1. Introduction

Information systems (IS) projects are notoriously known for being late and over-budget. Although there are many possible reasons for this, poor project estimation often has a part to play where the project manager has significantly underestimated the effort and cost involved in completing a project.

Software estimation is a very difficult task. The traditional approaches for software estimation are based on source lines of code (SLOC). This topic discusses various software estimation methods.

One attempt at base estimation on more relevant project features is function point (FP) analysis, which focuses on the functions the proposed system needs to accomplish. SLOC is still widely used and is the basis for constructive cost model (COCOMO). While these estimation approaches have limitations, significantly better estimating methods have not been widely reported.

The presentation below illustrates how activity estimation fits in the project management roadmap.

In the project planning and scheduling phase, the project manager is faced with the challenge of planning the project given the various resources, time and budget constraints. Here, one of the key methods a project manager applies is activity estimation.

In this topic, we will examine the various estimation methods used to make estimates of activities in an information systems (IS) project.

Objectives: Estimation Methods

Upon completion of this topic, you should be able to
- describe the various project estimation methods
- use source lines of code (SLOC) as a basis for making project estimates
- apply function point analysis (FPA) as a basis for making project estimates
- describe the limitations associated with both SLOC and FPA approaches
- estimate project cost and effort by applying COCOMO II
2. Size and Effort Estimation

Problems with Estimation

In a software project lifecycle, production (or the development of code) is usually relatively predictable. The other elements are highly variable.

There are some useful practical reports of software development, beginning with Brooks' Mythical Man Month (1995). In the same spirit, Armour (2002) suggested that:

1. Estimates cannot be accurate because data to support accurate estimates are not available.
2. Due to succeeding activities having to wait for all predecessors, estimates are invariably optimistic.
3. Estimation does not provide a date for completion, but rather a probability of completion.
4. The commitment process picks a date and seeks to optimise the probability of a good return on investment through risk management.
5. Software development is really a knowledge acquisition activity, so the size of the product may well be irrelevant. Thus the lines of code are not an accurate basis of estimation.
6. Historical data is of limited value, because the project being estimated by definition is different.
7. Function points are not directly measurable in the same way as lines of code. Furthermore, many aspects of system complexity are not directly related to the things that function points count.
8. Assigning more people does not speed up a software project. (Brooks even said that it would slow it down.)
9. Defect-free code is unattainable. Encountering bugs is an epiphany of knowledge, because by definition it is unexpected.

Boehm's 'Cone of Uncertainty' reflects the fact that there is greater uncertainty associated with the early stages of a project due to the vagueness regarding the scope and requirements of a project. However, as a clearer and more defined understanding of the system emerges, the cost of a project can be more accurately and reliably estimated.
The cone shows that at the very early stage of a project, what Boehm calls the "initial product definition", the estimates of project cost could be off by as much as between 25% and 400%. For example, a project that actually costs US$1 million dollars may be estimated to cost between US$250,000 and US$4 million. Similarly, the project schedule could be off by as much as 60%-160%. However, as the project progresses, the scope for errors is progressively reduced because the project is better understood and project estimates should be more accurate and reliable.

The size of the system is found to be an important variable in the cost and time required to develop software. Size can be measured on a number of dimensions. The most commonly used are the amount of information processed, technical requirements and performance drivers (cost, time and quality).

**Prediction tools**

The amount of time required to develop a set of code is a function of its size. Size is measured in terms of source lines of code (SLOC or LOC), or function point (FP) method. These methods are based primarily on McLeod and Smith (1996) and Pressman (1997). More recently, use case point analysis is also commonly used method for estimating size.

Project estimation is a necessary exercise that is needed for rational management of IS projects. However, precise estimates are unattainable. In this topic, we discussed two estimation methods that are widely used. It must be realised, however, that these methods are quite limited.

Web and eCommerce needs generate different kinds of IS projects. Mendes et al. (2001) discuss estimation in this environment. This is important not only for Web projects, but also as a paradigm for the inevitable new types of IS project environments that will arise in the future. As with any IS project, prediction is a necessary part of an effective software process, and realistic estimates are needed for project managers and developing organisations to effectively manage their resources. The prediction process involves:

- capturing data of past efforts by type of work.
- analysis to determine useful metrics for project time and cost
- application of prediction models
- assessing prediction model effectiveness

Mendes et al. (2001) propose linear regression as prediction tools applied to data similar to that collected for COCOMO (or for LOC).

**Productivity loss**

There are many causes for productivity loss. For one, the amount of time applied by humans to actual coding is only about 20 hours per week due to machine downtime, emergency diversion, meetings, paperwork, sick time and many other factors. A great deal of this waste is recovered by using automated tools, except for machine downtime.

Interactions require co-ordination with others, and the more the interactions, the less the productivity. High-level languages increase productivity by reducing programming through reusable components, but the custom parts of a system still have to be accomplished by the development team.

**3. Source Lines of Code**

SLOC refers to the number of lines of code excluding comment lines and other non-essential lines of code. Both LOC and FP methods begin with the scientifically sound approach of gathering historical records of experiences of past projects. This historical data is the basis for identifying the relationship between key measures of importance (such as the person-months of effort and dollars expended) and other factors of importance (such as the pages of documentation generated, errors encountered, system defects and people assigned). An implementation of the LOC approach uses the following key measures per line of code.

**Lines of code operation**

<table>
<thead>
<tr>
<th>Averages of past projects LOC</th>
<th>Effort</th>
<th>Budget (US$000)</th>
<th>Documentation (pages)</th>
<th>Errors</th>
<th>Defects</th>
<th>People</th>
</tr>
</thead>
<tbody>
<tr>
<td>20,543 months</td>
<td>361</td>
<td>1,194</td>
<td>201</td>
<td>52</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>Average per KLOC</td>
<td>1.606</td>
<td>17.573</td>
<td>58.122</td>
<td>9.784</td>
<td>2.531</td>
<td>0.195</td>
</tr>
</tbody>
</table>

To obtain SLOC, estimate thousands of lines of code (KLOC) and multiply by the averages for each category.

When a new project is encountered, an estimate is made of the lines of code that the project will require. For example, if a new project is estimated to involve 10,000 lines of code, estimates of these measures would be:

**Estimates for 10,000 lines of code**

| Effort | 1.606*10 KLOC | ~ 16 person-months |
| Budget | 17.573 * 10 KLOC | = US$175,730 |
This approach is admittedly rough, but provides various benefits. The following table lists the benefits and limitations of this method.

<table>
<thead>
<tr>
<th>#</th>
<th>Benefits</th>
<th>Limitations</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>It is a very easy-to-implement estimation method.</td>
<td>Ideally, firms could build their database in categories by type of work, as averaging radically different projects together will lead to obvious inaccuracies.</td>
</tr>
<tr>
<td>2.</td>
<td>Although gathering the data is time consuming, it is very quick once it is obtained.</td>
<td>It takes time to generate a database of historical project results. As with any statistical approach, the more data the greater the confidence. However, the older the data, the less likely they are appropriate to current operations.</td>
</tr>
<tr>
<td>3.</td>
<td>The more appropriate the data, the more accurate the estimation.</td>
<td>The model as demonstrated is totally linear, while project size may have different impacts at different size levels.</td>
</tr>
</tbody>
</table>

4. Function Point Analysis

Albrecht's function point analysis method (Pressman, 1997) supplements the project size bases of estimation. The International Function Point (IFP) users group has a website for function point analysis, presenting current views. The aims of this approach are a consistent measure meaningful to the end user with rules that are easy to apply. The method can be used to estimate cost and time based upon requirements specifications, and is independent of the technology used. Xerox in 1985 noted a rapid drop in productivity for IS projects with increasing system size.

The function point method works in a very similar way to lines of code, except that the basis for estimation is function points. The original function point method involved counts of the number of activities in five categories – user inputs, user outputs, user inquiries, files accessed and external interfaces. The number of functions are counted by complexity level for each factor and multiplied as follows:

**Count-total calculations**

<table>
<thead>
<tr>
<th>Measurement parameter</th>
<th>Complexity low</th>
<th>Complexity average</th>
<th>Complexity high</th>
<th>Product</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of user</td>
<td>_____ * 3 +</td>
<td>_____ * 4 +</td>
<td>_____ * 6</td>
<td>=</td>
</tr>
</tbody>
</table>
The count-total is the sum of the product column. To demonstrate this, let us look at a hypothetical software proposal.

View the animation for a demonstration of count-total calculation.

### Count-total calculation

Nambra bank is developing a new bank accounts record system. The system includes user inputs, user outputs, user inquiries, files accessed and external interfaces. These parameters are classified as simple, average or complex in complexity.

The complexity level for each parameter is as follows:

The number of functions is calculated by multiplying the complexity level for each parameter with its count. This is represented in the product column.

Count-total is the sum of the product column.

### Function calculation

The next step is to calculate Fi, which is calculated by adding the ratings over 14 factors on a 0–5 scale, with 0 representing no impact, 1 incidental, 2 moderate, 3 average, 4 significant and 5 essential impact. The following table lists the 14 factors, with their ratings, for calculation with a hypothetical set of impacts:

<table>
<thead>
<tr>
<th>#</th>
<th>Function</th>
<th>Rating</th>
<th>1-5 scale</th>
</tr>
</thead>
<tbody>
<tr>
<td>F1</td>
<td>Does the system require reliable backup and recovery?</td>
<td>Significant</td>
<td>4</td>
</tr>
</tbody>
</table>
Function points are then calculated by the formula:

\[ FP = \text{count-total} \times [0.65 + 0.01 \times \text{sum of } F_i] \]

In the hypothetical example, the estimation of effort would be:

\[ FP = 494 \times (0.65 + 0.01 \times 32) = 479.18 \]
The method is to take the calculated FP (in this case, 479.18) and multiply by these averages. This would yield the following estimates:

<table>
<thead>
<tr>
<th>Averages of past projects</th>
<th>Effort</th>
<th>Budget (US$000)</th>
<th>Documentation (pages)</th>
<th>Errors</th>
<th>Defects</th>
<th>People</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average of FP Estimate</td>
<td>479.18</td>
<td>0.05297</td>
<td>0.57945</td>
<td>1.91653</td>
<td>0.32263</td>
<td>0.08347</td>
</tr>
<tr>
<td>Estimate</td>
<td>25.4</td>
<td>278</td>
<td>918</td>
<td>155</td>
<td>40</td>
<td>3</td>
</tr>
</tbody>
</table>

Estimation of software project effort continues to be a difficult task despite the usefulness and availability of estimation methods based on lines of code and function points. Heiat and Heiat (1997) compared lines of code and function point estimation, finding that they were roughly equal in accuracy. Other estimation approaches continue to evolve as the nature of software and the software environment changes, with far less reliance on actual coding (which Boehm pointed out was the most predictable part of a software project).

There have been many adaptations of FPA with an aim to bridge the gaps in estimation. Use case point (UCP) analysis is one such adaptation that is more suited when Object oriented approach to development and more specifically use case based analysis is adopted.

**Reading:**

Read the following to learn more about UCP Clemmons (2006).


**5. Exercise 1**

Click the link below for an exercise to examine your knowledge of function point analysis.

Exercise: Function Point Analysis
Fortune Sports is building a new stock control system. An initial analysis of the functions of the system is given below:

<table>
<thead>
<tr>
<th>Function</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>20 user inputs</td>
<td>Classified as average in complexity</td>
</tr>
<tr>
<td>10 user outputs</td>
<td>Classified as simple in complexity</td>
</tr>
<tr>
<td>20 user inquiries</td>
<td>Classified as simple in complexity</td>
</tr>
<tr>
<td>5 file accessed</td>
<td>Classified as average in complexity</td>
</tr>
<tr>
<td>8 external interfaces</td>
<td>Classified as complex in complexity</td>
</tr>
</tbody>
</table>

Q1. What is the count-total for the new stock control system?
1. 280
2. 300
3. 310

The correct answer is
• option 3, 310

Q2. Assume that the Fi for the new stock control system is calculated at 30. How many function points is the new stock control system?
1. 294.5
2. 300
3. 302.5

The correct answer is
• option 1, 294.5

Q3. Assume that previous IS projects at Fortune Sports have indicated that the average effort per FP is 0.085 person months and the average cost is US$750 per FP. What would be the expected approximate effort and cost of the project?
1. 22 person months and US$200,875
2. 25 person months and US$220,875
3. 27 person months and US$240,875

The correct answer is
• option 2, 25 person months and US$220,875

6. Constructive Cost Model

Now, let us look at another estimation method that.

The constructive cost model (COCOMO) applies logarithmic regression on past data to more accurately reflect the time required to accomplish code development. COCOMO reflects the impact of learning, where programmers and developers are expected to gain in the rate of productivity as the project progresses.

Boehm (1981) presented a series of constructive cost models (COCOMO) for estimating the effort required for developing software. Companies could generate
their own using regression on their productivity data. Pressman (1997) contends that the COCOMO approach is not perfectly accurate in many conditions, but provides a good starting basis for projects oriented around work, which can be described in terms of lines of code generated.

The model was gradually enhanced and adapted for more generic as well as specific application development situations. The generic extended version of the model, called COCOMO II provides estimation for two stages – early design stage and post-architectural stage. The model provided formulae to take into account various cost drivers as well as process drivers.

**Reading: Constructive Cost Model**

The COCOMO is continually being refined and improved. The [Centre for Software Engineering](http://csse.usc.edu/csse/research/COCOMOII/cocomo2000.0/CII_modelman2000.0.pdf) at the University of Southern California is a good place for further articles and updates about COCOMO. For a comprehensive study of the model, read:


You can also read a wide variety of articles on the subject of estimation from R.S. Pressman & Associates's Downloadable Reference Library.

### 7. Exercise 2

Click the link below for an exercise to examine how to use different COCOMO models.

[Exercise: COCOMO Models](#)
8. Self-Assessment

Now, try the self-assessment questions to test your understanding of the topic. Click the following link to open the Self-Assessment in a new window.

Self-Assessment

Q1. During which phase of the project life cycle is there less uncertainty in project estimation?
   1. At the start of a project
   2. During the early part of a project
   3. During the middle part of a project
   4. At the later stages of a project

Q2. Which one of the following does NOT relate to estimation difficulty?
   1. Every project is new and past experiences are rarely exactly duplicated
   2. There is no common measure across software projects
   3. Human creativity is involved making it hard to accurately estimate
   4. Large software projects take so long that methods and people often change

Q3. Which one of the following is NOT a measurement parameter in function point analysis?
   1. The number of user inputs
   2. The number of user outputs
   3. The number of user inquires
   4. The number of user interfaces

9. Summary

This topic covered the following main points:

- It is very difficult to obtain accurate information systems project estimates due to normal project complications, such as the interaction of activities, as well as due to the creative nature of information systems projects.
- While system size is not a perfect basis for estimation, it has been found the most useful in the past.
- The traditional measure of project size has been lines of code (LOC).
- An alternative to SLOC is function points (FP), which is more representative of the functions that a system is to perform.
- SLOC and FP analysis rely on accurate information about the system and functionality. Thus, they may be of limited use early on in the software development process.
- SLOC, FP as well as UCP can be used as the basis of the constructive cost model (COCOMO II), a widely published commercial method of software project estimation.

References