**Software Development Lifecycle (SDLC)**

The software development lifecycle (SDLC) is a conceptual model used in project management that describes the stages involved in a software development project. SDLC consists of a detailed plan describing how to develop, maintain, replace and alter or enhance software. A SDLC consist of process model which is used to indicate the steps to be taken to ensure the success in software development.

**SDLC Spiral Model**

The spiral model has four phases.

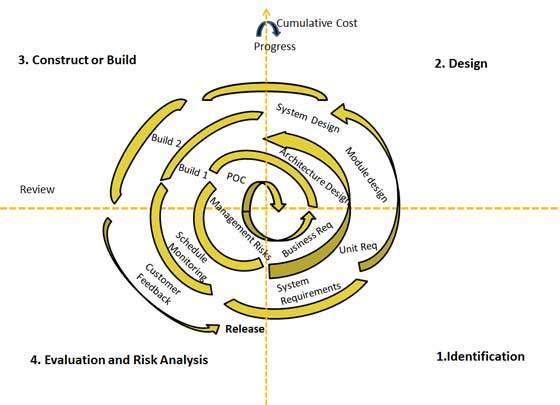
**Identification:** This phase start by gathering the business requirements in the baseline spiral. In the subsequent spirals, the product matures, identification of the system and subsystem requirements and unit requirement are all done at this phase.

**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 of every spiral when the product is just thought of and the design is being developed a POC (Proof of Concept) is developed in this phase. In the subsequent spirals which higher charity.

**Evaluation and Risk Analysis:** Risk Analysis includes identifying, estimating, and monitoring technically feasibility and management risks, such as schedule slippage and cost overrun.

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



**Spiral Model** **Fig 1.1**

**Waterfall Model**

The waterfall model has 6 phases.

**Requirement gathering and analysis:** All possible requirements of the system to be developed are captured in this phase and documented in a requirement specific document.

**System design:** The requirement specifications from the first phase are studied in this phase and system design is prepared. System Design helps in specifying hardware and system requirements and 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.

**Development 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:** To fix issues with patches and also to enhance the product functionality with newer version.

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



**Waterfall Model Fig 1.2**

**Agile Model**

The Agile Manifesto has 4 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.

|  |  |
| --- | --- |
| **Pros** | **Cons** |
| Is a very realistic approach to software development | Not suitable for handling complex dependencies. |
| Promotes teamwork and cross training | More risk of sustainability, maintainability and extensibility |
| Functionality can be developed rapidly and demonstrated. | An overall plan, an agile leader and agile PM practice is a must with which it will not work. |
| Resource requirements are minimum | Strict delivery management dictates the scope, functionality to be delivered, and adjustments to meet the deadlines. |
| Suitable for fixed or changing requirements | Depends heavily on customer interaction, so if customer is not clear, team can be driven in the wrong direction |
| Delivers early partial working solutions. | There is very high individual dependency, since there is minimum documentation generated. |
| Good model for environments that change steadily | Transfer of technology to new team members may be quite challenging due to lack of documentation |
| 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 |



**Agile Model Fig 1.3**