Software Estimation

A guide for PRACTICIONERS

Software Estimation Group on Linkedin

Compiled by
Santosh Ramachandran

11/7/13

Table of Contents

1.0 Introduction

In extensive research of thousands of software development projects spanning 18 years, it was found that more projects are doomed due to poor cost and schedule estimates than ever succumb to technical, political, or development team problems. Yet, few companies and individuals really understand that software estimating is a science and not just an art. It really is possible to accurately and consistently estimate development costs and schedules for a wide range of projects. This guideline aims at providing the skills needed to accurately predict the resources required to be successful, calculate realistic delivery times, and consistently deliver projects on time and on budget.

1.1 Why Bother Estimating?

According to the Standish Report (*The Standish Group International, Inc., Estimating: Art or Science?*), the odds of a Fortune 500 software development project coming in on time and on budget are only one in four. One of the primary reasons cited for this failure and its associated costs is that estimates of project cost and schedule are often made at different times by different people using different methods. This almost always produces either "lucky" or "lousy" estimates. These are more often than not "lousy" and occasionally "catastrophic." (*The Standish Group International, Inc., Estimating: Art or Science?*)

Inaccurate estimates often result in long hours, poor quality work and shattered customer or employer expectations.

Despite decades of research done on software engineering in the past several decades, software cost estimation still remains to be a very difficult task, and software projects still go over budget and over time. There are many reasons for this, most of which involved the large complexity of software projects, and they include difficulties in requirement management, lack of historical data, new technology, and others. Lots of research has been done in this area, and there are many methods, tools, and techniques available to perform software cost estimation. These include expert judgment, estimation by analogy, top-down method, bottom-up method, and algorithmic methods.

In the end, there is still no one tool, method, or technique that works well in all situations. Therefore, in an effort to make accurate cost estimates, a number of tools, methods, and techniques should be used.

Estimation is not exact science but a statistical technique. Accuracy of estimation depends upon the exactness of requirements, number of estimations, stage at which estimation is done and quality of project estimation sheet. Estimation is a very important part of software development. Many important decisions are made from software estimates alone. However, software estimation is a difficult task, and oftentimes the results of software estimates are very inaccurate.

The following are a few common examples of how estimates can be put to use within an organization, as taken from the Vidger and Kark study (Vidger/Kark 1994).

- **Planning and Budgeting –** Strategic decisions are made based on cost estimates. These types of decisions include whether a project is worth proceeding with, how much should be bid on a project, how should the budget be divided up between separate aspects of the project, etc.

- **Project Management –** Project managers use cost estimates to plan, monitor, and control the implementation of a project.
- **Communication Among Team Members –** A work breakdown structure (WBS) is usually derived from a cost estimate. Team members use this work breakdown structure to understand their own roles within a project as well understand the roles of others.

1.2 Objective & Scope

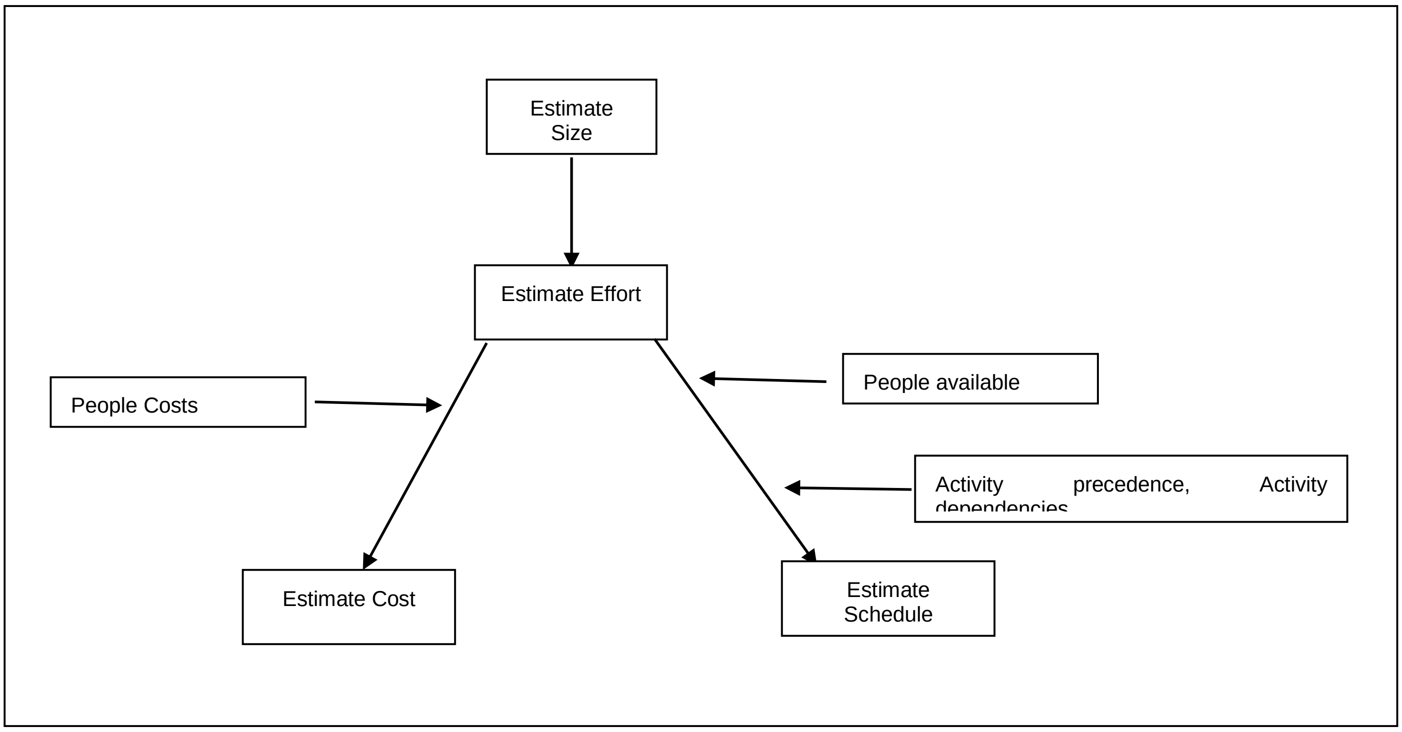
The estimation techniques described in this document can be used for software development projects. Techniques for estimating software size, effort, schedule and cost are included in this document.

2.0 Software Estimation

For the purposes of our discussion, we can define software estimates as a combination of estimating:

- **Size** – The Number of Lines of Code that need to be developed for implementing the functionality of the Software.
- **Effort –** The amount of effort required to complete a project, usually measured in units such as person-months or person-years
- **Schedule –** The amount of time that is needed to complete the project.
- **Cost –** The amount of money that needs to be spent for the development of the software.

The following diagram depicts the steps involved in the estimation process:



2.1 Estimation – Inputs and Outputs

Now that we have defined the dimensions of software estimation, we can give an overview of how they relate to one another. Here, we will view software estimation as a function of inputs and outputs. In the classical view of software estimation (Figure 1), software requirements are the primary inputs to the software cost estimation process and thus act as a basis for the software estimates. The estimate that is derived from the requirements alone is adjusted according to other cost drivers. Cost drivers can be anything that would affect the final estimate, including experience of developers, complexity, risk assessments, types of tools being used, etc. From the software estimation process, we derive estimates for size, effort, schedule, and cost as defined earlier.

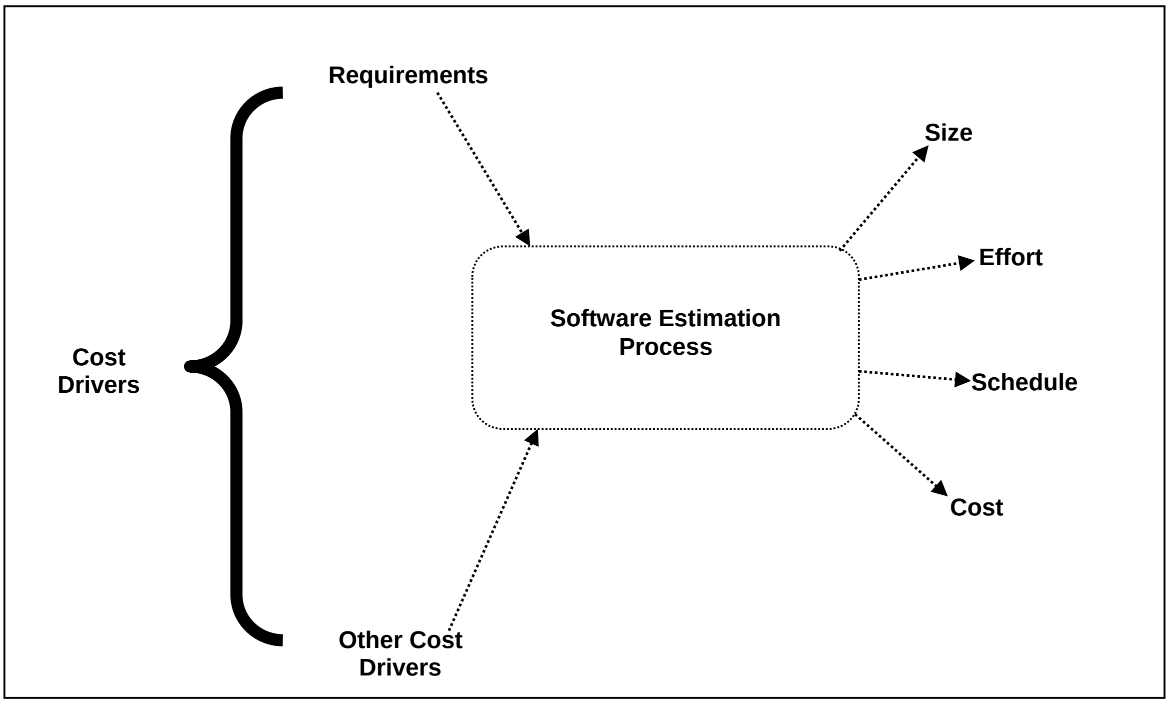


Figure 2 - Classical View of Software Estimation Process [Reference Vidger/Kark]

However, the classical model given above is rather simplistic. As the figure given below shows (Figure 2), a lot more factors are involved for both input and outputs to the software cost estimation process. Not only do we have cost drivers being the inputs to the software cost estimation process, we also have to take into account financial and other constraints, as well as other inputs such as risk factors, a fuzzy architecture, and the software process.

Also note that the requirements that are given for the software estimation process are vague at this point. As for the output of the software cost estimation process, not only do we have effort, duration, and loading, we also get less vague (and sometimes modified) requirements, a contingency plan, a tentative work breakdown structure, and a less fuzzy architecture. Of course, all of this depends on the type of project being undertaken and the type of organization delivering the project. Therefore, more or less factors might be involved in the software estimation process.

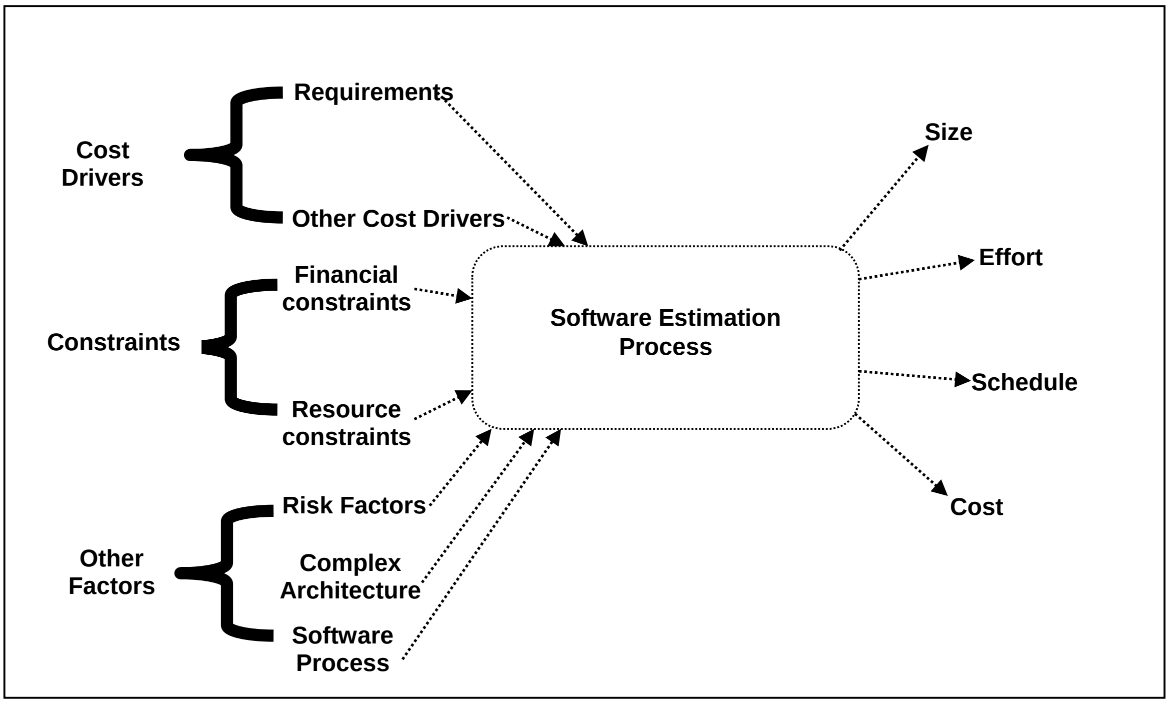


Figure 3 - Complex View of Software Estimation Process [Reference Vidger/Kark]

2.2 General Estimation Process

The way software estimation takes place varies widely from project to project and organization to organization. In addition, a wide variety of tools and methods are in existence that aid in software estimation. The following is a general estimation process, which does not make any assumptions on what kind of project is being done, what sort of organization is working on the project, and what sort of tool and methods are being used.

1. **Determine Objectives** – Determine exactly who needs what data for what purpose(s) that is to result from the software estimate. This will drive the level of detail that is to be given from the software estimate.
2. **Gather Data –** When gathering data for determining a estimates, focus should be given to 'Hard' data, such as well-defined requirements and available resources.
3. **Analyze Data Using A Variety Of Methods –** Not all methods are the same, and thus some methods are better in some situations than others. Therefore, using several methods will often compensate for weaknesses in certain methods.
4. **Re-estimate Throughout The Project –** This is done as a part of effectively monitoring the project, to refine and make changes as necessary as the situation surrounding the software development evolves.
5. **Compare Actual with Estimates –** At the end of a project, a comparison between expected and actual result should be done for the purposes of collecting historical data that can be used later to estimate the future projects.

2.3 Estimation Accuracy

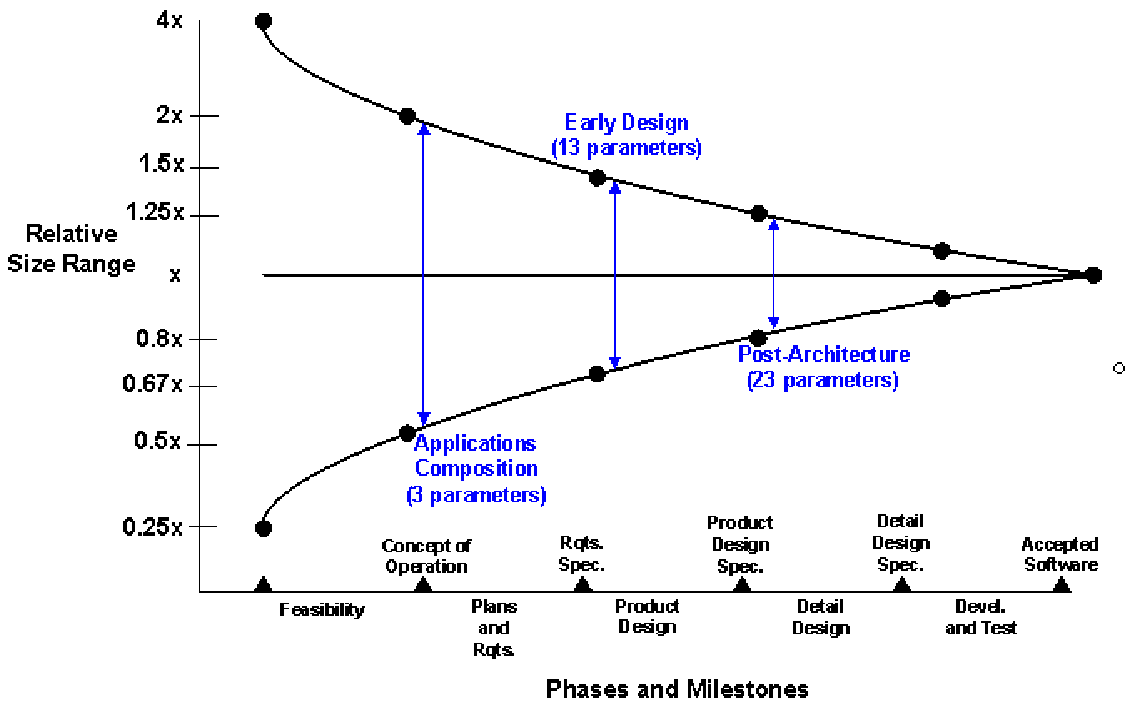


Figure 4 – COCOMO II Model Stages (Reference B. Boehm, C. Abts, A.W. Brown, S. Chulani, B. Clark, E. Horowitz, R. Madachy, D. Reifer, and B. Steece, Software Cost Estimation with COCOMO II, Prentice Hall, 2000.)

Figure 3 depicts the expected range of the Software Estimates through the various phases of the project. Estimation at the initial stages of the project is often very inaccurate and estimates narrow down towards actual as we progress through the various phases of the project and more information is available on the project. It is very important to perform estimation iteratively through the phases of the project to arrive at accurate estimates for the project.

2.4 Determining Estimation Accuracy

Estimation models, tools, and techniques are evaluated by simply comparing expected effort with actual effort. The following are a few formulas that allow us to do this.

Magnitude of Relative Error (MRE)

MRE = ABS((E – A) / A) x 100%

(where E = expected effort and A = actual effort)

This gives us a percentage of how far our estimates were from the actual.

Mean Magnitude of Relative Error (MMRE)

MMRE = 100/n * SUM(MREi) where i = 1….n

When several separate estimates are done on a single project, this formula gives us the average of how far our various estimates were from the actual.

2.5 Issues and Problems

Software estimation is certainly not an exact science. While every developer and project manager has their own ideas on what are the specific issues and problems surrounding software estimation, the following list are common reasons given by experts in the field.

- **Inaccurate Estimation Models, Methods, and Tools –** Some models, methods and tools work better than others. In addition, certain models, methods, and tools are better for certain situations and worse in others.
- **Ineffective Management of Requirements –** This is a large reason of why estimates are never accurate, at least in the early stages of the project. First, very vague or even incorrect requirements are often given at the beginning, thus making the estimation difficult from the start. Secondly, requirements are often changed or added throughout the project, thus making earlier estimates no longer accurate.
- **Duration, Size, and Number of People Involved –** In general, the longer the duration or the larger the size of the project or the greater the number of people involved, the more difficult it is to make accurate estimates.
- **Ineffective or Non-existent Monitoring of Project –** Due to ineffective management, estimates that could have been accurate are not because project management fails to ensure the project follows the original plan made at the project planning phase, thus not meeting expected budget and deadlines.
- **New Technology –** New technology is always being introduced, but since developer experience and historical data surrounding new technology are rare, then doing estimates on using new technology is very difficult.
- **Mixed Technology –** This is a major cause of cost inaccuracies, for the same reasons new technologies are. If developer experience and historical data are rare regarding the mixing of the technologies in question, then doing estimates on it are very difficult.
- **Inexperienced Project Managers and Estimators –** It is often argued that experienced project mangers are the best source of accurate cost estimates. Conversely, inexperienced project managers can be the best source of inaccurate estimates.
- **Lack of Application Domain Expertise –** A development team with less domain expertise is likely to take longer to complete a project because of the overhead of understanding a particular domain to do development work.

- **Software Processes and Process Maturity –** A mature, well-defined process within an organization is more likely to deliver on time and on budget than an organization that does not have a mature, well-defined process. In addition, some software processes are better than others are.
- **Lack of Historical Data –** This relates to new and mixed technology, but also applies to existing technology as well. Historical data is often the best way to make reliable estimates, but if they do not exist or nobody bothers to use them, then cost estimation is much more difficult.
- **Business / Legal / Monetary Issues –** This is another very large issue, since oftentimes a cost estimate is distorted due to business, legal, or monetary issues. For example, an overly optimistic cost estimate is used for a bid proposal to increase the chances of being awarded a contract.

2.6 What is needed To Make Reliable Estimates

Again, every developer and project manager has their own ideas on how software estimates can be improved. The following is a list of the more common solutions.

- **A Combination of Models, Methods, and Tools** – Not all tools and techniques are alike and give similar results. Using a variety of them can compensate for inaccuracies given by some models, tools, and techniques.
- **Gathering/Improving of Historical Data –** Using historical data is one of the best ways to produce accurate estimates. Not only should good use of historical data be made, but a good effort to collect historical data from current projects should take place as well.
- **Well-defined and Well-controlled Software Development Processes –** A well-defined and well-controlled process can ensure that projects are developed on time and on budget. In addition, with a well-defined and well-controlled process in place, estimates are made easier as well.
- **Better Managing of Requirements** – If accurate requirements are gathered from the start (which accurate initial estimates are derived from), and new or changed requirements are effectively dealt with by use of a good change request process, then costs can be controlled and thus estimates will be more accurate.
- **Experienced Project Managers, Estimators, and Team Members –** Of course, the best source of historical data are the experienced project mangers, estimators, and team members themselves! They should be more involved in the estimation process.

3.0 Methods and Tools of Software Estimation

There are several kinds of software estimation models and techniques, such as algorithmic methods, estimating by analogy, expert judgment method, price to win method, top-down method, and bottom-up method, etc. But it is important that no one method is necessarily better or worse than the other, in fact, their strengths and weaknesses are often complimentary to each other. Understanding the strengths and weaknesses of different techniques is very important for selecting the right estimating technique and accurately estimating projects.

3.1 Expert Judgment Method

Expert judgment techniques involve consulting with software estimation experts to use their experience and understanding of the proposed project to arrive at an estimate of its size, effort, schedule and cost. In general, this method is achieved by group consensus technique, such as Delphi technique.

The typical steps of expert judgment method as following:

1) Coordinator presents each expert with a specification and an estimation form.
2) Coordinator calls a group meeting in which the experts discuss estimation issues with the coordinator and each other.
3) Experts fill out forms anonymously
4) Coordinator prepares and distributes a summary of the estimation on an iteration form.
5) Coordinator calls a group meeting, specially focusing on having the experts discuss points where their estimates varied widely.
6) Experts fill out forms, again anonymously, and steps 4 and 6 are iterated for as many rounds as appropriate.

Advantages:

- The experts can factor in differences between past project experience and requirements of the proposed project. It is useful in the absence of quantified, empirical data.
- The experts can factor in project impacts caused by new technologies, architectures, applications and languages involved in the future project and can also factor in exceptional personnel characteristics and interactions, etc.

Disadvantages:

- It is difficult to be quantified with this method.
- It is hard to document the factors used by the experts or experts-group.
- It is subjective because expert may be some biased, optimistic, and pessimistic, even though they have been decreased by the group consensus.

3.2 Estimating by analogy

Estimating by analogy means comparing the proposed project to previously completed similar project where the project development information is known. Estimating by analogy is relatively straightforward compare to other methods.

The typical steps of using estimating by analogy as following:

1) Characterizing the proposed project.
2) Selecting the most similar completed projects whose characteristics have been stored in the historical data base.
3) Deriving the estimate for the proposed project from the most similar completed projects by analogy.

Advantages:

- The estimation is based on actual project characteristic data.
- The estimator's past experience and knowledge can be used which is not easy to be quantified.
- The differences between the completed and the proposed project can be identified and impacts estimated.

Disadvantages:

- It is difficult to ensure the degree of similarity between previous projects and the new one.

3.3 Top-Down Estimating Method

Top-down estimating method is also called Macro Model. Using it, estimation is derived from the global properties of the software project, and then the project is partitioned into various low-level components. The leading method using this approach is Putnam model. This method is better to be used in early stages of estimation if only global properties are known and there is lack of detailed information.

Advantages:

- It focuses on system-level activities such as integration, documentation, configuration management, etc., many of which may be ignored in other estimating methods and it will not miss the cost of system-level functions.
- It requires minimal project detail, and it is usually faster, easier to implement.

Disadvantages:

- It often does not identify difficult low-level problems that are likely to escalate costs and sometime tends to overlook low-level components.
- It provides no detailed basis for justifying decisions or estimates.

3.4 Bottom-up Estimating Method

Bottom-up estimating method, the effort for each software components is estimated and then combines the results to arrive at an estimated cost of overall project.

It focuses to constructing the estimate of a system from the knowledge accumulated about the small software components and their interactions.

Advantages:

- It is more detailed because it estimates all the components
- It permits the software group to handle an estimate in an almost traditional fashion and to handle estimate components for which the group has a feel.
- It is more stable because the estimation errors in the various components have a chance to balance out.

Disadvantages:

- It may overlook many of the system-level effort associated with software development in the early phases of the project.
- It may be inaccurate because the necessary information may not available in the early phases of the project

3.5 Algorithmic Method

Algorithmic method is designed to provide some mathematical equations to perform software estimation. It is also called parametric method. All the mathematical equations are based on research and historical data and use inputs such as SLOC, FP, and other cost drivers such as language, design methodology, skill-levels, risk assessments, etc.

Advantages:

- It can generate repeatable estimates.
- It is easy to modify input data with this method.
- It is easy to refine and customize formulas.
- It is objectively calibrated to previous experience.

Disadvantages:

- It is unable to deal with exceptional conditions.
- Some experience and factors can not be quantified.
- Sometimes algorithms may be proprietary.

3.6 Using Estimation Methods

From the above comparison, we know no one method is necessarily better or worse than the other, in fact, their strengths and weaknesses are often complimentary to each other. According to the experience, it is recommended that a combination of models and analogy or expert judgment estimation methods is useful to get reliable, accurate cost estimation for software development.

When we select software estimation method, there are some suggestions can be considered:

1. Do not depend on a single effort or schedule estimate.
2. Use several estimating techniques or estimation models, compare the results, and determine the reasons for any large variations.
3. Document the assumptions made when making the estimates.
4. Monitor the project to detect when assumptions that turn out to be wrong jeopardize the accuracy of the estimate.
5. Improve software process: An effective software process can be used to increase accuracy in estimation in a number of ways.
6. Maintaining and use the estimation historical database.

3.7 Estimation Tools

The first automated software estimation tools were developed independently by researchers in major corporations and military groups in the 1960s. Commercial software estimation tools began to be marketed in the 1970s. By 2002, about 50 commercial software estimation tools were marketed in the United States and another 25 in Europe. Although standard projects can now be estimated with fairly good accuracy, there are always new technologies that require improvements in estimating tools.

The major features of commercial software estimation tools include the following basic abilities:

- Sizing logic for specifications, source code, and test cases.
- Phase-level, activity-level, and task-level estimation.
- Support for both function point metrics and the older lines-of-code (LOC) metrics.
- Support for specialized metrics such as object-oriented metrics.
- Support for backfiring or conversion between LOC and function points.
- Support for software reusability of various artifacts.
- Support for traditional languages such as COBOL and FORTRAN.
- Support for modern languages such as Java and Visual Basic.
- Quality and reliability estimation.

For selecting tools to accurately predict development and sustaining engineering estimates for distributed systems and systems reuse, the criteria of assessment of commercial tools is as following:

- Does the tool estimate life-cycle effort?
- Can the tool interface with external applications (e.g., Excel)?
- Does the tool consider COTS?
- Does the tool consider reuse?
- Is the tool easy to use?
- How does the tool handle new technology/technology enhancement?

4.0 Software Estimation Techniques

Estimation for a Software projects involves estimating its size, effort, schedule and cost.

1. **Size:** The basis for any estimate is the size of a task; hence the process of software estimation starts with the estimating size of the software application or product.

2. **Effort:** The size estimates are then used to arrive at effort estimates using historical data or using effort estimation models.

3. **Schedule:** The effort estimates are converted to schedule estimates on the basis of:
   a. ***Delivery dates agreed with client.***
   b. ***Number of resources***
   c. ***Resource availability***
   d. ***Customer Priorities***
   e. ***Development Priorities***
   f. ***Dependencies***
   g. ***Critical Path***
   h. ***Number of working days***
   i. ***Resource leaves.***

4. **Cost:** Cost estimates for the software are arrived at using the effort estimates, schedule estimates and the billing method agreed with the customer.

4.1 Estimating Size

Estimating the amount of work to complete a project requires the estimator to know two things: the size of the project and the development staff’s skill set and experience level. The more detailed the knowledge of those two factors, the more accurate the estimate will be. Like everything else in the project, estimates of the work effort should be developed iteratively and be revisited regularly.
Size can be estimated using the following size estimation techniques:

4.1.1 Function Point Analysis

Function Point Analysis has been proven as a reliable method for measuring the size of computer software. In addition to measuring output, Function Point Analysis is extremely useful in estimating projects, managing change of scope, measuring productivity, and communicating functional requirements.

There have been many misconceptions regarding the appropriateness of Function Point Analysis in valuating emerging environments such as real time embedded code and Object Oriented programming. Since function points express the resulting work-product in terms of functionality as seen from the user's perspective, the tools and technologies used to deliver it are independent.

The following provides an introduction to Function Point Analysis and is followed by further discussion of potential benefits.

4.1.1.1 Introduction to Function Point Analysis

One of the initial design criteria for function points was to provide a mechanism that both software developers and users could utilize to define functional requirements. It was determined that the best way to gain an understanding of the users' needs was to approach their problem from the perspective of how they view the results an automated system produces.

Therefore, one of the primary goals of Function Point Analysis is to evaluate a system's capabilities from a user's point of view. To achieve this goal, the analysis is based upon the various ways users interact with computerized systems.

From a user's perspective a system assists them in doing their job by providing five (5) basic functions. Two of these address the data requirements of an end user and are referred to as Data Functions. The remaining three addresses the user's need to access data and are referred to as Transactional Functions.

The Five Components of Function Points

1. Data Functions
   a. • Internal Logical Files
   b. • External Interface Files
2. Transactional Functions
   a. • External Inputs
   b. • External Outputs
   c. • External Inquiries

4.1.1.2 Internal Logical Files (ILF)

ILF - The first data function allows users to utilize data they are responsible for maintaining. For example, a pilot may enter navigational data through a display in the cockpit prior to departure. The data is stored in a file for use and can be modified during the mission. Therefore the pilot is responsible for maintaining the file that contains the navigational information.

Logical groupings of data in a system, maintained by an end user, are referred
to as Internal Logical Files (ILF).

4.1.1.3 External Interface Files (EIF)

EIF - The second Data Function a system provides an end user is also related to logical groupings of data. In this case the user is not responsible for maintaining the data. The data resides in another system and is maintained by another user or system. The user of the system being counted requires this data for reference purposes only. For example, it may be necessary for a pilot to reference position data from a satellite or groundbased facility during flight. The pilot does not have the responsibility for updating data at these sites but must reference it during the flight. Groupings of data from another system that are used only for reference purposes are defined as External Interface Files (EIF).

The remaining functions address the user's capability to access the data contained in ILFs and EIFs. This capability includes maintaining, inquiring and outputting of data. These are referred to as Transactional Functions.

4.1.1.4 External Input (EI)

EI - The first Transactional Function allows a user to maintain Internal Logical Files (ILFs) through the ability to add, change and delete the data. For example, a pilot can add, change and delete navigational information prior to and during the mission. In this case the pilot is utilizing a transaction referred to as an External Input (EI). An External Input gives the user the capability to maintain the data in ILF's through adding, changing and deleting its contents.

External Output - The next Transactional Function gives the user the ability to produce outputs. For example a pilot has the ability to separately display ground speed, true air speed and calibrated air speed. The results displayed are derived using data that is maintained and data that is referenced. In function point terminology the resulting display is called an External Output (EO).

4.1.1.5 External Inquiries (EQ)

EI - The final capability provided to users through a computerized system addresses the requirement to select and display specific data from files. To accomplish this a user inputs selection information that is used to retrieve data that meets the specific criteria. In this situation there is no manipulation of the data. It is a direct retrieval of information contained on the files. For example if a pilot displays terrain clearance data that was previously set, the resulting output is the direct retrieval of stored information. These transactions are referred to as External Inquiries (EQ).

In addition to the five functional components described above there are two adjustment factors that need to be considered in Function Point Analysis.

4.1.1.6 Functional Complexity

Functional Complexity - The first adjustment factor considers the Functional Complexity for each unique function. Functional Complexity is determined based on the combination of data groupings and data elements of a particular function. The number of data elements and unique groupings are counted and compared to a complexity matrix that will rate the function as low, average or high complexity. Each of the five functional components (ILF, EIF, EI, EO and EQ) has its own unique complexity matrix.

All of the functional components are analyzed in this way and added together to derive an Unadjusted Function Point count.

4.1.1.7 *Value Adjustment Factor (VAF)*

VAF **-** The Unadjusted Function Point count is multiplied by the second adjustment factor called the Value Adjustment Factor. This factor considers the system's technical and operational characteristics and is calculated by answering 14 questions. The factors are:

4.1.1.7.1 Data Communications

The data and control information used in the application are sent or received over communication facilities.

4.1.1.7.2 Distributed Data Processing

Distributed data or processing functions are a characteristic of the application within the application boundary.

4.1.1.7.3 Performance

Application performance objectives, stated or approved by the user, in either response or throughput, influence (or will influence) the design, development, installation and support of the application.

4.1.1.7.4 Heavily Used Configuration

A heavily used operational configuration, requiring special design considerations, is a characteristic of the application.

4.1.1.7.5 Transaction Rate

The transaction rate is high and influences the design, development, installation and support.

4.1.1.7.6 On-line Data Entry

On-line data entry and control information functions are provided in the application.

4.1.1.7.7 End -User Efficiency

The on-line functions provided emphasize a design for end-user efficiency.

4.1.1.7.8 On-line Update

The application provides on-line update for the internal logical files.

4.1.1.7.9 Complex Processing

Complex processing is a characteristic of the application.

4.1.1.7.10 Reusability

The application and the code in the application have been specifically designed, developed and supported to be usable in other applications.

4.1.1.7.11 Installation Ease

Conversion and installation ease are characteristics of the application. A conversion and installation plan and/or conversion tools were provided and tested during the system test phase.

4.1.1.7.12 Operational Ease

Operational ease is a characteristic of the application. Effective start-up, backup and recovery procedures were provided and tested during the system test phase.

4.1.1.7.13 Multiple Sites

The application has been specifically designed, developed and supported to be installed at multiple sites for multiple organizations.

4.1.1.7.14 Facilitate Change

The application has been specifically designed, developed and supported to facilitate change.

Each of these factors is scored based on their influence on the system being counted. The resulting score will increase or decrease the Unadjusted Function Point count by 35%. This calculation provides us with the Adjusted Function Point count.

4.1.1.8 Steps for calculating function points

1. Determine type of function point count.
2. Determine the application boundary.
3. Identify and rate data function types to determine their contribution to the unadjusted function point count.
    a. External Interface Files
    b. Internal Logical Files
4. Identify and rate transactional function types to determine their contribution to the unadjusted function point count. Transactional function types can be classified into:
    a. External Inputs
    b. External Outputs

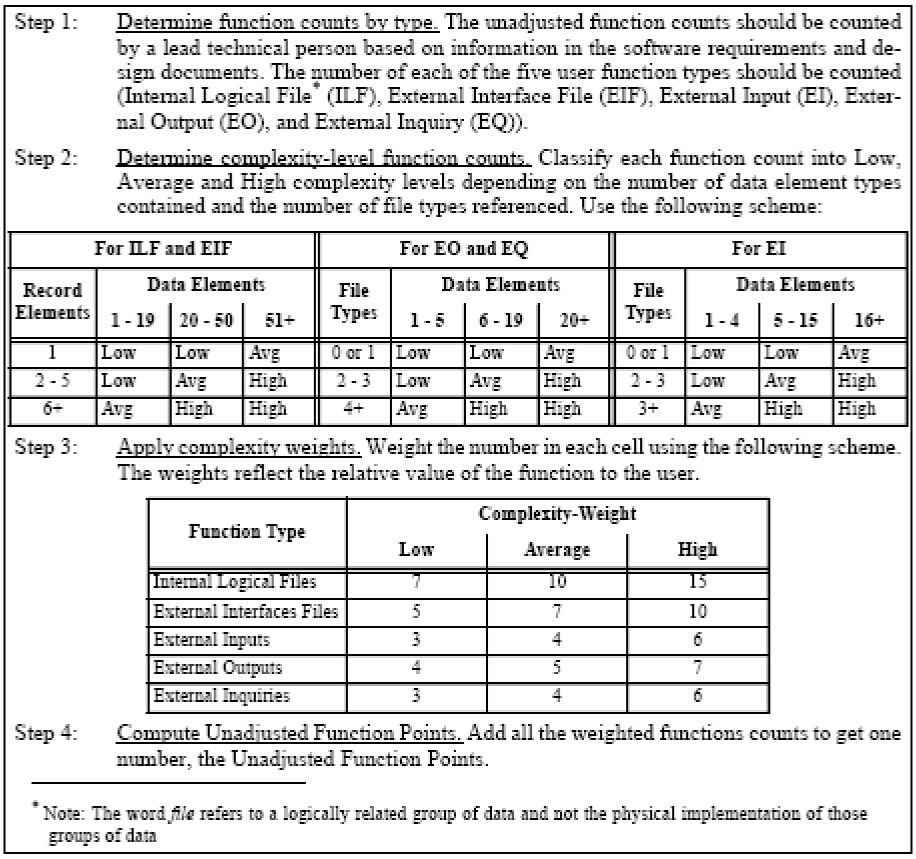
c. External Queries

Table 3: Function points as per function type and complexity

Function Type	Simple	Average	Complex
External Inputs	3	4	6
External Outputs	4	5	7
External Inquiries	3	4	6
Internal Logical Files	7	10	15
Ext Interfaces	5	7	10
(Algorithms)	7	10	15

5. Determine the value adjustment factor (VAF).
6. Calculate the adjusted function point count using the formulae:
   a. Function Points = UFP * (0.65 + 0.01 * VAF)

4.1.1.9 Function Point Quick Reference



Step 1: Determine function counts by type. The unadjusted function counts should be counted by a lead technical person based on information in the software requirements and design documents. The number of each of the five user function types should be counted (Internal Logical File* (ILF), External Interface File (EIF), External Input (EI), External Output (EO), and External Inquiry (EQ)).

Step 2: Determine complexity-level function counts. Classify each function count into Low, Average and High complexity levels depending on the number of data element types contained and the number of file types referenced. Use the following scheme:

For ILF and EIF				For EO and EQ				For EI			
Record Elements	Data Elements			File Types	Data Elements			File Types	Data Elements		
	1 - 19	20 - 50	51+		1 - 5	6 - 19	20+		1 - 4	5 - 15	16+
1	Low	Low	Avg	0 or 1	Low	Low	Avg	0 or 1	Low	Low	Avg
2 - 5	Low	Avg	High	2 - 3	Low	Avg	High	2 - 3	Low	Avg	High
6+	Avg	High	High	4+	Avg	High	High	3+	Avg	High	High

Step 3: Apply complexity weights. Weight the number in each cell using the following scheme. The weights reflect the relative value of the function to the user.

Function Type	Complexity-Weight		
	Low	Average	High
Internal Logical Files	7	10	15
External Interfaces Files	5	7	10
External Inputs	3	4	6
External Outputs	4	5	7
External Inquiries	3	4	6

Step 4: Compute Unadjusted Function Points. Add all the weighted functions counts to get one number, the Unadjusted Function Points.

* Note: The word *file* refers to a logically related group of data and not the physical implementation of those groups of data

*4.1.1.10 **Benefits of Function Point Analysis***

Organizations that adopt Function Point Analysis as a software metric realize many benefits including: improved project estimating; understanding project and maintenance productivity; managing changing project requirements; and gathering user requirements. Each of these is discussed below. Estimating software projects is as much an art as a science. While there are several environmental factors that need to be considered in estimating projects, two key data points are essential. The first is the size of the deliverable. The second addresses how much of the deliverable can be produced within a defined period of time. Size can be derived from Function Points, as described above. The second requirement for estimating is determining how long it takes to produce a function point. This delivery rate can be calculated based on past project performance or by using industry benchmarks. The delivery rate is expressed in function points per hour (FP/Hr) and can be applied to similar proposed projects to estimate effort (i.e. Project Hours = estimated project function points FP/Hr).

Productivity measurement is a natural output of Function Points Analysis. Since function points are technology independent they can be used as a vehicle to compare productivity across dissimilar tools and platforms. More importantly, they can be used to establish a productivity rate (i.e. FP/Hr) for a specific tool set and platform. Once productivity rates are established they can be used for project estimating as described above and tracked over time to determine the impact continuous process improvement initiatives have on productivity.

In addition to delivery productivity, function points can be used to evaluate the support requirements for maintaining systems. In this analysis, productivity is determined by calculating the number of function points one individual can support for a given system in a year (i.e. FP/FTE year). When compared with other systems, these rates help to identify which systems require the most support. The resulting analysis helps an organization develop a maintenance and replacement strategy for those systems that have high maintenance requirements.

*4.1.1.11 **Managing Change of Scope for an in-process project is another key benefit of Function Point Analysis.***

Once a project has been approved and the function point count has been established, it becomes a relatively easy task to identify, track and communicate new and changing requirements. As requests come in from users for new displays or capabilities, function point counts are developed and applied against the rate. This result is then used to determine the impact on budget and effort. The user and the project team can then determine the importance of the request against its impact on budget and schedule. At the conclusion of the project the final function point count can be evaluated against the initial estimate to determine the effectiveness of requirements gathering techniques. This analysis helps to identify opportunities to improve the requirements definition process. Communicating Functional Requirements was the original objective behind the development of function points. Since it avoids technical terminology and focuses on user requirements it is an excellent vehicle to communicate with users. The techniques can be used to direct customer interviews and document the results of Joint Application Design (JAD) sessions. The resulting documentation provides a framework that describes user and technical requirements.

In conclusion, Function Point Analysis has proven to be an accurate technique for sizing, documenting and communicating a system's capabilities. It has been successfully used to evaluate the functionality of real-time and embedded code systems, such as robot based warehouses and avionics, as well as traditional data processing. As computing environments become increasingly complex, it is proving to be a valuable tool that accurately reflects the systems we deliver and maintain.

4.1.2 Use Case Points

The Use Case Points methodology was developed in 1993 by Gustav Karner as an extension of the Function Points Analysis method used in procedural programming. The philosophy of this method is that project size can be estimated based on the functionality seen by the user. Use Case Points (UCP) are still relatively new, and few documented projects have used them from the beginning of the design process. This means the relative accuracy has not been empirically shown.

a. Identify all Actors and Use-cases

b. UCP requires the use cases to be fairly detailed, and more importantly, consistently detailed.

c. Categorize actors as Simple, Average or Complex.
    i. A simple actor represents an interaction through an API, like a call to a system operation.
    ii. An average actor is one accessed via some standardized protocol, such as a web service.
    iii. A complex actor represents a person interacting with the system.
    iv. The weighing factor for the different actors is 1, 2, and 3 for simple, average, and complex respectively.

Table 3: Unadjusted Use Case points for Actors

Actor Type	Weighing Factor
Simple	1

Average	2
Complex	3

d. Categorize the Use-Case based on the information available

   i. Assumptions can be made for when adequate information is not available.

   ii. When classifying Use-Case the following scenarios are possible (At-least the list of all use-cases that need to be developed must be available):

      1. **No Information Available:** In case no information is available to classify the uses cases:
         a. Identify Information on any one of the following and then follow the relevant section for estimation:
            i. Number of Business Use Cases.
            ii. Number of Use Case.
            iii. Number of Use Case Transactions.
            iv. Number of Analysis Classes.

      2. **Information on Business Use Cases available**: In case information is available on the number of Business Use Cases that need to be developed:

         a. The number of Business processes in the business use case can be identified.
         b. An assumption can be made on the number of use case per business process, this can be used for estimating the Use Case Points and the effort required to develop the software.

         c. Use the following table to classify the Business Use Cases as Simple, Medium or Complex.

Table 3: Unadjusted Use Case points based on number of Business processes

Business Use Case Type	**Number of Business Processes**	**Assumed Use Case per Business Process**	**Weighing Factor Per Use Case**
Simple	<=3	2	5
Average	4 to 7	4	10
Complex	>= 7	7	15

3. **Information on Number of Use Cases available:** In case information is available on the number of Use Cases

   a. No Information is available on the details of the Use Case:
      i. Use expert judgment to classify the Use-Cases as Simple, Medium or Complex.

   b. Information is available on the Number of Transactions of the Use Case:

      i. Use the following tables for allocating weights to the Use-Cases.

Table 2: Unadjusted Use Case points based on number of use case transctions

Use Case Type	Number of Transactions	Weighing Factor
Simple	<= 3	5
Average	4 to 7	10
Complex	>= 7	15

   c. Information is available on the Number of Analysis Classes of the Use Case:

      i. Use the following tables for allocating weights to the Use-Cases.

Table 3: Unadjusted Use Case points based on number of analysis classes

Use Case Type	Number of Analysis Classes	Weighing Factor
Simple	<= 5	5
Average	6 to 10	10
Complex	>10	15

e. Calculate the unadjusted Use Case Points (UUCP) using the formula:

Unadjusted Use Case points = (# of simple UCs * 5) + (# of Average UCs * 10) + (# of Complex UCs * 15) + (# of simple actors * 1) + (# of average actors * 2) + (# of complex actors * 3)

f. Calculate Technical Complexity factor (TCF).

g. Compute the technical complexity factor (TCF) by reviewing the factors in the following table and rating each factor from 0 to 5 (0=irrelevant, 5=essential).

Notes: Suggestions for improving the estimation technique

- *TCF or EF Weights can be adjusted for different domains. I.e. these weights are a generic set of weights for web based projects, but the weights would most probably differ for different domains like (securities, mobile computing, banking, finance, etc)*

Table 4: Use Case points Technical Factors

#	Technical Factor	Weight
1	Distributed System	2
2	Response or throughput performance objectives	1
3	End-user efficiency (online)	1
4	Complex internal processing	1
5	Code must be reusable	1
6	Easy to install	0.5
7	Easy to use	0.5
8	Portable	2
9	Easy to change	1
10	Concurrent	1
11	Includes special security features	1
12	Provides direct access for third parties	1
13	Special user training facilities required	1

h. Use the following formulas to calculate the TCF:
   i. Multiply the rating by the weight, and sum:

   ii. TFactor = Sum (Weight * Rating)
   iii. TCF = 0.6 + (0.01 * TFactor)

i. Calculate Environmental factor (EF).
j. Compute the environmental factor (EF) by rating the factors in the following table from 0 to 5 (0=No experience to 5=Expert).

Table 4: Use Case points Environment Factors

#	Environmental Factor	Weight
1	Familiar with Internal process	1.5
2	Application experience	0.5
3	Object-oriented experience	1
4	Lead analyst capability	0.5
5	Motivation	1
6	Stable requirements	2
7	Part-time workers	-1
8	Difficult programming language	-1

k. Use the following formulas to calculate the ECF
   i. Mutliply the rating by the weight and sum:
   ii. EFactor = Sum(Weight * Rating)
   iii. EF = 1.4 + (-0.03 * EFactor)

l. Calculate the Use Case Points using the formula:

   Use Case Points = UUCP * TCF * EF.

4.2 Estimating Effort

Effort required for developing software products can be estimated by using one of the following techniques:

1. COCOMO II
2. Use Case based effort estimation:

4.2.1 COCOMO II

COCOMO II is well matched effort estimation technique for custom build-to-specification software projects; COCOMO II is useful for a wide collection of techniques and technologies. COCOMO II provides up-to-date support for business software, object-oriented software, software developed using spiral or evolutionary development models, and software developed using commercial off-the-shelf application composition utilities (Boehm et al, 1997). COCOMO II includes the Application Composition model (for early prototyping efforts) and the more detailed Early Design and Post-Architecture models (for subsequent portions of the lifecycle).

The rationale for providing this tailorable mix of models rests on following premises:

First, unlike the initial COCOMO situation in the late 1970's, in which there was a single, preferred software life cycle model, current and future software projects will be tailoring their processes to their particular process drivers. These process drivers include COTS or reusable software availability; degree of understanding of architectures and requirements; market window or other schedule constraints; size; and required reliability (see [Boehm 1989, pp. 436-37] for an example of such tailoring guidelines).

Second, the granularity of the software cost estimation model used needs to be consistent with the granularity of the information available to support software cost estimation. In the early stages of a software project, very little may be known about the size of the product to be developed, the nature of the target platform, the nature of the personnel to be involved in the project, or the detailed specifics of the process to be used.

Third, given the situation in premises 1 and 2, COCOMO II enables projects to furnish coarse-grained cost driver information in the early project stages, and increasingly fine-grained information in later stages.

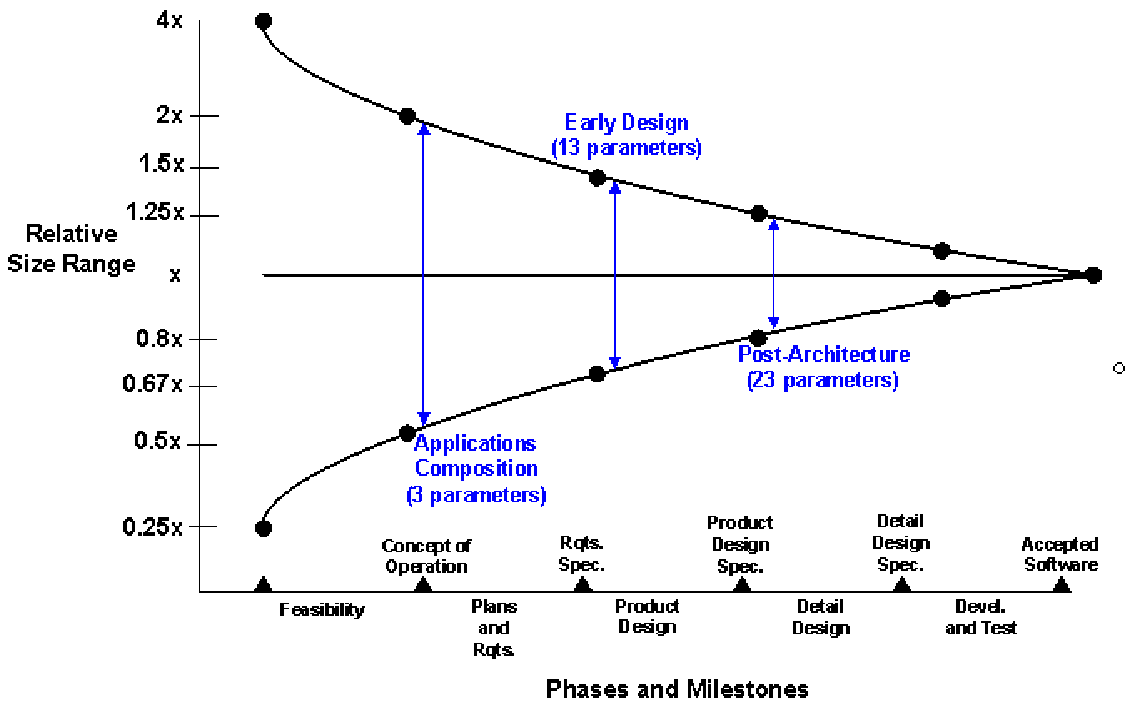


Figure 5 – COCOMO II Model Stages (Reference B. Boehm, C. Abts, A.W. Brown, S. Chulani, B. Clark, E. Horowitz, R. Madachy, D. Reifer, and B. Steece, Software Cost Estimation with COCOMO II, Prentice Hall, 2000.)

COCOMO II Model

This COCOMO II Software cost estimation model uses sets of multiplicative and exponential cost drivers to adjust for project, target platform, personnel, and product characteristics. The set of multiplicative cost drivers are called Effort Multipliers (EM). The nominal weight assigned to each EM is 1.0. If a rating level has a detrimental effect on effort, then its corresponding multiplier is above 1.0. Conversely, if the rating level reduces the effort then the corresponding multiplier is less than 1.0. The exponential cost drivers, called Scale Factors and represented by the B exponent, account for the relative economies or diseconomies of scale encountered as a software project increases its size. This set is described in the next section. A constant, A, is used to capture the linear effects on effort with projects of increasing size. The estimated effort for a given size project is expressed in person months (PM), see (2) The following sections discuss the new COCOMO 2.0 cost drivers.

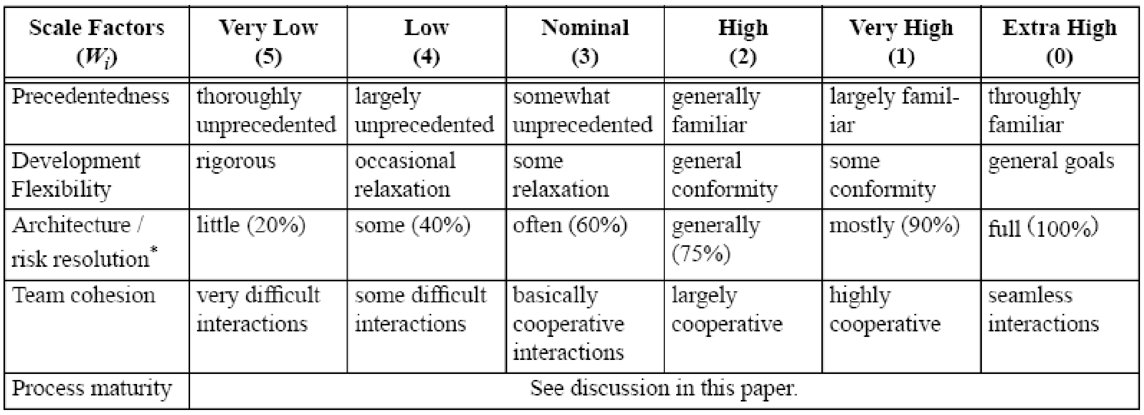
$$PM_{estimated} = \left(\prod_i EM_i\right) \times A \times (Size)^B$$

EXPONENT SCALE FACTORS

Table 3 provides the rating levels for the COCOMO 2.0 exponent scale factors. A project's numerical ratings Wi are summed across all of the factors, and used to determine a scale exponent B via the following formula:

$$B = 1.01 + 0.01\Sigma W_i$$

Thus, a 100 KSLOC project with Extra High (0) ratings for all factors will have Wi = 0, B = 1.01, and a relative effort E = 1001.01= 105 PM. A project with Very Low (5) ratings for all factors will haveWi= 25, B = 1.26, and a relative effort E = 331 PM. This represents a large variation, but the increase involved in a one-unit change in one of the factors is only about 4.7%. Thus, this approach avoids the 40% swings involved in choosing a development mode for a 100 KSLOC product in the original COCOMO.



Scale Factors (W_i)	Very Low (5)	Low (4)	Nominal (3)	High (2)	Very High (1)	Extra High (0)
Precedentedness	thoroughly unprecedented	largely unprecedented	somewhat unprecedented	generally familiar	largely familiar	throughly familiar
Development Flexibility	rigorous	occasional relaxation	some relaxation	general conformity	some conformity	general goals
Architecture / risk resolution*	little (20%)	some (40%)	often (60%)	generally (75%)	mostly (90%)	full (100%)
Team cohesion	very difficult interactions	some difficult interactions	basically cooperative interactions	largely cooperative	highly cooperative	seamless interactions
Process maturity	See discussion in this paper.					

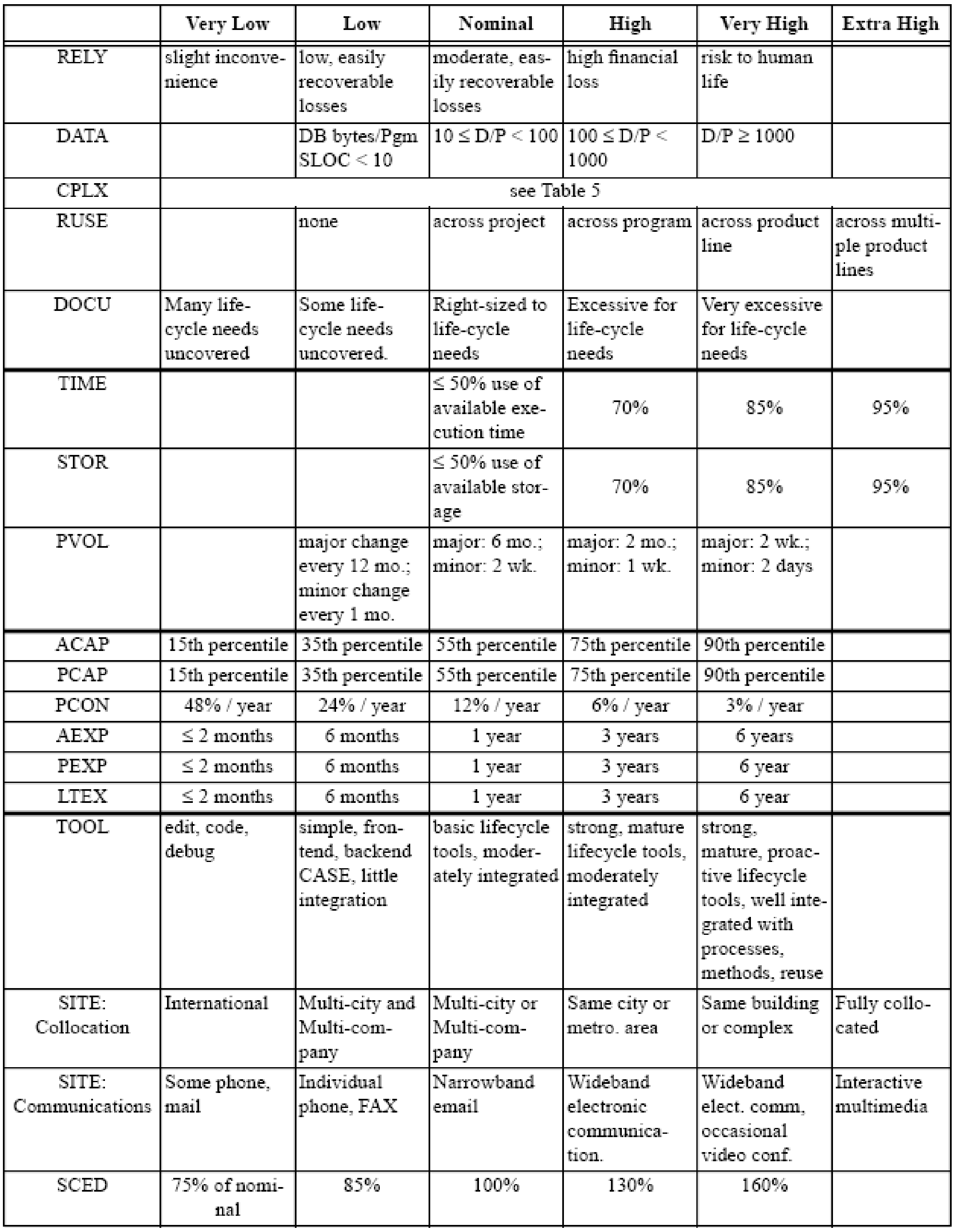
If B < 1.0, the project exhibits economies of scale. If the product's size is doubled, the project effort is less than doubled. The project's productivity increases as the product size is increased. Some project economies of scale can be achieved via project-specific tools (e.g., simulations, testbeds), but in general these are difficult to achieve. For small projects, fixed startup costs such as tailoring and setup of standards and administrative reports are a source of economies of scale.

If B = 1.0, the economies and diseconomies of scale are in balance. This linear model is often used for cost estimation of small projects. It is used for the COCOMO 2.0 Applications Composition model.

If B > 1.0, the project exhibits diseconomies of scale. This is generally due to two main factors: growth of interpersonal communications overhead and growth of large-system integration overhead. Larger projects will have more personnel, and thus more interpersonal communications paths consuming overhead. Integrating a small product as part of a larger product requires not only the effort to develop the small product, but also the additional overhead effort to design, maintain, integrate, and test its interfaces with the remainder of the product.

MULTIPLICATIVE COST DRIVERS

There are 17 cost drivers used in the COCOMO 2.0 Post-Architecture model to adjust the model to reflect the software product under development. They are grouped into four categories: product, platform, personnel, and project.



	Very Low	Low	Nominal	High	Very High	Extra High
RELY	slight inconvenience	low, easily recoverable losses	moderate, easily recoverable losses	high financial loss	risk to human life	
DATA		DB bytes/Pgm SLOC < 10	10 ≤ D/P < 100	100 ≤ D/P < 1000	D/P ≥ 1000	
CPLX	see Table 5					
RUSE		none	across project	across program	across product line	across multiple product lines
DOCU	Many life-cycle needs uncovered	Some life-cycle needs uncovered.	Right-sized to life-cycle needs	Excessive for life-cycle needs	Very excessive for life-cycle needs	
TIME			≤ 50% use of available execution time	70%	85%	95%
STOR			≤ 50% use of available storage	70%	85%	95%
PVOL		major change every 12 mo.; minor change every 1 mo.	major: 6 mo.; minor: 2 wk.	major: 2 mo.; minor: 1 wk.	major: 2 wk.; minor: 2 days	
ACAP	15th percentile	35th percentile	55th percentile	75th percentile	90th percentile	
PCAP	15th percentile	35th percentile	55th percentile	75th percentile	90th percentile	
PCON	48% / year	24% / year	12% / year	6% / year	3% / year	
AEXP	≤ 2 months	6 months	1 year	3 years	6 years	
PEXP	≤ 2 months	6 months	1 year	3 years	6 year	
LTEX	≤ 2 months	6 months	1 year	3 years	6 year	
TOOL	edit, code, debug	simple, frontend, backend CASE, little integration	basic lifecycle tools, moderately integrated	strong, mature lifecycle tools, moderately integrated	strong, mature, proactive lifecycle tools, well integrated with processes, methods, reuse	
SITE: Collocation	International	Multi-city and Multi-company	Multi-city or Multi-company	Same city or metro. area	Same building or complex	Fully collocated
SITE: Communications	Some phone, mail	Individual phone, FAX	Narrowband email	Wideband electronic communication.	Wideband elect. comm, occasional video conf.	Interactive multimedia
SCED	75% of nominal	85%	100%	130%	160%	

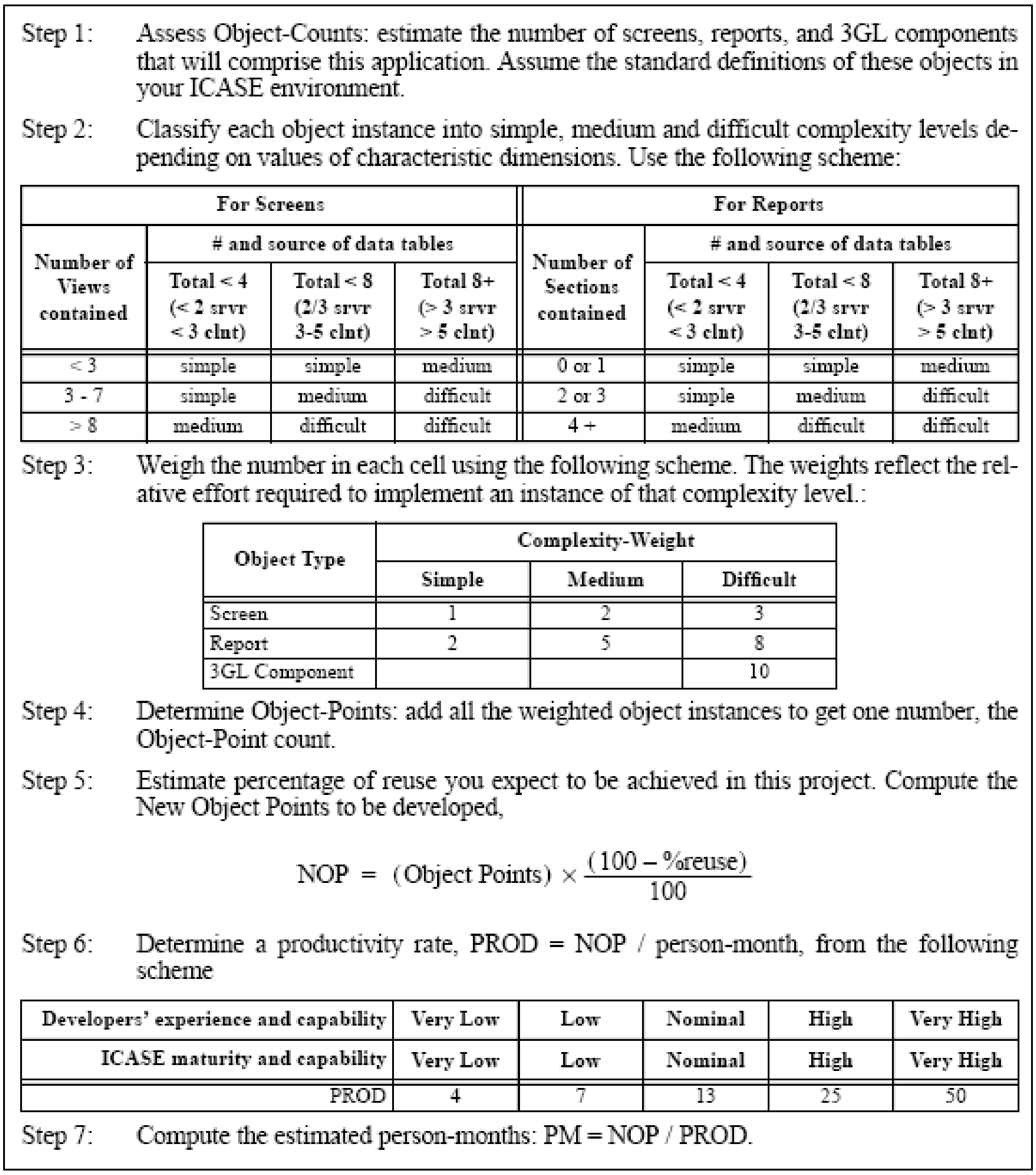
Development Schedule Estimates

The COCOMO II schedule equation predicts the number of months required to complete your software project. The duration of a project is based on the effort predicted by the effort equation:

$$TDEV = \left[2.5 \times (\overline{PM})^{(0.33 + 0.2 \times (B - 1.01))}\right] \times \frac{SCEDPercentage}{100}$$

4.2.1.1 The Application Composition Model

The Application Composition model is used in prototyping to resolve potential high-risk issues such as user interfaces, software/system interaction, performance, or technology maturity. Object points are used for sizing rather than the traditional LOC metric. An initial size measure is determined by counting the number of screens, reports, and third-generation components that will be used in the application.



Step 1: Assess Object-Counts: estimate the number of screens, reports, and 3GL components that will comprise this application. Assume the standard definitions of these objects in your ICASE environment.

Step 2: Classify each object instance into simple, medium and difficult complexity levels depending on values of characteristic dimensions. Use the following scheme:

For Screens				For Reports			
Number of Views contained	# and source of data tables			Number of Sections contained	# and source of data tables		
	Total < 4 (< 2 srvr < 3 clnt)	Total < 8 (2/3 srvr 3-5 clnt)	Total 8+ (> 3 srvr > 5 clnt)		Total < 4 (< 2 srvr < 3 clnt)	Total < 8 (2/3 srvr 3-5 clnt)	Total 8+ (> 3 srvr > 5 clnt)
< 3	simple	simple	medium	0 or 1	simple	simple	medium
3 - 7	simple	medium	difficult	2 or 3	simple	medium	difficult
> 8	medium	difficult	difficult	4 +	medium	difficult	difficult

Step 3: Weigh the number in each cell using the following scheme. The weights reflect the relative effort required to implement an instance of that complexity level.:

Object Type	Complexity-Weight		
	Simple	Medium	Difficult
Screen	1	2	3
Report	2	5	8
3GL Component			10

Step 4: Determine Object-Points: add all the weighted object instances to get one number, the Object-Point count.

Step 5: Estimate percentage of reuse you expect to be achieved in this project. Compute the New Object Points to be developed,

$$\text{NOP} = (\text{Object Points}) \times \frac{(100 - \%\text{reuse})}{100}$$

Step 6: Determine a productivity rate, PROD = NOP / person-month, from the following scheme

Developers’ experience and capability	Very Low	Low	Nominal	High	Very High
ICASE maturity and capability	Very Low	Low	Nominal	High	Very High
PROD	4	7	13	25	50

Step 7: Compute the estimated person-months: PM = NOP / PROD.

4.2.1.2 The Early Design Model

The Early Design model uses KSLOC for size. Unadjusted function points are converted to the equivalent SLOC and then to KSLOC. The application of project scale factors is the same for Early Design and the Post-Architecture models. In the Early Design model a reduced set of cost drivers are used. The Early Design cost drivers are obtained by combining the Post-Architecture model cost drivers from Table II-9. Whenever an assessment of a cost driver is between the rating levels always around to the Nominal rating, e.g. if a cost driver rating is between Very Low and Low, then select Low.

Table II-9: Early Design and Post-Architecture Effort Multipliers

Early Design Cost Driver	Counterpart Combined Post-Architecture Cost Drivers
RCPX	RELY, DATA, CPLX, DOCU
RUSE	RUSE
PDIF	TIME, STOR, PVOL
PERS	ACAP, PCAP, PCON
PREX	AEXP, PEXP, LTEX
FCIL	TOOL, SITE
SCED	SCED

4.2.1.3 Post Architecture

These are the 17 effort multipliers used in COCOMO II Post-Architecture model to adjust the nominal effort, Person Months, to reflect the software product under development. They are grouped into four categories: product, platform, personnel, and project.

5.0 References

- THE COCOMO 2.0 SOFTWARE COST ESTIMATION MODEL - Barry Boehm, Bradford Clark, Ellis Horowitz, Chris Westland USC Center for Software Engineering Ray, Madachy USC Center for Software Engineering and Litton Data Systems, Richard SelbyUC Irvine and Amadeus Software Research.
- The COCOMO II Model Definition Manaual
- Estimation Manual, Scandent Solutions.
- IFPUG - Function Point Training Material - http://softwaremetrics.com/freemanual.htm
- Software Cost Estimation By Tong (Leo) Chen, Harprit Grewal, Jerrall Prakash - Department of Computer Science, University Of Calgary
- Use Case Points - An Estimation Approach Gautam Banerjee
- Metrics for Objectory – Gustav Karner
- Overview of COCOMO - http://www.softstarsystems.com/overview.htm

- Cost Expert 3.3 – Software Cost Estimation and Project Planning – Quick start guide.
- Software Cost Estimation with COCOMO II, Prentice Hall, 2000. - B. Boehm, C. Abts, A.W. Brown, S. Chulani, B. Clark, E. Horowitz, R. Madachy, D. Reifer, and B. Steece.