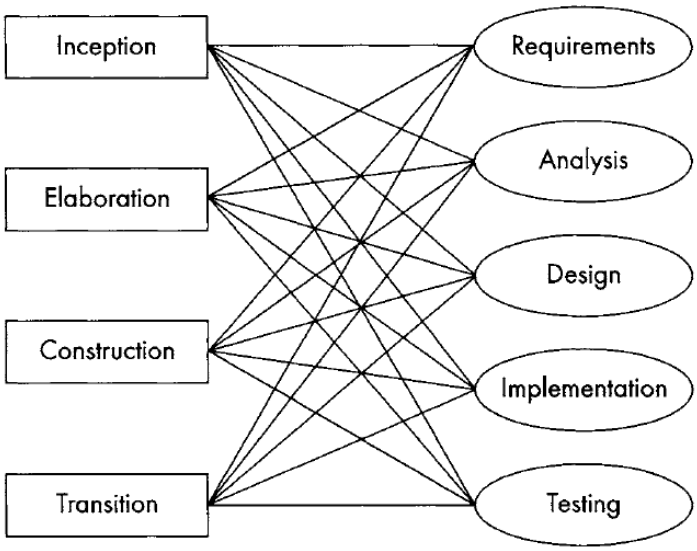
The traditional system life cycle

- Requirements: What are the problems, needs, and wishes of clients and users.	**Planning and requirement analysis**
- Analysis: What does the system look like from the perspective of the clients and users.	**Defining requirements**
- Design: How can the system be constructed to satisfy the requirements.	**Designing the product architecture**
- Implementation: How can the models produced be translated into code.	**Building or developing the product**
- Installation: What is needed to support clients and users so that they can use the new system effectively.	**Testing the product** **Deployment in the market and maintenance**



Traditional life cycle models

- Waterfall: This early life cycle model represents the stages of development as a straightforward sequence, where one stage must be completed before the next begins. However, the waterfall model is not really a true reflection of what actually happens in system development, since it does not emphasize the need to **iterate** over the stages.

- V-model: This is a variation of the waterfall, where the stages are visualized in the form of the letter 'V'. It emphasizes how later stages of development are related to earlier stages; for example, how testing should be derived from the activities that are carried out during requirements and analysis.

- Spiral: This model is also derived from the waterfall. It incorporates iteration of life cycle stages and focuses on identifying and addressing the risks involved in development

- Prototyping: In the prototyping life cycle, implementation takes place early in the development process. The working model produced is subsequently refined and enhanced during a series of iterations until it is acceptable to the client.

- Iterative development: A skeleton version covering the complete functionality of the system is produced and then refined as development progresses.

- Incremental development: In this life cycle model the system is partitioned according to areas of functionality. Each major functional area is developed and delivered independently to the client. For example, in the bike hire system, tasks relating to issuing a bike might be developed and delivered, followed by returning a bike and then maintaining customer records.

The object-oriented approach

The phases in object-oriented development are known as inception, elaboration, construction and transition, indicating the state of the system, rather than what happens at that point in development.

- Inception: it covers the initial work required to set up and agree terms for the project. It includes establishing the business case for the project, incorporating basic risk assessment and the scope of the system that is to be developed.

- Elaboration: it deals with putting the basic architecture of the system in place and agreeing a plan for construction. During this phase a design is produced that shows that the system can be developed within the agreed constraints of time and cost.

- Construction: it involves a series of iterations covering the bulk of the work on building the system; it ends with the beta release of the system, which means that it still has to undergo rigorous testing.

- Transition: covers the processes involved in transferring the system to the clients and users. This includes sorting out errors and problems that have arisen during the development process.

From the next figure we can see that **Any phase may involve all workflows, and a workflow may be carried out during any phase.**

Requirements engineering is traditionally divided into three main stages:

- Elicitation, when information is gathered relating to the existing

- Specification, when the information that has been collected is ordered and documented

- Validation, when the recorded requirements are checked to ensure that they are consistent with what the clients and users actually want and need.

Elitation: During requirements elicitation, the focus is on collecting as much information as possible about what the clients and users want and need from the new system.

- interviews,
- questionnaires,
- study of documents,
- observation of people carrying out day-to-day tasks, assessment of the current computer system if there is one.
- Scenarios:
  * Scenario for the return of a bike

Requirements specification: The main purpose of requirements specification is to collate, order and record the mass of information gathered during the elicitation stage

In this chapter we illustrate two ways of recording requirements in the early stages of development:

- The Problem definition
- The problems and requirements list

No.	*Source*	*Date*	*Description*	*Priority*	*Related Reqs.*	*Related Docs.*	*Change Details*
5.1	Meeting with Annie Price at Wheels	10 Feb. 2004	Keep track of how many bikes a customer is hiring, so that he gets one unified receipt	Essential	1.3 2.3 6.1	Bike cards	

No.	*Source*	*Date*	*Description*	*Priority*	*Related Reqs.*	*Related Docs.*	*Change Details*
2.2	Customer survey	March 2004	Keep record of previous bikes hired by customers	Desirable	2.1 2.4	Bike cards	

Use cases

Use cases describes the main activities of the system from the perspective of the user.

Use cases model the user's view of the functionality of a system. What it does that is of value to the user. Use cases are normally presented in a graphical form, the use case diagram, supported by textual descriptions, use case and actor descriptions, and scenarios.

The use case model consists of:

- A use case diagram,
- A set of use case descriptions,
- A set of actor descriptions, and

- A set of scenarios.

The actor: specifies a role played by a user or any other system that interacts with the subject

Actor descriptions: An actor description briefly describes the actor in terms of role and job title.

The process of breaking down a problem into successively smaller parts in order to understand it better is known as **Decomposition,**

The group of classes involved in a particular use case is known as a **collaboration**.
The situaation where a class needs the help of another class to fulfill one of tis responsabilties.

Identifying use cases from the actors:
Identifying use cases from scenarios: Another approach to identifying use cases is to start with the scenarios. A scenario describes a series of interactions between the user and the system in order to achieve a specified goal. A scenario describes a specific sequence of events, for example what happened when Annie successfully issued a bike to a customer.

Agregations:
Is oftent thought of as a tighter form of association; it models a whole-part relationship between classes. e.g. Wheels, doors and an engine are part of a car. An agregations relationship can be ideniify

The class diagram
The class diagram is central to object-oriented analysis and design, it defines both the software architecture, i.e. the overall structure of the system, and the structure of every object in the system. We use it to model classes and the relationships between classes, and also to model higher-level structures comprising collections of classes grouped into packages.

Relationship between classes and bojects
a class is a factory for creating object, an object is a collection of data and behaviorus that represents some entity. A class defines the structure and behaviour of all antitties.

Association: Associations. In the class diagram in Figure 5.1 hires is modelled as an association between Customer and Bike.

CRC cards
CRC (class-responsibility-collaboration) cards are not officially part of the UML, but are regarded as a valuable technique that works extremely well with it.

The aim of the CRC card technique is to divide the overall functionality of the system into responsibilities which are then allocated to the most appropriate classes. Once we know the responsibilities of a class, we can see whether it can fulfil them on its own, or whether it will need to collaborate with other classes to do this.

On the front of the card is a high-level description of the class, and the back of the card records the class name, responsibilities and collaborations (if any)

The responsibilities will be reorganized, new classes can be introduced etc etc.

Sequence diagrams
Sequence diagrams show clearly and simply the flow of control between objects required to execute a scenario. A scenario outlines the sequence of steps in one instance of a use case from the user's side of the computer screen, a sequence diagram shows how these steps translate into messaging between objects on the computer's side of the screen.

Even for those experienced in object technology, it is very hard to follow the overall flow of control in object-oriented code. Programmers, maintainers and developers need a route map to guide them; this is provided by the sequence diagram.

How do the objects communicate with each other?

Activity diagram
It's a flowchart to represent the flow from one activity to another activity. Capture the dynamic behavior of the system.

State diagrams
What aspect of a system is modelled by a state diagram?
A state diagram models the diferent states that objects of a class can be in, and the events that cause an object to move from one state to another.

Agile software development
Agile software development describes an approach to software development under which requirements and solutions evolve through the collaborative effort of self-organizing and cross-functional teams and their customer(s)/end user(s).

It advocates adaptive planning, evolutionary development, early delivery, and continual improvement, and it encourages rapid and flexible response to change.

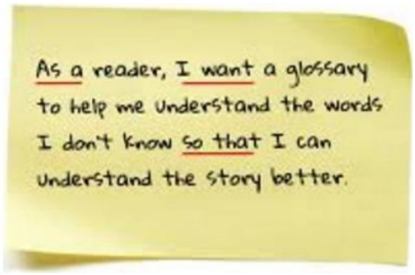
The term Adaptive planning means: Welcome changing requirements, even in late development... being able to adapt to changing requirements if necessary.

Agile software development practices:
- Pair programming: Sometimes referred to as peer programming, is an agile software development technique in which two programmers work as a pair together on one workstation. One, the driver, writes code while the other, the observer, pointer or navigator, reviews each line of code as it is typed in. The two programmers switch roles frequently.

While reviewing, the obsever also considers the "strategic" direction of the work, coming up with ideas for improvements and likely future problems to address. This frees the driver to focus all of his or her attention on the "tactical" aspects of completing the current task, using the observer as a safely net and guide.

- Timeboxing: In time management, timeboxing allocates a fixed time-period, called a time box, to each planned activity. Several project management approaches use timeboxing. It is also used for individual use to address personal tasks in a smaller time frame. It often involves having deliverables and deadlines, which will improve the productivity of the user.

- User stories User stories are one of the primary development artefacts for Scrum and Extreme Programming (XP) project teams. A user story is a very high-level definition of a requirement, containing just enough information so that the developers can produce a reasonable estimate of the effort to implement it.



Popular agile software development methos/frameworks include:
- Scrum
- Extreme programming (XP)
  * User stories

Scrum:
Scrum is an agile framework for managing work with an emphasis on software development. It is designed for teams of three to nine developers who break their work into actions that can be completed

within timeboxed iterations, called (30 days or less) and track Sprints progress and re-plan in 15-minute stand-up meetings, called Daily Scrums.

- Daily Scrums:
  * Every day, a 15 min meeting is held, and the Scrum Master ask the 3 questions:
    ~ What have you accomplished since the last meeting.
    ~ Are there any obstacles in the way of meeting your goal.
    ~ What will you accomplish before the next meeting.

- Sprints:
  * 30 days or less.
  * New functionality is demostrated at the end of each sprint.

Scenario for the buying of a cinema ticket:
- When the customer arrives to the machine the main menu will show the movies available:
- The customer chooses the movie
- The customer enters the number of tickets
- The chooses seat(s) in the room
- The system asks if the user has a loyalty card:
  * If yes, the system then ask if the customer to scan the loyalty card
    ~ The system shows the points in the card
    ~ The system shows an option to cancel the card, redeem all points and return the card back to Odeon Cinemas.
    ~ Go to payment
  * If not:
    ~ The system asks if the customer wants to have one
      * If yes, go to registration
      * If not, go to payment
- Payment menu: The system then shows the different options to pay:
  * Debit card / Credit card
  ~ Insert the card and follow the instructions of the card reader
    - Insert PIN
    - Payment OK / Not OK (Try again)
    - If payment not successful more than three times, the operation will be canceled
    - If the customer has a loyalty card, credit points into the card
  * Use loyalty points (points debited from the card)
- Dispense ticket(s) and Receipt

Use case description:
User case: payment
Actors: Customer
Goal: to buy a ticket
Description: If the payment if not successful bla bla..

Customer
- name
- surname
- email
- loyaltyCard

+ byTicket()

Ticket:
- movie
- seat
- room
- price

Receipt
- price
- date
- paymentMethod

Loyalty card:
- points balance
- credited points
- debited points

---
* registration()

Payment
- Debit card
- Credit card
- PIN
- debitedAmount

Machine
- availableMovie
* displayMovies()
* readCard()
* issuTicket()
* issueReceipt()

Movie
- title
- time
- description