Software Engineering

Avanthika k nair c[1], Anju Jayan[2], Akshay tr[3]

B.tech III Year, Department of Computer Science and Engineering,
Kerala Technological University
avanthikak2016@gmail.com

anjujayan2016@gmail.com

akshaytr@yahoo.com

Abstract **- This paper explains the importance of Software Engineering in today's world and its significance in current lifestyle in binging down the manual efforts in various domains and in several walks of life. These includes the processes of designing and building something that serves a specific purpose and find an economical solution to problems. Software Engineering may be a systematic approach to the planning, development, operation, and maintenance of a software. Keywords – Software Engineering, Software Testing,**

Software Development, Programming, Software Design, Quality Analysis, Maintenance.

I. INTRODUCTION

Software engineering is the appliance of principles utilized within the world of engineering, which usually deals with physical systems, to the planning, development, testing, deployment and management of software systems. the world of software engineering applies the disciplined, structured approach to programming that's utilized in engineering for the software development with the stated goal of improving the quality , time and budget efficiency, in conjunction with the reassurance of structured testing and engineer certification.

II. OBJECTIVES OF SOFTWARE ENGINEERING:

A. *Maintainability*

It should be feasible for the software to evolve to satisfy changing requirements.

B. *Correctness*

A software package is correct, if the various requirements as laid out in the SRS document are correctly implemented.

C. *Reusability*

A software package has good reusability, if the various modules of the merchandise can easily be reused to develop new products.

D. *Testability*

Here software facilitates both the establishment of test criteria and therefore the evaluation of the software with reference to those criteria.

E. *Reliability*

It is an attribute of software quality. The extent to which a program can be expected to perform its desired function, over an arbitrary time period.

F. *Portability*

In this case, software are often transferred from one computing system or environment to a different.

Fig 1. Overview of Software Engineering

G. *Adaptability*

In this case, software allows differing system constraints and user must be satisfied by making changes to the software.

III. SOFTWARE REQUIREMENTS

Consistent with IEEE standard 729, a requirement is defined as follows:

- A condition or capability needed by a user to unravel a drag or achieve an objective
- A condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification or other formally imposed documents
- A documented representation of a condition or capability as in 1 and 2.

A. Classification of Software Requirements

A software requirement can be of 3 types:

- Functional requirements
- Non-functional requirements
- Domain requirements

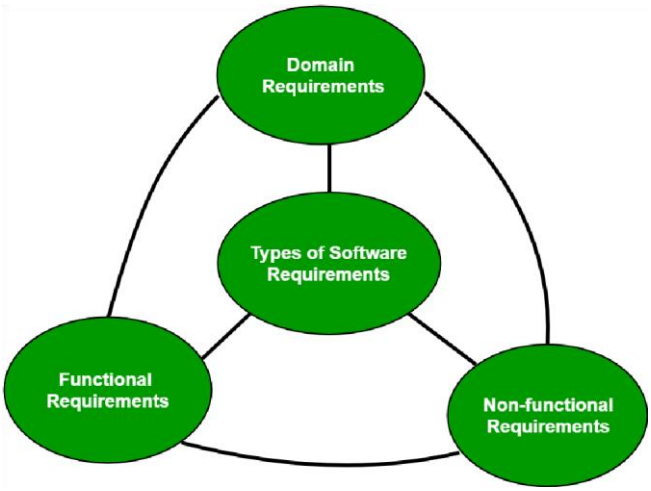

Fig 2. Types of software requirements

1) *Functional Requirements:* These are the requirements that the end user specifically demands as basic facilities that the system should offer. All these functionalities need to be necessarily incorporated into the system as a part of the contract. These are represented or stated in the form of input to be given to the system, the operation performed and the output expected. They are basically the requirements stated by the user which one can see directly in the final product, unlike the non-functional requirements.
   For example, in a hospital management system, a doctor should be able to retrieve the information of his patients. Each high-level functional requirement may involve several interactions or dialogues between the system and the outside world. In order to accurately describe the functional requirements, all scenarios must be enumerated.
   There are many ways of expressing functional requirements e.g., natural language, a structured or formatted language with no rigorous syntax and formal specification language with proper syntax.
2) *Non-functional requirements:* These are basically the quality constraints that the system must satisfy according to the project contract. The priority or extent to which these factors are implemented varies fromone project to other. They are also called non-behavioural requirements.

They basically deal with issues like:

- Portability
- Security
- Maintainability
- Reliability
- Scalability
- Performance
- Reusability
- Flexibility

NFR's are classified into following types:

- Interface constraints
- Performance constraints: response time, security, storage space, etc.
- Operating constraints
- Life cycle constraints: maintainability, portability, etc.
- Economic constraints

The process of specifying non-functional requirements requires the knowledge of the functionality of the system, as well as the knowledge of the context within which the system will operate.

3) *Domainrequirements*: Domain requirements are the requirements which are characteristic of a particular category or domain of projects. The basic functions that a system of a specific domain must necessarily exhibit come under this category. For instance, in an academic software that maintains records of a school or college, the functionality of being able to access the list of faculty and list of students of each grade is a domain requirement. These requirements are therefore identified from that domain model and are not user specific.

IV. SOFTWARE DEVELOPMENT LIFE CYCLE

The following figure is a graphical representation of the various stages of a typical SDLC.

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 amethodology for improving the quality of software and the overall development process.

Fig 3. Software development life cycle

A. Stages of SDLC

A typicalSoftware Development Life Cycle consists of the following stages −

1) Stage 1: Planning and Requirement Analysis

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.

Planning for the quality assurance requirements and identification of the risks associated with the project is also done in the planning stage. The outcome of the technical feasibility study is to define the various technical approaches that can be followed to implement the project successfully with minimum risks.

2) Stage 2: Defining Requirements

Once the requirement analysis is done the next step is to clearly define and document the product requirements and get them approved from the customer or the market analysts. This is done through an SRS (Software Requirement Specification) document which consists of all the product requirements to be designed and developed during the project life cycle.

3) Stage 3: Designing the Product Architecture

SRS is the reference for product architects to come out with the best architecture for the product to be developed. Based on the requirements specified in SRS, usually more than one design approach for the product architecture is proposed and documented in a DDS - Design Document Specification.

This DDS is reviewed by all the important stakeholders and based on various parameters as

risk assessment, product robustness, design modularity, budget and time constraints, the best design approach is selected for the product.

A design approach clearly defines all the architectural modules of the product along with its communication and data flow representation with the external and third party modules (if any). The internal design of all the modules of the proposed architecture should be clearly defined with the minutest of the details in DDS.

4) Stage 4: Building or Developing the Product

In this stage of SDLC the actual development starts and the product is built. The programming code is generated as per DDS during this stage. If the design is performed in a detailed and organized manner, code generation can be accomplished without much hassle.

Developers must follow the coding guidelines defined by their organization and programming tools like compilers, interpreters, debuggers, etc. are used to generate the code.

Different high level programming languages such as C, C+ +, Pascal, Java and PHP are used for coding. The programming language is chosen with respect to the type of software being developed.

5) Stage 5: Testing the Product

This stage is usually a subset of all the stages as in the modern SDLC models, the testing activities are mostly involved in all the stages of SDLC. However, this stage refers to the testing only stage of the product where product defects are reported, tracked, fixed and retested, until the product reaches the quality standards defined in the SRS.

6) Stage 6: Deployment in the Market and Maintenance

Once the product is tested and ready to be deployed it is released formally in the appropriate market. Sometimes product deployment happens in stages as per the business strategy of that organization. The product may first be released in a limited segment and tested in the real business environment (UAT- User acceptance testing).

Then based on the feedback, the product may be released as it is or with suggested enhancements in the targeting market segment. After the product is released in the market, its maintenance is done for the existing customer base.

B. Software Development Models

The software development paradigm helps 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:

1) Waterfall Model

Waterfall model is the simplest model of software development paradigm. It says the 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.

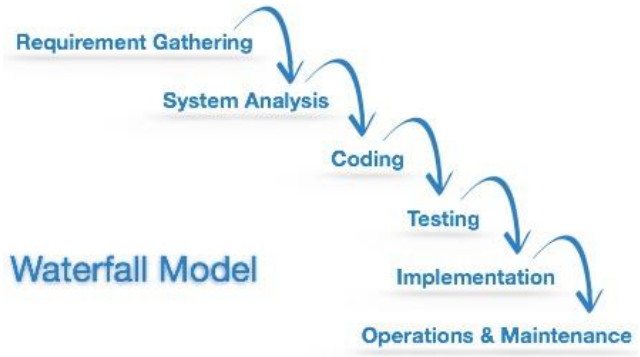

Fig 4. Waterfall model

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 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 are aware of all its domains.

2) 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.

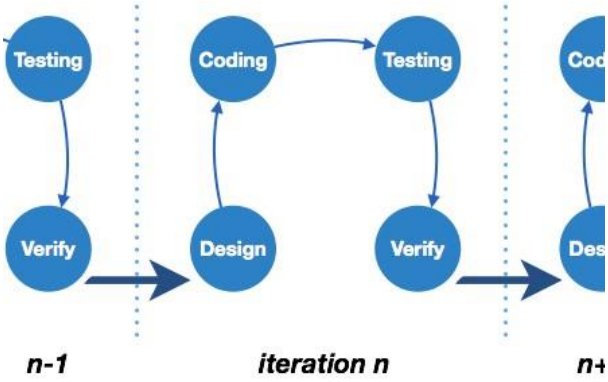

Fig 5. Iterative model

3) Spiral Model

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 combine it with cyclic process (iterative model).

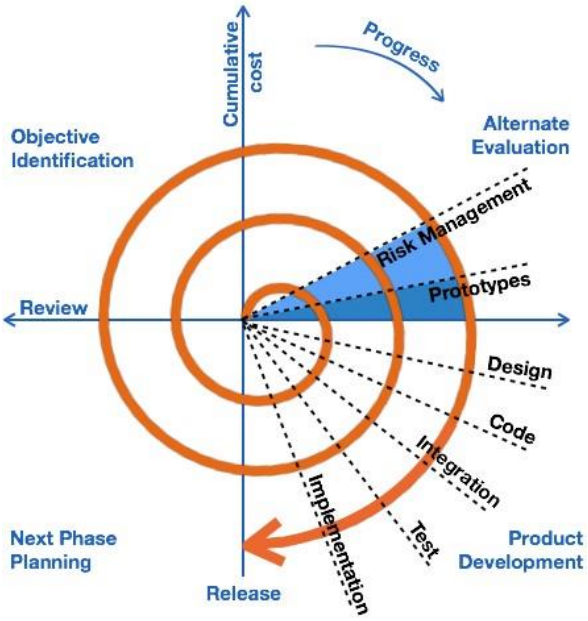

Fig 6. spiral model

This model considers risk, which often goes un-noticed 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.

4) 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.

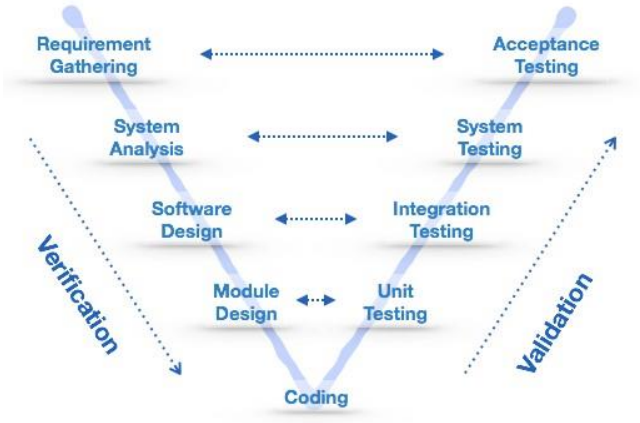

Fig 7. V-model

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.

5) 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.

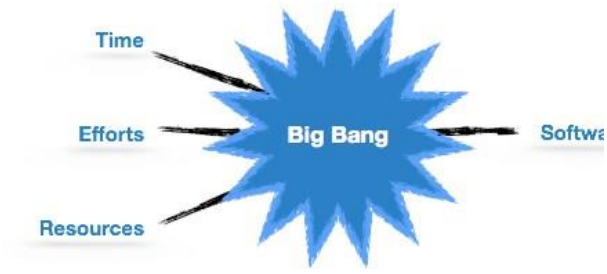

Fig 8. big bang model

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.

C. Software Development Paradigm

This Paradigm is known as software engineering paradigms where all the engineering concepts pertaining to the development of software are applied. It includes various researches and requirement gathering which helps the software product to build. It consists of –

- Requirement gathering
- Software design
- Programming

D. Software Design Paradigm

This paradigm is a part of Software Development and includes –

- Design
- Maintenance
- Programming

E. Programming Paradigm

This paradigm is related closely to programming aspect of software development. This includes

- Coding
- Testing,
- Integration

V. NEED OF SOFTWARE ENGINEERING

The need of software engineering arises because of higher rate of change in user requirements and environment on which the software is working.

A. *Large software* - It is easier to build a wall than to a house or building, likewise, as the size of software become large engineering has to step to give it a scientific process.

B. *Scalability*- If the software process were not based on scientific and engineering concepts, it would be easier to re-create new software than to scale an existing one.

C. *Cost*- As hardware industry has shown its skills and huge manufacturing has lower down he price of computer and electronic hardware. But the cost ofsoftware remains high if proper process is not adapted.

D. *Dynamic Nature*- The always growing and adapting nature of software hugely depends uponthe environment in which user works. If the nature of software is always changing, new enhancements need to be done in the existing one. This is where software engineering plays a good role.

E. *Quality Management*- Better process of software development provides better and quality software product.

VI. CHARACTERISTICS OF GOOD SOFTWARE

A software product can be judged by what it offers and how well it can be used. This software must satisfy on the following grounds:

- Operational
- Transitional
- Maintenance

Well-engineered and crafted software is expected to have the following characteristics:

A. Operational

This tells us how well software works in operations. It can be measured on:

- Budget
- Usability
- Efficiency
- Correctness
- Functionality
- Dependability
- Security
- Safety

B. Transitional

This aspect is important when the software is moved from one platform to another:

- Portability

- Interoperability
- Reusability
- Adaptability

C. Maintenance

This aspect briefs about how well a software has the capabilities to maintain itself in the ever-changing environment:

- Modularity
- Maintainability
- Flexibility
- Scalability

In short, Software engineering is a branch of computer science, which uses well-defined engineering concepts required to produce efficient, durable, scalable, in-budget and on-time software products.

VII. CONCLUSION

It is important to make technology decisions at the right time and for the right reasons. Good business decisions provide good people with appropriate supporting tools so they can produce good products. When it comes to software development, dealing with tough language issues head-on is one requirement for today's visionary manager. When combined with other software engineering considerations, a good language decision can support the development of cost-effective software systems that, in turn, provide valuable, reliable business support.

When developing good software systems, you should focus on the users' needs and, wherever possible, make use of replaceable and reusable modules – components. The overall software architecture should be constructed around the users' requirements.

We then introduced the role of modelling in the development of software. In particular, the concepts of object orientation allow us to represent users' requirements in a way that reflects our natural tendency to view the world around us in terms of objects. The way we relate the various activities of software development and associated artefacts (including models) was then described.

REFERENCE

[1] Ardis, M., Budgen, D., Hislop, G., Offutt, J., Sebern, M.,
[2] Visser, W., "Software Engineering 2014: Curriculum.
[3] Guidelines for Undergraduate Degree Programs in Software Engineering," joint effort of the ACM and the IEEE-Computer Society, 2014, http://securriculum.org.
[4] http://ncees.org/exams/fe-exam/, accessed 12/6/2015
[5] www.engineeringjobs.com, accessed 12/1/2015
[6] Bogost, I., The Atlantic, Nov. 5, 2015, "Programmers stop Calling Yourselves Engineers" .
[7] Pyster, A., "Software Engineering 2009 (GSwE2009): Curriculum Guidelines for Graduate Degree Programs in Software Engineering," Integrated Software & Systems Engineering Curriculum Project Stevens Institute (2009).
[8] http://2016.icse.cs.txstate.edu/keynotes, accessed 12/6/2105.
[9] Gallois, B., Sheppard, K., "The Design Spine: Revision Of The Engineering Curriculum To Include A Design Experience Each Semester." (2009) Paper presented at 1999 Annual Conference, Charlotte, North Carolina. https://peer.asee.org/755.
[10] Brown, H., & Ciuffetelli, D.C. (Eds.). Foundationalmethods: Understanding teaching and learning, p. 508. Toronto: Pearson Education. 2009.
[11] Hmelo-Silver, C., Duncan, R., & Clark, A., " Scaffoldingand Achievement in Problem-Based and InquiryLearning: A Response to Kischner, Sweller, and Clark (2006)". Educational Pyschologist , 42 (2). 2007.
[12] Bull, C., & Whittle, J., "Observations of a Software Engineering Studio: Reflecting with the Studio Framework." CSEE&T 2014 (pp. 74-83). Klagenfurt: IEEE. 2014.
[13] http://www.acm.org/education/curric_vols/CC2005-March06Final.pdf.
[14] Sturgis, J., private conversation, November 2015.
[15] http://www.nsf.gov/pubs/2014/nsf14542/nsf14542.htm
[16] Interim Report on 21st Century Cyber-Physical Systems Education National Academies Press,2014.