SOFTWARE ENGINEERING

UNIT-2

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.

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

CODECHAMP
CREATED WITH ARBOK

Software Project Planning Objectives:

Project planning is an organized and integrated management process, which focuses on activities required for successful completion of the project. It prevents obstacles that arise in the project such as changes in projects or organization's objectives, non-availability of resources, and so on. Project planning also helps in better utilization of resources and optimal usage of the allotted time for a project. The other objectives of project planning are listed below.

- It defines the roles and responsibilities of the project management team members.
- It ensures that the project management team works according to the business objectives.
- It checks feasibility of the schedule and user requirements.
- It determines project constraints.

Decomposition Techniques:

The Project Estimation Approach that is widely used is Decomposition Technique. Decomposition techniques take a divide and conquer approach. Size, Effort and Cost estimation are performed in a stepwise manner by breaking down a Project into major Functions or related Software Engineering Activities.

Software Sizing:

Estimation of the size of the software is an essential part of Software Project Management. It helps the project manager to further predict the effort and time which will be needed to build the project. Various measures are used in project size estimation. Some of these are:

A. Lines of Code
B. Number of entities in ER diagram
C. Total number of processes in detailed data flow diagram
D. Function points

A. Lines of Code (LOC):

As the name suggests, LOC count the total number of lines of source code in a project. The units of LOC are:

- KLOC- Thousand lines of code
- NLOC- Non-comment lines of code
- KDSI- Thousands of delivered source instruction

CODECHAMP
CREATED WITH ARBOK

The size is estimated by comparing it with the existing systems of the same kind. The experts use it to predict the required size of various components of software and then add them to get the total size.

Advantages:

- Universally accepted and is used in many models like COCOMO.
- Estimation is closer to the developer’s perspective.
- Simple to use.

Disadvantages:

- Different programming languages contain a different number of lines.
- No proper industry standard exists for this technique.
- It is difficult to estimate the size using this technique in the early stages of the project.

B. Number of entities in ER diagram:

ER model provides a static view of the project. It describes the entities and their relationships. The number of entities in ER model can be used to measure the estimation of the size of the project. The number of entities depends on the size of the project. This is because more entities needed more classes/structures thus leading to more coding.

Advantages:

- Size estimation can be done during the initial stages of planning.
- The number of entities is independent of the programming technologies used.

Disadvantages:

- No fixed standards exist. Some entities contribute more project size than others.
- Just like FPA, it is less used in the cost estimation model. Hence, it must be converted to LOC.

C. Total number of processes in detailed data flow diagram:

Data Flow Diagram (DFD) represents the functional view of software. The model depicts the main processes/functions involved in software and the flow of data between them. Utilization of the number of functions in DFD to predict software size. Already existing processes of similar type are studied and used to estimate the size of the process. Sum of the estimated size of each process gives the final estimated size.

Advantages:

- It is independent of the programming language.
- Each major process can be decomposed into smaller processes. This will increase the accuracy of estimation

Disadvantages:

CODECHAMP
CREATED WITH ARBOK

- Studying similar kinds of processes to estimate size takes additional time and effort.
- All software projects are not required for the construction of DFD.

D. Function Point Analysis:

In this method, the number and type of functions supported by the software are utilized to find FPC (function point count). The steps in function point analysis are:

- Count the number of functions of each proposed type.
- Compute the Unadjusted Function Points (UFP).
- Find Total Degree of Influence (TDI).
- Compute Value Adjustment Factor (VAF).
- Find the Function Point Count (FPC).

Advantages:

- It can be easily used in the early stages of project planning.
- It is independent of the programming language.
- It can be used to compare different projects even if they use different technologies (database, language, etc.).

Disadvantages:

- It is not good for real-time systems and embedded systems.
- Many cost estimation models like COCOMO uses LOC and hence FPC must be converted to LOC.

Problem-based estimation:

Lines of code and function points were described as measures from which productivity metrics can be computed. LOC and FP data are used in two ways during software project estimation: (1) as an estimation variable to "size" each element of the software and (2) as baseline metrics collected from past projects and used in conjunction with estimation variables to develop cost and effort projections.

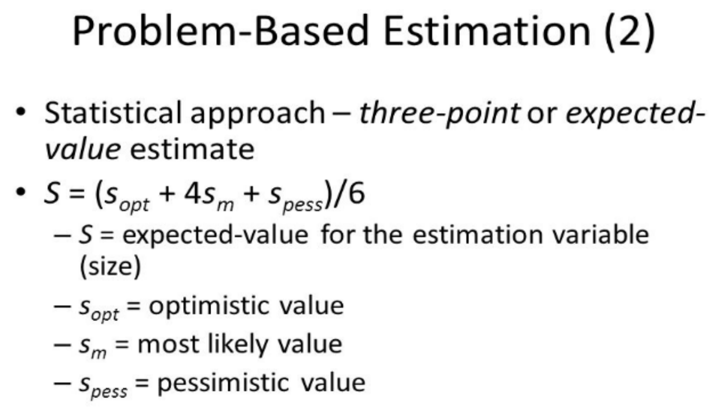
LOC and FP estimation are distinct estimation techniques. Yet both have a number of characteristics in common. The project planner begins with a bounded statement of software scope and from this statement attempts to decompose software into problem functions that can each be estimated individually. LOC or FP (the estimation variable) is then estimated for each function. Alternatively, the planner may choose another component for sizing such as classes or objects, changes, or business processes affected.

Baseline productivity metrics (e.g., LOC/pm or FP/pm) are then applied to the appropriate estimation variable, and cost or effort for the function is derived. Function estimates are combined to produce an overall estimate for the entire project.

CODECHAMP
CREATED WITH ARROK

A three-point or expected value can then be computed. The expected value for the estimation variable (size), S, can be computed as a weighted average of the optimistic (sopt), most likely (sm), and pessimistic (spess) estimates.

For example:



Problem-Based Estimation (2)

- Statistical approach – *three-point* or *expected-value* estimate
- $S = (s_{opt} + 4s_m + s_{pess})/6$
  - S = expected-value for the estimation variable (size)
  - s_{opt} = optimistic value
  - s_m = most likely value
  - s_{pess} = pessimistic value

Process-based estimation:

The most common technique for estimating a project is to base the estimate on the process that will be used. That is, the process is decomposed into a relatively small set of tasks and the effort required to accomplish each task is estimated.

Like the problem-based techniques, process-based estimation begins with a delineation of software functions obtained from the project scope. A series of software process activities must be performed for each function. Functions and related software process activities may be represented as part of a table.

Once problem functions and process activities are melded, the planner estimates the effort (e.g., person-months) that will be required to accomplish each software process activity for each software function. These data constitute the central matrix of the table. Average labor rates (i.e., cost/unit effort) are then applied to the effort estimated for each process activity. It is very likely the labor rate will vary for each task. Senior staff heavily involved in early activities are generally more expensive than junior staff involved in later design tasks, code generation, and early testing.

Costs and effort for each function and software process activity are computed as the last step. If process-based estimation is performed independently of LOC or FP estimation, we now have two or three estimates for cost and effort that may be compared and reconciled. If both sets of estimates show reasonable agreement, there is good reason to believe that the estimates are reliable. If, on the other hand, the results of these decomposition techniques show little agreement, further investigation and analysis must be conducted.

CODECHAMP
CREATED WITH ARBOK

4 Project Cost Estimation Techniques:

1. Analogous Estimating

Through analogous estimating, a project manager calculates the expected costs of a project-based upon the known costs associated with a similar project that was completed in the past. This method of estimation relies upon a combination of historical data and expert judgment of the project manager.

Because no two projects are exactly the same, analogous estimating does have its limitations. As such, it is often leveraged in the earliest stages of project planning, when a rough estimate can suffice. Analogous estimating can also be used when there is relatively little information about the current project available.

2. Parametric Estimating

In parametric estimating, historical data and statistical modeling are used to assign a dollar value to certain project costs. This approach determines the underlying unit cost for a particular component of a project and then sales that unit cost as appropriate. It is much more accurate than analogous estimating but requires more initial data to accurately assess costs.

Parametric estimating is often used in construction. For example, an experienced construction manager might understand that the typical new home will cost a certain number of dollars per square foot (assuming a particular margin of error). If this average cost, the margin of error, and the square footage of a new project are known, then parametric estimating will allow them to identify a budget that should accurately fall within this range. Other examples might include estimating the cost per unit to print and bind a book or to build an electronic device.

3. Bottom-Up Estimating

In bottom-up estimating, a larger project is broken down into a number of smaller components. The project manager then estimates costs specifically for each of these smaller work packages. For example, if a project includes work that will be split between multiple departments within an organization, costs might be split out by department. Once all costs have been estimated, they are tallied into a single larger cost estimate for the project as a whole.

Because bottom-up estimating allows a project manager to take a more granular look at individual tasks within a project, this technique allows for a very accurate estimation process.

4. Three-Point Estimating

CODECHAMP
CREATED WITH ARBOK

In three-point estimating, a project manager identifies three separate estimates for the costs associated with a project. The first point represents an “optimistic” estimate, where work is done and funds spent most efficiently; the second point represents the “pessimistic” estimate, where work is done and funds spent in the least efficient manner; and the third point represents the “most likely” scenario, which typically falls somewhere in the middle.

Three-point estimating relies on a number of weighted formulas and originates from the Program Analysis and Review Technique (PERT).

Software Cost Estimation:

For any new software project, it is necessary to know how much it will cost to develop and how much development time will it take. These estimates are needed before development is initiated, but how is this done? Several estimation procedures have been developed and are having the following attributes in common.

- Project scope must be established in advanced.
- Software metrics are used as a support from which evaluation is made.
- The project is broken into small PCs which are estimated individually.
  To achieve true cost & schedule estimate, several option arise.
- Delay estimation
- Used symbol decomposition techniques to generate project cost and schedule estimates.
- Acquire one or more automated estimation tools.

Uses of Cost Estimation:

During the planning stage, one needs to choose how many engineers are required for the project and to develop a schedule.

In monitoring the project's progress, one needs to access whether the project is progressing according to the procedure and takes corrective action, if necessary.

COCOMO MODEL:

Boehm proposed COCOMO (Constructive Cost Estimation Model) in 1981.COCOMO is one of the most generally used software estimation models in the world. COCOMO predicts the efforts and schedule of a software product based on the size of the software.

The necessary steps in this model are:

1. Get an initial estimate of the development effort from evaluation of thousands of delivered lines of source code (KDLOC).
2. Determine a set of 15 multiplying factors from various attributes of the project.
3. Calculate the effort estimate by multiplying the initial estimate with all the multiplying factors i.e., multiply the values in step1 and step2.

CODECHAMP CREATED WITH ARBOK

The initial estimate (also called nominal estimate) is determined by an equation of the form used in the static single variable models, using KDLOC as the measure of the size. To determine the initial effort Ei in person-months the equation used is of the type is shown below

Ei=a*(KDLOC)b

The value of the constant a and b are depending on the project type.

In COCOMO, projects are categorized into three types:

1. Organic
2. Semidetached
3. Embedded

1.Organic: A development project can be treated of the organic type, if the project deals with developing a well-understood application program, the size of the development team is reasonably small, and the team members are experienced in developing similar methods of projects. Examples of this type of projects are simple business systems, simple inventory management systems, and data processing systems.

2. Semidetached: A development project can be treated with semidetached type if the development consists of a mixture of experienced and inexperienced staff. Team members may have finite experience in related systems but may be unfamiliar with some aspects of the order being developed. Example of Semidetached system includes developing a new operating system (OS), a Database Management System (DBMS), and complex inventory management system.

3. Embedded: A development project is treated to be of an embedded type, if the software being developed is strongly coupled to complex hardware, or if the stringent regulations on the operational method exist. For Example: ATM, Air Traffic control.

For three product categories, Bohem provides a different set of expression to predict effort (in a unit of person month) and development time from the size of estimation in KLOC (Kilo Line of code) efforts estimation takes into account the productivity loss due to holidays, weekly off, coffee breaks, etc.

According to Boehm, software cost estimation should be done through three stages:

1. Basic Model
2. Intermediate Model
3. Detailed Model

1. Basic COCOMO Model:

The basic COCOMO model provide an accurate size of the project parameters. The following expressions give the basic COCOMO estimation model:

Effort=a1*(KLOC) a2 PM
Tdev=b1*(efforts)b2 Months

CODECHAMP
CREATED WITH ARBOK

Where

KLOC is the estimated size of the software product indicate in Kilo Lines of Code,

a1,a2,b1,b2 are constants for each group of software products,

Tdev is the estimated time to develop the software, expressed in months,

Effort is the total effort required to develop the software product, expressed in person months (PMs).

2. Intermediate Model:

The basic Cocomo model considers that the effort is only a function of the number of lines of code and some constants calculated according to the various software systems. The intermediate COCOMO model recognizes these facts and refines the initial estimates obtained through the basic COCOMO model by using a set of 15 cost drivers based on various attributes of software engineering.

Classification of Cost Drivers and their attributes:

(i) Product attributes -

- Required software reliability extent
- Size of the application database
- The complexity of the product

Hardware attributes -

- Run-time performance constraints
- Memory constraints
- The volatility of the virtual machine environment
- Required turnabout time

Personnel attributes -

- Analyst capability
- Software engineering capability
- Applications experience
- Virtual machine experience
- Programming language experience

Project attributes -

- Use of software tools
- Application of software engineering methods
- Required development schedule

CODECHAMP
CREATED WITH ARBOK

3. Detailed COCOMO Model:

Detailed COCOMO incorporates all qualities of the standard version with an assessment of the cost drivers effect on each method of the software engineering process. The detailed model uses various effort multipliers for each cost driver property. In detailed cocomo, the whole software is differentiated into multiple modules, and then we apply COCOMO in various modules to estimate effort and then sum the effort.

The Six phases of detailed COCOMO are:

1. Planning and requirements
2. System structure
3. Complete structure
4. Module code and test
5. Integration and test
6. Cost Constructive model

CODECHAMP
CREATED WITH ARBOK