Effort Estimation for Embedded Software Development

By Gustavo Cubas,
Software Engineering Manager,
Avionyx, 2010
Software cost estimation has been a subject of academic and practitioners research since the start of software engineering in the 1950s. Considering the number of variables that affect software productivity, it is no surprise that today, sixty years later, most companies continue to rely on expert judgment to estimate software development costs. The majority of the software estimation models have been developed and used mostly in the domain of business applications. This article provides a brief introduction to the subject of software estimation and how to apply them to the development of embedded applications. Follow-up articles will elaborate on the methods enumerated in this introductory article.

Introduction

In his classic paper “No Silver Bullet”¹, Turing Prize winner Frederick Brooks identified four essential characteristics of software:

- **Complexity**: because the interactions of components in a software program grow in a non-linear fashion with respect to its size.
- **Conformity**: because software has to conform to an arbitrary complexity imposed by human institutions and the systems it has to interface with.
- **Changeability**: because software is constantly subject to pressure of change.
- **Invisibility**: because software is invisible and difficult to visualize, depriving developers from the most powerful conceptual tool for reasoning about it: a visual representation.

In the development process, the software engineer has to deal with these essential characteristics along with other challenges that affect productivity, such as environmental factors, tool and platform issues.

Software development can be seen as a construction process or as a problem-solving endeavor. In the first case, software estimation would consist of sizing the software product and using a software-size/time productivity metric to determine the effort required. If software development is seen as a problem solving activity, then we would need do measure product complexity and the team’s problem solving productivity to estimate effort. In practice, software development effort is affected by both product size and product complexity.

In 1981 Barry Boehm² made us realize that the accuracy of the estimation improves as we progress through the software lifecycle. The concept was later made more general and named the Cone of Uncertainty by Steve McConnell³.

From this point of view, the estimation of the verification tasks should be more accurate once the requirements and code of the system are already written. Figure 1 shows how estimation accuracy improves as the project approaches completion.

---

Software is an abstract product resulting from the conjunction of a number of variables that affect the software development process, all of which are very hard to characterize and to measure, much less predict. Norman Fenton identified three types of entities that intervene in the software process:

- Product
- Process
- Resources

The number of variables that can be included under any one of these categories can be overwhelming. Multiple surveys and reports have identified numerous critical factors that make software projects overrun budgets and schedules. While it is tempting to consider the possibility of developing a model that takes into consideration all of the possible factors, its complexity would make it unpractical. Table 1 shows some of the most commonly referred factors that affect project performance. Not only is it difficult to characterize and measure these variables, it is also difficult to identify critical factors that are unknown when preparing the estimate: these factors are commonly called risks.

---

In spite of all the uncertainty you have to deal with, customers and managers expect that you, as a software engineer, are able to provide an estimate within a reasonable degree of confidence. Fortunately, there are several models you can keep in your toolbox to use depending on the problem at hand.

<table>
<thead>
<tr>
<th>Process Factors</th>
<th>Product Factors</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Project lifecycle</td>
<td>• Requirements volatility</td>
</tr>
<tr>
<td>• Process maturity</td>
<td>• Requirements clarity</td>
</tr>
<tr>
<td>• Criticality</td>
<td>• Component complexity</td>
</tr>
<tr>
<td>• Budgetary constraints</td>
<td>• Concurrency</td>
</tr>
<tr>
<td>• Schedule constraints</td>
<td>• Timing constraints</td>
</tr>
<tr>
<td></td>
<td>• Inter-component complexity</td>
</tr>
<tr>
<td><strong>Resource Factors</strong></td>
<td>• Reuse requirements</td>
</tr>
<tr>
<td>• Turnover</td>
<td>• Usability requirements</td>
</tr>
<tr>
<td>• Developer and team experience</td>
<td></td>
</tr>
<tr>
<td>• Developer capability</td>
<td></td>
</tr>
<tr>
<td>• Team size</td>
<td></td>
</tr>
<tr>
<td>• Team location</td>
<td></td>
</tr>
<tr>
<td>• Team dynamics</td>
<td></td>
</tr>
<tr>
<td>• Platform capabilities</td>
<td></td>
</tr>
<tr>
<td>• Development tools</td>
<td></td>
</tr>
</tbody>
</table>

Table 1. Some factors that affect productivity

Software Measurement

In order to compare projects and identify their commonalities and differences, we must first characterize and measure software projects. Therefore, the first step towards estimating and controlling projects effectively is to establish a proper software measurement system.

Due to its abstract nature, software measurement is different from measuring physical variables, which rely on comparing the measured object to a pattern. Measurement implies assigning quantities to attributes of an entity. Software measurement uses different types of scales. Those scales were first named by psychologist S. S. Stevens in 1946 and they are classified by the admissible scale transformations.

Nominal: In a nominal scale, entities are categorized using labels such as classifying computer programs by the programming language.

Ordinal: Ordinal scales allow using rank order relationship, saying nothing about the difference between two ranks. An example would be a subjective classification of the complexity a computer program using a Likert scale.

---

**Interval:** In interval scales, quantitative attributes are used to measure the distance between two values. These values are meaningful but the reference point is arbitrary. This type of scale is not commonly used in software measurement. Examples of an interval scale are the Celsius or Fahrenheit temperature scales.

**Ratio:** Ratio scales are the most information rich. Ratio scales derive their name from the fact that measurement is done estimating the ratio between a unit measurement and the magnitude of a contiguous quantity. Ratio scales have a non-arbitrary zero point. Take for instance, the case of temperature; the Kelvin scale is a ratio scale because it is not possible to have a temperature under 0 Kelvins. In software measurement, we could use a ratio scale to measure source lines of code, cyclomatic complexity, software size in unadjusted function points, etc.

Implementing a software measurement program has an overhead cost; therefore, it needs to be optimized for return on investment. According to Mike Cohn\(^6\), the accuracy of estimation is not linear with the effort spent on the process. As an example, you could spend time tracking the hours you spend on every single configuration management activity; or, you could realize they normally represent around 5% of the total project effort, in which case you wouldn’t need per-activity metrics.

The number of factors that affect the software process can be overwhelming (see Table 1) and you can’t pretend to be able to measure them all. In order to select the metrics you need for estimation purposes, you can follow a process such as the GQM method (Goal-Question-Metric) by Victor Basili\(^7\). The GQM paradigm states that you need to set objectives before embarking on a software measurement program. Once you have set your objectives, you ask yourself questions that allow you to characterize the object of measurement in order to achieve the goal set. Finally, a set of metrics is proposed that allow answering those questions in a measurable way. Van Solingen and Berghout\(^8\) describe the GQM method as the four-phase process illustrated in Figure 2.

---


For a measurement program to be effective for estimation, the goal should be characterizing projects in order to compare them. This will allow you to know whether or not a project is relevant to be considered as a reference for the estimation of another project. Also, it will tell you if your estimation model is effective or if the results you are getting are a matter of chance. It could happen that, at the end of a project, you realize you spent exactly what you had budgeted, but the actual conditions were completely different from the ones you had assumed when you did your estimation.

**Estimation Methods**

In general, all estimation methods (whether performed by a computer or a human being) follow a similar approach:

1. Characterize the product size
2. Characterize the product complexity
3. Characterize environmental factors
4. Characterize projects developed in the past
5. Assess commonalities and differences between reference project and target project
6. Extrapolate the variables that need to be estimated based on the reference project

The way each of these steps is performed determines the estimation method.

Estimation methods can be classified in numerous ways. The software estimation literature does not clearly distinguish between model, method or technique. In this article, we refer to them simply as estimation methods.
We can classify estimation methods by the level of granularity as:

- **Top-Down or System-level methods**: These methods (also called macro-estimation techniques) estimate the development effort at a system level without looking into the individual tasks that comprise the product development activity.

- **Bottom-Up or Activity-based methods**: These methods (also called micro-estimation and activity-based techniques) start by decomposing the project deliverables in the Work Breakdown Structure (WBS) into tasks that can be estimated individually. The final effort is simply the sum of the effort required for each task.

Estimation methods can also be classified by the way the estimated variable is calculated:

- **Based on Expert Judgment**: Judgment based estimation methods rely on the experience of the estimator to identify commonalities and patterns between past projects and the one that needs to be estimated. Then variables are estimated based on the expert’s assessment of size and complexity differences.

- **Parametric Models**: Parametric models utilize formulas, algorithms, tables and user defined values to model the way product and process variables affect software development effort. Most parametric models are the result of the analysis of statistically significant project metric databases; still those models have to be calibrated to match your specific organization and conditions.

- **Analogy methods**: Analogy based methods use a relevant project as a reference to estimate the variables of the new project.

In future articles we will be reviewing some of these methods in more detail.

**Estimating Embedded Software Projects**

Most literature, tools and benchmarks for software estimation are focused on business applications. This does not mean they cannot be used to develop real-time or embedded software, but there are considerations that have to be taken into account.

**Product complexity**: Software complexity comes from two sources: component complexity and inter-component complexity. When developing embedded or real-time software, complexity can severely affect development productivity. The number interfaces, timing constraints and concurrency make component interaction difficult to model and predict. Also, system complexity can significantly affect component complexity.

**Criticality**: Embedded software often has to deal with mission critical or safety critical applications. In those cases, criticality imposes process requirements that can affect the level of effort, especially if certification by a regulatory agency is required.
Input/Output & Read/Write models: While business applications use hard discs and databases for data storage, embedded software relies on hardware interfaces and limited memory resources (paged RAM, EEPROMs). Most embedded applications simply cannot afford the luxury of database servers and query languages.

Development Environment: Embedded software can be developed in different ways: using model-based tools, high-level languages or assembly language. The development environment and processor type have to be characterized when comparing two projects to ensure you are not comparing apples to oranges.

In general, it can be said that when developing embedded applications, it is particularly important to document the environmental conditions in order to ensure the metrics used as a reference for estimation are relevant to the target project.

A Good Estimate

In the PMBOK® Guide, The Project Management Institute (PMI) tells us that the planning process is a continuous activity that extends throughout the project lifecycle. As part of the planning process, resource estimation must be reviewed and updated constantly. A good estimate will produce accurate results if project conditions behave as originally assumed.

A good estimate starts with the documentation of all assumptions, product and environment conditions. As soon as these conditions change, the project estimate must be updated. Having a checklist of factors that may affect the project outcome will force you to think about the effect of each factor and will allow you to learn from past experience.

A good estimate is the result of a technical process. Often the estimation process is biased by external factors such as budgetary constraints, competition or market circumstances. While these conditions cannot be denied, they must be given the right name: if your budget is 20% lower than your estimate, you can try to reduce the scope or try to increase productivity or lower your profitability. But you can’t magically lower your effort estimate by 20% and expect that the productivity will increase by that factor.

A good estimate sets the right expectations from your customer and your team. Unrealistic expectations will create either a stressful environment or will not challenge your team to give their best effort.

In a series of follow-up articles, we will be reviewing these methods in more detail in order to provide you with a tool set that can help you produce good estimates.

---
