**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**What is SDLC?**

The **software development life cycle (SDLC)** is the entire process of formal, logical steps taken to develop a software product.

A Software Development Life Cycle Model is a set of activities together with an ordering relationship between activities which if performed in a manner that satisfies the ordering relationship that will produce desired product. Software Development Life Cycle Model is an abstract representation of a development process.

In a software development effort the goal is to produce high quality software. The development process is, therefore, the sequence of activities that will produce such software. A software development life cycle model is broken down into distinct activities. A software development life cycle model specifies how these activities are organized in the entire software development effort. We discuss each software development life cycle model in detail.

The phases of SDLC can vary somewhat but generally include the following:

* conceptualization;
* requirements and cost/benefits analysis;
* detailed specification of the software requirements;
* software design;
* programming;
* testing;
* user and technical training; and
* maintenance.

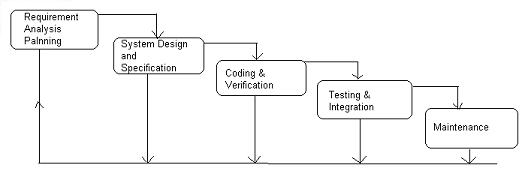
# **What are the different types of Software Development Life Cycle Models (SDLC)?**

1. Waterfall Software Development Life Cycle Model
2. Prototyping Software Development Life Cycle Model
3. Iterative Enhacement Model/Incremental model
4. The Spiral Model
5. Object Oriented Methodology
6. Dynamic System Development Method

**Explain waterfall model.**

The simplest software development life cycle model is the waterfall model, which states that the phases are organized in a linear order. A project begins with feasibility analysis. On the successful demonstration of the feasibility analysis, the requirements analysis and project planning begins.

The design starts after the requirements analysis is done. And coding begins after the design is done. Once the programming is completed, the code is integrated and testing is done. On succeeful completion of testing, the system is installed. After this the regular operation and maintenance of the system takes place. The following figure demonstrates the steps involved in waterfall life cycle model.

  
*The Waterfall Software Life Cycle Model*

With the waterfall model, the activities performed in a software development project are requirements analysis, project planning, system design, detailed design, coding and unit testing, system integration and testing. Linear ordering of activities has some important consequences. First, to clearly identify the end of a phase and beginning of the others. Some certification mechanism has to be employed at the end of each phase. This is usually done by some verification and validation. Validation means confirming the output of a phase is consistent with its input (which is the output of the previous phase) and that the output of the phase is consistent with overall requirements of the system.

The consequences of the need of certification is that each phase must have some defined output that can be evaluated and certified. Therefore, when the activities of a phase are completed, there should be an output product of that phase and the goal of a phase is to produce this product. The outputs of the earlier phases are often called intermediate products or design document. For the coding phase, the output is the code. From this point of view, the output of a software project is to justify the final program along with the use of documentation with the requirements document, design document, project plan, test plan and test results.

Another implication of the linear ordering of phases is that after each phase is completed and its outputs are certified, these outputs become the inputs to the next phase and should not be changed or modified. However, changing requirements cannot be avoided and must be faced. Since changes performed in the output of one phase affect the later phases, that might have been performed. These changes have to made in a controlled manner after evaluating the effect of each change on the project.This brings us to the need for configuration control or configuration management.

The certified output of a phase that is released for the best phase is called baseline. The configuration management ensures that any changes to a baseline are made after careful review, keeping in mind the interests of all parties that are affected by it. There are two basic assumptions for justifying the linear ordering of phase in the manner proposed by the waterfall model.

For a successful project resulting in a successful product, all phases listed in the waterfall model must be performed anyway.

Any different ordering of the phases will result in a less successful software product.

## Project Output in a Waterfall Model

As we have seen, the output of a project employing the waterfall model is not just the final program along with documentation to use it. There are a number of intermediate outputs, which must be produced in order to produce a successful product.

The set of documents that forms the minimum that should be produced in each project are:

* Requirement document
* Project plan
* System design document
* Detailed design document
* Test plan and test report
* Final code
* Software manuals (user manual, installation manual etc.)
* Review reports

Except for the last one, these are all the outputs of the phases. In order to certify an output product of a phase before the next phase begins, reviews are often held. Reviews are necessary especially for the requirements and design phases, since other certification means are frequently not available. Reviews are formal meeting to uncover deficiencies in a product. The review reports are the outcome of these reviews.

## Advantages of Waterfall Life Cycle Models

1. Easy to explain to the user
2. Stages and activities are well defined
3. Helps to plan and schedule the project
4. Verification at each stage ensures early detection of errors / misunderstanding

## Limitations of the Waterfall Life Cycle Model

The waterfall model assumes that the requirements of a system can be frozen (i.e. baselined) before the design begins. This is possible for systems designed to automate an existing manual system. But for absolutely new system, determining the requirements is difficult, as the user himself does not know the requirements. Therefore, having unchanging (or changing only a few) requirements is unrealistic for such project.

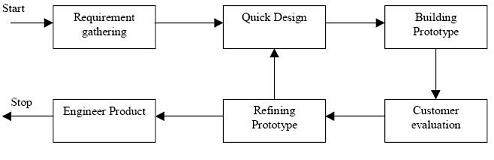
Freezing the requirements usually requires choosing the hardware (since it forms a part of the requirement specification). A large project might take a few years to complete. If the hardware is selected early, then due to the speed at which hardware technology is changing, it is quite likely that the final software will employ a hardware technology that is on the verge of becoming obsolete. This is clearly not desirable for such expensive software.

The waterfall model stipulates that the requirements should be completely specified before the rest of the development can proceed. In some situations it might be desirable to first develop a part of the system completely, an then later enhance the system in phase. This is often done for software products that are developed not necessarily for a client (where the client plays an important role in requirement specification), but for general marketing, in which the requirements are likely to be determined largely by developers.

**Explain prototyping Software Life Cycle Model.**

The goal of prototyping based development is to counter the first two limitations of the waterfall model discussed earlier. The basic idea here is that instead of freezing the requirements before a design or coding can proceed, a throwaway prototype is built to understand the requirements. This prototype is developed based on the currently known requirements. Development of the prototype obviously undergoes design, coding and testing. But each of these phases is not done very formally or thoroughly. By using this prototype, the client can get an "actual feel" of the system, since the interactions with prototype can enable the client to better understand the requirements of the desired system.

Prototyping is an attractive idea for complicated and large systems for which there is no manual process or existing system to help determining the requirements. In such situations letting the client "plan" with the prototype provides invaluable and intangible inputs which helps in determining the requirements for the system. It is also an effective method to demonstrate the feasibility of a certain approach. This might be needed for novel systems where it is not clear that constraints can be met or that algorithms can be developed to implement the requirements. The process model of the prototyping approach is shown in the figure below.

  
*Prototyping Model*

The basic reason for little common use of prototyping is the cost involved in this built-it-twice approach. However, some argue that prototyping need not be very costly and can actually reduce the overall development cost. The prototype are usually not complete systems and many of the details are not built in the prototype. The goal is to provide a system with overall functionality. In addition, the cost of testing and writing detailed documents are reduced. These factors helps to reduce the cost of developing the prototype. On the other hand, the experience of developing the prototype will very useful for developers when developing the final system. This experience helps to reduce the cost of development of the final system and results in a more reliable and better designed system.

## Advantages of Prototyping

1. Users are actively involved in the development
2. It provides a better system to users, as users have natural tendency to change their mind in specifying requirements and this method of developing systems supports this user tendency.
3. Since in this methodology a working model of the system is provided, the users get a better understanding of the system being developed.
4. Errors can be detected much earlier as the system is mode side by side.
5. Quicker user feedback is available leading to better solutions.

## Disadvantages

1. Leads to implementing and then repairing way of building systems.
2. Practically, this methodology may increase the complexity of the system as scope of the system may expand beyond original plans.

**What is a prototype? When should one use prototype model of SDLC?**

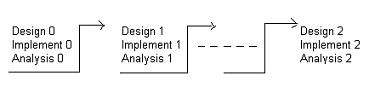
A prototype is a working model that is functionally equivalent to a component of the product.  
  
In many instances the client only has a general view of what is expected from the software product. In such a scenario where there is an absence of detailed information regarding the input to the system, the processing needs and the output requirements, the prototyping model may be employed.

**Explain iterative model.**

The iterative enhancement life cycle model counters the third limitation of the [waterfall model](http://www.freetutes.com/systemanalysis/sa2-waterfall-software-life-cycle.html) and tries to combine the benefits of both [prototyping](http://www.freetutes.com/systemanalysis/sa2-prototyping-model.html) and the [waterfall model](http://www.freetutes.com/systemanalysis/sa2-waterfall-software-life-cycle.html). The basic idea is that the software should be developed in increments, where each increment adds some functional capability to the system until the full system is implemented. At each step extensions and design modifications can be made. An advantage of this approach is that it can result in better testing, since testing each increment is likely to be easier than testing entire system like in the waterfall model. Furthermore, as in [prototyping](http://www.freetutes.com/systemanalysis/sa2-prototyping-model.html), the increments provides feedback to the client which is useful for determining the final requirements of the system.

In the first step of iterative enhancement model, a simple initial implementation is done for a subset of the overall problem. This subset is the one that contains some of the key aspects of the problem which are easy to understand and implement, and which forms a useful and usable system. A project control list is created which contains, in an order, all the tasks that must be performed to obtain the final implementation. This project control list gives an idea of how far the project is at any given step from the final system.

Each step consists of removing the next step from the list. Designing the implementation for the selected task, coding and testing the implementation, and performing an analysis of the partial system obtained after this step and updating the list as a result of the analysis. These three phases are called the design phase, implementation phase and analysis phase. The process is iterated until the project control list is empty, at the time the final implementation of the system will be available. The process involved in iterative enhancement model is shown in the figure below.

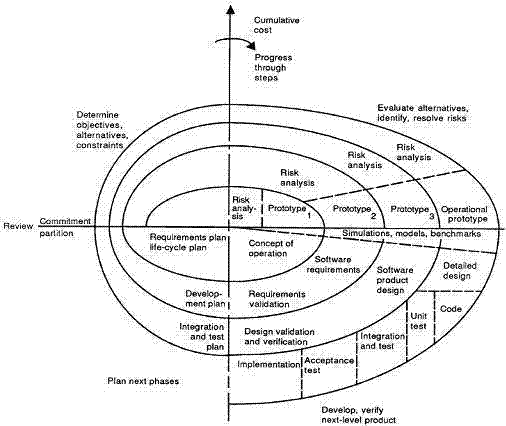
  
*The Iterative Enhancement Model*

The project control list guides the iteration steps and keeps track of all tasks that must be done. The tasks in the list can be include redesign of defective components found during analysis. Each entry in that list is a task that should be performed in one step of the iterative enhancement process, and should be simple enough to be completely understood. Selecting tasks in this manner will minimize the chances of errors and reduce the redesign work.

**Explain spiral model.**

This is a recent model that has been proposed by Boehm. As the name suggests, the activities in this model can be organized like a spiral. The spiral has many cycles. The radial dimension represents the cumulative cost incurred in accomplishing the steps dome so far and the angular dimension represents the progress made in completing each cycle of the spiral. The structure of the spiral model is shown in the figure given below. Each cycle in the spiral begins with the identification of objectives for that cycle and the different alternatives are possible for achieving the objectives and the imposed constraints.

The next step in the spiral life cycle model is to evaluate these different alternatives based on the objectives and constraints. This will also involve identifying uncertainties and risks involved. The next step is to develop strategies that resolve the uncertainties and risks. This step may involve activities such as benchmarking, simulation and prototyping. Next, the software is developed by keeping in mind the risks. Finally the next stage is planned.



The next step is determined by remaining risks. For example, its performance or user-interface risks are considered more important than the program development risks. The next step may be evolutionary development that involves developing a more detailed prototype for resolving the risks. On the other hand, if the program development risks dominate and previous prototypes have resolved all the user-interface and performance risks; the next step will follow the basic waterfall approach.

The risk driven nature of the spiral model allows it to accommodate any mixture of specification-oriented, prototype-oriented, simulation-oriented or some other approach. An important feature of the model is that each cycle of the spiral is completed by a review, which covers all the products developed during that cycle, including plans for the next cycle. The spiral model works for developed as well as enhancement projects.

## Spiral Model Description

The development spiral consists of four quadrants as shown in the figure above

Quadrant 1: Determine objectives, alternatives, and constraints.

Quadrant 2: Evaluate alternatives, identify, resolve risks.

Quadrant 3: Develop, verify, next-level product.

Quadrant 4: Plan next phases.

Although the spiral, as depicted, is oriented toward software development, the concept is equally applicable to systems, hardware, and training, for example. To better understand the scope of each spiral development quadrant, let’s briefly address each one.

### Quadrant 1: Determine Objectives, Alternatives, and Constraints

Activities performed in this quadrant include:

1. Establish an understanding of the system or product objectives—namely performance, functionality, and ability to accommodate change.
2. Investigate implementation alternatives—namely design, reuse, procure, and procure/ modify
3. Investigate constraints imposed on the alternatives—namely technology, cost, schedule, support, and risk. Once the system or product’s objectives, alternatives, and constraints are understood, Quadrant 2 (Evaluate alternatives, identify, and resolve risks) is performed.

### Quadrant 2: Evaluate Alternatives, Identify, Resolve Risks

Engineering activities performed in this quadrant select an alternative approach that best satisfies technical, technology, cost, schedule, support, and risk constraints. The focus here is on risk mitigation. Each alternative is investigated and prototyped to reduce the risk associated with the development decisions. Boehm describes these activities as follows:

. . . This may involve prototyping, simulation, benchmarking, reference checking, administering user  
questionnaires, analytic modeling, or combinations of these and other risk resolution techniques.

The outcome of the evaluation determines the next course of action. If critical operational and/or technical issues (COIs/CTIs) such as performance and interoperability (i.e., external and internal) risks remain, more detailed prototyping may need to be added before progressing to the next quadrant. Dr. Boehm notes that if the alternative chosen is “operationally useful and robust enough to serve as a low-risk base for future product evolution, the subsequent risk-driven steps would be the evolving series of evolutionary prototypes going toward the right (hand side of the graphic) . . . the option of writing specifications would be addressed but not exercised.” This brings us to Quadrant 3.

### Quadrant 3: Develop, Verify, Next-Level Product

If a determination is made that the previous prototyping efforts have resolved the COIs/CTIs, activities to develop, verify, next-level product are performed. As a result, the basic “waterfall” approach may be employed—meaning concept of operations, design, development, integration, and test of the next system or product iteration. If appropriate, incremental development approaches may also be applicable.

### Quadrant 4: Plan Next Phases

The spiral development model has one characteristic that is common to all models—the need for advanced technical planning and multidisciplinary reviews at critical staging or control points. Each cycle of the model culminates with a technical review that assesses the status, progress, maturity, merits, risk, of development efforts to date; resolves critical operational and/or technical issues (COIs/CTIs); and reviews plans and identifies COIs/CTIs to be resolved for the next iteration of the spiral.

Subsequent implementations of the spiral may involve lower level spirals that follow the same quadrant paths and decision considerations.

**What is difference between iterative model and waterfall model? Explain what is the difference between iterative model and prototype model? Explain.**

WaterFall Model: This is a flow based model, in which u

pass every phase once, and can not go back to that phase

again. That is why, it is used rarely now a days.

Drawback: If there is nay change in requirements, then u

can not make any chnages in requirements.

Prototype Model: Here, we recieved Prototypes of the

product, before the final release. We relaese 4-5 Protypes

with some diffrences b/w them, and take client opinion, and

modifies the final Product, as per client suggestions.

Iterative Model: In this u can come back to previous

phases, and make the changes accordingly. In this we

revieved a final output product at the end of the SDLC.

Agile software development

**What is agile software development?**

Agile software development refers to a group of software development methodologies based on iterative development, where requirements and solutions evolve through collaboration between self-organizing [cross-functional teams](http://team). The term was coined in the year 2001 when the [Agile Manifesto](http://manifesto) was formulated.

Agile methods generally promote a disciplined project management process that encourages frequent inspection and adaptation, a leadership philosophy that encourages teamwork, self-organization and accountability, a set of engineering best practices intended to allow for rapid delivery of high-quality software, and a business approach that aligns development with customer needs and company goals.

**Compare agile methodology with other software development process.**

Agile methods are sometimes characterized as being at the opposite end of the spectrum from "plan-driven" or "disciplined" methods. This distinction is misleading, as it implies that agile methods are "unplanned" or "undisciplined". A more accurate distinction is that methods exist on a continuum from "adaptive" to "predictive".[[9]](http://en.wikipedia.org/wiki/Agile_software_development#cite_note-boehm2004App-8) Agile methods lie on the "adaptive" side of this continuum.

*Adaptive methods focus on adapting quickly to changing realities.* *When the needs of a project change, an adaptive team changes as well.* An adaptive team will have difficulty describing exactly what will happen in the future. The further away a date is, the more vague an adaptive method will be about what will happen on that date. An adaptive team can report exactly what tasks are being done next week, but only which features are planned for next month. When asked about a release six months from now, an adaptive team may only be able to report the mission statement for the release, or a statement of expected value vs. cost.

Predictive methods, in contrast, focus on planning the future in detail. A predictive team can report exactly what features and tasks are planned for the entire length of the development process. Predictive teams have difficulty changing direction. The plan is typically optimized for the original destination and changing direction can cause completed work to be thrown away and done over differently. Predictive teams will often institute a [change control board](http://board) to ensure that only the most valuable changes are considered.

Agile methods have much in common with the "[Rapid Application Development](http://development)" techniques from the 1980/90s as espoused by James Martin and others.

### *Contrasted with other iterative development methods*

Most agile methods share other [iterative and incremental development](http://development) methods' emphasis on building releasable software in short time periods. Agile development differs from other development models: in this model, time periods are measured in weeks rather than months and work is performed in a highly collaborative manner. Most agile methods also differ by treating their time period as a [timebox](http://timeboxing).

### *Contrasted with the Waterfall Methodology*

Agile development has little in common with the [waterfall model](http://model). The Waterfall methodology is the most structured of the methods, stepping through requirements, analysis, design, coding, and testing in a strict, pre-planned, "all at once" sequence. Progress is often measured in terms of deliverable artifacts: requirement specifications, design documents, test plans, code reviews and the like.

A common criticism of the waterfall model is its inflexible division of a project into separate stages, where commitments are made early on, making it difficult to react to changes in requirements as the project executes. This means that the waterfall model is likely to be unsuitable if requirements are not well understood/defined or change in the course of the project.[[10]](http://en.wikipedia.org/wiki/Agile_software_development#cite_note-9)

Agile methods, in contrast, produce completely developed and tested features (but a very small subset of the whole) every few weeks. The emphasis is on obtaining the smallest workable piece of functionality to deliver business value early and continually improving it and/or adding further functionality throughout the life of the project. If a project being delivered under Waterfall is cancelled at any point up to the end, there is often nothing to show for it beyond a huge resources bill. With Agile, being cancelled at any point will still leave the customer with some worthwhile code that has likely already been put into live operation.

Adaptations of Scrum show how agile methods are augmented to produce and continuously improve a strategic plan.

Some agile teams use the waterfall model on a small scale, repeating the entire waterfall cycle in every iteration. Other teams, most notably [Extreme Programming](http://programming) teams, work on activities simultaneously.

**How do you choose between choose between adaptive ("agile") and predictive ("plan-driven") methods?**

Agile home ground:

* Low criticality
* Senior developers
* Requirements change often
* Small number of developers
* Culture that thrives on chaos

Plan-driven home ground:

* High criticality
* Junior developers
* Requirements do not change often
* Large number of developers
* Culture that demands order

**Name the various agile software development methods.**

Some of the well-known agile software development methods:

* [Agile Modeling](http://modeling)
* [Agile Unified Process](http://process) (AUP)
* [DSDM](http://dsdm)
* [Essential Unified Process](http://process) (EssUP)
* [Extreme Programming](http://programming) (XP)
* [Feature Driven Development](http://development) (FDD)
* [Open Unified Process](http://process) (OpenUP)
* Scrum

**Name the various agile practices.**

* Test Driven Development (TDD)
* Behavior Driven Development (BDD)
* Code refactoring
* [Continuous Integration](http://integration)
* [Pair Programming](http://programming)
* [Planning poker](http://poker)
* [RITE Method](http://method)

Note: Although these are often considered methodologies in and of themselves, they are simply practices used in different methodologies.