SOFTWARE ENGINEERING

SOFTWARE ENGINEERING

Software engineering is an engineering approach for software development. We can alternatively view it as a systematic collection of past experience. The experience is arranged in the form of methodologies and guidelines. A small program can be written without using software engineering principles. But if one wants to develop a large software product, then software engineering principles are indispensable to achieve a good quality software cost effectively. These definitions can be elaborated with the help of a building construction analogy.

PROJECT PLANNING

Once a project is found to be feasible, software project managers undertake project planning . Project planning is undertaken and completed even before any development activity starts. Project planning consists of the following essential activities:

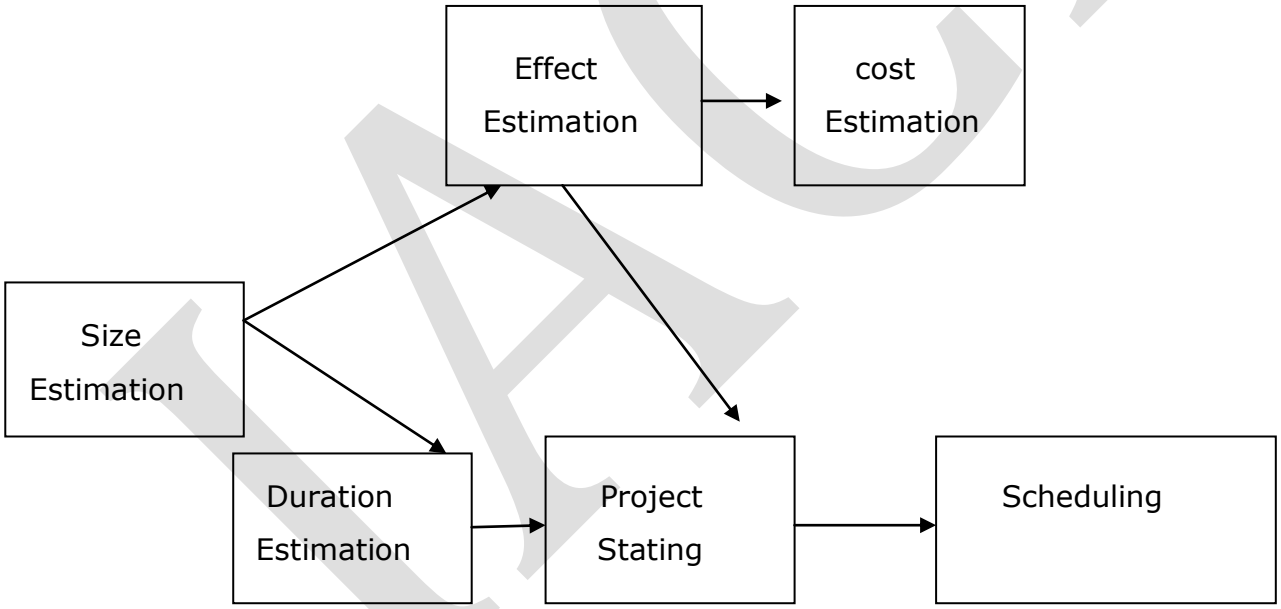
Estimating the following attributes of the project:

Project size: What will be problem complexity in terms of the accuracy of these estimations.

- Scheduling man power and other resources
- Staff organization and staffing plans
- Risk identification, analysis, and abatement planning
- Miscellaneous plans such as quality assurance plan, configuration management plan, etc

PRECEDENCE ORDERING AMONG PROJECT PLANING ACTIVITIES

The project planning activity is undertaken before the development starts to plan the activities to be undertaken during development. The project monitoring and control activities are undertaken once the development activities start with the aim of ensuring that the development proceeds as per plan and changing the plan whenever required to cope up with the situation. It is also the most fundamental parameter based on which all other planning activities are carried out. Other estimations such as estimation of effort, cost, resourced and project duration are also very important components of project planning



SLIDING WINDOW PLANNING

project planning requires utmost care and attention since commitment to unrealistic time and resource estimates results in schedule slippage. Schedule delays can cause customer dissatisfaction and adversely affect team morale. It can even cause project failure. However, project planning is a very challenging activity. Especially for large projects. It is very much difficult to make accurate plans. A part of this difficulty is due to the fact that the proper parameters, scope of the projects, project staff, etc. may change during the span of the project. In order to overcome this problem, sometimes project managers undertake project planning in stages. Planning from making big commitments too early. This technique of staggered planning is known as sliding window planning. In the sliding window technique, starting with an initial plan, the project is planned more accurately in successive development stages.

PROJECT MANAGEMENT CONCEPTS

Project management remains a very necessary activity when computer based systems and products are built. Project management involves the planning. Monitoring and control of the people, process, and event that occur as software evolves from a preliminary concept to an operational implementation. Everyone “manages” to some extent, but the scope of

management activities varies with the person doing it. A software engineer manages her day-to-day activities, planning, monitoring, and controlling technical tasks. Project managers plan, monitor, and control the work of a team of software engineers. Senior managers coordinate the interface between the business and the software professionals. Building computer software is a complex undertaking. Particularly if it involves many people working over a relatively long time. That's why software projects need to be managed.

Understand the four p's --- people, product, process, and project. people must be organized to perform software work effectively. Communication with the customer must occur so that product scope and requirements are understood. A process must be selected that is appropriate for the people and the product. The project must be planned by estimating effort and calendar time to accomplish work tasks; defining work products, establishing quality checkpoints, and establishing mechanisms to monitor and control work defined by the plan.

PROJECT PLANNING OBJECTIVES

The objective of software project planning is to provide a framework that enables the manager to make reasonable estimates of resources, cost, and schedule. These estimates are made within a limited time frame at the beginning of a software project and should be updated regularly as the project progresses. In addition, Estimates should attempt to define best case and worst case scenarios so that project outcomes can be process of information discovery that leads to reasonable estimates.

SYSTEM ENGINEERING

System engineering is the activity of specifying designing implementing, validating deploying and maintaining system. System engineer are not just concerned with software but also with hardware and the system's interactions with users and its environment. They must think about the services that the system provides the constraints under which the system must be built and operated and the ways in which the system is used to fulfill its purpose.

PROGRAM TESTING

Testing a program consists of providing the program with a set of test inputs (or test cases) and observing if the program behaves as expected, If the program falls to behave as expected, then the conditions under which failure occurs are noted for later debugging and correction.

Some commonly used terms associated with testing are:

- **Failure: This** is a manifestation of an error (or defect or bug). But, the mere presence of an error may not necessarily lead to a failure.
- **Test case:** This is the triplet [I,S,S], where I is the data input to the system. S is the state of the system at which the data is input, and O is the expected output of the system.
- **Test suite:** This is the set of all test cases with which a given software product is to be tested.

Design of test cases

Exhaustive testing of almost any non-trivial system is impractical due to the fact that the domain of input data values to most practical software systems is either extremely large or infinite. Therefore, we must design an optical test suite that is of reasonable size and can uncover as many errors existing in the system as possible. Testing a system using a large collection of test cases that are selected at random does not guarantee that all (or even most) of the error in th system will be uncovered.

BLACK BOX TESTING

In the black – box testing, test cases are designed from an examination of the input/output values only and no knowledge of design or code is required. The following are the two main approaches to designing black box test cases.

- Equivalence class portioning
- Boundary value analysis

Equivalence Class Partitioning

The domain of input values to a program is partitioned into a set of equivalence classes. This partitioning is don such that the behavior of the program is similar for every input data belonging to the same equivalence class. The main idea behind defining the equivalence classes is that testing the code with any one value belonging to an equivalence class is as good as testing the software with any other value belonging to that equivalence class. Equivalence classes for a software can be designed by examining the input data and output data. The following are some general guidelines for designing the equivalence classes:

1. If the input data values to a system can be specified by a range of values. Then one valid and two invalid equivalence classes should be defined.

2. If the input data assumes values from a set of discrete members of some domain, then one equivalence class for valid input values and another equivalence class for invalid input values should be defined.

Example : For a software that computes the square root of an input integer which can assume values in the range of 0 to 5000, there are three equivalence classes; The set of negative integers, the set of integers in the range of 0 to 5000, and the integers larger than 5000. Therefore, the test cases must include representatives for each of the three equivalence classes and a possible test set can be:

{-5, 500, 6000}.

Boundary Value Analysis

A type of programming error frequently occurs at the boundaries of different equivalence class of inputs. The reason behind such errors might purely be due to psychological factors. Programmers often fail to see the special processing required by the input values that lie at the boundary of the different equivalence classes. For example, programmers may improperly use < instead of <=; or conversely <= for<.Boundary value analysis leads to selection of test cases at the boundaries of the different equivalence classes.

Example: For a function that computers the square root of integer values in the range of 0 and 5000. The test cases must include the following values:

{0,-1, 5000, 5001}.

WHITE BOX TESTING

One white – box testing strategy is said to be stronger than another strategy, if all types of errors detected by the first testing strategy is also detected by the second testing strategy, and the second testing strategy additionally detects some more types of errors. When two testing strategies detect errors that are different at least with respect to some types of errors, then they are called complementary.

VERFICATION AND VALIDATION

Verification is the process of determining whether the output of one phase of software development conforms to that of its previous phase, whereas validation is the process of determining whether a fully developed system conforms to its requirements specification. Thus while verification is concerned with phase containment of errors, the aim of validation is that the final product be error free.

SOFTWARE DEVELOPMENT LIFE CYCLE

The system Development Life Cycle (SDLC), or Software Development Life Cycle in system engineering, information system and software engineering, information systems and software engineering, is the process of creating or altering system , and the models and methodologies that people use to develop these system. The concept generally refers to computer or information system. System and development life Cycle (SDLC) is a process used by a systems analyst to develop an information system, including requirements, validation, training, and user (stake holder) ownership. Any SDLC should result in a high quality system that meets or exceeds customer expectations, reaches completion within time and cost estimates, works effectively and efficiently in the current and planned information Technology infrastructure, and is inexpensive to maintain and cost-effective to enhance. These stages can be characterized and divided up in different ways, including the following:

- **Project planning, feasibility** study: Establishes a high level view of the intended project and determines its goals.
- **System analysis**, **requirements definition :**

  Refines project goals into defined functions and operation of the intended application. Analyzes end – user information needs.
- **System design:** Describes desired features and operations in detail, including screen layouts, business rules process diagrams, pseudo code and other documentation.
- **Implementation :** The real code is written here's
- **Integration and testing**: Brings all the pieces together into a special testing environment, then checks for error, bugs and interoperability.
- **Acceptance, installation, deployment:** The final stage of initial development where the software is put into production and runs actual business.

Maintenance: What happens during the rest of the software's life: change, correction, additions, moves to a different computing platform and more. This, the least glamorous and perhaps most important step of all, goes on seemingly forever.

Complementary Software development methods to system Development Life Cycle (SDLC) are:

- Software prototyping
- Joint Application Design (JAD)
- Rapid Application Development (RAD)
- Extreme Programming (XP)
- Open Source Development
- End – user Development

The waterfall mode is a sequential software development process, in which progress is seen as flowing steadily downwards (Like a waterfall) through the phases of Conception, Initiation, Analysis, Design, Construction, Testing and Maintenance.

In original waterfall model, the following phases are followed in order:

- Requirements Specification
- Design
- Construction (AKA implementation or coding)
- Integration

- Testing and debugging (AKA Validation)
- Installation
- Maintenance

Computing- aided software engineering (CASE) is the scientific application of a set of tools and methods to a software system which is meant to result in high quality, defect-free, and maintainable software products. [1] It also refers to methods for the development of information system together with automated tools that can be used in the software development process. CASE tools are a class of software that automates many of the activities involved in various life cycle- phases

Existing CASE tools can be classified along 4 different dimensions :

- Life – cycle support
- Integration Dimension
- Construction Dimension
- Knowledge Based CASE Dimension

In software engineering, multi-tier architecture (often referred to as n-tier architecture) is a client- server architecture in which the presentation, the application processing. And the data management are logically separate 96| Information Technology

Processes. For example, an application that uses middleware to service data requests between a user and a database employs multi-tier architecture. The most widespread use of multi-tier architecture is the three-tier architecture.

N-tier application architecture provides a model for developers to create a flexible and reusable application. By breaking up an application into tiers, developers only have to modify or add a specific layer, rather than have to rewrite the entire application over. There should be a presentation tier, a business or data access tier, and a data tier.

The concepts of layer and tier are often used interchangeably. However, one fairly common point of view is that is indeed difference, and that a layer is a logical structuring mechanism for the elements that make up the software solution, while a tier is a physical structuring mechanism for the system infrastructure. Three-tier is a client – server architecture in which the user interface, functional process logic (" business rules"), computer data storage and data access are developed and maintained as independent modules, most often on separate platforms.

Three –tier architecture has the following three tiers:

(1) Presentation tier: This is the topmost level of the application. The presentation tier displays information related to such services as browsing merchandise, purchasing, and shopping cart contents .It communicates with other tiers by outputting results to the browser/client tier and all other tiers in the network.

(2) Application tier (business logic, logic tier, data access tier, or middle tier): The logic tier is pulled out from the presentation tier and, as its own layer; it controls an application's functionality by performing detailed processing.

(3) Data tier: This tier consists of database servers. Here information is stored and retrieved. This tier keeps data neutral and independent from application servers or business logic. Giving data its own tier also improves scalability and performance.

Most programming: Modular programming (also known as top down design and stepwise refinement) is a software design technique that increases the extent to which software is composed of separate, interchangeable components called modules by breaking down program functions into modules, each of which accomplishes one function and contains everything necessary to accomplish this. Conceptually, modules represent a separation of concerns, and improve maintainability by enforcing logical boundaries between components. Modules are typically incorporated into the program through interface. A module interface expresses the elements that are provided and required by the module. The elements defined in the interface are detectable by other modules. The implementation contains the working code that corresponding to the elements declared in the interface.

Software Metric: A software metric is a measure of some property of a piece of software or its specifications. Since quantitative measurements are essential in all sciences, there is a continuous effort by computer science practitioners and theoreticians to bring similar approaches to software development. The goal is obtaining objective, reproducible and quantifiable measurements, which may have numerous valuable application, quality assurance testing, software debugging, software performance optimization, and optimal personnel task assignment.