**What is SDLC?**

 What is the SDLC?  
Software is a complex product that is developed and delivered through a series of steps. That is the one thing all the various methods have in common: one way or another, software, like all products, starts as an idea. The idea then becomes a document, or perhaps a prototype, depending on the method in use. Whether a document, diagram, or working software, the artifact created in one step becomes the input to the next step. Eventually, the software is delivered to the customer. The sequence of steps used by these methods is commonly referred to as the Software Development Lifecycle (SDLC.)  
  
How does the SDLC work?  
The process of software development is a never-ending cycle. The first release of a software application is rarely “finished.” There are almost always additional features and bug fixes waiting to be designed, developed, and deployed.  
  
Reports from error monitoring software about usability and bugs feed back into the process of software development, and become new feature requests and improvements to existing features. This is why the Software Development Life Cycle is the most general term for software development methods. The steps of the process and their order vary by method. Regardless of method, they typically run in cycles, starting over with each iteration.  
  
It’s very difficult to carry out a complex, team effort such as software development without some kind of plan. Each software development methodology (several will be detailed below) is a plan framework for how to develop software. There is much debate about which method is best overall, which is best suited to a particular type of software, and how to measure success in software development. One thing, however, is certain: any plan is better than no plan.  
  
Without some kind of structured plan, software development teams tend to devolve into a “herd of cats.” Developers don’t know what they’re supposed to create. Project managers have no idea how much progress is made towards completion of a project. Without a plan, the business doesn’t even have a way to decide whether the final product meets their requirements.

**What are the benefits of using SDLC?**

 The Software Development Life Cycle (SDLC) is a systematic process for planning, creating, testing, deploying, and maintaining software. Adopting SDLC methodologies offers several benefits, including:  
  
1. \*\*Structured Approach:\*\* SDLC provides a structured and organized framework for developing software. It helps in breaking down the entire process into manageable phases, making it easier to plan, track, and control.  
  
2. \*\*Improved Project Management:\*\* SDLC facilitates better project management by defining roles and responsibilities, setting realistic expectations, and establishing clear communication channels. This leads to more effective coordination among team members.  
  
3. \*\*Risk Management:\*\* SDLC helps in identifying and addressing potential risks early in the development process. By conducting thorough risk assessments during each phase, development teams can proactively manage and mitigate risks, reducing the likelihood of project failure.  
  
4. \*\*Quality Assurance:\*\* SDLC emphasizes testing and validation at every stage of development. This focus on quality assurance ensures that defects and issues are identified and resolved early in the process, reducing the likelihood of bugs in the final product.  
  
5. \*\*Cost and Time Efficiency:\*\* SDLC promotes better resource management, leading to cost and time savings. By following a systematic approach, development teams can optimize their workflows, streamline processes, and avoid costly rework.  
  
6. \*\*Customer Satisfaction:\*\* SDLC encourages customer involvement throughout the development process. Continuous feedback from stakeholders ensures that the final product aligns with user expectations and requirements, leading to higher customer satisfaction.  
  
7. \*\*Enhanced Communication:\*\* SDLC fosters clear communication among team members, stakeholders, and customers. By establishing a common understanding of project goals, requirements, and timelines, teams can work together more effectively.  
  
8. \*\*Easier Maintenance and Updates:\*\* The systematic and organized nature of SDLC makes it easier to maintain and update software. Changes can be implemented more smoothly, and new features can be integrated without disrupting the overall stability of the system.  
  
9. \*\*Documentation:\*\* SDLC promotes thorough documentation of all phases of development. This documentation serves as a valuable resource for future maintenance, troubleshooting, and knowledge transfer within the team.  
  
10. \*\*Scalability:\*\* SDLC methodologies are scalable and adaptable to different project sizes and types. Whether it's a small-scale project or a large enterprise-level application, SDLC can be tailored to meet the specific needs and complexities of the development effort.  
  
By incorporating these benefits, SDLC contributes to the overall success of software development projects, ensuring they are completed on time, within budget, and with high-quality outcomes.

**Benefits of using SDLC**

 The Software Development Life Cycle (SDLC) is a systematic process for planning, creating, testing, deploying, and maintaining software. Adopting SDLC methodologies offers several benefits, including:  
  
1. \*\*Structured Approach:\*\* SDLC provides a structured and organized framework for developing software. It helps in breaking down the entire process into manageable phases, making it easier to plan, track, and control.  
  
2. \*\*Improved Project Management:\*\* SDLC facilitates better project management by defining roles and responsibilities, setting realistic expectations, and establishing clear communication channels. This leads to more effective coordination among team members.  
  
3. \*\*Risk Management:\*\* SDLC helps in identifying and addressing potential risks early in the development process. By conducting thorough risk assessments during each phase, development teams can proactively manage and mitigate risks, reducing the likelihood of project failure.  
  
4. \*\*Quality Assurance:\*\* SDLC emphasizes testing and validation at every stage of development. This focus on quality assurance ensures that defects and issues are identified and resolved early in the process, reducing the likelihood of bugs in the final product.  
  
5. \*\*Cost and Time Efficiency:\*\* SDLC promotes better resource management, leading to cost and time savings. By following a systematic approach, development teams can optimize their workflows, streamline processes, and avoid costly rework.  
  
6. \*\*Customer Satisfaction:\*\* SDLC encourages customer involvement throughout the development process. Continuous feedback from stakeholders ensures that the final product aligns with user expectations and requirements, leading to higher customer satisfaction.  
  
7. \*\*Enhanced Communication:\*\* SDLC fosters clear communication among team members, stakeholders, and customers. By establishing a common understanding of project goals, requirements, and timelines, teams can work together more effectively.  
  
8. \*\*Easier Maintenance and Updates:\*\* The systematic and organized nature of SDLC makes it easier to maintain and update software. Changes can be implemented more smoothly, and new features can be integrated without disrupting the overall stability of the system.  
  
9. \*\*Documentation:\*\* SDLC promotes thorough documentation of all phases of development. This documentation serves as a valuable resource for future maintenance, troubleshooting, and knowledge transfer within the team.  
  
10. \*\*Scalability:\*\* SDLC methodologies are scalable and adaptable to different project sizes and types. Whether it's a small-scale project or a large enterprise-level application, SDLC can be tailored to meet the specific needs and complexities of the development effort.  
  
By incorporating these benefits, SDLC contributes to the overall success of software development projects, ensuring they are completed on time, within budget, and with high-quality outcomes.

**The seven important phases of SDLC**

 The Seven Phases of the SDLC  
These steps are (very) roughly the same from one methodology to another. They tend to occur in this order, though they can also be mixed together, such that several steps occur in parallel.  
  
As we’ll discuss later, Agile methods tend to “wind together” all of these steps into a tight, rapidly-repeating cycle. Waterfall methods tend to take each of these steps in turn. Outputs from one become inputs to the following step.  
  
The following figure is a graphical representation of the various stages of a typical SDLC.  
  
SDLC Image  
  
1. Requirements Analysis/Planning  
  
The planning phase involves aspects of project and product management. This may include:  
  
Resource allocation (both human and materials)  
  
Capacity planning  
Project scheduling  
Cost estimation  
Provisioning  
The outputs of the planning phase include: project plans, schedules, cost estimations, and procurement requirements. Ideally, Project Managers and Development staff collaborate with Operations and Security teams to ensure all perspectives are represented.  
  
2. Defining/Feasibility  
  
The business must communicate with IT teams to convey their requirements for new development and enhancement. The requirements phase gathers these requirements from business stakeholders and Subject Matter Experts (SMEs.)  
  
There are mainly five types of feasibility checks:  
  
Economic: Can we complete the project within the budget or not?  
Legal: Can we handle this project as cyber law and other regulatory framework/compliance.  
Operation feasibility: Can we create operations which is expected by the client?  
Technical: Need to check whether the current computer system can support the software  
Schedule: Decide that the project can be completed within the given schedule or not.  
Architects, Development teams, and Product Managers work with the SMEs to document the business processes that need to be automated through software. The output of this phase in a Waterfall project is usually a document that lists these requirements. Agile methods, by contrast, may produce a backlog of tasks to be performed.  
  
3. Design and prototyping  
  
Once the requirements are understood, software architects and developers can begin to design the software. The design process uses established patterns for application architecture and software development. Architects may use an architecture framework such as TOGAF to compose an application from existing components, promoting reuse and standardization.  
  
Developers use proven Design Patterns to solve algorithmic problems in a consistent way. This phase may also include some rapid prototyping, also known as a spike, to compare solutions to find the best fit. The output of this phase includes:  
  
Design documents that list the patterns and components selected for the project  
Code produced by spikes, used as a starting point for development  
There are two kinds of design documents developed in this phase:  
  
High-Level Design (HLD)  
Brief description and name of each module  
An outline about the functionality of every module  
Interface relationship and dependencies between modules  
Database tables identified along with their key elements  
Complete architecture diagrams along with technology details  
Low-Level Design (LLD)  
Functional logic of the modules  
Database tables, which include type and size  
Complete detail of the interface  
Addresses all types of dependency issues  
Listing of error messages  
Complete input and outputs for every module  
4. Coding/Software development  
  
This phase produces the software under development. Depending on the methodology, this phase may be conducted in time-boxed “sprints,” (Agile) or may proceed as a single block of effort (Waterfall.) Regardless of methodology, development teams should produce working software as quickly as possible. Business stakeholders should be engaged regularly, to ensure that their expectations are being met. The output of this phase is testable, functional software.  
  
5. Testing  
  
The testing phase of the SDLC is arguably one of the most important. It is impossible to deliver quality software without testing. There is a wide variety of testing necessary to measure quality:  
  
Code quality  
Unit testing (functional tests)  
Integration testing  
Performance testing  
Security testing  
The best way to ensure that tests are run regularly, and never skipped for expediency, is to automate them. Tests can be automated using Continuous Integration tools, like Codeship, for example. The output of the testing phase is functional software, ready for deployment to a production environment.  
  
6. Deployment  
  
The deployment phase is, ideally, a highly automated phase. In high-maturity enterprises, this phase is almost invisible; software is deployed the instant it is ready. Enterprises with lower maturity, or in some highly regulated industries, the process involves some manual approvals. However, even in those cases it is best for the deployment itself to be fully automated in a continuous deployment model. Application Release Automation (ARA) tools are used in medium and large-size enterprises to automate the deployment of applications to Production environments. ARA systems are usually integrated with Continuous Integration tools. The output of this phase is the release to Production of working software.  
  
7. Operations and maintenance  
  
The operations and maintenance phase are the “end of the beginning,” so to speak. The Software Development Life Cycle doesn’t end here. Software must be monitored constantly to ensure proper operation. Bugs and defects discovered in Production must be reported and responded to, which often feeds work back into the process. Bug fixes may not flow through the entire cycle, however, at least an abbreviated process is necessary to ensure that the fix does not introduce other problems (known as a regression.)

**SDLC Models**

 Popular SDLC Models  
This SDLC model is documentation-intensive, with earlier phases documenting what need be performed in the subsequent phases. Here, are some most important phases of SDLC life cycle:  
  
1. V-Model  
  
The V-model is an SDLC model where execution of processes happens in a sequential manner in a V-shape. It is also known as Verification and Validation model. The V-Model is an extension of the waterfall model and is based on the 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 the next phase starts only after completion of the previous phase.  
  
V-Model - Design  
Under the 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. The Coding Phase joins the two sides of the V-Model.  
  
The following illustration depicts the different phases in a V-Model of the SDLC.  
  
V Model Image  
  
There are several Verification phases in the V-Model, each of these are explained in detail below.  
  
Business Requirement Analysis  
This is the first phase in the development cycle where the product requirements are understood from the customer’s perspective. This phase involves detailed communication with the customer to understand his expectations and exact requirement. This is a very important activity and needs 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 is time to design the complete system. The system design will have the understanding and detailing the complete hardware and communication setup for the product under development. The system test plan is developed based on the system design. Doing this at an earlier stage leaves more time for the 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. The 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. The unit tests are an essential part of any development process and helps eliminate the maximum faults and errors at a very early stage. These unit tests can be designed at this stage based on the internal module designs.

**What are the pro and cons of using V-Model?**

 The V-Model, also known as the Verification and Validation Model, is a software development and testing process that emphasizes a systematic and parallel progression of development and testing activities. Here are the pros and cons of using the V-Model:  
  
### Pros:  
  
1. \*\*Clarity and Simplicity:\*\*  
- The V-Model is straightforward and easy to understand. It provides a clear and simple path from requirements to testing.  
  
2. \*\*Early Test Planning:\*\*  
- Testing activities are planned early in the development process, allowing for early detection and correction of defects.  
  
3. \*\*Parallel Development and Testing:\*\*  
- Development and testing activities are carried out simultaneously in a parallel fashion, which can accelerate the development life cycle.  
  
4. \*\*Traceability:\*\*  
- There is a direct relationship between the stages of development and their corresponding testing phases, providing traceability between requirements, design, and testing.  
  
5. \*\*Defects Identification:\*\*  
- Defects can be identified early in the life cycle, making it easier and less costly to fix issues.  
  
6. \*\*Comprehensive Documentation:\*\*  
- The model encourages the creation of comprehensive documentation at each stage, which can be beneficial for future reference and maintenance.  
  
### Cons:  
  
1. \*\*Rigidity:\*\*  
- The V-Model can be perceived as rigid, and any changes in the requirements can be challenging to accommodate after the project has started.  
  
2. \*\*Late Feedback:\*\*  
- Testing activities are typically initiated after the completion of the development phase, which may lead to late detection and correction of defects.  
  
3. \*\*Limited Flexibility:\*\*  
- The model may not be well-suited for projects with evolving or unclear requirements, as it lacks the flexibility to adapt to changes easily.  
  
4. \*\*Sequential Nature:\*\*  
- The model follows a sequential path, and each phase depends on the completion of the previous one. This can slow down the overall development process.  
  
5. \*\*Client Involvement:\*\*  
- Client involvement primarily occurs during the requirements phase, and there might be limited opportunities for feedback during the development and testing stages.  
  
6. \*\*Not Ideal for Complex Projects:\*\*  
- The V-Model may not be the most suitable choice for large and complex projects where a more iterative and adaptive approach, like Agile, might be more effective.  
  
In summary, the V-Model can be a good fit for projects with well-defined and stable requirements, where a sequential and systematic approach to development and testing is appropriate. However, its rigidity and lack of flexibility can be limitations in dynamic and rapidly changing project environments.

**Activity 1**

 The V-Model, also known as the Verification and Validation Model, is a software development and testing process that emphasizes a systematic and parallel progression of development and testing activities. Here are the pros and cons of using the V-Model:  
  
### Pros:  
  
1. \*\*Clarity and Simplicity:\*\*  
- The V-Model is straightforward and easy to understand. It provides a clear and simple path from requirements to testing.  
  
2. \*\*Early Test Planning:\*\*  
- Testing activities are planned early in the development process, allowing for early detection and correction of defects.  
  
3. \*\*Parallel Development and Testing:\*\*  
- Development and testing activities are carried out simultaneously in a parallel fashion, which can accelerate the development life cycle.  
  
4. \*\*Traceability:\*\*  
- There is a direct relationship between the stages of development and their corresponding testing phases, providing traceability between requirements, design, and testing.  
  
5. \*\*Defects Identification:\*\*  
- Defects can be identified early in the life cycle, making it easier and less costly to fix issues.  
  
6. \*\*Comprehensive Documentation:\*\*  
- The model encourages the creation of comprehensive documentation at each stage, which can be beneficial for future reference and maintenance.  
  
### Cons:  
  
1. \*\*Rigidity:\*\*  
- The V-Model can be perceived as rigid, and any changes in the requirements can be challenging to accommodate after the project has started.  
  
2. \*\*Late Feedback:\*\*  
- Testing activities are typically initiated after the completion of the development phase, which may lead to late detection and correction of defects.  
  
3. \*\*Limited Flexibility:\*\*  
- The model may not be well-suited for projects with evolving or unclear requirements, as it lacks the flexibility to adapt to changes easily.  
  
4. \*\*Sequential Nature:\*\*  
- The model follows a sequential path, and each phase depends on the completion of the previous one. This can slow down the overall development process.  
  
5. \*\*Client Involvement:\*\*  
- Client involvement primarily occurs during the requirements phase, and there might be limited opportunities for feedback during the development and testing stages.  
  
6. \*\*Not Ideal for Complex Projects:\*\*  
- The V-Model may not be the most suitable choice for large and complex projects where a more iterative and adaptive approach, like Agile, might be more effective.  
  
In summary, the V-Model can be a good fit for projects with well-defined and stable requirements, where a sequential and systematic approach to development and testing is appropriate. However, its rigidity and lack of flexibility can be limitations in dynamic and rapidly changing project environments.

**More about SDLC Models**

 2. Spiral Model  
  
The spiral model is a risk-driven process model. This SDLC model helps the team to adopt elements of one or more process models like a waterfall, incremental, waterfall, etc. This model adopts the best features of the prototyping model and the waterfall model. The spiral methodology is a combination of rapid prototyping and concurrency in design and development activities.  
  
The following illustration is a representation of the Spiral Model, listing the activities in each phase.  
  
Spiral Model Image  
  
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 phase 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.  
  
Design  
The Design phase starts with the conceptual design in the baseline spiral and involves architectural design, logical design of modules, physical product design and the final design in the subsequent spirals.  
  
Construct or Build  
The 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 the customer for feedback.  
  
Evaluation and Risk Analysis  
Risk Analysis includes identifying, estimating and monitoring the 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.  
  
3. Big bang model  
  
Big bang model is focusing on all types of resources in software development and coding, with no or very little planning. The requirements are understood and implemented when they come. This model works best for small projects with smaller size development team which are working together. It is also useful for academic software development projects. It is an ideal model where requirements are either unknown or final release date is not given.  
  
Big Bang Model - Pros and Cons  
The advantage of this Big Bang Model is that it is very simple and requires very little or no planning. Easy to manage 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.  
  
The advantages of the Big Bang Model are as follows:  
  
This is a very simple model  
Little or no planning required  
Easy to manage  
Very few resources required  
Gives flexibility to developers  
It is a good learning aid for new comers or students.  
The disadvantages of the Big Bang Model are as follows:  
  
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.  
4. Waterfall model  
  
The waterfall is a widely accepted SDLC model. In this approach, the whole process of the software development is divided into various phases. In this SDLC model, the outcome of one phase acts as the input for the next phase.  
  
Waterfall methodology begins with long planning and design phases. Once developed, the software then goes through phases of testing, and is finally deployed for use. Waterfall is considered by many to be too rigid to adapt to changing requirements. It does not support feedback throughout the process, leading to the implementation of requirements that may have changed during the development effort. This weakness in Waterfall led to the development of more flexible methodologies, such as Agile.  
  
Requirement Doc Image  
  
The Waterfall method of software development follows a rigid, predetermined path through a set of phases. This method was adapted from traditional engineering. Ironically, the paper credited as the origin of the Waterfall method, describes it as being fundamentally flawed. The method that is known today as “Waterfall” was mistakenly derived from a misunderstanding of this original work. Despite that fact, Waterfall became a very common, even standard methodology for large projects around the world.  
  
Waterfall Model - Advantages  
The advantages of waterfall development are 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.  
Some of the major advantages of the Waterfall Model are as follows −  
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.

**5. Agile Model**

Agile methodology is a practice which promotes continue interaction of development and testing during the SDLC process of any project. In the Agile method, the entire project is divided into small incremental builds. All of these builds are provided in iterations, and each iteration lasts from one to three weeks. The Manifesto for Agile Software Development was drafted and signed by a group of software developers in 2001. Reading the manifesto, you can see clearly the contrast between Waterfall, then the de-facto standard for development methods, and Agile, the newer method.

The Manifesto addresses key problems with Waterfall that led to challenges in software delivery. Where Waterfall tends to be a “one-way road,” Agile is a more flexible framework that allows for uncertainty. Agile emphasizes teamwork, prototyping, and feedback loops that can change the direction of the development effort in response to changing requirements. Several variants of Agile have emerged since the signing of the Manifesto. Scrum defines specific roles and events, known as ceremonies, as part of its practice. Kanban is simpler, with fewer prescriptions and more flexibility. Agile teams often combine these together to adapt a bespoke process that fits them best.



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 customers to understand their requirements, 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 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.

#### ****The advantages of the Agile Model are as follows****

* 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.

#### ****The disadvantages of the Agile Model are as follows****

* 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 a 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.

## **Comparison of Various SDLC Models**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Properties of Model** | **Water-Fall Model** | **Incremental Model** | **Spiral Model** | **Rad Model** |
| Planning in early stage | Yes | Yes | Yes | No |
| Returning to an earlier phase | No | Yes | Yes | Yes |
| Handle Large-Project | Not Appropriate | Not Appropriate | Appropriate | Not Appropriate |
| Detailed Documentation | Necessary | Yes, but not much | Yes | Limited |
| Cost | Low | Low | Expensive | Low |
| Requirement Specifications | Beginning | Beginning | Beginning | Time boxed release |
| Flexibility to change | Difficult | Easy | Easy | Easy |
| User Involvement | Only at beginning | Intermediate | High | Only at the beginning |
| Maintenance | Least | Promotes Maintainability | Typical | Easily Maintained |
| Duration | Long | Very long | Long | Short |
| Risk Involvement | High | Low | Medium to high risk | Low |
| Framework Type | Linear | Linear + Iterative | Linear + Iterative | Linear |
| Testing | After completion of coding phase | After every iteration | At the end of the engineering phase | After completion of coding |
| Overlapping Phases | No | Yes (As parallel development is there) | No | Yes |
| Maintenance | Least Maintainable | Maintainable | Yes | Easily Maintainable |
| Re-usability | Least possible | To some extent | To some extent | Yes |
| Time-Frame | Very Long | Long | Long | Short |
| Working software availability | At the end of the life-cycle | At the end of every iteration | At the end of every iteration | At the end of the life cycle |
| Objective | High Assurance | Rapid Development | High Assurance | Rapid development |
| Team size | Large Team | Not Large Team | Large Team | Small Team |
| Customer control over administrator | Very Low | Yes | Yes | Yes |

**Software Prototyping**

 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 a stepwise approach explained 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. While, 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 the actual implementation. The changes accepted are again incorporated in the new Prototype developed and the cycle repeats until the customer expectations are met.  
  
Prototypes can have horizontal or vertical dimensions. A 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 load 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. By using evolutionary prototyping, the 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.  
  
Best practices of Prototyping  
  
Here, are a few things which you should watch for during the prototyping process:  
You should use Prototyping when the requirements are unclear  
It is important to perform planned and controlled Prototyping.  
Regular meetings are vital to keep the project on time and avoid costly delays.  
The users and the designers should be aware of the prototyping issues and pitfalls.  
At a very early stage, you need to approve a prototype and only then allow the team to move to the next step.  
In software prototyping method, you should never be afraid to change earlier decisions if new ideas need to be deployed.  
You should select the appropriate step size for each version.  
Implement important features early on so that if you run out of the time, you still have a worthwhile system  
Advantages of the Prototyping Model  
  
Here, are important pros/benefits of using Prototyping models:  
  
Users are actively involved in development. Therefore, errors can be detected in the initial stage of the software development process.  
Missing functionality can be identified, which helps to reduce the risk of failure as Prototyping is also considered as a risk reduction activity.  
Helps team member to communicate effectively  
Customer satisfaction exists because the customer can feel the product at a very early stage.  
There will be hardly any chance of software rejection.  
Quicker user feedback helps you to achieve better software development solutions.  
Allows the client to compare if the software code matches the software specification.  
It helps you to find out the missing functionality in the system.  
It also identifies the complex or difficult functions.  
Encourages innovation and flexible designing.  
It is a straightforward model, so it is easy to understand.  
No need for specialized experts to build the model  
The prototype serves as a basis for deriving a system specification.  
The prototype helps to gain a better understanding of the customer's needs.  
Prototypes can be changed and even discarded.  
A prototype also serves as the basis for operational specifications.  
Prototypes may offer early training for future users of the software system.  
Disadvantages of the Prototyping Model  
  
Here, are important cons/drawbacks of prototyping model:  
Prototyping is a slow and time taking process.  
The cost of developing a prototype is a total waste as the prototype is ultimately thrown away.  
Prototyping may encourage excessive change requests.  
Sometimes customers may not be willing to participate in the iteration cycle for the longer time duration.  
There may be far too many variations in software requirements when each time the prototype is evaluated by the customer.  
Poor documentation because the requirements of the customers are changing.  
It is very difficult for software developers to accommodate all the changes demanded by the clients.  
After seeing an early prototype model, the customers may think that the actual product will be delivered to him soon.  
The client may lose interest in the final product when he or she is not happy with the initial prototype.  
Developers who want to build prototypes quickly may end up building sub-standard development solutions.

**A quick summary on Software Prototyping**

 A quick Summary on Software Prototyping  
In Software Engineering, Prototype methodology is a software development model in which a prototype is built, test and then reworked when needed until an acceptable prototype is achieved.  
Requirements gathering and analysis, 2) Quick design, 3) Build a Prototype, 4) Initial user evaluation, 5) Refining prototype, 6)Implement Product and Maintain; are 6 steps of the prototyping process  
Type of prototyping models are 1) Rapid Throwaway prototypes 2) Evolutionary prototype 3) Incremental prototype 4) Extreme prototype  
Regular meetings are essential to keep the project on time and avoid costly delays in prototyping approach.  
Missing functionality can be identified, which helps to reduce the risk of failure as Prototyping is also considered as a risk reduction activity in SDLC.  
Prototyping may encourage excessive change requests.