**Software Development Life Cycle (SDLC)**

**Description**

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. It consist of 5 stages, requirements engineering, analysis & design, implementation (coding), Testing, and deployment.

**Stages/Phases**

Requirements engineering

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.

Pitfalls

* Fuzzy, difficult to specify precisely
* Incomplete requirements
* Unstable requirements
* Misunderstanding user goals

Analysis and design

Uses a combination of text and diagrammatic forms to depict requirements and produce a representation of an entity that will later be built. Include architectural design, user interface design, and database design.

Pitfalls

* Not meeting the requirements
* Compromised design due to cost / time
* Different designs – which is better?
* Explosion of *derived* requirements
* Poor, ad-hoc design

Implementation (coding)

The implementation of software design starts in terms of writing program code in the suitable programming language and developing error-free executable programs efficiently

Pitfalls

* Tied down to certain technology
* Future enhancements not considered
* Different developers / teams
* Bad compromises made due to deadline
* Conflicting choices

- Cost vs Time

- Speed vs Memory Usage

Testing

Test are conducted to make sure that the software is reliable and meet the user’s needs

Pitfalls

* Testing done by persons who are not developers
* Testing an after-thought
* Testing done late in project
* Insufficient testing (test cases / test data)

Deployment

Application is distributed among a group of selected customers prior to official release, and application is delivered to customer.

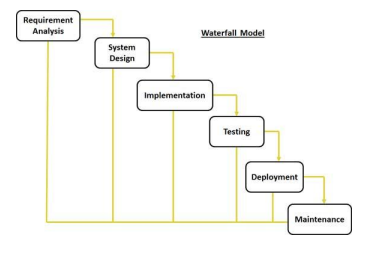
Pitfall

* Incorrect expectation of production environment
* Not testing out production environment
* Mismatch between development and production environment
* Version control

**Software development Models**

Waterfall model

In Waterfall model, all the phases of SDLC will function one after another in linear manner. When the first phase is finished then only the second phase will start and so on. This model assumes that everything is carried out and taken place perfectly as planned in the previous stage and there is no need to think about the past issues that may arise in the next phase. The sequential nature of model does not allow us go back and undo or redo our actions.



Pros

* Divide complex task into smaller and hence more manageable tasks (maintain control)
* Each task produces a well-defined deliverable (documentation driven)
* Easy to control and monitor because you deal with one activity at a time.

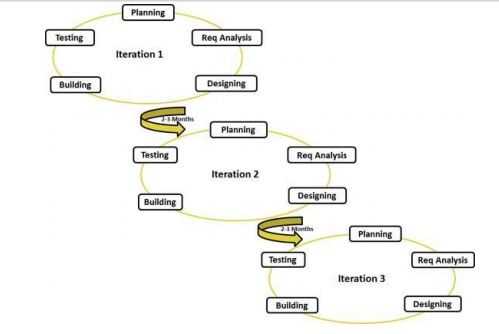
Cons

* Can only see the product at the end
* no opportunity to validate user requirements at early stages of development:
* If a problem is discovered at an earlier stage, nothing can be done about it (traditional waterfall)
* Does not stress the need for anticipating changes

Agile Model

Agile method breaks 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.

The end of the iteration a working product is displayed to the customer and important stakeholders. Every project needs to be handled differently and the existing methods need to be tailored to best suit the project requirements to deliver specific features



Pros

* 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

Cons

* 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

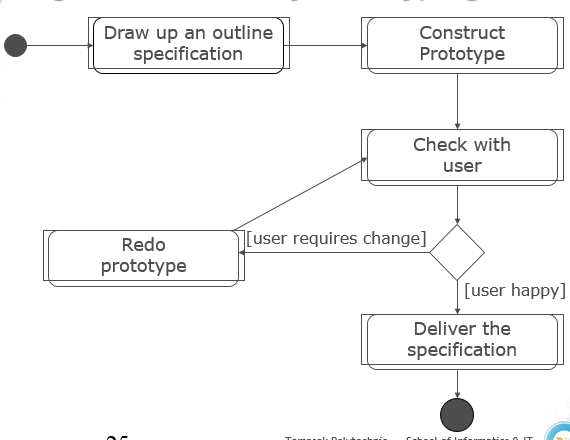
Prototyping model

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. The goal of using prototyping model is to clarify requirements. It can also be used to design the feasibility, verify that the new technology will work.

Software prototyping types

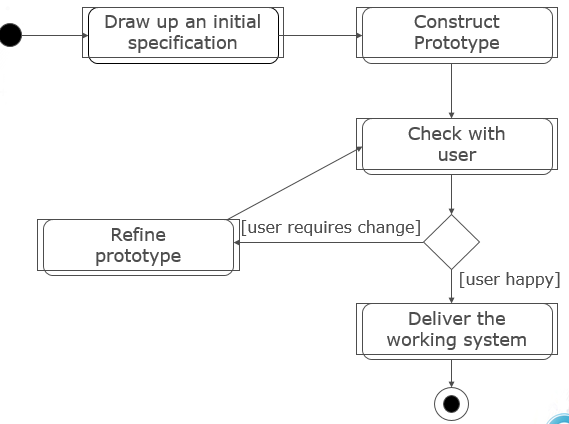
* Throwaway/Rapid Prototyping

Initial specifications for the software contains some requirements that are poorly understood, the throwaway prototype implementing those poorly understood requirements is built. After showing and validating with client, writes a full specification and throw away the prototype. Full-scale system is then build based on this specification



* Evolutionary Prototyping

Develop an initial system implementing parts of the system that are well understood, and show the developed system to user for comments for clarification of requirements and improvement. Refine the system and repeat the process to implement more parts of the system until the full system is completed.



Pros

* Clarify user requirements
* Specifications can be developed incrementally, giving users opportunity to change their mind

Cons

* Estimating, planning and managing a prototype project can be difficult because there is no regular deliverables e.g. how to predict how many iterations?
* Continual changes tend to corrupt software structure. Changes become more costly and difficult.

**References**

<http://www.tutorialspoint.com/software_engineering/software_development_life_cycle.htm>

<http://www.tutorialspoint.com/sdlc/sdlc_overview.htm>

<http://www.tutorialspoint.com/sdlc/sdlc_tutorial.pdf>