# **Software Engineering**

## **SDLC (Software Development Life Cycle)**

Term used to describe a process for planning, creating, testing, and deploying an information system

Is also known as

* **Software Development Methodology** /  system development methodology
* **Software Development Process** / software process
* **Software Development Paradigms** (a Greek word meaning example)

## **Image result for sdlc phases**Different Phases/Stages/Activities of SDLC

* **Planning** - Cost estimation, budgeting, scheduling
* **Requirement analysis/Gathering (requirements definition)**
* Feasibility study
* System Analysis
* **Software Design**
* **Implementation /Coding**
* **Testing & Integrating**
* **Deployment**
* Measuring products and processes.
* Quality improvement.
* **Maintenance** and operation

## Common Methodologies/Approaches (Software development Process / Paradigm)

1. **Various types of**[**Agile Software**](https://en.wikipedia.org/wiki/Agile_methodology) **Development**
2. [Waterfall](https://en.wikipedia.org/wiki/Waterfall_model) (Traditional) Model
3. [Prototyping](https://en.wikipedia.org/wiki/Software_prototyping)
4. [**Iterative and incremental development**](https://en.wikipedia.org/wiki/Iterative_and_incremental_development)
5. [Spiral development](https://en.wikipedia.org/wiki/Spiral_development)
6. [Rapid application development](https://en.wikipedia.org/wiki/Rapid_application_development)
7. [Extreme programming](https://en.wikipedia.org/wiki/Extreme_programming)
8. [Code and fix](http://en.wikipedia.org/wiki/Software_development_process#Code_and_fix)
9. [Lightweight methodologies](http://en.wikipedia.org/wiki/Software_development_process#Lightweight_methodologies)
10. Crystal Methods
11. Dynamic Systems Development Model (DSDM)
12. Extreme Programming (XP)
13. Feature Driven Development (FDD)
14. Joint Application Development (JAD)
15. Lean Development (LD)

# Agile

**Agile Development**” is an umbrella term for several iterative and incremental software **development methodologies**. The most popular **agile methodologies** include Extreme Programming (XP), Scrum, Crystal, Dynamic Systems **Development** Method (DSDM), Lean **Development**, and Feature-Driven **Development** (FDD)

## Agile principles

The Agile Manifesto is based on twelve principles:

1. Customer satisfaction by early and continuous delivery of valuable software
2. Welcome changing requirements, even in late development
3. Working software is delivered frequently (weeks rather than months)
4. Close, daily cooperation between business people and developers
5. Projects are built around motivated individuals, who should be trusted
6. Face-to-face conversation is the best form of communication (co-location)
7. Working software is the principal measure of progress
8. Sustainable development, able to maintain a constant pace
9. Continuous attention to technical excellence and good design
10. Simplicity—the art of maximizing the amount of work not done—is essential
11. Best architectures, requirements, and designs emerge from self-organizing teams
12. Regularly, the team reflects on how to become more effective, and adjusts accordingly

## Popular agile software development frameworks include (but are not limited to):

* Scrum
* [Kanban](https://en.wikipedia.org/wiki/Kanban_(development))
* [Scrumban](https://en.wikipedia.org/wiki/Scrumban)
* [Adaptive software development](https://en.wikipedia.org/wiki/Adaptive_software_development) (ASD)
* [Agile modeling](https://en.wikipedia.org/wiki/Agile_modeling)
* [Agile Unified Process](https://en.wikipedia.org/wiki/Agile_Unified_Process) (AUP)
* [Crystal Clear methods](https://en.wikipedia.org/wiki/Crystal_Clear_(software_development))
* [Disciplined agile delivery](https://en.wikipedia.org/wiki/Disciplined_agile_delivery)
* [Dynamic systems development method](https://en.wikipedia.org/wiki/Dynamic_systems_development_method) (DSDM)
* [Extreme programming](https://en.wikipedia.org/wiki/Extreme_programming) (XP)
* [Feature-driven development](https://en.wikipedia.org/wiki/Feature-driven_development) (FDD)
* [Lean software development](https://en.wikipedia.org/wiki/Lean_software_development)
* [Rapid application development](https://en.wikipedia.org/wiki/Rapid_application_development) (RAD)

[**https://en.wikipedia.org/wiki/List\_of\_software\_development\_philosophies**](https://en.wikipedia.org/wiki/List_of_software_development_philosophies)

# Software Architecture vs design

|  |  |  |
| --- | --- | --- |
| Category | Architecture styles | Description |
| Communication | Service-Oriented Architecture (SOA) | *Refers to applications that expose and consume functionality as a service using contracts and messages.* |
| Message Bus  (EBS Middleware) | *An architecture style that prescribes use of a software system that can receive and send messages using one or more communication channels, so that applications can interact without needing to know specific details about each other.* |
| Deployment | Client/Server | *Segregates the system into two applications, where the client makes requests to the server. In many cases, the server is a database with application logic represented as stored procedures.* |
| N-Tier / 3-Tier | *Segregates functionality into separate segments in much the same way as the layered style, but with each segment being a tier located on a physically separate computer.* |
| Domain | Domain Driven Design | *An object-oriented architectural style focused on modeling a business domain and defining business objects based on entities within the business domain.* |
| Structure | Layered Architecture | *Partitions the concerns of the application into stacked groups (layers).* |
| Object-Oriented | *A design paradigm based on division of responsibilities for an application or system into individual reusable and self-sufficient objects, each containing the data and the behavior relevant to the object.* |
| Component-Based | *Decomposes application design into reusable functional or logical components that expose well-defined communication interfaces.* |
|  | * [Cloud computing](https://en.wikipedia.org/wiki/Cloud_computing) patterns * Micro Services * ROA (REST Oriented Architecture) * SAAS * distributed internet architecture |  |

<http://www.slideshare.net/arslantumbin/software-architecture-vs-design>

Software Architecture 
Software Architecture vs. Software Design 
All architecture is design, but not all design is archit...

# **Software Architecture**

Please refer to Microsoft Application Architecture Guide, 2nd Edition

## Architecture styles/Patterns

#### Combining Architectural Styles

The architecture of a software system is almost never limited to a single architectural style, but is often a combination of architectural styles that make up the complete system. For example, you might have a SOA design composed of services developed using a layered architecture approach and an object-oriented architecture style.

A combination of architecture styles is also useful if you are building a public facing Web application, where you can achieve effective separation of concerns by using the layered architecture style. This will separate your presentation logic from your business logic and your data access logic. Your organization's security requirements might force you to deploy the application using either the 3-tier deployment approach, or a deployment of more than three tiers. The presentation tier may be deployed to the perimeter network, which sits between an organization's internal network and an external network. On your presentation tier, you may decide to use a separated presentation pattern (a type of layered design style), such as Model-View-Controller (MVC), for your interaction model. You might also choose a SOA architecture style, and implement message-based communication, between your Web server and application server.

If you are building a desktop application, you may have a client that sends requests to a program on the server. In this case, you might deploy the client and server using the client/server architecture style, and use the component-based architecture style to decompose the design further into independent components that expose the appropriate communication interfaces. Using the object-oriented design approach for these components will improve reuse, testability, and flexibility.

Many factors will influence the architectural styles you choose. These factors include the capacity of your organization for design and implementation; the capabilities and experience of your developers; and your infrastructure and organizational constraints. The following sections will help you to determine the appropriate styles for your applications.

# **Software Design**

Software design is the process of defining software methods, functions, objects, and the overall structure and interaction of your code so that the resulting functionality will satisfy your user’s requirements

Book: Agile Software Development: Principles, Patterns, and Practices.pdf

## Software Design Approaches

* Top Down
* Bottom-UP

## Software Design Principles

Software design principles represent a set of guidelines that helps us to avoid having a bad design. The software designs helps to make a software well designed, well read and best maintainable

### SOLID (object-oriented design)

#### S: Single Responsibility Principle (SRP):

* A class should have only one reason to change.

#### O: Open closed Principle (OSP)

* *Software entities like classes, modules and functions should be open for extension but closed for modifications*.
* Abstract Classes and concrete classes for implementing their behavior.
* Template Pattern and Strategy Pattern.

#### L: Liskov substitution Principle (LSP)

* The new derived classes should be able to replace the base classes without any change in the code

#### I: Interface Segregation Principle (ISP)

* Clients should not be forced to depend upon interfaces that they don't use.

#### D: Dependency Inversion Principle (DIP) - Inversion of Control

* we should decouple high level modules from low level modules, introducing an abstraction layer between the high level classes and low level classes.
* Implemented by
  + Factories(3 Factory patterns)
    - Factory Method
    - Abstract Factory patterns
    - Simple Factory pattern
  + Service Locators
  + Dependency Injection
    - IoC/DI Container (Unity)
    - Constructor Injection
    - Property/Setter Injection
    - Interface-based injection
    - Service Locator Injection

<https://www.codeproject.com/Tips/654769/Software-Design-Principles>

<http://www.c-sharpcorner.com/UploadFile/damubetha/solid-principles-in-C-Sharp/>

<https://www.codeproject.com/Articles/703634/SOLID-architecture-principles-using-simple-Csharp>

### DRY principle

Don’t repeat yourself

### YAGNI principle

You aren’t going to need it. So don’t implement it

### KISS principle

Keep it Simple, Stupid

## Software Design Patterns

<https://en.wikipedia.org/wiki/Software_design_pattern>

Book: Professional ASP.NET Design Patterns pdf