

## Estimating software projects

R. Agarwal, Manish Kumar<sup>†</sup>, Yogesh, S. Mallick, R.M. Bharadwaj, D. Anantwar  
 Infosys Technologies Limited, <sup>†</sup>IIM, Calcutta, India  
 Email: {rakesh\_a; yogesh; sudeepm; bharadrm; d\_anantwar}@infy.com

### Abstract

Software Cost Estimation (SCE) continues to be a weak link in software project management. It is the responsibility of the project manager to make accurate estimations of effort and cost. This is particularly true for projects subject to competitive bidding where a bid too high compared with competitors would result in losing the contract or a bid too low could result in a loss to the organization. From an estimate, the management often decides whether to proceed with the project. Industry has a need for accurate estimates of effort and size at a very early stage in a project. However, when software cost estimates are done early in the software development process the estimate can be based on wrong or incomplete requirements. Software cost estimate process is the set of techniques and procedures that organizations use to arrive at an estimate.

Why SCE is difficult and error prone?

- Software cost estimation requires a significant amount of effort to perform it correctly.
- SCE is often done hurriedly, without an appreciation for the effort required.
- You need experience at developing estimates, especially for large projects.
- Human bias i.e. an Estimator is likely to consider how long a certain portion of the system would take, and then to merely extrapolate this estimate to the rest of the system, ignoring the non-linear aspects of software development.
- Costs and schedules are often pre-determined by an outside source.
- An in-depth analysis of the software development process was not undertaken or in many cases, is not fully understood
- There is a general lack of acceptance that developing software is an expensive endeavor.

The causes of poor and inaccurate estimation are: (a) imprecise and drifting requirements. (b) New software projects are nearly always different from the last. (c) Software practitioners don't collect enough information about past projects. (d) Estimates are forced to match the resources available.

The software estimation process discussed in this paper describes the steps required for establishing initial software Life Cycle Cost estimates and then tracking and refining those estimates throughout the life of the project. Establishment of this process early in the life cycle will result in greater accuracy and credibility of estimates and a clearer understanding of the factors that influence software development costs.

**Keywords:** Estimation, software project, risk, software engineering.

### 1.1 Introduction

Software Cost Estimation continues to be a weak link in software project management. It is the responsibility of the project manager to make accurate estimations of effort and cost. This is very important for projects subject to competitive bidding, where a bid too high compared with competitors would result in losing the contract or a bid too low result in a loss to the organisation. From a project leaders estimate the management often decide whether to proceed with the project. Industry has a need for accurate estimates of effort and size at a very early stage in a project. However, when software cost estimates are done early in the software development process the estimate can be based on wrong or incomplete requirements. Software cost estimate process is the set of techniques and procedures that an organisation uses to arrive at an estimate. An important aspect of software projects is to know the cost; the major contributing factor is effort.

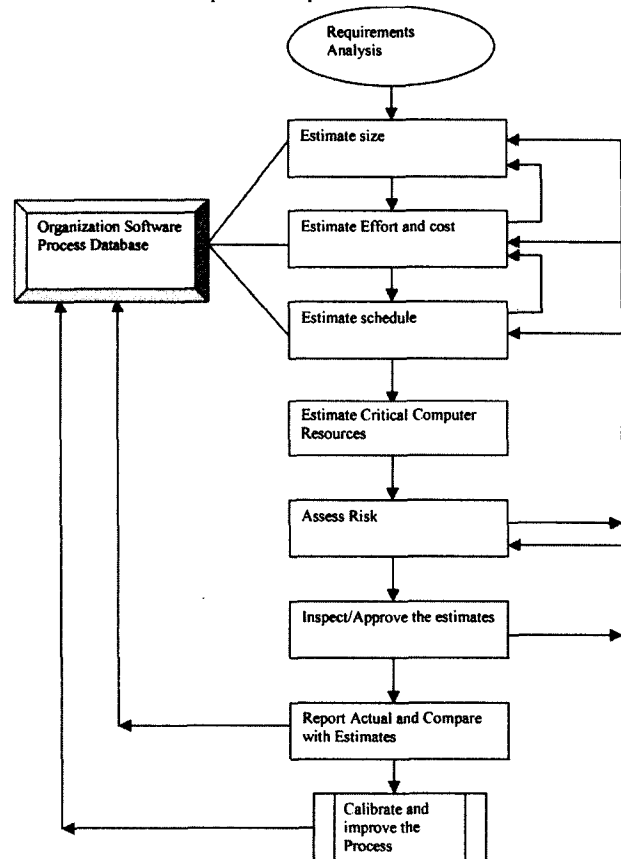
The software estimation process is discussed in the following sections. The steps required for establishing initial software Life Cycle Cost (LCC) estimates and then tracking and refining those estimates throughout the life of the project are described. Establishment of this process early in the life cycle will result in greater accuracy and credibility of estimates and a clearer understanding of the factors that influence software development costs. This process also provides methods for project personnel to identify

and monitor cost and to schedule risk factors. Next sections deals with specific methods generally used for cost estimation. They are COCOMO, COCOMO2, Function Points and wideband Delphi technique.

### 1.2 Process for software estimation activities

Software estimation is a continual process that should be used throughout the life cycle of a project. The software estimation process consists of the following procedures:

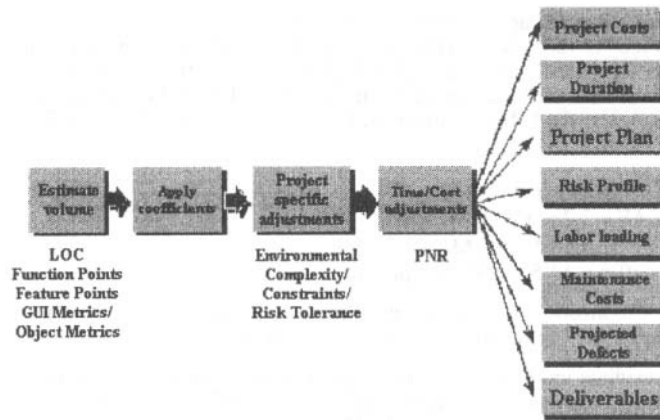
- Estimate size
- Estimate cost and effort
- Estimate schedule
- Estimate critical computer resources
- Assess risks
- Inspect/Approve
- Track and report estimates
- Measure and improve the process.



The process activities for developing the size, effort, and cost are shown before the schedule estimate in Figure because this is the sequence often used by the cost models. However, a development schedule is often mandated before the scope of the effort is clearly understood. The early establishment of a work breakdown structure (WBS) helps to divide the effort into distinct work segments that can be scheduled and prioritized. The following sections provide an overview of the steps used to develop a thorough software project estimate.

### 1.3 The Software Cost Estimating Lifecycle

The first step in estimating the development cost of a given project is to estimate the volume of the application to be developed. In other words, how large is this effort. There are a variety of approaches to estimating volume, including parametric and heuristic methods. The computer uses coefficients to generate a project specific set of volume versus cost curves and to determine the total effort and schedule for this particular project.



variables need to apply. Management constraints such as the time-cost trade off can still be applied, and the entire set of calculated outputs available for parametric estimates are applicable.

**1.5 Parametric Approaches**

In this section we will discuss the Boehm's COCOMO, COCOMO-II, COCOTS and Albrecht's function points models.

**1.5.1 COCOMO**

The CONstructive COST Model (COCOMO) was first proposed by Barry W. Boehm in his book Software Engineering Economics in 1981.

The most fundamental calculation in the COCOMO model is the use of the Effort Equation to estimate the number of Person-Months required to develop a project. The other COCOMO results, like estimates of Person requirements and schedule, are derived from this quantity.

$$EFFORT = A \times (SIZE)^B$$

Where A is proportionality constant and B represents economy or diseconomy of scale. B depends on the development mode. The estimate of a project's size is in SLOC. SLOC is defined such that:

Only SOURCE LINES that are delivered as part of the product are included -- test drivers and other support software are excluded.

- Source lines are created by the project staff -- code created by applications generators are excluded.
- One Instruction is one line of CODE.
- Declarations are counted as instructions.
- Comments are not counted as instructions.

The Development Mode: This is one of the most important factors that contribute to a project's duration and cost. It effects the economy and diseconomy of scale. Every project is considered to be developed in one of the three modes:

Organic Mode: The project is developed in a familiar, stable environment, and the product is similar to previously developed products. The product is relatively small, and requires little innovation. Example: An accounting system.

Semidetached Mode: The project's characteristics are intermediate between Organic and Embedded.

Embedded Mode. The project is characterized by tight, inflexible constraints and interface requirements. An embedded mode project will require a great deal of innovation. Example: A real time system with timing constraints and customized hardware.

COCOMO is defined in terms of three different models: the Basic model, the Intermediate model, and the Detailed model. The more complex models account for more factors that influence software projects, and make more accurate estimates.

**1.5.1.1 The Basic Model**

The Basic model makes its estimates of required effort (measured in Person-Months-PM) based primarily on estimate of the software project's size (as measured in thousands of source lines of code-- KSLOC):

Development Mode	Basic Effort Equation
Organic:	Effort = 2.4 * (KSLOC) <sup>1.05</sup>
Semidetached:	Effort = 3.0 * (KSLOC) <sup>1.12</sup>
Embedded:	Effort = 3.6 * (KSLOC) <sup>1.20</sup>

The Basic model bases its estimates of the duration (TDEV -- the time to develop) of the project on these equations:

Development Mode	Basic Schedule Equation
Organic:	TDEV = 2.5 * (Effort) <sup>0.38</sup>
Semidetached:	TDEV = 2.5 * (Effort) <sup>0.35</sup>
Embedded:	TDEV = 2.5 * (Effort) <sup>0.32</sup>

As an example, an Organic mode project consisting of 3,000 source lines of code is estimated to require 7.6 Person-Months of effort, and 5.4 months to complete it:

**Effort = 2.4 \* 31.05 = 7.6 Person-Months**

**TDEV = 2.5 \* 7.60.38 = 5.4 Months**

**Average staffing: 7.6 / 5.4 = 1.4 person**

The Basic model is suitable for early, rough, estimates of a project's effort,

The next step is to adjust this estimate for project specific environmental factors like team capability, problem complexity, development environment etc. The result is an estimate of the total effort and time assuming that the project is developed at this optimal pace. In the real world, it is not unusual for business pressures to require that the project be developed at a non-optimal, typically accelerated, pace. This is possible, but efficiency goes down and the costs therefore go up. In the following sections approaches to estimating volume are discussed.

**1.4 Approaches for Defining Software Volume**

There are two major approaches to estimating software volume, parametric and heuristic. For larger projects, it is recommended to use more than one approach and to correlate the results.

**1.4.1 Parametric Approaches**

Parametric estimates approximate the software delivered volume using a predictor that can be more easily determined earlier in the software lifecycle, called a metric. Effective metrics should be strongly correlated with the project development effort, and should be easier to estimate than a direct estimate of labor. Source Lines of Code (SLOC) and Function Points (FP) are two metric commonly used. No single metric has shown itself to be ideal for all types of projects. Instead, the estimator must carefully select an appropriate metric for the specific project being estimated.

As shown in the figure, for data driven systems, typical of the MIS and commercial development worlds, function points dominate. Function points uses screens, reports, tables, interfaces, and queries as the metrics to predict program volume. For computation driven systems, typical of the embedded and systems development worlds, SLOC dominate. SLOC measures the application in terms of lines of code including job control language but not including comments. For applications that have a strong data driven component but that also include a strong algorithmic component Feature Points were developed, adding the algorithm count to the traditional function point based metrics.

Two more new approaches are object metrics and Graphical User Interface (GUI) metrics. Object metrics use the number of objects to be developed to give required functionality and are primarily applicable to object oriented development. Graphical User Interface metrics use menu choices, dialog boxes, windows, tables, and reports and are primarily applicable to client-server development. They are still under development. One major advantage of parametric approaches is that the basis of volume estimate is quantified as part of the estimate. This can be useful to detect when a project is experiencing scope creep at an early stage in the software lifecycle.

**1.4.2 Heuristic Approaches**

There are two primary heuristic approaches, top down and bottom up. Examples of top down estimation include historical comparisons, design to cost specifications, quick-and-dirty estimates, and the Delphi technique whereby various experts arrive at a consensus.

In bottom up estimate, developers estimate the effort required to code each module in the application. Developers are normally pretty good at estimating the coding only portion of a job. Standard allocations for this type of project are then applied to calculate the effort for non-coding activities. The individuals responsible for each area, review these estimates and adjust if required. The effort for each individual task is added to determine the total development cost.

When using heuristic approaches, neither coefficients nor environmental

duration, and cost. The estimates are generally within a factor of 2 of the actual results 60% of the time. The Intermediate model provides much better estimates --its projections are within 20% of the actual results 68% of the time.

1.5.1.2 The Intermediate Model

The Intermediate and Detailed models use an Effort Adjustment Factor (EAF) and slightly different coefficients for the effort equations than the Basic model:

Development Mode	Intermediate Effort Equation
Organic:	Effort = EAF * 3.2 * (KSLOC) <sup>1.05</sup>
Semidetached:	Effort = EAF * 3.0 * (KSLOC) <sup>1.12</sup>
Embedded:	Effort = EAF * 2.8 * (KSLOC) <sup>1.20</sup>

The Intermediate model produces better results because it uses settings for 15 Cost Drivers that determine the effort and duration of software projects. The Cost Drivers include factors such as Product Complexity, Programmer Capability, and "Use of Software Tools". The table lists the cost drivers used in the standard COCOMO model and the Effort Multipliers associated with different ratings.

The Effort Adjustment Factor is simply the product of the Effort Multi-

		{PRIVATE}Cost Driver		Very Low	Low	Nominal	High	Very High	Extra High
Product Attributes		Required Software Reliability	RELY	0.75	0.88	1.00	1.15	1.40	-
		Database Size	DATA	-	0.94	1.00	1.08	1.16	-
		Product Complexity	CPLX	0.70	0.85	1.00	1.15	1.30	1.65
Computer Attributes		Execution Time Constraint	TIME	-	-	1.00	1.11	1.30	1.66
		Main Storage Constraint	STOR	-	-	1.00	1.06	1.21	1.56
		Virtual Machine Volatility	VIRT	-	0.87	1.00	1.15	1.30	-
		Computer Turnaround Time	TURN	-	0.87	1.00	1.07	1.15	-
Personnel Attributes		Analyst Capability	ACAP	1.46	1.19	1.00	0.86	0.71	-
		Applications Experience	AEXP	1.29	1.13	1.00	0.91	0.82	-
		Programmer Capability	PCAP	1.42	1.17	1.00	0.86	0.70	-
		Virtual Machine Experience	VEXP	1.21	1.10	1.00	0.90	-	-
		Language Experience	LEXP	1.14	1.07	1.00	0.95	-	-
Project Attributes		Modern Programming Practices	MODP	1.24	1.10	1.00	0.91	0.82	-
		Use of Software Tools	TOOL	1.24	1.10	1.00	0.91	0.83	-
		Required Development Schedule	SCED	1.23	1.08	1.00	1.04	1.10	-

1.5.1.3 The Detailed Model

The Detailed model differs from the Intermediate model in only one major aspect: the Detailed model uses different Effort Multipliers for each phase of a project. This phase dependent Effort Multipliers yield better estimates than the Intermediate model. The six phases COCOMO defines are Requirements, Product Design, Detailed Design, Code and Unit Test, Integrate & Test and Maintenance.

The phases from Product Design through Integrate & Test are called the Development phases. Estimates for the Requirements phase and for the Maintenance phase are performed in a different way than estimates for the four Development phases.

The Programmer Capability cost driver is a good example of a phase dependent cost driver. The Very High rating for the Programmer Capability Cost Driver corresponds to an Effort Multiplier of 1.00 (no influence) for the Product Design phase of a project, but an Effort Multiplier of 0.65 is used for the Detailed Design phase. These ratings indicate that good programmers can save time and money on the later phases of the project, but they don't have an impact on the Product Design phase because they aren't involved.

A word of Caution: The numerical values for the effort multipliers should be calibrated using historical data of completed projects.

1.5.2 COCOMO II

This is an advanced version of old COCOMO model. It is still underdevelopment. Two models have been developed till 1999 to be used in two stages of software development. They are Early design model and post

ers corresponding to each of the cost drivers for the project. For example, if the project is rated Very High for Complexity (Effort Multiplier of 1.30), and Low for Tools Use (Effort Multiplier of 1.10), and all of the other cost drivers are rated to be Nominal. These Effort Multipliers are used to calculate the Effort Adjustment Factor, which is used in the effort equation:

$$EAF = 1.30 \times 1.10 = 1.43$$

$$Effort = EAF \times 3.2 \times 3^{1.05} = 14.5 \text{ SM}$$

$$TDEV = 2.5 \times 14.5^{0.38} = 6.9 \text{ Months}$$

$$\text{Average staffing: } 14.5 \div 6.9 = 2.1 \text{ people}$$

Software Engineering Economics defines each of the cost drivers, and defines the Effort Multiplier associated with each rating.

The Intermediate model also produces better results than the Basic model because the system can be divided into "components". SLOC values and Cost Drivers can be chosen for individual components, instead for the system as a whole. COCOMO can estimate the staffing, cost, and duration of each of the components and allows experimenting with different development strategies, to find the plan that best suits the needs and resources.

architectural Modal. The stages can be defined as in the table:

Early Design model	Post-Architecture Model
Well-understood project	Detailed project characterization
Moderate fidelity estimates	High-fidelity estimates
Well-understood requirement	Stable requirements baseline
Well-understood architecture	Stable architecture baseline

Both of them use following basic Equations

$$PM = 2.45 \times EAF \times (Size)^B$$

Where, EAF is effort adjustment factor. EAF is a product of seven Effort Multipliers in Early design model and seventeen in Post architecture model. Effort Multipliers are rated into very low, low, nominal, high, very high, and extra high categories. Numerical weights are assigned to them based on their effect on development effort. List of the effort multipliers is given in the table.

B is a scaling factor (1.01 to 1.26), representing diseconomies of scale.

$$B \text{ is given by } B = 1.01 + \sum W_i$$

$\sum W_i$  is sum of five components, which effect economy of scale. They are also rated and weights (0.00 to 0.05) are assigned to them as shown in the table below.

Table: Scale Factors of COCOMO II Early and Post-Architecture Models

	Scale Factors ( $W_i$ )	Very Low (0.00)	Low (0.01)	Nominal (0.02)	High (0.03)	Very High (0.04)	Extra High (0.05)
Precedentedness	PREC	Thoroughly unprecedented	Largely unprecedented	Somewhat unprecedented	generally familiar	Largely familiar	thoroughly familiar
Development Flexibility	FLEX	Rigorous	Occasional relaxation	Some Relaxation	general conformity	Some Conformity	general goals
Architecture/ Risk Resolution	RESL	Little (20%)	Some (40%)	often (60%)	generally (75%)	Mostly (90%)	full (100%)
Team Cohesion	TEAM	Very difficult interactions	Some difficult interactions	Basically co-operative interactions	largely cooperative	Highly Cooperative	seamless interactions
Process Maturity	PMAT	CMM Level 1 (lower half)	CMM Level 1 (Upper half)	CMM Level 2	CMM Level 3	CMM Level 4	CMM Level 5

Table: Early Design and Post-Architecture Effort Multipliers

Early Design Cost Driver	Counterpart Combined Post-Architecture Cost Drivers
Product Reliability and Complexity (RCPX)	Required Software Reliability (RELY), Data Base Size (DATA), Product Complexity (CPLX), Documentation match to life-cycle needs (DOCU)
Required Reuse (RUSE)	Required Reuse (RUSE)
Platform Difficulty (PDIF)	Execution Time Constraint (TIME), Main Storage Constraint (STOR), Platform Volatility (PVOL)
Personnel Capability (PERS)	Analyst Capability (ACAP), Programmer Capability (PCAP), Personnel Continuity (PCON)
Personnel Experience (PREX)	Applications Experience (AEXP), Platform Experience (PEXP), Language and Tool Experience (LTEX)
Facilities (FCIL)	Use of Software Tools (TOOL), Multisite Development (SITE)
Schedule (SCED)	Required Development Schedule (SCED)

Research on COCOMO II is still on and as COCOMO II evolves, it is supposed to have a more extensive schedule estimation model, reflecting the different classes of process model a project can use; the effects of reusable and COTS software; and the effects of applications composition capabilities.

1.5.3 COCOTS

There is an increase in emphasis to use Commercial Off The Shelf (COTS) software in software development to reduce overall development and maintenance costs. The name COCOTS stands for CONstructive COTS model. COTS software is commercially available as stand-alone products and that offers specific functionality needed by a larger system into which they might be incorporated. The two primary distinguishing characteristics of the COTS software are: (1) its source code is not available to the application developer, and (2) its evolution is not under the control of the application developer.

Software development using COTS will involve these four activities. (1) Candidate COTS component assessment, (2) COTS component tailoring, (3) the development and testing of any integration or "glue" code needed to plug a COTS component into a larger system, and (4) increased system level programming due to volatility in incorporated COTS components. COCOTS is a actually a a malgam o f four r elated s ub-models, e ach addressing individually four primary sources of COTS software integration effort.

Assessment is the process by which COTS components are selected for use in the larger system being developed. Tailoring refers to those activities that would have to be performed to prepare a particular COTS program for use, regardless of the system into which it is being incorporated, or even if operating as a stand-alone item. These are things such as initializing parameter values, specifying I/O screens or report formats, setting up security protocols, etc. Glue code development and testing refers to the new code external to the COTS component itself that must be written in order to plug the component into the larger system. This code by nature is unique to the particular context in which the COTS component is being used, and must not be confused with tailoring activity. Volatility in this context refers to the frequency with which new versions or updates of the COTS software being used in a larger system are released by the vendors over the course of the system's development and subsequent deployment.

Assessment: Assessment is the activity whereby COTS software products

are vetted and selected as viable components for integration into a larger system. In general terms, viable COTS candidates are determined based on the following:

- Functional Requirements: capability offered
- Performance Requirements: timing and sizing constraints
- Non-functional Requirements: cost/training/installation/maintenance/reliability

Assessment should be done in two phases. In the first phase, filtering is done by summarily eliminating those COTS products, which are unsuitable in the current context and in the second phase, final selection is done based on an assessment of each product in the light of certain product attributes. Depending on the project domain, more effort will be expended assessing a COTS product in terms of some attributes as opposed to others.

**Tailoring:** Tailoring is the activity whereby COTS software products are configured for use in a specific context. These are the normal things that would have to be done to a product no matter what system into which it is being integrated. These are things like parameter initialization; input/GUI screen and output report layout, security protocols set-up, etc.

The basic approach taken to capturing this effort in our model is illustrated in the chart below. The formulation of the tailoring submodel presumes that the difficulty or complexity of the tailoring work that is going to be required to get a given COTS product integrated into a system can be anticipated and even characterized by standardized rating criteria. The submodel then presumes that a parameterized value for a rough average tailoring effort per complexity rating can be determined for a given domain. The total tailoring effort for a project then becomes a function of the number of COTS products whose needed tailoring is estimated to be at a given rated level of complexity and the average tailoring effort at the given complexity rating per candidate, again as specified by the project's domain and summed over all tailoring complexity rating levels.

Dimensions of tailoring activity difficulty are:

- Parameter Specification
- Script Writing menu driven;
- I/O Report & GUI Screen Specification & Layout automated or standard templates used
- Security/Access Protocol Initialization & Set-up
- Availability of COTS Tailoring Tools

Based on the complexity levels on these five dimensions over all com-

plexity rating for a particular COTS product is assigned. Then based on the calibrated effort for that overall complexity level in the particular application domain effort required can be estimated. Summing such effort for all COTS product we can get tailoring estimate.

**Glue Code:** Glue code (sometimes called "glueware" or "binding" code) is the new code needed to get a COTS product integrated into a larger system. It can be code needed to connect a COTS component either to higher level system code, or to other COTS components also being used in the system.

The table below summarizes the thirteen linear effort multipliers and one nonlinear scale factor.

Personnel Drivers	COTS Integrator Experience with Product
	COTS Integrator Personnel Capability
	Integrator Experience with COTS Integration
	Integrator Personnel Continuity
COTS Component Drivers	COTS Product Maturity
	COTS Supplier Product Extension Willingness
	COTS Product Interface Complexity
	COTS Supplier Product Support
	COTS Supplier Provided Training
Application/System Drivers	Constraints on System/Subsystem Reliability
	Application Interface Complexity
	Constraints on COTS Technical Performance
	Application System Portability
Nonlinear Scale Factor	Application Architectural Engineering

**1.5.4 Function point**

Function Point Analysis was developed by Allan J. Albrecht in the mid 1970s. It was an attempt to overcome difficulties associated with lines of code as a measure of software size, and to assist in developing a mechanism to predict effort associated with software development.

Since Function Points measures systems from a functional perspective they are independent of technology. Regardless of language, development method, or hardware platform used, the number of function points for a system will remain constant. The only variable is the amount of effort needed to deliver a given set of function points; therefore, Function Point Analysis can be used to determine whether a tool, an environment, a language is more productive compared with others within an organization or among organizations. This is a critical point and one of the greatest values of Function Point Analysis.

Function Point Analysis can provide a mechanism to track and monitor scope creep. Function Point Counts at the end of requirements, analysis, design, code, testing and implementation can be compared. The function point count at the end of requirements and/or designs can be compared to function points actually delivered. If the project has grown, there has been scope creep. The amount of growth is an indication of how well requirements were gathered by and/or communicated to the project team. If the amount of growth of projects declines over time it is a natural assumption that communication with the user has improved.

**Characteristic of Quality Function Point Analysis**

Current application documentation should be utilized to complete a function point count. For example, screen formats, report layouts, listing of interfaces with other systems and between systems, logical and/or preliminary physical data models will all assist in Function Points Analysis.

The task of counting function points should be included as part of the overall project plan. That is, counting function points should be scheduled and planned. The first function point count should be developed to provide sizing used for estimating.

**The Five Major Components**

Since it is common for computer systems to interact with other computer systems, a boundary must be drawn around each system to be measured prior to classifying components. This boundary must be drawn according to the user's point of view. In short, the boundary indicates the border between the project or application being measured and the external applications or user domain. Once the border has been established, components can be classified, ranked and tallied.

**External Inputs (EI)** - is an elementary process in which data crosses the boundary from outside to inside. This data may come from a data input

screen or another application. The data may be used to maintain one or more internal logical files. The data can be either control information or business information. If the data is control information it does not have to update an internal logical file.

**External Outputs (EO)** - an elementary process in which derived data passes across the boundary from inside to outside. Additionally, an EO may update an ILF. The data creates reports or output files sent to other applications. These reports and files are created from one or more internal logical files and external interface file

**External Inquiry (EQ)** - an elementary process with both input and output components that result in data retrieval from one or more internal logical files and external interface files. The input process does not update any Internal Logical Files, and the output side does not contain derived data.

**Internal Logical Files (ILF's)** - a user identifiable group of logically related data that resides entirely within the applications boundary and is maintained through external inputs.

**External Interface Files (EIF's)** - a user identifiable group of logically related data that is used for reference purposes only. The data resides entirely outside the application and is maintained by another application. The external interface file is an internal logical file for another application.

After the components have been classified as one of the five major components (EI's, EO's, EQ's, ILF's or EIF's), a ranking of low, average or high is assigned. For transactions (EI's, EO's, EQ's) the ranking is based upon the number of files updated or referenced (FTR's) and the number of data element types (DET's). For both ILF's and EIF's files the ranking is based upon record element types (RET's) and data element types (DET's). A record element type is a user recognizable subgroup of data elements within an ILF or EIF. A data element type is a unique user recognizable nonrecursive field.

Each of the following tables assists in the ranking process. For example an EI that references or updates 2 File Types Referenced (FTR's) and has 7 data elements would be assigned a ranking of average and associated rating of 4. Where FTR's are the combined number of Internal Logical Files (ILF's) referenced or updated and External Interface Files referenced.

Complexity of External Inputs:

Files updated or referenced	Number of data element types		
	1-4	5-15	>15
0-1	Low	Low	Average
2	Low	Average	High
3 or more	Average	High	High

Files updated or referenced	Number of data element types		
	1-5	6-19	>19
0-1	Low	Low	Average
2-3	Low	Average	High
>3	Average	High	High

Like all components, EQ's are rated and scored. Basically, an EQ is rated (Low, Average or High) like an EO, but assigned a value like an EI. The rating is based upon the total number of unique (combined unique input and out sides) data elements (DET's) and the file types referenced (FTR's) (combined unique input and output sides). If the same FTR is used on both the input and output side, then it is counted only one time. If the same DET is used on both the input and output side, then it is only counted one time.

For both ILF's and EIF's the number of record element types and the number of data elements types are used to determine a ranking of low, average or high. A Record Element Type is a user recognizable subgroup of data elements within an ILF or EIF. A Data Element Type (DET) is a unique user recognizable, non-recursive field on an ILF or EIF. Complexity of Internal logical files and External Interface files:

Record element types	Number of data element types		
	1-19	20-50	>50
1	Low	Low	Average

2-5	Low	Average	High
>5	Average	High	High

Complexity weightage

Complexity Ratings	Weightage				
	EO	EQ	EI	ILF	EIF
Low	4	3	3	7	5
Average	5	4	4	10	7
High	7	6	6	15	10

The counts for each level of complexity for each type of component can be entered into a table such as the following one. Each count is multiplied by the numerical rating shown to determine the rated value. The rated values on each row are summed across the table, giving a total value for each type of component. These totals are then summed across the table, giving a total value for each type of component. These totals are then summoned down to arrive at the Total Number of Unadjusted Function Points.

Type of Component	Complexity of Components			Total
	Low	Average	High	
External Inputs	___ × 3 =	___ × 4 =	___ × 6 =	
External Outputs	___ × 4 =	___ × 5 =	___ × 7 =	
External Inquiries	___ × 3 =	___ × 4 =	___ × 6 =	
Internal Logical Files	___ × 7 =	___ × 10 =	___ × 15 =	
External Interface Files	___ × 5 =	___ × 7 =	___ × 10 =	
Total Number of unadjusted function points				
Multiplied value adjustment factor				
Total Adjusted Function Points				

The value adjustment factor (VAF) is based on 14 general system characteristics (GSC's) that rate the general functionality of the application being counted. Each characteristic has associated descriptions that help determine the degrees of influence of the characteristics. The degrees of influence range on a scale of zero to five, from no influence to strong influence. The IFPUG Counting Practices Manual provides detailed evaluation criteria for each of the GSC'S, the listing below is intended to provide an overview of each GSC.

**General System Characteristic Brief Description**

Data communications: How many communication links are there to aid in the transfer or exchange of information with the application or system?

- 1) Distributed data processing: How are distributed data and processing functions handled?
- 2) Performance: Was response time or throughput required by user?
- 3) Heavily used configuration: How heavily used is the current hardware platform where the application will be executed?
- 4) Transaction rate: How frequently is transactions executed daily, weekly, monthly, etc.?
- 5) On-Line data entry: What percentage of the information is entered on-line?
- 6) End-user efficiency: Was the application designed for end-user efficiency?
- 7) On-Line update: How many ILF's are updated by on-line transaction?
- 8) Complex processing: Does the application have extensive logical or mathematical processing?
- 9) Reusability: Was the application developed to meet one or many user's needs?
- 10) Installation ease: How difficult is conversion and installation?
- 11) Operational ease: How effective and/or automated are start-up, back up, and recovery procedures?
- 12) Multiple sites: Was the application specifically designed, developed, and supported to be installed at multiple sites for multiple organizations?
- 13) Facilitate change: Was the application specifically designed, developed, and supported to facilitate change?

Once all the 14 GSC's have been answered, they should be tabulated using the IFPUG Value Adjustment Equation (VAF) --

$$VAF = 0.65 + [(\sum C_i) / 100]$$

Where: C<sub>i</sub> = degree of influence for each General System Characteristic its value is from 0 to 5

i is from 1 to 14 representing each GSC.

The final Function Point Count is obtained by multiplying the VAF times the Unadjusted Function Point (UAF).

$$FP = UAF \times VAF$$

Benefits of Function Point Analysis:

- Function Points can be used to size software applications accurately. Sizing is an important component in determining productivity (outputs/inputs).
- Different people can count them at different times, to obtain the same measure within a reasonable margin of error.
- Function Points are easily understood by the non-technical user. This helps communicate sizing information to a user or customer.
- Function Points can be used to determine whether a tool, a language, an environment, is more productive when compared with others.

**1.6 Heuristic Approaches**

Expert judgement- Bottom Up, Top Down, Wideband Delphi Approach and Estimation by analogy

**1.6.1 Bottom Up**

The Project is divided into small modules with work break down structure. Direct estimation of effort and schedule is done for every module. Percentage of effort for each activity in the project plan is compared with standard percentages to check that allocation of effort in the activities is reasonable or not. If planned percentages deviate from the standard percentages significantly then re-estimation of some of the activities should be done. In this method minor details are taken care of but system level costs of integration, software quality assurance and configuration management may be overlooked.

**1.6.2 Top Down**

An overall cost estimates is derived from the global properties of the software product. This estimate will usually be based on previous projects and will include the costs of all function in a project, e.g. integration, software quality assurance and configuration management. This method has less detailed basis.

**1.6.3 Expert judgement: Wideband Delphi Approach**

In this section the wide-band Delphi approach for software estimation is discussed.

This procedure states that a group of experts is formed and asked not to discuss their work with one another. They give estimates of the product individually and anonymously. These estimates are collected and a coordinator assesses them. This assessment has to fall within an acceptable range, otherwise the coordinator asks for revised estimates, this process continues until a consensus is reached. Overall, this process usually works well, but may be expensive and time consuming. Key factors are:

- The Delphi Estimation Team shall consist of at least four estimators. The following steps outline the approach for using the Delphi technique:
- After sufficient familiarization with the task requirements, each estimator is given an estimation form as shown in the Figure.
- The estimators meet to discuss the task and any estimation issues.
- Each independently completes the estimation form.
- The estimates are given to the Project Coordinator, who tabulates the results and returns them to the estimators on an estimation form with the previous round's data filled in on the table.
- Only the estimator's personal estimates are identified, all others are anonymous.
- The estimators may meet to discuss the results and revise their estimates, as they feel appropriate.
- This cycle is repeated until the estimates converge to an acceptable range or until the completion of three cycles, whichever occurs first.

Documentation required in Delphi method:

- Delphi estimation process requires the use of an estimation form.
- The task number, the estimator's name, and the estimation date should be entered in the appropriate blocks on the form. Each requirement is identified by number from the Analysis Report.
- Estimators provide estimates for each requirement on a separate estimation form.
- Estimates are based on the source lines of code (LOC) and lines of documentation (LOD) to be inserted and deleted (Ins+Del) to im-

plement the various requirements. Estimates for each requirement are entered in the LOC and LOD columns on the form(s).

- Person-hour estimates are made for each of the process blocks/cells in which development and testing work is to be done (cells 601, 602, 603, 701, and 702). Estimates may be given in whole hours or hours and tenths of hours.
- For each subsequent estimation cycle, the same estimation form is used. Statistics from the previous estimation cycle are provided for

each requirement. These include the estimator's previous estimate, other highest and lowest estimates of the other estimators, and the average estimate.

- Additional space is provided for re-estimates of any or all of the estimator's previous estimates. If any re-estimates are made, the estimator may explain the rationale for each re-estimate at the bottom of the form.

Delphi Estimation Form								
Name:					Date:			
Task#:					Requirement#:			
Estimation Round (circle selection):      1      2      3								
	LOC	Ins+Del	LOD Ins+Del	601 (Hours)	602 (Hours)	603 (Hours)	701 (Hours)	702 (Hours)
Highest Estimate								
Average Estimate								
Lowest Estimate								
Your Previous Estimate								
Your New Estimate								
Estimator's Comments:								
Note: First round estimates will have no data in the table above. Estimators complete only the bottom row of the table in any round.								

**1.6.4 Estimation by analogy**

The majority of research work carried out in the software cost estimation field has been devoted to algorithmic models. However, by an overwhelming majority, expert judgement is the most commonly used estimation method. A Dutch study carried out by [HEMSTRA] revealed that 62% of estimators/organisations use this intuition technique and a study carried out by [VIGDER & KARK] also confirmed the techniques popularity. In its crudest form the expert judgement method involves consultation with one or more local experts who are knowledgeable about the development environment or application domain to estimate the effort required to complete a software project.

The method relies heavily on the experience of their knowledge in similar development environments and historically maintained databases on completed projects and the accuracy of the past projects. However, the study carried out by [VIGDER & KARK] indicated that in general estimators did not refer to previous projects as it was too difficult to access or the expert could not see how the information would help in the accuracy of the estimate. The study claimed that the majority of estimators tended to use their memories of previous projects. If more than one expert is used, the weighted average of their estimates are taken. There are obvious risks with this method. As the project may have some unique features which could take longer than anticipated. The weighted average is also very much dependent on the competence of the estimator. However, a particular strength of using an expert is that they can raise unique strengths and weaknesses of the local organisational characteristics.

Despite widespread use, the method seems to have received a rather poor reputation and is often regarded as subjective and unstructured making it vulnerable against more structured methods. A more structured than an expert judgement is the Wideband Delphi Approach as discussed in the previous section.

**1.7 Practice at Infosys**

At Infosys, estimation is generally done after requirement's analysis. At this stage details of the project are known and requirements are well understood. The project work is first divided into major programs (or units). Each program is classified as simple, medium, or complex, and the build effort for each program is directly estimated based on past experience of similar projects. The effort for other stages of the project is estimated using the effort distribution on similar projects. Guidelines for classifying into simple, medium, complex (S/M/C) are provided at company Intranet. The main data source for estimation is the process database and the process-capability baselines. The procedure for estimation has the following steps:

- Identify programs in the system and classify them as simple, medium, or complex (S/M/C). Definitions given for this purpose or notes on similar project entries in the process database are used for classification.
- If a project-specific baseline exists, get the average build effort for S/M/C programs from the baseline.
- If a project-specific baseline does not exist, use project type, technology, language, and other attributes to look for similar projects in the process database. Use data from these projects to define the build effort of S/M/C programs.
- If no similar project exists in process database and no project specific baseline exists, (i.e. this project is in an area/technology in which we have no experience), then use the average build effort for S/M/C programs from the general process-capability baseline.
- Use project specific factors to further refine the build effort for S/M/C.
- Use the effort distribution data in the capability baseline or similar projects in process database, to estimate effort for other tasks and the total effort.
- Refine the total effort estimates based on project specific factors. Take special care to factor in project specific factors like expected volatility of requirements, clarity of requirements, degree of willingness of the customer to work with you to generate clarity of the requirements etc. Document the estimates & assumptions made in estimation.

This method of classifying programs into a few categories and using an average build effort for each category is followed for overall estimation. In general, however, where each program unit must be scheduled and assigned to some programmer, characteristics of a unit may be taken into account to give more or less time than the average for its development. This approach uses a combination of bottom up approach, a model for classifying S/M/C, past project experience and project specific factors.

**1.8 Comparison of Methods**

Three issues are important from project leader's point of view:

- Which cost estimation model to use
- Whether to measure software size in source lines of code or function points
- What constitutes a good estimates

None is better than the others in all aspects. Strengths and weaknesses of others are complementary:

- Algorithmic vs. expert judgment
- Top-down vs. bottom-up

Method	Strengths	Weaknesses
Algorithmic Models	<ul style="list-style-type: none"> <li>▪ Objective, repeatable, analyzable formula</li> <li>▪ Efficient, good for sensitivity analysis</li> </ul>	<ul style="list-style-type: none"> <li>▪ Subjective inputs</li> <li>▪ Assessment of exceptional circumstances</li> </ul>

	<ul style="list-style-type: none"> <li>▪ Objectivity calibrated to experience</li> </ul>	<ul style="list-style-type: none"> <li>▪ Calibrated to past not future</li> </ul>
Expert Judgement	<ul style="list-style-type: none"> <li>▪ Assessment of representativeness, interactions, exceptional circumstances</li> </ul>	<ul style="list-style-type: none"> <li>▪ No better than participants</li> <li>▪ Biases, incomplete recall</li> </ul>
Analogy	<ul style="list-style-type: none"> <li>▪ Based on representative experience</li> </ul>	<ul style="list-style-type: none"> <li>▪ Representativeness of experience</li> </ul>
Top-down	<ul style="list-style-type: none"> <li>▪ System level focus</li> <li>▪ Efficient</li> </ul>	<ul style="list-style-type: none"> <li>▪ Less detailed basis</li> <li>▪ Less stable</li> </ul>
Bottom-up	<ul style="list-style-type: none"> <li>▪ More detailed basis</li> <li>▪ More stable</li> <li>▪ Fosters individual commitment</li> </ul>	<ul style="list-style-type: none"> <li>▪ May overlook system level costs</li> <li>▪ Requires more efforts</li> </ul>

The COCOMO and the function points, both are formula based but lot of subjective judgement is involved in judging the complexity of the attributes, and selecting a suitable multiplying factor. In COCOMO, the effort multiplying factors have to be calibrated based on the project implementation experience. This calibration is not easy because even for a completed project it is difficult to uniquely rank seventeen parameters from very low to extra high.

It is important to use a combination of techniques, and to compare and iterate the estimates obtained from each. The particular combination to choose will depend on the cost estimation objectives (for example, more top-down for rough early estimates and more bottom-up for detailed planning estimates). Following is an effective combination:

- A top down estimate using the judgement of more than one expert, using analogy estimation where a comparable previous project is available.
- Bottom-up estimation using an algorithmic model, with inputs and component-level estimates provided by future performers.
- Comparison and iteration of both estimates.

**About the size of the project.** SLOC is a more definitive indicator of size. People feel more comfortable with it. It can be counted without ambiguity after the product is developed. It is equally difficult to estimate in the beginning. But language advances and the use of components, automatic source code generation, and object orientation has made SLOC an ambiguous measure. The primary advantage of using function points is that this method is independent of technology and is therefore a much better primitive unit for comparisons among projects and organizations. It can be counted very early in the project life cycle. The main disadvantage is that it is more abstract and measurements are not easily derived directly from the product. Within organization the variation in function point counts about the mean is 30% [16]. This is attributed to the analyst's assessment of the complexity of the user function type, interpretation of the specification, value judgements, perception of the object boundaries, etc.

**1.9 Conclusion**

We have interviewed key persons responsible for estimations at Infosys. The estimation is done after requirements analysis phase using bottom up approach. Some times top down and Delphi technique is also utilized. At other places also parametric models are rarely used. Requirements analysis phase is major source of error in the estimation process. Given the stable requirements people are confident of estimating within 10%. Among the parametric methods only function points is sometimes used. After studying available practices for estimation we can conclude that good software cost estimate should have the following attributes:

- It is conceived and supported by the project manager, architecture is accepted by all stakeholders as ambitious but realizable.
- It is based on well-defined software cost model with a credible basis.
- It is based on a database of relevant project experience that includes similar processes, technologies, similar environment, development team, and test team accountable for performing the work.
- It, similar quality requirements, and similar people.
- It is defined in enough detail so that its key risk areas are understood and the probability of success is objectively assessed.

It implies that detailed history should be kept for:

- Accountability
- Basis for model calibration and modifications in the guidelines
- Basis for future cost estimations

**1.10 References**

Some relevant sites:

- I. [http://sunset.usc.edu/research/COCOMOII/cocomo\\_main.html#downloads](http://sunset.usc.edu/research/COCOMOII/cocomo_main.html#downloads)
- II. <http://www.jsc.nasa.gov/bu2/COCOMO.html>
- III. <http://ourworld.compuserve.com/homepages/softcomp/fpfaq.htm>
- IV. <http://sepo.nosc.mil/estimation.html>
- V. <http://www.jsc.nasa.gov/bu2/PCEHHTML/pceh.htm>

Softw. Eng., 9(6), pp639-648, 1983.

- [2] Banker, R.D. and C.F. Kemerer, 'Scale economies in new software development', IEEE Trans. on Softw. Eng., 15(10), 199-204, 1989
- [3] Boehm, B.W., Software Engineering Economics. Prentice-Hall: Englewood Cliffs, NJ, 1981.
- [4] Cowderoy, A.J.C. and J.O Jenkins, 'Cost estimation by analogy as a good management practice', in Proc. Software Engineering 88, ed. Pyle, I.C., Liverpool: IEE/BCS, pp80-84, 1988
- [5] DeMarco, T., Controlling Software Projects. Management, measurement and estimation. Yourdon Press: NY, 1982.
- [6] Fenton, N.E., 'Software Metrics: a rigorous approach'. Chapman & Hall, 1991.
- [7] Fenton, N.E. and S. Pfleeger, 'Software Metrics: a rigorous and practical approach'. Thomson Computer Press, 1997.
- [8] Heemstra, F.J., 'Software cost estimation', Information & Softw. Technol., 34(10), pp627-639, 1992.
- [9] Hughes, R.T., 'Expert judgement as an estimating method', Information & Softw. Technol., 38(2), pp67-75, 1996.
- [10] Jack R. and M. Mannion, 'Improving the software cost estimation process', Software Quality Management. 1995 1 pp245-56.
- [11] Jalote Pankaj, 'CMM in practice: processes for executing software projects at Infosys' Addison-Wesley 1999
- [12] Kemerer, C.F., 'An empirical validation of software cost estimation models', CACM, 36(2), 1993.
- [13] Kitchenham, B.A., 'Empirical studies of assumptions that underlie software cost estimation', Information and Softw. Technol., 34(4), 211-18, 1992.
- [14] Londeix, B., Cost Estimation for Software Development. Addison-Wesley: Workingham, 1987.
- [15] Londeix, B., 'Aspects of estimation practice in software development', in Proc. Software Engineering 88, ed. Pyle, I.C., Liverpool: IEE/BCS, pp 75-79, 1988
- [16] Low, G.C and D.R. Jeffery, 'Function points in the estimation and evaluation of the software process', IEEE Trans. on Softw. Eng., 16(1), 64-71, 1990.
- [17] Low, G.C. and D.R. Jeffery, 'Calibrating estimation tools for software development', Softw. Eng. J., 5(4), pp215-221, 1990.
- [18] McDermid, J.A., Software Engineer's Reference Book, Butterworth-Heinemann: Oxford, UK, 1991.
- [19] Pengelly, A., 'Performance of effort estimating techniques in current development environments', Softw. Eng. J.I, September 1995, pp162-169
- [20] Putnam, L.H., 'A general empirical solution to the macro software sizing and estimating problem', IEEE Trans. on Softw. Eng., 4(4), 345-61, 1978.
- [21] Roetzheim W.H. and Beasley R.A., 'Software project cost & schedule estimating: best practices', Prentice Hall, Inc, 1998.
- [22] Shepperd, M.J., Foundations of Software Measurement. Prentice Hall: Hemel Hempstead, UK, 1995.
- [23] Shepperd, M.J., C. Schofield and B.A. Kitchenham. 'Effort estimation using analogy', in Proc. 18th Intl. Conf. on Softw. Eng. Berlin: IEEE Computer Press, 1996.
- [24] Symons, C.R., Software Sizing and Estimating. Mk II FPA, John Wiley: Chichester, 1991.
- [25] Vigder, M.R. and A.W. Kark, 'Software Cost Estimation and Control, February 1994, Report available from the link
- [26] Walston, C.E. and C.P. Felix, 'A method of programming measurement and estimation', IBM Syst. J., 16(1), 54-73, 1977.

[1] Albrecht, A.J. and J.R. Gaffney, 'Software function, source lines of code, and development effort prediction: a software science validation', IEEE Trans. on