Project Planning and Project Estimation Techniques

Naveen Aggarwal
Responsibilities of a software project manager

• The job responsibility of a project manager ranges from invisible activities like building up team morale to highly visible customer presentations.

• Most managers take responsibility for project proposal writing, project cost estimation, scheduling, project staffing, software process tailoring, project monitoring and control, software configuration management, risk management, interfacing with clients, managerial report writing and presentations etc.
Project Planning Activities

• Estimating the following attributes of the project:
  – Project size: What will be problem complexity in terms of the effort and time required to develop the product?
  – Cost: How much is it going to cost to develop the project?
  – Duration: How long is it going to take to complete development?
  – Effort: How much effort would be required?

• The effectiveness of the subsequent planning activities is based on the accuracy of these estimations.
  – Scheduling manpower and other resources
  – Staff organization and staffing plans
  – Risk identification, analysis, and abatement planning
  – Miscellaneous plans such as quality assurance plan, configuration management plan, etc.
Precedence ordering among project planning activities
Sliding Window Planning

- Planning a project over a number of stages protects managers from making big commitments too early.
- This technique of staggered planning is known as Sliding Window Planning.
- In the sliding window technique, starting with an initial plan, the project is planned more accurately in successive development stages.
- At the start of a project, project managers have incomplete knowledge about the details of the project. Their information base gradually improves as the project progresses through different phases.
- After the completion of every phase, the project managers can plan each subsequent phase more accurately and with increasing levels of confidence.
1. Introduction
(a) Objectives (b) Major Functions (c) Performance Issues
(d) Management and Technical Constraints

2. Project Estimates
(a) Historical Data (b) Estimation Techniques
(c) Effort, Resource, Cost, and Project Duration Estimates

3. Schedule
(a) WBS (b) Network Representation (c) Gantt Chart Representation
(d) PERT Chart

4. Project Resources
(a) People (b) Hardware and Software (c) Special Resources

5. Staff Organization
(a) Team Structure (b) Management Reporting

6. Risk Management Plan
(a) Risk Analysis (b) Risk Identification (c) Risk Estimation
(d) Risk Abatement Procedures

7. Project Tracking and Control Plan

8. Miscellaneous Plans
(a) Process Tailoring (b) Quality Assurance Plan
(c) Configuration Management Plan
Metrics for software project size estimation

- Currently two metrics are popularly being used widely to estimate size: lines of code (LOC) and function point (FP). The usage of each of these metrics in project size estimation has its own advantages and disadvantages.
  - LOC
  - Function Point Metric
    - Object Point Metric
    - Use Case Points Metrics
Lines of Code (LOC)

- LOC is the simplest among all metrics available to estimate project size.
- This metric is very popular because it is the simplest to use. Using this metric, the project size is estimated by counting the number of source instructions in the developed program.
- Obviously, while counting the number of source instructions, lines used for commenting the code and the header lines should be ignored.
- In order to estimate the LOC count at the beginning of a project, project managers usually divide the problem into modules, and each module into submodules and so on, until the sizes of the different leaf-level modules can be approximately predicted.
LOC Shortcomings

• LOC gives a numerical value of problem size that can vary widely with individual coding style.
• It should consider the local effort needed to specify, design, code, test, etc. and not just the coding effort.
• LOC measure correlates poorly with the quality and efficiency of the code.
• LOC metric penalizes use of higher-level programming languages, code reuse, etc.
• LOC metric measures the lexical complexity of a program and does not address the more important but subtle issues of logical or structural complexities.
• It is very difficult to accurately estimate LOC in the final product from the problem specification.
Function point (FP)

- Function point metric was proposed by Albrecht [1983].
- The conceptual idea behind the function point metric is that the size of a software product is directly dependent on the number of different functions or features it supports.
- Each function when invoked reads some input data and transforms it to the corresponding output data.
- The computation of the number of input and the output data values to a system gives some indication of the number of functions supported by the system.
- Albrecht postulated that in addition to the number of basic functions that a software performs, the size is also dependent on the number of files and the number of interfaces.
Besides using the number of input and output data values, function point metric computes the size of a software product (in units of functions points or FPs) using three other characteristics of the product as shown in the following expression.

The size of a product in function points (FP) can be expressed as the weighted sum of these five problem characteristics.

The weights associated with the five characteristics were proposed empirically and validated by the observations over many projects. Function point is computed in two steps. The first step is to compute the unadjusted function point (UFP).

\[
UFP = (\text{Number of inputs}) \times 4 + (\text{Number of outputs}) \times 5 + (\text{Number of inquiries}) \times 4 + (\text{Number of files}) \times 10 + (\text{Number of interfaces}) \times 10
\]
FP Contd ...

• **Number of inputs**
  – Each data item input by the user is counted. Data inputs should be distinguished from user inquiries. Inquiries are user commands such as print-account-balance. A group of related inputs are considered as a single input.

• **Number of outputs**
  – The outputs considered refer to reports printed, screen outputs, error messages produced, etc.

• **Number of inquiries**
  – It is the number of distinct interactive queries which can be made by the users.

• **Number of files**
  – Each logical file is counted. A logical file means groups of logically related data.

• **Number of interfaces**
  – Here the interfaces considered are the interfaces used to exchange information with other systems.
## Counting function points

<table>
<thead>
<tr>
<th>Functional Units</th>
<th>Weighting factors</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Low</td>
</tr>
<tr>
<td>External Inputs (EI)</td>
<td>3</td>
</tr>
<tr>
<td>External Output (EO)</td>
<td>4</td>
</tr>
<tr>
<td>External Inquiries (EQ)</td>
<td>3</td>
</tr>
<tr>
<td>External logical files (ILF)</td>
<td>7</td>
</tr>
<tr>
<td>External Interface files (EIF)</td>
<td>5</td>
</tr>
<tr>
<td>Functional Units</td>
<td>Count Complexity</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>------------------</td>
</tr>
<tr>
<td><strong>External Inputs (EIs)</strong></td>
<td>Low x 3</td>
</tr>
<tr>
<td></td>
<td>Average x 4</td>
</tr>
<tr>
<td></td>
<td>High x 6</td>
</tr>
<tr>
<td><strong>External Outputs (EOs)</strong></td>
<td>Low x 4</td>
</tr>
<tr>
<td></td>
<td>Average x 5</td>
</tr>
<tr>
<td></td>
<td>High x 7</td>
</tr>
<tr>
<td><strong>External Inquiries (EQs)</strong></td>
<td>Low x 3</td>
</tr>
<tr>
<td></td>
<td>Average x 4</td>
</tr>
<tr>
<td></td>
<td>High x 6</td>
</tr>
<tr>
<td><strong>External Logical Files (ILFs)</strong></td>
<td>Low x 7</td>
</tr>
<tr>
<td></td>
<td>Average x 10</td>
</tr>
<tr>
<td></td>
<td>High x 15</td>
</tr>
<tr>
<td><strong>External Interface Files (EIFs)</strong></td>
<td>Low x 5</td>
</tr>
<tr>
<td></td>
<td>Average x 7</td>
</tr>
<tr>
<td></td>
<td>High x 10</td>
</tr>
</tbody>
</table>

Total Unadjusted Function Point Count

\[
UFP = \sum_{i=1}^{5} \sum_{J=1}^{3} Z_{ij} W_{ij}
\]
FP Adjustment

Organizations that use function point methods develop a criterion for determining whether a particular entry is Low, Average or High. Nonetheless, the determination of complexity is somewhat subjective.

\[ FP = UFP \times CAF \]

Where CAF is complexity adjustment factor and is equal to \([0.65 + 0.01 \times \Sigma F_i]\). The \(F_i\) (i=1 to 14) are the degree of influence and are based on responses to questions.
FP Adjustments

Rate each factor on a scale of 0 to 5.

0 1 2 3 4 5
No Influence  Incidental  Moderate  Average  Significant  Essential

Number of factors considered ($F_i$)

1. Does the system require reliable backup and recovery?
2. Is data communication required?
3. Are there distributed processing functions?
4. Is performance critical?
5. Will the system run in an existing heavily utilized operational environment?
6. Does the system require on line data entry?
7. Does the on line data entry require the input transaction to be built over multiple screens or operations?
8. Are the master files updated on line?
9. Is the inputs, outputs, files, or inquiries complex?
10. Is the internal processing complex?
11. Is the code designed to be reusable?
12. Are conversion and installation included in the design?
13. Is the system designed for multiple installations in different organizations?
14. Is the application designed to facilitate change and ease of use by the user?
Consider a project with the following parameters.

(i) External Inputs:
   (a) 10 with low complexity
   (b) 15 with average complexity
   (c) 17 with high complexity

(ii) External Outputs:
   (a) 6 with low complexity
   (b) 13 with high complexity

(iii) External Inquiries:
   (a) 3 with low complexity
   (b) 4 with average complexity
   (c) 2 high complexity

(iv) Internal logical files:
   (a) 2 with average complexity
   (b) 1 with high complexity

(v) External Interface files:
   (a) 9 with low complexity
Example

In addition to above, system requires

i. Significant data communication

ii. Performance is very critical

iii. Designed code may be moderately reusable

iv. System is not designed for multiple installation in different organizations.

Other complexity adjustment factors are treated as average. Compute the function points for the project.
<table>
<thead>
<tr>
<th>Functional Units</th>
<th>Count</th>
<th>Complexity</th>
<th>Complexity Totals</th>
<th>Functional Unit Totals</th>
</tr>
</thead>
<tbody>
<tr>
<td>External Inputs</td>
<td>10</td>
<td>Low x 3</td>
<td>= 30</td>
<td></td>
</tr>
<tr>
<td>(EIs)</td>
<td>15</td>
<td>Average x 4</td>
<td>= 60</td>
<td></td>
</tr>
<tr>
<td></td>
<td>17</td>
<td>High x 6</td>
<td>= 102</td>
<td>192</td>
</tr>
<tr>
<td>External Outputs</td>
<td>6</td>
<td>Low x 4</td>
<td>= 24</td>
<td></td>
</tr>
<tr>
<td>(EOs)</td>
<td>0</td>
<td>Average x 5</td>
<td>= 0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>13</td>
<td>High x 7</td>
<td>= 91</td>
<td>115</td>
</tr>
<tr>
<td>External Inquiries</td>
<td>3</td>
<td>Low x 3</td>
<td>= 9</td>
<td></td>
</tr>
<tr>
<td>(EQs)</td>
<td>4</td>
<td>Average x 4</td>
<td>= 16</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>High x 6</td>
<td>= 12</td>
<td>37</td>
</tr>
<tr>
<td>External logical</td>
<td>0</td>
<td>Low x 7</td>
<td>= 0</td>
<td></td>
</tr>
<tr>
<td>Files (ILFs)</td>
<td>2</td>
<td>Average x 10</td>
<td>= 20</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>High x 15</td>
<td>= 15</td>
<td>35</td>
</tr>
<tr>
<td>External Interface</td>
<td>9</td>
<td>Low x 5</td>
<td>= 45</td>
<td></td>
</tr>
<tr>
<td>Files (EIFs)</td>
<td>0</td>
<td>Average x 7</td>
<td>= 0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>High x 10</td>
<td>= 0</td>
<td>45</td>
</tr>
</tbody>
</table>

Total Unadjusted Function Point Count: 424
Final FP

\[ \sum_{i=1}^{14} F_i = 3+4+3+5+3+3+3+3+3+3+2+3+0+3=41 \]

\[ \text{CAF} = (0.65 + 0.01 \times \Sigma F_i) \]
\[ = (0.65 + 0.01 \times 41) \]
\[ = 1.06 \]

\[ \text{FP} = \text{UFP} \times \text{CAF} \]
\[ = 424 \times 1.06 \]
\[ = 449.44 \]

Hence \[ \text{FP} = 449 \]
Feature point metric

- Feature point metric incorporates an extra parameter algorithm complexity.
- This parameter ensures that the computed size using the feature point metric reflects the fact that the more is the complexity of a function, the greater is the effort required to develop it and therefore its size should be larger compared to simpler functions.
Project Estimation techniques

• Estimation of various project parameters is a basic project planning activity. The important project parameters that are estimated include: project size, effort required to develop the software, project duration, and cost. These estimates not only help in quoting the project cost to the customer, but are also useful in resource planning and scheduling. There are three broad categories of estimation techniques:
  – Empirical estimation techniques
  – Heuristic techniques
  – Analytical estimation techniques
Empirical estimation techniques are based on making an educated guess of the project parameters.

Although empirical estimation techniques are based on common sense, different activities involved in estimation have been formalized over the years. Two popular empirical estimation techniques are

- Expert judgment technique and
- Delphi cost estimation.
Expert Judgment Technique

• In this approach, an expert makes an educated guess of the problem size after analyzing the problem thoroughly and estimates the cost of the different components (i.e. modules or subsystems) of the system and then combines them to arrive at the overall estimate.

• Problems
  – Human errors and individual bias.
  – Expert may overlook some factors inadvertently.
  – Lack of experience and knowledge

• A more refined form of expert judgment is the estimation made by group of experts.
  – It minimizes factors such as individual oversight, lack of familiarity with a particular aspect of a project, personal bias, and the desire to win contract through overly optimistic estimates.
Delphi cost estimation

- Delphi estimation is carried out by a team of experts and a coordinator.

- In this approach, the coordinator provides each estimator with a copy of SRS document to make initial estimate.

- In their estimates, the estimators mention any unusual characteristic of the product which has influenced his estimation.

- The coordinator prepares and distributes the summary of the responses of all the estimators, and includes any unusual rationale noted by any of the estimators and asks the estimators re-estimate.

- This process is iterated for several rounds.

- However, no discussion among the estimators is allowed during the entire estimation process to avoid any biasing.

- After the completion of several iterations of estimations, the coordinator takes the responsibility of compiling the results and preparing the final estimate.
Heuristic Techniques

- Heuristic techniques assume that the relationships among the different project parameters can be modeled using suitable mathematical expressions. Once the basic (independent) parameters are known, the other (dependent) parameters can be easily determined by substituting the value of the basic parameters.

- Different heuristic estimation models can be divided into the following two classes:
  - single variable model and
  - the multi variable model.
Single Variable Model

• Single variable estimation models provide a means to estimate the desired characteristics of a problem, using some previously estimated basic (independent) characteristic of the software product such as its size.

• A single variable estimation model takes the following form:

  \[ \text{Estimated Parameter} = c^1 \times e^{d_1} \]

• In the above expression, \( e \) is the characteristic of the software which has already been estimated (independent variable).

• \( \text{Estimated Parameter is the dependent parameter to be estimated which could be effort, project duration, staff size, etc.} \ c^1 \text{ and } d_1 \text{ are constants.} \)

• \( \text{The values of the constants } c^1 \text{ and } d_1 \text{ are usually determined using data collected from past projects (historical data). The basic COCOMO model is an example of single variable cost estimation model.} \)
Multivariable cost estimation model

Estimated Resource = $c^1 e^{1d1} + c^2 e^{2d2} + ...$

• Where $e^1, e^2, ...$ are the basic (independent) characteristics of the software already estimated, and $c^1, c^2, d^1, d^2, ...$ are constants.

• Multivariable estimation models are expected to give more accurate estimates compared to the single variable models,

• The independent parameters influence the dependent parameter to different extents. This is modeled by the constants $c^1, c^2, d^1, d^2, ...$.

• Values of these constants are usually determined from historical data.

• The intermediate COCOMO model can be considered to be an example of a multivariable estimation model.
Analytical Estimation Techniques

• Analytical estimation techniques derive the required results starting with basic assumptions regarding the project.

• Thus, unlike empirical and heuristic techniques, analytical techniques do have scientific basis.

• Halstead’s software science is an example of an analytical technique.
  – It can be used to derive some interesting results starting with a few simple assumptions.
  – Halstead’s software science is especially useful for estimating software maintenance efforts. In fact, it outperforms both empirical and heuristic techniques when used for predicting software maintenance efforts.
Halstead’s Software Science – An Analytical Technique

• Halstead’s software science is an analytical technique to measure size, development effort, and development cost of software products.

• Halstead used a few primitive program parameters to develop the expressions for overall program length, potential minimum value, actual volume, effort, and development time.

• For a given program, let:
  – $\eta_1$ be the number of unique operators used in the program,
  – $\eta_2$ be the number of unique operands used in the program,
  – $N_1$ be the total number of operators used in the program,
  – $N_2$ be the total number of operands used in the program.
Different Parameters

• **Length and Vocabulary**
  
  – The length of a program is total usage of all operators and operands in the program. Thus, length \( N = N_1 + N_2 \).
  
  – Program vocabulary is the number of unique operators and operands used in the program. Thus, \( \text{program vocabulary} \eta = \eta_1 + \eta_2 \).

• **Program Volume**
  
  – \( V = N \log_2 \eta \)
  
  – Here the program volume \( V \) is the minimum number of bits needed to encode the program.
Different Parameters

• **Potential Minimum Volume**
  – The potential minimum volume $V^*$ is defined as the volume of most succinct program in which a problem can be coded.
  – The minimum volume is obtained when the program can be expressed using a single source code instruction., say a function call like `foo()`;
  – In other words, the volume is bound from below due to the fact that a program would have at least two operators and no less than the requisite number of operands.
  – Thus, if an algorithm operates on input and output data $d_1, d_2, \ldots, d_n$, the most succinct program would be $f(d_1, d_2, \ldots, d_n)$; for which $\eta_1 = 2, \eta_2 = n$. Therefore, $V^* = (2 + \eta_2)\log_2(2 + \eta_2)$.

• The program level $L$ is given by $L = V^*/V$.
  – The concept of program level $L$ is introduced in an attempt to measure the level of abstraction provided by the programming language.
  – The above result implies that the higher the level of a language, the less effort it takes to develop a program using that language.
Different Parameters

• **Effort and Time**
  – The effort required to develop a program can be obtained by dividing the program volume with the level of the programming language used to develop the code.
  – Thus, effort \( E = \frac{V}{L} \),
  – Thus, the programming effort \( E = \frac{V^2}{V^*} \) (since \( L = \frac{V^*}{V} \)) varies as the square of the volume.

• The programmer’s time \( T = \frac{E}{S} \), where \( S \) the speed of mental discriminations. The value of \( S \) has been empirically developed from psychological reasoning, and its recommended value for programming applications is 18.
Length of Program

- In terms of unique operators and operands
  - it can be assumed that any program of length N consists of \( N/\eta \) unique strings of length \( \eta \).
  - Now, it is standard combinatorial result that for any given alphabet of size K, there are exactly \( K^r \) different strings of length \( r \).

- Thus,
  - \( N/\eta \leq \eta \)
  - \( N \leq \eta^{\eta+1} \)

- Since operators and operands usually alternate in a program, the upper bound can be further refined into \( N \leq \eta \eta_1 \eta_2 \). Also, N must include not only the ordered set of n elements, but it should also include all possible subsets of that ordered sets, i.e. the power set of N strings.

Therefore,

\[
2^N = \eta \eta_1 \eta_2 \text{ Or, taking logarithm on both sides,}
\]

\[
N = \log_2 \eta + \log_2 (\eta_1 \eta_2)
\]

So we get, \( N \approx \log_2 (\eta_1 \eta_2) \) approx

\[
N = \eta_1 \log_2 \eta_1 + \eta_2 \log_2 \eta_2
\]
Example:
Let us consider the following C program:

```c
main( )
{
    int a, b, c, avg;

    scanf("%d %d %d", &a, &b, &c);
    avg = (a+b+c)/3;
    printf("avg = %d", avg);
}
```

The unique operators are:
main,(),{},int,scanf,&,",",",";",=,+,,/, printf

The unique operands are:
a, b, c, &a, &b, &c, a+b+c, avg, 3,
"%d %d %d", "avg = %d"

Therefore,
\[ \eta_1 = 12, \eta_2 = 11 \]

Estimated Length = \((12\log_{10}12 + 11\log_{10}11)\)
= \((12\times3.58 + 11\times3.45)\)
= \((43+38) = 81\)

Volume = Length*\log(23)
= \(81\times4.52\)
= \(366\)
CoCoMo

• Organic, Semidetached and Embedded software projects
  – Organic:
    • if the project deals with developing a well understood application program, the size of the development team is reasonably small, and the team members are experienced in developing similar types of projects.
  – Semidetached:
    • if the development consists of a mixture of experienced and inexperienced staff. Team members may have limited experience on related systems but may be unfamiliar with some aspects of the system being developed.
  – Embedded:
    • if the software being developed is strongly coupled to complex hardware, or if the stringent regulations on the operational procedures exist.
Basic COCOMO Model

• The basic COCOMO model gives an approximate estimate of the project parameters. The basic COCOMO estimation model is given by the following expressions:

\[ \text{Effort} = a_1 \times (\text{KLOC})^{a_2} \times \text{PM} \]

\[ \text{Tdev} = b_1 \times (\text{Effort})^{b_2} \times \text{Months} \]

Where

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

– \( a_1, a_2, b_1, b_2 \) are constants for each category of software products,

– Tdev is the estimated time to develop the software, expressed in months,

– Effort is the total effort required to develop the software product, expressed in person months (PMs).
Estimation of development effort

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

• Organic : $\text{Effort} = 2.4(KLOC)^{1.05} \text{ PM}$

• Semi-detached : $\text{Effort} = 3.0(KLOC)^{1.12} \text{ PM}$

• Embedded : $\text{Effort} = 3.6(KLOC)^{1.20} \text{ PM}$

Estimation of development time

• Organic : $T_{\text{dev}} = 2.5(Effort)^{0.38} \text{ Months}$

• Semi-detached : $T_{\text{dev}} = 2.5(Effort)^{0.35} \text{ Months}$

• Embedded : $T_{\text{dev}} = 2.5(Effort)^{0.32} \text{ Months}$
Effort Vs Size
Staff Size & Productivity

When effort and development time are known, the average staff size to complete the project may be calculated as:

\[
\text{Average staff size (SS)} = \frac{E}{D} \text{ Persons}
\]

When project size is known, the productivity level may be calculated as:

\[
\text{Productivity (P)} = \frac{KLOC}{E} \frac{KLOC}{PM}
\]
Example

Suppose that a project was estimated to be 400 KLOC. Calculate the effort and development time for each of the three modes i.e., organic, semidetached and embedded.

The basic COCOMO equation take the form:

(ii) Semidetached mode

\[ E = a_b (KLOC)^{b_b} \]

\[ D = c_b (KLOC)^{d_b} \]

Estimated size of the project = 400 KLOC

(iii) Embedded mode

\[ E = 3.6(400)^{1.20} = 4772.81 \text{ PM} \]

\[ D = 2.5(4772.8)^{0.32} = 38 \text{ PM} \]
Intermediate Model

Cost drivers

(i) Product Attributes
- Required s/w reliability
- Size of application database
- Complexity of the product

(ii) Hardware Attributes
- Run time performance constraints
- Memory constraints
- Virtual machine volatility
- Turnaround time

(iii) Personal Attributes
- Analyst capability
- Programmer capability
- Application experience
- Virtual m/c experience
- Programming language experience

(iv) Project Attributes
- Modern programming practices
- Use of software tools
- Required development Schedule
## Multipliers of Cost Drivers

<table>
<thead>
<tr>
<th>Cost Drivers</th>
<th>Very low</th>
<th>Low</th>
<th>Nominal</th>
<th>High</th>
<th>Very high</th>
<th>Extra high</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Product Attributes</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RELY</td>
<td>0.75</td>
<td>0.88</td>
<td>1.00</td>
<td>1.15</td>
<td>1.40</td>
<td>--</td>
</tr>
<tr>
<td>DATA</td>
<td>--</td>
<td>0.94</td>
<td>1.00</td>
<td>1.08</td>
<td>1.16</td>
<td>--</td>
</tr>
<tr>
<td>CPLX</td>
<td>0.70</td>
<td>0.85</td>
<td>1.00</td>
<td>1.15</td>
<td>1.30</td>
<td>1.65</td>
</tr>
<tr>
<td><strong>Computer Attributes</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TIME</td>
<td>--</td>
<td>--</td>
<td>1.00</td>
<td>1.11</td>
<td>1.30</td>
<td>1.66</td>
</tr>
<tr>
<td>STOR</td>
<td>--</td>
<td>--</td>
<td>1.00</td>
<td>1.06</td>
<td>1.21</td>
<td>1.56</td>
</tr>
<tr>
<td>VIRT</td>
<td>--</td>
<td>0.87</td>
<td>1.00</td>
<td>1.15</td>
<td>1.30</td>
<td>--</td>
</tr>
<tr>
<td>TURN</td>
<td>--</td>
<td>0.87</td>
<td>1.00</td>
<td>1.07</td>
<td>1.15</td>
<td>--</td>
</tr>
</tbody>
</table>
## Multipliers of Cost Drivers

<table>
<thead>
<tr>
<th>Cost Drivers</th>
<th>RATINGs</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Very low</td>
</tr>
<tr>
<td><strong>Personnel Attributes</strong></td>
<td></td>
</tr>
<tr>
<td>ACAP</td>
<td>1.46</td>
</tr>
<tr>
<td>AEXP</td>
<td>1.29</td>
</tr>
<tr>
<td>PCAP</td>
<td>1.42</td>
</tr>
<tr>
<td>VEXP</td>
<td>1.21</td>
</tr>
<tr>
<td>LEXP</td>
<td>1.14</td>
</tr>
<tr>
<td><strong>Project Attributes</strong></td>
<td></td>
</tr>
<tr>
<td>MODP</td>
<td>1.24</td>
</tr>
<tr>
<td>TOOL</td>
<td>1.24</td>
</tr>
<tr>
<td>SCED</td>
<td>1.23</td>
</tr>
</tbody>
</table>
Intermediate COCOMO equations

\[ E = a_i (KLOC)^{b_i} \times EAF \]

\[ D = c_i (E)^{d_i} \]

<table>
<thead>
<tr>
<th>Project</th>
<th>(a_i)</th>
<th>(b_i)</th>
<th>(c_i)</th>
<th>(d_i)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Organic</td>
<td>3.2</td>
<td>1.05</td>
<td>2.5</td>
<td>0.38</td>
</tr>
<tr>
<td>Semidetached</td>
<td>3.0</td>
<td>1.12</td>
<td>2.5</td>
<td>0.35</td>
</tr>
<tr>
<td>Embedded</td>
<td>2.8</td>
<td>1.20</td>
<td>2.5</td>
<td>0.32</td>
</tr>
</tbody>
</table>
Complete CoCoMo

**COCOMO-II**

The following categories of applications / projects are identified by COCOMO-II:

- **End user programming**
- **Application composition**
- **System integration**
- **Application generators & composition aids**
- **Infrastructure**
### Stages of COCOMO-II

<table>
<thead>
<tr>
<th>Stage No</th>
<th>Model Name</th>
<th>Application for the types of projects</th>
<th>Applications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stage I</td>
<td>Application composition estimation model</td>
<td>Application composition</td>
<td>In addition to application composition type of projects, this model is also used for prototyping (if any) stage of application generators, infrastructure &amp; system integration.</td>
</tr>
<tr>
<td>Stage II</td>
<td>Early design estimation model</td>
<td>Application generators, infrastructure &amp; system integration</td>
<td>Used in early design stage of a project, when less is known about the project.</td>
</tr>
<tr>
<td>Stage III</td>
<td>Post architecture estimation model</td>
<td>Application generators, infrastructure &amp; system integration</td>
<td>Used after the completion of the detailed architecture of the project.</td>
</tr>
</tbody>
</table>
Application Composition Estimation Model

1. Assess object counts
2. Classify complexity levels of each object
3. Assign complexity weights to each object
4. Determine object points
5. Compute new object points
6. Calculate productivity rate
7. Compute the estimated effort in person months
Steps

i. **Assess object counts:** Estimate the number of screens, reports and 3 GL components that will comprise this application.

ii. **Classification of complexity levels:** We have to classify each object instance into simple, medium and difficult complexity levels depending on values of its characteristics.

<table>
<thead>
<tr>
<th>Number of views contained</th>
<th># and sources of data tables</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Total &lt; 4</td>
</tr>
<tr>
<td></td>
<td>(&lt; 2 server</td>
</tr>
<tr>
<td></td>
<td>&lt; 3 client)</td>
</tr>
<tr>
<td>&lt; 3</td>
<td>Simple</td>
</tr>
<tr>
<td>3 – 7</td>
<td>Simple</td>
</tr>
<tr>
<td>&gt; 8</td>
<td>Medium</td>
</tr>
</tbody>
</table>
### Number of sections contained

<table>
<thead>
<tr>
<th>Number of sections contained</th>
<th>Total &lt; 4 (&lt; 2 server &lt; 3 client)</th>
<th>Total &lt; 8 (2 – 3 server 3 – 5 client)</th>
<th>Total 8+ (&gt; 3 server, &gt; 5 client)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 or 1</td>
<td>Simple</td>
<td>Simple</td>
<td>Medium</td>
</tr>
<tr>
<td>2 or 3</td>
<td>Simple</td>
<td>Medium</td>
<td>Difficult</td>
</tr>
<tr>
<td>4+</td>
<td>Medium</td>
<td>Difficult</td>
<td>Difficult</td>
</tr>
</tbody>
</table>

iii. Assign complexity weight to each object: The weights are used for three object types i.e., screen, report and 3GL components using the Table 10.

### Object Type

<table>
<thead>
<tr>
<th>Object Type</th>
<th>Complexity Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Simple</td>
</tr>
<tr>
<td>Screen</td>
<td>1</td>
</tr>
<tr>
<td>Report</td>
<td>2</td>
</tr>
<tr>
<td>3GL Component</td>
<td>—</td>
</tr>
</tbody>
</table>
iv. Determine object points: Add all the weighted object instances to get one number and this known as object-point count.

v. Compute new object points: We have to estimate the percentage of reuse to be achieved in a project. Depending on the percentage reuse, the new object points (NOP) are computed.

\[
\text{NOP} = \frac{(\text{object points}) \times (100-\%\text{reuse})}{100}
\]

NOP are the object points that will need to be developed and differ from the object point count because there may be reuse.

vi. Calculation of productivity rate: The productivity rate can be calculated as:

<table>
<thead>
<tr>
<th>Developer’s experience &amp; capability; ICASE maturity &amp; capability</th>
<th>PROD (NOP/PM)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Very low</td>
<td>4</td>
</tr>
<tr>
<td>Low</td>
<td>7</td>
</tr>
<tr>
<td>Nominal</td>
<td>18</td>
</tr>
<tr>
<td>High</td>
<td>25</td>
</tr>
<tr>
<td>Very high</td>
<td>50</td>
</tr>
</tbody>
</table>
Final Effort

vii. Compute the effort in Persons-Months: When PROD is known, we may estimate effort in Person-Months as:

\[
\text{NOP} \\
\text{Effort in PM} = \frac{\text{NOP}}{\text{PROD}}
\]
Staffing level estimation

• **Norden’s Work**
  - Staffing pattern can be approximated by the Rayleigh distribution Curve

• **Where**
  - $E$: Effort required at time $t$.  
  - $K$ is the area under the curve, and 
  - $t_d^d$ is the time at which the curve attains its maximum value.

\[
E = \frac{K}{t_d^2} t \cdot e^{-t^2/2 t_d^2}
\]
Staffing level estimation

- **Putnam’s Work**
  - Putnam studied the problem of staffing of software projects and found that number of delivered lines of code to the effort and the time required to develop the project.

- K is the total effort expended (in PM) in the product development and L is the product size in KLOC.
- $t^d$ corresponds to the time of system and integration testing. Therefore, $t^d$ can be approximately considered as the time required to develop the software.

$$L = C_k K^{1/3} t^d^{4/3}$$

$C^k$ is the state of technology constant and reflects constraints that impede the progress of the programmer. Typical values of $C^k = 2$ for poor development environment $(C^k = 8$ for good software development environment $C^k = 11$ for an excellent environment)
Effect of schedule change on cost

\[ K \propto \frac{1}{t_d^4} \]
\[ \text{cost} \propto \frac{1}{t_d} \]

From the above expression, it can be easily observed that when the schedule of a project is compressed, the required development effort as well as project development cost increases in proportion to the fourth power of the degree of compression. It means that a relatively small compression in delivery schedule can result in substantial penalty of human effort as well as development cost. For example, if the estimated development time is 1 year, then in order to develop the product in 6 months, the total effort required to develop the product (and hence the project cost) increases 16 times.