Software Development Life Cycle (SDLC)

SOFTWARE DEVELOPMENT LIFE CYCLE (SDLC)

Simply Easy Learning by tutorialspoint.com

ABOUT THE TUTORIAL

SDLC Tutorial

SDLC stands for Software Development Life Cycle. SDLC is the process consisting of a series of planned activities to develop or alter the software products.

This tutorial will give you an overview of the SDLC basics, SDLC models available and their application in the industry. This tutorial also elaborates on the other related methodologies like Agile, RAD and Prototyping.

Audience

The SDLC tutorial is relevant to all software professionals contributing in any manner to the Software product development and its release. It is a handy reference for the quality stakeholders of a Software project and the program/project managers. By the end of this tutorial the readers will develop sound understanding of the SDLC and related concepts and will be in a position to select and follow the right model for a given Software project.

Prerequisites

There are no specific prerequisites for the SDLC tutorial and any software professional can go through this tutorial to get a bigger picture of how high quality software applications and products are designed. A good understanding of programming or testing or project management will give you an added advantage and help you gain maximum from this tutorial.

Copyright & Disclaimer Notice

Table of Contents

SDLC Tutorial i
Audience i
Prerequisites i
Copyright & Disclaimer Notice i
SDLC Overview 1
What is SDLC? 1
SDLC Models 3
Waterfall Model 4
Waterfall Model design 4
Waterfall Model Application 5
Waterfall Model Pros & Cons 6
Iterative Model 7
Iterative Model design 7
Iterative Model Application 8
Iterative Model Pros and Cons 8
Spiral Model 10
Spiral Model design 10
Spiral Model Application 12
Spiral Model Pros and Cons 12
V - Model 13
V- Model design 13
V- Model Application 15
V- Model Pros and Cons 15
Big Bang Model 17
Big Bang Model design and Application 17
Big Bang Model Pros and Cons 17
Agile Model 19
What is Agile? 19
Agile Vs Traditional SDLC Models 20
Agile Model Pros and Cons 21
RAD Overview 22
What is RAD? 22

RAD Model Design 22
RAD Model Vs Traditional SDLC 24
RAD Model Application 24
RAD Model Pros and Cons 24
Software Prototyping Overview 26
What is Software Prototyping? 26
Software Prototyping Types 27
Software Prototyping Application 28
Software Prototyping Pros and Cons 28
SDLC Summary 29
References 30
Books: 30
Websites: 30

CHAPTER

1

SDLC Overview

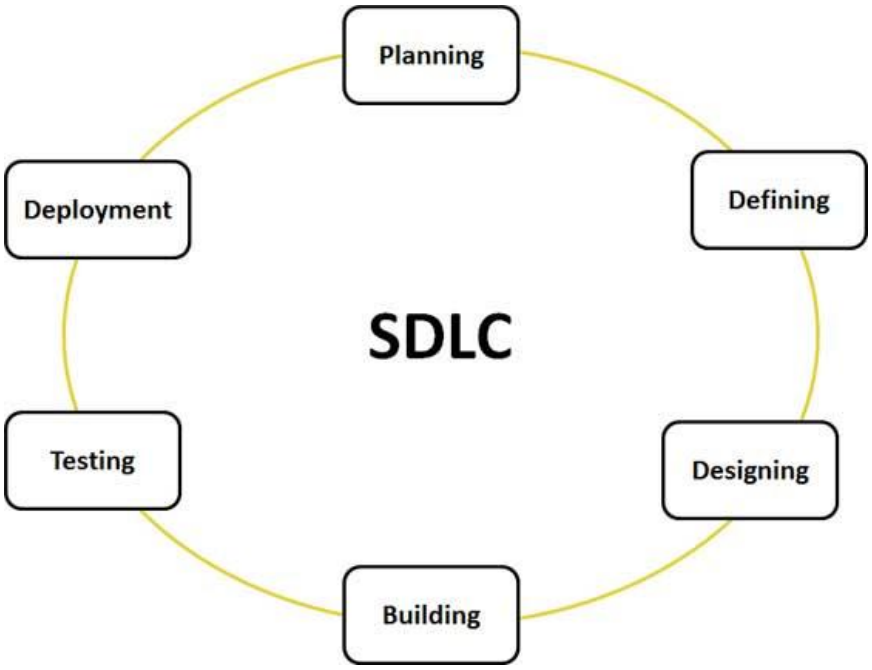
SDLC, Software Development Life Cycle is a process used by software industry to design, develop and test high quality softwares. The SDLC aims to produce a high quality software that meets or exceeds customer expectations, reaches completion within times and cost estimates.

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.

CHAPTER

2

Waterfall Model

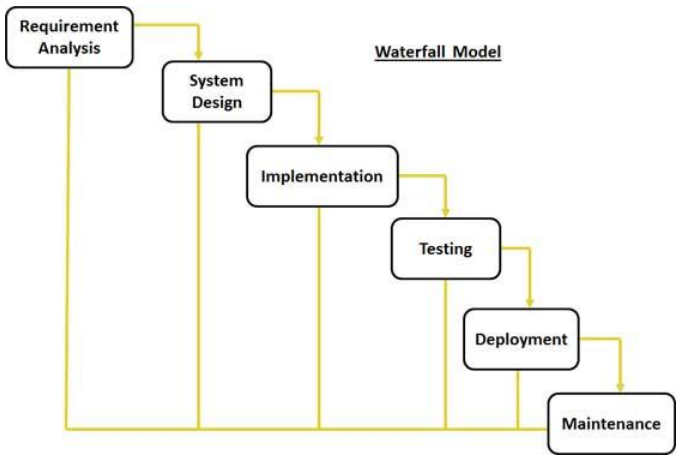
The Waterfall Model was first Process Model to be introduced. It is also referred to as a linear-sequential life cycle model. It is very simple to understand and use. In a waterfall model, each phase must be completed before the next phase can begin and there is no overlapping in the phases.

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

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.

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. ▪ No working software is produced until late in the life cycle. ▪ 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.

CHAPTER

3

Iterative Model

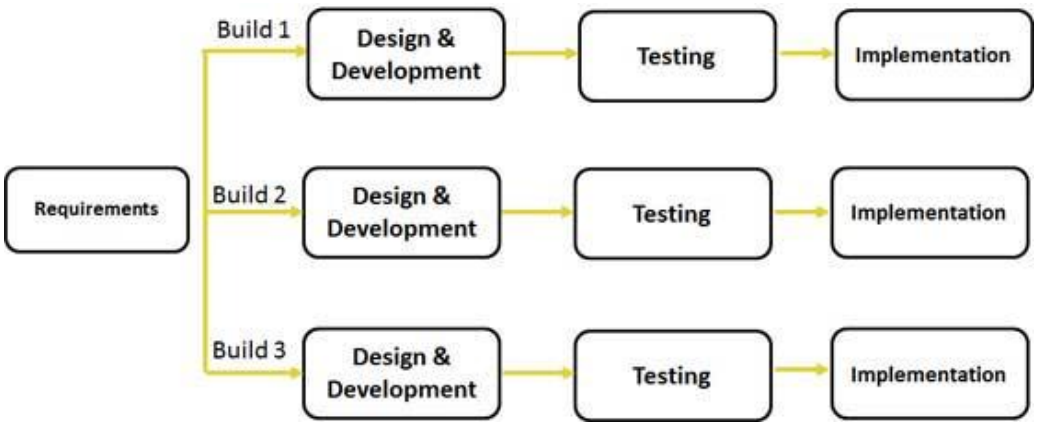
In Iterative model, it*erative process starts with a simple implementation of a small set of the software requirements and iteratively enhances the evolving versions until the complete system is implemented and ready to be deployed.*

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.

CHAPTER

4

Spiral Model

The spiral model combines the idea of iterative development with the systematic, controlled aspects of the waterfall model.

Spiral model is a combination of iterative development process model and sequential linear development model i.e. waterfall model with very high emphasis on risk analysis. It allows for incremental releases of the product, or incremental refinement through each iteration around the spiral.

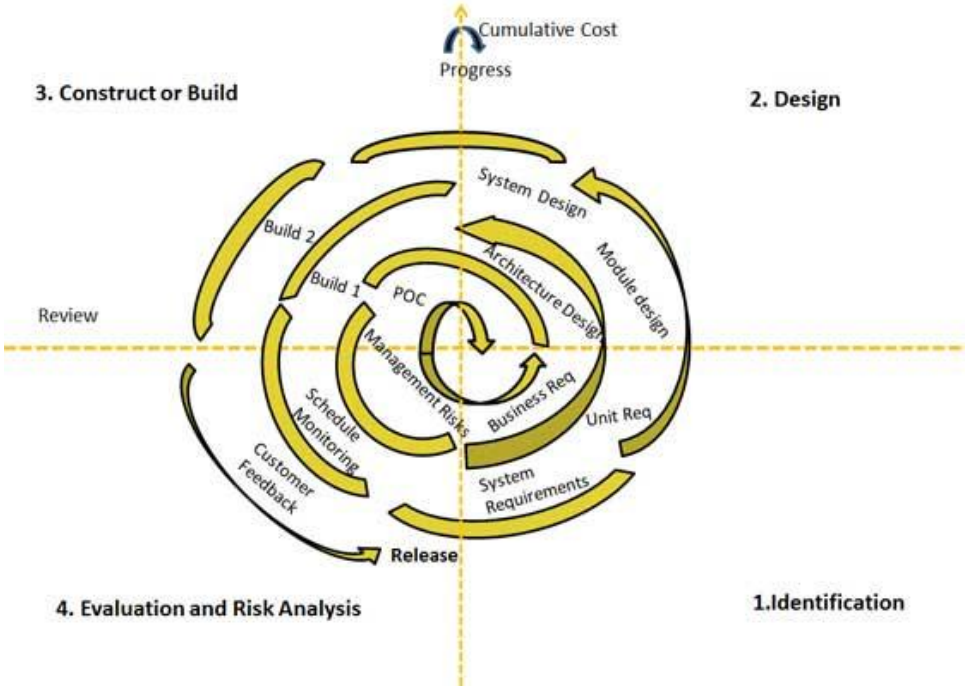
Spiral Model design

The spiral model has four phases. A software project repeatedly passes through these phases in iterations called Spirals.

- **Identification**
  This phase starts with gathering the business requirements in the baseline spiral. In the subsequent spirals as the product matures, identification of system requirements, subsystem requirements and unit requirements are all done in this phase.

  This also includes understanding the system requirements by continuous communication between the customer and the system analyst. At the end of the spiral the product is deployed in the identified market.

Following is a diagrammatic representation of spiral model listing the activities in each phase



- **Design**
  Design phase starts with the conceptual design in the baseline spiral and involves architectural design, logical design of modules, physical product design and final design in the subsequent spirals.

- **Construct or Build**
  Construct phase refers to production of the actual software product at every spiral. In the baseline spiral when the product is just thought of and the design is being developed a POC (Proof of Concept) is developed in this phase to get customer feedback.

  Then in the subsequent spirals with higher clarity on requirements and design details a working model of the software called build is produced with a version number. These builds are sent to customer for feedback.

- **Evaluation and Risk Analysis**
  Risk Analysis includes identifying, estimating, and monitoring technical feasibility and management risks, such as schedule slippage and cost overrun. After testing the build, at the end of first iteration, the customer evaluates the software and provides feedback.

  Based on the customer evaluation, software development process enters into the next iteration and subsequently follows the linear approach to implement the feedback suggested by the customer. The process of iterations along the spiral continues throughout the life of the software.

Spiral Model Application

Spiral Model is very widely used in the software industry as it is in synch with the natural development process of any product i.e. learning with maturity and also involves minimum risk for the customer as well as the development firms. Following are the typical uses of Spiral model:

- When costs there is a budget constraint and risk evaluation is important
- For medium to high-risk projects
- Long-term project commitment because of potential changes to economic priorities as the requirements change with time
- Customer is not sure of their requirements which is usually the case
- Requirements are complex and need evaluation to get clarity
- New product line which should be released in phases to get enough customer feedback
- Significant changes are expected in the product during the development cycle

Spiral Model Pros and Cons

The advantage of spiral lifecycle model is that it allows for elements of the product to be added in when they become available or known. This assures that there is no conflict with previous requirements and design. This method is consistent with approaches that have multiple software builds and releases and allows for making an orderly transition to a maintenance activity. Another positive aspect is that the spiral model forces early user involvement in the system development effort.

On the other side, it takes very strict management to complete such products and there is a risk of running the spiral in indefinite loop. So the discipline of change and the extent of taking change requests is very important to develop and deploy the product successfully.

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

Pros	Cons
▪ Changing requirements can be accommodated. ▪ Allows for extensive use of prototypes ▪ Requirements can be captured more accurately. ▪ Users see the system early. ▪ Development can be divided into smaller parts and more risky parts can be developed earlier which helps better risk management.	▪ Management is more complex. ▪ End of project may not be known early. ▪ Not suitable for small or low risk projects and could be expensive for small projects. ▪ Process is complex ▪ Spiral may go indefinitely. ▪ Large number of intermediate stages requires excessive documentation.

CHAPTER

5

V - Model

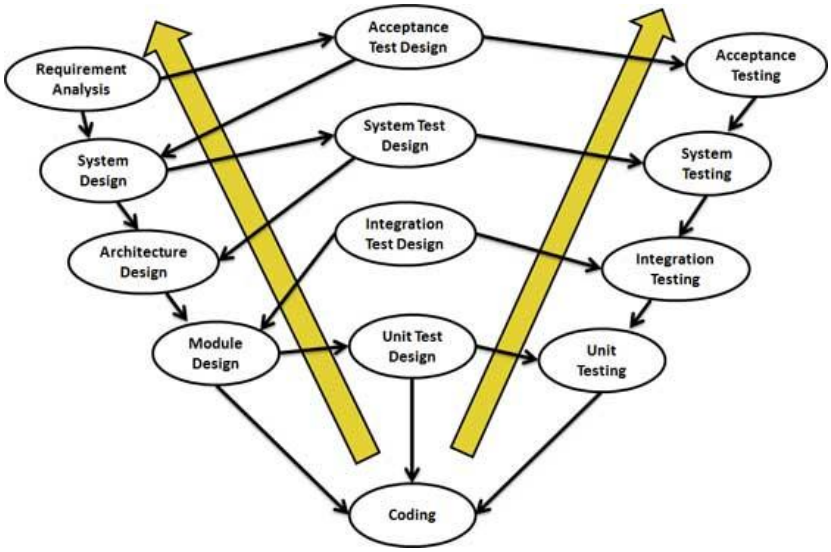
The V- model is SDLC model where execution of processes happens in a sequential manner in V-shape. It is also known as Verification and Validation model.

V-Model is an extension of the waterfall model and is based on association of a testing phase for each corresponding development stage. This means that for every single phase in the development cycle there is a directly associated testing phase. This is a highly disciplined model and next phase starts only after completion of the previous phase.

V- Model design

Under V-Model, the corresponding testing phase of the development phase is planned in parallel. So there are Verification phases on one side of the 'V' and Validation phases on the other side. Coding phase joins the two sides of the V-Model.

The below figure illustrates the different phases in V-Model of SDLC.



Verification Phases

Following are the Verification phases in V-Model:

- **Business Requirement Analysis :**
  This is the first phase in the development cycle where the product requirements are understood from the customer perspective. This phase involves detailed communication with the customer to understand his expectations and exact requirement. This is a very important activity and need to be managed well, as most of the customers are not sure about what exactly they need. The acceptance test design planning is done at this stage as business requirements can be used as an input for acceptance testing.

- **System Design:**
  Once you have the clear and detailed product requirements, it's time to design the complete system. System design would comprise of understanding and detailing the complete hardware and communication setup for the product under development. System test plan is developed based on the system design. Doing this at an earlier stage leaves more time for actual test execution later.

- **Architectural Design:**
  Architectural specifications are understood and designed in this phase. Usually more than one technical approach is proposed and based on the technical and financial feasibility the final decision is taken. System design is broken down further into modules taking up different functionality. This is also referred to as High Level Design (HLD).

  The data transfer and communication between the internal modules and with the outside world (other systems) is clearly understood and defined in this stage. With this information, integration tests can be designed and documented during this stage.

- **Module Design:**
  In this phase the detailed internal design for all the system modules is specified, referred to as Low Level Design (LLD). It is important that the design is compatible with the other modules in the system architecture and the other external systems. Unit tests are an essential part of any development process and helps eliminate the maximum faults and errors at a very early stage. Unit tests can be designed at this stage based on the internal module designs.

Coding Phase

The actual coding of the system modules designed in the design phase is taken up in the Coding phase. The best suitable programming language is decided based on the system and architectural requirements. The coding is performed based on the coding guidelines and standards. The code goes through numerous code reviews and is optimized for best performance before the final build is checked into the repository.

Validation Phases

Following are the Validation phases in V-Model:

- **Unit Testing**
  Unit tests designed in the module design phase are executed on the code during this validation phase. Unit testing is the testing at code level and helps eliminate bugs at an early stage, though all defects cannot be uncovered by unit testing.

- **Integration Testing**
  Integration testing is associated with the architectural design phase. Integration tests are performed to test the coexistence and communication of the internal modules within the system.

- **System Testing**
  System testing is directly associated with the System design phase. System tests check the entire system functionality and the communication of the system under development with external systems. Most of the software and hardware compatibility issues can be uncovered during system test execution.

- **Acceptance Testing**
  Acceptance testing is associated with the business requirement analysis phase and involves testing the product in user environment. Acceptance tests uncover the compatibility issues with the other systems available in the user environment. It also discovers the non functional issues such as load and performance defects in the actual user environment.

V- Model Application

V- Model application is almost same as waterfall model, as both the models are of sequential type. Requirements have to be very clear before the project starts, because it is usually expensive to go back and make changes. This model is used in the medical development field, as it is strictly disciplined domain. Following are the suitable scenarios to use V-Model:

- Requirements are well defined, clearly documented and fixed.
- Product definition is stable.
- Technology is not dynamic and is well understood by the project team.
- There are no ambiguous or undefined requirements
- The project is short.

V- Model Pros and Cons

The advantage of V-Model is that it's very easy to understand and apply. The simplicity of this model also makes it easier to manage. The disadvantage is that the model is not flexible to changes and just in case there is a requirement change, which is very common in today's dynamic world, it becomes very expensive to make the change.

The following table lists out the pros and cons of V-Model:

Pros	Cons
• This is a highly disciplined model and Phases are completed one at a time. • Works well for smaller projects where requirements are very well understood. • Simple and easy to understand and use. • Easy to manage due to the	▪ High 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.

	rigidity of the model – each phase has specific deliverables and a review process.	▪ Once an application is in the testing stage, it is difficult to go back and change a functionality ▪ No working software is produced until late during the life cycle.

CHAPTER

6

Big Bang Model

The Big Bang model is SDLC model where there is no specific process followed. The development just starts with the required money and efforts as the input, and the output is the software developed which may or may not be as per customer requirement.

Big Bang Model is SDLC model where there is no formal development followed and very little planning is required. Even the customer is not sure about what exactly he wants and the requirements are implemented on the fly without much analysis. Usually this model is followed for small projects where the development teams are very small.

Big Bang Model design and Application

Big bang model comprises of focusing all the possible resources in software development and coding, with very little or no planning. The requirements are understood and implemented as they come. Any changes required may or may not need to revamp the complete software.

This model is ideal for small projects with one or two developers working together and is also useful for academic or practice projects. It's an ideal model for the product where requirements are not well understood and the final release date is not given.

Big Bang Model Pros and Cons

The advantage of Big Bang is that its very simple and requires very little or no planning. Easy to mange and no formal procedure are required.

However the Big Bang model is a very high risk model and changes in the requirements or misunderstood requirements may even lead to complete reversal or scraping of the project. It is ideal for repetitive or small projects with minimum risks.

Following table lists out the pros and cons of Big Bang Model

Pros	Cons

▪ This is a very simple model ▪ Little or no planning required ▪ Easy to manage ▪ Very few resources required ▪ Gives flexibility to developers ▪ Is a good learning aid for new comers or students	▪ Very High risk and uncertainty. ▪ Not a good model for complex and object-oriented projects. ▪ Poor model for long and ongoing projects. ▪ Can turn out to be very expensive if requirements are misunderstood

CHAPTER

7

Agile Model

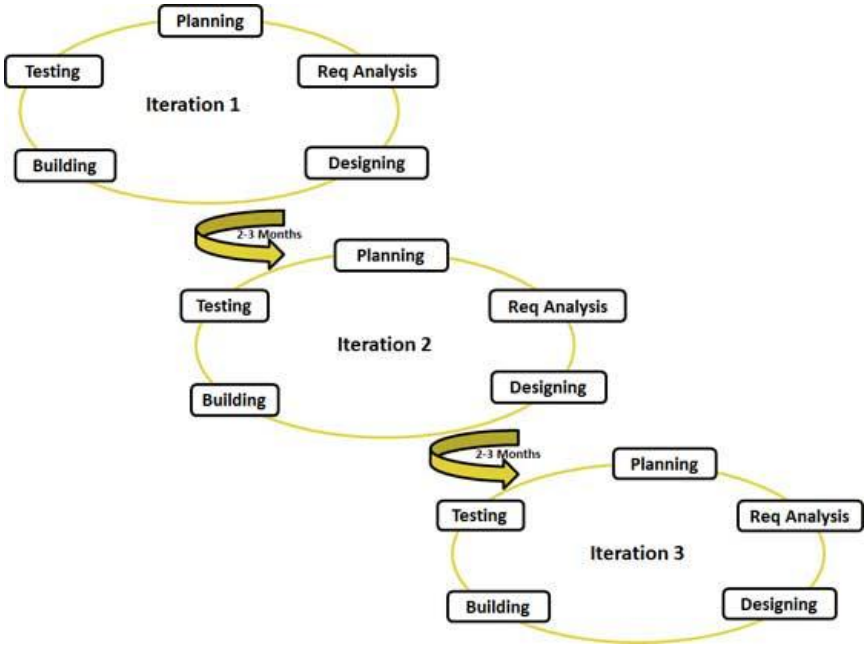
Agile SDLC model is a combination of iterative and incremental process models with focus on process adaptability and customer satisfaction by rapid delivery of working software product.

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.

CHAPTER

8

RAD Overview

The RAD (Rapid Application Development) model is based on prototyping and iterative development with no specific planning involved. The process of writing the software itself involves the planning required for developing the product.

Rapid Application development focuses on gathering customer requirements through workshops or focus groups, early testing of the prototypes by the customer using iterative concept, reuse of the existing prototypes (components), continuous integration and rapid delivery.

What is RAD?

Rapid application development (RAD) is a software development methodology that uses minimal planning in favor of rapid prototyping. A prototype is a working model that is functionally equivalent to a component of the product. In RAD model the functional modules are developed in parallel as prototypes and are integrated to make the complete product for faster product delivery.

Since there is no detailed preplanning, it makes it easier to incorporate the changes within the development process. RAD projects follow iterative and incremental model and have small teams comprising of developers, domain experts, customer representatives and other IT resources working progressively on their component or prototype. The most important aspect for this model to be successful is to make sure that the prototypes developed are reusable.

RAD Model Design

RAD model distributes the analysis, design, build, and test phases into a series of short, iterative development cycles. Following are the phases of RAD Model:

- **Business Modeling:**
  The business model for the product under development is designed in terms of flow of information and the distribution of information between various business channels. A complete business analysis is performed to find the vital information for business, how it can be obtained, how and when is the information processed and what are the factors driving successful flow of information.

- **Data Modeling:**

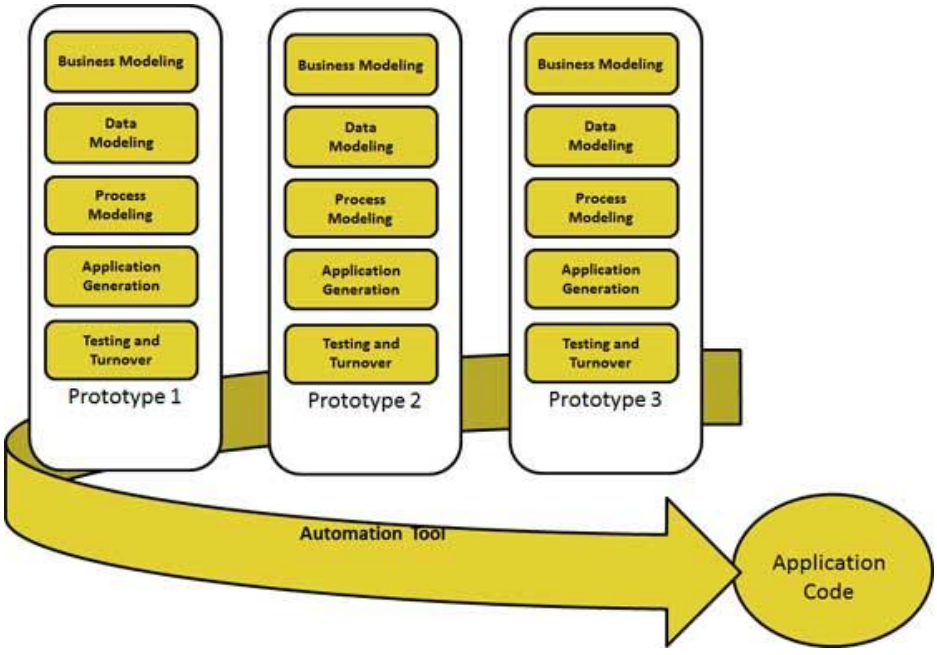
The information gathered in the Business Modeling phase is reviewed and analyzed to form sets of data objects vital for the business. The attributes of all data sets is identified and defined. The relation between these data objects are established and defined in detail in relevance to the business model.

- **Process Modeling:**
  The data object sets defined in the Data Modeling phase are converted to establish the business information flow needed to achieve specific business objectives as per the business model. The process model for any changes or enhancements to the data object sets is defined in this phase. Process descriptions for adding , deleting, retrieving or modifying a data object are given.

- **Application Generation:**
  The actual system is built and coding is done by using automation tools to convert process and data models into actual prototypes.

- **Testing and Turnover:**
  The overall testing time is reduced in RAD model as the prototypes are independently tested during every iteration. However the data flow and the interfaces between all the components need to be thoroughly tested with complete test coverage. Since most of the programming components have already been tested, it reduces the risk of any major issues.

Following image illustrates the RAD Model:



RAD Model Vs Traditional SDLC

The traditional SDLC follows a rigid process models with high emphasis on requirement analysis and gathering before the coding starts. It puts a pressure on the customer to sign off the requirements before the project starts and the customer doesn't get the feel of the product as there is no working build available for a long time.

The customer may need some changes after he actually gets to see the software, however the change process is quite rigid and it may not be feasible to incorporate major changes in the product in traditional SDLC.

RAD model focuses on iterative and incremental delivery of working models to the customer. This results in rapid delivery to the customer and customer involvement during the complete development cycle of product reducing the risk of non conformance with the actual user requirements.

RAD Model Application

RAD model can be applied successfully to the projects in which clear modularization is possible. If the project cannot be broken into modules, RAD may fail. Following are the typical scenarios where RAD can be used:

- RAD should be used only when a system can be modularized to be delivered in incremental manner.
- It should be used if there's high availability of designers for modeling
- It should be used only if the budget permits use of automated code generating tools.
- RAD SDLC model should be chosen only if domain experts are available with relevant business knowledge
- Should be used where the requirements change during the course of the project and working prototypes are to be presented to customer in small iterations of 2-3 months.

RAD Model Pros and Cons

RAD model enables rapid delivery as it reduces the overall development time due to reusability of the components and parallel development.

RAD works well only if high skilled engineers are available and the customer is also committed to achieve the targeted prototype in the given time frame. If there is commitment lacking on either side the model may fail.

Following table lists out the pros and cons of RAD Model

Pros	Cons
▪ Changing requirements can be accommodated. ▪ Progress can be measured. ▪ Iteration time can be short with use of powerful RAD tools. ▪ Productivity with fewer people in short time. ▪ Reduced development time. ▪ Increases reusability of components	▪ Dependency on technically strong team members for identifying business requirements. ▪ Only system that can be modularized can be built using RAD ▪ Requires highly skilled developers/designers. ▪ High dependency on modeling skills ▪ Inapplicable to cheaper projects as cost

▪ Quick initial reviews occur ▪ Encourages customer feedback ▪ Integration from very beginning solves a lot of integration issues.	of modeling and automated code generation is very high. ▪ Management complexity is more. ▪ Suitable for systems that are component based and scalable. ▪ Requires user involvement throughout the life cycle. ▪ Suitable for project requiring shorter development times.

CHAPTER

9

Software Prototyping Overview

The Software Prototyping refers to building software application prototypes which display the functionality of the product under development but may not actually hold the exact logic of the original software.

Software prototyping is becoming very popular as a software development model, as it enables to understand customer requirements at an early stage of development. It helps get valuable feedback from the customer and helps software designers and developers understand about what exactly is expected from the product under development.

What is Software Prototyping?

Prototype is a working model of software with some limited functionality. The prototype does not always hold the exact logic used in the actual software application and is an extra effort to be considered under effort estimation. Prototyping is used to allow the users evaluate developer proposals and try them out before implementation. It also helps understand the requirements which are user specific and may not have been considered by the developer during product design.

Following is the stepwise approach to design a software prototype:

- **Basic Requirement Identification**
  This step involves understanding the very basics product requirements especially in terms of user interface. The more intricate details of the internal design and external aspects like performance and security can be ignored at this stage.

- **Developing the initial Prototype**
  The initial Prototype is developed in this stage, where the very basic requirements are showcased and user interfaces are provided. These features may not exactly work in the same manner internally in the actual software developed and the workarounds are used to give the same look and feel to the customer in the prototype developed.

- **Review of the Prototype**
  The prototype developed is then presented to the customer and the other important stakeholders in the project. The feedback is collected in an organized manner and used for further enhancements in the product under development.

- **Revise and enhance the Prototype**
  The feedback and the review comments are discussed during this stage and some negotiations happen with the customer based on factors like , time and budget constraints and technical feasibility of actual implementation. The changes accepted are again incorporated in the new Prototype developed and the cycle repeats until customer expectations are met.

Prototypes can have horizontal or vertical dimensions. Horizontal prototype displays the user interface for the product and gives a broader view of the entire system, without concentrating on internal functions. A vertical prototype on the other side is a detailed elaboration of a specific function or a sub system in the product.

The purpose of both horizontal and vertical prototype is different. Horizontal prototypes are used to get more information on the user interface level and the business requirements. It can even be presented in the sales demos to get business in the market. Vertical prototypes are technical in nature and are used to get details of the exact functioning of the sub systems. For example, database requirements, interaction and data processing loads in a given sub system.

Software Prototyping Types

There are different types of software prototypes used in the industry. Following are the major software prototyping types used widely:

- **Throwaway/Rapid Prototyping**
  Throwaway prototyping is also called as rapid or close ended prototyping. This type of prototyping uses very little efforts with minimum requirement analysis to build a prototype. Once the actual requirements are understood, the prototype is discarded and the actual system is developed with a much clear understanding of user requirements.

- **Evolutionary Prototyping**
  Evolutionary prototyping also called as breadboard prototyping is based on building actual functional prototypes with minimal functionality in the beginning. The prototype developed forms the heart of the future prototypes on top of which the entire system is built. Using evolutionary prototyping only well understood requirements are included in the prototype and the requirements are added as and when they are understood.

- **Incremental Prototyping**
  Incremental prototyping refers to building multiple functional prototypes of the various sub systems and then integrating all the available prototypes to form a complete system.

- **Extreme Prototyping**
  Extreme prototyping is used in the web development domain. It consists of three sequential phases. First, a basic prototype with all the existing pages is presented in the html format. Then the data processing is simulated using a prototype services layer. Finally the services are implemented and integrated to the final prototype. This process is called Extreme Prototyping used to draw attention to the second phase of the process, where a fully functional UI is developed with very little regard to the actual services.

Software Prototyping Application

Software Prototyping is most useful in development of systems having high level of user interactions such as online systems. Systems which need users to fill out forms or go through various screens before data is processed can use prototyping very effectively to give the exact look and feel even before the actual software is developed.
Software that involves too much of data processing and most of the functionality is internal with very little user interface does not usually benefit from prototyping. Prototype development could be an extra overhead in such projects and may need lot of extra efforts.

Software Prototyping Pros and Cons

Software prototyping is used in typical cases and the decision should be taken very carefully so that the efforts spent in building the prototype add considerable value to the final software developed. The model has its own pros and cons discussed as below.

Following table lists out the pros and cons of Big Bang Model

Pros	Cons
▪ Increased user involvement in the product even before implementation ▪ Since a working model of the system is displayed, the users get a better understanding of the system being developed. ▪ Reduces time and cost as the defects can be detected much earlier. ▪ Quicker user feedback is available leading to better solutions. ▪ Missing functionality can be identified easily ▪ Confusing or difficult functions can be identified	▪ Risk of insufficient requirement analysis owing to too much dependency on prototype ▪ Users may get confused in the prototypes and actual systems. ▪ Practically, this methodology may increase the complexity of the system as scope of the system may expand beyond original plans. ▪ Developers may try to reuse the existing prototypes to build the actual system, even when its not technically feasible ▪ The effort invested in building prototypes may be too much if not monitored properly

CHAPTER

10

SDLC Summary

This was about the various SDLC models available and the scenarios in which these SDLC models are used. The information in this tutorial will help the project managers decide what SDLC model would be suitable for their project and it would also help the developers and testers understand basics of the development model being used for their project.

We have discussed all the popular SDLC models in the industry, both traditional and Modern. This tutorial also gives you an insight into the pros and cons and the practical applications of the SDLC models discussed.

Waterfall and V model are traditional SDLC models and are of sequential type. Sequential means that the next phase can start only after the completion of first phase. Such models are suitable for projects with very clear product requirements and where the requirements will not change dynamically during the course of project completion.

Iterative and Spiral models are more accommodative in terms of change and are suitable for projects where the requirements are not so well defined, or the market requirements change quite frequently. Big Bang model is a random approach to Software development and is suitable for small or academic projects.

Agile is the most popular model used in the industry. Agile introduces the concept of fast delivery to customers using prototype approach. Agile divides the project into small iterations with specific deliverable features. Customer interaction is the backbone of Agile methodology, and open communication with minimum documentation are the typical features of Agile development environment.

RAD (Rapid Application Development) and Software Prototype are modern techniques to understand the requirements in a better way early in the project cycle. These techniques work on the concept of providing a working model to the customer and stockholders to give the look and feel and collect the feedback. This feedback is used in an organized manner to improve the product.

The 'Resources' section below lists some suggested books and online resources to gain further understanding of the SDLC concepts.

Please send us your feedback at **webmaster@tutorialspoint.com**

Keep visiting to us, Happy Learning!

References

Books:

1) Lean Software Development: An Agile Toolkit for Software Development Managers - by Mary Poppendieck, Tom Poppendieck, Ken Schwaber

2) The Art Of Software Testing - By Glenford J Mayers

3) Agile Project Management with Scrum - by Ken Schwaber

4) Extreme Programming Explained - Book by Kent Beck

Websites:

http://en.wikipedia.org/wiki/Systems_Development_Life_Cycle

http://en.wikipedia.org/wiki/Agile_software_development

http://agilemanifesto.org/

http://www.agilealliance.org/