In this chapter we explain the typical and distinct types of requirements that make up a software development or enhancement project; the various effort estimation approaches that are covered in this book; what is involved in producing a detailed estimate; and the use of functional size measurement in effort estimation.

Throughout this book we concentrate on estimating the effort and duration involved in a software project. Effort and duration estimation normally leads to the estimation of cost, so we have provided an introduction to cost estimation in Chapter 15.

Types of Project Requirements

Before we delve into the different estimation approaches, it is important to understand the different types of requirements that make up a project and to be aware of what is, and is not, included in the estimation approaches in this book.

The project estimation approaches explained in this book rely on the functional size of the software as a key input variable and are

1 Functional size is the size of the software to be developed. It is expressed in units such as function points. The units may vary depending on the chosen functional size measurement method (FSMM). Functional size measurement can be compared to the measurement of a building being expressed in square meters or square feet.
applicable to projects where software is developed or enhanced. This will be explained further after the discussion on types of requirements, since not every project that involves software or systems is suitable for functional size measurement. Functional size measurement pertains specifically to projects where software is developed, modified, or enhanced.

To make sense of functional size measurement and where it fits with estimating, it is useful to discuss the three types of software project requirements. Figure 1-1 shows the different types of software project requirements. Elsewhere in this book we will discuss a number of ways to establish the functional size of a piece of software without needing detailed knowledge of functional sizing.

Note The word “user” in the context of functional size measurement means any person or thing that interacts with the software at any time (such as other pieces of software, hardware, end users, and administrators) that has a requirement for data or services supported by the software being developed. This is an important concept because functional size measurement can be used to size software that has no human users. For example, the software interacts with other software or hardware. It may be useful to think of a “user” as analogous to an actor in the Use Case methodology. (For other definitions, refer to the Glossary.)

As depicted in Figure 1-1, project requirements can be categorized into three distinct types (this breakdown also increases understanding

between the users and the project team). The three types of requirements are as follows:

- **Functional requirements** These represent WHAT functions will be included in the software. Functional requirements are the business processes performed by or supported by the software (for example, record and store ambient temperature) and include the functions that the software must perform. The size of functional requirements is expressed in function points.

- **Nonfunctional requirements** This is the second type of software requirement and represents HOW the software must perform. Nonfunctional requirements describe how the software must operate and are not included in functional size. Sometimes known as “quality requirements,” the nonfunctional requirements include suitability, accuracy, interoperability, compliance, security, reliability, efficiency, maintainability, portability, and quality in use, as described by the ISO (International Organization for Standardization) standard ISO/IEC (International Electrotechnical Commission) 9126 series, plus a range of performance requirements. While these requirements should also be defined by the system’s users/customers, they are often not articulated separately (or at all), but rather are sprinkled throughout requirements documents.

The nonfunctional requirements are the contracted specifications for the software and include requirements for security (for example, data encryption), performance (for example, response time and reliability), accuracy (for example, governmental approvals required), and other specifications of how the software must perform.

- **Technical (build) requirements** These requirements address how the software will be developed or “built” and include tools, methods, type of project, resource skill levels, and so on. These requirements are where architectural design, configuration management methods, development methodology, use of packages, and use of CASE (Computer Aided Software Engineering) tools, for example, come into play. The technical requirements include hardware and software requirements, infrastructure requirements, database type, and so on.

All three types of project requirements are necessary to produce a realistic estimate of the total software project effort.

---

**Functional Size**

Knowing the functional size of the software to be developed is essential for macro estimation. Chapter 18 provides an introduction to functional sizing.
Functional size represents the size of the functional requirements. Functional size is an important input in software estimation, but it is only one of a number of required variables. For a new development project, functional size is the size of all of the delivered or installed functionality (analogous to a building’s floor plan). For an enhancement project, functional size is the total size of all functional requirements that are new, renovated (changed), or removed (deleted) from the software.

Nonfunctional requirements fall outside functional size. The value adjustment factor (VAF)—which is an optional step in the IFPUG (International Function Point Users Group) function point method—is intended to address a portion of nonfunctional requirements.\(^3\) According to industry experts including Barry Boehm (COCOMO II), Watts Humphrey (Software Engineering Institute), and Bill Perry (Quality Assurance Institute), the impact of nonfunctional requirements can double the effort required to develop software depending on the exact constraints involved.

**Note** Functional size measurement pertains only to the size of the software’s functional user requirements.

### Software Estimation Approaches

There are two major software estimation approaches: *macro* (for example, top-down; parametric) and *micro* (for example, bottom-up; task based), although some estimation approaches combine typical aspects of both macro and micro techniques. Within each approach are several estimating techniques, as shown in Table 1-1.

Note that the estimating techniques listed in Table 1-1 are the techniques presented in this book, not a definitive or exhaustive list of estimating techniques. Any of the techniques could be used at any point in the life cycle. However, the more accurate our estimate of the project’s size, the more precise our effort and duration estimates can be. The relative precision of our resultant estimates will match the precision of our inputs.

Table 1-2 outlines some of the strengths and weaknesses of each estimation technique.

Note that all the macro techniques have problems with small projects as a result of the greater variation in the ratios of size to effort and duration typically seen in smaller projects.

---

\(^3\) Note that the VAF may be phased out in the future and replaced by an alternative option.
### Chapter 1: Project Estimation

<table>
<thead>
<tr>
<th>Approach</th>
<th>Estimation Technique</th>
<th>When Applicable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Macro-Estimation</td>
<td><strong>Equation Use</strong> In this method, the size of the project is applied to an appropriate equation that has been derived from project data. The result is a useful indicative, or “ballpark” estimate of effort and duration. Includes Program Evaluation and Review Technique (PERT) equations.</td>
<td>Useful when little information is known or when requirements are incomplete. High-level estimate.</td>
</tr>
<tr>
<td></td>
<td><strong>Comparison</strong> Essentially, this involves finding a group of completed projects with project attributes similar to those of the proposed project, and using the data from those projects to provide a guide for the estimate of the effort and duration for your new project.</td>
<td>Useful when enough project attributes and a range for the functional size are known. This allows the estimator to adequately gauge that the comparison projects are similar.</td>
</tr>
<tr>
<td></td>
<td><strong>Analogy</strong> This method is based on being able to find a completed project that is a very good match to your proposed project based on its major attributes. The project delivery rate and speed of delivery from the analog are then used to guide the estimate of the effort and duration for your new project.</td>
<td>Useful when even more information is known about the project being estimated. Best accomplished after requirements are complete.</td>
</tr>
<tr>
<td>Micro-Estimation</td>
<td><strong>Work Breakdown</strong> In this method, the effort and duration associated with each component or activity of the software project is separately estimated and the results aggregated to produce an estimate of the whole job. This is a bottom-up technique.</td>
<td>Useful when the project scope is well defined and an accurate work breakdown structure can be defined. Typically, experienced project team members estimate their project tasks based on historical completed similar tasks, and the overall estimate is the aggregated sum of all work breakdown structure task estimates.</td>
</tr>
</tbody>
</table>

**Table 1-1** Estimation Approaches and Techniques
### Table 1-2  Estimation Techniques Strengths and Weaknesses

<table>
<thead>
<tr>
<th>Technique</th>
<th>Strengths</th>
<th>Weaknesses</th>
</tr>
</thead>
<tbody>
<tr>
<td>Equation Use</td>
<td>Based on a depth of historical data. Ideal for an indicative estimate early in the life of a project.</td>
<td>Too imprecise for accurate estimation. You need to be confident that the equation being used is relevant to your project. The equation always provides an estimate, even if your project is unusual or exceptional. Not very useful for small project estimation.</td>
</tr>
<tr>
<td>Comparison</td>
<td>Based on representative experience. Objective, repeatable, verifiable, defensible. Efficient and if used correctly, provides a good guide to the likely effort your project will consume.</td>
<td>Based on representative past experience that may no longer be relevant. For best results, the technique needs to be aligned to your environment/organization. Cannot be used when no past experience is available.</td>
</tr>
<tr>
<td>Analogy</td>
<td>Based on representative experience. Objective, repeatable, verifiable, defensible.</td>
<td>Based on a past experience that may no longer be relevant. Difficult to find suitable analog projects. For best results, needs to be closely aligned to your environment or organization.</td>
</tr>
<tr>
<td>Work Breakdown</td>
<td>Detailed and specific to this project.</td>
<td>Subjective, can be optimistic. Requires detailed knowledge of the proposed project’s structure and individual components. Requires extensive knowledge of the organization and development environment. May overlook items or activities.</td>
</tr>
</tbody>
</table>

---

4 Although there is no specified size as to what constitutes a small project, for a project measured in function points, most software metrics consultants agree on a lower limit of around 30 function points.
Other Techniques

Techniques from artificial intelligence research have also been applied to develop software effort estimation models. For example, artificial neural networks and decision trees have been used to estimate effort. These methods do not require the user to propose an explicit functional form for the model, only the input and output metrics. These techniques are beyond the scope of this book.

Estimate Ranges

Remember that the earlier an estimate (or if there is little known data, a “guesstimate”) is performed, the less accurate it will be. For this reason, when relaying an estimate to your customer, you should always provide a plus/minus range to accompany the estimate to indicate the degree of confidence in the estimate. Your original estimate is the most likely estimate, while upper and lower figures are generally the optimistic and pessimistic estimates. The Project Management Institute’s Project Management Body of Knowledge (PMBOK® version 4) provides useful guidance on estimate ranges.

A number of equation approaches can be used to present a weighted average of the estimate (examples include PERT, CPM, Monte Carlo). The following example uses the Program Evaluation and Review Technique (PERT) approach to estimate likely effort for individual project activities:

\[ Te = \frac{To + 4 Tm + Tp}{6} \]

where

- \( Te \) = expected effort
- \( To \) = most optimistic estimate
- \( Tm \) = most likely estimate
- \( Tp \) = most pessimistic estimate

Rather than giving the customer a fixed, single number of effort hours, it is far more helpful to state: “Our estimate is 250 hours, plus or minus 50 hours, based on what we know about the project at this stage.”

Timing of Estimates

Figure 1-2 shows you the impact that your increasing knowledge of the system requirements will have on the accuracy of your estimates.

Figure 1-3 is provided to assist you in deciding when particular estimating methods are most appropriate in the life cycle of your project.

\(^5\) Figure 1-3 is supplied by Charles Symons of Software Measurement Services Ltd.
All of the macro-estimation techniques presented in this book can be applied with both approximate and detailed functional sizing. However, as the graph shows, the precision of the resultant estimates will improve as the precision of the functional sizing improves.

### Producing a Detailed Estimate

To produce a detailed estimate—as opposed to an initial ballpark or indicative estimate for feasibility consideration—typically a micro-estimating technique (for example, work breakdown) will be used to develop the effort estimate. A macro-estimating technique can then be used to validate the micro-estimate.

**Note** Where the macro- and micro-estimates vary by more than 10 to 15 percent, you should identify why and rework your estimates.
Chapter 1: Project Estimation

Estimates are best derived from an organization’s own experience database. You can build your own experience database by entering your project data in the ISBSG Repository. If you have not yet established your own “experience” database, you can use the ISBSG Repository as your source for macro-estimations.

**Note** If you have entered data from your projects into the ISBSG Repository, you have the best of both worlds. You can extract your projects to derive the project delivery rate (expressed in hours per function point) to be used. Then you can extract similar projects from other organizations for comparison. The project data from other organizations will be particularly useful where you are estimating a project that includes a variable that you have no previous experience of, for example: a platform or language that you have not used previously.

Functional size is only one of the many variables known to influence effort, but it is recognized as a key driver. As the functional size increases, so does associated effort.

In its simplest form, this relationship is expressed as:

\[ \text{Effort} = \text{Size} \times \text{Project Delivery Rate} \]

where Project Delivery Rate is expressed as **Hours per Functional Point** and hours are effort hours.

If you are using comparison or analog macro-estimation methods, the information shown in Table 1-3 should be included in your set of attributes for selecting similar projects.

Use your common sense when matching projects and/or adjusting project delivery rates. For example:

- If the only similar project identified was negatively impacted by the learning associated with the introduction of a new technology, but the skills acquired will be utilized by this project, then the project delivery rate (PDR) can be expected to be better (that is, have a lower PDR) than the similar project.
- If this project is similar to a previous project, with the exception that you have to provide additional deliverables (for example, a user manual), then the project delivery rate can be expected to be less productive (that is, higher).

---

6 Refer to the appendixes for a detailed description of the data in the ISBSG Repository. Go to www.isbsg.org for information on how to submit projects for inclusion in the Repository.

7 For the estimating examples in this book, the functional size measure used is units of function points according to the IFPUG method (FP). Note that all of the IFPUG releases use the same units of measure: IFPUG function points. At this printing, the most recent IFPUG release is IFPUG 4.3 (published in September 2009). For further information visit www.ifpug.org.

8 Both these approaches are covered in detail in Chapter 13 in this book.
A commercial estimating tool can be used for estimating using industry data or your organization’s experience or knowledge database. Be aware that you need to know the basics of software project estimating and how your organization supports estimation before shopping for an estimating tool.

### Use of Function Point Sizing (Functional Size Measurement) in Effort Estimation

Functional requirements are sized in function points and are measured using a functional size measurement method such as COSMIC (Common Software Measurement International Consortium), FiSMA (Finnish Software Measurement Association), IFPUG, or NESMA (Netherlands Software Metrics Association). Each of these ISO/IEC standardized functional size measurement methods has its own units of measure and approach to determining functional size. A tutorial on functional size measurement can be found in Chapter 18. Simple case studies that illustrate the counting of function points are also provided.

Functional size has a role to play in both the macro- and micro-estimating approaches, as shown in Table 1-4.

<table>
<thead>
<tr>
<th>Project type</th>
<th>Development, Enhancement, or Redevelopment (on a new platform).</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size</td>
<td>Functional size measurement.</td>
</tr>
<tr>
<td>Project goals</td>
<td>In terms of quality, cost, schedule, and constraints (that is, priority of each). Note that cost, scope (functionality and quality), and time (effort) are the famous “triple constraint” of project management.</td>
</tr>
<tr>
<td>Development platform</td>
<td>Mainframe, midrange, PC, or multiplatform.</td>
</tr>
<tr>
<td>Language</td>
<td>Programming language or language level.</td>
</tr>
<tr>
<td>Task selection</td>
<td>Similar project profile in terms of activities and deliverables from those activities. (Phases and work activities included.)</td>
</tr>
</tbody>
</table>

**Table 1-3** Attributes for Estimation by Comparison and Analogy

A commercial estimating tool can be used for estimating using industry data or your organization’s experience or knowledge database. Be aware that you need to know the basics of software project estimating and how your organization supports estimation before shopping for an estimating tool.

### Use of Function Point Sizing (Functional Size Measurement) in Effort Estimation

<table>
<thead>
<tr>
<th>Approach</th>
<th>Use of Functional Sizing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Macro-estimating</td>
<td>Functional size is a key input to most estimating equations and project comparisons.</td>
</tr>
<tr>
<td>Micro-estimating</td>
<td>The functional size allows you to calculate the &quot;expected&quot; project delivery rate for comparison with past projects.</td>
</tr>
</tbody>
</table>

**Table 1-4** Use of Functional Size Approximation in Estimation Approaches
Summary

In this book we explain the three macro-estimation techniques in detail and define the data and tools that you need to appropriately use these techniques. We also provide an overview of micro-estimation.

Any technique is only as good as the data and information on which it is based. You cannot expect any technique to compensate for lack of definition, understanding, or agreement on the scope of the software job to be done. Just as a chain is only as strong as its weakest link, estimates of effort are only as reliable as the least reliable input variable.

And finally: *Never rely on a single estimation method for a project.* The more cross-checks and sanity checks you can employ, the better.