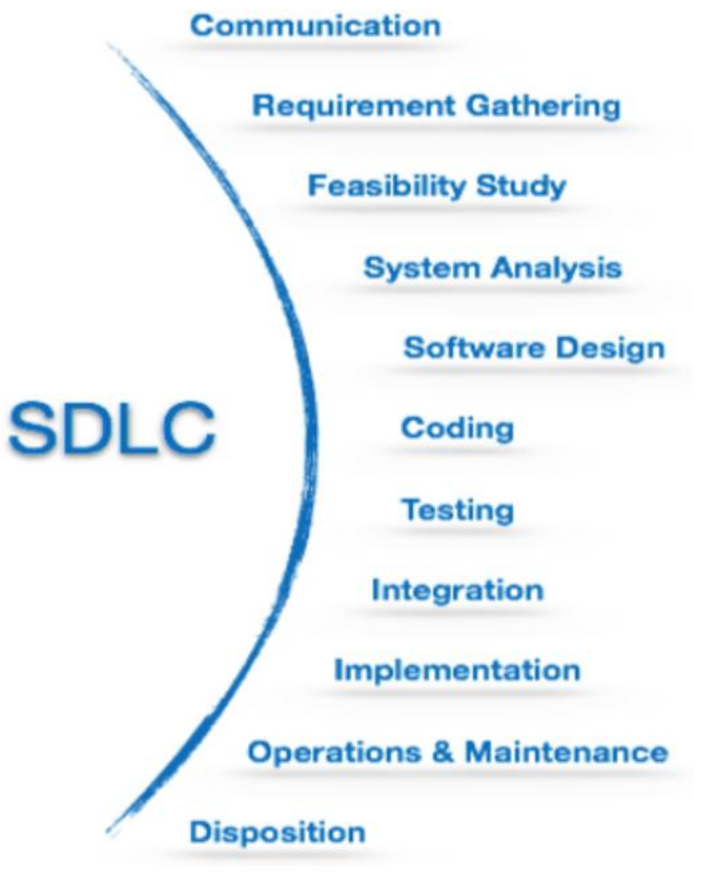
Week 2: SOFTWARE PROCESS MODELS

INTRODUCTION:

A **software process model** is an abstract representation of a software process. Each process model represents a process from a particular perspective, and thus provides only partial information about that process. In this section, a number of very general process models **(sometimes called process paradigms)** will be introduced and present these from an architectural perspective. That is, we see the framework of the process but not the details of specific activities.

These generic models are not definitive descriptions of software processes. Rather, they are abstractions of the process that can be used to explain different approaches to software development. You can think of them as process frameworks that may be extended and adapted to create more specific software engineering processes.



SOFTWARE DEVELOPMENT LIFE CYCLE

Software Development Life Cycle*, SDLC for short,* is a well-defined, structured sequence of stages in software engineering to develop the intended software product.

SDLC provides a series of steps to be followed to design and develop a software product efficiently. SDLC framework includes the following steps:

(1) **Communication**. This is the first step where the user initiates the request for a desired software product. The user contacts the service provider and tries to negotiate the terms, submits the request to the service providing organization in writing.

(2) **Requirement Gathering**. This step onwards the software development team works to carry on the project. The team holds discussions with various stakeholders from problem domain and tries to bring out as much information as possible on their requirements. The requirements are contemplated and segregated into user requirements, system requirements and functional requirements. The requirements are collected using a number of practices as given –

a. studying the existing or obsolete system and software,

b. conducting interviews of users and developers,

c. referring to the database or

d. collecting answers from the questionnaires.

(3) **Feasibility Study**. After requirement gathering, the team comes up with a rough plan of software process. At this step the team analyzes if a software can be designed to fulfill all requirements of the user, and if there is any possibility of software being no more useful. It is also analyzed if the project is financially, practically, and technologically feasible for the organization to take up. There are many algorithms available, which help the developers to conclude the feasibility of a software project.

(4) **System Analysis**. At this step the developers decide a roadmap of their plan and try to bring up the best software model suitable for the project. System analysis includes understanding of software product limitations, learning system related problems or changes to be done in existing systems beforehand, identifying and addressing the impact of project on organization and personnel etc. The project team analyzes the scope of the project and plans the schedule and resources accordingly.

(5) **Software Design**. Next step is to bring down whole knowledge of requirements and analysis on the desk and design the software product. The inputs from users and information gathered in requirement gathering phase are the inputs of this step. The output of this step comes in the form of two designs; **logical design**, and **physical design**. Engineers produce meta-data and data dictionaries, logical diagrams, data-flow diagrams, and in some cases pseudo codes.

(6) **Coding**. This step is also known as programming phase. The implementation of software design starts in terms of writing program code in the suitable programming language and developing error-free executable programs efficiently.

(7) **Testing**. An estimate says that 50% of whole software development process should be tested. Errors may ruin the software from critical level to its own removal. Software testing is done while coding by the developers and thorough testing is conducted by testing experts at various levels of code such as module testing, program testing, product testing, in-house testing, and testing the product at user's end. Early discovery of errors and their remedy is the key to reliable software.

(8) **Integration**. Software may need to be integrated with the libraries, databases, and other program(s). This stage of SDLC is involved in the integration of software with outer world entities.

(9) **Implementation**. This means installing the software on user machines. At times, software needs post-installation configurations at user end. Software is tested for portability and adaptability and integration related issues are solved during implementation.

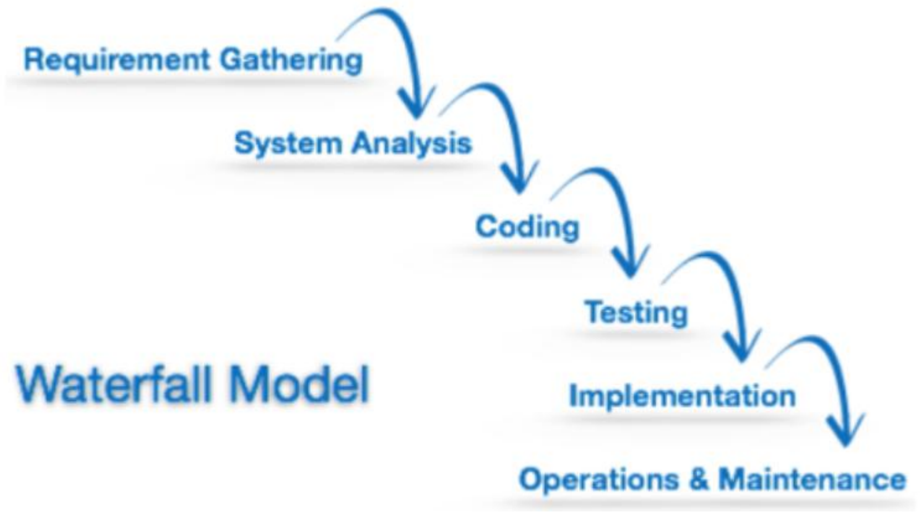
(10) **Operation and Maintenance**. This phase confirms the software operation in terms of more efficiency and less errors. If required, the users are trained on, or aided with the documentation on how to operate the software and how to keep the software operational. The software is maintained timely by updating the code

according to the changes taking place in user end environment or technology. This phase may face challenges from hidden bugs and real-world unidentified problems.

SOFTWARE DEVELOPMENT PARADIGM

The **software development paradigm** helps a developer to select a strategy to develop the software. A software development paradigm has its own set of tools, methods, and procedures, which are expressed clearly and defines software development life cycle. A few of software development paradigms or process models are defined as follows:

Waterfall Model



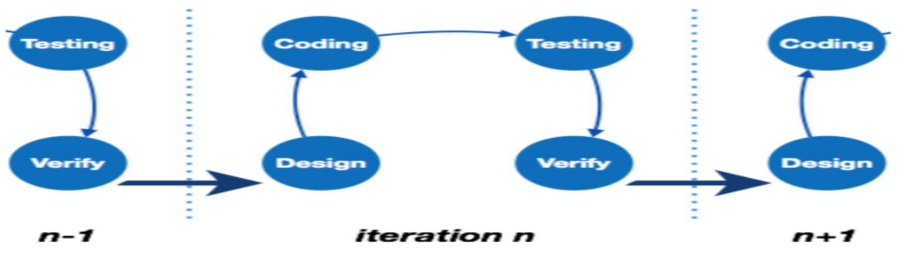
The first published model of the software development process was derived from more general system engineering processes (Royce, 1970). Because of the cascade from one phase to another, this model is known as the waterfall model or software life cycle.

Waterfall model is the simplest model of software development paradigm. All the phases of SDLC will function one after another in linear manner. That is, when the first phase is finished then only the second phase will start and so on. In practice, these stages overlap and feed information to each other. During design, problems with requirements are identified; during coding design problems are found and so on. The software process is not a simple linear model but involves a sequence of iterations of the development activities.

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. This model does not work smoothly if there are some issues left at the previous step. The sequential nature of model does not allow us to go back and undo or redo our actions.

This model is best suited when developers already have designed and developed similar software in the past and is aware of all its domains.

Iterative Model

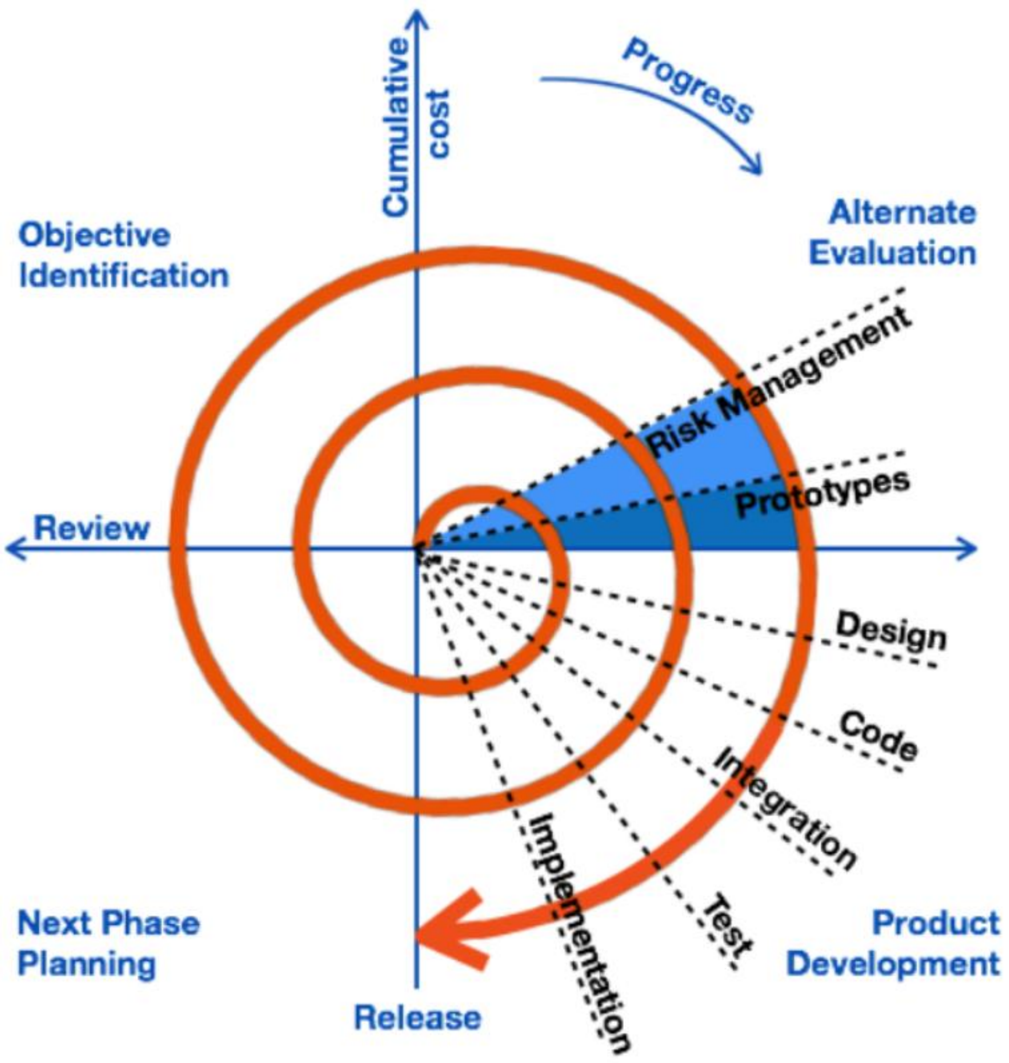


This model leads the software development process in iterations. It projects the process of development in cyclic manner repeating every step after every cycle of SDLC process.

The software is first developed on very small scale and all the steps are followed which are taken into consideration. Then, on every next iteration, more features and modules are designed, coded, tested, and added to the software. Every cycle produces a software, which is complete in itself and has more features and capabilities than that of the previous one.

After each iteration, the management team can do work on risk management and prepare for the next iteration. Because a cycle includes small portion of whole software process, it is easier to manage the development process but it consumes more resources.

Spiral Model

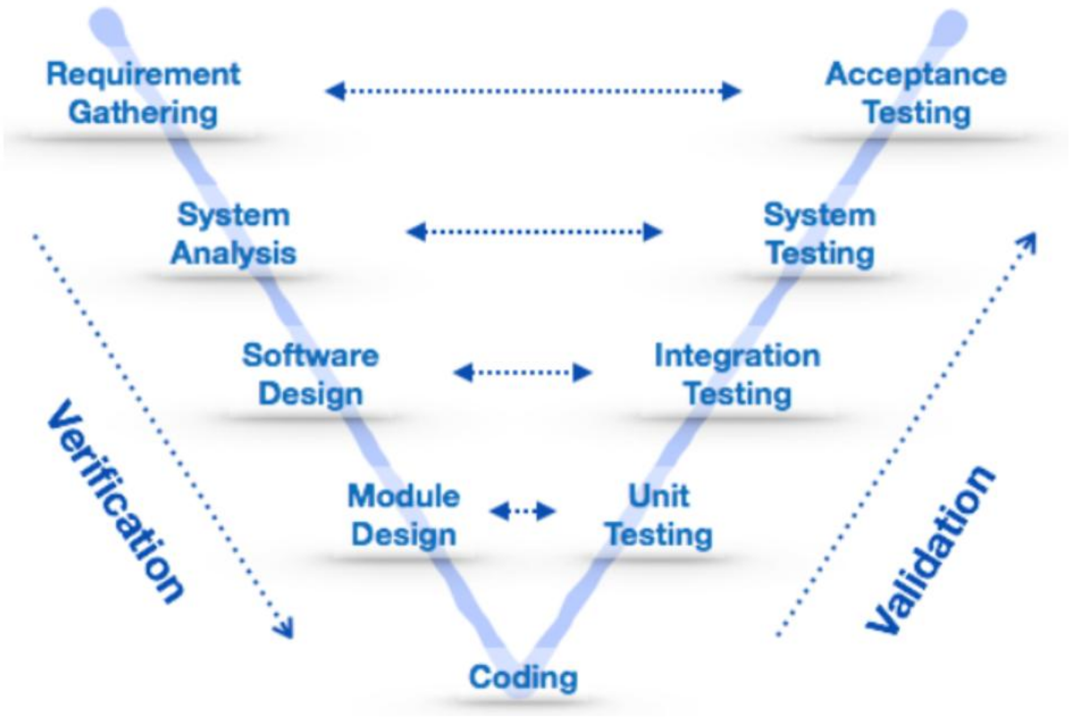


The spiral model of the software process was originally proposed by Boehm (Boehm, 1988). Rather than represent the software process as a sequence of activities with some backtracking from one activity to another, the process is represented as a spiral. Each loop in the spiral represents a phase of the software process. Thus, the innermost loop might be concerned with system feasibility, the next loop with requirements definition, the next loop with system design and so on.

Spiral model is a combination of both, iterative model and one of the SDLC model. It can be seen as if you choose one SDLC model and combined it with cyclic process (iterative model).

This model considers risk, which often goes unnoticed by most other models. The model starts with determining objectives and constraints of the software at the start of one iteration. Next phase is of prototyping the software. This includes risk analysis. Then one standard SDLC model is used to build the software. In the fourth phase of the plan of next iteration is prepared.

V – model

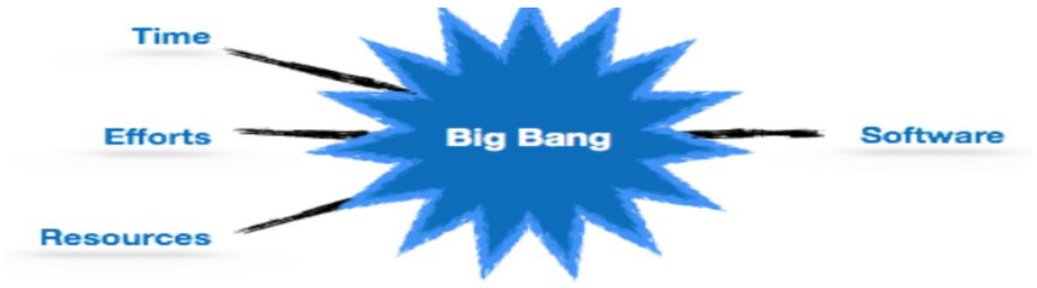


The major drawback of waterfall model is we move to the next stage only when the previous one is finished and there was no chance to go back if something is found wrong in later stages. V-Model provides means of testing of software at each stage in reverse manner.

At every stage, test plans and test cases are created to verify and validate the product according to the requirement of that stage. For example, in requirement gathering stage the test team prepares all the test cases in correspondence to the requirements. Later, when the product is developed and is ready for testing, test cases of this stage verify the software against its validity towards requirements at this stage.

This makes both verification and validation go in parallel. This model is also known as verification and validation model.

Big Bang Model

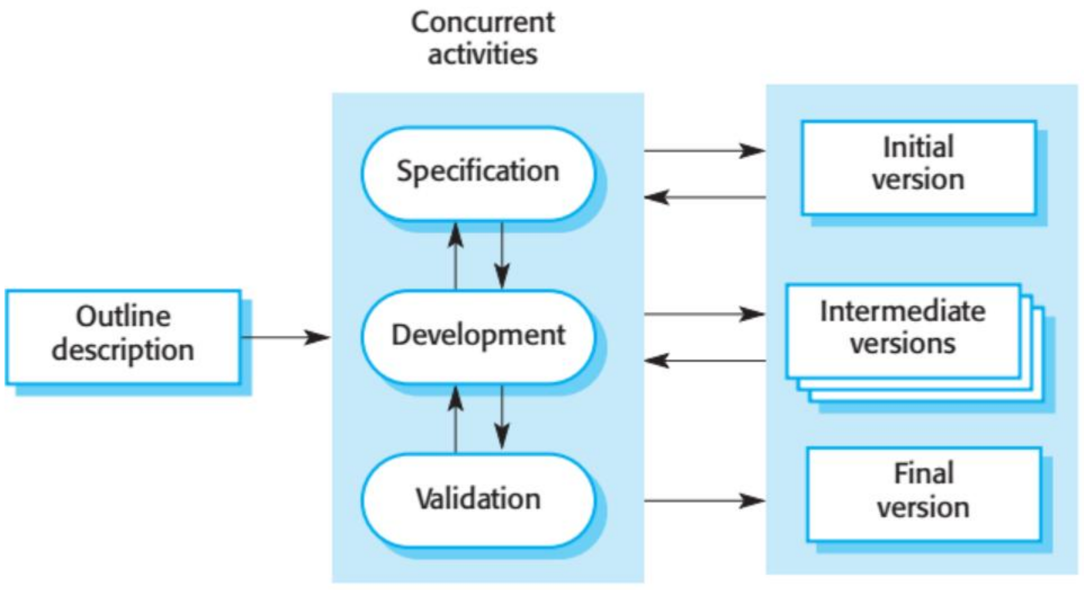


This model is the simplest model in its form. It requires little planning, lots of programming and lots of funds. This model is conceptualized around the big bang of universe. As scientists say that after big bang lots of galaxies, planets, and stars evolved just as an event. Likewise, if we put together lots of programming and funds, you may achieve the best software product.

For this model, very small amount of planning is required. It does not follow any process, or at times the customer is not sure about the requirements and future needs. So, the input requirements are arbitrary.

This model is not suitable for large software projects but good one for learning and experimenting.

Evolutionary development



Evolutionary development is based on the idea of developing an initial implementation, exposing this to user comment and refining it through many versions until an adequate system has been developed. Specification, development and validation activities are interleaved rather than separate, with rapid feedback across activities
There are two fundamental types of evolutionary development:

(1) **Exploratory development** where the objective of the process is to work with the customer to explore their requirements and deliver a final system. The development starts with the parts of the system that are understood. The system evolves by adding new features proposed by the customer.

(2) **Throwaway prototyping** where the objective of the evolutionary development process is to understand the customer's requirements and hence develop a better requirements definition for the system. The prototype concentrates on experimenting with the customer requirements that are poorly understood.

An evolutionary approach to software development is often more effective than the waterfall approach in producing systems that meet the immediate needs of customers. The advantage of a software process that is based on an evolutionary approach is that the specification can be developed incrementally. As users develop a better understanding of their problem, this can be reflected in the software system.