Definitions

A

Abstraction is a technique for dealing with complexity. It works by establishing a level of complexity we are interested in, and suppressing the more complex details below that level.

Actor: An actor (in a use case) is a role played by a user. An actor can be a human or another system. Actors are not part of the system; they reside outside the system.

Architecture: The high level structures of a software system, the discipline of creating such structures, and the documentation of these structures. These structures are needed to reason about the software system. Each structure comprises software elements, relations among them, and properties of both elements and relations. The architecture of a software system is a metaphor, analogous to the architecture of a building.

B

Brainstorming: A group activity designed to *generate a large number of diverse and creative ideas* for the *solution* of a problem.

Brooks' law: Adding people to a late project will make it later. -- Fred Brooks (author of *The Mythical Man-Month*)

C

Client component/method/object: The component/method/object that is interacting with a given code.

CLI application: An application that has a *Command Line Interface*. i.e. user interacts with the app by typing in commands.

Commit (noun): a change (aka a *revision*) saved in the Git revision history.
(verb): the act of creating a commit i.e., saving a change in the working directory into the Git revision history.

Coupling: The degree of interdependence between software modules; a measure of how closely connected two routines or modules are; the strength of the relationships between modules.

D

Dependency inversion principle (DIP):

1. High-level modules should not depend on low-level modules. Both should depend on abstractions.
2. Abstractions should not depend on details. Details should depend on abstractions.

Design pattern: An *elegant reusable solution* to a *commonly recurring problem* within a *given context* in software design.

Domain expert: An expert of a discipline to which the product is connected e.g., for a software used for Accounting, a domain expert is someone who is an expert of Accounting.

DRY (Don't Repeat Yourself) principle: Every piece of knowledge must have a single, unambiguous, authoritative representation within a system. -- *The Pragmatic Programmer*, by Andy Hunt and Dave Thomas

Dynamic binding (aka late binding): a mechanism where method calls in code are resolved at runtime, rather than at compile time.

E

Enterprise application: 'enterprise applications' refer to software applications used by organizations and therefore have to meet much higher demands (such as in scalability, security, performance, and robustness) than software meant for individual use.

Exception:

> The term *exception* is shorthand for the phrase "exceptional event." An *exception* is an event, which occurs during the execution of a program, that disrupts the normal flow of the program's instructions. –- Java Tutorial (Oracle Inc.)

Equivalence partition (aka equivalence class): A group of test inputs that are likely to be processed by the SUT in the same way.

F

Feature list: A list of features of a product *grouped according to some criteria* such as aspect, priority, order of delivery, etc.

Focus groups: A kind of *informal interview* within an *interactive group setting*.

G

Glossary: A glossary serves to ensure that *all stakeholders have a common understanding* of the noteworthy terms, abbreviations, acronyms etc.

I

Interface segregation principle (ISP): No client should be forced to depend on methods it does not use.

L

Law of Demeter (LoD):

- An object should have limited knowledge of another object.
- An object should only interact with objects that are closely related to it.

Also known as

- Don't talk to strangers.
- Principle of least knowledge

Liskov substitution principle (LSP): Derived classes must be substitutable for their base classes. -- proposed by Barbara Liskov

O

Open-closed principle (OCP): A module should be *open* for extension but *closed* for modification. That is, modules should be written so that they can be extended, without requiring them to be modified. -- proposed by Bertrand Meyer

P

Pair programming:

> Pair programming is an agile software development technique in which two programmers work together at one workstation. One, the driver, writes code while the other, the observer or navigator, reviews each line of code as it is typed in. The two programmers switch roles frequently. [source: Wikipedia]

Prototype: A prototype is a *mock up, a scaled down version, or a partial system constructed*

- to get users' feedback.
- to validate a technical concept (a "proof-of-concept" prototype).
- to give a preview of what is to come, or to compare multiple alternatives on a small scale before committing fully to one alternative.
- for early field-testing under controlled conditions.

Polymorphism:

> The ability of different objects to respond, each in its own way, to identical messages is called polymorphism. -- Object-Oriented Programming with Objective-C, Apple

R

RCS: **Revision control software** are the software tools that automate the process of *Revision Control* i.e. managing revisions of software artifacts.

Repository (*repo* for short): The database of the history of a directory being tracked by an RCS software (e.g. Git).

S

Separation of concerns principle (SoC): To achieve better modularity, separate the code into distinct sections, such that each section addresses a separate *concern*. -- Proposed by Edsger W. Dijkstra

Single responsibility principle (SRP): A class should have one, and only one, reason to change. -- Robert C. Martin

Software engineering: Software Engineering is the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software" -- IEEE Standard Glossary of Software Engineering Terminology

Stage (verb): Instructing Git to prepare a file for committing.

Stakeholder: A party that is potentially affected by the software project. e.g. users, sponsors, developers, interest groups, government agencies, etc.

Static analysis: Static analysis is the analysis of code without actually executing the code.

Static binding (aka early binding): When a method call is resolved at compile time.

Stub: A stub has the same interface as the component it replaces, but its implementation is so simple that it is unlikely to have any bugs. It mimics the responses of the component, but only for a limited set of predetermined inputs. That is, it does not know how to respond to any other inputs. Typically, these mimicked responses are hard-coded in the stub rather than computed or retrieved from elsewhere, e.g. from a database.

SUT: Software Under Test

T

Testing: Operating a system or component under specified conditions, observing or recording the results, and making an evaluation of some aspect of the system or component. –- source: IEEE

Type signature: The *type signature* of an operation is the type sequence of the parameters. The return type and parameter names are not part of the type signature. However, the parameter order is significant.

Example:

Method	Type Signature
`int add(int X, int Y)`	`(int, int)`
`void add(int A, int B)`	`(int, int)`
`void m(int X, double Y)`	`(int, double)`

Method	Type Signature
`void m(double X, int Y)`	`(double, int)`

U

Unified Modeling Language (UML) is a graphical notation to describe various aspects of a software system. UML is the brainchild of three software modeling specialists James Rumbaugh, Grady Booch and Ivar Jacobson (also known as the Three Amigos). Each of them had developed their own notation for modeling software systems before joining forces to create a unified modeling language (hence, the term 'Unified' in UML). UML is currently the *de facto* modeling notation used in the software industry.

Use case: A description of a set of sequences of actions, including variants, that a system performs to yield an observable result of value to an actor [📖 : uml-user-guide].

User story: User stories are short, simple descriptions of a feature told from the perspective of the person who desires the new capability, usually a user or customer of the system. [Mike Cohn]

User story format: `As a {user type/role} I can {function} so that {benefit}`

W

Working directory: the root directory revision-controlled by Git (e.g., the directory in which the repo was initialized).

Y

YAGNI (You Aren't Gonna Need It!) Principle: Do not add code simply because 'you might need it in the future'.