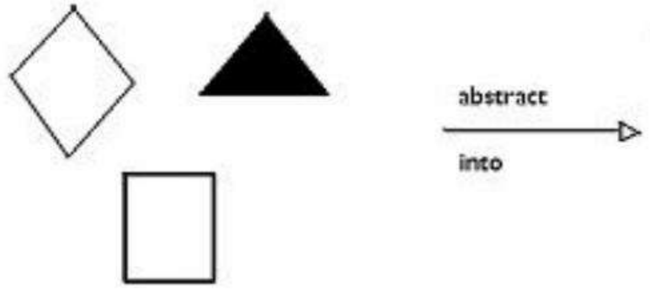
- Identity means that data is organized into discrete, distinguishable entities called objects.
- E.g. for objects: personal computer, bicycle
- Objects can be concrete (such as a file in a file system) or
- conceptual (such as scheduling policy in a multiprocessing
- OS). Each object has its own inherent identity. (i.e two objects
- are distinct even if all their attribute values are identical).

Classification

- It means that objects with same data structure (attribute) and
- behavior (operations) are grouped into a class.
- Each object is said to be an instance of its class.
- A class is simply a representation of a type of object. It is the blueprint/ plan/ template that describe the details of an object.
- A class is the blueprint from which the individual objects are created.



Classification



Polygon class:

Attributes-

- -vertices.
- -border class.
- -fill color.

Operations-

- -draw.
- -erase.
- -move.

Bycycle class:

Attributes-

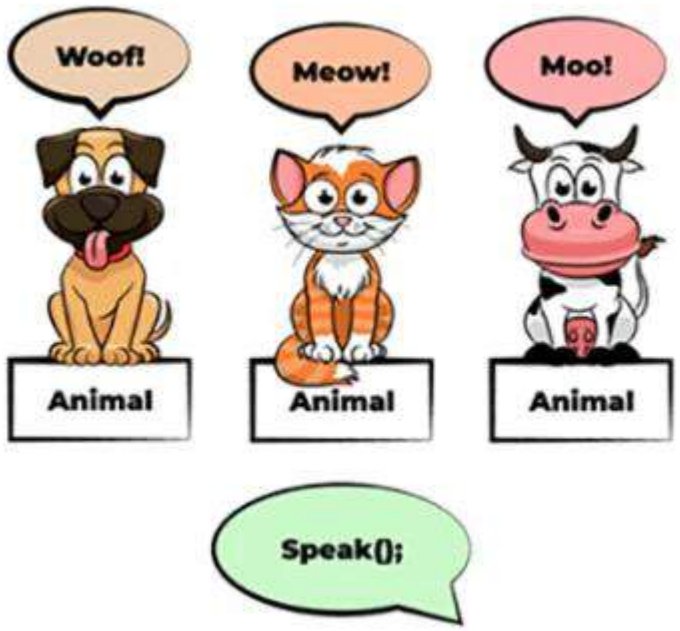
- - frame size
- -wheel size
- no. of gears
- -material

Operations-

- -shift
- -move
- -repair

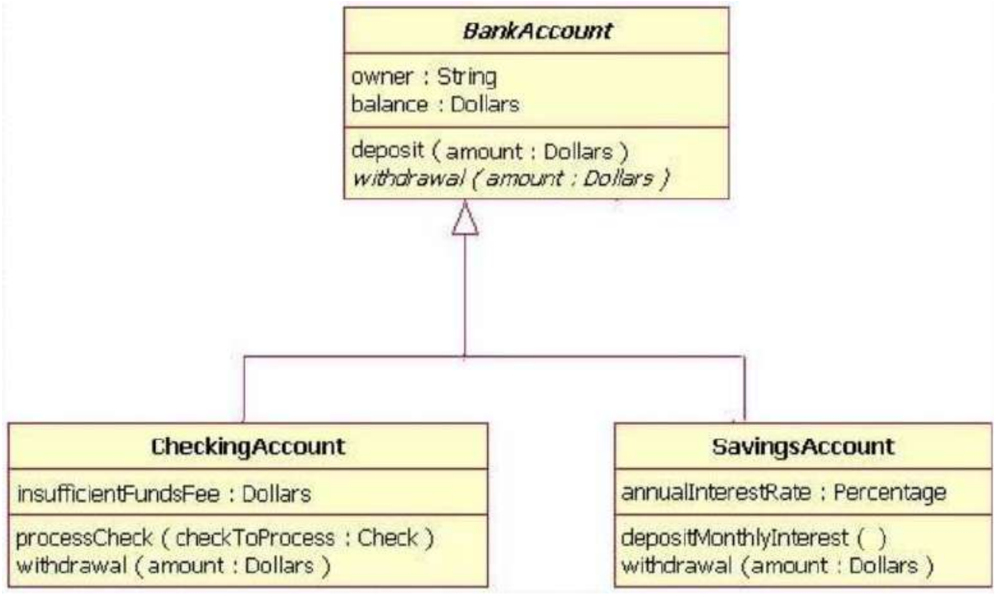
Polymorphism

- It means that the same operation (i.e. action or transformation that the object performs) may behave differently on different classes.
- Ability of different objects to response same message in different ways.



Inheritance

- One class can be used to derive another via inheritance
- Classes can be organized into hierarchies.
- t is the sharing of attributes and operations among classes based on a hierarchical relationship.
- Subclasses can be formed from broadly defined class.


BankAccount
owner : String
balance : Dollars
deposit (amount : Dollars)
withdrawal (amount : Dollars)
CheckingAccount
insufficientFundsFee : Dollars
processCheck (checkToProcess : Check)
withdrawal (amount : Dollars)
SavingsAccount
annualInterestRate : Percentage
depositMonthlyInterest ( )
withdrawal (amount : Dollars)

Decomposition

→It is the process of partioning the problem domain into smaller parts so that the overall complexity of the problem can be comprehended and tackled easily (divide and conquer approach)

- Algorithmic/Functional decomposition

  Dividing the problem domain into smaller parts by focusing on the functionalities that the system provides. For example, functionalities of a Library Management System may include issue book, return book, search book

- Object Oriented Decomposition

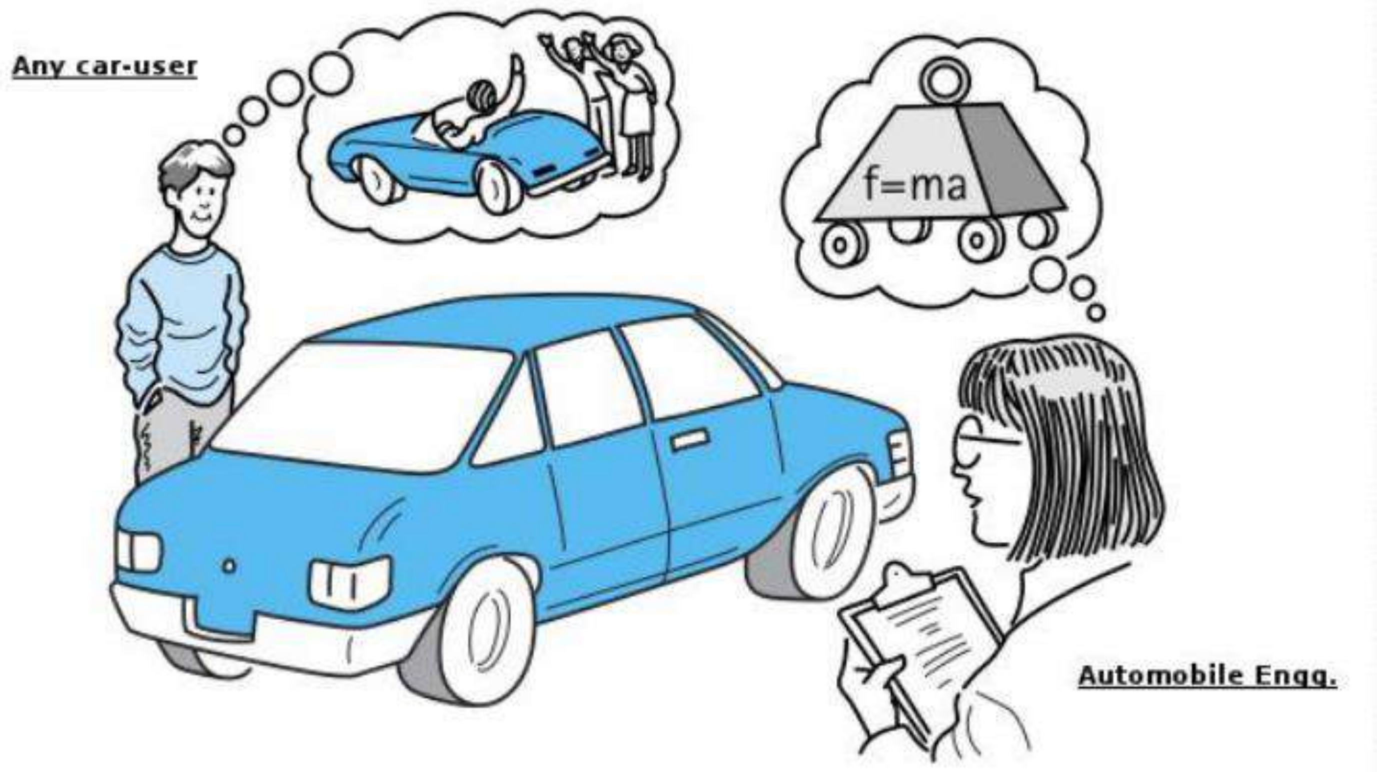
  Dividing the problem domain into smaller parts by focusing on the objects with specific responsibilities that form the system. For example, objects in a Library Management System may include Issue Manager, Book, Librarian

→In an object-oriented system, the overall goal of the system is decomposed into sub goals and then responsibilities are assigned to specific objects. So, the overall functionality of the system is achieved thru the collaboration of these objects.

Abstraction

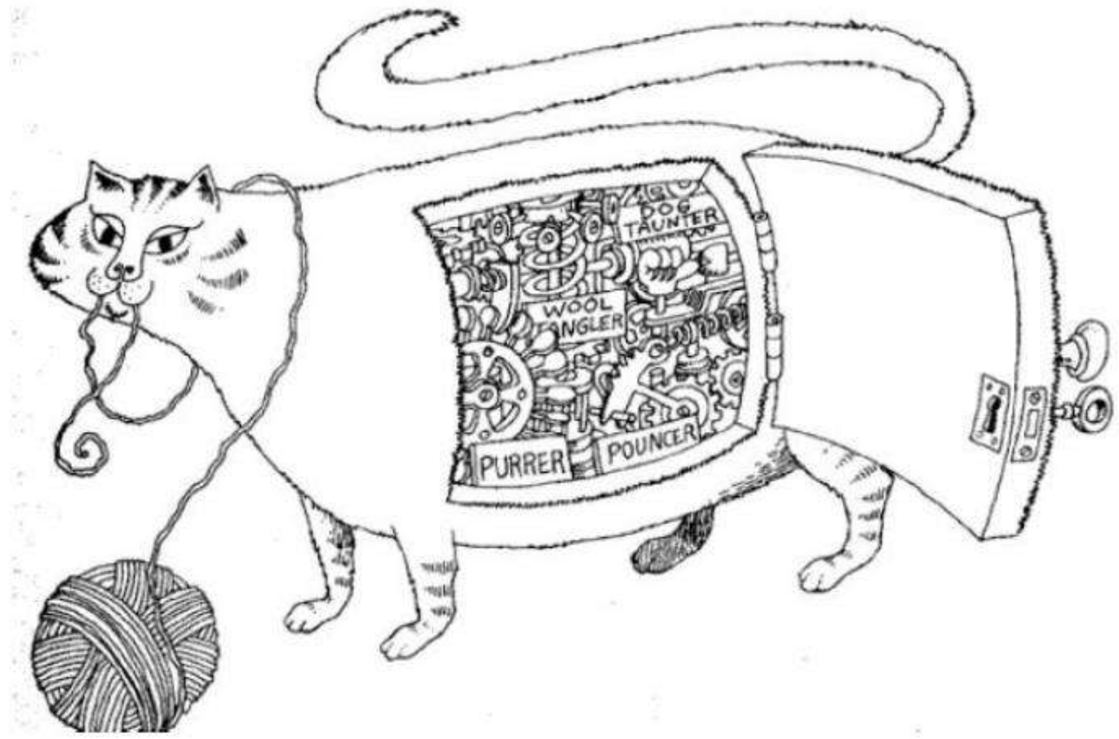
- Abstractions are created by hiding the details and representing only the necessary features
- Forming abstractions help in communicating ideas with others and in representing complexity in a much-simplified form.
- Abstractions can be formed of objects, ideas or functionalities.
  - moving towards the door
  - grabbing its handle
  - pulling it towards you

- So, if you want to ask someone to open the door you do not need to relate to him the details regarding how it is done every time
- A person who is a human being , who has a name and id, who works in a library, who can issue, return and stock book is represented by an abstraction called Librarian. It means to filter out an object's property and operations until just the one needed are left



An abstraction includes the essential details relative to the perspective of the viewer

Encapsulation



Encapsulation hides the details of the implementation of an object.

encapsulation describes the idea of bundling data and methods that work on that data within one unit, like a class in Java. This concept is also often used to hide the internal representation, or state of an object from the outside. This is called information hiding.

At the end of the course the student would have the:

- The Knowledge of the basic concepts of Object oriented modeling and Design.
- Will be able to use the Object Oriented notations and process that extends from analysis through design to implementations.
- Be able to use all the standard UML notations.
- Capable to model the requirements with use cases and describe the dynamic behavior and structure of the design.
- Easily create a modular design with components and relate the logical design to the physical environment.
- The Student will be able to use the concept of design patterns and apply it where suitable

Object-oriented analysis(OOA)

- It is method of analysis that examines requirements from the perspective of the classes and objects found in the vocabulary of the problem domain.
- A thorough investigation of the problem domain is done during this phase by asking the WHAT questions (rather than how a solution is achieved)
- During OO analysis, there is an emphasis on finding and describing the objects (or concepts) in the problem domain.
- For example, concepts in a Library Information System include Book, and Catalog.

Object-oriented design(OOD)

- It is method of design encompassing the process of the object-oriented decomposition and notation for depicting both logical and physical as well as static and dynamic models of the system under design
- There are two important parts of this definition
  - It leads to object-oriented decomposition
  - Uses different notations to express different models of the logical (class and object) structure and physical (module and cross architecture) design of a system in the addition to the static and dynamic aspects of a system
- Emphasizes a conceptual solution that fulfils the requirements.
- There is a need to define software objects and how they collaborate to fulfill the requirements.
- For example, in the Library Information System, a Book software object may have a title attribute and a display() method. The issueManager may be asked to issue a book, in response this object may ask the BookRecord object to update the status of the book.

- Designs are implemented in a programming language.

OOA	OOD
Elaborate a problem	Provide conceptual solution
WHAT type of questions asked	HOW type of questions asked
During Analysis phase questions asked include Q. What is required in the Library Information System? A. Authentication!!	Later during Design phase questions asked include Q. How is Authentication in the Library Information System achieved? A. Thru Smart Card!! Or Fingerprint!!

Object Oriented Programming

- It is a method of implementation in which programs are organized as co operative collection of objects each of which represents an instance of some class and whose classes are all members of a hierarchy of classes united via inheritance relationship.
- Three important aspects of definition
  - Uses objects not algorithms as fundamental building blocks
  - Each object is an instance of some class
  - Classes are related to one another via inheritance

Characteristics of OOD

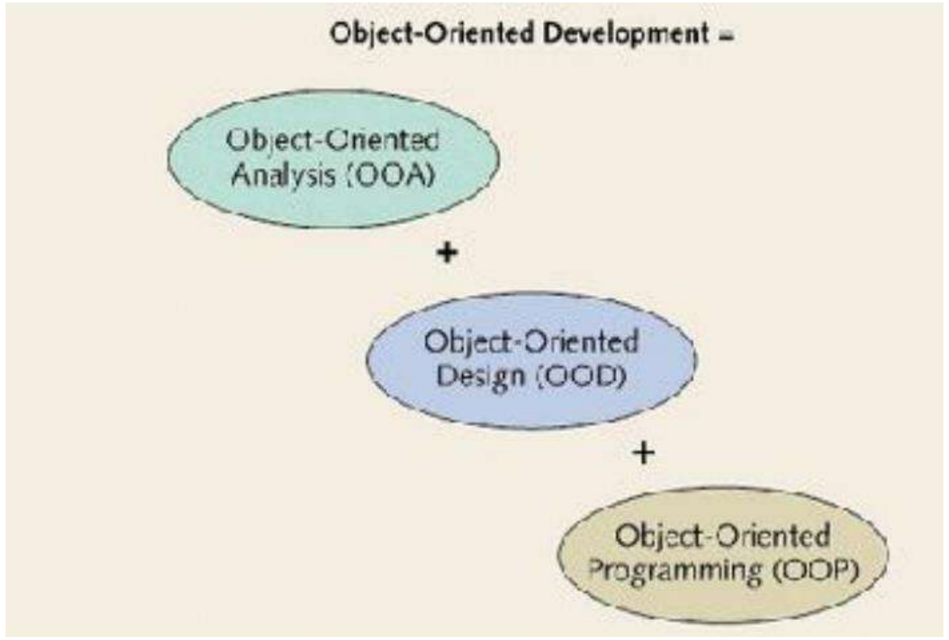
- Objects are abstractions of real-world or system entities and manage themselves
- Objects are independent and encapsulate state and representation information.
- System functionality is expressed in terms of object services
- Shared data areas are eliminated. Objects communicate by message passing
- Objects may be distributed and may execute sequentially or in parallel

Advantages of OOD

- Easier maintenance. Objects may be understood as stand-alone entities
- Objects are appropriate reusable components
- For some systems, there may be an obvious mapping from real world entities to system objects

Object-oriented development

- Object-oriented analysis, design and programming are related but distinct
- OOA is concerned with developing an object model of the application domain
- OOD is concerned with developing object-oriented system model to implement requirements
- OOP is concerned with realizing an OOD using an OO programming language such as C++


Object-Oriented Development =
Object-Oriented Analysis (OOA)
+
Object-Oriented Design (OOD)
+
Object-Oriented Programming (OOP)

Object-oriented design methods

- Some methods which were originally based on functions (such as the Yourdon method) have been adapted to object-oriented design.
- Other methods such as the Booch method have been developed specifically for OOD
- OOD is an object-oriented design method developed to support Ada programming.

OO Design method commonality

- The identification of objects, their attributes and services
- The organization of objects into an aggregation hierarchy
- The construction of dynamic object-use descriptions which show how services are used
- The specification of object interfaces

Object-Oriented Design and Modelling (OODM)

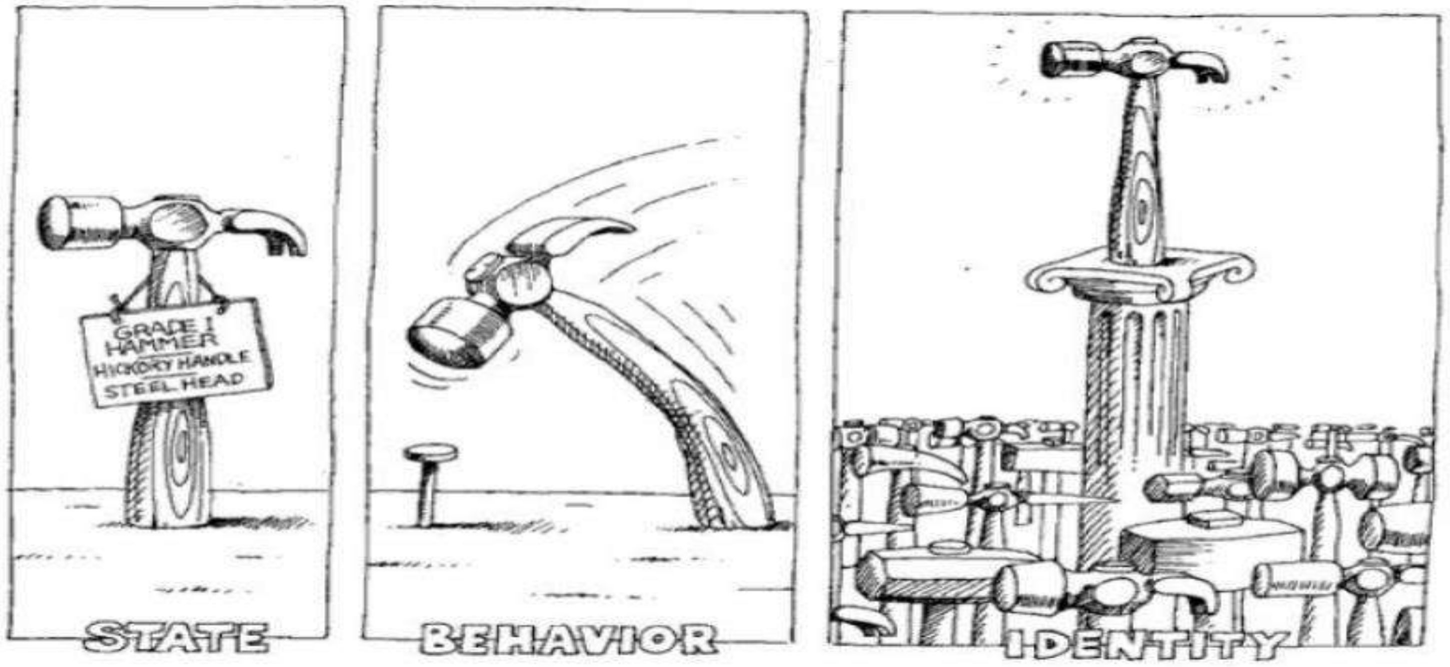
- Object-Oriented Modelling and Design is a way of thinking about problems using models organized around real world concepts. The fundamental construct is the object, which combines both data and behavior.

Objects

- An object is an entity which has a state and a defined set of operations which operate on that state.
- The state is represented as a set of object attributes.
- The operations associated with the object provide services to other objects (clients) which request these services when some computation is required.
- Objects are created according to some object class definition. An object class definition serves as a template for objects. It includes declarations of all the attributes and services which should be associated with an object of that class.
- Objects are entities in a software system, that represent instances of real-world and system entities
- Object classes are templates for objects. They may be used to create objects
- Object classes may inherit attributes and services from other object classes

Objects have three responsibilities:

- What they know about themselves – (e.g- Attributes)
- What they do – (e.g-Operations)
- What they know about other objects – (e.g-Relationships)

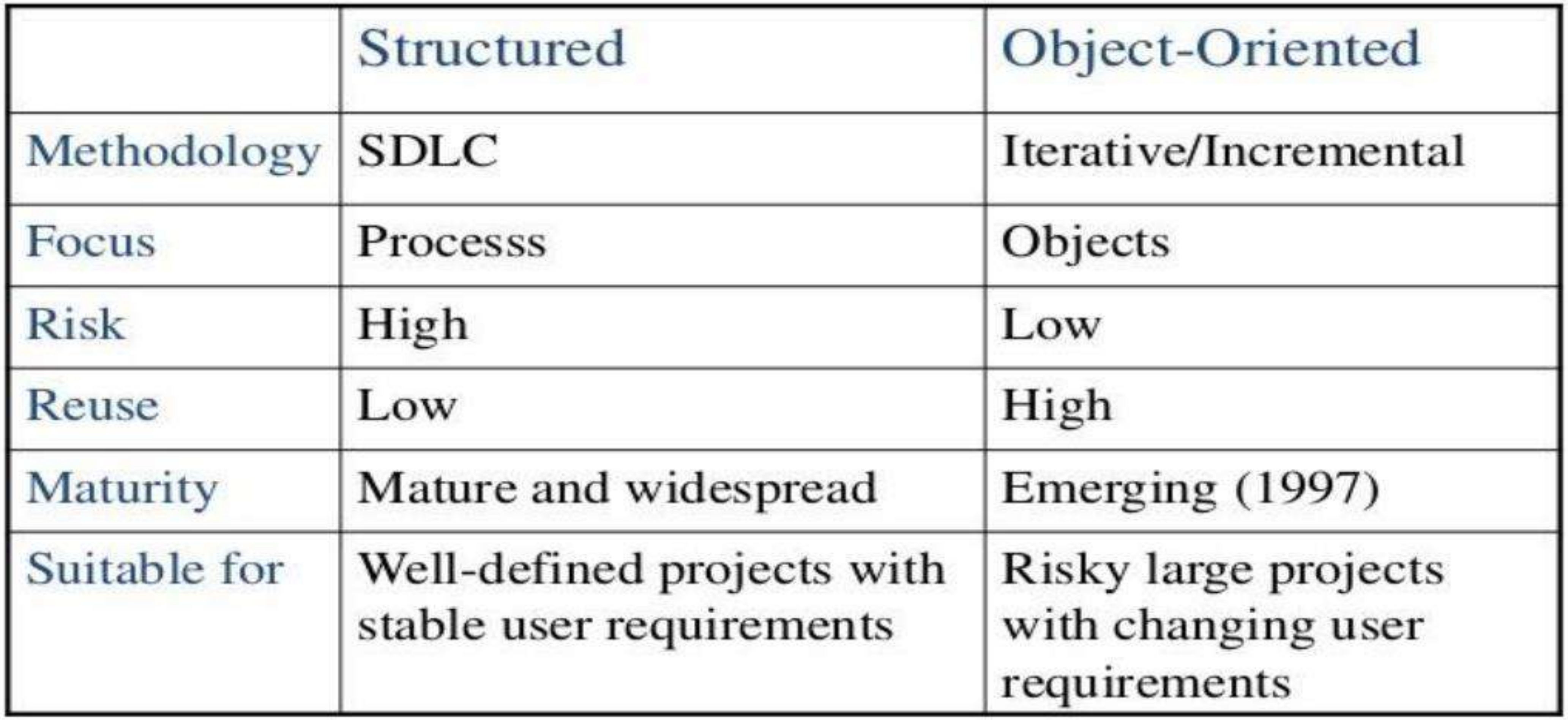


An object has state, exhibits some well-defined behavior, and has a unique identity.

Assigning Responsibilities

- A critical, fundamental ability in OOA/D is to skillfully assign responsibilities to software components
- It strongly influences the robustness, maintainability, and reusability of software components.
- In an object-oriented system every object can accomplish a specific goal and it does so by fulfilling the responsibilities assigned to it.
- e.g., a dice object can come up with a random face value by fulfilling the responsibility it has been assigned that is to roll()
- e.g., a book object can reveal its name by fulfilling the responsibility it has been assigned that is to getBookInfo()

Key Differences Between Structured and Object-Oriented Analysis and Design



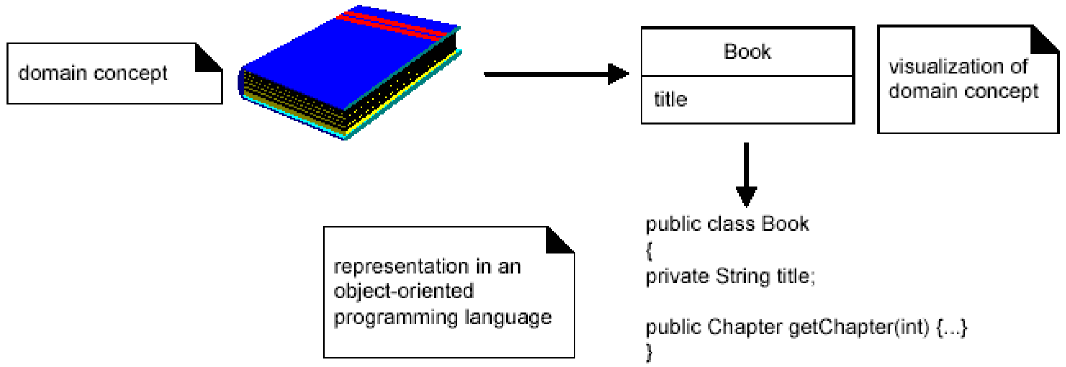
	Structured	Object-Oriented
Methodology	SDLC	Iterative/Incremental
Focus	Processs	Objects
Risk	High	Low
Reuse	Low	High
Maturity	Mature and widespread	Emerging (1997)
Suitable for	Well-defined projects with stable user requirements	Risky large projects with changing user requirements

Analysis and design

- Analysis emphasizes an investigation of the problem and requirements, rather than a solution. For example, if a new computerized library information system is desired, how will it be used?
- “Analysis” is a broad term, best qualified, as in requirements analysis (an investigation of the requirements) or object analysis (an investigation of the domain objects).
- Design emphasizes a conceptual solution that fulfills the requirements, rather than its implementation. For example, a description of a database schema and software objects. Ultimately, designs can be implemented.
- Analysis and design have been summarized in the phase do the right thing (analysis),and do the thing right (design).

Object oriented analysis and design

- During object-oriented analysis, there is an emphasis on finding and describing the objects—or concepts—in the problem domain. For example, in the case of the library information system, some of the concepts include Book, Library, and Patron.
- During object-oriented design, there is an emphasis on defining software objects and how they collaborate to fulfill the requirements. For example, in the library system, a Book software object may have a title attribute and a getChapter method
- Finally, during implementation or object-oriented programming, design objects are implemented, such as a Book class in Java.
- Object-orientation emphasizes representation of objects.


domain concept
Book
title
visualization of domain concept
representation in an object-oriented programming language

```
public class Book
{
private String title;

public Chapter getChapter(int) {...}
}
```

Requirements

- Capabilities and conditions to which the system/ project must conform
- Types of requirements
- Functional
- Non functional

Functional requirements

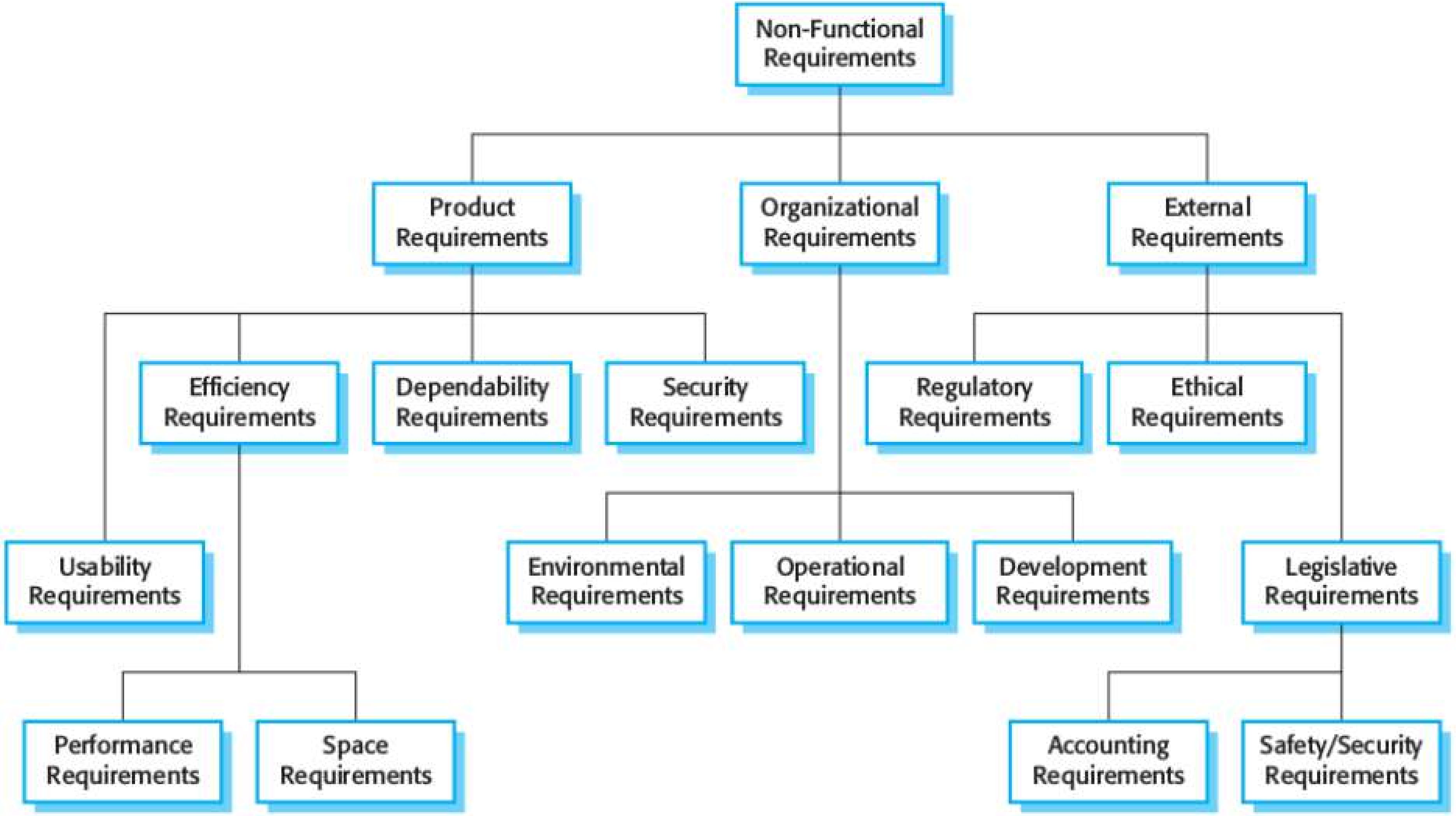
- What the system do?
- List of actual service which a system will provide.
- List of the services/functions which users want from the s/w.
- Functional requirements are product features or functions that developers must implement to enable users to accomplish their tasks.
- Generally, functional requirements describe system behavior under specific conditions.
- It is a description of the service that the software must offer.
- It describes a software system or its component.

Functional Requirements of a system should include the following things:

- Details of operations conducted in every screen
- Data handling logic should be entered into the system
- It should have descriptions of system reports or other outputs
- Complete information about the workflows performed by the system
- It should clearly define who will be allowed to create/modify/delete the data in the system
- How the system will fulfill applicable regulatory and compliance needs should be captured in the functional document
- Types:
- Transaction Handling
- Business Rules
- Certification Requirements
- Reporting Requirements

Non-Functional requirements

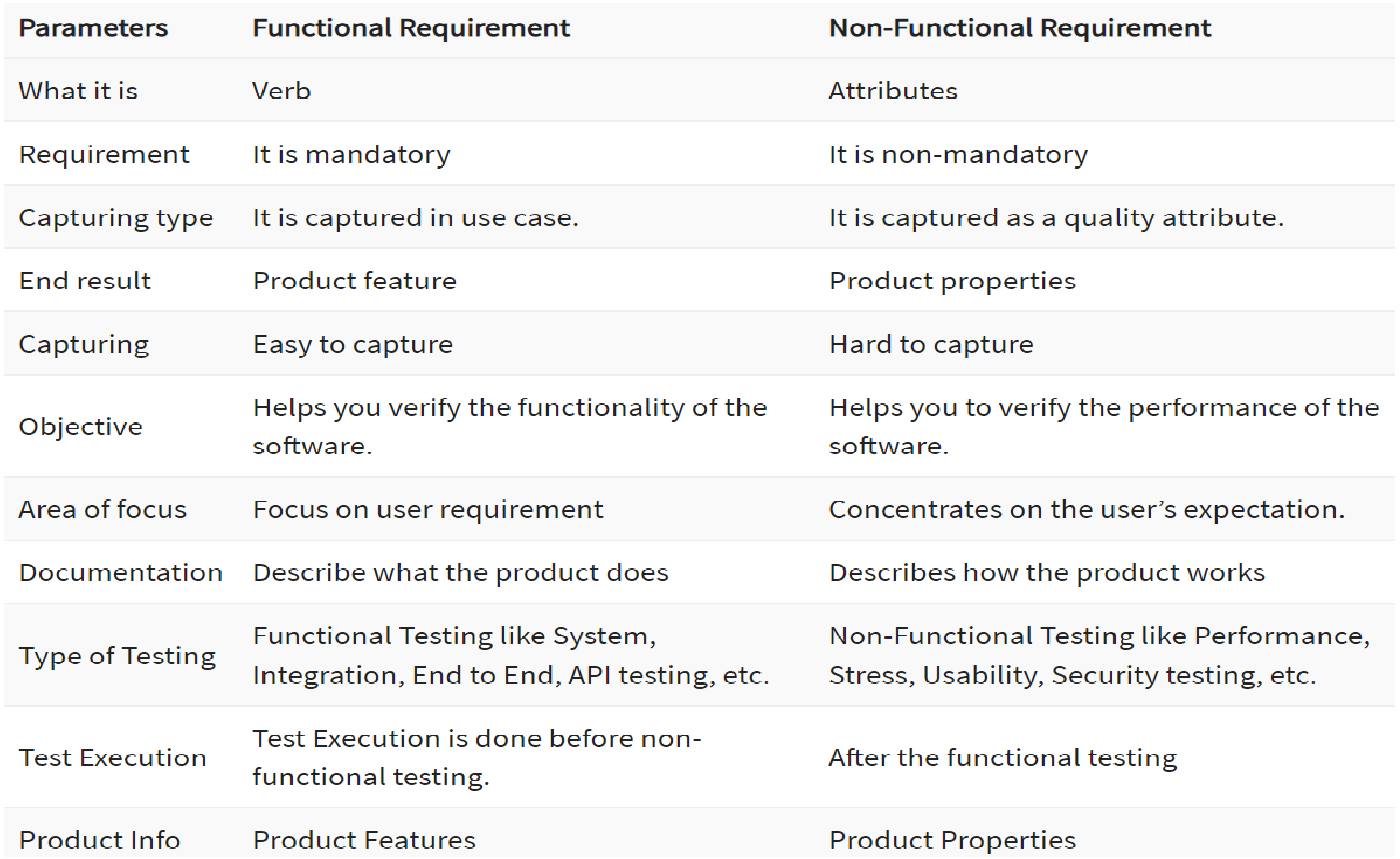
- How a system should behave while performing the operations.
- These are basically the quality constraints that the system must satisfy according to the project contract. The priority or extent to which these factors are implemented varies from one project to other. They are also called non-behavioral requirements.
- They basically deal with issues like:
- Portability
- Security
- Maintainability
- Reliability
- Scalability
- Performance
- Reusability
- Flexibility


Non-Functional Requirements
Product Requirements
Organizational Requirements
External Requirements
Efficiency Requirements
Dependability Requirements
Security Requirements
Regulatory Requirements
Ethical Requirements
Usability Requirements
Environmental Requirements
Operational Requirements
Development Requirements
Legislative Requirements
Performance Requirements
Space Requirements
Accounting Requirements
Safety/Security Requirements

Metrics for specifying nonfunctional requirements

Property	Measure
Speed	Processed transactions/second User/event response time Screen refresh time
Size	Mbytes Number of ROM chips
Ease of use	Training time Number of help frames
Reliability	Mean time to failure Probability of unavailability Rate of failure occurrence Availability
Robustness	Time to restart after failure Percentage of events causing failure Probability of data corruption on failure
Portability	Percentage of target dependent statements Number of target systems

Functional vs non Functional



Parameters	Functional Requirement	Non-Functional Requirement
What it is	Verb	Attributes
Requirement	It is mandatory	It is non-mandatory
Capturing type	It is captured in use case.	It is captured as a quality attribute.
End result	Product feature	Product properties
Capturing	Easy to capture	Hard to capture
Objective	Helps you verify the functionality of the software.	Helps you to verify the performance of the software.
Area of focus	Focus on user requirement	Concentrates on the user's expectation.
Documentation	Describe what the product does	Describes how the product works
Type of Testing	Functional Testing like System, Integration, End to End, API testing, etc.	Non-Functional Testing like Performance, Stress, Usability, Security testing, etc.
Test Execution	Test Execution is done before non-functional testing.	After the functional testing
Product Info	Product Features	Product Properties

Types of Requirements:

- FURPS +
- An acronym representing a model for classifying software quality attributes (functional & non-functional requirement)
- Functionality—features, capabilities, security.
- Usability—human factors, help, documentation.
- Reliability—frequency of failure, recoverability, predictability
- Performance/ Productivity—response times, throughput, accuracy, availability, resource usage
- Supportability—adaptability, maintainability, internationalization, configurability.
- The "+" in FURPS+ indicates ancillary and sub-factors, such as:
- Implementation—resource limitations, languages and tools, hardware, ...
- Operations—system management in its operational setting.
- Packaging Legal—licensing and so forth.

REQUIREMENT PROCESS

1) Fact Finding and Requirements Elicitations

- In order to identify all relevant requirements analysts must use a range of techniques:-
- • Fact-Finding Overview
- – First, you must identify the information you need
- – Develop a fact-finding plan
- Fact finding can be done through Document review, observation, questionnaire, surveys,
- sampling, research.

2) Interviewing

- The analyst starts by asking context-free questions;

- - a set of questions that will lead to a basic understanding of the problem, the people
- who want a solution, the nature of the solution desired, and the effectiveness of the first
- encounter itself.
- Structured Questioning Techniques usually lead by software engineer :
- –why are we building this system
- –who are the other users
- –determine critical functionality

3) Open-ended questions

–useful when not much is known yet , broad-oriented, free of context.

4) Closed-ended questions

–when enough about the system is known try to ask specific questions

–example "how often should sales reports be generated?”

Try to proceed from open-ended to closed-ended questions!!

Find out who else to interview

–who else uses the system

–who interacts with you

–who will agree / disagree with you

5) Facilitated Application Specification Technique (FAST)

- Facilitated Application Specification Techniques (FAST), this approach encourages the creation of a joint team of customers and developers who work together to identify the problem,
- propose elements of the solution, negotiate different approaches, and specify a preliminary set of requirements.
- The idea is to overcome we/them attitude between developers and users team-oriented approach

Guidelines

- participants must attend entire meeting
- all participants are equal
- preparation is as important as meeting
- all pre-meeting documents are to be viewed as “proposed”
- off-site meeting location
- set an agenda and maintain it
- don’t get mired in technical detail

Approaches to FAST

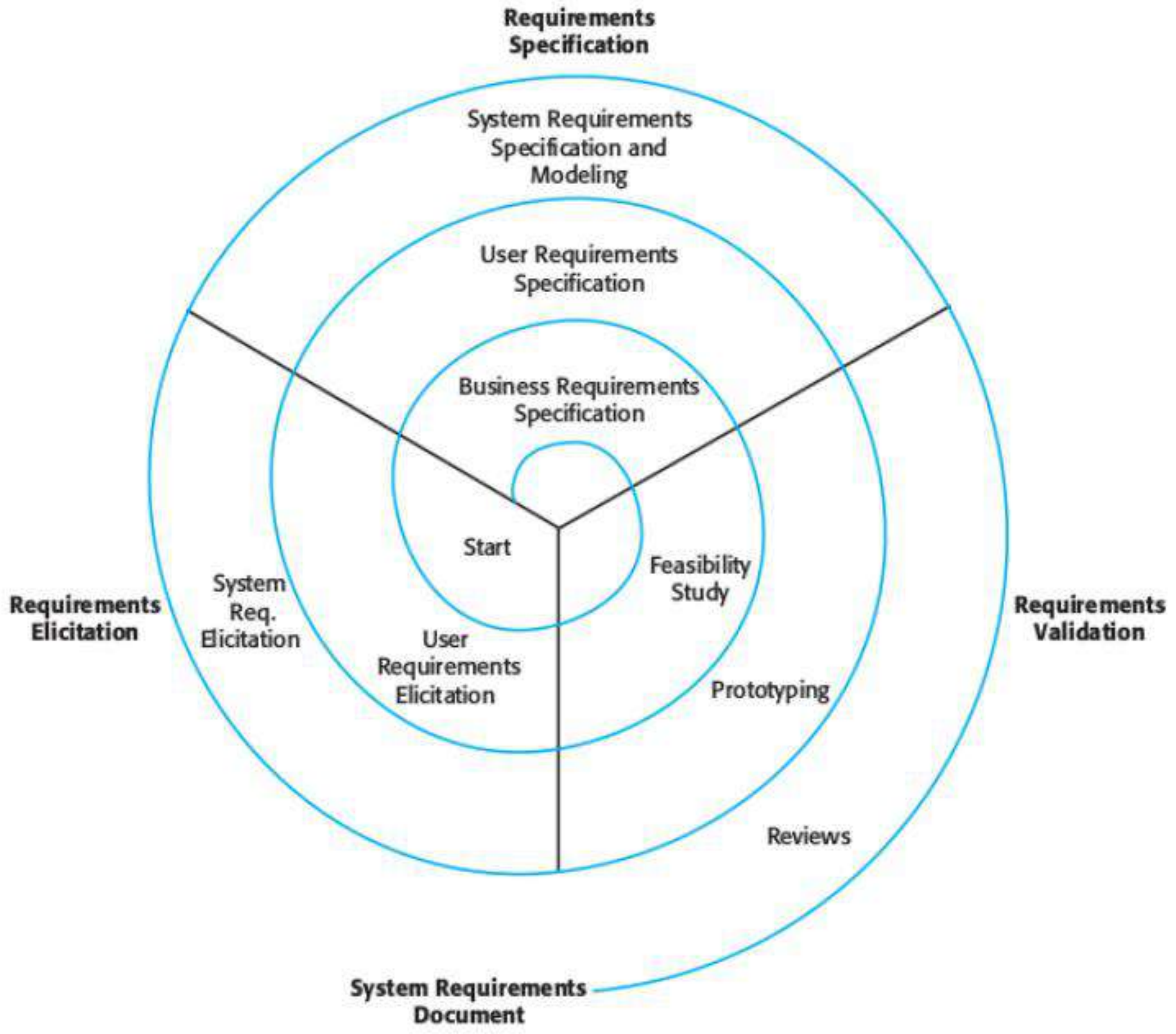
- Joint Application Design (IBM)
- The Method (Performance Resources)
- JAD stands for Joint Application Development.
- Sit down with the client and design a paper UI that they can see what the application will look like and behave like.
- Give the user a chance to work through common scenarios and see if the application will work for them.
- Keep refining until the user feels the application is doing what they want it to do.
- As you get functionality implemented, bring the user in and have them work through those
- scenarios and see if it still works.
- If they want a change, have a solid estimate of how long the change will add to the schedule and how much it will cost.

System stakeholders

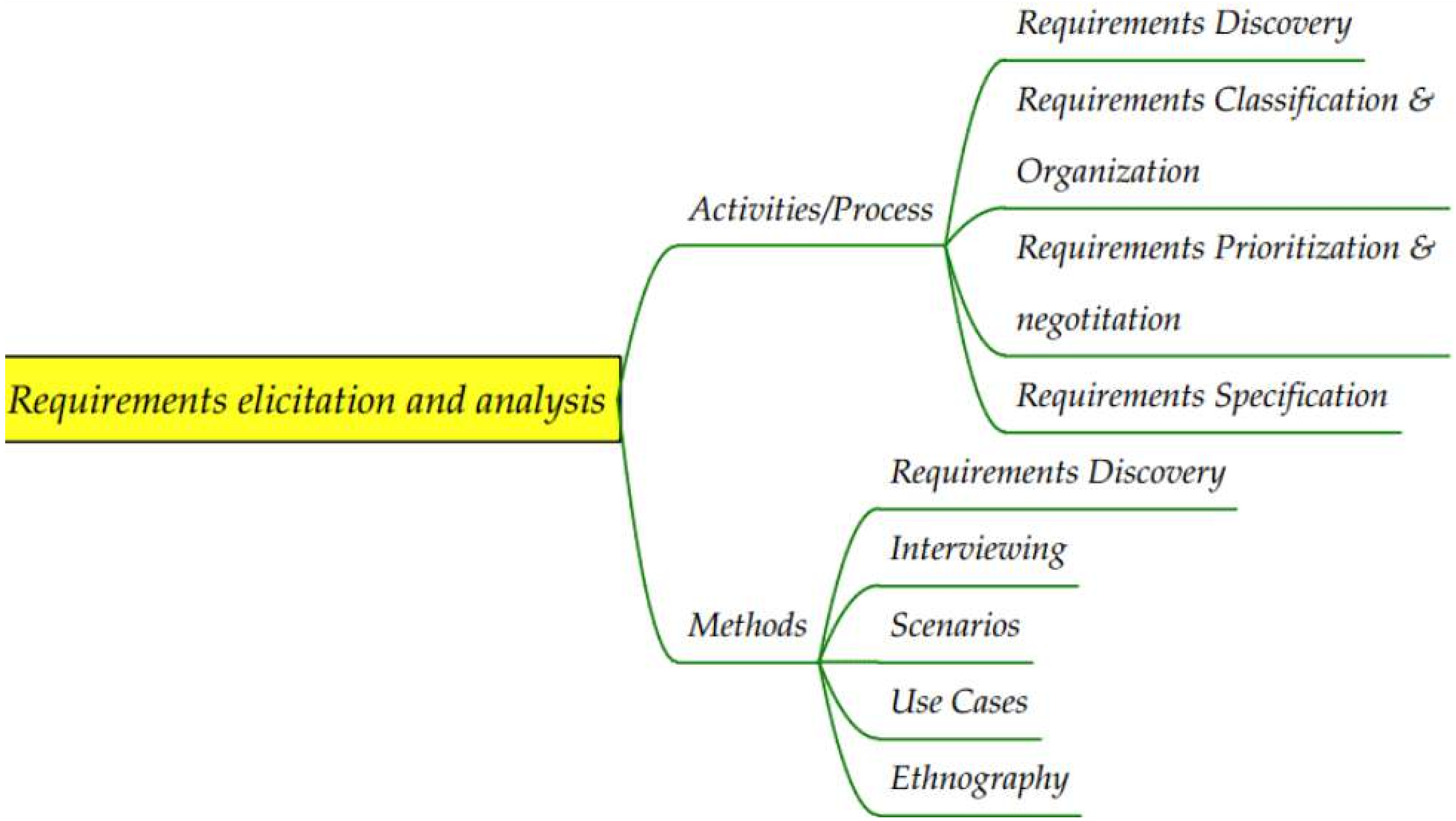
- Any person or organization who is affected by the system in some way and so who has a legitimate interest
- Stakeholder types
- End users
- System managers
- System owners
- External stakeholders

Requirements engineering processes

- there are a number of generic activities common to all processes
- Requirements elicitation;
- Requirements analysis;
- Requirements validation;
- Requirements management

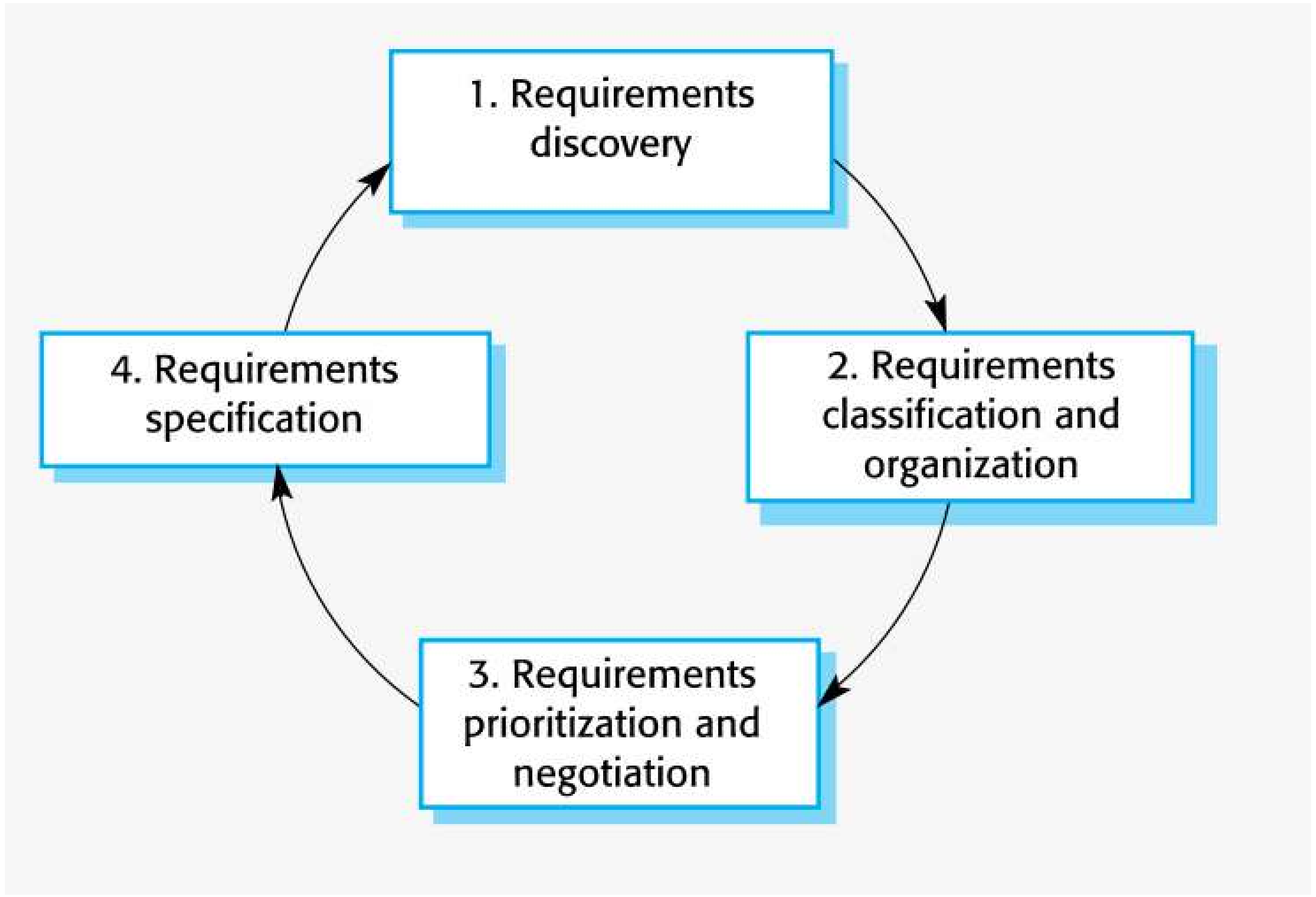

Requirements Specification
System Requirements Specification and Modeling
User Requirements Specification
Business Requirements Specification
Start
Feasibility Study
Requirements Elicitation
System Req. Elicitation
User Requirements Elicitation
Prototyping
Requirements Validation
Reviews
System Requirements Document

Requirement Elicitation and Analysis


Requirements elicitation and analysis
Activities/Process
Requirements Discovery
Requirements Classification & Organization
Requirements Prioritization & negotitation
Requirements Specification
Methods
Requirements Discovery
Interviewing
Scenarios
Use Cases
Ethnography

Requirement Elicitation

- Requirements elicitation is a requirements discovery process.
- Software engineers work with a range of system stakeholders to find out
- about the application domain, the services that the system should provide,
- the required system performance, hardware constraints, other systems, etc.
- Stages include:
- Requirements discovery,
- Requirements classification and organization,
- Requirements prioritization and negotiation,
- Requirements specification.


1. Requirements discovery
2. Requirements classification and organization
3. Requirements prioritization and negotiation
4. Requirements specification

Requirements discovery

- The process of gathering information about the required and existing systems and distilling the user and system requirements from this information.
- Interaction is with system stakeholders from managers to external regulators.
- Systems normally have a range of stakeholders.
- Requirement discovery methods:
- Interview
- Questionnaire
- survey

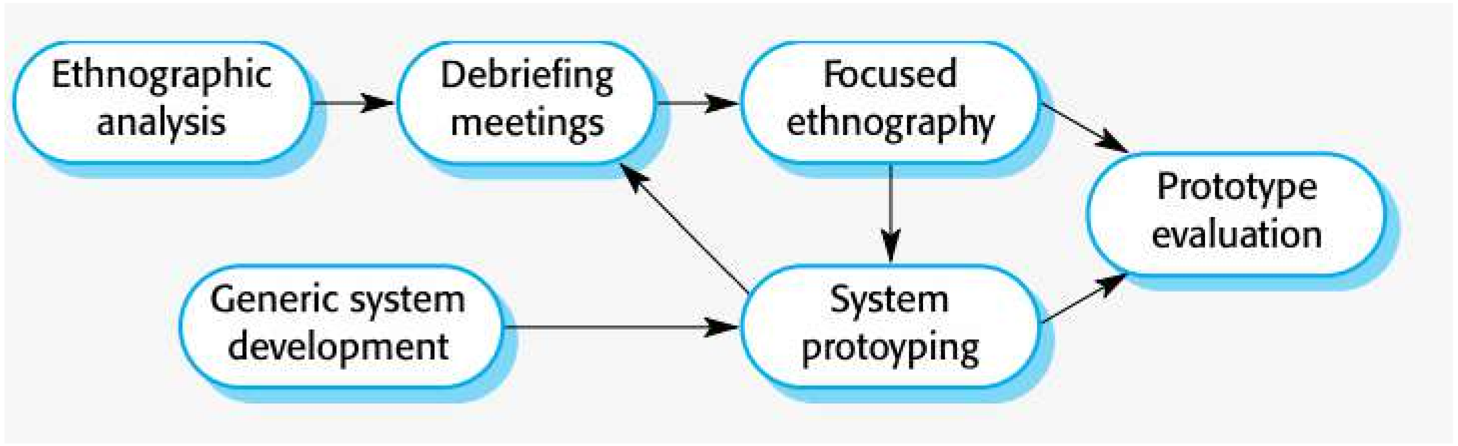
Interviewing

- Types of interview
- Closed interviews based on per-determined list of questions
- Open interviews where various issues are explored with stakeholders.
- Effective interviewing
- Be open-minded
- avoid per-conceived ideas about the requirements
- willing to listen to stakeholders.
- Prompt the user to talk by giving them some requirement idea.

Ethnography

- A social scientist spends a considerable time observing and analyzing how people actually work.
- Social and organizational factors of importance may be observed.
- Ethnographic studies have shown that work is usually richer and more complex than suggested by simple system models.
- Combines ethnography with prototyping
- Prototype development results in unanswered questions which focus the ethnographic analysis.
- The problem with ethnography is that it studies existing practices
- which may have some historical basis which is no longer relevant

Ethnography and prototyping for requirements analysis



Stories and scenarios

- Scenarios and user stories are real-life examples of how a system can be used.
- Stories and scenarios are a description of how a system may be used for a particular task.
- Because they are based on a practical situation, stakeholders can relate to them and can comment on their situation with respect to the story.

Scenarios

- A structured form of user story

Scenarios should include

- A description of the starting situation;
- A description of the normal flow of events;
- A description of what can go wrong;
- Information about other concurrent activities;
- A description of the state when the scenario finished.

Use cases

- Use-cases are a kind of scenario that are included in the UML.
- Use cases identify the actors in an interaction and which describe the interaction itself.
- A set of use cases should describe all possible interactions with the system.
- UML sequence diagrams may be used to add detail to use-cases by showing the sequence of event processing in the system.

Models in OODM

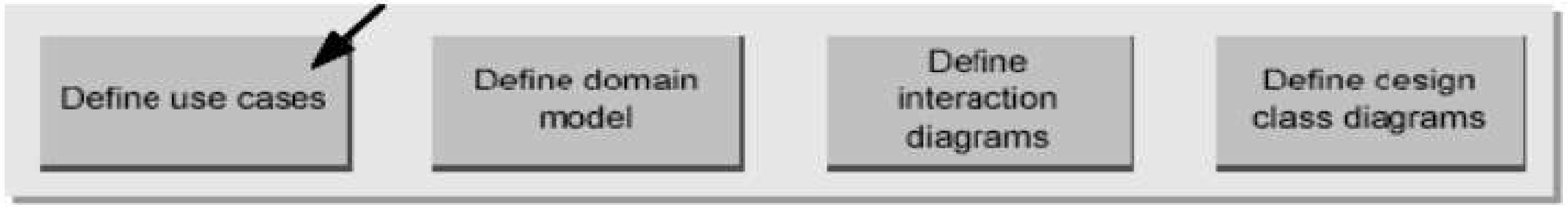
- Define use case: Define actors and goals(OOA)
- Define domain model: Identify concepts and requirements(OOD)
- Interaction diagram: Sequence diagram and communication diagram
- Class diagram:-Implementation
- Design to code-Testing and coding

Defining models

- A birds eye view steps and diagrams involved in a dice game in which a player rolls 2 die. If the total is seven they win otherwise they lose.

Define Use Cases

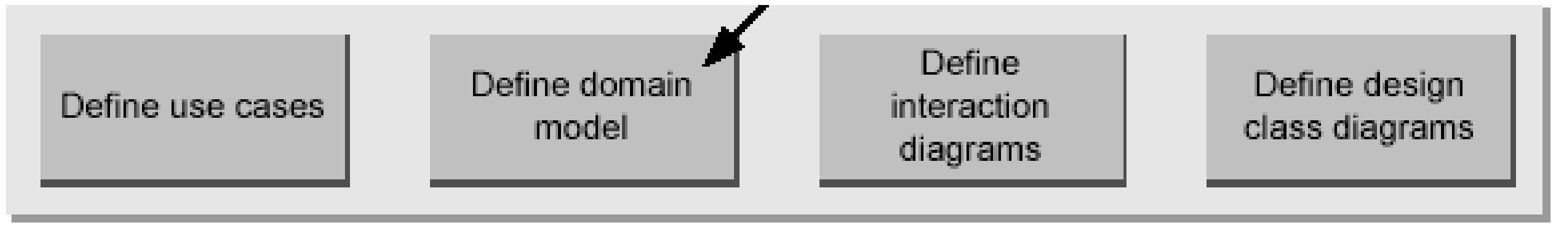
- Requirements analysis may include a description of related domain processes; these can be written as use cases.
- Use cases are not an object-oriented artifact—they are simply written stories.



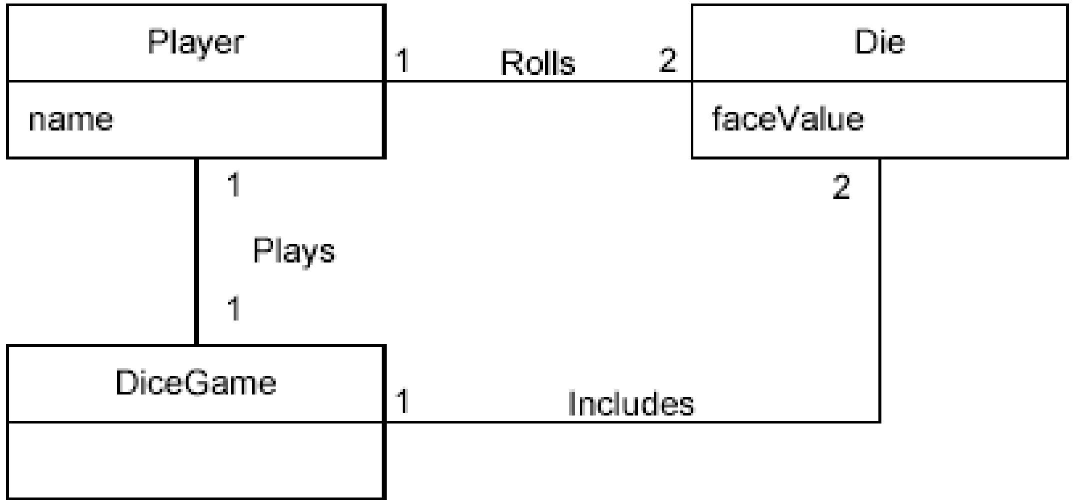
- However, they are a popular tool in requirements analysis and are an important part of the Unified Process. For example, here is a brief version of the Play aDice Game use case:
- Play a Dice Game: A player picks up and rolls the dice. If the dice face value total seven, they win; otherwise, they lose.

Define a domain model

- Object-oriented analysis is concerned with creating a description of the domain from the perspective of classification by objects. A decomposition of the domain involves an identification of the concepts, attributes, and associations that are considered noteworthy. The result can be expressed in a domain model, which is illustrated in a set of diagrams that show domain concepts or objects.

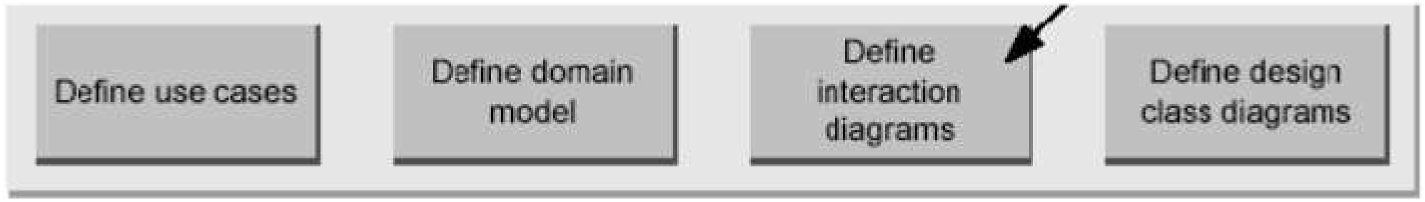


Note that a domain model is not a description of software objects; it is a visualization of concepts in the real-world domain.

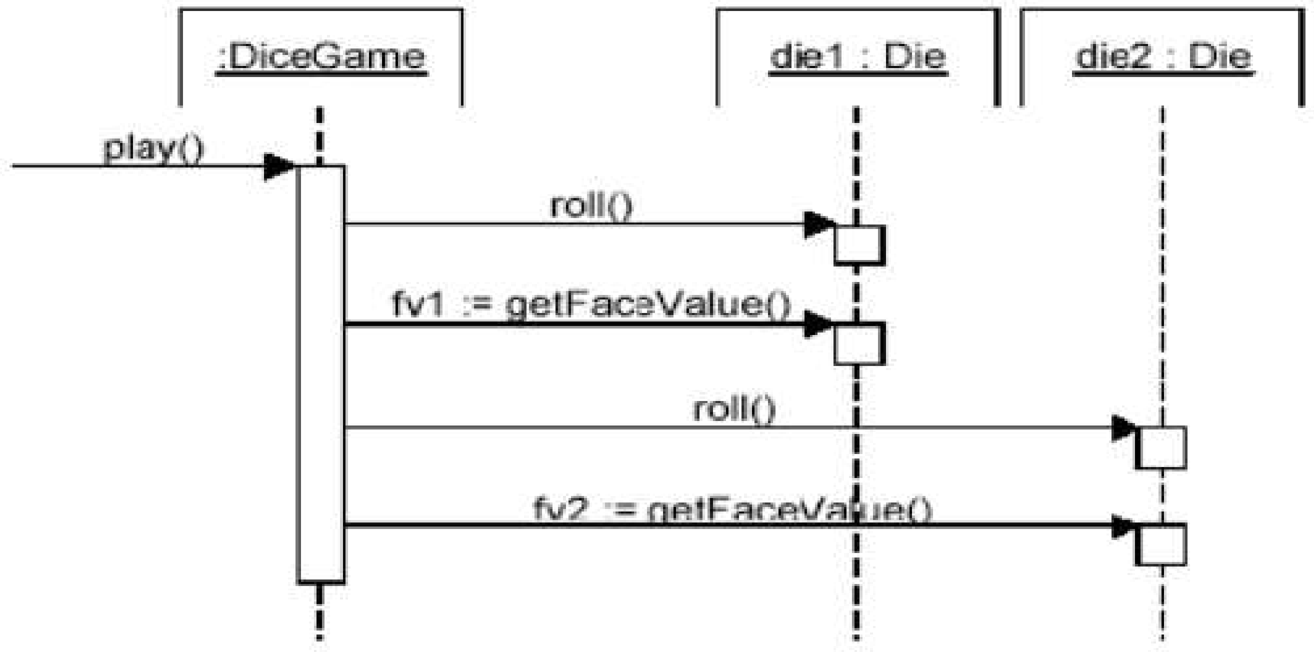


Define Interaction Diagrams

- Object-oriented design is concerned with defining software objects and their collaborations.
- A common notation to illustrate these collaborations is the interaction diagram. It shows the flow of messages between software objects, and thus the invocation of methods.



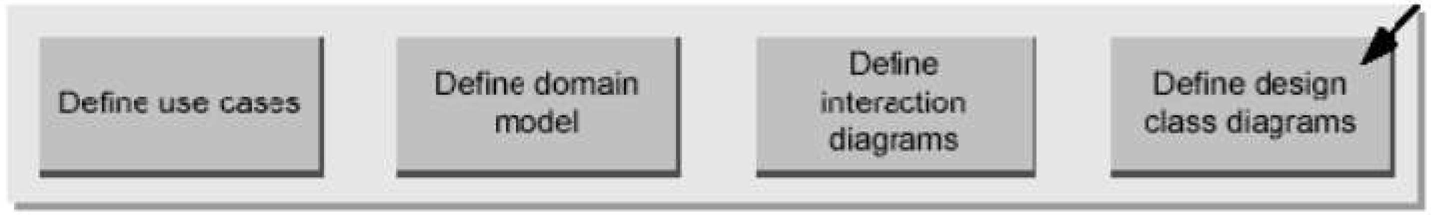
- For example, assume that a software implementation of the dice game is desired. The interaction diagram in figure illustrates the essential step of playing, by sending messages to instances of the DiceGame and Die classes.

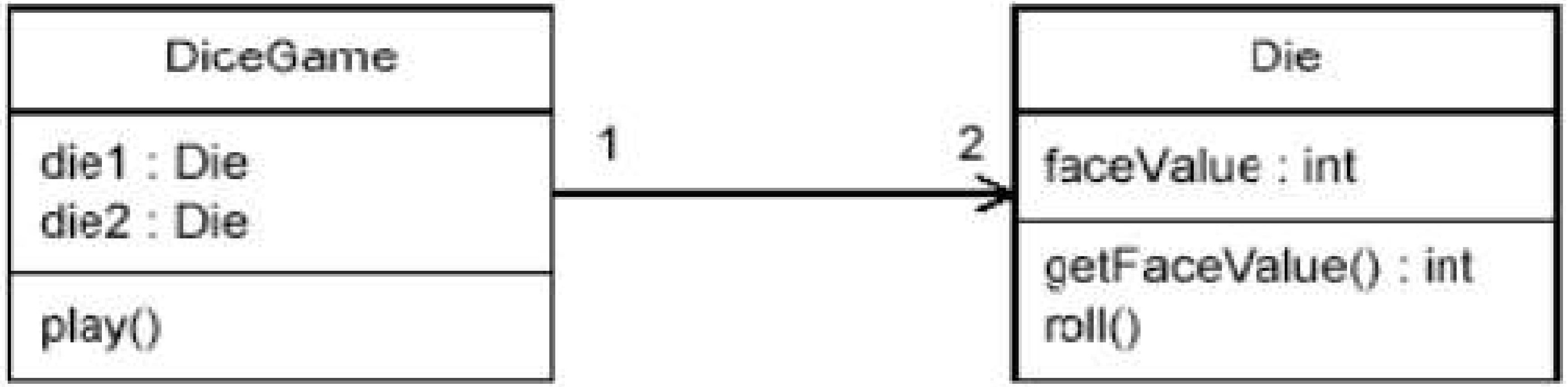


- Although in the real world a player rolls the dice, in the software design the DiceGame object “rolls” the dice (that is, sends messages to Die objects). Software object designs and programs do take some inspiration from real-world domains, but they are not direct models or simulations of the real world.

Define Design Class Diagrams

- In addition to a dynamic view of collaborating objects shown in interaction diagrams, it is useful to create a static view of the class definitions with a design class diagram. This illustrates the attributes and methods of the classes.
- For example, in the dice game, an inspection of the interaction diagram leads to the partial design class diagram . Since a play message is sent to a DiceGame object, the DiceGame class requires a play method, while class Die requires a roll and getFaceValue method.





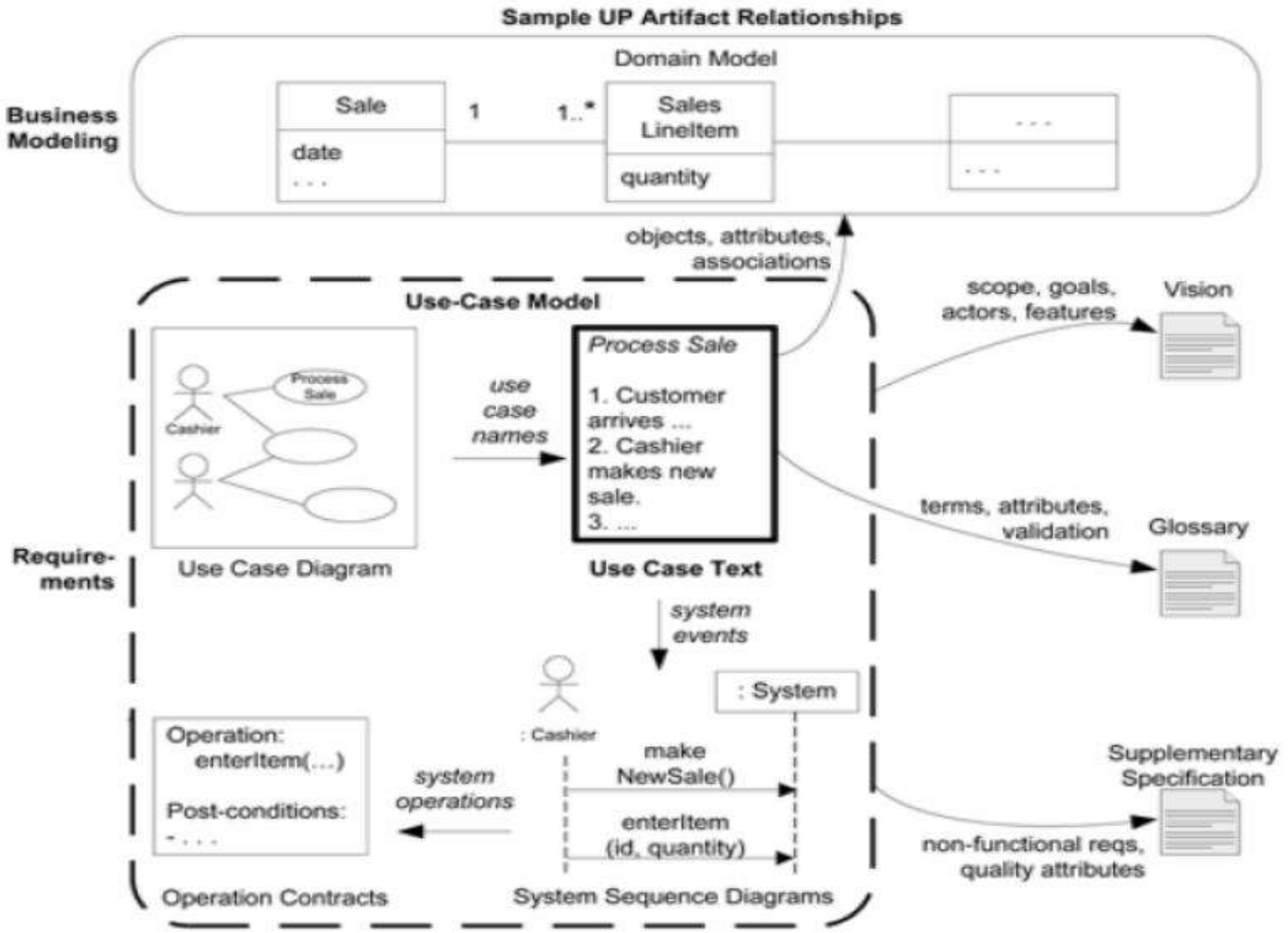
- In contrast to the domain model, this diagram does not illustrate real-world concepts; rather, it shows software classes.

Case study

- The NEXT Generation POS system
- Distributed Multimedia Applications
- Design Methodology for Development of Web Applications

Use Cases: Describing Processes

- Definition: - It is a narrative document that describes the sequence of events of an actor (an external agent) using a system to complete a process.
- Writing use cases, stories of using a system, is an excellent technique to understand and describe requirements. Informally, they are stories of using a system to meet goals.
- The essence is discovering and recording functional requirements by writing stories of using a system to help fulfill various stakeholder goals
- “Use cases are narrative description of domain processes”
- Use cases are dependent on having at least partial understanding of the system, ideally expressed in a requirements specification document.



Definition: - It is a narrative document that describes the sequence of events of an actor (an external agent) using a system to complete a process.

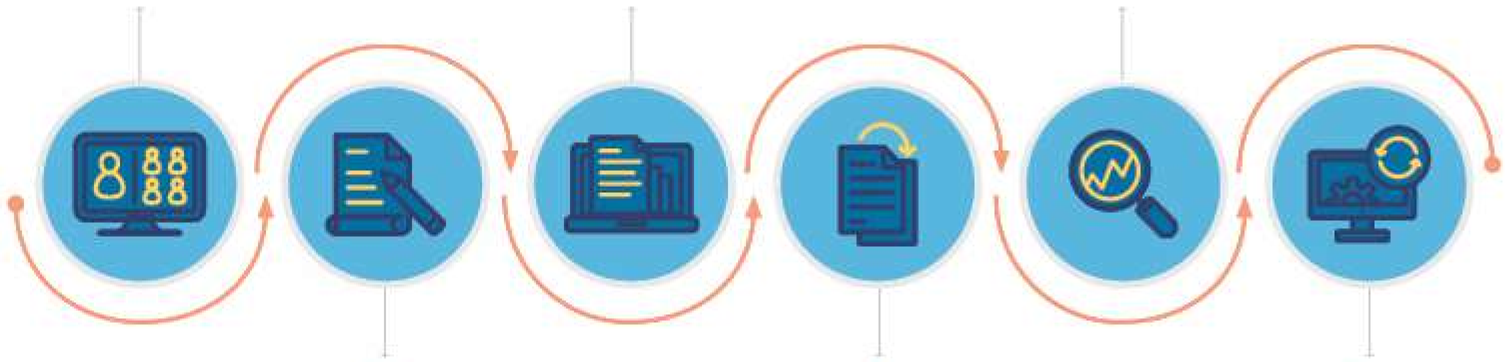
A use case should display the following characteristics:

- Organizes functional requirements.
- Models the goals of system/actor interactions.
- Records paths -- called scenarios -- from trigger events to goals.
- Describes one main flow of events and various alternate flows.
- Multi-level, so that one use case can use the functionality of another one.

The writing process includes:

- Identifying all system users and creating a profile for each one. This includes every role played by a user who interacts with the system.
- Selecting one user and defining their goal -- or what the user hopes to accomplish by interacting with the system. Each of these goals becomes a use case.
- Describing the course taken for each use case through the system to reach that goal.
- Considering every alternate course of events and extending use cases -- or the different courses that can be taken to reach the goal.
- Identifying commonalities in journeys to create common course use cases and write descriptions of each.
- Repeating steps two through five for all other system users.

Use case methodology



1
Identify all system users and create a profile for each one

2
Select one user and define their goal

3
Describe the course taken through the system to reach that goal

4
Consider every alternate course of events and extend use case

5
Identify all commonalities in user journeys to create a common course use case

6
Repeat steps 2-5 for all other system users

What Use Cases Include	What Use Cases Do NOT Include
• Who is using the website • What the user want to do • The user's goal • The steps the user takes to accomplish a particular task • How the website should respond to an action	• Implementation-specific language • Details about the user interfaces or screens.

The reasons why an organization would want to use case include:

- Represent the goals of systems and users.
- Specify the context a system should be viewed in.
- Specify system requirements.
- Provide a model for the flow of events when it comes to user interactions.
- Provide an outside view of a system.
- Show’s external and internal influences on a system

Three use case formats

- **Brief (High Level):-**terse one-paragraph summary, usually of the main success scenario.

→It describes a process very briefly, usually in two or three sentences.

→It is useful to create this type of use case during the initial requirements and project scoping in order quickly understand the degree of complexity and functionally in a system.

They are very terse and vague on design decisions.

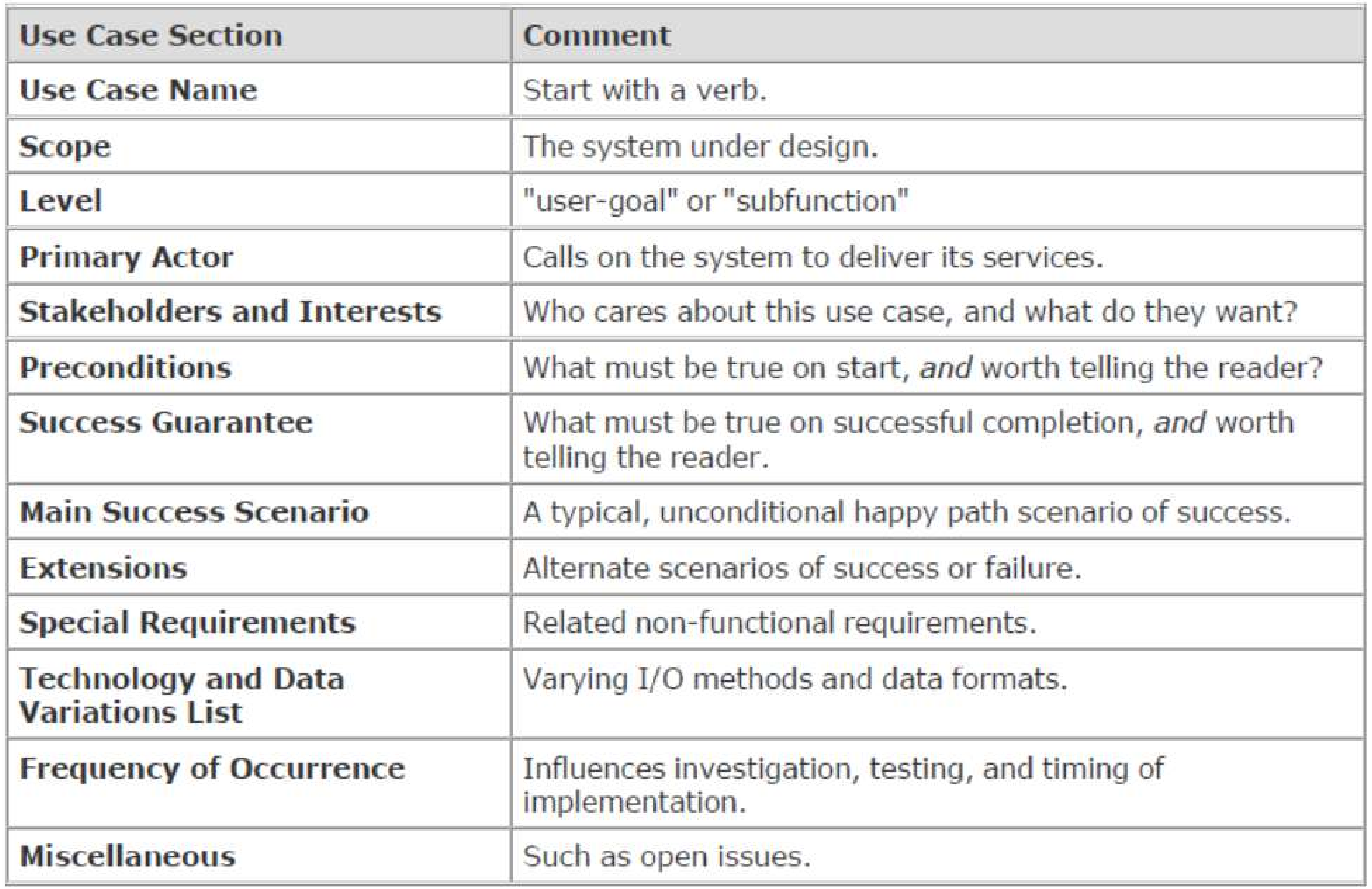
Process Sale: A customer arrives at a checkout with items to purchase. The cashier uses the POS system to record each purchased item. The system presents a running total and line-item details. The customer enters payment information, which the system validates and records. The system updates inventory. The customer receives a receipt from the system and then leaves with the items It is useful to start with high level use cases to quickly obtain some understanding of overall major processes

2. Casual:

- informal paragraph format. Multiple paragraphs that cover various scenarios.
- **Handle Returns**
- Main Success Scenario: A customer arrives at a checkout with items to return. The cashier uses the POS(Point of sale) system to record each returned item ...
- **Alternate Scenarios:**
- If the credit authorization is reject, inform the customer and ask for an alternate payment method.
- If the item identifier is not found in the system, notify the Cashier and suggest manual entry of the identifier code (perhaps it is corrupted).
- If the system detects failure to communicate with the external tax calculator system, ...

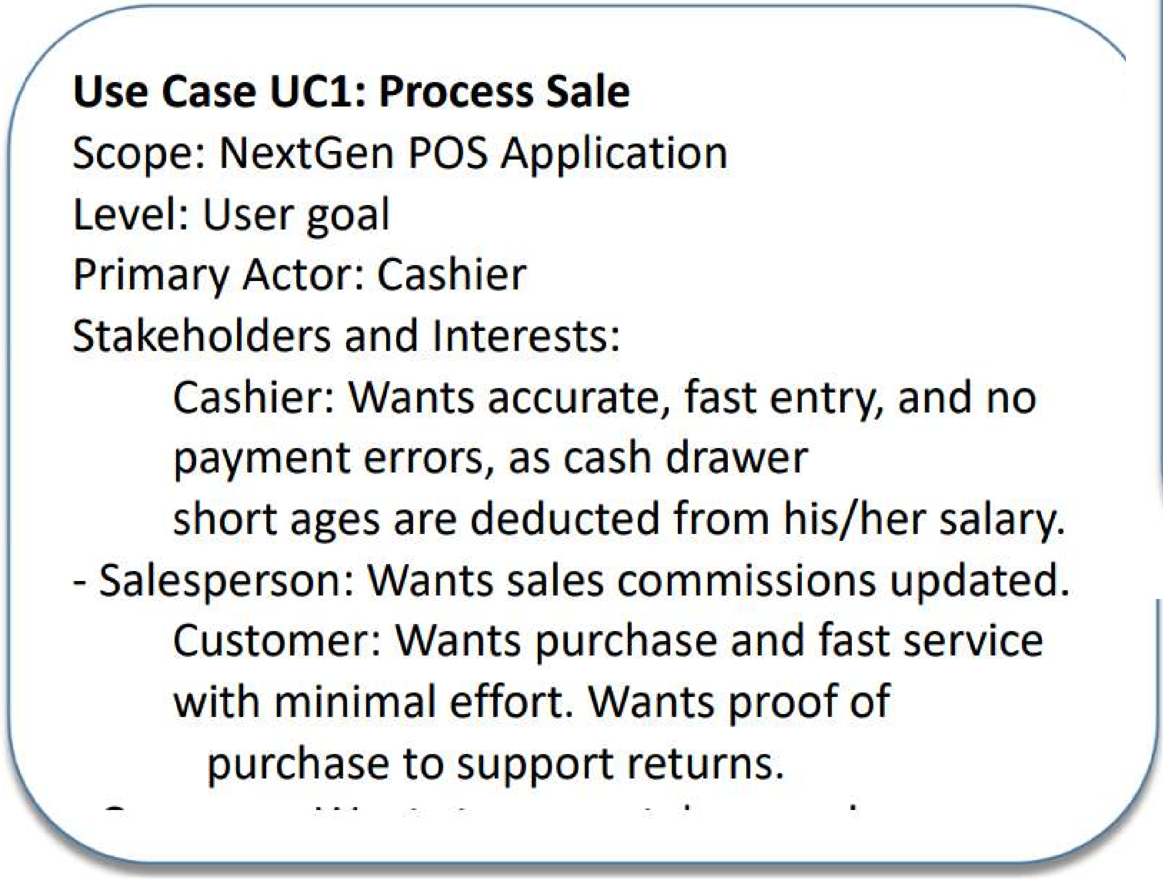
- **Fully Dressed**:
- the most elaborate. All steps and variations are written in detail, and there are supporting sections, such as preconditions and success guarantees.
- These detailed use cases are written after many uses cases have been identified and written in a brief format

Template



Use Case Section	Comment
Use Case Name	Start with a verb.
Scope	The system under design.
Level	"user-goal" or "subfunction"
Primary Actor	Calls on the system to deliver its services.
Stakeholders and Interests	Who cares about this use case, and what do they want?
Preconditions	What must be true on start, *and* worth telling the reader?
Success Guarantee	What must be true on successful completion, *and* worth telling the reader.
Main Success Scenario	A typical, unconditional happy path scenario of success.
Extensions	Alternate scenarios of success or failure.
Special Requirements	Related non-functional requirements.
Technology and Data Variations List	Varying I/O methods and data formats.
Frequency of Occurrence	Influences investigation, testing, and timing of implementation.
Miscellaneous	Such as open issues.

Example

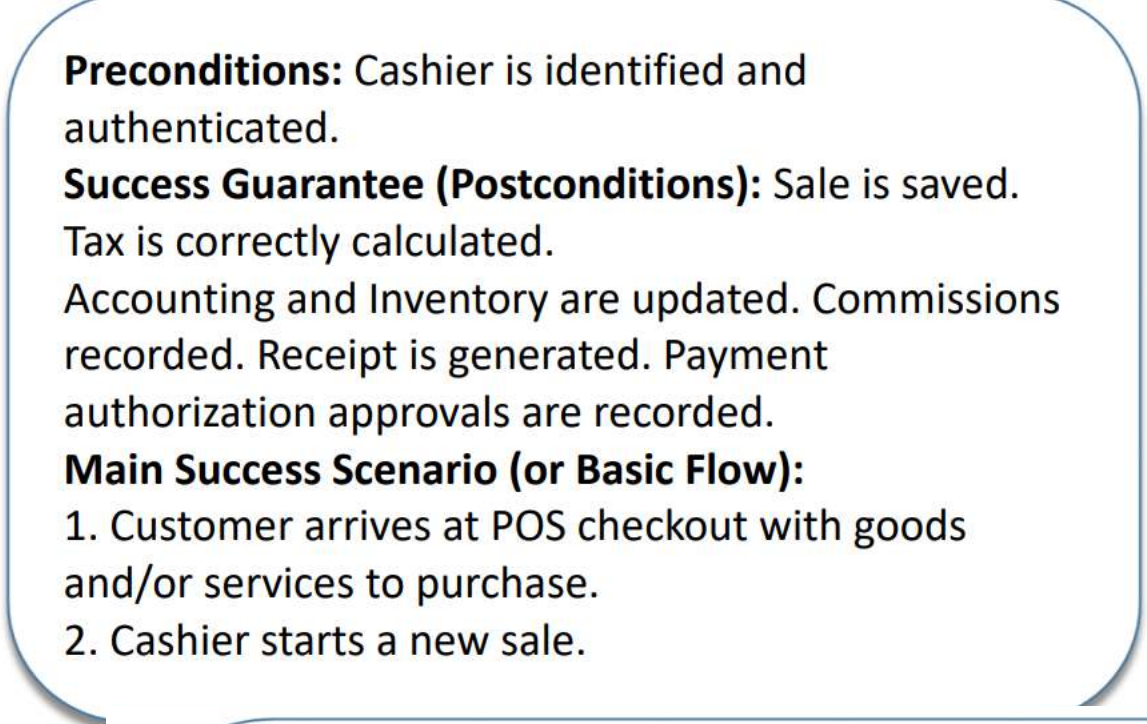


Use Case UC1: Process Sale
Scope: NextGen POS Application
Level: User goal
Primary Actor: Cashier
Stakeholders and Interests:

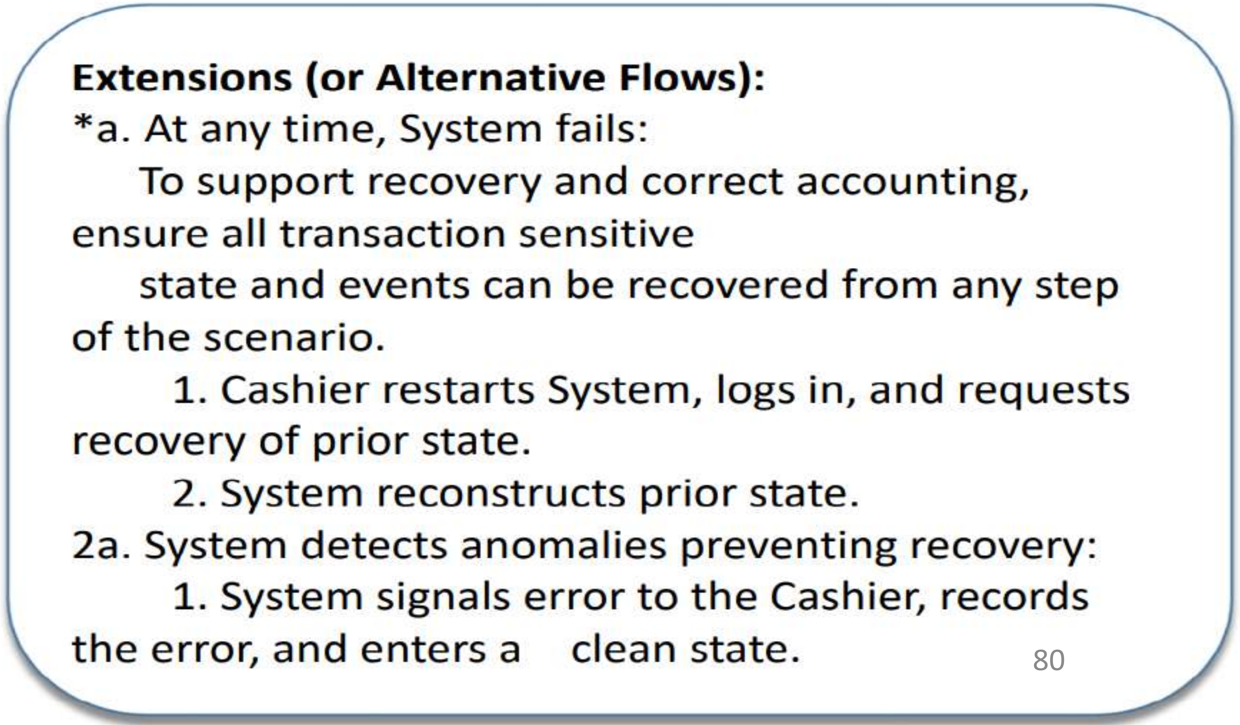
Cashier: Wants accurate, fast entry, and no payment errors, as cash drawer short ages are deducted from his/her salary.

- Salesperson: Wants sales commissions updated.

Customer: Wants purchase and fast service with minimal effort. Wants proof of purchase to support returns.



Preconditions: Cashier is identified and authenticated.
Success Guarantee (Postconditions): Sale is saved. Tax is correctly calculated.
Accounting and Inventory are updated. Commissions recorded. Receipt is generated. Payment authorization approvals are recorded.
Main Success Scenario (or Basic Flow):
1. Customer arrives at POS checkout with goods and/or services to purchase.
2. Cashier starts a new sale.



Extensions (or Alternative Flows):
*a. At any time, System fails:
To support recovery and correct accounting, ensure all transaction sensitive
state and events can be recovered from any step of the scenario.
1. Cashier restarts System, logs in, and requests recovery of prior state.
2. System reconstructs prior state.
2a. System detects anomalies preventing recovery:
1. System signals error to the Cashier, records the error, and enters a clean state.

Frequency of Occurrence:

- Could be nearly continuous.
- Open Issues:

- What are the tax law variations?

- Explore the remote service recovery issue.

- What customization is needed for different businesses?

- Must a cashier take their cash drawer when they log out?

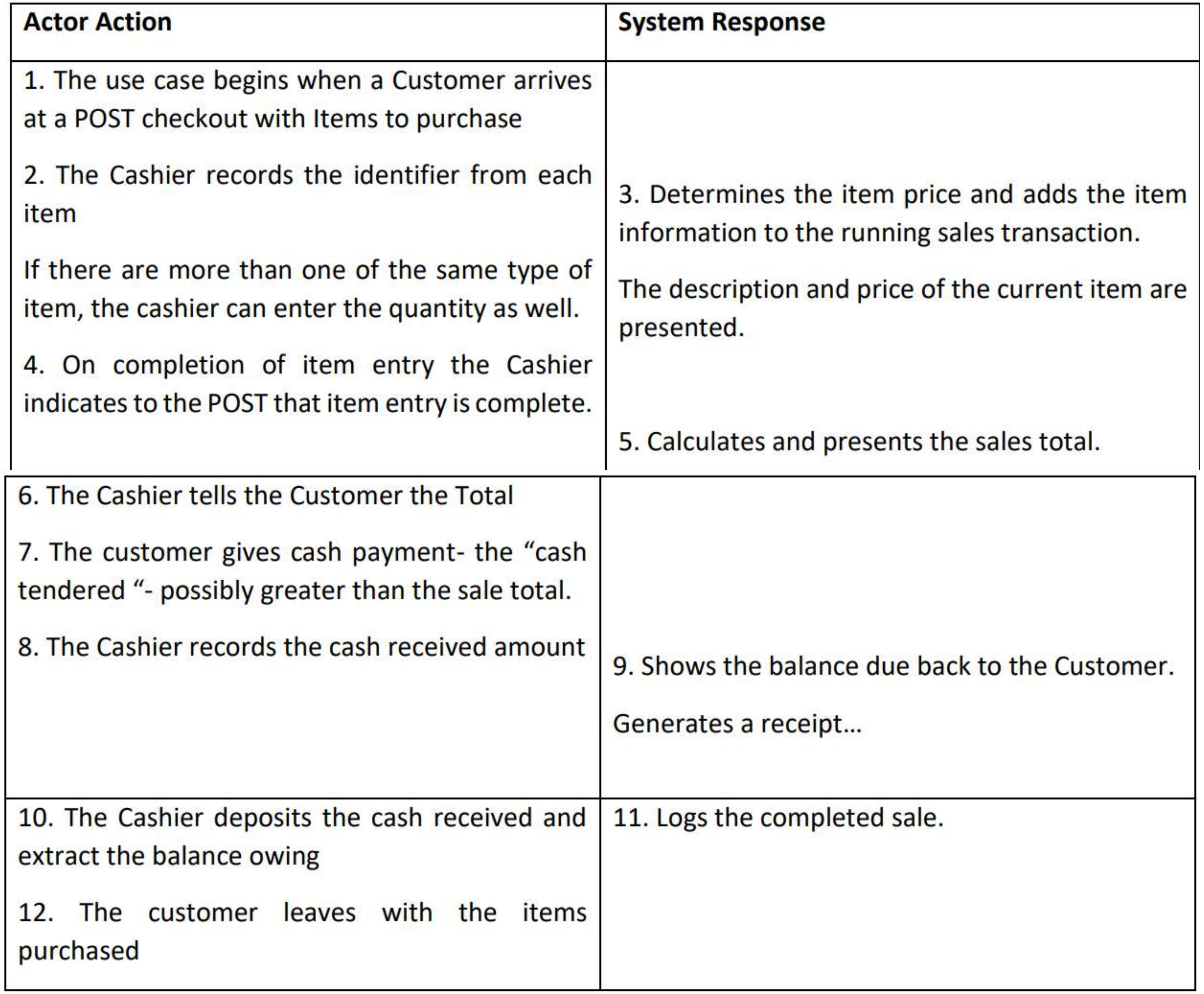
- Can the customer directly use the card reader, or does the cashier have to do it?

- **Preconditions** state what must always be true before beginning a scenario in the use case.
- A precondition implies a scenario of another use case that has successfully completed, such as
- logging in, or "cashier is identified and authenticated”.
- **Success guarantees (or postconditions)** state what must be true on successful completion of
- the use case. either the main success scenario or some alternate path. The guarantee should meet the needs of all stakeholders.
- Sale is saved. Tax is correctly calculated.
- Accounting and Inventory are updated. Commissions recorded. Receipt is generated.
- **Extensions (or Alternate Flows)** indicate all the other scenarios or branches, both success and
- failure. They are also known as "Alternative Flows.“
- **Special Requirements**
- If a non-functional requirement, quality attribute, or constraint relates specifically to a use case, record it with the use case. These include qualities such as performance, reliability, and usability, and design constraints (often in I/O devices) that have been mandated or considered likely.

A Two Column Format : Typical Course of Events

Actor Action	**System Response**
Numbered action of the actor	Numbered description of system response

Alternative Course of events describes important alternatives or exceptions that may arise with respect to typical course.

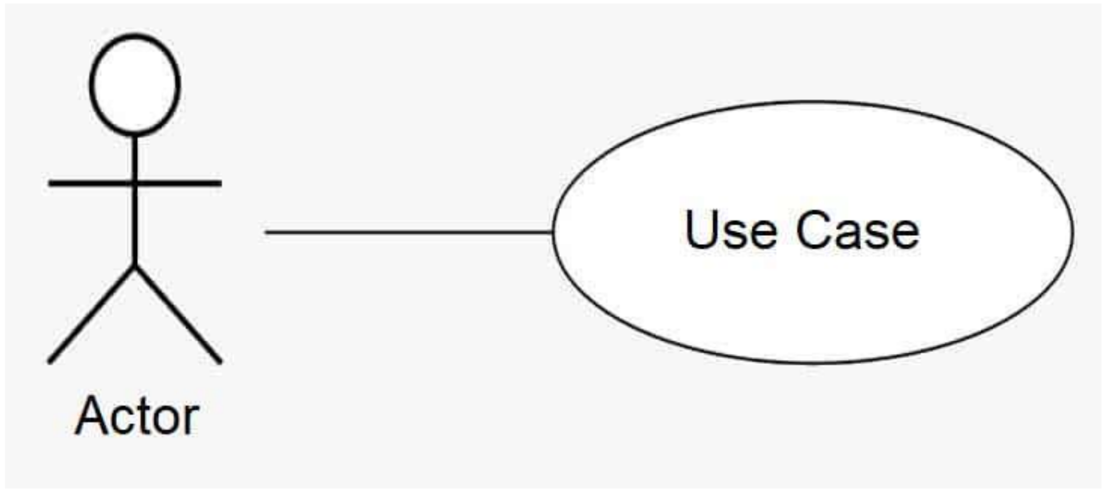


Actor Action	System Response
1. The use case begins when a Customer arrives at a POST checkout with Items to purchase 2. The Cashier records the identifier from each item If there are more than one of the same type of item, the cashier can enter the quantity as well. 4. On completion of item entry the Cashier indicates to the POST that item entry is complete.	3. Determines the item price and adds the item information to the running sales transaction. The description and price of the current item are presented. 5. Calculates and presents the sales total.
6. The Cashier tells the Customer the Total 7. The customer gives cash payment- the “cash tendered “- possibly greater than the sale total. 8. The Cashier records the cash received amount	9. Shows the balance due back to the Customer. Generates a receipt…
10. The Cashier deposits the cash received and extract the balance owing 12. The customer leaves with the items purchased	11. Logs the completed sale.

Alternative Course
Line 2: Invalid identifier entered. Indicate error
Line 7: Customer didn’t have enough cash. Cancel sales transaction

Actor

- It is an entity that is external to the system, who in some way participates in the story of Use Case.
- An Actor typically stimulates the system with input events, or receives something form it. Actors are represented by the role they play in the Use Case.
- e.g. “User”, “Stakehoder” or “External System”.
- the **creator of an action** (such as the initiator of an application case)
- but also the **recipient of an expected result**.



Actors can include
-roles that people play
-Computer system
-Electrical and Mechanical devices

Types of Actors

- **Primary actors** have user goals fulfilled through using services of system under discussion. e.g. cashier .These are identified to find user goals that drive use cases
- **Supporting actors** provide service or information to the system. e.g. automated payment authorization. These are identified to clarify external interfaces and protocols.
- **Offstage actor** has interest in the behavior of the use case . e.g. government tax agency. These are identified to ensure all interests are identified and satisfied

Scenario

- A scenario is a specific sequence of actions and interactions between actors and the system under discussion; it is also called a use case instance.
- A use case is a collection of related success and failure scenarios that describes an actor using a system to support a goal
- Use cases are functional or behavioral requirements that indicate what the system will do

Common Mistakes with the Use Cases Remember: A use case is a relatively large end to end process description that typically includes many steps or transactions; it is not normally an individual step or activity in a process. e.g printing a receipt; is not a use case but a step in a use case called Buy_Item

Finding Primary Actors, Goals, and Use Cases

1. Choose the system boundary.

- Is it just a software application, the hardware and application as a unit, that plus a person using it, or an entire organization?

2. Identify the primary actors.

- Those that have user goals fulfilled through using services of the system.

3. For each, identify their user goals.

- Raise them to the highest user goal level that satisfies the EBP guideline.

4. Define use cases that satisfy user goals; name them according to their goal.

- Usually, user goal-level use cases will be one-to-one with user goals.

System and their boundaries:

- A use case describes interaction with the system. Typical boundaries includes

- Hardware/software boundary of a device or computer system.

- Department of an organization.

- Entire organization.

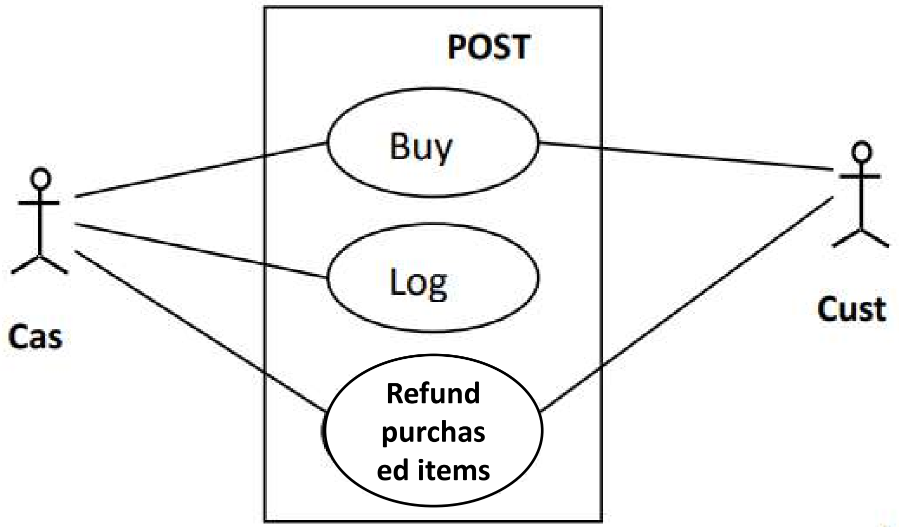
- The system boundary is important as it identifies:

- what is external versus internal.

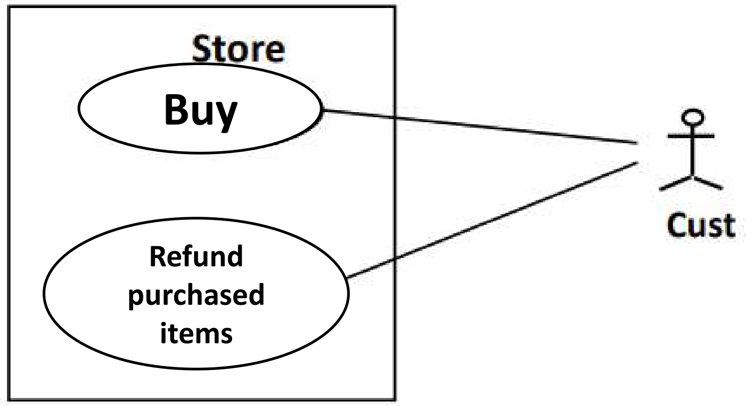
- the responsibilities of the system

Scenario one

- If we choose the entire store or business as the system then the only customer is an actor not the cashier because the cashier acts as a resource within the business system.



Use case and actors when the POST



Use case and actors when the Store is the

Scenario two

- If we choose the point of sale hardware and software as the system both cashier as well as customer may be treated as actors.

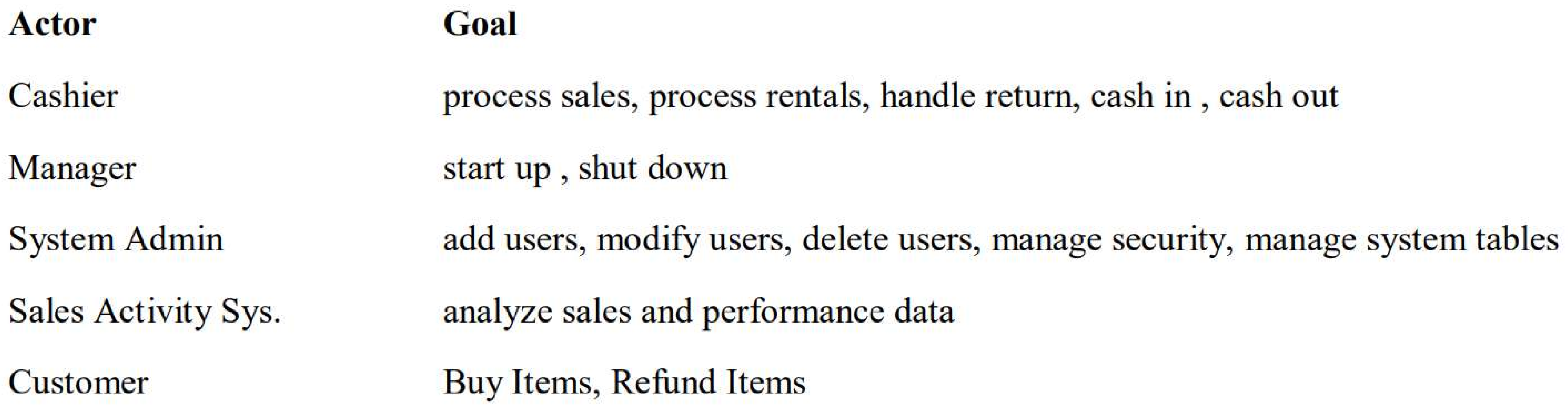
METHODS OF IDENTIFYING THE USE CASES

- Method 1 (Actor based):Identifying the actor related to a system or organization and then for each actor, identifying the processes they initiate or participate in.
- Using actor goal list

A recommended procedure:

1. Find the user goals.

2. Define a use case for each.



Actor	**Goal**
Cashier	process sales, process rentals, handle return, cash in , cash out
Manager	start up , shut down
System Admin	add users, modify users, delete users, manage security, manage system tables
Sales Activity Sys.	analyze sales and performance data
Customer	Buy Items, Refund Items

Method 2 (External events based)

- Identifying the external events that a system must respond to and then relating the events to actor and use cases.

External Event	Actor	Goal/use case
Enter sale line item	Cashier	process a sale
Enter payment	Cashier or customer	process a sale

Naming Use Case and Domain Processes

- A use case describes a process (Business process) , it is a complete run of a scenario
- e.g. all of the events that occur from the time a customer enters the ATM station till the time he comes out with cash with one goal achieved that includes inserting card, validation etc.
- A process describes, from start to finish, a sequence of events, actions and transactions required to produce or complete something of value to an organization or actor Process
- e.g. Withdraw Cash form ATM, Order a Product
- A use case name starts with a verb. So naming a uses case based on goal:
- Goal: Process a sale   Use case: ProcessSale

Use Cases, System Functions and Traceability

- Identified system functions in requirements specification should be allocated to Use Cases
- Using Cross reference of the Use Cases, all functions that have been allocated should be verified, this helps in traceability between the artifacts

Essential versus Real use cases:

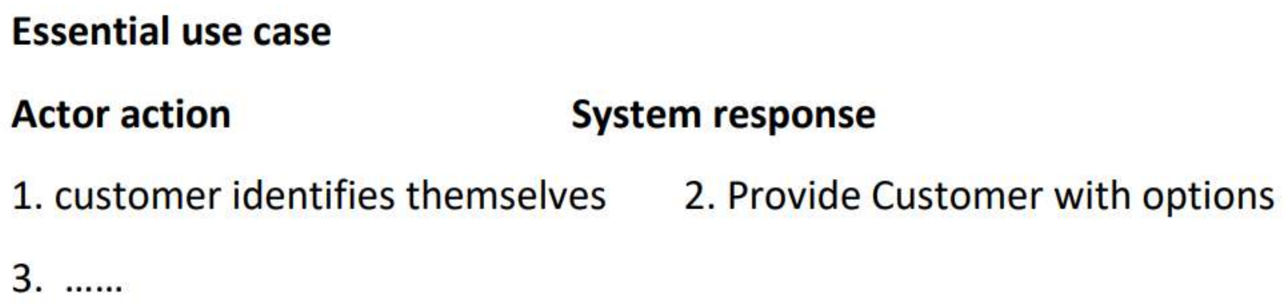
- Essential use cases

- created during early stage of eliciting the requirements

- independent of the design decisions

- very abstract

- High level use cases are always essential in nature due to their brevity (concise) and abstractions.
- It is desirable to create essential use cases during only requirements, elicitations in order to more fully understand the scope of the problem and the functions required.



Essential use case

Actor action	**System response**
1. customer identifies themselves	2. Provide Customer with options
3.	

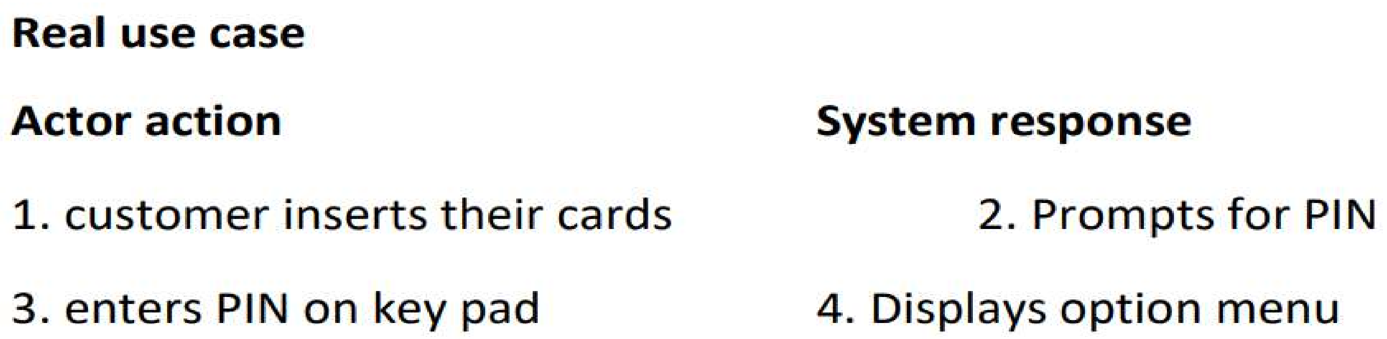
Real use cases

- Describe the functionality of the system in terms of its actual current design committed to specific input output technologies
- Developed only after the design decisions have been made

- Design artifacts

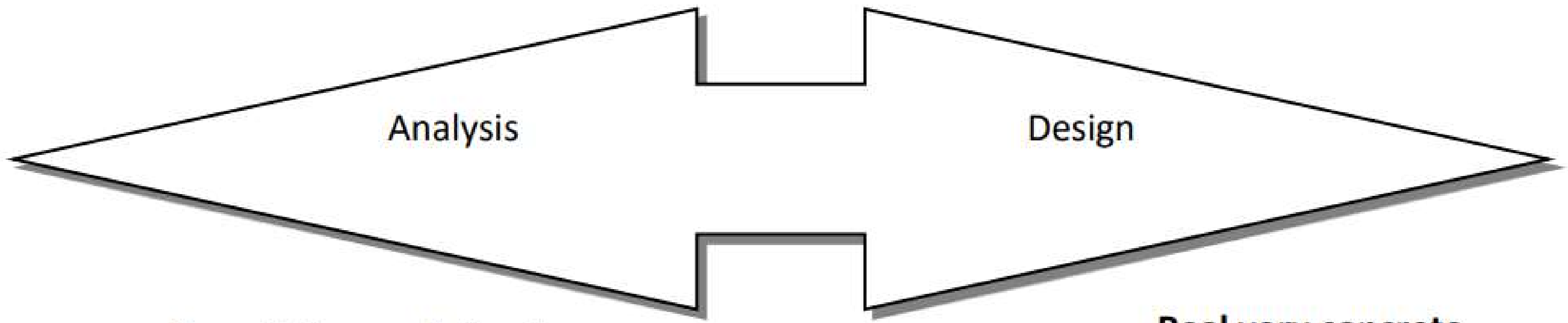
- very concrete

- This concretely describes the process in terms in its rare current design committed to specific
- input and output technologies and so on.



Real use case

Actor action	**System response**
1. customer inserts their cards	2. Prompts for PIN
3. enters PIN on key pad	4. Displays option menu



Essential very abstract

Real very concrete

Buy Items use case

Essential

Actor action	System response
1. Cashier **records identifier** for each item If there is more than one of the same The cashier enters quantity	2. Determine the item price and add the item info to the running sales transaction description price of the current item in item presented

Real

Actor action	System response
1. For each item cashier types **UPC** in the UPC input field of window then They **press "enter item" button** **with the mouse or keyboard**	2. Display item price and add the item information to the running sales transactions the description and price of current item are displayed

Use Cases within a Development Process

- Plan and Elaborate Phase steps
- After the System Functions have been listed, define the System boundary and then Identify Actors and the Use Cases
- Write down all the Use Cases in High Level Format and categorize them as Primary and Secondary or optional

Draw a Use Case diagram

- Relate Use Cases
- Write the most critical, influential and risky use cases in the expanded format to better understand and estimate the size and nature of the problem.
- Real use case should be deferred (delayed) as they involve design decisions
- Rank Use cases

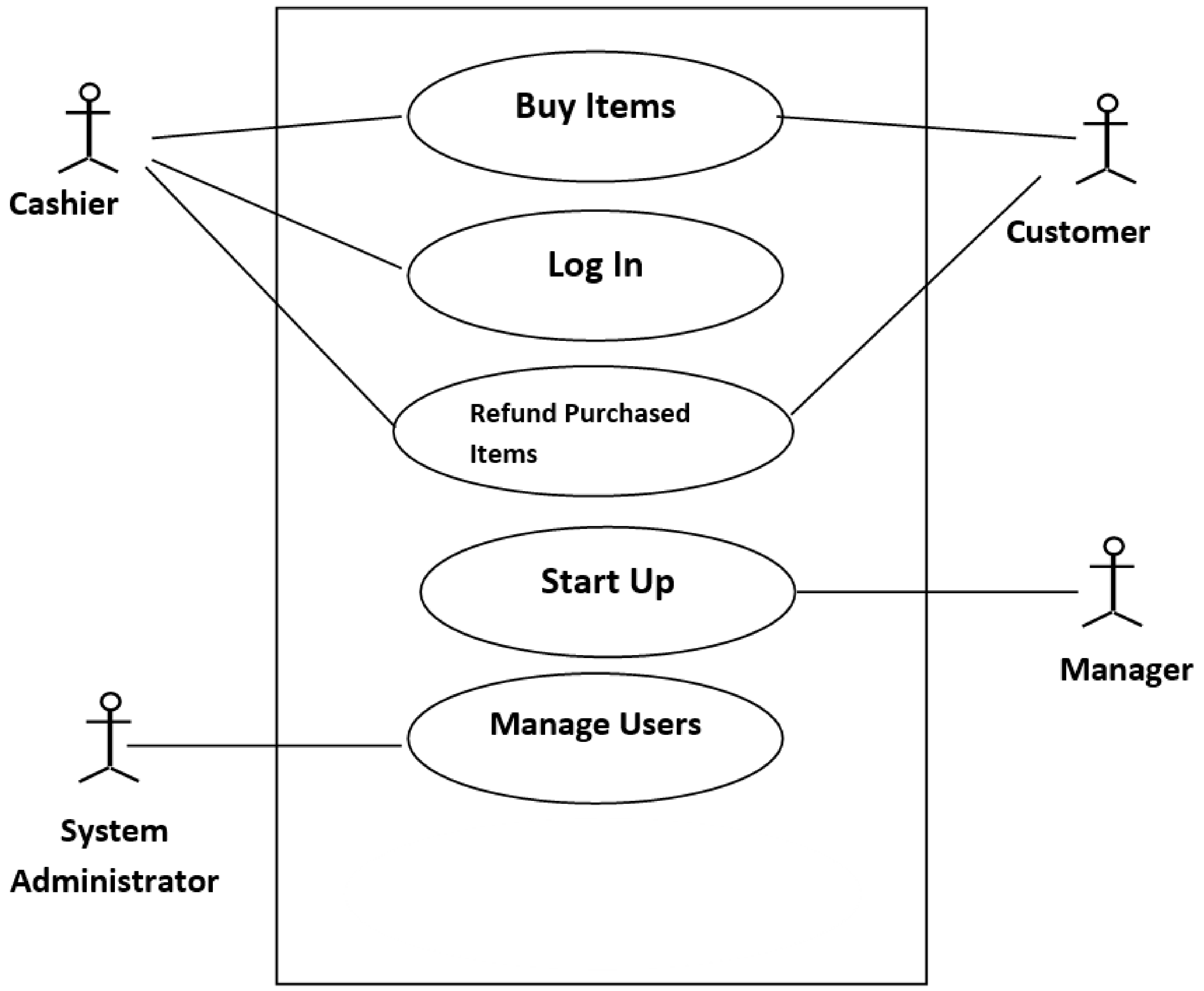
Process Steps for the Point of Sale System

Identifying Actors and Use Cases

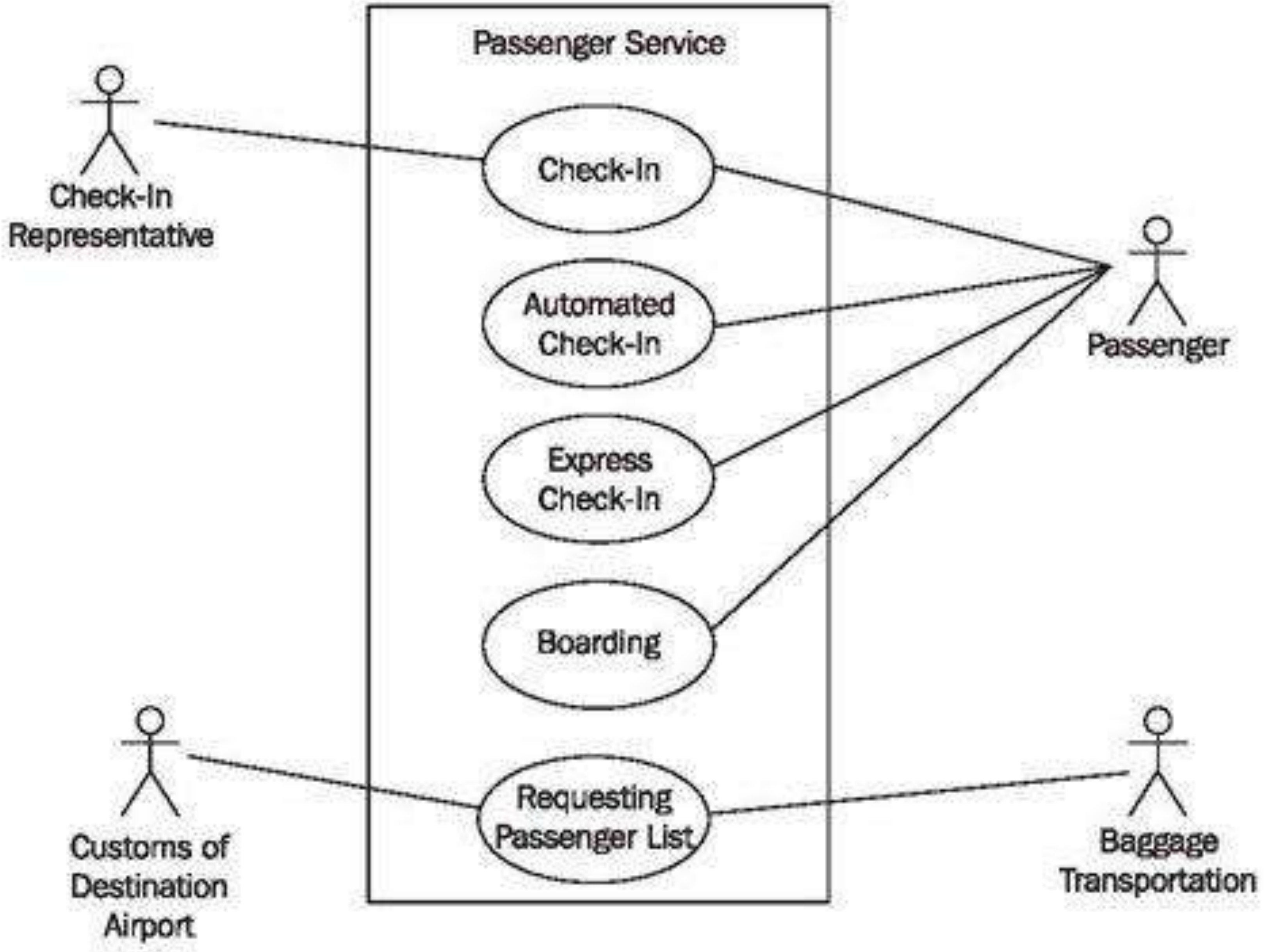
The system boundary is defined as Software and Hardware system---the Usual Case

A list of relevant Actors include

Cashier	Log In/Cash Out
Customer	Buy Items/Refund Items
Manager	Start Up/Shut Down
System Administrator	Add New User



Partial Use Case Diagram for the POST Application


Passenger Service
Check-In
Automated Check-In
Express Check-In
Boarding
Requesting Passenger List
Check-In Representative
Passenger
Customs of Destination Airport
Baggage Transportation

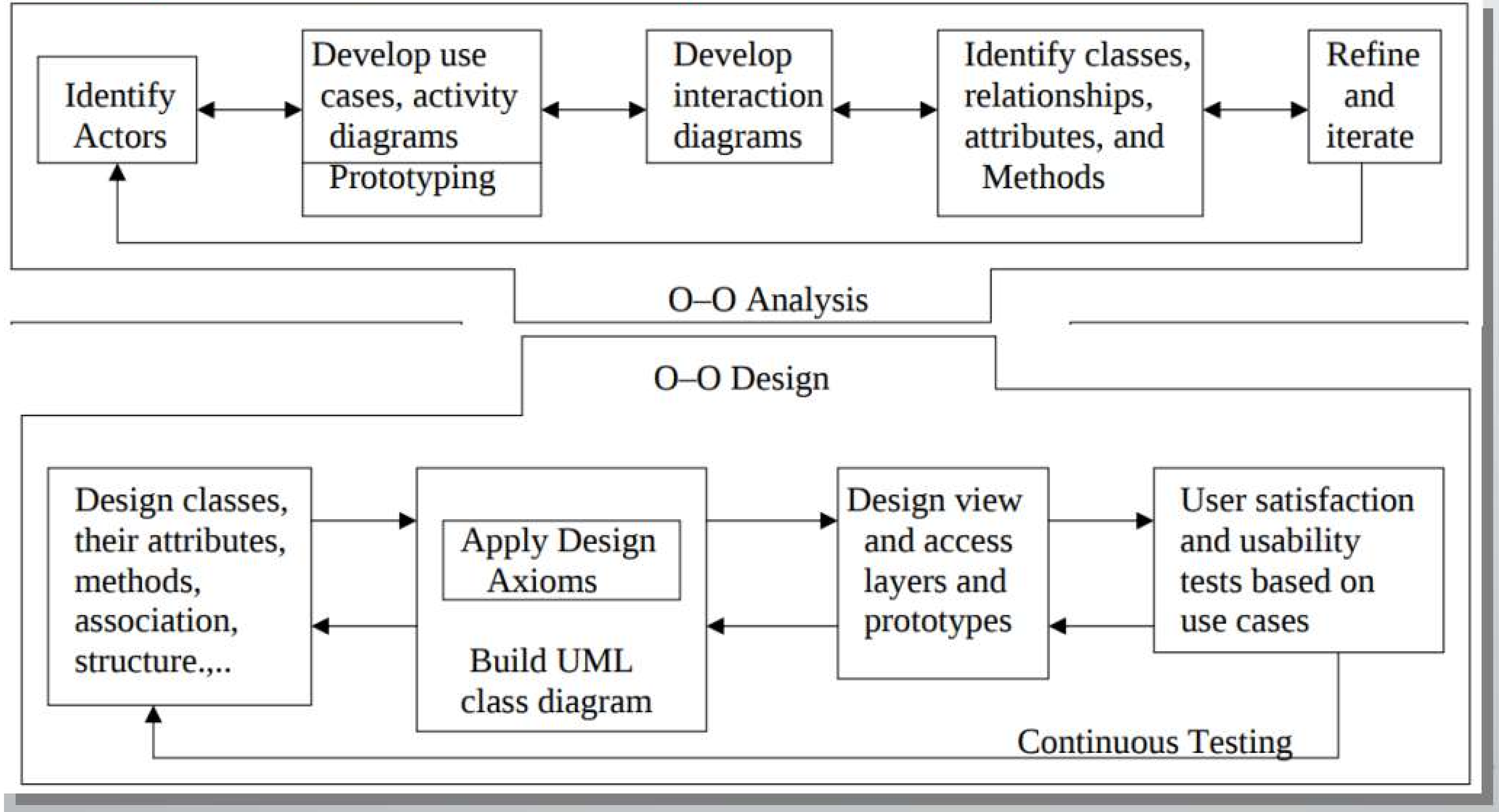
Object Oriented Development Cycle

- System development can be viewed as a process. Furthermore, the development itself, in essence, is a process of change, refinement, transformation, or addition to existing product. The process can be divided into small, interacting phases – subprocesses.
- Each subprocess must have the following:
- A description in terms of how it works
- Specification of the input required for the process
- Specification of the output to be produced
- Generally, the software development process can be viewed as a series of transformations, where the output of one transformation becomes the input of the subsequent transformation
- Transformation 1 (analysis) – translates the users' needs into system requirements and responsibilities.
- Transformation 2 (design) – begins with a problem statement and ends with a detailed design that can be transformed into an operational system.
- Transformation 3 (implementation) – refines the detailed design into the system deployment that will satisfy users' needs.

Cont.

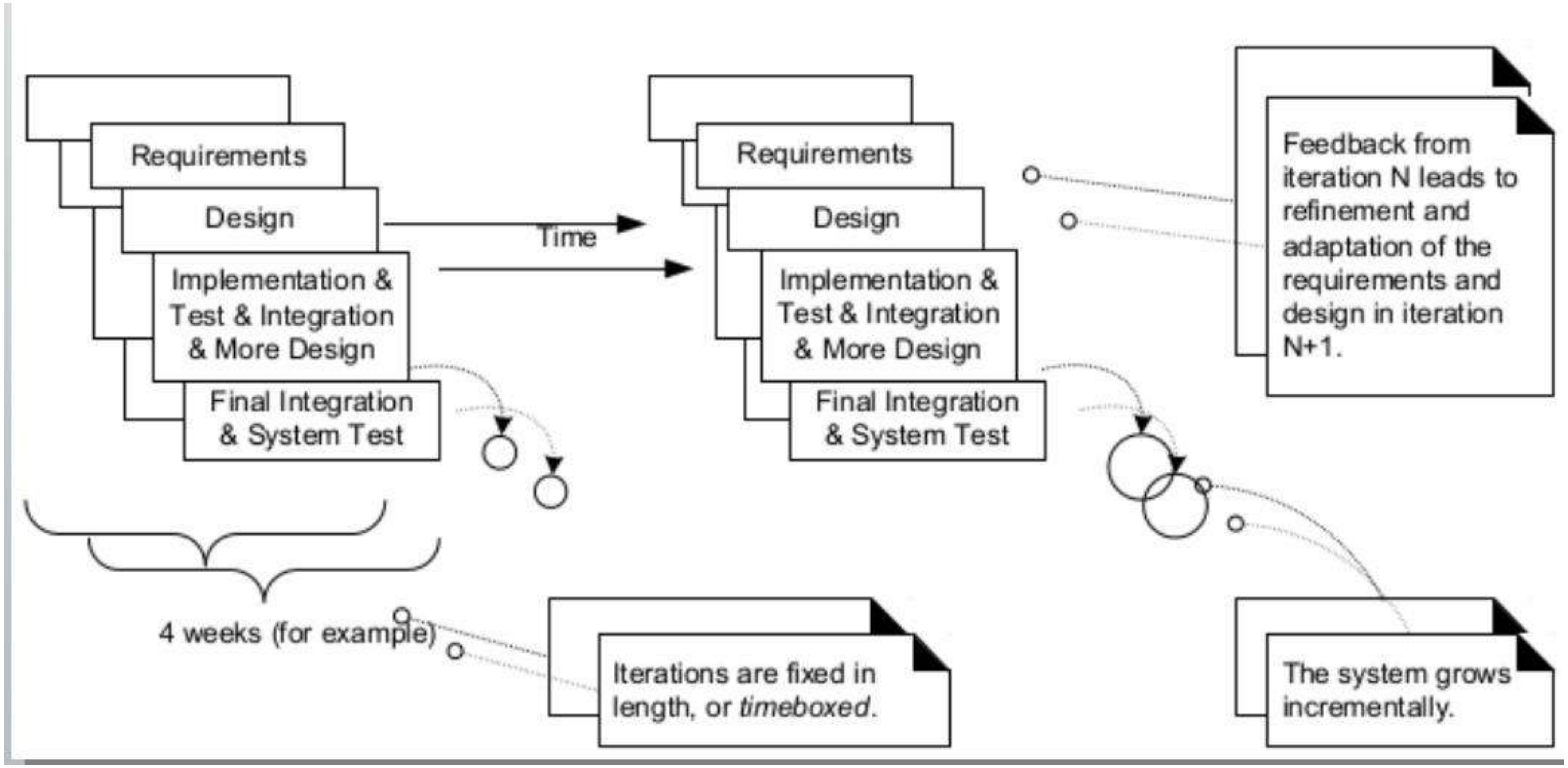
- An example of the software development process is the waterfall approach,
- which starts with deciding what is to be done. Once the requirements have been determined, we next must decide how to accomplish them. This is followed by a step in which we do it, whatever “it” has required us to do.
- We then must test the result to see if we have satisfied the users’ requirements.
- In the real world, the problems are not always well-defined and that is why the waterfall model has limited utility

The object oriented software development life cycle (SDLC) consists of three macro processes: object–oriented **analysis**, object–oriented **design** and object–oriented **implementation**.


Identify Actors
Develop use cases, activity diagrams
Prototyping
Develop interaction diagrams
Identify classes, relationships, attributes, and Methods
Refine and iterate
O–O Analysis
O–O Design
Design classes, their attributes, methods, association, structure.,..
Apply Design Axioms
Build UML class diagram
Design view and access layers and prototypes
User satisfaction and usability tests based on use cases
Continuous Testing

OO Development Cycle

- object–oriented analysis
- object–oriented design
- Iterative Development
- Continuous Testing
- UML Modeling
- Layered Architecture:
- → User Interface Layer
- → Business Layer
- → Access Layer

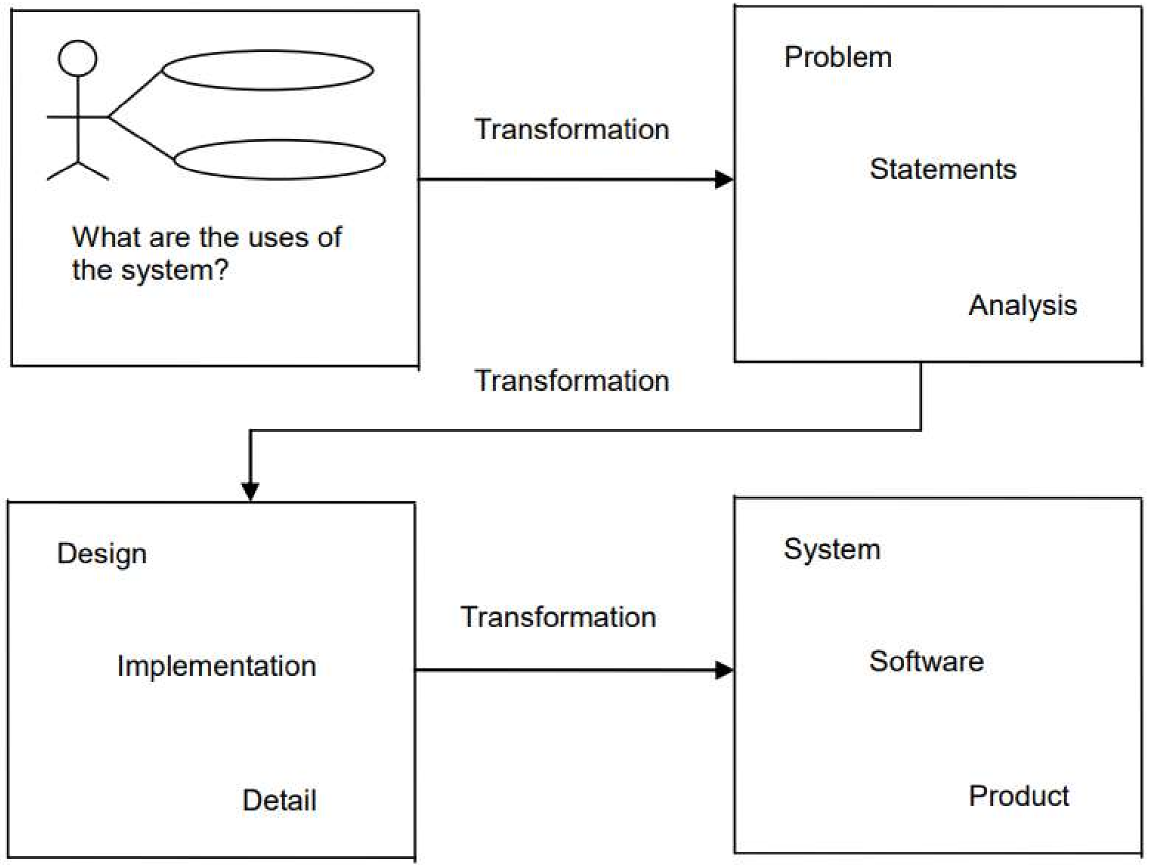

Requirements
Design
Implementation & Test & Integration & More Design
Final Integration & System Test
Time
Requirements
Design
Implementation & Test & Integration & More Design
Final Integration & System Test
Feedback from iteration N leads to refinement and adaptation of the requirements and design in iteration N+1.
4 weeks (for example)
Iterations are fixed in length, or *timeboxed*.
The system grows incrementally.

Iterative Incremental Model

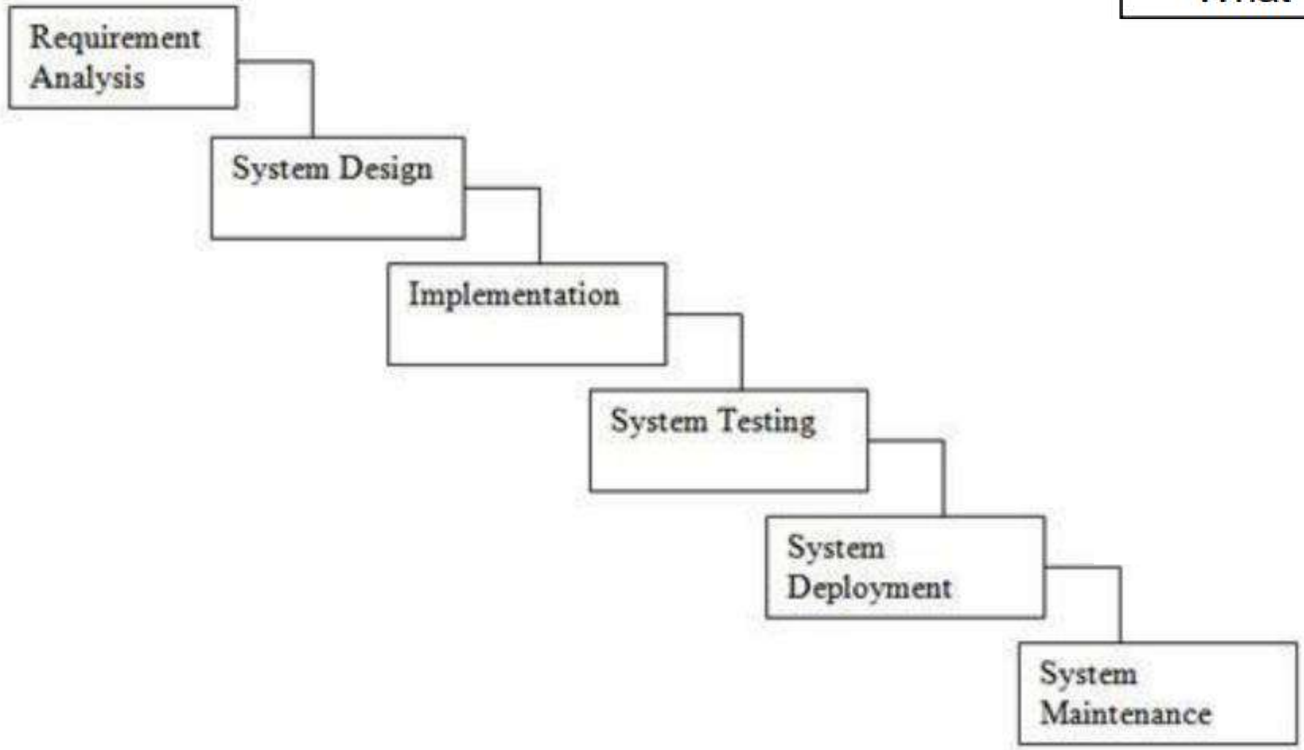
- In complex, changing systems (such as most software projects) feedback and adaptation are key ingredients for success.
- Feedback from early development, programmers trying to read specifications, and client demos to refine the requirements.
- Feedback from tests and developers to refine the design or models.
- Feedback from the progress of the team tackling early features to refine the schedule and estimates.
- Feedback from the client and marketplace to re-prioritize the features to tackle in the next iteration

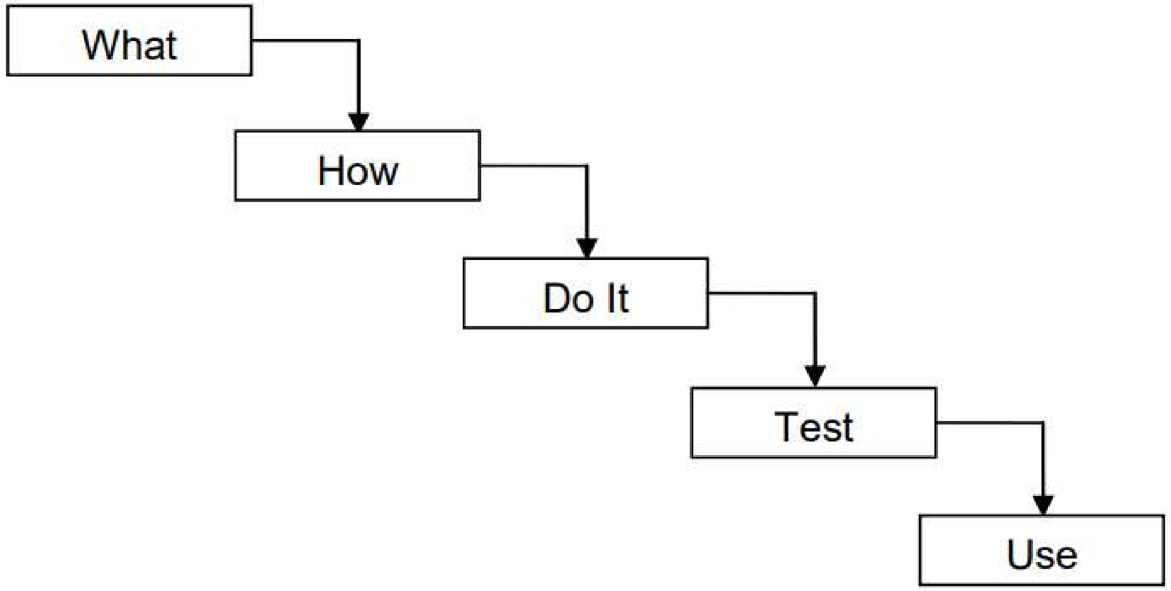
Benefits include:

- less project failure, better productivity, and lower defect rates; shown by research into iterative and evolutionary methods
- early rather than late mitigation of high risks (technical, requirements, objectives, usability, and so forth) early visible progress
- early feedback, user engagement, and adaptation, leading to a refined system that more closely meets the real needs of the stakeholders
- managed complexity; the team is not overwhelmed by "analysis paralysis" or very long and complex steps
- the learning within an iteration can be methodically used to improve the development process itself, iteration by iteration



Software process reflecting transformation from needs to a software product that satisfies those needs





The waterfall software development process

Disadvantage of Waterfall model

Once an application is in the testing stage, it is very difficult to go back and change something that was not well-thought out in the concept stage.

No working software is produced until late during the life cycle.

High amounts of risk and uncertainty.

Not a good model for complex and object-oriented projects.

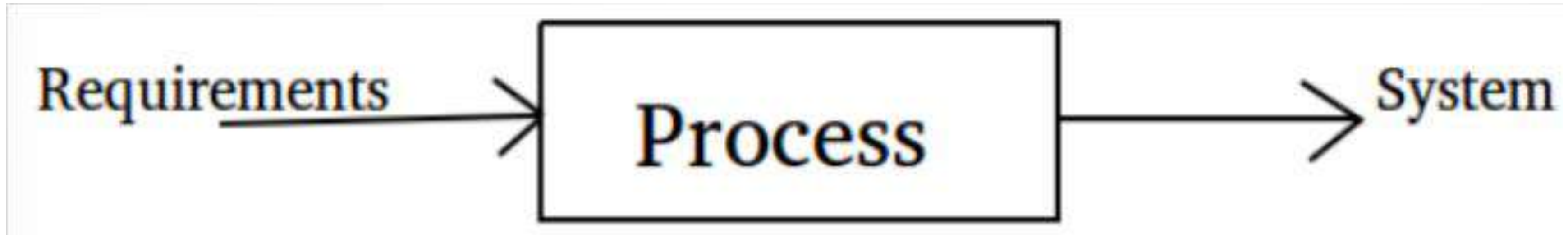
Poor model for long and ongoing projects.

Not suitable for the projects where requirements are at moderate to high risk of changing.

Unified Process

Process:

- Process in OOAD stands for Software Engineering Process(SEP). Process defines who, what, when and how of developing software.
- “People are more important than any process. Good people with a good process will outperform good people with no process every time.” —Grady Booch



Unified Process

- The Unified Software Development Process or Unified Process is a popular iterative and incremental software development process framework.
- Unified Process is an architecture-centric, use- case driven, iterative and incremental development process that leverages unified modeling language and is compliant with the system process engineering meta-model.
- The best-known and extensively documented refinement of the Unified Process is the Rational Unified Process (RUP).
- – Original name: Rational Unified Process (RUP) – Next name: Unified Software Development Process (USDP)
- Other examples are OpenUP and Agile Unified Process.
- A Unified process (UP) is a popular iterative modern process model (framework) derived from the work on the UML and associated process.

Cont

- The Unified Process is not a specific series of steps for constructing a software product since
- – There is a wide variety of different types of software products
- – No such single “one size fits all” methodology could exist
- The Unified Process is an adaptable methodology

– It has to be modified for the specific software product to be developed

- The Unified Process is a modeling technique
- – A model is a set of UML diagrams that represent various aspects of the software product to be developed
- UML stands for unified modeling language

– UML is the tool that we use to represent (model) the target software product

– UML is graphical since a picture is worth a thousand words

Core Workflows of the Unified Process

- i. The Requirements Workflow
- The aim of the requirements workflow is to determine the client's needs

– Gain an understanding of the application domain, that is, the specific business environment in which the target software product is to operate

– Build a business model

- Use UML to describe the client's business processes
- If at any time the client does not feel that the cost is justified, development terminates immediately
- The preliminary investigation of the client's needs is called concept exploration
- It is vital to determine exactly what the client needs and to find out the client's
- constraints
- – Deadline
- – Reliability
- – Cost
- The client will rarely inform the developer how much money is available
- A bidding procedure is used instead

– Parallel running, Portability, and Rapid response time

ii. The Analysis Workflow

- The aim of the analysis workflow is to analyze and refine the requirements to achieve the detailed understanding of the requirements essential for developing a software product correctly and maintaining it easily.
- Why not do this during the requirements workflow?
- – The requirements artifacts must be totally comprehensible by the client
- – The artifacts of the requirements workflow must therefore be expressed in a natural (human) language
- All natural languages are imprecise! An example from a manufacturing information system:
- – “A part record and a plant record are read from the database. If it contains
- the letter A directly followed by the letter Q, then calculate the cost of
- transporting that part to that plant”
- – To what does it refer?
- Two separate workflows are needed

– The requirements artifacts must be expressed in the language of the client

– The analysis artifacts must be precise and complete enough for the designers

iii. The Design Workflow

- The aim of the design workflow is to refine the analysis workflow until the material is in a form that can be implemented by the programmers
- – Many nonfunctional requirements need to be finalized at this time, including
- Choice of programming language
- Reuse issues
- Portability issues
- Retain design decisions

– To backtrack and redesign certain pieces when a dead-end is reached

– To prevent the maintenance team reinventing the wheel

iv. The Implementation Workflow

- The aim of the implementation workflow is to implement the target software product in the selected implementation language
- – A large software product is partitioned into smaller subsystems, which are implemented in parallel by coding teams

– The subsystems consist of components or code artifacts implemented by an individual programmer

v. The Test Workflow

• The test workflow is the responsibility of

– Every developer and maintainer, and

– The quality assurance group

• Traceability of artifacts is an important requirement for successful testing And finally, Postdelivery Maintenance Postdelivery maintenance is an essential component of software

development

– More money is spent on postdelivery maintenance than on all other activities combined

• Problems can be caused by

– Lack of documentation of all kinds

• Two types of testing are needed

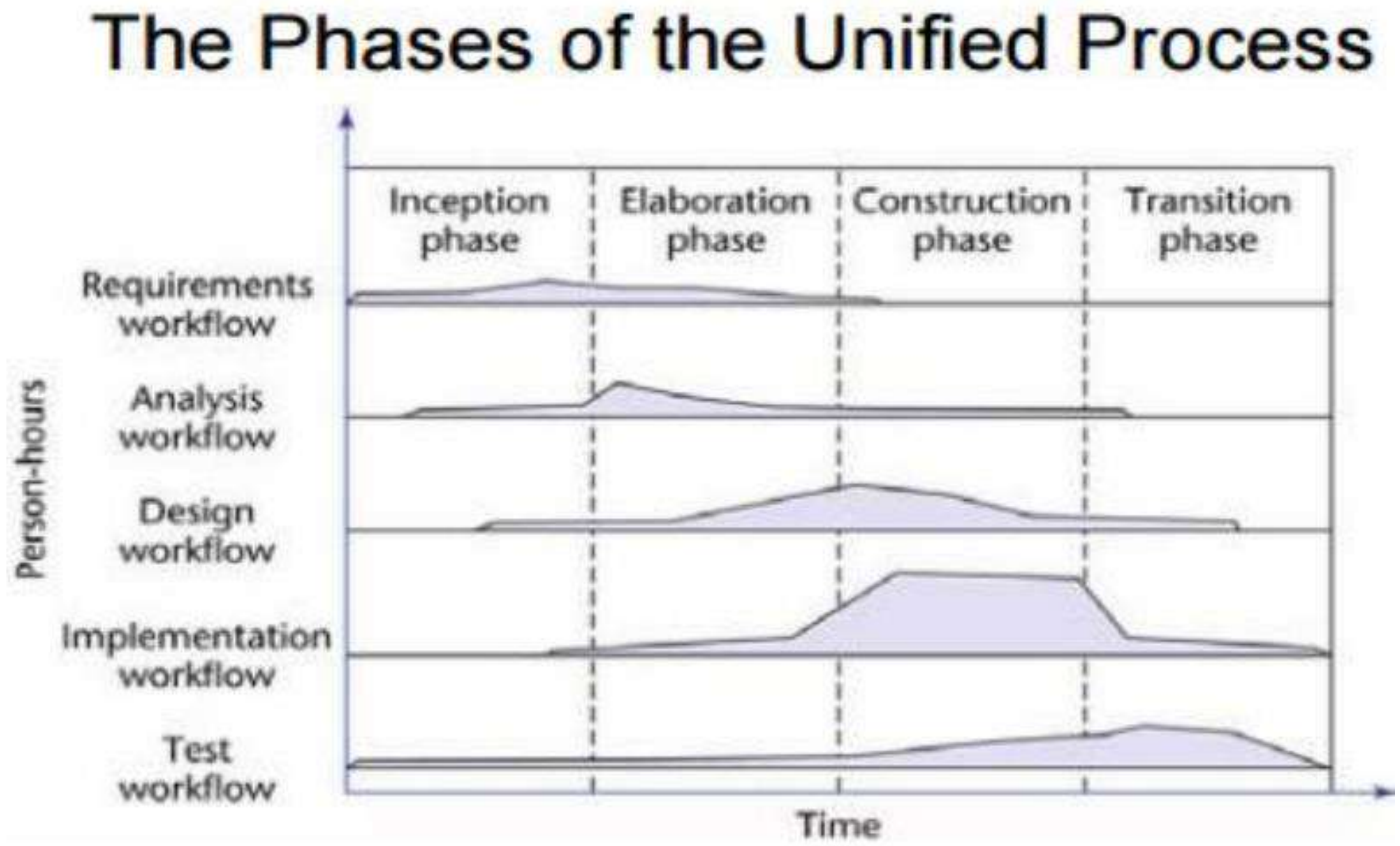
– Testing the changes made during postdelivery maintenance

– Regression testing:

• Make sure that the functionality of the rest of the product has not been compromised.

• All previous test cases (and their expected outcomes) need to be retained

Phases Of unified Process



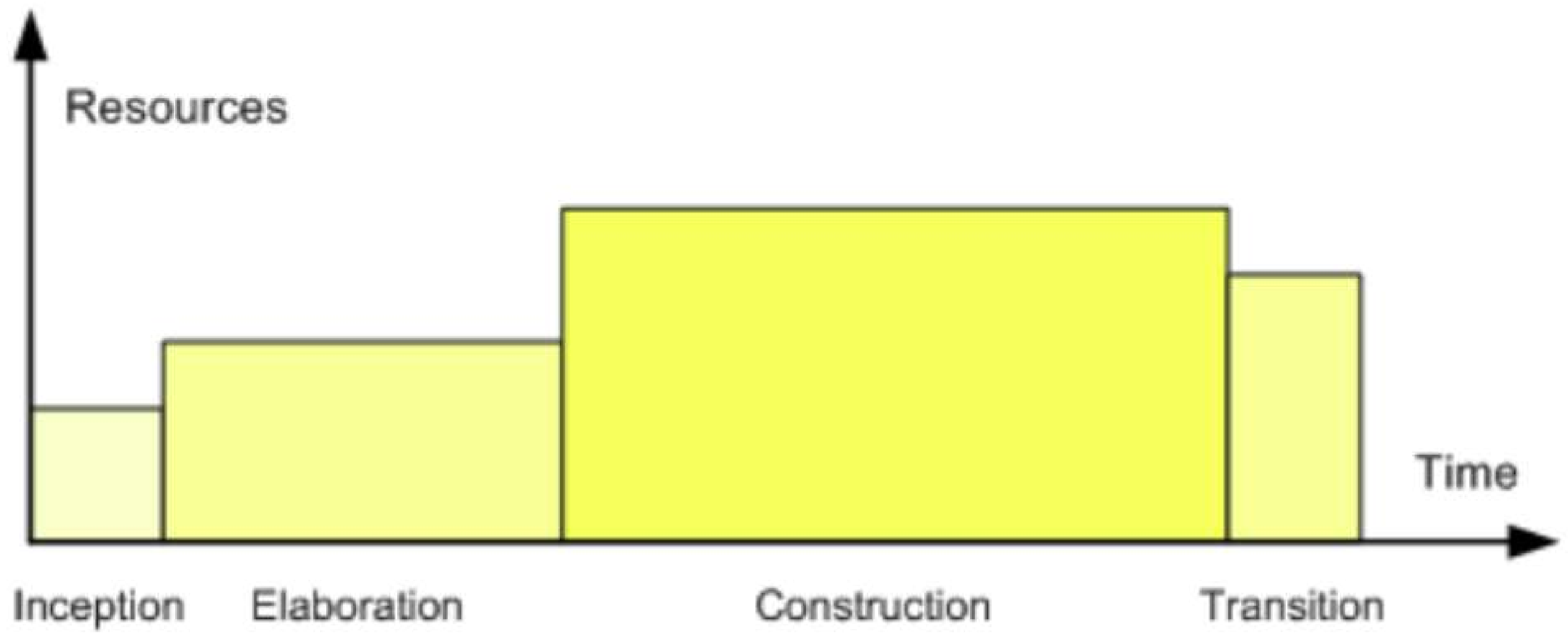


figure: UP phases – Resource-Time diagram

Phases

- 1. Inception (mile stone : life cycle objective)

[approximate vision, business case, scope, vague estimates.]

- In this phase, we develop an approximate vision of the system, make the business case,
- define the scope, and produce rough estimates for cost and schedule
- The aim of the inception phase is to determine whether the proposed software product is economically viable.
- Steps for this phase:

a. Gain an understanding of the domain

b. Build the business model

.Understand precisely how the client organization operates in the domain

c. Delimit the scope of the proposed project

• Focus on the subset of the business model that is covered by the proposed software product

d. Begin to make the initial business case

The Inception Phase: Requirements Workflow -- The Initial Business Case (Sample Questions)

- Is the proposed software product cost effective?

– Have the necessary marketing studies been performed?

– How long will it take to obtain a return on investment?

– What will be the cost not to develop the product?

- Can the proposed software product be delivered in time?
- – What will be the impact if the product is delivered late?
- What are the risks involved in developing the software product, and how can these risks be mitigated?

– Does the team have the necessary experience?

– Is new hardware needed for this software product?

- If so, is there a risk that it will not be delivered in time?
- If so, is there a way to mitigate that risk, perhaps by ordering back-up hardware from another supplier?

– Are software tools needed? Are they currently available? Do they have all the necessary functionality?

The Inception Phase: Requirements Workflow – Risks

- There are three major risk categories:

– Technical risks

– Not getting the requirements right

- Mitigated by performing the requirements workflow correctly

– Not getting the architecture right

- The architecture may not be sufficiently robust
- To mitigate all three classes of risks

– The risks need to be ranked so that the critical risks are mitigated first

The Inception Phase: Analysis, Design, Implementation, Testing Workflows

- A small amount of the analysis workflow may be performed

– Information needed for the design of the architecture is extracted

- A small amount of the design workflow may be performed
- Coding is generally not performed. However, a proof of concept prototype is sometimes built to test the feasibility of constructing part of the software product.
- The test workflow commences almost at the start of the inception phase

– The aim is to ensure that the requirements have been accurately determined

The Inception Phase: Planning

- There is insufficient information at the beginning of the inception phase to plan the entire development

– The only planning that is done at the start of the project is the planning for the inception phase itself

- Similarly, the only planning that can be done at the end of the inception phase is the plan for the elaboration phase

The Inception Phase: Documentation

- The deliverables of the inception phase include:

– The initial version of the domain model

– The initial version of the business model

– The initial version of the requirements artifacts

– A preliminary version of the analysis artifacts

– A preliminary version of the architecture

– The initial list of risks

– The initial ordering of the use cases

– The plan for the elaboration phase – The initial version of the business case

The Inception Phase: The Initial Business Case

- Obtaining the initial version of the business case is the overall aim of the inception phase
- This initial version incorporates – A description of the scope of the software product – Financial details – For marketed product, the business case will include
- Revenue projections, market estimates, initial cost estimates – For in-house product, the business case will include
- The initial cost–benefit analysis TVM, NPV

Elaboration Phase(milestone:life cycle architecture)

- The aim of the elaboration phase is to – Refine the initial requirements

– Refine the architecture

– Monitor the risks and refine their priorities

– Refine the business case

– Produce the project management plan

- The major activities of the elaboration phase are refinements or elaborations of the previous phase

The Tasks of the Elaboration Phase

- The tasks of the elaboration phase correspond to:

– All but completing the requirements workflow

– Performing virtually the entire analysis workflow

– Starting the design of the architecture

The Elaboration Phase: Documentation

- The deliverables of the elaboration phase include:
  - The completed domain model
  - The completed business model
  - The completed requirements artifacts
  - The completed analysis artifacts
  - An updated version of the architecture
  - An updated list of risks
  - The project management plan (for the rest of the project)
  - The completed business case

Construction Phase(milestone: initial operational capability)

• The aim of the construction phase is to produce the first operational-quality version of the software product

– This is sometimes called the beta release

• The emphasis in this phase is on

– Implementation

– Testing

• Unit testing of modules

• Integration testing of subsystems

• Product testing of the overall system

The Transition Phase(milestone:product release)

• The aim of the transition phase is to ensure that the client’s requirements have indeed been met

– Faults in the software product are corrected

– All the manuals are completed

– Attempts are made to discover any previously unidentified risks

• This phase is driven by feedback from the site(s) at which the beta release has been installed

The Transition Phase: Documentation

- The deliverables of the transition phase include:

– All the artifacts (final versions)

– The completed manuals

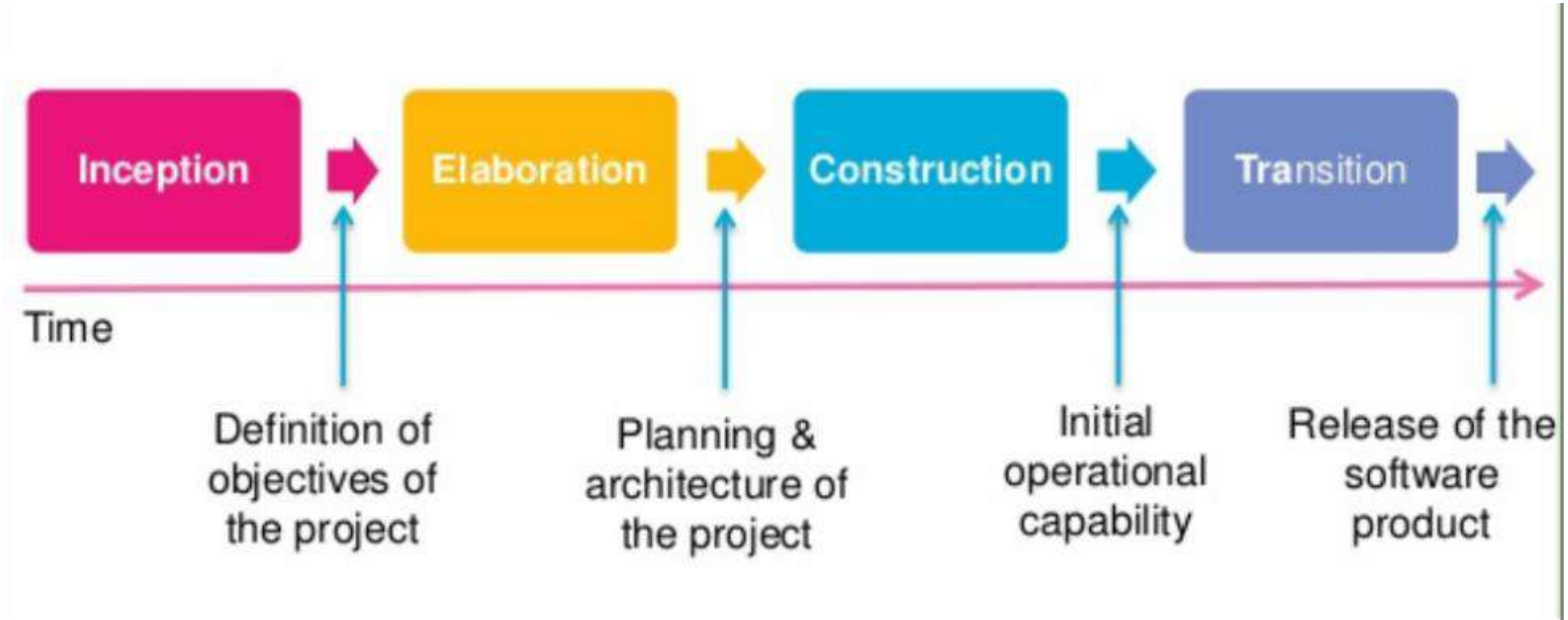

Inception
Elaboration
Construction
Transition
Time
Definition of objectives of the project
Planning & architecture of the project
Initial operational capability
Release of the software product

Figure: UP-Phases outcomes