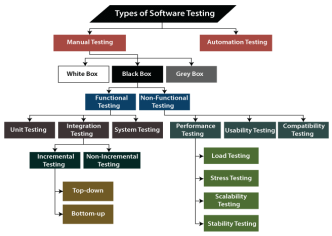
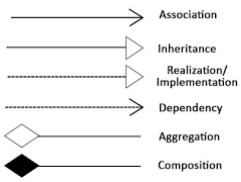
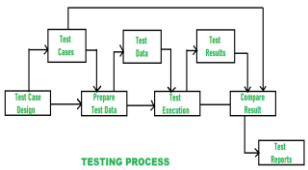
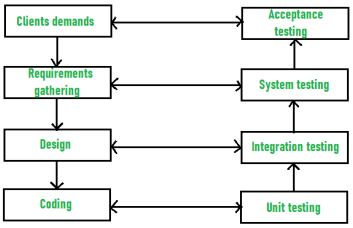






Software Design and Testing

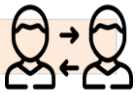
Behavioral Modeling



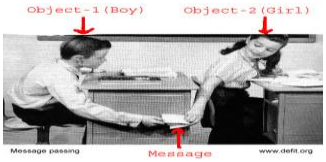
Outline

- Interactions,
- Use-cases,
- Use-Case Diagrams,
- Interaction Diagrams,
- Activity Diagrams,
- State-chart Diagrams.

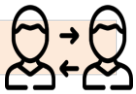
Interactions



- In every interesting system, objects don't just sit idle; they interact with one another by passing messages.
- An interaction is a behavior that comprises a set of messages exchanged among a set of objects within a context to accomplish a purpose.
- We use interactions to model the dynamic aspect of collaborations, representing various objects playing specific roles, all working together to carry out some behavior that's bigger than the sum of the elements.
- These roles represent prototypical instances of classes, interfaces, components, nodes, and use cases.



Interactions



- Their dynamic aspects are visualized, specified, constructed, and documented as flows of control that may encompass simple, sequential threads through a system, as well as more-complex flows that involve branching, looping, recursion, and concurrency.
- You can model each interaction in two ways:
  - by emphasizing its time ordering of messages, or **(Sequence)**
  - by emphasizing its sequencing of messages in the context of some structural organization of objects **(Collaboration)**.
  - Well-structured interactions are like well-structured algorithms - efficient, simple, adaptable, and understandable.

Interactions

- In the UML, you model the static aspects of a system by using such elements as class diagrams and object diagrams.
- These diagrams let you visualize, specify, construct, and document the things that live in your system, including classes, interfaces, components, nodes, and use cases and their instances, together with the way those things sit in relationship to one another.
- In the UML, you model the dynamic aspects of a system by using interactions.
- Like an object diagram, an interaction statically sets the stage for its behavior by introducing all the objects that work together to carry out some action.
- Going beyond object diagrams, however, interactions also introduce messages that are dispatched from object to object.

5

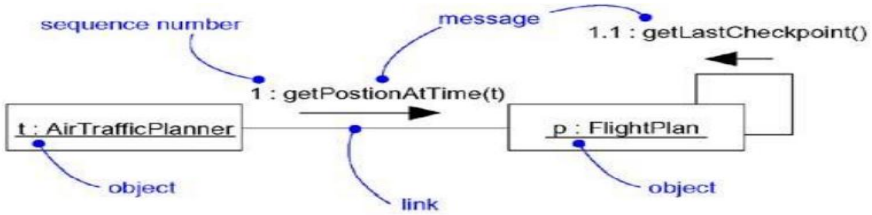
Interactions

- Most often, messages involve the invocation of an operation or the sending of a signal; messages may also encompass the creation and destruction of other objects.
- You use interactions to model the flow of control within an operation, a class, a component, a use case, or the system as a whole.
- Using interaction diagrams, you can reason about these flows in two ways.
  - First, you can focus on how messages are dispatched across time.
  - Second, you can focus on the structural relationships among the objects in an interaction and then consider how messages are passed within the context of that structure.

6

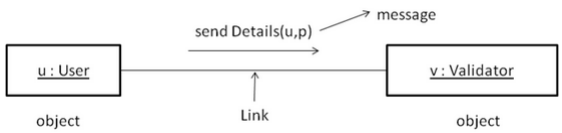
Interactions

- The critical component in an interaction diagram is lifeline and messages.
- The UML provides a graphical representation of messages, as Figure shows(Message, links and Sequencing).
- This notation permits you to visualize a message in a way that lets you emphasize its most important parts: its name, parameters (if any), and sequence.
- Graphically, a message is rendered as a directed line and almost always includes the name of its operation.



Terms and concept, context

- An **interaction** is a behavior that comprises a set of messages exchanged among a set of objects within a context to accomplish a purpose.
- A **message** is a specification of a communication between objects that conveys information with the expectation that activity will ensue.



- You may find an interaction wherever objects are linked to one another.

8

Terms and concept, context

- You'll find interactions in the collaboration of objects that exist in the context of your system or subsystem.
- You will also find interactions in the context of an operation.
- Finally, you'll find interactions in the context of a class.
- **Note :** An interaction may also be found in the representation of a component, node, or use case, each of which, in the UML, is really a kind of classifier.
- In the context of a use case, an interaction represents a scenario that, in turn, represents one thread through the action of the use case.

9

Terms and concept, context

- Most often, you'll find **interactions in the collaboration of objects** that exist in the context of your system or subsystem as a whole.
- For example, in a *system for Web commerce*, you'll find objects on the client (such as instances of the classes BookOrder and OrderForm) interacting with one another.
- You'll also find objects on the client (again, such as instances of BookOrder) interacting with objects on the server (such as instances of BackOrderManager).

10

Terms and concept, context

- These interactions therefore not only involve localized collaborations of objects (such as the interactions surrounding OrderForm), but they may also cut across many conceptual levels of your system (such as the interactions surrounding BackOrderManager).
- You'll also find interactions among **objects in the implementation of an operation**.
- The parameters of an operation, any variables local to the operation, and any objects global to the operation (but still visible to the operation) may interact with one another to carry out the algorithm of that operation's implementation.

11

Terms and concept, context

- For example, invoking the operation moveToPosition(p : Position) defined for a class in a mobile robot will involve the interaction of a parameter (p), an object global to the operation (such as the object currentPosition), and possibly several local objects (such as local variables used by the operation to calculate intermediate points in a path to the new position).
- Finally, you will find **interactions in the context of a class.**
- You can use interactions to visualize, specify, construct, and document the semantics of a class.
- For example, to understand the meaning of a class RayTraceAgent, you might create interactions that show how the attributes of that class collaborate with one another (and with objects global to instances of the class and with parameters defined in the class's operations).

12

Objects and Roles

- The objects that participate in an interaction are either concrete things or prototypical things.
- As a concrete thing, an object represents something in the real world.
- For example, p an instance of the class Person, might denote a particular human.
- Alternately, as a prototypical thing, p might represent any instance of Person.
- When we say "Dhawan is the man of the match", we are referring to a particular instance of the player class. This is an example for concrete things.
- When we say "a player is selected on merit", we are referring to any instance of player class. This is an example for prototypical things.

13

Links

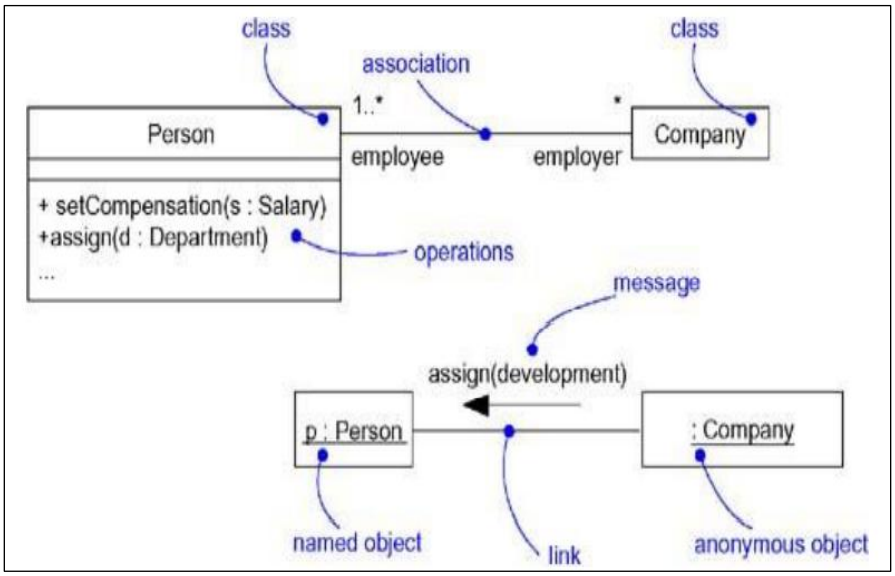
- A link is a semantic connection among objects.
- In general, a link is an instance of association.
- Wherever, a class has an association with another class, there may be a link between the instances of the two classes.
- Wherever there is a link between two objects, one object can send messages to another object.
- We can adorn the appropriate end of the link with any of the following standard stereotypes:

association	Specifies that the corresponding object is visible by association
self	Specifies that the corresponding object is visible as it is the dispatcher of the operation
global	Specifies that the corresponding object is visible as it is in an enclosing scope
local	Specifies that the corresponding object is visible as it is in local scope
parameter	Specifies that the corresponding object is visible as it is a parameter

Links

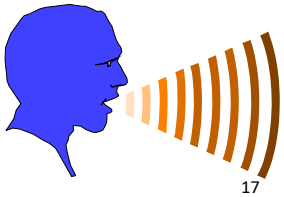
- A link is a semantic connection among objects.
- In general, a link is an instance of association.
- Wherever, a class has an association with another class, there may be a link between the instances of the two classes.
- Wherever there is a link between two objects, one object can send messages to another object.
- We can adorn the appropriate end of the link with any of the following standard stereotypes:

Links



Message

- A message is the specification of communication among objects that conveys information with the expectation that activity will succeed.
- The receipt of a message instance may be considered an instance of an event.
- When a message is passed, the action that results is an executable statement that forms an abstraction of a computational procedure.
- An action may result in a change of state.



17

Message

- In UML, we can model several kinds of actions like

Call	Invokes an operation on an object
Return	Returns a value to the caller
Send	Sends a signal to the object
Create	Creates an object
Destroy	Destroys an object

18

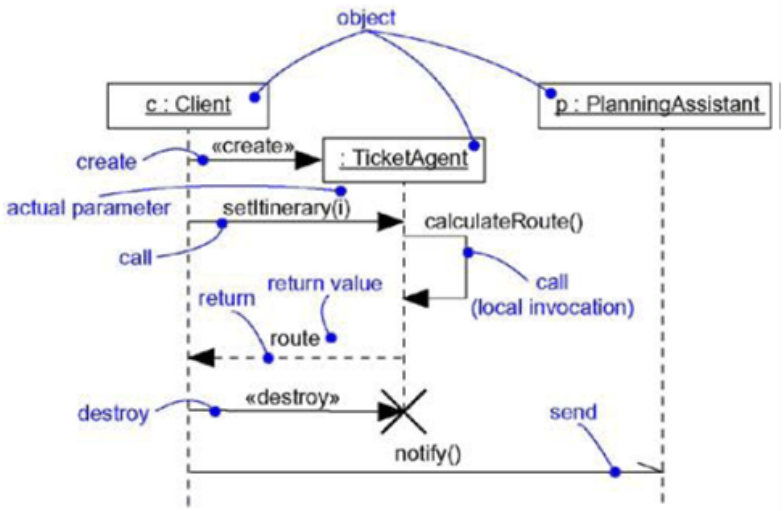
Message

- The most common kind of message you'll model is the call, in which one object invokes an operation of another (or the same) object.
- An object can't just call any random operation.
- If an object, such as c in the example, calls the operation setItinerary on an instance of the class TicketAgent, the operation setItinerary must not only be defined for the class TicketAgent (that is, it must be declared in the class TicketAgent or one of its parents), it must also be visible to the caller c.

19

Message

- In UML, we can model several kinds of actions like



20

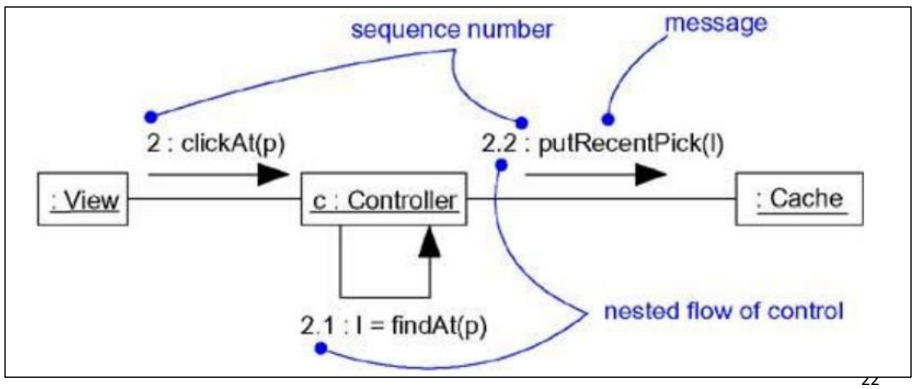
Sequencing

- When an object passes a message to another object, the receiving object might in turn send a message to another object, which might send a message to yet a different object and so on.
- This stream of messages forms a sequence. So, we can define a sequence as a stream of messages. Any sequence must have a beginning. The start of every sequence is associated with some process or thread.
- To model the sequence of a message, we can explicitly represent the order of the message relative to the start of the sequence by prefixing the message with a sequence number set apart by a colon separator.

21

Sequencing

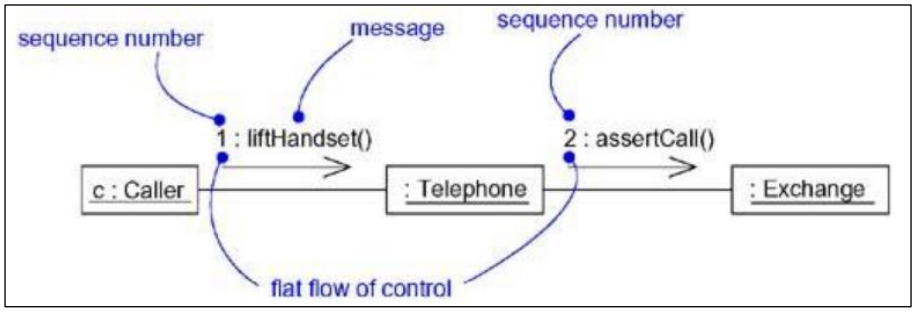
- Most commonly, you can specify a procedural or nested flow of control, rendered using a filled solid arrowhead, as Figure **(Procedural Sequence)** shows.
- In this case, the message findAt is specified as the first message nested in the second message of the sequence (2.1).



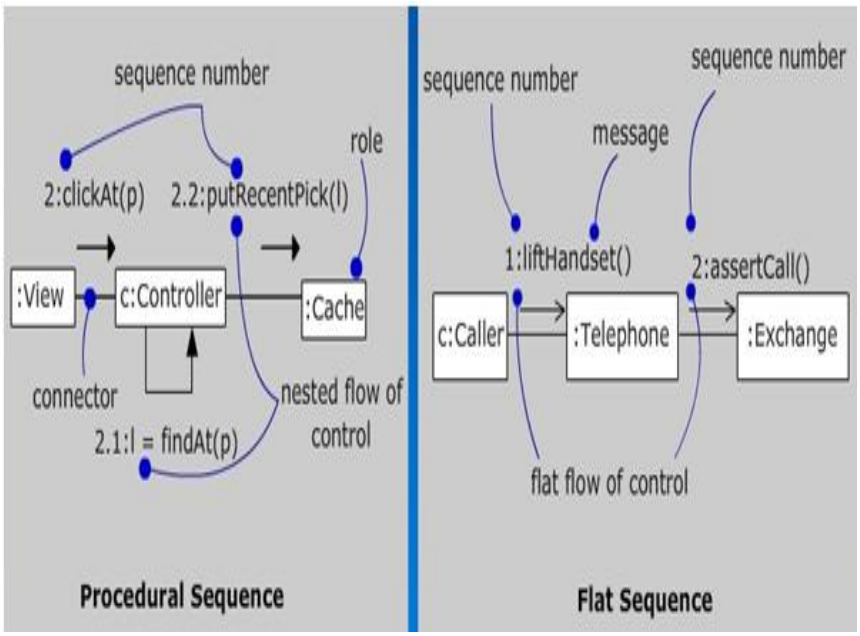
22

Sequencing

- Less common but also possible, as Figure **(Flat Sequence)** shows, you can specify a flat flow of control, rendered using a stick arrowhead, to model the nonprocedural progression of control from step to step.
- In this case, the message assertCall is specified as the second message in the sequence



23



Sequencing

- When you are modeling interactions that involve multiple flows of control, it's especially important to identify the process or thread that sent a particular message.
- In the UML, you can distinguish one flow of control from another by prefixing a message's sequence number with the name of the process or thread that sits at the root of the sequence.
- For example, the expression **D5 : ejectHatch(3)** specifies that the operation ejectHatch is dispatched (with the actual argument 3) as the fifth message in the sequence rooted by the process or thread named **D**.
- Not only can you show the actual arguments sent along with an operation or a signal in the context of an interaction, you can show the return values of a function as well.

25

Sequencing

- As the following expression shows, the value p is returned from the operation find, dispatched with the actual parameter "**Race**".

  1.3.2 : p := find("Race“)
- This is a nested sequence, dispatched as the second message nested in the third message nested in the first message of the sequence.
- In the same diagram, **p** can then be used as an actual parameter in other messages.

26

Creation, Modification, and Destruction

- Most of the time, the objects you show participating in an interaction exist for the entire duration of the interaction.
- However, in some interactions, objects may be created (specified by a create message) and destroyed (specified by a destroy message).
- The same is true of links: the relationships among objects may come and go.
- To specify if an object or link enters and/or leaves during an interaction, you can attach one of the following constraints to the element:

new	Specifies that the instance or link is created during execution of the enclosing interaction
destroyed	Specifies that the instance or link is destroyed prior to completion of execution of the enclosing interaction
transient	Specifies that the instance or link is created during execution of the enclosing interaction but is destroyed before completion of execution

27

Representation

- When you model an interaction, you typically include both objects (each one playing a specific role) and messages (each one representing the communication between objects, with some resulting action).
- We can visualize those objects and messages involved in an interaction in two ways: by emphasizing the **time ordering** of messages and by emphasizing the **structural organization** of the objects that send and receive messages.
- In UML, the first kind of representation is called a **sequence** diagram and the second kind of representation is called a **collaboration** diagram.
- Both sequence and collaboration diagrams are known as interaction diagrams.

28

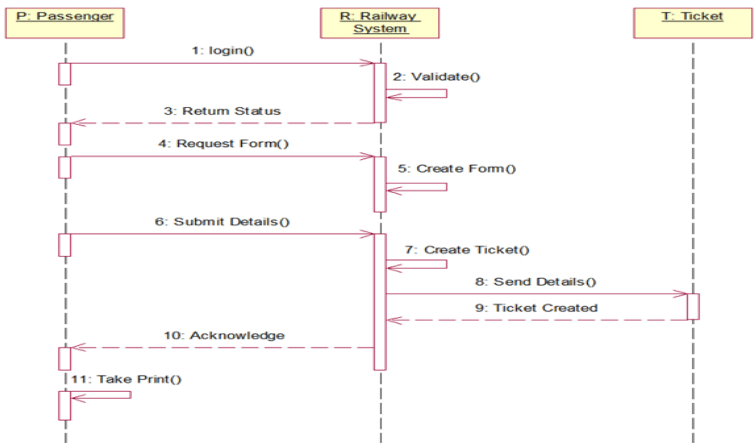
Representation

- Sequence diagrams and collaboration diagrams are isomorphic, meaning that we can take one and transform it into the other without loss of information.
- Sequence diagram lets us to model the lifeline of an object. An object's lifeline represents the existence of the object at a particular time.
- A collaboration diagram lets us to model the structural links that may exist among the objects in the interaction.

29

Representation

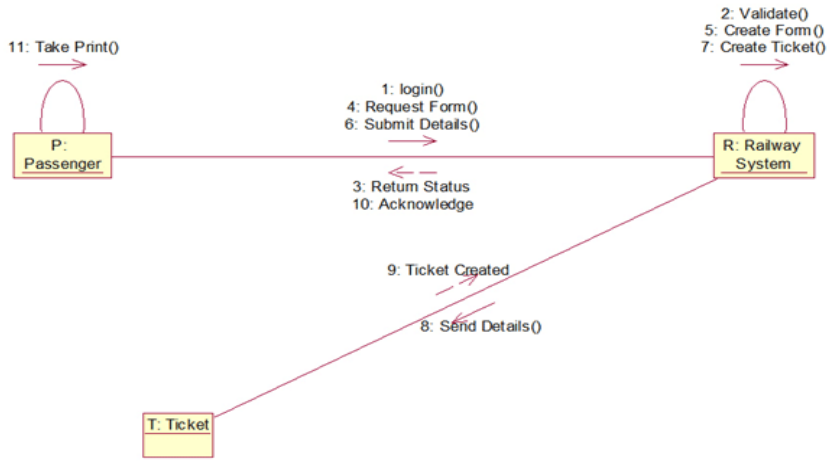
- Consider the following example of railway reservation system's sequence and collaboration diagrams:



30

Representation

- Consider the following example of railway reservation system's sequence and collaboration diagrams:



Modeling a flow of control

- The most common purpose for which you'll use interactions is to model the flow of control that characterizes the behavior of a system as a whole, including use cases, patterns, mechanisms, and frameworks, or the behavior of a class or an individual operation.
- Whereas classes, interfaces, components, nodes, and their relationships model the static aspects of your system, interactions model its dynamic aspects.
- When you model an interaction, you essentially build a storyboard of the actions that take place among a set of objects.
- Techniques such as CRC cards are particularly useful in helping you to discover and think about such interactions.

32

Modeling a flow of control

1. Set the context for the interaction, whether it is the system as a whole, a class or an individual operation.
2. Identify the objects and their initial properties which participate in the interaction.
3. Identify the links between objects for communication through messages.
4. In time order, specify the messages that pass from object to object. Use parameters and return values as necessary.
5. To add semantics, adorn each object at every moment in time with its state and role.

33

34