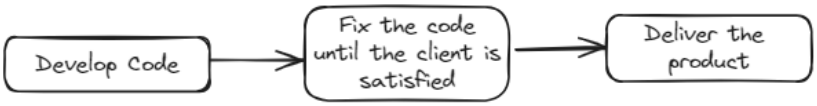
B1U1 Software Engineering and its models

Development Models

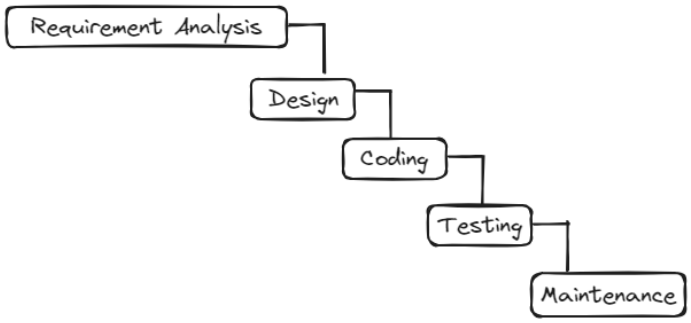
1) *Build and Fix Model*

It is a simple two phase model. In one phase, code is developed and in another, code is fixed.



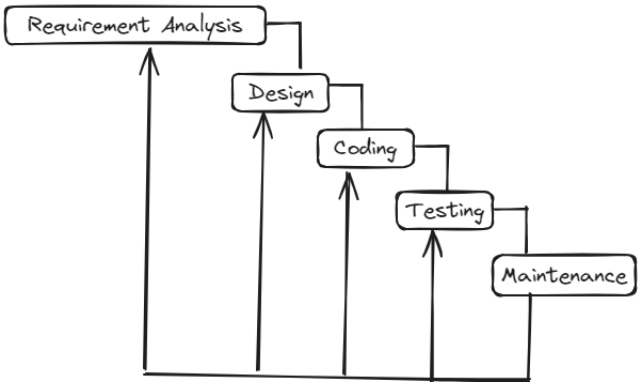
2) *Waterfall Model*

In this model, each phase of the life cycle is completed before the start of a new phase. It is actually the first engineering approach of software development.



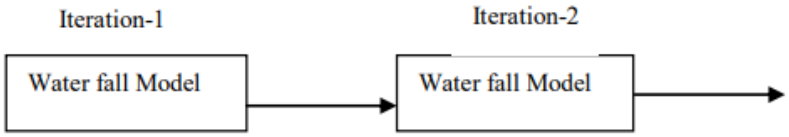
The waterfall model provides a systematic and sequential approach to software development and is better than the build and fix approach.But, in this model, complete requirements should be available at the time of commencement of the project, but in actual practice, the requirements keep on originating during different phases. The water fall model can accommodate the new requirements only in maintenance phase. Moreover, it does not incorporate any kind of risk assessment. In waterfall model, a working model of software is not available.

A slight modification of the waterfall model is a model with feedback. Once software is developed and is operational, then the feedback to various phases may be given.



3) *Iterative Enhancement Model*

This model was developed to remove the shortcomings of waterfall model. In this model, the phases of software development remain the same, but the construction and delivery is done in the iterative mode. In the first iteration, a less capable product is developed and delivered for use. This product satisfies only a subset of the requirements. In the next iteration, a product with incremental features is developed. Every iteration consists of all phases of the waterfall model. The complete product is divided into releases and the developer delivers the product release by release.

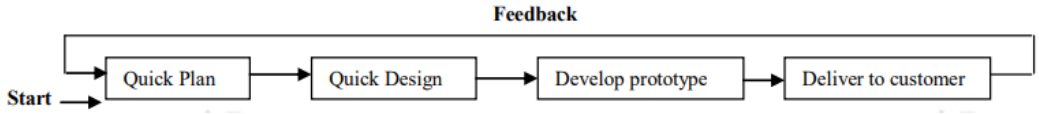


The main disadvantage of this model is that iteration may never end, and the user may have to endlessly wait for the final product. The cost estimation is also tedious because it is difficult to relate the software development cost with the number of requirements.

4) ***Prototyping Model***

In this model, a working model of actual software is developed initially. The prototype is just like a sample software having lesser functional capabilities and low reliability and it does not undergo through the rigorous testing phase. Developing a working prototype in the first phase overcomes the disadvantage of the waterfall model where the reporting about serious errors is possible only after completion of software development

The prototype model has the following features: (i) It helps in determining user requirements more deeply. (ii) At the time of actual product development, the customer feedback is available. (iii) It does consider any types of risks at the initial level



5) ***Spiral Model***

This model can be considered as the model, which combines the strengths of various other models. Conventional software development processes do not take uncertainties into account. Important software projects have failed because of unforeseen risks. The other models view the software process as a linear activity whereas this model considers it as a spiral process. This is made by representing the iterative development cycle as an expanding spiral. The following are the primary activities in this model:

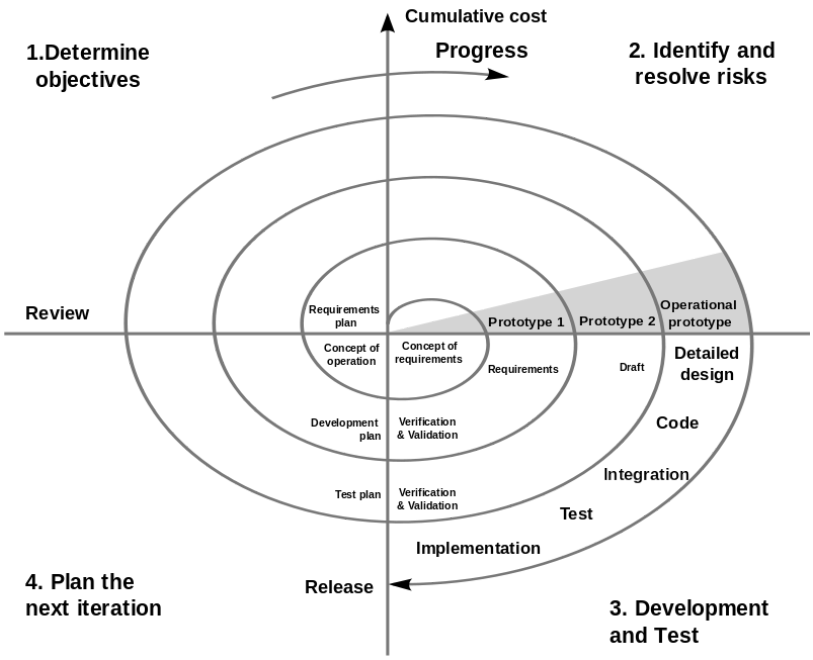
Finalising Objective: The objectives are set for the particular phase of the project.

Risk Analysis: The risks are identified to the extent possible. They are analysed and necessary steps are taken.

Development: Based on the risks that are identified, an SDLC model is selected and is followed.

Planning: At this point, the work done till this time is reviewed. Based on the review, a decision regarding whether to go through the loop of spiral again or not will be decided. If there is need to go, then planning is done accordingly.

In this model Software development starts with lesser requirements specification, lesser risk analysis, etc. The radical dimension this model represents cumulative cost. The angular dimension represents progress made in completing the cycle



6) RAD Approach

As the name suggests, this model gives a quick approach for software development and is based on a linear sequential flow of various development processes. The software is constructed on a component basis. Thus multiple teams are given the task of different component development. It increases the overall speed of software development. It gives a fully functional system within very short time. This approach emphasises the development of reusable program components. It follows a modular approach for development. The problem with this model is that it may not work when technical risks are high.

CAPABILITY MATURITY MODELS

The Capability Maturity Model (CMM) is a methodology used to develop and refine an organization's software development process. The model describes a five-level evolutionary path of increasingly organized and systematically more mature processes.

Level 1 (Initial):

Characteristics: Ad hoc development, no strategic approach, success depends on individual skills, unpredictable process, simple testing.

Key Points: Lack of a systematic approach, high dependency on individual capabilities.

Level 2 (Repeatable):

Characteristics: Basic project management policies and procedures, learning from experience, reuse of successful practices, disciplined development.

Key Points: Basic project management is established, some level of consistency and discipline in the development process.

Level 3 (Defined):

Characteristics: Well-defined, managed, and documented processes, training for staff, tailoring standard practices for new projects.

Key Points: Processes are well-documented and managed, staff is trained, standard practices are adapted for specific projects.

Level 4 (Managed):

Characteristics: Quantitative standards for products and processes, project analysis at an integrated organizational level, performance measurement, well-defined development processes.

Key Points: Detailed measurement and control of processes and products, focus on quantitative aspects, high-quality software.

Level 5 (Optimizing):

Characteristics: Continuous process improvement, proactive error prevention, analysis of weaknesses, adapting to new technologies based on cost-benefit analysis.

Key Points: Continuous improvement is the focus, the organization is proactive in preventing errors, adapts to new technologies.

Key Process Areas

The KPA is an indicative measurement of goodness of software engineering functions like project planning, requirements management, etc. The KPA consists of the following parameters:

Goals: Objectives to be achieved.

Commitments: The requirements that the organisation should meet to ensure the claimed quality of product.

Abilities : The capabilities an organisation has.

Activities : The specific tasks required to achieve KPA function.

Level 1 KPAs : There is no key process area at Level 1.

Level 2 KPAs:

- Software Project Planning: Gives concrete plans for software management.
- Software Project Tracing & Oversight: Establish adequate visibility into actual process to enable the organisation to take immediate corrective steps if Software performance deviates from plans.
- Requirements Management: The requirements are well specified to develop a contract between developer and customer.
- Software Subcontract Management: Select qualified software subcontractors and manage them effectively.
- Software Quality Assurance (SQA): To Assure the quality of developed product.
- Software Configuration Management (SCM): Establish & maintain integrity through out the lifecycle of project.

Level 3 KPAs:

- Organisation Process Focus (OPF): The organisations responsibility is fixed for software process activities that improve the ultimate software process capability.
- Training Program (TP): It imparts training to develop the skills and knowledge of organisation staff.
- Organisation Process Definition (OPD): It develops a workable set of software process to enhance cumulative long term benefit of organisation by improving process performance.

- Integrated Software Management (ISM): The software management and software engineering activities are defined and a tailor made standard and software process suiting to organisations requirements is developed.
- Software Product Engineering (SPE): Well defined software engineering activities are integrated to produce correct, consistent software products effectively and efficiently.
- Inter group co-ordination (IC): To satisfy the customer‟s needs effectively and efficiently, Software engineering groups are created. These groups participate actively with other groups.
- Peer reviews (PR): They remove defects from software engineering work products.

Level 4 KPAs:

- Quantitative Process Management (QP): It defines quantitative standards for software process.
- Software Quality Management (SQM): It develops quantitative understanding of the quality of Software products and achieves specific quality goals.

Level 5 KPAs:

- Defect Prevention (DP): It discovers the causes of defects and devises the techniques which prevent them from recurring.
- Technology Change Management (TCM): It continuously upgrades itself according to new tools, methods and processes.
- Process Change Management (PCM): It continually improves the software processes used in organisation to improve software quality, increase productivity and decrease cycle time for product development.

SOFTWARE PROCESS TECHNOLOGY

In general, different phases of SDLC are defined as following:

- Requirements Analysis
- Design
- Coding
- Software
- Testing
- Maintenance

Requirements Analysis

In the requirements analysis phase, the requirements are properly defined and noted down. The output of this phase is SRS (Software Requirements Specification) document written in natural language. According to IEEE, requirements analysis may be defined as (1) the process of studying user‟s needs to arrive at a definition of system hardware and software requirements (2) the process of studying and refining system hardware or software requirements.

Design

Design phase of software development deals with transforming the customer‟s requirements into a logically working system. Normally, design is performed in the following two steps:

i) Primary Design Phase: In this phase, the system is designed at block level. The blocks are created on the basis of analysis done in the problem identification phase. Different blocks are created for different functions emphasis is put on minimising the information flow between blocks. Thus, all activities which require more interaction are kept in one block.

ii) Secondary Design Phase: In the secondary design phase the detailed design of every block is performed.

Coding

The input to the coding phase is the SDD document. In this phase, the design document is coded according to the module specification. This phase transforms the SDD document into a high level language code.

Testing

Testing is the process of running the software on manually created inputs with the intention to find errors. In the process of testing, an attempt is made to detect errors, to correct the errors in order to develop error free software. The testing is performed keeping the user‟s requirements in mind and before the software is actually launched on a real system, it is tested. Testing is the process of executing a program with the intention of finding errors.

White Box Testing:

Definition: White box testing, also known as clear box testing, glass box testing, or structural testing, is a testing method where the tester has complete knowledge of the internal code, logic, and structure of the software being tested.

Objective: The primary goal of white box testing is to verify the correctness of the code, ensure that all statements, branches, and conditions have been executed and tested, and assess the internal structure of the code.

Testing Levels: It is usually applied at the unit, integration, and system testing levels.

Techniques: Techniques include statement coverage, branch coverage, path coverage, and data flow testing.

Test Cases: Test cases are designed based on the internal logic of the code, with the aim of exercising all possible paths and conditions.

Black Box Testing:

Definition: Black box testing is a testing method where the tester has no knowledge of the internal workings or implementation details of the software being tested. The focus is on testing the software's functionality without knowing how it achieves that functionality.

Objective: The primary goal of black box testing is to validate the software against its specified requirements, checking whether it produces the expected outputs for a given set of inputs.

Testing Levels: It can be applied at all levels of testing, including unit, integration, system, and acceptance testing.

Techniques: Techniques include equivalence partitioning, boundary value analysis, decision table testing, and state transition testing.

Test Cases: Test cases are designed based on the software's specifications, inputs, and expected outputs.

Maintenance

Maintenance in the normal sense means correcting the problems caused by wear and tear, but software maintenance is different. Software is either wrong in the beginning or later as some additional requirements have been added. Software maintenance is done because of the following factors. i) To rectify the errors which are encountered during the operation of software. ii) To change the program function to interface with new hardware or software. iii) To change the

program according to increased requirements. There are three categories of maintenance: i) Corrective Maintenance ii) Adaptive Maintenance iii) Perfective Maintenance