K.D. K. College of Engineering

Department of Information Technology

B.E. (Information Technology) Fifth Semester (C.B.S.)

Software Engineering

UNIT: 02

Measures Metrics and Indicator

Metrics is often used interchangeably with measure and measurement. However, it is important to note the differences between them. Measure can be defined as quantitative indication of amount, dimension, capacity, or size of product and process attributes. Measurement can be defined as the process of determining the measure. Metrics can be defined as quantitative measures that allow software engineers to identify the efficiency and improve the quality of software process, project, and product.

To understand the difference, let us consider an example. A measure is established when a number of errors is (single data point) detected in a software component. Measurement is the process of collecting one or more data points. In other words, measurement is established when many components are reviewed and tested individually to collect the measure of a number of errors in all these components. Metrics are associated with individual measure in some manner. That is, metrics are related to detection of errors found per review or the average number of errors found per unit test.

Once measures and metrics have been developed, indicators are obtained. These indicators provide a detailed insight into the software process, software project, or intermediate product. Indicators also enable software engineers or project managers to adjust software processes and improve software products, if required. For example, measurement dashboards or key indicators are used to monitor progress and initiate change. Arranged together, indicators provide snapshots of the system's performance.

Metrics represent the different methods we employ to understand change over time across a number of dimensions or criteria. It is often used as a catch-all term to describe the method used to measure something, the resulting values obtained from measuring, as well as a calculated or combined set of measures.
We use the term **measures** when we mean the value measured by whatever mechanism we employ and the term **indicator** for values we combine and use to hint to specific outcomes and trends.
Note: All measures and indicators reflect events that occured in a specific period of time. When representing these values (specially when aggregating them), indicate the period of time it relates to.

Measures

A measure is a number or a quantity that records a directly observable value or performance. All measures are composed of a value (a number) and a unit of measure. The

number provides magnitude for the measure (how much), while the unit gives number meaning (what is measured).

- 1,234,567 Pageviews
- 8,901,234 Sessions
- 567,890 Facebook Likes

An indicator is a qualitative or quantitative factor or variable that provides a simple and reliable mean to express achievement, the attainment of a goal, or the results stemming from a specific change. It often aggregates or combines multiple measures in an explicit formula.

- 1M weekly active users
- 1:3 users complete the story
- 23% homepage bounce rate

Metrics for process & projects

1. Process Metrics

   These are metrics that pertain to Process Quality. They are used to measure the efficiency and effectiveness of various processes.

2. Project Metrics

   These are metrics that relate to Project Quality. They are used to quantify defects, cost, schedule, productivity and estimation of various project resources and deliverables.

Project Metrics

1. Schedule Variance: Any difference between the scheduled completion of an activity and the actual completion is known as Schedule Variance.

   Schedule variance = ((Actual calendar days – Planned calendar days) + Start variance)/ Planned calendar days x 100.

2. Effort Variance: Difference between the planned outlined effort and the effort required to actually undertake the task is called Effort variance.

   Effort variance = (Actual Effort – Planned Effort)/ Planned Effort x 100.

3. Size Variance: Difference between the estimated size of the project and the actual size of the project (normally in KLOC or FP).

   Size variance = (Actual size – Estimated size)/ Estimated size x 100.

4. Requirement Stability Index: Provides visibility to the magnitude and impact of requirements changes.

RSI = 1- ((Number of changed + Number of deleted + Number of added) / Total number of initial requirements) x100.

5. Productivity (Project): Is a measure of output from a related process for a unit of input.

   Project Productivity = Actual Project Size / Actual effort expended in the project.

6. Productivity (for test case preparation) = Actual number of test cases/ Actual effort expended in test case preparation.

7. Productivity (for test case execution) = Actual number of test cases / actual effort expended in testing.

8. Productivity (defect detection) = Actual number of defects (review + testing) / actual effort spent on (review + testing).

9. Productivity (defect fixation) = actual no of defects fixed/ actual effort spent on defect fixation.

10. Schedule variance for a phase: The deviation between planned and actual schedules for the phases within a project.

    Schedule variance for a phase = (Actual Calendar days for a phase – Planned calendar days for a phase + Start variance for a phase)/ (Planned calendar days for a phase) x 100

11. Effort variance for a phase: The deviation between a planned and actual effort for various phases within the project.

    Effort variance for a phase = (Actual effort for a phase – a planned effort for a phase)/ (planned effort for a phase) x 100.

Process Metrics:

1. Cost of quality: It is a measure of the performance of quality initiatives in an organization. It's expressed in monetary terms.

   Cost of quality = (review + testing + verification review + verification testing + QA + configuration management + measurement + training + rework review + rework testing)/ total effort x 100.

2. Cost of poor quality: It is the cost of implementing imperfect processes and products.

   Cost of poor quality = rework effort/ total effort x 100.

3. Defect density: It is the number of defects detected in the software during development divided by the size of the software (typically in KLOC or FP)

   Defect density for a project = Total number of defects/ project size in KLOC or FP

4. Review efficiency: defined as the efficiency in harnessing/ detecting review defects in the verification stage.

   Review efficiency = (number of defects caught in review)/ total number of defects caught) x 100.

5. Testing Efficiency: Testing efficiency = 1 – ((defects found in acceptance)/ total number of testing defects) x 100.

6. Defect removal efficiency: Quantifies the efficiency with which defects were detected and prevented from reaching the customer.

   Defect removal efficiency = (1 – (total defects caught by customer/ total number of defects)) x 100.

7. Residual defect density = (total number of defects found by a customer)/ (Total number of defects including customer found defects) x 100.

Software measurement

To assess the quality of the engineered product or system and to better understand the models that are created, some measures are used. These measures are collected throughout the software development life cycle with an intention to improve the software process on a continuous basis. Measurement helps in estimation, quality control, productivity assessment and project control throughout a software project. Also, measurement is used by software engineers to gain insight into the design and development of the work products. In addition, measurement assists in strategic decision-making as a project proceeds.

Software measurements are of two categories, namely, direct measures and indirect measures. Direct measures include software processes like cost and effort applied and products like lines of code produced, execution speed, and other defects that have been reported. Indirect measures include products like functionality, quality, complexity, reliability, maintainability, and many more.

Generally, software measurement is considered as a management tool which if conducted in an effective manner, helps the project manager and the entire software team to take decisions that lead to successful completion of the project. Measurement process is characterized by a set of five activities, which are listed below.

- **Formulation:** This performs measurement and develops appropriate metric for software under consideration.
- **Collection:** This collects data to derive the formulated metrics.
- **Analysis:** This calculates metrics and the use of mathematical tools.
- **Interpretation:** This analyzes the metrics to attain insight into the quality of representation.

- **Feedback:** This communicates recommendation derived from product metrics to the software team.

Note that collection and analysis activities drive the measurement process. In order to perform these activities effectively, it is recommended to automate data collection and analysis, establish guidelines and recommendations for each metric, and use statistical techniques to interrelate external quality features and internal product attributes.

Metrics for software quality

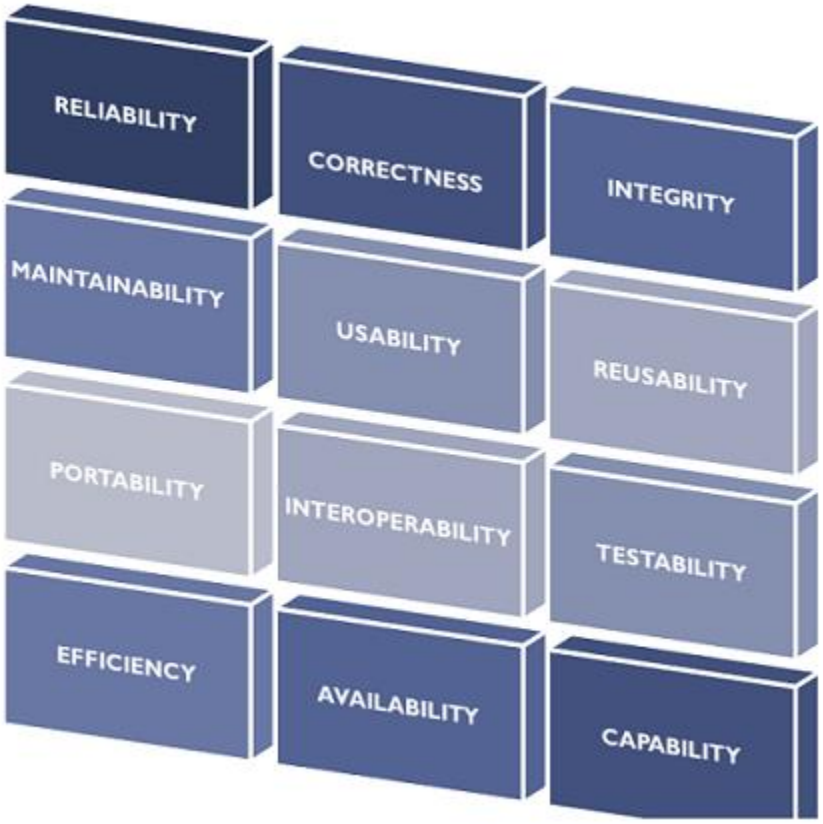
The word '**metrics**' refer to standards for measurements. Software Quality Metrics means measurement of attributes, pertaining to software quality along with its process of development.

The term **"software quality metrics"** illustrate the picture of measuring the software qualities by recording the number of defects or security loopholes present in the software. However, quality measurement is not restricted to counting of defects or vulnerabilities but also covers other aspects of the qualities such as maintainability, reliability, integrity, usability, customer satisfaction, etc.

Why Software Quality Metrics?

1. To define and categorize elements in order to have better understanding of each and every process and attribute.
2. To evaluate and assess each of these process and attribute against the given requirements and specifications.
3. Predicting and planning the next move w.r.t software and business requirements.
4. Improving the Overall quality of the process and product, and subsequently of project.

What are the quality factors that define & impact the user experience?



Software Quality Metrics: sub-category of Software Metrics

It is basically, a subclass of **software metrics** that mainly emphasizes on quality assets of the software product, process and project. Software metric is a broader concept that incorporates software quality metrics in it, and mainly consists of three types of metrics:

- **Product Metrics:**

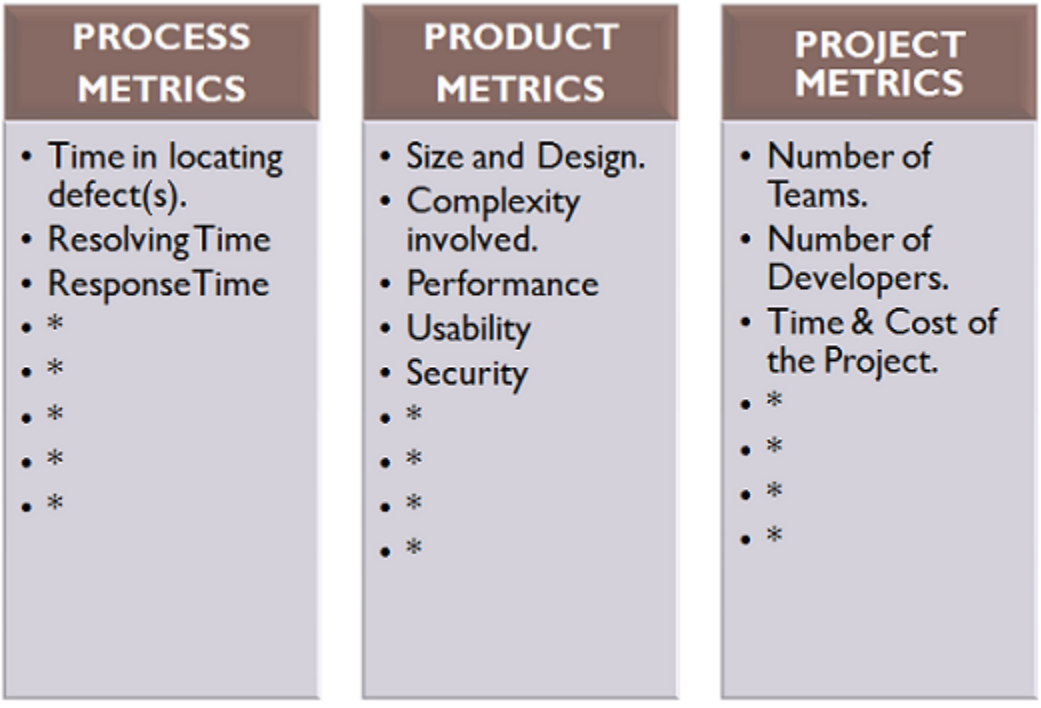
  It includes size, design, complexity, performance and other parameters that are associated with the product's quality.

- **Process Metrics:**

  It involves parameters like, time-duration in locating and removing defects, response time for resolving issues, etc.

- **Project Metrics:**

  It may include number of teams, developers involved, cost and duration for the project, etc.



Methodology Of Software Quality Metrics

The methodology behind software quality metric is as under:

- Identify and prepare the list of possible requirements of quality, and subsequently, assigning direct metric, such as understanding, learning and operation time, to each of these requirements.
- Apply metrics framework, along with the cost-benefit analysis.
- Implementing metrics via collecting and defining data to compute metric values.
- Interpret and analyse the results, to ensure the fulfilment of requirements.
- Validate the metrics through validation methodology and thereafter proper documentation of the results.

Features of good Software Quality Metrics

- Should be specific to measure the particular attribute or an attribute of greater importance.
- Comprehensive for wide variety of scenarios.
- Should not consider attributes that have already been measured by some other metric.
- Reliable to work similarly in all conditions.
- Should be easy and simple to understand and operate.

Metrics for small organization

 Most software organizations have fewer than 20 software engineers.
 Best advice is to choose simple metrics that provide value to the organization and don't require a lot of effort to collect.
 Even small groups can expect a significant return on the investment required to collect metrics, if this activity leads to process improvement

Introduction
The main motivation for this investigation is describing a practical guideline for planning, implementing, and using a useful metrics program in a very small company.

The basis for the metrics program as practiced today is the Quality Circle.
Metrics data about product and process. We focus on process.

What is a very small Company ?

Metrics data about product and process. We focus on process. * 3 - 30 Software Engineers * Software development ad-hoc * Don't have any certifications (SPICE, CMM, ISO)

Metrics data

The process database invariably captured information about project size, effort, schedule, defects, and risks. Lines of code (LOC), function points, components. Effort Actual and estimates dates. Schedule Total number of defects, distribution of defects with respect to where was detected and captured, origin of defects, and removed defect. Person-hours, person-months, hours in a project. Defects Risk mitigation strategies, number of change requests, number of baseline, number of risks, number of reviews, and so on.

Use of Metrics data

1) Project planning 2) Monitoring and controlling a project 3) Overall process management and improvement

Project Planning

Effort estimation, schedule, quantitative goals for quality, control activities and defect prevention, and risk management.

Project monitoring and control Current state of the project, milestone analysis, event-driven analysis.

Process analysis and improvement

Process capability, organization goal in terms of improvement in quality and productivity (reducing the variance in performance, reducing the defect injection rate or improving the review effectiveness).

A first metrics program (Example 1)

Step 1: Document the Software Development Process

Step 2: State the Goals

Step 3: Define Metrics Required to Reach Goals

Step 4: Identify Data to Collect

Step 5: Define Data Collection Procedures

Step 6: Assemble a Metrics Toolset

Step 7: Create a Metrics Database

Step 8: Define the Feedback Mechanism

Software scope and Feasibility

Software scope is a **well-defined boundary**, which encompasses all the activities that are done to develop and deliver the software product.

The software scope clearly defines all functionalities and artifacts to be delivered as a part of the software. The scope identifies what the product will do and what it will not do, what the end product will contain and what it will not contain.

Scope definition is the process of analyzing, prioritizing, agreeing on and documenting the scope of the project while managing a consistent stream of communication to the stakeholders. While this usually is a continuous process throughout a project, a definite foundation on which the scope is built on will define the project (product or service) outcome.

Software scope

Scope definition is the process of analyzing, prioritizing, agreeing on and documenting the scope of the project while managing a consistent stream of communication to the stakeholders. While this usually is a continuous process throughout a project, a definite foundation on which the scope is built on will define the project (product or service) outcome.

This process draws out the functionality the business is looking to get and also identifies any constraints and potential risks avoiding surprises at a later time.

The process touches all the teams and departments that would be involved creating a collaborative and transparent environment and reducing friction.

Whatever methodology your organization uses – Agile (Scrum), Kanban, etc or a traditional Waterfall model, our engagements start off with a thorough analysis of the business problem.

We use a domain specific approach to define the scope of the project. We have domain expertise in several areas of

- Technology
- Capital markets
- Healthcare
- Insurance

Translation of the business problem into a technical specification is a multi-step process. The process involves collecting, documenting and disseminating information of the following items:

- *Identify all the stakeholders who will need to be involved in the process*
    - Is there a buy-in from all the stakeholders?
    - What if any are the concerns of any of the stakeholders?
- *Determine an appropriate high level software based approach to the problem*
    - How is the problem approached today?
    - What is the most time-consuming part of the process?
    - Are there opportunities to automate the process?
- *Determine the integration points with other systems*
    - Is there a documented API available?
    - What kind of security protocols are available to ensure an easy an secure connection?
    - What is the structure of the data available?
- *Identify the audience*
    - What kind of devices does the solution have to work with?
    - How many different roles need to be defined and what responsibilities do these roles have?
    - Is the user-base internal/external/mixed?
    - What kind of load is expected on the system on day 1 -> year 1 -> year 3
- *Define the core software functionality?*
    - Define the technology stack
    - Define the data model
    - Define the UI flows
    - Define the build and deployment strategy

Feasibility

A feasibility study is carried out to select the best system that meets performance requirements. The main aim of the feasibility study activity is to determine whether it would be financially and technically feasible to develop the product. The feasibility study activity involves the analysis of the problem and collection of all relevant information relating to the product such as the different

data items which would be input to the system, the processing required to be carried out on these data, the output data required to be produced by the system as well as various constraints on the behaviour of the system.

Technical Feasibility

This is concerned with specifying equipment and software that will successfully satisfy the user requirement. The technical needs of the system may vary considerably, but might include :

- The facility to produce outputs in a given time.
- Response time under certain conditions.
- Ability to process a certain volume of transaction at a particular speed.
- Facility to communicate data to distant locations.

In examining technical feasibility, configuration of the system is given more importance than the actual make of hardware. The configuration should give the complete picture about the system's requirements:

How many workstations are required, how these units are interconnected so that they could operate and communicate smoothly.

What speeds of input and output should be achieved at particular quality of printing.

Economic Feasibility

Economic analysis is the most frequently used technique for evaluating the effectiveness of a proposed system. More commonly known as Cost / Benefit analysis, the procedure is to determine the benefits and savings that are expected from a proposed system and compare them with costs. If benefits outweigh costs, a decision is taken to design and implement the system. Otherwise, further justification or alternative in the proposed system will have to be made if it is to have a chance of being approved. This is an outgoing effort that improves in accuracy at each phase of the system life cycle.

Operational Feasibility

This is mainly related to human organizational and political aspects. The points to be considered are:

- What changes will be brought with the system?
- What organizational structure are disturbed?
- What new skills will be required? Do the existing staff members have these skills? If not, can they be trained in due course of time?

This feasibility study is carried out by a small group of people who are familiar with information system technique and are skilled in system analysis and design process.

Proposed projects are beneficial only if they can be turned into information system that will meet the operating requirements of the organization. This test of feasibility asks if the system will work when it is developed and installed.

Software project estimation

Effective software project estimation is one of the most challenging and important activities in software development. Proper project planning and control is not possible without a sound and

reliable estimate. As a whole, the software industry doesn't estimate projects well and doesn't use estimates appropriately. We suffer far more than we should as a result and we need to focus some effort on improving the situation. Under-estimating a project leads to under-staffing it (resulting in staff burnout), under-scoping the quality assurance effort (running the risk of low quality deliverables), and setting too short a schedule (resulting in loss of credibility as deadlines are missed). For those who figure on avoiding this situation by generously padding the estimate, over-estimating a project can be just about as bad for the organization! If you give a project more resources than it really needs without sufficient scope controls it will use them. The project is then likely to cost more than it should (a negative impact on the bottom line), take longer to deliver than necessary (resulting in lost opportunities), and delay the use of your resources on the next project. Software Project Estimation 101 The four basic steps in software project estimation are: 1) Estimate the size of the development product. This generally ends up in either Lines of Code (LOC) or Function Points (FP), but there are other possible units of measure. A discussion of the pros & cons of each is discussed in some of the material referenced at the end of this report. 2) Estimate the effort in person-months or person-hours. 3) Estimate the schedule in calendar months. 4) Estimate the project cost in dollars (or local currency) Estimating size An accurate estimate of the size of the software to be built is the first step to an effective estimate. Your source(s) of information regarding the scope of the project should, wherever possible, start with formal descriptions of the requirements - for example, a customer's requirements specification or request for proposal, a system specification, a software requirements specification. If you are [re-]estimating a project in later phases of the project's lifecycle, design documents can be used to provide additional detail. Don't let the lack of a formal scope specification stop you from doing an initial project estimate. A verbal description or a whiteboard outline are sometimes all you have to start with. In any case, you must communicate the level of risk and uncertainty in an estimate to all concerned and you must re-estimate the project as soon as more scope information is determined. Two main ways you can estimate product size are: 1) By analogy. Having done a similar project in the past and knowing its size, you estimate each major piece of the new project as a percentage of the size of a similar piece of

the previous project. Estimate the total size of the new project by adding up the estimated sizes of each of the pieces. An experienced estimator can produce reasonably good size estimates by analogy if accurate size values are available for the previous project and if the new project is sufficiently similar to the previous one. 2) By counting product features and using an algorithmic approach such as Function Points to convert the count into an estimate of size. Macro-level "product features" may include the number of subsystems, classes/modules, methods/functions. More detailed "product features" may include the number of screens, dialogs, files, database tables, reports, messages, and so on. Estimating effort Once you have an estimate of the size of your product, you can derive the effort estimate. This conversion from software size to total project effort can only be done if you have a defined software development lifecycle and development process that you follow to specify, design, develop, and test the software. A software development project involves far more than simply coding the software – in fact, coding is often the smallest part of the overall effort. Writing and reviewing documentation, implementing prototypes, designing the deliverables, and reviewing and testing the code take up the larger portion of overall project effort. The project effort estimate requires you to identify and estimate, and then sum up all the activities you must perform to build a product of the estimated size. There are two main ways to derive effort from size: 1) The best way is to use your organization's own historical data to determine how much effort previous projects of the estimated size have taken. This, of course, assumes (a) your organization has been documenting actual results from previous projects, (b) that you have at least one past project of similar size (it is even better if you have several projects of similar size as this reinforces that you consistently need a certain level of effort to develop projects of a given size), and (c) that you will follow a similar development lifecycle, use a similar development methodology, use similar tools, and use a team with similar skills and experience for the new project. 2) If you don't have historical data from your own organization because you haven't started collecting it yet or because your new project is very different in one or more key aspects, you can use a mature and generally accepted algorithmic approach such as Barry Boehm's COCOMO model or the Putnam Methodology to convert a size estimate into an effort estimate. These models have been derived by studying a

significant number of completed projects from various organizations to see how their project sizes mapped into total project effort. These “industry data” models may not be as accurate as your own historical data, but they can give you useful ballpark effort estimates. Estimating schedule The third step in estimating a software development project is to determine the project schedule from the effort estimate. This generally involves estimating the number of people who will work on the project, what they will work on (the Work Breakdown Structure), when they will start working on the project and when they will finish (this is the “staffing profile”). Once you have this information, you need to lay it out into a calendar schedule. Again, historical data from your organization’s past projects or industry data models can be used to predict the number of people you will need for a project of a given size and how work can be broken down into a schedule. If you have nothing else, a schedule estimation rule of thumb [McConnell 1996] can be used to get a rough idea of the total calendar time required: Schedule in months = 3.0 * (effort-months) 1/3 Opinions vary as to whether 2.0 or 2.5 or even 4.0 should be used in place of the 3.0 value – only by trying it out will you see what works for you. Estimating Cost There are many factors to consider when estimating the total cost of a project. These include labor, hardware and software purchases or rentals, travel for meeting or testing purposes, telecommunications (e.g., longdistance phone calls, video-conferences, dedicated lines for testing, etc.), training courses, office space, and so on. Exactly how you estimate total project cost will depend on how your organization allocates costs. Some costs may not be allocated to individual projects and may be taken care of by adding an overhead value to labor rates ($ per hour). Often, a software development project manager will only estimate the labor cost and identify any additional project costs not considered “overhead” by the organization. The simplest labor cost can be obtained by multiplying the project’s effort estimate (in hours) by a general labor rate ($ per hour). A more accurate labor cost would result from using a specific labor rate for each staff position (e.g., Technical, QA, Project Management, Documentation, Support, etc.). You would have to determine what percentage of total project effort should be allocated to each position. Again, historical data or industry data models can help

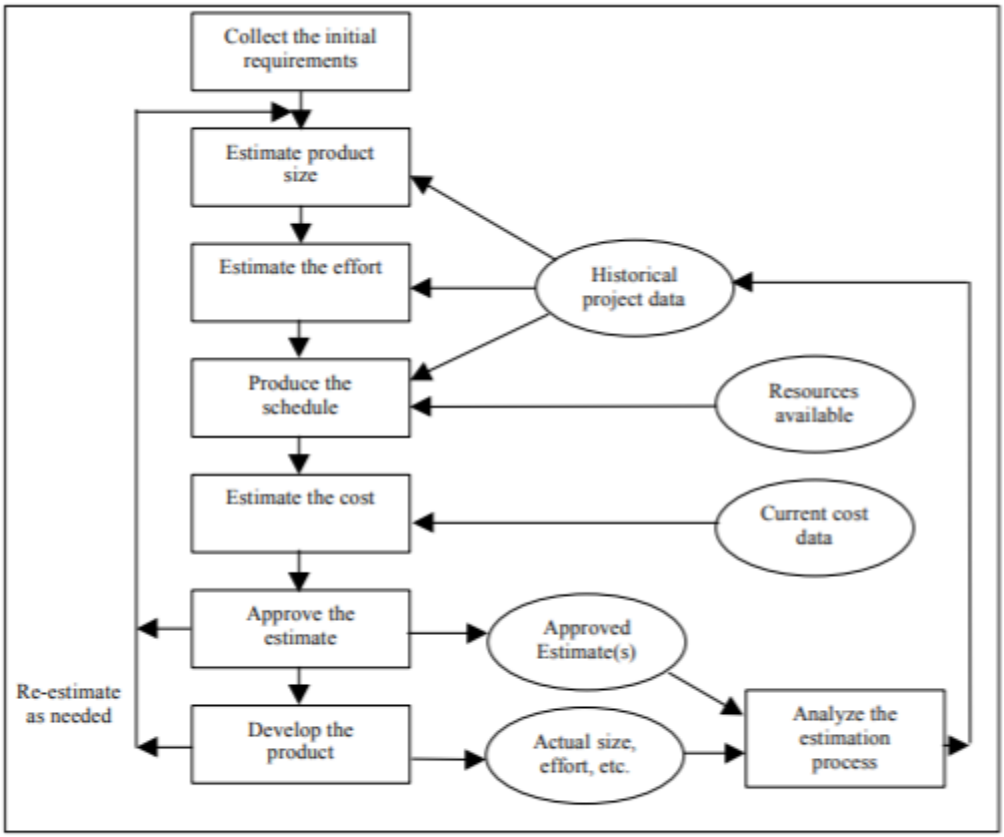


Figure 1 – The Basic Project Estimation Process

Decomposition Techniques

Decomposition Techniques

Software project estimation is a form of problem solving, and in most cases, the problem to be solved (i.e., developing a cost and effort estimate for a software project) is too complex to be

considered in one piece. For this reason, we decompose the problem, re-characterizing it as a set of smaller (and hopefully, more manageable) problems. Before an estimate can be made, the project planner must understand the scope of the software to be built and generate an estimate of its "size."

Software Sizing

The accuracy of a software project estimate is predicated on a number of things:

(1) The degree to which the planner has properly estimated the size of the product to be built

(2) The ability to translate the size estimate into human effort, calendar time, and dollars (a function of the availability of reliable software metrics from past projects)

(3) The degree to which the project plan reflects the abilities of the software team

(4) The stability of product requirements and the environment that supports the software engineering effort.

As project estimate is only as good as the estimate of the size of the work to be accomplished, sizing represents the project planner's first major challenge.

In the context of project planning, size refers to a quantifiable outcome of the software project. If a direct approach is taken, size can be measured in LOC. If an indirect approach is chosen, size is represented as FP There are four different approaches to the sizing problem:

1. "Fuzzy logic" sizing: This approach uses the approximate reasoning techniques that are the cornerstone of fuzzy logic. To apply this approach, the planner must identify the type of application, establish its magnitude on a qualitative scale, and then refine the magnitude within the original range. Although personal experience can be used, the planner should also have

2. Function point sizing: The planner develops estimates of the information domain. Its characteristics will be discussed later in the session.

3. Standard component sizing: Software is composed of a number of different "standard components" that are generic to a particular application area. For example, the standard components for an information system are subsystems, modules, screens, reports, interactive programs, batch programs, files, LOC, and object-level instructions. The project planner estimates the number of occurrences of each standard component and then uses historical project data to determine the delivered size per standard component. To illustrate, consider an information systems application. The planner estimates that 18 reports will

be generated. Historical data indicates that 967 lines of COBOL [PUT92) are required *per* report. This enables the planner to estimate that 17,000 LOC will be required for the reports component. Similar estimates and computation are made for other standard components, and a combined size value (adjusted statistically) results.

4. Change sizing: This approach is used when a project encompasses the use of existing software that must be modified in some way as part of a project. The planner estimates the number and type (e.g., reuse, adding code, changing code, and deleting code) of modifications that must be accomplished. Using an "effort ratio" [PUT92) for each type of change, the size of the change may be estimated

Empirical Estimation Models

COCOMO - An Empirical Estimation Model for Effort

Cocomo (Constructive Cost Model) is a regression model based on LOC, i.e **number of Lines of Code**. It is a procedural cost estimate model for software projects and 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 1970 and is based on the study of 63 projects, which make 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:

- **Effort:** Amount of labor that will be required to complete a task. It is measured in person-months units.
- **Schedule:** Simply means the amount of time required for the completion of the job, which is, of course, proportional to the effort put. It is measured in the units of time such as weeks, months.

Different models of Cocomo have been proposed to predict the cost estimation at different levels, based on the amount of accuracy and correctness required. All of these models can be applied to a variety of projects, whose characteristics determine the value of constant to be used in subsequent calculations. These characteristics pertaining to different system types are mentioned below.

Boehm’s definition of organic, semidetached, and embedded systems:

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.
2. **Semi-detached –** A software project is said to be a Semi-detached type if the vital characteristics such as team-size, experience, knowledge of the various programming environment lie in between that of organic and Embedded. The projects classified as Semi-Detached are comparatively less familiar and difficult to develop compared to the organic ones and require more experience and better guidance and creativity. Eg: Compilers or different Embedded Systems can be considered of Semi-Detached type.
3. **Embedded –** A software project with 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.

All the above system types utilize different values of the constants used in Effort Calculations.

Types of Models: COCOMO consists of a hierarchy of three increasingly detailed and accurate forms. Any of the three forms can be adopted according to our requirements. These are types of COCOMO model:

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

The first level, **Basic COCOMO** can be used for quick and slightly rough calculations of Software Costs. Its accuracy is somewhat restricted due to the absence of sufficient factor considerations.

Intermediate COCOMO takes these Cost Drivers into account and **Detailed COCOMO** additionally accounts for the influence of individual project phases, i.e in case of Detailed it accounts for both these cost drivers and also calculations are performed phase wise henceforth producing a more accurate result. These two models are further discussed below.

Estimation of Effort: Calculations –

4. **Basic Model –**

5.

The above formula is used for the cost estimation of for the basic COCOMO model, and also is used in the subsequent models. The constant values a and b for the Basic Model for the different categories of system:

SOFTWARE PROJECTS	A	B
Organic	2.4	1.05
Semi Detached	3.0	1.12
Embedded	3.6	1.20

The effort is measured in Person-Months and as evident from the formula is dependent on Kilo-Lines of code. These formulas are used as such in the Basic Model calculations, as not much consideration of different factors such as reliability, expertise is taken into account, henceforth the estimate is rough.

6. **Intermediate Model –**

The basic Cocomo model assumes that the effort is only a function of the number of lines of code and some constants evaluated according to the different software system. However, in reality, no system's effort and schedule can be solely calculated on the basis of Lines of Code. For that, various other factors such as reliability, experience, Capability. These factors are known as Cost Drivers and the Intermediate Model utilizes 15 such drivers for cost estimation.

Classification of Cost Drivers and their attributes:

(i) Product attributes –

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

(ii) Hardware attributes –

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

(iii) Personnel attributes –

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

(iv) Project attributes –

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

4. ;

COST DRIVERS	VERY LOW	LOW	NOMINAL	HIGH	VERY HIGH
Product Attributes					
Required Software Reliability	0.75	0.88	1.00	1.15	1.40
Size of Application Database		0.94	1.00	1.08	1.16
Complexity of The Product	0.70	0.85	1.00	1.15	1.30
Hardware Attributes					
Runtime Performance Constraints			1.00	1.11	1.30
Memory Constraints			1.00	1.06	1.21
Volatility of the virtual machine environment		0.87	1.00	1.15	1.30
Required turnabout time		0.94	1.00	1.07	1.15
Personnel attributes					
Analyst capability	1.46	1.19	1.00	0.86	0.71
Applications experience	1.29	1.13	1.00	0.91	0.82
Software engineer capability	1.42	1.17	1.00	0.86	0.70

COST DRIVERS	VERY LOW	LOW	NOMINAL	HIGH	VERY HIGH
Virtual machine experience	1.21	1.10	1.00	0.90	
Programming language experience	1.14	1.07	1.00	0.95	
Project Attributes					
Application of software engineering methods	1.24	1.10	1.00	0.91	0.82
Use of software tools	1.24	1.10	1.00	0.91	0.83
Required development schedule	1.23	1.08	1.00	1.04	1.10

5. The project manager is to rate these 15 different parameters for a particular project on a scale of one to three. Then, depending on these ratings, appropriate cost driver values are taken from the above table. These 15 values are then multiplied to calculate the EAF (Effort Adjustment Factor). The Intermediate COCOMO formula now takes the form:

6.

7. The values of a and b in case of the intermediate model are as follows:

SOFTWARE PROJECTS	A	B
Organic	3.2	1.05
Semi Detached	3.0	1.12
Embeddedc	2.8	1.20

8. **Detailed Model –**

Detailed COCOMO incorporates all characteristics of the intermediate version with an assessment of the cost driver's impact on each step of the software engineering process. The detailed model uses different effort multipliers for each cost driver attribute. In detailed cocomo, the whole software is divided into different modules and then we apply COCOMO in different modules to estimate effort and then sum the effort.

The Six phases of detailed COCOMO are:

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

The effort is calculated as a function of program size and a set of cost drivers are given according to each phase of the software lifecycle.

Make-buy Decision

The decision to Make or Buy in software acquisition is often made, in spite of our best efforts, subjectively. Many different factors play into the decision making process and it is the purpose of this study to find out what exactly these factors are. Some of these factors may include: Availability of ready-made solutions• Cost of ready-made solutions• Past success or failure in software acquisition• Perceived risk• Since it is the perceived effectiveness and efficiency of the decision, something that is inherently subjective, that is being investigated and the study is interpretive and descriptive, qualitative research methods are used. 2 The focus of this study, Make Vs Buy decisions, falls under the subject of software acquisition. Many of the principles of software acquisition have relevance within the context of this study. The Make Vs Buy decision is part of the software acquisition process. In focusing on only this particular step in the process, it is hoped that some new information may be found on software acquisition.

Make and Buy have specific meanings which may be unique to this study. In order to clarify the situation, an explanation of each term follows. A Make decision constitutes the creation of a new piece of software. This includes insourcing and outsourcing the development of said software. A Buy decision constitutes the purchase or acquisition of a pre-made piece of software. There is also a third, more ambiguous option available. This is Buying a piece of software which partially fits the needs of the company, and then Making it fit the needs more closely. As an example, a company could buy a piece of software and then significantly modify/have it modified. This option has facets of both the Make and Buy decision. For the purposes of this study, this situation will be avoided so as to provide a clearer focus for the study. This allows for future studies to comparatively analyse this third

option within the context of this study. The core problem that this study addresses is that the Make Vs Buy decision is difficult, and can be influenced by human subjectivity and bias. The Make Vs Buy decision, for the purposes of this study, is always a part of software acquisition process. Even if no consideration was given to the decision, the decision was in fact made. This case is completely valid and will not be ignored in this study, since the decision was made, there were motivational factors involved in the decision, and there is a perception as to the overall success or failure of the decision.

Business processes and users

Another potential key factor in the decision-making process is the internal business processes or business methods used in the company, as well as the users of the system. Many different software solutions provide the same general functionality, but in different ways. The way in which a given solution works may fit the internal business process of a company better or worse depending on how it works. The intended users of the solution may also affect the decision making process. As an example, people working in the IT industry may desire high customisability and extensibility, whereas less IT-literate users may prefer a simple user interface.

Risk

The perceived risk involved in a Make Vs Buy decision is of importance to this study. The facets of risk assessment this study takes into consideration are the biased and subjective behaviours in risk assessment, not the efficacy of risk 8 assessment techniques. To clarify, this study is concerned with the subjective perception of decision makers.

Project scheduling

Scheduling in project management is the listing of activities, deliverables, and milestones within a project. A schedule also usually includes the planned start and finish date, duration, and resources assigned to each activity. Effective project scheduling is a critical component of successful time management.

In fact, when people discuss the processes for building a schedule, they are usually referring to the first six processes of time management:

1. Plan schedule management.
2. Define project activities.
3. Sequence activities.
4. Estimate resources.
5. Estimate durations.
6. Develop the project schedule.

How to do scheduling in project management

There are three main types of schedules:

1. Master project schedule. A master schedule tends to be a simplified list of tasks with a timeline or project calendar.
2. Milestone schedule or summary schedule. This type of schedule tracks major milestones and key deliverables, but not every task required to complete the project.
3. A detailed project schedule. This is the most thorough project schedule, as it identifies and tracks every project activity. If you have a complex, large, or lengthy project, it's important to have a detailed project schedule to help track everything.

The most common form of project schedule is a Gantt chart. Both a milestone schedule and a detailed project schedule can be created as a Gantt chart. When choosing a scheduling software, look for scheduling tools that allow you to create different views from the same schedule. If you create a detailed schedule with milestones as a Gantt Chart, make sure it can be summarized up to that level for a simpler view that can be easily shared with your team or stakeholders. This gives you the ability to present the same schedule in different formats depending on the level of detail required and the target audience.

Benefits of project scheduling in project management

Project scheduling provides the following benefits:

- Assists with tracking, reporting on, and communicating progress.
- Ensures everyone is on the same page as far as tasks, dependencies, and deadlines.
- Helps highlight issues and concerns, such as a lack of resources.

- Helps identify task relationships.
- Can be used to monitor progress and identify issues early.

7 tips for creating a solid project schedule

1. The time management processes identified earlier are the key steps to creating a project schedule. However, keep these seven tips in mind to make sure your schedule is realistic.Get input from stakeholders. Make sure you don't create your schedule in isolation. It's important to use your team and other stakeholders to identify tasks, resources, dependencies, and durations.

2. Reference past projects. Looking at previous projects with similar scope and requirements can help create realistic estimates and ensure you haven't forgotten any tasks.

3. Include project milestones. Milestones are events or markers that stand for an important point in your project. They're useful for creating a summary schedule, reporting to executives, and identifying problems early. Here are some milestone examples:

4. Project kickoff

5. Design approvals

6. Completion of requirements

7. Product implementation

8. Project closeout

9. Consider any non-work time. For example, make sure vacations and holidays are reflected in your schedule so that you're not assuming people will be working when they're not.

10. Define the critical path on your project. Identifying your project's critical path allows you to prioritize and allocate resources to the most important tasks in the project.

11. Record scheduling assumptions. Write down the logic behind your scheduling predictions. For example, if you assume it will only take 10 hours to complete a task because you're going to have a senior engineer. That way, if you end up with a junior engineer, you can understand and explain why it took twice as long as planned.

12. Keep risk in mind. Identify and document any factors that pose a risk to staying on schedule. This will help your risk management efforts.