Definitions UML.

Model: abstraction of reality.

Purpose of modeling: Modeling simplifies the comprehension and communication around the problem (but it doesn't simplify the problem itself!).
Why model? To manage the complexity, to perpetuate the expertise (some projects may take many years), to increase productivity, to manage variability (notion of *Software Product Lines* to manage variants of same software), to facilitate software evolution management.

Software product line: Creating variants (different versions) from one complete software. It is a collection of software that are similar or belonging to the same domain.
A collection of software variants: same domain + share a set of commonalties but also contain some variability.

Software = Documentation + Models + Code

UML (Unified Modeling Language) is a language for visualizing, specifying, constructing, documentating.

Use case diagram: Defines the functional requirements of a system. Shows use cases, actors, and their interrelationships.
Use cases: the desired functionality.

Actor: *External* entity that *interacts* with the system.
Uses one or more functionalities of the system → primary actor.
Participates to the realization of one or more functionalities → secondary actor.

Class: Abstract representation of an entity inside the system. (The object structure: classes)

Attribute: An attribute defines a value that *qualifies* an object.

Class diagram: Shows a collection of static model elements such as classes and types, their contents and their relationships.

Operation: Defines a service offered and carried out by an object. Conceptually, operations describe the responsibilities of a class, but more obviously, they correspond to the programmatic methods defined in a class.
Visibility: is used to specify encapsulation (hides implementation from clients). (*specify the public interface and hidden implementation*)

Association: stable connection (persistent) between two objects. A reflexive association links objects of the same class.
Navigability is the way to access the properties (attributes and operations) of other objects.

Difference between aggregation/composition: Aggregation expresses a weak composition (shared). The life cycle of the component is not linked to that of the composite: if the composite is deleted, the component remains. (empty diamond)

A composition expresses a strong composition (not shared). The life cycle of the component is linked to that of the composite: if the composite is deleted, the component will also be deleted.
Inheritance (generalization) allows to share common [attributes, operations and associations] and preserve differences. *Identifiable with words as "is a kind of".*

Package: Allows to gather a set of elements (*classes, use cases, etc.*). A package is a namespace (naming): two classes with the same name cannot be in the same package. A package can contain other packages, and import or use others.

Object diagram: shows a snapshot of the objects and connections between them at a point in time. It is used to test and demonstrate complex data structures.

Sequence diagram: Specify how objects work together to realize a single functionality. (*inter-object view*). Is also called "interaction diagram".
It shows a sequence of messages exchanged by the objects of a system.
It helps us find the methods of our classes, and validate that the data logical data structure is sufficient to realize the functionality.
Key concepts: **Participant**: entities participating in the interaction. **Messages**: communications between objects.

Communication diagram: It is an isomorphic (same meaning, different syntax) form of sequence diagrams. They place the emphasis first on object structure, then on message sequences. Message order is shown by a hierarchical numbering scheme.

Difference between sequence & communication diagram: In sequence diagram, the order is defined. Time is specified whereas in communication diagram it is not.

Difference between UML models and UML diagrams: A model is an abstract representation of our system, whereas a diagram is a complete/partial graphical view on our model.

State machines: Specify the global behavior (participation in all functionalities) of a single object. (*intra-object view*) Two main concepts: state and transition.
A state is a "stage of existence" during which an object satisfies certain conditions, executes an activity, or waits for an event. The state defines how the object reacts to new events in its "life".

Super state:
History:

Code generation: GOALS: Productivity gain (the structure is generated in a single mouse-click), quality gain (re-use of know-how). The traditional software developer is replaced by: a *domain model creator* (specialized in domain analysis) and a *code generator developer* (specialized in technical code design).

Reverse engineering: Automatic operation that generates UML models from the source code. It allows the generation of representations of a system in a language of higher abstraction than the source code of a system. INPUT: code, OUTPUT: model. Reverse engineering generates a UML model (and not diagrams) possible to extract views (diagrams).

Round trip engineering: Operation where you manipulate at the same time models and the code. Keep synchronization between the model and the source code.
From model to code and back again. Keeps model and code in perfect (!) synchronization.