Software Project Estimation

Software project estimation is one of the first steps to validate your new product idea. It's the document based on which you're going to choose a software provider. Unfortunately, very often these estimates are nowhere near realistic. That's why we kept working on our estimating process until we achieved the most accurate estimates and now we want to share how we do it.

What is software project estimation?

Estimation is the 4th step in our process of evaluating the project before the development begins. Below you can see all of the stages that allow us to deliver true business value to our clients.

The go or no-go decision on developing a new product often depends on the estimation. These types of projects tend to be significant to the budget and resources distribution. That's why getting an estimation and comparing it to your financial capabilities is a natural part of the process. A typical software project estimation consists of:

- cost estimation
- resources estimation
- effort estimation

Estimation focuses on the team composition and the amount of work in man-hours, needed to be done in order to deliver a project. **The main reason for estimating is to avoid situation when the project exceeds the budget maximum or the expenses are higher than the profits expected from the finished product.**

The basis of each of these methods is always project requirements specification. To estimate a project correctly we need to move past the evaluating stage. The documentation, with mapped out features and architecture, is our guideline. Based on the list of functionalities from that document we can focus on estimating the whole project.

Estimation is the process of finding an estimate, or approximation, which is a value that can be used for some purpose even if input data may be incomplete, uncertain, or unstable.

Estimation determines how much money, effort, resources, and time it will take to build a specific system or product. Estimation is based on −

- Past Data/Past Experience
- Available Documents/Knowledge
- Assumptions
- Identified Risks

The four basic steps in Software Project Estimation are −

- Estimate the size of the development product.
- Estimate the effort in person-months or person-hours.
- Estimate the schedule in calendar months.
- Estimate the project cost in agreed currency.

Observations on Estimation

- Estimation need not be a one-time task in a project. It can take place during −
    - Acquiring a Project.
    - Planning the Project.
    - Execution of the Project as the need arises.
- Project scope must be understood before the estimation process begins. It will be helpful to have historical Project Data.
- Project metrics can provide a historical perspective and valuable input for generation of quantitative estimates.
- Planning requires technical managers and the software team to make an initial commitment as it leads to responsibility and accountability.
- Past experience can aid greatly.
- Use at least two estimation techniques to arrive at the estimates and reconcile the resulting values.
- Plans should be iterative and allow adjustments as time passes and more details are known.

General Project Estimation Approach

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.

Step 1 − Understand the scope of the software to be built.

Step 2 − Generate an estimate of the software size.

- Start with the statement of scope.
- Decompose the software into functions that can each be estimated individually.
- Calculate the size of each function.

- Derive effort and cost estimates by applying the size values to your baseline productivity metrics.
- Combine function estimates to produce an overall estimate for the entire project.

Step 3 − Generate an estimate of the effort and cost. You can arrive at the effort and cost estimates by breaking down a project into related software engineering activities.

- Identify the sequence of activities that need to be performed for the project to be completed.
- Divide activities into tasks that can be measured.
- Estimate the effort (in person hours/days) required to complete each task.
- Combine effort estimates of tasks of activity to produce an estimate for the activity.
- Obtain cost units (i.e., cost/unit effort) for each activity from the database.
- Compute the total effort and cost for each activity.
- Combine effort and cost estimates for each activity to produce an overall effort and cost estimate for the entire project.

Step 4 − Reconcile estimates: Compare the resulting values from Step 3 to those obtained from Step 2. If both sets of estimates agree, then your numbers are highly reliable. Otherwise, if widely divergent estimates occur conduct further investigation concerning whether −

- The scope of the project is not adequately understood or has been misinterpreted.
- The function and/or activity breakdown is not accurate.
- Historical data used for the estimation techniques is inappropriate for the application, or obsolete, or has been misapplied.

Step 5 − Determine the cause of divergence and then reconcile the estimates.

Estimation Accuracy

Accuracy is an indication of how close something is to reality. Whenever you generate an estimate, everyone wants to know how close the numbers are to reality. You will want every estimate to be as accurate as possible, given the data you have at the time you generate it. And of course you don't want to present an estimate in a way that inspires a false sense of confidence in the numbers.

Important factors that affect the accuracy of estimates are −

- The accuracy of all the estimate's input data.
- The accuracy of any estimate calculation.

- How closely the historical data or industry data used to calibrate the model matches the project you are estimating.
- The predictability of your organization's software development process.
- The stability of both the product requirements and the environment that supports the software engineering effort.
- Whether or not the actual project was carefully planned, monitored and controlled, and no major surprises occurred that caused unexpected delays.

Following are some guidelines for achieving reliable estimates −

- Base estimates on similar projects that have already been completed.
- Use relatively simple decomposition techniques to generate project cost and effort estimates.
- Use one or more empirical estimation models for software cost and effort estimation.

To ensure accuracy, you are always advised to estimate using at least two techniques and compare the results.

Estimation Issues

Often, project managers resort to estimating schedules skipping to estimate size. This may be because of the timelines set by the top management or the marketing team. However, whatever the reason, if this is done, then at a later stage it would be difficult to estimate the schedules to accommodate the scope changes.

While estimating, certain assumptions may be made. It is important to note all these assumptions in the estimation sheet, as some still do not document assumptions in estimation sheets.

Even good estimates have inherent assumptions, risks, and uncertainty, and yet they are often treated as though they are accurate.

The best way of expressing estimates is as a range of possible outcomes by saying, for example, that the project will take 5 to 7 months instead of stating it will be complete on a particular date or it will be complete in a fixed no. of months. Beware of committing to a range that is too narrow as that is equivalent to committing to a definite date.

You could also include uncertainty as an accompanying probability value. For example, there is a 90% probability that the project will complete on or before a definite date.

- Organizations do not collect accurate project data. Since the accuracy of the estimates depend on the historical data, it would be an issue.
- For any project, there is a shortest possible schedule that will allow you to include the required functionality and produce quality output. If there is a schedule constraint by

management and/or client, you could negotiate on the scope and functionality to be delivered.

- Agree with the client on handling scope creeps to avoid schedule overruns.
- Failure in accommodating contingency in the final estimate causes issues. For e.g., meetings, organizational events.
- Resource utilization should be considered as less than 80%. This is because the resources would be productive only for 80% of their time. If you assign resources at more than 80% utilization, there is bound to be slippages.

Estimation Guidelines

One should keep the following guidelines in mind while estimating a project −

- During estimation, ask other people's experiences. Also, put your own experiences at task.
- Assume resources will be productive for only 80 percent of their time. Hence, during estimation take the resource utilization as less than 80%.
- Resources working on multiple projects take longer to complete tasks because of the time lost switching between them.
- Include management time in any estimate.
- Always build in contingency for problem solving, meetings and other unexpected events.
- Allow enough time to do a proper project estimate. Rushed estimates are inaccurate, high-risk estimates. For large development projects, the estimation step should really be regarded as a mini project.
- Where possible, use documented data from your organization’s similar past projects. It will result in the most accurate estimate. If your organization has not kept historical data, now is a good time to start collecting it.
- **Use developer-based estimates**, as the estimates prepared by people other than those who will do the work will be less accurate.
- Use several different people to estimate and use several different estimation techniques.
- Reconcile the estimates. Observe the convergence or spread among the estimates. Convergence means that you have got a good estimate. Wideband-Delphi technique can be used to gather and discuss estimates using a group of people, the intention being to produce an accurate, unbiased estimate.
- Re-estimate the project several times throughout its life cycle.

Software Engineering | COCOMO Model

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 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:

- **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 in. 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 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 –** $E = a_b(KLOC)^b b$

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,b,c and d for the Basic Model for the different categories of system:

Software Projects	a	b	c	d
Organic	2.4	1.05	2.5	0.38
Semi Detached	3.0	1.12	2.5	0.35
Embedded	3.6	1.20	2.5	0.32

The effort is measured in Person-Months and as evident from the formula is dependent on Kilo-Lines of code.
The development time is measured in months.

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.

1. **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 systems. 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

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
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

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:

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
Embedded	2.8	1.20

2. **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.

Delphi cost estimation

Delphi Method is a **structured communication technique, originally developed as a systematic, interactive forecasting method which relies on a panel of experts. The experts answer questionnaires in two or more rounds.** After each round, a facilitator provides an anonymous summary of the experts' forecasts from the previous round with the reasons for their judgments. Experts are then encouraged to revise their earlier answers in light of the replies of other members of the panel.

It is believed that during this process the range of answers will decrease and the group will converge towards the "correct" answer. Finally, the process is stopped after a predefined stop criterion (e.g. number of rounds, achievement of consensus, and stability of results) and the mean or median scores of the final rounds determine the results.

Delphi Method was developed in the 1950-1960s at the RAND Corporation.

Wideband Delphi Technique

In the 1970s, Barry Boehm and John A. Farquhar originated the Wideband Variant of the Delphi Method. **The term "wideband" is used because, compared to the Delphi Method, the Wideband Delphi Technique involved greater interaction and more communication between the participants.**

In Wideband Delphi Technique, the estimation team comprise the project manager, moderator, experts, and representatives from the development team, constituting a 3-7 member team. There are two meetings −

- Kickoff Meeting
- Estimation Meeting

Wideband Delphi Technique – Steps

Step 1 − Choose the Estimation team and a moderator.

Step 2 − The moderator conducts the **kickoff meeting**, in which the **team is presented with the problem specification and a high level task list, any assumptions or project constraints.** The team discusses on the problem and estimation issues, if any. They also decide on the units of estimation. The moderator guides the entire discussion, monitors time and after the kickoff meeting, **prepares a structured document containing problem specification, high level task list, assumptions, and the units of estimation that are decided.** He then forwards copies of this document for the next step.

Step 3 − Each Estimation team member then individually generates a detailed WBS, estimates each task in the WBS, and documents the assumptions made.

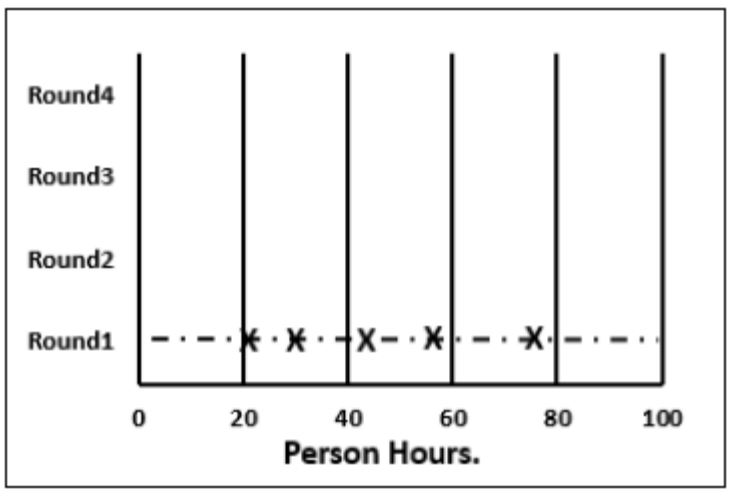
Wideband Delphi Estimation Sheet						
Project: <Project Name>				**Estimation Units:**	Person Hours	
Estimation Team Member:	<Name>			**Date:**	<MM-DD-YY>	
Task	**Initial Estimate**	**Change 1**	**Change 2**	**Change 3**	**Change 4**	**Final**
Task1	n_1					
Task2	n_2					
Task3	n_3					
Task4	n_4					
Task5	n_5					
Task6	n_6					
Task7	n_7					
Task8	n_8					
Net Change						
Total	$\sum n_i$					

Step 4 − The moderator calls the Estimation team for the Estimation meeting. If any of the Estimation team members respond saying that the estimates are not ready, the moderator gives more time and resends the Meeting Invite.

Step 5 − The entire Estimation team assembles for the estimation meeting.

Step 5.1 − At the beginning of the Estimation meeting, the moderator collects the initial estimates from each of the team members.

Step 5.2 − He then plots a chart on the whiteboard. He plots each member's total project estimate as an X on the Round 1 line, without disclosing the corresponding names. The Estimation team gets an idea of the range of estimates, which initially may be large.



Step 5.3 − Each team member reads aloud the detailed task list that he/she made, identifying any assumptions made and raising any questions or issues. The task estimates are not disclosed.

The individual detailed task lists contribute to a more complete task list when combined.

Step 5.4 − The team then discusses any doubt/problem they have about the tasks they have arrived at, assumptions made, and estimation issues.

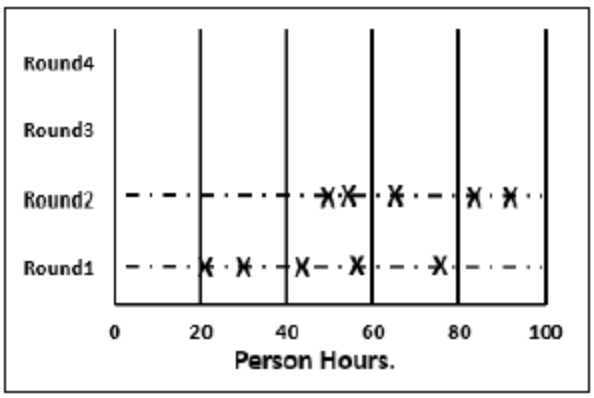
Step 5.5 − Each team member then revisits his/her task list and assumptions, and makes changes if necessary. The task estimates also may require adjustments based on the discussion, which are noted as +N Hrs. for more effort and –N Hrs. for less effort.

The team members then combine the changes in the task estimates to arrive at the total project estimate.

Wideband Delphi Estimation Sheet						
Project: <Project Name>				**Estimation Units:**	Person Hours	
Estimation Team Member:	<Name>			**Date:**	<MM-DD-YY>	
Task	**Initial Estimate**	**Change 1**	**Change 2**	**Change 3**	**Change 4**	**Final**
Task1	n_1	-1				
Task2	n_2	-2				
Task3	n_3	-4				
Task4	n_4	5				
Task5	n_5	0				
Task6	n_6	0				
Task7	n_7	2				
Task8	n_8	-3				
Net Change		**-3**				
Total	$\sum n_i$	$\sum n_i$-3				

Step 5.6 − The moderator collects the changed estimates from all the team members and plots them on the Round 2 line.

In this round, the range will be narrower compared to the earlier one, as it is more consensus based.



Step 5.7 − The team then discusses the task modifications they have made and the assumptions.

Step 5.8 − Each team member then revisits his/her task list and assumptions, and makes changes if necessary. The task estimates may also require adjustments based on the discussion.

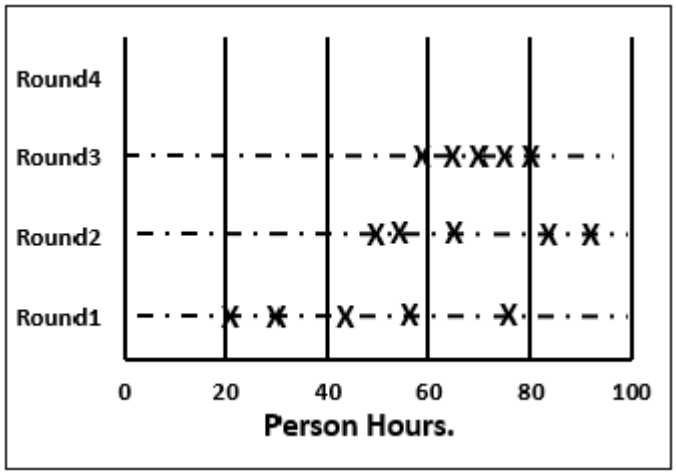
The team members then once again combine the changes in the task estimate to arrive at the total project estimate.

Step 5.9 − The moderator collects the changed estimates from all the members again and plots them on the Round 3 line.

Again, in this round, the range will be narrower compared to the earlier one.

Step 5.10 − Steps 5.7, 5.8, 5.9 are repeated till one of the following criteria is met −

- Results are converged to an acceptably narrow range.
- All team members are unwilling to change their latest estimates.
- The allotted Estimation meeting time is over.



Step 6 − The Project Manager then assembles the results from the Estimation meeting.

Step 6.1 − He compiles the individual task lists and the corresponding estimates into a single master task list.

Step 6.2 − He also combines the individual lists of assumptions.

Step 6.3 − He then reviews the final task list with the Estimation team.

Advantages and Disadvantages of Wideband Delphi Technique

Advantages

- Wideband Delphi Technique is a consensus-based estimation technique for estimating effort.
- Useful when estimating time to do a task.
- Participation of experienced people and they individually estimating would lead to reliable results.
- People who would do the work are making estimates thus making valid estimates.

- Anonymity maintained throughout makes it possible for everyone to express their results confidently.
- A very simple technique.
- Assumptions are documented, discussed and agreed.

Disadvantages

- Management support is required.
- The estimation results may not be what the management wants to hear.

Functional Point Analysis - **Estimation of the software product including its testing could also be made in the terms of its functionality or functions**

Whenever a software project comes, the organization usually try to make estimate of the project in the terms of cost and time, so that accordingly resources such as testers, tools, frameworks and time schedules for each phase of the project could be assessed and measured. But, did you know that an estimate of the software product including its testing could also be made in the terms of its functionality or functions. The concept is all about the **functional point analysis**.

What is Functional Point Analysis?

Abbreviated as FPA, functional point analysis is one of the mostly preferred and widely used estimation technique used in the software engineering. FPA is used to make estimate of the software project, including its **testing in the terms of functionality or function size of the software product.** However, functional point analysis may be used for the test estimation of the product.

The functional size of the product is measured in the terms of the function point, which is a standard of measurement to measure the software application.

Functional Point Analysis Objectives:

The **basic and primary objective of the functional point analysis is to measure and provide the software application functional size to the client, customer and the stakeholder on their request.** Further, it is used to measure the software project development along with its maintenance, consistently throughout the project irrespective of the tools and the technologies.

Functional Point Analysis Components:

The functional point analysis approach consists of some components. Based on the interaction of the system with the external world such as other system, application, users, its **component** may be categorized into five different forms.

- **External Inputs:**

A process where inputs are imported from outside the boundary of the system through external sources such as external system or from the user's input screen to form internal logic database file, where input data consists of control information and business information, but internal logic file gets updated only for the business information.

- **External Output:**

A process where data is passed to the outside system or application from inside in the form of reports or files. Generally, these reports or files are get derived from the data present in the internal logic file.

- **Internal Logic Files:**

It is a group of data present within the system which are interrelated to each other, and is being maintained by the inputs provided from the external sources or system.

- **External Logic Files:**

It may be seen as a group of inter-related data present outside the system boundary and may be the internal logic file of some other external application, but it works as an external logic or interface file for the system for it use or reference purpose.

- **Inquiries:**

A process consisting of both input and output component to extract data from the internal and external logic files.

Steps to Count the Functional Points:

Below given are the subsequent steps, used in counting the functional points of a system.

- **Type of count**The First step involves the task of determining the type of functional count for the application. Basically, there are three forms of functional point (FP) count, as stated below:
    - **Development Project FP Count:**It measures the functionality or functions of the system being delivered to the end-users, including its first installation.

    - **Enhancement Project FP Count:** Measure the modification brought in the system.
    - **Application FP count:** Measures the functionality of the already existing system being provided to the users.

- **Scope and Boundary of the Count** In the second step, scope and boundary of the count needs to be identified, where scope could be determined with the help of data screens, reports and files, whereas boundary depend upon the perspective of drawing out the boundary for the external system or application.
- **Unadjusted Function Point Count** It is the core step of the process, where unadjusted functional point count produced by the five components as stated above **(External Inputs, External Output, Internal Logic files, External Logic files, Inquiries)** are measured.
- **Value Adjustment Factor** This step involves the task of determining the value adjustment factor(VAF), where VAF comprises of 14 General system characteristics(GSC) of the system or application representing the different types of characteristic possessed by the application, and is measured or rated on the scale of 0 to 5. The rated value of all the 14 GSC are combined to give out a mathematical value, namely Total Degree Influence(TDI), used in calculating the VAF. The value of TDI may vary from 0 to 35. Below given is the 14 GSCs and the mathematical formula for calculating the VAF.

14 GSCs

1. Data Communication.
2. Distributed data processing.
3. Performance.
4. Heavily used configuration.
5. Transaction rate.
6. Online data entry.
7. End user efficiency.
8. Online update.
9. Complex processing.
10. Reusability.
11. Installation ease.
12. Operational ease.

13. Facilitate change.
14. Multiple sites.

VAF= (TDI*0.01) + 0.65

Adjusted Functional Point Count

After calculating the unadjusted functional point count and value adjustment factor, this step makes use of both the value to result out the Adjusted Functional point count with the help of following formula.

Adjusted FPC= Non-adjusted FPC*VAF

Benefits of Functional Point Analysis/ FPA:

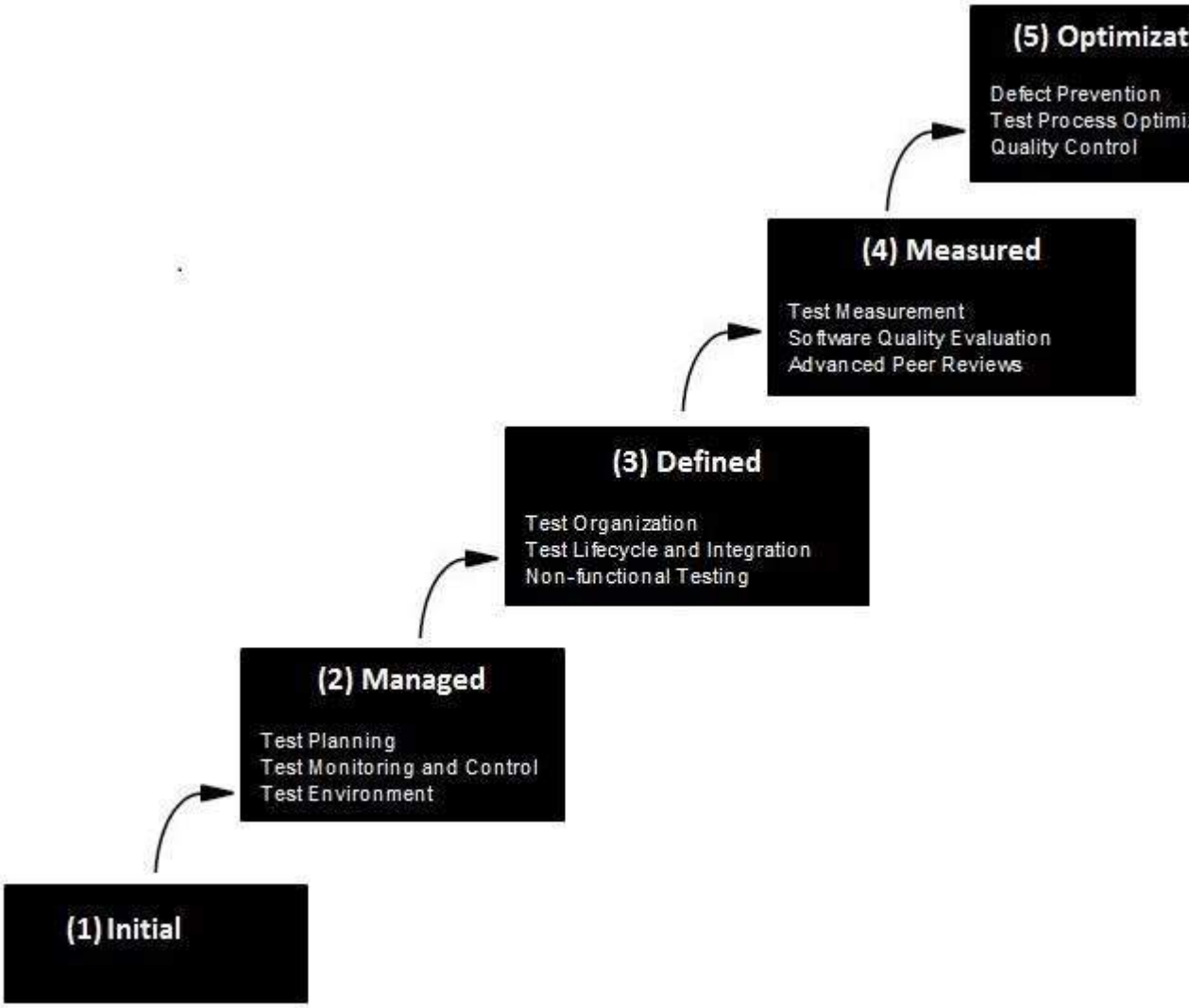
The merits associated with the functional point analysis are:

- Measures the productivity of the project and the process.
- Technology independent approach.
- May be used to measure the support requirement, needed for the purpose of system maintenance.
- Useful in the estimation of the time required in bringing changes to the product or in developing a new application.
- Useful for non-technical people in understanding the functional size of the application such as end-users, customers, etc.
- May be repeated.
- Works well for the use cases and user stories.

What is Capability Maturity Model?

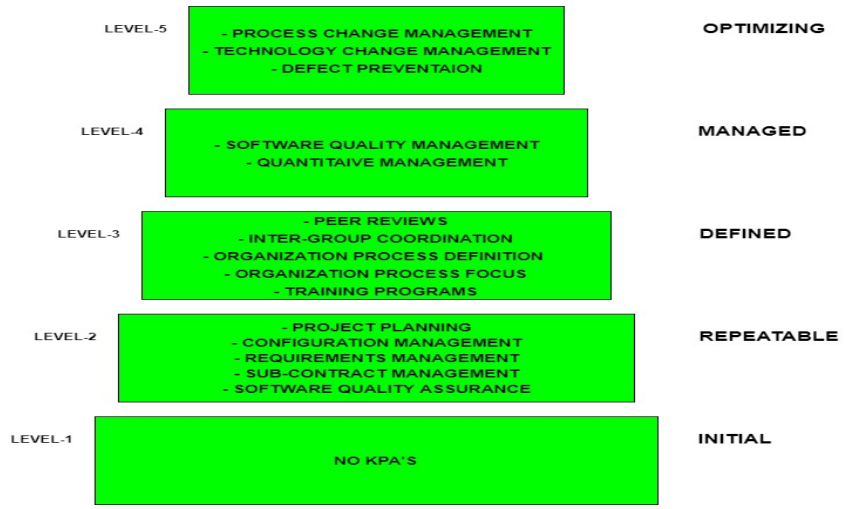
The Software Engineering Institute (SEI) Capability Maturity Model (CMM) specifies an increasing series of levels of a software development organization. The higher the level, the better the software development process, hence reaching each level is an expensive and time-consuming process.

Levels of CMM



- CMM was developed by the Software Engineering Institute (SEI) at Carnegie Mellon University in 1987.
- It is not a software process model. It is a framework that is used to analyze the approach and techniques followed by any organization to develop software products.
- It also provides guidelines to further enhance the maturity of the process used to develop those software products.
- It is based on profound feedback and development practices adopted by the most successful organizations worldwide.
- This model describes a strategy for software process improvement that should be followed by moving through 5 different levels.
- Each level of maturity shows a process capability level. All the levels except level-1 are further described by Key Process Areas (KPA's).
- **Shortcomings of SEI/CMM:**

- It encourages the achievement of a higher maturity level in some cases by displacing the true mission, which is improving the process and overall software quality.
- It only helps if it is put into place early in the software development process.
- It has no formal theoretical basis and in fact is based on the experience of very knowledgeable people.
- It does not have good empirical support and this same empirical support could also be constructed to support other models.
- **Key Process Areas (KPA's):**
  Each of these KPA's defines the basic requirements that should be met by a software process in order to satisfy the KPA and achieve that level of maturity.
- Conceptually, key process areas form the basis for management control of the software project and establish a context in which technical methods are applied, work products like models, documents, data, reports, etc. are produced, milestones are established, quality is ensured and change is properly managed.
-



-
- The 5 levels of CMM are as follows:
- **Level-1: Initial –**
- No KPA's defined.
- Processes followed are Adhoc and immature and are not well defined.
- Unstable environment for software development.
- No basis for predicting product quality, time for completion, etc.
- **Level-2: Repeatable –**

- Focuses on establishing basic project management policies.
- Experience with earlier projects is used for managing new similar natured projects.
- **Project Planning-** It includes defining resources required, goals, constraints, etc. for the project. It presents a detailed plan to be followed systematically for the successful completion of good quality software.
- **Configuration Management-** The focus is on maintaining the performance of the software product, including all its components, for the entire lifecycle.
- **Requirements Management-** It includes the management of customer reviews and feedback which result in some changes in the requirement set. It also consists of accommodation of those modified requirements.
- **Subcontract Management-** It focuses on the effective management of qualified software contractors i.e. it manages the parts of the software which are developed by third parties.
- **Software Quality Assurance-** It guarantees a good quality software product by following certain rules and quality standard guidelines while developing.
- **Level-3: Defined –**
- At this level, documentation of the standard guidelines and procedures takes place.
- It is a well-defined integrated set of project-specific software engineering and management processes.
- **Peer Reviews-** In this method, defects are removed by using a number of review methods like walkthroughs, inspections, buddy checks, etc.
- **Intergroup Coordination-** It consists of planned interactions between different development teams to ensure efficient and proper fulfilment of customer needs.
- **Organization Process Definition-** Its key focus is on the development and maintenance of the standard development processes.
- **Organization Process Focus-** It includes activities and practices that should be followed to improve the process capabilities of an organization.
- **Training Programs-** It focuses on the enhancement of knowledge and skills of the team members including the developers and ensuring an increase in work efficiency.
- **Level-4: Managed –**
- At this stage, quantitative quality goals are set for the organization for software products as well as software processes.

- The measurements made help the organization to predict the product and process quality within some limits defined quantitatively.
- **Software Quality Management-** It includes the establishment of plans and strategies to develop quantitative analysis and understanding of the product's quality.
- **Quantitative Management-** It focuses on controlling the project performance in a quantitative manner.
- **Level-5: Optimizing –**
- This is the highest level of process maturity in CMM and focuses on continuous process improvement in the organization using quantitative feedback.
- Use of new tools, techniques, and evaluation of software processes is done to prevent recurrence of known defects.
- **Process Change Management-** Its focus is on the continuous improvement of the organization's software processes to improve productivity, quality, and cycle time for the software product.
- **Technology Change Management-** It consists of the identification and use of new technologies to improve product quality and decrease product development time.
- **Defect Prevention-** It focuses on the identification of causes of defects and prevents them from recurring in future projects by improving project-defined processes.

- **Level One :Initial** - The software process is characterized as inconsistent, and occasionally even chaotic. Defined processes and standard practices that exist are abandoned during a crisis. Success of the organization majorly depends on an individual effort, talent, and heroics. The heroes eventually move on to other organizations taking their wealth of knowledge or lessons learnt with them.
- **Level Two: Repeatable** - This level of Software Development Organization has a basic and consistent project management processes to track cost, schedule, and functionality. The process is in place to repeat the earlier successes on projects with similar applications. Program management is a key characteristic of a level two organization.
- **Level Three: Defined** - The software process for both management and engineering activities are documented, standardized, and integrated into a standard software process for the entire organization and all projects across the organization use an approved, tailored version of the organization's standard software process for developing, testing and maintaining the application.

- **Level Four: Managed** - Management can effectively control the software development effort using precise measurements. At this level, organization set a **quantitative quality goal** for both software process and software maintenance. At this maturity level, the performance of processes is controlled using statistical and other quantitative techniques, and is quantitatively predictable.
- **Level Five: Optimizing** - The Key characteristic of this level is focusing on continually improving process performance through both incremental and innovative technological improvements. At this level, changes to the process are to improve the process performance and at the same time maintaining statistical probability to achieve the established quantitative process-improvement objectives.

Software Configuration Management

When we develop software, the product (software) undergoes many changes in their maintenance phase; we need to handle these changes effectively.

Several individuals (programs) works together to achieve these common goals. This individual produces several work product (SC Items) e.g., Intermediate version of modules or test data used during debugging, parts of the final product.

The elements that comprise all information produced as a part of the software process are collectively called a software configuration.

As software development progresses, the number of Software Configuration elements (SCI's) grow rapidly.

These are handled and controlled by SCM. This is where we require software configuration management.

A configuration of the product refers not only to the product's constituent but also to a particular version of the component.

Therefore, SCM is the discipline which

- Identify change
- Monitor and control change
- Ensure the proper implementation of change made to the item.
- Auditing and reporting on the change made.

Configuration Management (CM) is a technique of identifying, organizing, and controlling modification to software being built by a programming team.

The objective is to maximize productivity by minimizing mistakes (errors).

CM is used to essential due to the inventory management, library management, and updation management of the items essential for the project.

Why do we need Configuration Management?

Multiple people are working on software which is consistently updating. It may be a method where multiple version, branches, authors are involved in a software project, and the team is geographically distributed and works concurrently. It changes in user requirements, and policy, budget, schedules need to be accommodated.

Importance of SCM

It is practical in controlling and managing the access to various SCIs e.g., by preventing the two members of a team for checking out the same component for modification at the same time.

It provides the tool to ensure that changes are being properly implemented.

It has the capability of describing and storing the various constituent of software.

SCM is used in keeping a system in a consistent state by automatically producing derived version upon modification of the same component.

COCOMO Model Examples:

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

a_1,a_2,b_1,b_2 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)**.

Estimation of development effort

For the three classes of software products, the formulas for estimating the effort based on the code size are shown below:

- **E** = a_b(KLOC)^{b}b **[man-months]**
- **D** = c_b(E)^{d}b **[months]**
- **People required** = Effort Applied / Development Time **[count]**

Where

E=Effort Applied in Person-Months

D= Development Time in Months

Organic: Effort = 2.4(KLOC) 1.05 PM

Semi-detached: Effort = 3.0(KLOC) 1.12 PM

Embedded: Effort = 3.6(KLOC) 1.20 PM

Estimation of development time

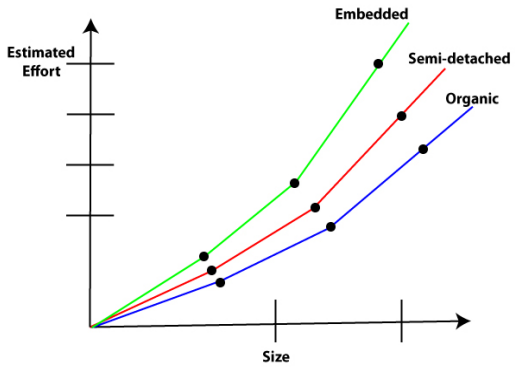
For the three classes of software products, the formulas for estimating the development time based on the effort are given below:

Organic: Tdev = 2.5(Effort) 0.38 Months

Semi-detached: Tdev = 2.5(Effort) 0.35 Months

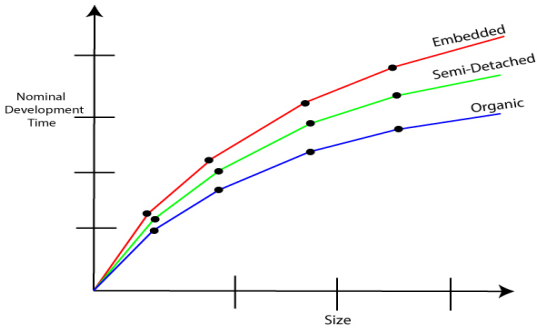
Embedded: Tdev = 2.5(Effort) 0.32 Months

Some insight into the basic COCOMO model can be obtained by plotting the estimated characteristics for different software sizes. Fig shows a plot of estimated effort versus product size. From fig, we can observe that the effort is somewhat superliner in the size of the software product. Thus, the effort required to develop a product increases very rapidly with project size.



Effort versus product size

The development time versus the product size in KLOC is plotted in fig. From fig it can be observed that the development time is a sub linear function of the size of the product, i.e. when the size of the product increases by two times, the time to develop the product does not double but rises moderately. This can be explained by the fact that for larger products, a larger number of activities which can be carried out concurrently can be identified. The parallel activities can be carried out simultaneously by the engineers. This reduces the time to complete the project. Further, from fig, it can be observed that the development time is roughly the same for all three categories of products. For example, a 60 KLOC program can be developed in approximately 18 months, regardless of whether it is of organic, semidetached, or embedded type.



Development time versus size

From the effort estimation, the project cost can be obtained by multiplying the required effort by the manpower cost per month. But, implicit in this project cost computation is the assumption that the entire project cost is incurred on account of the manpower cost alone. In addition to manpower cost, a project would incur costs due to hardware and software required for the project and the company overheads for administration, office space, etc.

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.

Example1: Suppose a project was estimated to be 400 KLOC. Calculate the effort and development time for each of the three model i.e., organic, semi-detached & embedded.

Solution: The basic COCOMO equation takes the form:

Effort=a_1*(KLOC) ^ a_2 PM
Tdev=b_1*(efforts) ^ b_2 Months
Estimated Size of project= 400 KLOC

(i)Organic Mode

E = 2.4 * (400) ^ 1.05 = 1295.31 PM
D = 2.5 * (1295.31) ^ 0.38=38.07 PM

(ii)Semidetached Mode

E = 3.0 * (400) ^1.12=2462.79 PM
D = 2.5 * (2462.79) ^ 0.35=38.45 PM

(iii) Embedded Mode

E = 3.6 * (400) ^ 1.20 = 4772.81 PM
D = 2.5 * (4772.8) ^ 0.32 = 38 PM

Example2: A project size of 200 KLOC is to be developed. Software development team has average experience on similar type of projects. The project schedule is not very tight. Calculate the Effort, development time, average staff size, and productivity of the project.

Solution: The semidetached mode is the most appropriate mode, keeping in view the size, schedule and experience of development time.

Hence E=3.0(200) ^ 1.12=1133.12PM
D=2.5(1133.12)^ 0.35=29.3PM

$$\text{Average Staff Size (SS)} = \frac{E}{D} \text{ Persons}$$

$$= \frac{1133.12}{29.3} = 38.67 \text{ Persons}$$

$$\text{Productivity} = \frac{KLOC}{E} = \frac{200}{1133.12} = 0.1765 \text{ KLOC/PM}$$

P = 176 LOC/PM

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

Project	a_i	b_i	c_i	d_i
Organic	2.4	1.05	2.5	0.38
Semidetached	3.0	1.12	2.5	0.35
Embedded	3.6	1.20	2.5	0.32