Thakur Educational Trust's (Regd.)

THAKUR RAMNARAYAN COLLEGE OF ARTS & COMMERCE

ISO 21001:2018 Certified

PROGRAMME: B.Sc (I.T)

CLASS: S.Y.B.Sc (I.T)

SUBJECT NAME: SOFTWARE ENGINEERING

SEMESTER: IV

FACULTY NAME: Ms. SMRITI DUBEY

UNIT IV

Chapter 4 – Software Cost Estimation

Concepts:

Introduction

Software Productivity

Cost Estimation Technique

Algorithmic Cost Modeling

COCOMO (Cost Constructive Model)

COCOMO II Model

Introduction

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?

Every project quality depends upon the estimated effort and time with the project activities. Estimation involves answering the following questions:

1. How much effort is required to complete each activity?

2. How much calendar time is needed to complete each activity?

3. What is the total cost of each activity?

Project cost estimation and project scheduling is normally carried out together. The cost of development is primarily the cost of the efforts involved, so the effort computation is used in both the cost and the schedule estimate.

There are three parameters involved in computing the total cost of a software development project:

1. Hardware and software cost including maintenance

2. Travel and training cost

3. Effort costs

Software Productivity

You can measure productivity in a manufacturing system by counting the number of units that are produced and dividing this by the number of person-hours required to produce them. However, for any software problem, there are many different solutions, each of which has different attributes.

Productivity estimates are usually based on measuring attributes of the software and dividing this by the total effort required for development. **There are two types of metrics that have been used:**

1. **Size-related metrics -** These are related to the size of some output from an activity. The most commonly used size-related metric is lines of delivered source code. Other metrics that may be used are the number of delivered object code instructions or the number of pages of system documentation.

2. **Function-related metrics -** These are related to the overall functionality of the delivered software. Productivity is expressed in terms of the amount of useful functionality produced in some given time. Function points and application points are the best-known metrics of this type.

Lines of source code per programmer-month (LOC/pm or SLOC/pm) is a widely used software productivity metric. You can compute LOC/pm by counting the total number of lines of source code that are delivered, then divide the count by the total time in programmer-months required to

complete the project. This time therefore includes the time required for all other activities (requirements, design, coding, testing and documentation) involved in software development. However, the number of lines of source code depends on the programming language used and this can sometimes result in productivity anomalies.

Cost Estimation Techniques

Project schedule estimation is difficult. Initial estimates on the basis of a high-level user requirements definition. The software may have to run on unfamiliar computers or use new development technology. The people involved in the project and their skills will probably not be known.

Project estimates are often self-fulfilling. The estimate is used to define the project budget and the product is adjusted so that the budget figure is realized. A project that is within budget may have achieved this at the expense of features in the software being developed.

There are various types of technique that can be used to do this:

1. Experience-based techniques- The estimate of future effort requirements is based on the manager's experience of past projects and the application domain. Essentially, the manager makes an informed judgment of what the effort requirements are likely to be.

2. Algorithmic cost modeling - In this approach, a formulaic approach is used to compute the project effort based on estimates of product attributes, such as size, and process characteristics, such as experience of staff involved.

3. Analogous Estimating Method - Analogous cost estimating uses the values such as scope, cost, budget, and duration or measures of scale such as size, weight, and complexity from a previous, similar project as the basis for estimating the same parameter or measurement for a current project. When estimating costs, this technique relies on the actual cost of previous, similar projects as the basis for estimating the cost of the current project.

4. Parkinson's Law – Parkinson's Law states that work expands to fill the time available. The cost is determined by available resources rather than by objective assessment. If product has to be delivered in 12 months and 5 people are available then the effort required to estimated to be 60 person-months. No overspend takes place.

5. Pricing to win – The software cost is estimated to be whatever the customer has available to spend on the project. The estimated effort depends on the customer's budget and not on the

software functionality. You get the contract but the probability that the customer gets the system he or she wants is small. Costs do not accurately reflect the work required.

Algorithmic Cost Modeling

Algorithmic cost modelling uses a mathematical formula to predict project costs based on estimates of the project size, the number of software engineers, and other process and product factors. An algorithmic cost model can be built by analyzing the costs and attributes of completed projects and finding the closest fit formula to actual experience.

Algorithmic cost models are primarily used to make estimates of software development costs, but Boehm (Boehm, et al., 2000) discusses a range of other uses for algorithmic cost estimates, including estimates for investors in software companies, estimates of alternative strategies to help assess risks, and estimates to inform decisions about reuse, redevelopment or outsourcing.

In its most general form, an algorithmic cost estimate for software cost can be expressed as:

$$\text{Effort} = A\ \text{Size}^B\ M$$

A is a constant factor that depends on local organizational practices and the type of software that is developed. **Size** may be either an assessment of the code size of the software or a functionality estimate expressed in function or object points. The value of **exponent B** usually **lies between 1 and 1.5**. **M** is a multiplier made by combining process, product and development attributes, such as the dependability requirements for the software and the experience of the development team.

If you use an algorithmic cost estimation model, you should develop a range of estimates (worst, expected and best) rather than a single estimate and apply the costing formula to all of them. Estimates are most likely to be accurate when you understand the type of software that is being developed, when you have calibrated the costing model using local data, and when programming language and hardware choices are predefined.

COCOMO (Cost Constructive Model)

A number of algorithmic models have been proposed as the basis for estimating the effort, schedule and costs of a software project. **The COCOMO model is an empirical model that was derived by collecting data from a large number of software projects.** These data were analyzed to discover formulae that were the best fit to the observations. These formulae link the size of the system and product, project and team factors to the effort to develop the system.

It is a procedural cost estimate model for software projects and is often used as a process of reliably predicting the various parameters associated with making a project such as size, effort, cost, time, and quality. It was proposed by Barry Boehm in 1981 and is based on the study of 63 projects, which makes it one of the best-documented models.

The key parameters which define the quality of any software products, which are also an outcome of the COCOMO are primarily Effort & Schedule:

1. Effort: Amount of labor that will be required to complete a task. It is measured in person-months units.

2. Schedule: Simply means the amount of time required for the completion of the job, which is, of course, proportional to the effort put in. It is measured in the units of time such as weeks, and months.

The first version of the COCOMO model (COCOMO 81) was a **three-level model** where the levels corresponded to the detail of the analysis of the cost estimate. **The first level (basic)** provided an initial rough estimate; **the second level** modified this using a number of project and process multipliers; and the **most detailed level** produced estimates for different phases of the project. Figure shows the basic COCOMO formula for different types of projects. The multiplier M reflects product, project and team characteristics.

Project complexity	Formula	Description
Simple	$PM = 2.4\ (KDSI)^{1.05} \times M$	Well-understood applications developed by small teams
Moderate	$PM = 3.0\ (KDSI)^{1.12} \times M$	More complex projects where team members may have limited experience of related systems
Embedded	$PM = 3.6\ (KDSI)^{1.20} \times M$	Complex projects where the software is part of a strongly coupled complex of hardware, software, regulations and operational procedures

The basic COCOMO 81 model

COCOMO applies to three classes of software projects:

1. Organic – A software project is said to be an organic type if the team size required is adequately small, the problem is well understood and has been solved in the past and also the team members have a nominal experience regarding the problem. **Examples of this type of**

projects are simple business systems, simple inventory management systems, and data processing systems.

2. Semi-detached – A software project is said to be a Semi-detached type if the vital characteristics such as team size, experience, and knowledge of the various programming environment lie in between that of organic and embedded. **Example of Semidetached system includes developing a new operating system (OS), a Database Management System (DBMS), and complex inventory management system.**

3. Embedded – A software project requiring the highest level of complexity, creativity, and experience requirement fall under this category. Such software requires a larger team size than the other two models and also the developers need to be sufficiently experienced and creative to develop such complex models. **For Example: ATM, Air Traffic control.**

It is important to note that the effort and the duration estimations obtained using the COCOMO model are called a nominal effort estimate and nominal duration estimate. The term nominal implies that if anyone tries to complete the project in a time shorter than the estimated duration, then the cost will increase drastically. But, if anyone completes the project over a longer period of time than the estimated, then there is almost no decrease in the estimated cost value.

COCOMO II Model

COCOMO-II is the revised version of the original COCOMO (Constructive Cost Model) and is developed at University of Southern California. It is the model that allows one to estimate the cost, effort and schedule when planning a new software development activity.

The COCOMO II model recognizes different approaches to software development such as prototyping, development by component composition and use of database programming. COCOMO II supports a spiral model of development and embeds several sub-models that produce increasingly detailed estimates. These can be used in successive rounds of the development spiral. Figure shows COCOMO II sub-models and where they are used.

The sub-models that are part of the COCOMO II model are:

1. An application-composition model - This assumes that systems are created from reusable components, scripting or database programming. It is designed to make estimates of prototype development. Software size estimates are based on application points, and a simple size/productivity formula is used to estimate the effort required. Application points are the same

as object points but the name was changed to avoid confusion with objects in object-oriented development.

2. An early design model - This model is used during early stages of the system design after the requirements have been established. Estimates are based on function points, which are then converted to number of lines of source code. The formula follows the standard form discussed above with a simplified set of seven multipliers.

3. A reuse model - This model is used to compute the effort required to integrate reusable components and/or program code that is automatically generated by design or program translation tools. It is usually used in conjunction with the post-architecture model.

4. A post-architecture model - Once the system architecture has been designed, a more accurate estimate of the software size can be made. Again, this model uses the standard formula for cost estimation discussed above. However, it includes a more extensive set of 17 multipliers reflecting personnel capability and product and project characteristics.