**Objective-C and swift** are recent programming languages. Version control usually refers to the management of changes to source code (although other files may be tracked). Revisions are numbered, logged and comparison can be made with earlier version of the code.

**A software metric** is a standard of measure of a degree to which a software system or process possesses some property; for example, cohesion. Since software quality is correlated with some properties, for example, high cohesion, software metrics are used to seek high-quality software. This, the statement is true

**An application programming interface (API)** is a set of subroutine definitions, communication protocols, and tools for building software. Significant effort is placed in their design to facilitate other software developers use of APIs to build complex software

**The Unified Modeling Language (UML)** is a general-purpose, developmental, modelling language, that is intended to provide a standard way to visualize the design of a system. It has several diagrammatic tools to construct models that represent structure and behaviour of software systems

**A use case** is a list of actions or event steps typically defining the interactions between an actor and a system to achieve a goal. The actor can be a human or other external system. Use case analysis is an important and valuable requirement analysis technique. They define the boundary between system functionality and actor actions

**doxygen automates** the process of producing documentation for special comment in the source code, not a tool for continuous integration. Continuous integration (CI) is the practice of merging all developer working copies to a shared mainline several times a day. A famous tools for CI is Jenkins

A software design is more robust if it is modularised. **Cohesion** is a measure that defines the degree of intra-dependability within elements of a module. **Cohesion** also means responsibilities assigned to a module are strongly related and relevant among each other. Highly cohesive modules exhibit more re-use. The greater the cohesion, the better is the software’s design.

**State diagram show** how some component (an object, or a component) changes from one state to another. A state diagram is a model of behaviour, of how things change, thus it is a dynamic model. In the UML, state diagrams are part of the tools that describe the internal behaviour of a system. Statechart diagrams describe the states of individual objects and the possible transitions between states.

**A software metric** is a measure of software characteristics which are quantifiable or countable. Software metrics are important for many reasons, including measuring software performance, planning work items, and measuring productivity. They are usually believed to evaluate a quality property of the software. The level of nesting of if-then-else statements is an example, because usually very high nesting affects the understandability of the code, resulting in software difficult to maintain.

**Functional requirements** describe how a product must behave, what its features and functions. For example, when a user in Minesweeper uncover a bomb the game is over. Nonfunctional requirements describe the general characteristics of a system. They are also known as quality attributes. For example, the system shall

**A software component** is an operational software package, a web service, a web resource, or a module that encapsulates a set of related functions (or data). Software components often take the form of executable objects (not classes) or collections of objects, The motivation is an approach for development of software from prefabricated components and their services, that scales rapidly because of high re-use.

**Design by contract** prescribes that software designers should define formal, precise and verifiable interface specifications for software components. Such precise definition go beyond the ordinary axiomatic definitions of abstract data types because contracts specify preconditions, postconditions and invariants that users of components must be aware and that components respect. These specifications are referred to as contracts, analogously to conditions and obligations of business contracts. Contacts specify under which conditions a component/class/API will behave appropriately and define specifically what id does when. It also enables software tools to validate these conditions automaticly facilitating bug detection and maintainability

**Doors are of the data-type SecureLocations,** while the SecurityCard, AccessAttempt, and FailedPINAttempt are stereotyped as entities which indicates there be some persistence storage for them (probably a database).

The semantics of “extends” is somewhat ambiguous in UML, but is reasonable to say that to “update a profile” is a followup, and thus an extension, activity to “logging on”. But it could be also modeled by the “includes” relationship (although not every change of profile requires to log on, but it is a pre-requisite the uses is logged on).

**Software engineering (SE)** is concerned with developing and maintaining software systems that behave reliably and efficiently, software applications that are affordable to develop and maintain, and satisfy all the requirements that customers have defined for them. It is important because of the impact of large, expensive software systems and the role of software in safety-critical applications. In software engineering it hard to measure product size fro requirements, and has little raw materials (like in civil engineering).

**A class diagram details** what are the shape of all the information objects in the system. It specifies their constitution, and what are their attribute/properties (information they hold) plus what interface they have to their behaviors. Which objects know about or hold references to other objects (associations) and describes semantic relationship like inheritance and aggregation. This structure will not change while the software is in execution

**An abstract data type (ADT)** is a mathematical model for data types, where a data type is defined by its behavior (semantics) from the point of view of a user of the data; that is what it does. It is independent of the concrete data structure that implements the values and operations defined by the ADT. So and ADT aligns with the idea of design of values and behaviors and leaves aside the specific implementation. For example a stack is a data type with associates operations: push and pop. However, the specific implementation is not specified. It should be implemented by a linked list or by an array. In this way, the design (showing the behaviour) and the implementation (showing how those behaviours can be executed) is separated.

**ERDs (Entity Relationship Diagrams**) are abstract data models whose primary purpose is to define a data or information structure that can be implemented in a database, typically a relational database.

**MVC (Model View Controller)** is a software architectural pattern for implementing user interfaces on computers. It divides a given application into three interconnected parts

**Coupling** is the degree of interdependence between software modules. Primarily, with respect to Tasks, Loosely coupled components can be replaced with alternative implementations, enabling software maintenance or just iterative refinement. **High cohesion** makes durable software product. **High cohesion and low coupling** provide a design with the capacity to adapt the software enables to meet user needs (Software Product). Therefore, with respect to Software engineering, achieving low coupling is best practice, it enables modularity, and manage complexity. 2. The systematic approach to the development is supported by efforts to keep cohesion low during the life cycle. High-cohesion provides usability and extendibility in software development.

**DFD (Data Flow Diagram**) is a graphical representation of the flow of data through an information system, modelling its process aspects

**Classification** - A set of definitions for a particular object which will define the type of behaviour it will exhibit and the attributes it will maintain.

**Encapsulation** - The combining of attributes and behaviour into a single object whereby the actual implementation details are hidden.

**Inheritance** The ability of an object to indicate that it will acquire the attributes and behaviour of some other objects.

**Polymorphism** The ability of different objects to perform the appropriate behaviour in response to the same message.

**information hiding** contributes to low coupling. We will see this is an important property of system design

**Encapsulation** protects from accessing information one should not be aware One needs to know the interface/protocol/signature of how to communicate to the object Not how it is implemented. One does not need to know the exact class of the object, just that the message can eb composed for that object

**80:20 rule in software** is that 80% of users only use 20% of features. Although some several features are not used, the are still important to somebody. Thus, categorizing functional requirement by several 80:20 scales, priorities can be assigned to the most relevant features that will get users moving productively.