**What is Spring MVC framework?**

The Spring web MVC framework provides [model-view-controller](http://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller) architecture and ready components that can be used to develop flexible and loosely coupled web applications. The MVC pattern results in separating the different aspects of the application (input logic, business logic, and UI logic), while providing a loose coupling between model, view and controller parts of application. Spring framework provides lots of advantages over other MVC frameworks e.g.

1. Clear separation of roles – controller, validator, command object, form object, model object, DispatcherServlet, handler mapping, view resolver, etc. Each role can be fulfilled by a specialized object.
2. Powerful and straightforward configuration of both framework and application classes as JavaBeans.
3. Reusable business code – no need for duplication. You can use existing business objects as command or form objects instead of mirroring them in order to extend a particular framework base class.
4. Customizable binding and validation
5. Customizable handler mapping and view resolution
6. Customizable locale and theme resolution
7. A JSP form tag library, introduced in Spring 2.0, that makes writing forms in JSP pages much easier. etc.

**What is DispatcherServlet and ContextLoaderListener?**

Spring’s web MVC framework is, like many other web MVC frameworks, request-driven, designed around a central Servlet that handles all the HTTP requests and responses. Spring’s DispatcherServlet however, does more than just that. It is completely integrated with the Spring IoC container so it allows you to use every feature that Spring has.

After receiving an HTTP request, DispatcherServlet consults the HandlerMapping (configuration files) to call the appropriate Controller. The Controller takes the request and calls the appropriate service methods and set model data and then returns view name to the DispatcherServlet. The DispatcherServlet will take help from ViewResolver to pickup the defined view for the request. Once view is finalized, The DispatcherServlet passes the model data to the view which is finally rendered on the browser.

|  |
| --- |
| <web-app>    <display-name>Archetype Created Web Application</display-name>      <servlet>          <servlet-name>spring</servlet-name>              <servlet-class>                  org.springframework.web.servlet.DispatcherServlet              </servlet-class>          <load-on-startup>1</load-on-startup>      </servlet>        <servlet-mapping>          <servlet-name>spring</servlet-name>          <url-pattern>/</url-pattern>      </servlet-mapping>    </web-app> |

By default, DispatcherServlet loads its configuration file using <servlet\_name>-servlet.xml. E.g. with above web.xml file, DispatcherServlet will try to find spring-servlet.xml file in classpath.

ContextLoaderListener reads the spring configuration file (with value given against “**contextConfigLocation**” in web.xml), parse it and loads the beans defined in that config file. e.g.

|  |
| --- |
| <servlet>      <servlet-name>spring</servlet-name>      <servlet-class>          org.springframework.web.servlet.DispatcherServlet      </servlet-class>        <init-param>          <param-name>contextConfigLocation</param-name>          <param-value>/WEB-INF/applicationContext.xml</param-value>      </init-param>        <load-on-startup>1</load-on-startup>  </servlet> |

**What is the front controller class of Spring MVC?**

A front controller is defined as “a controller which handles all requests for a Web Application.” **DispatcherServlet (actually a servlet) is the front controller in Spring MVC that intercepts every request and then dispatches/forwards requests to an appropriate controller.**

When a web request is sent to a Spring MVC application, dispatcher servlet first receives the request. Then it organizes the different components configured in Spring’s web application context (e.g. actual request handler controller and view resolvers) or annotations present in the controller itself, all needed to handle the request.

**How to use Java based configuration?**

To configure java based MVC application, first add required dependencies.

|  |
| --- |
| <!-- Spring MVC support -->    <dependency>      <groupId>org.springframework</groupId>      <artifactId>spring-webmvc</artifactId>      <version>4.1.4.RELEASE</version>  </dependency>    <dependency>      <groupId>org.springframework</groupId>      <artifactId>spring-web</artifactId>      <version>4.1.4.RELEASE</version>  </dependency>    <!-- Tag libs support for view layer -->    <dependency>      <groupId>javax.servlet</groupId>      <artifactId>jstl</artifactId>      <version>1.2</version>      <scope>runtime</scope>  </dependency>    <dependency>      <groupId>taglibs</groupId>      <artifactId>standard</artifactId>      <version>1.1.2</version>      <scope>runtime</scope>  </dependency> |

Now add DispatcherServlet entry in web.xml file so that all incoming requests come though DispatcherServlet only.

|  |
| --- |
| <servlet>      <servlet-name>spring</servlet-name>          <servlet-class>              org.springframework.web.servlet.DispatcherServlet          </servlet-class>      <load-on-startup>1</load-on-startup>  </servlet>   <servlet-mapping>      <servlet-name>spring</servlet-name>      <url-pattern>/</url-pattern>  </servlet-mapping> |

Now add below entries in spring configuration file.

|  |
| --- |
| <beans>      <!-- Scan all classes in this path for spring specific annotations -->      <context:component-scan base-package="com.howtodoinjava.demo" />        <bean class="org.springframework.web.servlet.mvc.annotation.DefaultAnnotationHandlerMapping" />      <bean class="org.springframework.web.servlet.mvc.annotation.AnnotationMethodHandlerAdapter" />       <!-- Vierw resolver configuration -->      <bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">          <property name="prefix" value="/WEB-INF/views/" />          <property name="suffix" value=".jsp" />      </bean>    </beans> |

Add controller code.

|  |
| --- |
| @Controller  @RequestMapping("/employee-module")  public class EmployeeController  {      @Autowired      EmployeeManager manager;        @RequestMapping(value = "/getAllEmployees", method = RequestMethod.GET)      public String getAllEmployees(Model model)      {          model.addAttribute("employees", manager.getAllEmployees());          return "employeesListDisplay";      }  } |

Additionally you should add manager and dao layer classes as well. Finally you add the jsp file to display the view.

**What is MVC pattern?**

MVC is a design pattern called Model-View-Controller. It decouples data access logic from business logic.

**Model:**

The Model contains the core of the application's functionality. It encapsulates the state of the application. Sometimes the only functionality it contains is state. It knows nothing about the view or controller.

**View:**

The view provides the presentation of the model. It is the look and feel of the application. The view can access the model getters, but it has no knowledge of the setters. In addition, it knows nothing about the controller. The view should be notified when changes to the model occur.

Controller:

The controller reacts to the user input. It creates and sets the model and helps to identify which view should be part of response.

**What is Spring?**

Spring is a framework for developing enterprise JavaTM applications. The benefit of using Spring over other frameworks is that it’s open source. This means developers can build reusable code without any vendor lock-in. Another major advantage of Spring is the layered architecture that enables you to select only the components you need while offering a seamless J2EE application development framework.

**List the advantages of the Spring framework.**

Spring has the following advantages:

Layered architecture that allows you to use what you need while leaving what you don’t need.

Spring allows developers to focus on Plain Old Java Object (POJO) Programming. This allows for continuous testing and integration.

Being an open source, there is no vendor lock-in.

Dependency injection and inversion of control makes Java Database Connectivity (JDBC) simpler.

List some features of Spring.

**Container:**

Spring manages and contains the configuration and life cycle of application objects.

**Lightweight:**

When it comes to transparency and size, Spring is a lightweight application framework. The lightest version of the Spring framework only takes up 1MB. Additionally, the overhead in terms of processing is similarly minuscule.

**MVC Framework:**

Spring utilizes the model-view-controller (MVC) web application framework which is built on the core Spring functionality. This framework accommodates several view technologies such as JSP, Tiles, Velocity, POI, and iText. Additionally, it is highly configurable with the use of strategy interfaces. However, several other frameworks could be easily used in place of the Spring MVC Framework. You can learn more about other MVC frameworks from this course on ASP.NET MVC 4.

**Inversion Of Control (IOC):**

Spring achieves loose coupling through the use of Inversion of Control. Objects provide their dependencies rather than looking for or creating dependent objects.

**Transaction Management:**

A generic abstraction layer is supplied by the Spring framework for the purpose of transaction management. This allows developers to include pluggable transaction managers while making it very easy to separate transactions while avoiding low-level issues. This transaction support is not connected with the J2REE environment. Also, it can be utilized in containerless environments.

**Aspect Oriented Programming (AOP):**

By separating system services from application business logic, Spring supports aspect oriented programming. This also allows for cohesive deployment of applications.

**JDBC Exception Handling:**

The Java Database Connectivity (JDBC) abstraction layer of Spring provides a useful exception hierarchy. This makes error handling strategy much easier to develop. Additionally, Spring offers great integration services with JDO, iBATIS, and Hibernate.

**What is Dependency Injection (AKA IOC)?**

Dependency Injection or Inversion of Control (IOC), at the most basic level, allows you to describe how objects should be created rather than creating them directly. That is, you describe the services that are needed by different components using a configuration file rather than directly connecting these components and services in code. In the case of the Spring framework, these services and components are then connected by the IOC container.

For example, objects are given their dependencies when they are created by an external process that manages each object within a system. In other words, the dependencies are inserted into objects. This is contrary from the way dependencies are handled in other frameworks, which is why the term inversion of control was coined. This signifies an inversion of responsibility for creating references to dependencies within objects. Here’s a course that can teach you more about the basics of Spring 3.2 Framework.

List some different types of Dependency Injection (IOC)

The three (3) types of dependency injection, or IOC, are:

**Setter Injection** (used by Spring): JavaBeans properties are used to assign dependencies.

**Interface Injection** (used by Avalon): An interface is used for injection.

**Constructor Injection** (used by Spring, Pico container, and others): Constructor parameters are used to provide dependencies.

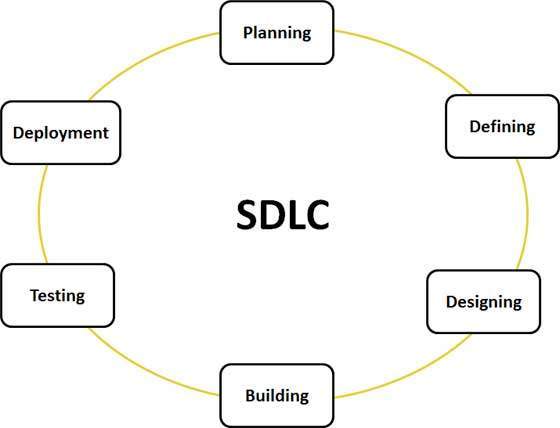
SDLC, Software Development Life Cycle is a process used by software industry to design, develop and test high quality softwares. The SDLC aims to produce a high quality software that meets or exceeds customer expectations, reaches completion within times and cost estimates.

* SDLC is the acronym of Software Development Life Cycle.
* It is also called as Software development process.
* The software development life cycle (SDLC) is a framework defining tasks performed at each step in the software development process.
* ISO/IEC 12207 is an international standard for software life-cycle processes. It aims to be the standard that defines all the tasks required for developing and maintaining software.

What is SDLC?

SDLC is a process followed for a software project, within a software organization. It consists of a detailed plan describing how to develop, maintain, replace and alter or enhance specific software. The life cycle defines a methodology for improving the quality of software and the overall development process.

The following figure is a graphical representation of the various stages of a typical SDLC.



A typical Software Development life cycle consists of the following stages:

Stage 1: Planning and Requirement Analysis

Requirement analysis is the most important and fundamental stage in SDLC. It is performed by the senior members of the team with inputs from the customer, the sales department, market surveys and domain experts in the industry. This information is then used to plan the basic project approach and to conduct product feasibility study in the economical, operational, and technical areas.

Planning for the quality assurance requirements and identification of the risks associated with the project is also done in the planning stage. The outcome of the technical feasibility study is to define the various technical approaches that can be followed to implement the project successfully with minimum risks.

Stage 2: Defining Requirements

Once the requirement analysis is done the next step is to clearly define and document the product requirements and get them approved from the customer or the market analysts. This is done through .SRS. . Software Requirement Specification document which consists of all the product requirements to be designed and developed during the project life cycle.

Stage 3: Designing the product architecture

SRS is the reference for product architects to come out with the best architecture for the product to be developed. Based on the requirements specified in SRS, usually more than one design approach for the product architecture is proposed and documented in a DDS - Design Document Specification.

This DDS is reviewed by all the important stakeholders and based on various parameters as risk assessment, product robustness, design modularity , budget and time constraints , the best design approach is selected for the product.

A design approach clearly defines all the architectural modules of the product along with its communication and data flow representation with the external and third party modules (if any). The internal design of all the modules of the proposed architecture should be clearly defined with the minutest of the details in DDS.

Stage 4: Building or Developing the Product

In this stage of SDLC the actual development starts and the product is built. The programming code is generated as per DDS during this stage. If the design is performed in a detailed and organized manner, code generation can be accomplished without much hassle.

Developers have to follow the coding guidelines defined by their organization and programming tools like compilers, interpreters, debuggers etc are used to generate the code. Different high level programming languages such as C, C++, Pascal, Java, and PHP are used for coding. The programming language is chosen with respect to the type of software being developed.

Stage 5: Testing the Product

This stage is usually a subset of all the stages as in the modern SDLC models, the testing activities are mostly involved in all the stages of SDLC. However this stage refers to the testing only stage of the product where products defects are reported, tracked, fixed and retested, until the product reaches the quality standards defined in the SRS.

Stage 6: Deployment in the Market and Maintenance

Once the product is tested and ready to be deployed it is released formally in the appropriate market. Sometime product deployment happens in stages as per the organizations. business strategy. The product may first be released in a limited segment and tested in the real business environment (UAT- User acceptance testing).

Then based on the feedback, the product may be released as it is or with suggested enhancements in the targeting market segment. After the product is released in the market, its maintenance is done for the existing customer base.

SDLC Models

There are various software development life cycle models defined and designed which are followed during software development process. These models are also referred as "Software Development Process Models". Each process model follows a Series of steps unique to its type, in order to ensure success in process of software development.

Following are the most important and popular SDLC models followed in the industry:

* Waterfall Model
* Iterative Model
* Spiral Model
* V-Model
* Big Bang Model

The other related methodologies are Agile Model, RAD Model, Rapid Application Development and Prototyping Models.

The Waterfall Model was first Process Model to be introduced. It is also referred to as a linear-sequential life cycle model. It is very simple to understand and use. In a waterfall model, each phase must be completed before the next phase can begin and there is no overlapping in the phases.

Waterfall model is the earliest SDLC approach that was used for software development .

The waterfall Model illustrates the software development process in a linear sequential flow; hence it is also referred to as a linear-sequential life cycle model. This means that any phase in the development process begins only if the previous phase is complete. In waterfall model phases do not overlap.

## Waterfall Model design

Waterfall approach was first SDLC Model to be used widely in Software Engineering to ensure success of the project. In "The Waterfall" approach, the whole process of software development is divided into separate phases. In Waterfall model, typically, the outcome of one phase acts as the input for the next phase sequentially.

Following is a diagrammatic representation of different phases of waterfall model.



The sequential phases in Waterfall model are:

* **Requirement Gathering and analysis:** All possible requirements of the system to be developed are captured in this phase and documented in a requirement specification doc.
* **System Design:** The requirement specifications from first phase are studied in this phase and system design is prepared. System Design helps in specifying hardware and system requirements and also helps in defining overall system architecture.
* **Implementation:** With inputs from system design, the system is first developed in small programs called units, which are integrated in the next phase. Each unit is developed and tested for its functionality which is referred to as Unit Testing.
* **Integration and Testing:** All the units developed in the implementation phase are integrated into a system after testing of each unit. Post integration the entire system is tested for any faults and failures.
* **Deployment of system:** Once the functional and non functional testing is done, the product is deployed in the customer environment or released into the market.
* **Maintenance:** There are some issues which come up in the client environment. To fix those issues patches are released. Also to enhance the product some better versions are released. Maintenance is done to deliver these changes in the customer environment.

All these phases are cascaded to each other in which progress is seen as flowing steadily downwards (like a waterfall) through the phases. The next phase is started only after the defined set of goals are achieved for previous phase and it is signed off, so the name "Waterfall Model". In this model phases do not overlap.

## Waterfall Model Application

Every software developed is different and requires a suitable SDLC approach to be followed based on the internal and external factors. Some situations where the use of Waterfall model is most appropriate are:

* Requirements are very well documented, clear and fixed.
* Product definition is stable.
* Technology is understood and is not dynamic.
* There are no ambiguous requirements.
* Ample resources with required expertise are available to support the product.
* The project is short.

## Waterfall Model Pros & Cons

### Advantage

The advantage of waterfall development is that it allows for departmentalization and control. A schedule can be set with deadlines for each stage of development and a product can proceed through the development process model phases one by one.

Development moves from concept, through design, implementation, testing, installation, troubleshooting, and ends up at operation and maintenance. Each phase of development proceeds in strict order.

### Disadvantage

The disadvantage of waterfall development is that it does not allow for much reflection or revision. Once an application is in the testing stage, it is very difficult to go back and change something that was not well-documented or thought upon in the concept stage.

The following table lists out the pros and cons of Waterfall model:

|  |  |
| --- | --- |
| **Pros** | **Cons** |
| * Simple and easy to understand and use * Easy to manage due to the rigidity of the model . each phase has specific deliverables and a review process. * Phases are processed and completed one at a time. * Works well for smaller projects where requirements are very well understood. * Clearly defined stages. * Well understood milestones. * Easy to arrange tasks. * Process and results are well documented. | * No working software is produced until late during the life cycle. * High amounts of risk and uncertainty. * Not a good model for complex and object-oriented projects. * Poor model for long and ongoing projects. * Not suitable for the projects where requirements are at a moderate to high risk of changing. So risk and uncertainty is high with this process model. * It is difficult to measure progress within stages. * Cannot accommodate changing requirements. * Adjusting scope during the life cycle can end a project. * Integration is done as a "big-bang. at the very end, which doesn't allow identifying any technological or business bottleneck or challenges early. |

# SDLC - Iterative Model

Advertisements

[Previous Page](https://www.tutorialspoint.com/sdlc/sdlc_waterfall_model.htm)

[Next Page](https://www.tutorialspoint.com/sdlc/sdlc_spiral_model.htm)

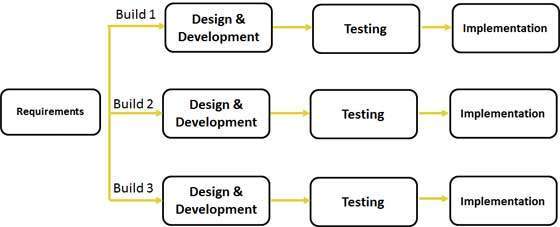
In Iterative model, iterative process starts with a simple implementation of a small set of the software requirements and iteratively enhances the evolving versions until the complete system is implemented and ready to be deployed.

An iterative life cycle model does not attempt to start with a full specification of requirements. Instead, development begins by specifying and implementing just part of the software, which is then reviewed in order to identify further requirements. This process is then repeated, producing a new version of the software at the end of each iteration of the model.

## Iterative Model design

Iterative process starts with a simple implementation of a subset of the software requirements and iteratively enhances the evolving versions until the full system is implemented. At each iteration, design modifications are made and new functional capabilities are added. The basic idea behind this method is to develop a system through repeated cycles (iterative) and in smaller portions at a time (incremental).

Following is the pictorial representation of Iterative and Incremental model:



Iterative and Incremental development is a combination of both iterative design or iterative method and incremental build model for development. "During software development, more than one iteration of the software development cycle may be in progress at the same time." and "This process may be described as an "evolutionary acquisition" or "incremental build" approach."

In incremental model the whole requirement is divided into various builds. During each iteration, the development module goes through the requirements, design, implementation and testing phases. Each subsequent release of the module adds function to the previous release. The process continues till the complete system is ready as per the requirement.

The key to successful use of an iterative software development lifecycle is rigorous validation of requirements, and verification & testing of each version of the software against those requirements within each cycle of the model. As the software evolves through successive cycles, tests have to be repeated and extended to verify each version of the software.

## Iterative Model Application

Like other SDLC models, Iterative and incremental development has some specific applications in the software industry. This model is most often used in the following scenarios:

* Requirements of the complete system are clearly defined and understood.
* Major requirements must be defined; however, some functionalities or requested enhancements may evolve with time.
* There is a time to the market constraint.
* A new technology is being used and is being learnt by the development team while working on the project.
* Resources with needed skill set are not available and are planned to be used on contract basis for specific iterations.
* There are some high risk features and goals which may change in the future.

## Iterative Model Pros and Cons

The advantage of this model is that there is a working model of the system at a very early stage of development which makes it easier to find functional or design flaws. Finding issues at an early stage of development enables to take corrective measures in a limited budget.

The disadvantage with this SDLC model is that it is applicable only to large and bulky software development projects. This is because it is hard to break a small software system into further small serviceable increments/modules.

The following table lists out the pros and cons of Iterative and Incremental SDLC Model:

|  |  |
| --- | --- |
| **Pros** | **Cons** |
| * Some working functionality can be developed quickly and early in the life cycle. * Results are obtained early and periodically. * Parallel development can be planned. * Progress can be measured. * Less costly to change the scope/requirements. * Testing and debugging during smaller iteration is easy. * Risks are identified and resolved during iteration; and each iteration is an easily managed milestone. * Easier to manage risk - High risk part is done first. * With every increment operational product is delivered. * Issues, challenges & risks identified from each increment can be utilized/applied to the next increment. * Risk analysis is better. * It supports changing requirements. * Initial Operating time is less. * Better suited for large and mission-critical projects. * During life cycle software is produced early which facilitates customer evaluation and feedback. | * More resources may be required. * Although cost of change is lesser but it is not very suitable for changing requirements. * More management attention is required. * System architecture or design issues may arise because not all requirements are gathered in the beginning of the entire life cycle. * Defining increments may require definition of the complete system. * Not suitable for smaller projects. * Management complexity is more. * End of project may not be known which is a risk. * Highly skilled resources are required for risk analysis. * Project.s progress is highly dependent upon the risk analysis phase. |

# SDLC - Agile Model

Advertisements

[Previous Page](https://www.tutorialspoint.com/sdlc/sdlc_bigbang_model.htm)

[Next Page](https://www.tutorialspoint.com/sdlc/sdlc_rad_model.htm)

Agile SDLC model is a combination of iterative and incremental process models with focus on process adaptability and customer satisfaction by rapid delivery of working software product.

Agile Methods break the product into small incremental builds. These builds are provided in iterations. Each iteration typically lasts from about one to three weeks. Every iteration involves cross functional teams working simultaneously on various areas like planning, requirements analysis, design, coding, unit testing, and acceptance testing.

At the end of the iteration a working product is displayed to the customer and important stakeholders.

## What is Agile?

Agile model believes that every project needs to be handled differently and the existing methods need to be tailored to best suit the project requirements. In agile the tasks are divided to time boxes (small time frames) to deliver specific features for a release.

Iterative approach is taken and working software build is delivered after each iteration. Each build is incremental in terms of features; the final build holds all the features required by the customer.

Here is a graphical illustration of the Agile Model:



Agile thought process had started early in the software development and started becoming popular with time due to its flexibility and adaptability.

The most popular agile methods include Rational Unified Process (1994), Scrum (1995), Crystal Clear, Extreme Programming (1996), Adaptive Software Development, Feature Driven Development, and Dynamic Systems Development Method (DSDM) (1995). These are now collectively referred to as agile methodologies, after the Agile Manifesto was published in 2001.

Following are the Agile Manifesto principles

* **Individuals and interactions** - in agile development, self-organization and motivation are important, as are interactions like co-location and pair programming.
* **Working software** - Demo working software is considered the best means of communication with the customer to understand their requirement, instead of just depending on documentation.
* **Customer collaboration** - As the requirements cannot be gathered completely in the beginning of the project due to various factors, continuous customer interaction is very important to get proper product requirements.
* **Responding to change** - agile development is focused on quick responses to change and continuous development.

## Agile Vs Traditional SDLC Models

Agile is based on the adaptive software development methods where as the traditional SDLC models like waterfall model is based on predictive approach.

Predictive teams in the traditional SDLC models usually work with detailed planning and have a complete forecast of the exact tasks and features to be delivered in the next few months or during the product life cycle. Predictive methods entirely depend on the requirement analysis and planning done in the beginning of cycle. Any changes to be incorporated go through a strict change control management and prioritization.

Agile uses adaptive approach where there is no detailed planning and there is clarity on future tasks only in respect of what features need to be developed. There is feature driven development and the team adapts to the changing product requirements dynamically. The product is tested very frequently, through the release iterations, minimizing the risk of any major failures in future.

Customer interaction is the backbone of Agile methodology, and open communication with minimum documentation are the typical features of Agile development environment. The agile teams work in close collaboration with each other and are most often located in the same geographical location.

## Agile Model Pros and Cons

Agile methods are being widely accepted in the software world recently, however, this method may not always be suitable for all products. Here are some pros and cons of the agile model.

Following table lists out the pros and cons of Agile Model:

|  |  |
| --- | --- |
| **Pros** | **Cons** |
| * Is a very realistic approach to software development * Promotes teamwork and cross training. * Functionality can be developed rapidly and demonstrated. * Resource requirements are minimum. * Suitable for fixed or changing requirements * Delivers early partial working solutions. * Good model for environments that change steadily. * Minimal rules, documentation easily employed. * Enables concurrent development and delivery within an overall planned context. * Little or no planning required * Easy to manage * Gives flexibility to developers | * Not suitable for handling complex dependencies. * More risk of sustainability, maintainability and extensibility. * An overall plan, an agile leader and agile PM practice is a must without which it will not work. * Strict delivery management dictates the scope, functionality to be delivered, and adjustments to meet the deadlines. * Depends heavily on customer interaction, so if customer is not clear, team can be driven in the wrong direction. * There is very high individual dependency, since there is minimum documentation generated. * Transfer of technology to new team members may be quite challenging due to lack of documentation. |