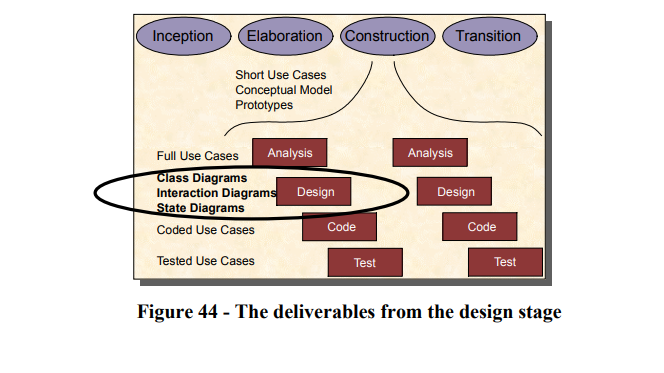
**Chapter 12**

**The Construction Phase : Design**

**Introduction**

We have developed the Use Cases for the first iteration to a deep level of detail, and we are now ready to design the solution to the problem.

Use Cases are satisfied by objects interacting🡺 we need to decide on what objects we need, what the objects are responsible for doing, and when the objects need to interact.



Goals

Have to produce 3 types of model after at the end of design stage:

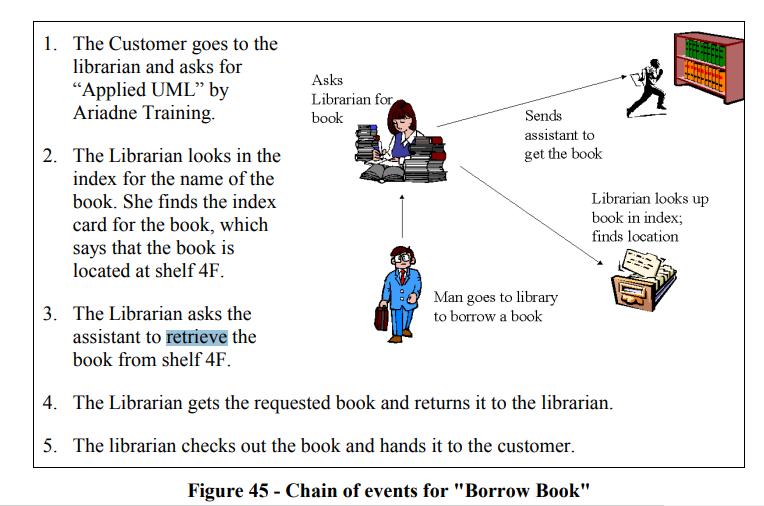
Class diagram: express the classes of objects we have, and how the classes are related.(In fact,almost the Class Diagram is already finish since we will expand our Conceptual Model which has been done already.)

Interaction Diagrams: UML support 2 diagrams(Sequence Diagram, Collaboration Diagram) which is used to express the interaction of objects.

State Model

Collaboration Diagrams

Real life example

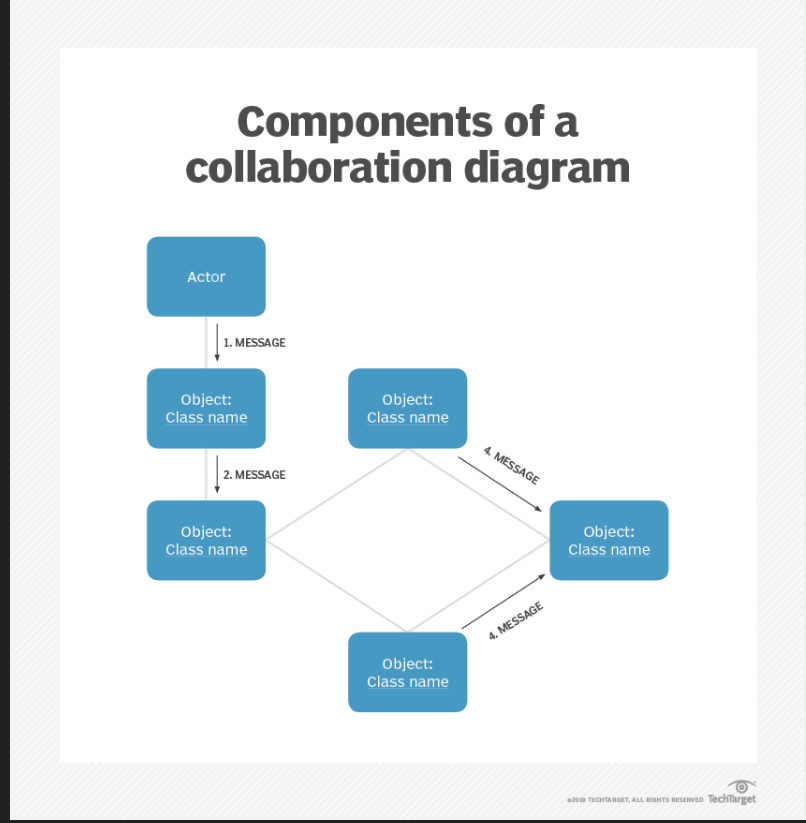


**Define:**

is an illustration of the relationships and interactions among software [objects](https://www.techtarget.com/searchapparchitecture/definition/object) in the Unified Modeling Language ([UML](https://www.techtarget.com/searchsoftwarequality/definition/Unified-Modeling-Language)). These diagrams can be used to portray the dynamic behavior of a particular [use case](https://www.techtarget.com/searchsoftwarequality/definition/use-case) and define the role of each object.

Collaboration diagrams are created by first identifying the structural elements required to carry out the functionality of an interaction. A model is then built using the relationships between those elements. Several vendors offer software for creating and editing collaboration diagrams.

Components and Notaions of Collaboration Diagram:



1. Objects- Objects are shown as rectangles with naming labels inside. The naming label follows the convention of object name: class name. If an object has a property or state that specifically influences the collaboration, this should also be noted.



1. Actors- Actors are instances that invoke the interaction in the diagram. Each actor has a name and a role, with one actor initiating the entire use case.
2. Links- Links connect objects with actors and are depicted using a solid line between two elements. Each link is an instance where messages can be sent.



1. messages- Messages between objects are shown as a labeled arrow placed near a link. These messages are communications between objects that convey information about the activity and can include the sequence number.



**Steps for Creating Collaboration Diagrams**

1. Identify behavior whose realization and implementation is specified
2. Identify the structural elements (class roles, objects, subsystems) necessary to carry out the functionality of the collaboration
   * Decide on the context of interaction: system, subsystem, use case and operation
3. Model structural relationships between those elements to produce a diagram showing the context of the interaction
4. Consider the alternative scenarios that may be required
   * Draw instance level collaboration diagrams, if required.
   * Optionally draw a specification level collaboration diagram to summarize the alternative scenarios in the instance level sequence diagrams

**Collaboration Diagram VS Sequence Diagram**

Difference:

* Collaboration diagrams are used to visualize the structural organization of objects and their interactions.
* Sequence diagrams, on the other hand, focus on the order of messages that flow between objects.

Common: in most scenarios, a single figure is not sufficient in describing the behavior of a system and both figures are required.

When we should use collaboration diagram?(Pros and conns)

Because of the format of the collaboration diagram, they tend to better suited for analysis activities (see Activity: Use-Case Analysis).   Specifically, they tend to be better suited to depicting simpler interactions of smaller numbers of objects.  However, if the number of objects and messages grows, the diagram becomes increasingly hard to read.  In addition, it is difficult to show additional descriptive information such as timing, decision points, or other unstructured information that can be easily added to the notes in a sequence diagram. So, here are some use cases that we want to create a collaboration diagram for:

* Model collaborations between objects or roles that deliver the functionalities of use cases and operations
* Model mechanisms within the architectural design of the system
* Capture interactions that show the messages passing between objects and roles within the collaboration
* Model alternative scenarios within use cases or operations that involve the collaboration of different objects and interactions
* Support the identification of objects (hence classes) that participate in use cases
* Each message in a collaboration diagram has a sequence number.
* The top-level message is numbered 1. Messages sent during the same call have the same decimal prefix but suffixes of 1, 2, etc. according to when they occur.

**Some issue can be cause when created collaboration diagram**

1. We have not mentioned anything on the diagram about how the user inputs data into the system, and how the data (such as the list of runners) is output on the screen. Somehow all of this happens as if by magic “inside” the actor. We’ll see later that this is good design. We want to make our design as flexible as possible, and by including detail about the User Interface at this stage, we are tying ourselves down to one specific solution.
2. How does the “Race” object find out what runners are part of that race? Clearly, there is some kind of database (or even network) operation going on here. Again, we do not want to tie our design down at this stage, so we defer these details until later.
3. Why have we made the “runner” object responsible for tracking which bets have been placed on it? Why didn’t we create another class, perhaps called “bet handler” or “betting system”? This issue will be explored in the following chapter

**Guidelines**

* Keep the diagram simple
* Don’t try to capture every scenario
* Avoid creating classes whose name contains “controller”, “handler”, manager” or “driver”
* Avoid God classes.