UNIT 2:

Software Project Planning

Software Project Planning

A Software Project is the complete methodology of programming advancement from requirement gathering to testing and support, completed by the execution procedures, in a specified period to achieve intended software product.

Need of Software Project Management

Software development is a sort of all new streams in world business, and there's next to no involvement in structure programming items. Most programming items are customized to accommodate customer's necessities. The most significant is that the underlying technology changes and advances so generally and rapidly that experience of one element may not be connected to the other one. All such business and ecological imperatives bring risk in software development; hence, it is fundamental to manage software projects efficiently.

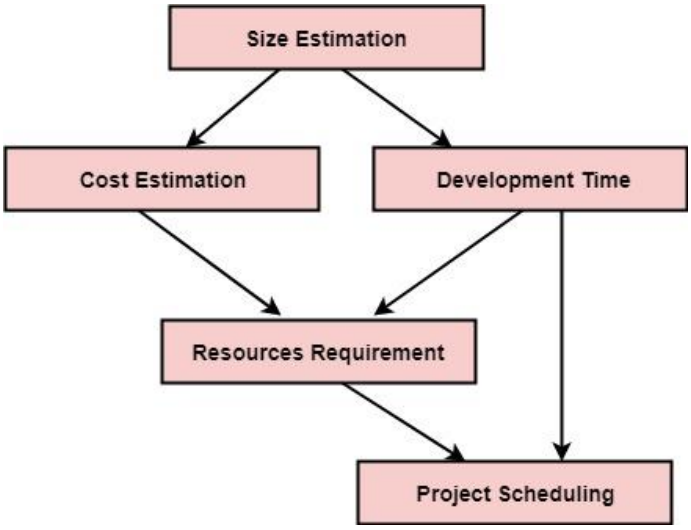
Software Project Manager

Software manager is responsible for planning and scheduling project development. They manage the work to ensure that it is completed to the required standard. They monitor the progress to check that the event is on time and within budget. The project planning must incorporate the major issues like size & cost estimation scheduling, project monitoring, personnel selection evaluation & risk management. To plan a successful software project, we must understand:

- Scope of work to be completed
- Risk analysis

- The resources mandatory
- The project to be accomplished
- Record of being followed

Software Project planning starts before technical work start. The various steps of planning activities are:



The size is the crucial parameter for the estimation of other activities. Resources requirement are required based on cost and development time. Project schedule may prove to be very useful for controlling and monitoring the progress of the project. This is dependent on resources & development time.

Decomposition Techniques

Decomposition techniques are methods used in software estimation to break down a project into smaller, more manageable components. These techniques help in estimating the size, effort, and resources required for developing a software system. Here are three common decomposition techniques used in software estimation:

1. **S/W Sizing (Software Sizing)**: Software sizing is a decomposition technique where the focus is on breaking down the software project into smaller functional components that can be measured in terms of size metrics. These metrics can include lines of code (LOC), function points, or use case points. The basic idea is to quantify the software by measuring the size of its individual components. By knowing the size of each component, one can estimate the effort and resources required for the development.

Example: In the function point analysis technique, the software is divided into logical components like external inputs, external outputs, external inquiries, internal logical files, and external interface files. Each component is then assigned a weight based on complexity, and the total function points are calculated. The function points can then be used to estimate the effort and cost.

2. **Problem-Based Estimation (Top-Down Estimation):** Problem-based estimation involves breaking down the software project based on the problem domain or functionality. The estimation starts with a high-level view of the entire project and then gradually breaks it down into smaller sub-problems or modules. Each sub-problem is further decomposed into finer-grained components until a detailed estimation is achieved. This approach is usually top-down, starting from the overall system and diving into subsystems and modules.

Example: When estimating the development of a web application, the problem-based approach may start by identifying major functional areas like user authentication, database management, user interface design, and so on. Each area is then decomposed into smaller tasks, such as login functionality, registration, profile management, etc., until a detailed estimation is obtained.

3. **Process-Based Estimation (Bottom-Up Estimation):** Process-based estimation involves decomposing the software project based on the development process or tasks required to complete the project. In this approach, each task or activity required for software development is identified and estimated separately. The individual estimates are then aggregated to get an overall estimate for the entire project.

Example: In a software development project, the process-based approach may involve breaking down the tasks into stages like requirements gathering, design, coding, testing, and deployment. Each stage is then further decomposed into specific activities like creating user stories, designing database schema, writing code for individual modules, and conducting unit testing. Estimates for each activity are made, and then all the estimates are summed up to obtain the total effort and resources required for the project.

Each of these decomposition techniques has its advantages and limitations, and the choice of technique depends on the nature and complexity of the software project, as well as the available information and historical data. Skilled estimators often use a combination of these techniques to arrive at more accurate and reliable software estimates.

Models of Cost Estimation

Software development efficiency would alleviate the present challenges of software production, which have resulted in cost overruns or even project cancellations. Software engineering cost models, like any other discipline, has had its share of difficulties. The fast-paced nature of software development has made developing parametric models that deliver high accuracy for software development in all disciplines very challenging. S/w development expenses are rising at an extraordinary pace, and practitioners are constantly lamenting their inability to effectively forecast the costs involved. Software models help to describe the development lifecycle and anticipate the cost of building a software product. Many software estimating models have emerged during the past two decades as a result of academics' pioneering work. Because most models are private, they cannot be compared and contrasted in terms of model structure. The functional shape of these models is determined by theory or experimentation. These are the following −

COCOMO 81

(a) COCOMO fundamentals

The term COCOMO stands for Constructive Cost Model. Barry Boehm's book Software Engineering Economics was initially published in 1981. Due to the model's ease of transparency, it provides the magnitude of the project's expense. It is designed for small projects since it has a limited number of cost drivers. When the team size is small, i.e. when the staff is tiny, it is helpful.

It's useful for getting a quick, early, rough estimate of software costs, but its accuracy is limited due to the lack of factors to account for differences in hardware constraints, personnel quality and experience, use of modern tools and techniques, and other project attributes that are known to have a significant impact on s/w costs.

$$EFFORT = a^* (KDSI)b \quad EFFORT = a^* (KDSI)$$

Constants a and b have different values depending on the project type. The KLOC is the projected number of delivered lines of code for the project.

The following are the three kinds of modes available in COCOMO −

- Organic Mode − A modest, basic software project involving a small group of people with prior application knowledge. Efforts, E, and Development, D are the following −

$$E = 2.4*(KLOC)^{1.05}$$

$$D=2.5*(E)^{0.38}$$

- Semi-detached Mode − An intermediate software project in which teams with varying levels of expertise collaborate.

$$E= 3.0*(KLOC)^{1.12}$$

$$D=2.5*(E)^{0.35}$$

- Embedded Mode − A software project that must be built under strict hardware, software, and operational limitations.

$$E= 3.6*(KLOC)^{1.20}$$

D= 2.5*(E)^0.32

(b) COCOMO Intermediate

It assesses software development effort as a function of program size and a set of cost drivers, which include a subjective assessment of goods, hardware, employees, and project characteristics.

It's appropriate for medium-sized tasks. Intermediate to basic and advanced COCOMO are the cost drivers. Cost factors influence product dependability, database size, execution, and storage. The team has a modest size. The COCOMO intermediate model looks like this −

EFFORT = a*(KLOC)b*EAF

Here, effort is measured in person-months, and KLOC is the project's projected amount of delivered lines of code.

(c) COCOMO in depth

It's designed for large-scale undertakings. The cost drivers are determined by requirements, analysis, design, testing, and maintenance. The team is rather big. The comprehensive COCOMO Model incorporates all of the characteristics of the intermediate version, as well as an evaluation of the cost driver's impact on each phase of the software engineering process (analysis, design, and so on).

COCOMO-II

COCOMO II is a research project that began in 1994 at USC. It places a strong emphasis on non-sequential and quick development process models, reengineering, reuse-driven methodologies, object-oriented approaches, and so on. It is the outcome of a combination of three models: application composition, early design, and post architecture.

- On projects that employ Integrated Computer Aided Software Engineering technologies for fast application development, the Application Composition model is used to estimate effort and schedule. It is based on the concept of Object Points (Object Points are a tally of the screens, reports and 3 GL language modules developed in the application).
- The Early Design Model entails looking at alternative system designs and operational approaches.
- The Post-Architecture Model is utilized when the apex level design is complete and detailed information about the project is available, and the software architecture is well-defined and well-known, as the name implies. It is a comprehensive expansion of the Early-Design paradigm, accounting for the whole development life-cycle. This is a COCOMO model that ranges from lean to intermediate and is defined as follows −

  EFFORT = 2.9(KLOC)^1.10

The Software Equation

SOFTWARE ENGINEERING

The software equation is a dynamic multivariable model that assumes a specific distribution of effort over the life of a software development project. The model has been derived from productivity data collected for over 4000 contemporary software projects. Based on these data, an estimation model of the form

E = [LOC B0.333/P]3 (1/t4) where

E = effort in person-months or person-years
t = project duration in months or years
B = “special skills factor”

P = “productivity parameter” that reflects:

• Overall process maturity and management practices
• The extent to which good software engineering practices are used
• The level of programming languages used
• The state of the software environment
• The skills and experience of the software team
• The complexity of the application

Typical values might be P = 2,000 for development of real-time embedded software; P = 10,000 for telecommunication and systems software; P = 28,000 for business systems applications. The productivity parameter can be derived for local conditions using historical data collected from past development efforts. It is important to note that the software equation has two independent parameters:
(1) an estimate of size (in LOC) and (2) an indication of project duration in calendar months or years.

To simplify the estimation process and use a more common form for their estimation model, Putnam and Myers suggest a set of equations derived from the software equation. Minimum development time is defined as

tmin = 8.14 (LOC/P)0.43 in months for tmin > 6 months

E = 180 Bt3 in person-months for E ≥ 20 person-months

tmin = 8.14 (33200/12000)0.43
tmin = 12.6 calendar months
E = 180 0.28 (1.05)3
E = 58 person-months