Use case diagram

A **use case diagram** is a graphical depiction of a user's possible interactions with a system. A use case diagram shows various use cases and different types of users the system has and will often be accompanied by other types of diagrams as well. The use cases are represented by either circles or ellipses. The actors are often shown as stick figures.

Class diagram

In software engineering, a **class diagram** in the Unified Modelling Language (UML) is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among objects.

The class diagram is the main building block of object-oriented modelling. It is used for general conceptual modelling of the structure of the application, and for detailed modelling, translating the models into programming code. Class diagrams can also be used for data modelling. The classes in a class diagram represent both the main elements, interactions in the application, and the classes to be programmed.

In the diagram, classes are represented with boxes that contain three compartments:

- The top compartment contains the name of the class. It is printed in bold and cantered, and the first letter is capitalized.
- The middle compartment contains the attributes of the class. They are left-aligned, and the first letter is lowercase.
- The bottom compartment contains the operations the class can execute. They are also left-aligned, and the first letter is lowercase.

In the design of a system, several classes are identified and grouped together in a class diagram that helps to determine the static relations between them. In detailed modelling, the classes of the conceptual design are often split into subclasses.

Activity (UML)

An **activity** in Unified Modelling Language (UML) is a major task that must take place in order to fulfil an operation contract. The Student Guide to Object-Oriented Development defines an activity as a sequence of activities that make up a process. Activities can be represented in activity diagrams

An activity can represent:

- The invocation of an operation.
- A step in a business process.
- An entire business processes.

Activities can be decomposed into sub activities, until at the bottom we find atomic actions.

The underlying conception of an activity has changed between UML 1.5 and UML 2.0. In UML 2.0 an activity is no longer based on the state-chart rather it is based on a Petri net like coordination mechanism. There the activity represents user-defined behaviour coordinating actions. Actions in turn are pre-defined (UML offers a series of actions for this).

Sequence diagram

A **sequence diagram** or **system sequence diagram** (SSD) shows object interactions arranged in time sequence in the field of software engineering. It depicts the objects involved in the scenario and the sequence of messages exchanged between the objects needed to carry out the functionality of scenario. Sequence diagrams are typically associated with use case realizations in the logical view of the system under development. Sequence diagrams are sometimes called **event diagrams** or **event scenarios**.

Component diagram

In Unified Modeling Language (UML), a component diagram depicts how components are wired together to form larger components or software systems. They are used to illustrate the structure of arbitrarily complex systems.

A component diagram allows verification that a system's required functionality is acceptable. These diagrams are also used as a communication tool between the developer and stakeholders of the system. Programmers and developers use the diagrams to formalize a

roadmap for the implementation, allowing for better decision-making about task assignment or needed skill improvements. System administrators can use component diagrams to plan ahead, using the view of the logical software components and their relationships on the system.[1]

Deployment diagram

A **deployment diagram** in the Unified Modelling Language models the *physical* deployment of artifacts on nodes.[1] To describe a web site, for example, a deployment diagram would show what hardware components ("nodes") exist (e.g., a web server, an application server, and a database server), what software components ("artifacts") run on each node (e.g., web application, database), and how the different pieces are connected (e.g. JDBC, REST, RMI).

The nodes appear as boxes, and the artifacts allocated to each node appear as rectangles within the boxes. Nodes may have sub nodes, which appear as nested boxes. A single node in a deployment diagram may conceptually represent multiple physical nodes, such as a cluster of database servers.

There are two types of Nodes:

1. Device Node
2. Execution Environment Node

Device nodes are physical computing resources with processing memory and services to execute software, such as typical computers or mobile phones. An execution environment node (EEN) is a software computing resource that runs within an outer node and which itself provides a service to host and execute other executable software elements.

Entity–relationship model

An **entity–relationship model** (or **ER model**) describes interrelated things of interest in a specific domain of knowledge. A basic ER model is composed of entity types (which classify the things of interest) and specifies relationships that can exist between entities (instances of those entity types).

In software engineering, an ER model is commonly formed to represent things a business needs to remember in order to perform business processes. Consequently, the ER model becomes an abstract data model, that defines a data or information structure which can be implemented in a database, typically a relational database.

Entity–relationship modeling was developed for database and design by Peter Chen and published in a 1976 paper,[1] with variants of the idea existing previously.[2] Some ER models show super and subtype entities connected by generalization-specialization relationships,[3] and an ER model can be used also in the specification of domain-specific ontologies.