FUNCTION POINT FOCUS: Requirements are (the Size of) the Problem

by Carol Dekkers, Quality Plus Technologies, Inc.

email: dekkers@qualityplustech.com

Published in IT Metrics Strategies
Vol. III, No. 10, October 1997

Not much has changed in software development since 1988 when David Card stated "Software estimation occurs at an earlier stage in software development than in most other engineering disciplines... most engineers can wait until after preliminary design to estimate product cost"[i] Just two months ago in ITMS, Lawrence Putnam and Ware Myers' made a similar observation. One of their recommendations to improve software estimation stated: "Work toward getting clients to let you size the risks and nail down the requirements before expecting a firm bid".[ii]

Software estimation is the process by which an approximation of work effort to develop software is derived based on an estimation model or equation. This work effort is a function of the software size, development team skills, development language, methodology, and many other factors which vary depending on the estimation model. As the technology independent measure of software size, function points are only dependent on the user requirements. Here we have a fundamental challenge: how can we count function points from user requirements, when we haven't identified the user requirements? This article explores the issue of user requirements throughout the development life cycle, and the challenges they pose to estimation and function point counting if they are ill defined.

A Quick Refresher on Function Points

Frequent readers of this column will recall that Function Points are a quantification of software size based on its functional user requirements. Similar to the square feet derived from a floor plan (requirements), function points are derived from functional user requirements. Function points reflect "what" the application must do and are independent of "how" the software is implemented. (Given the **same set of user requirements**, the function point size will be identical for two applications developed using different programming languages). The biggest advantages of function points over lines-of-code for estimating may not be intuitive: 1. Function points are independent of the development language and technology, lines-of-code are not. 2. Function points can be counted as soon as user requirements have been identified, lines-of-code can only be counted once coding is finished.

The biggest challenge in function point counting lies not with the method or the counting rules, but with the identification of the functional user requirements from which we can do a function point count. Once we have the functional user requirements and basic function point training, it is a straightforward process to derive the function point count.

Most estimating models require an input of the estimated software size, together with many other factors (such as development language, methodology, etc.) to predict the work effort. If there is no value to input for software size due to a lack of requirements, the estimating equation will be incomplete. It is important to note that estimating models are passive and can only respond to the input data they are provided. Without a software size input, developers are as challenged as a house builder trying to quote a price without knowing the square foot size of the house.

A look at requirements gathering at various life cycle stages illustrates the challenges (and opportunities) to obtaining a realistic value of software size.

Preliminary Application Requirements

When a project is first conceived, the requirements may be little more than "notes on a napkin". Not only is it difficult to derive user requirements from napkin scrawls, it is downright impossible to build software to meet them. However, it is at this stage that IT managers are often tasked to transform crude requirements into a realistic estimate of work effort. Famous management phrases such as "Give me a ballpark number" or "I won't hold you to this estimate" are often forgotten once the ballpark is published as *the estimate*.

How can we provide a "ballpark" estimate with some basis in reality at this early stage? Our only hope is to document our assumptions of what the functional user requirements could be, do a high level function point estimate,

and feed the numbers and our assumptions about the project into an estimating tool to derive an estimated work effort.

For example, if we are given a sketchy set of requirements for an employee system that includes worker demographics, job moves and early retirement, we can do a high level estimate of software size in function points. We do this by translating these "requirements" into functional user requirements terms, and then doing a first cut function point count based on a one file model or other approximation method.[1] This provides us with an auditable, factual base for later reference should customers (or others) challenge our work effort estimates. Additionally, when further requirements details are known as we progress into system design, the application size and thus the estimated work effort will increase.

It is helpful to have the original, anticipated set of requirements and the preliminary function point count available to illustrate why the work effort estimate has changed throughout the development project.

Even if we have the luxury of documented functional user requirements and have "frozen" the design (analogous to signing a contract to build a house based on a final floor plan), these requirements may not equal the size of the finished product . Typically the functional size (FP) of an application increases during development for one of the following reasons:

1. Users may have forgotten to mention requirements, or have assumed that requirements that they take for granted will be included when the system is implemented. Missed or overlooked requirements increase the FP count size and can have an exponential impact on the work effort required to deliver the application.

2. Both developers and users explore requirements based on their existing way of doing business. We don't know what queries will arise once we get our new reports, because we haven't yet seen the data. In many development shops these new requirements may be deferred as enhancements while in others, these new requirements become "drop dead" requirements for the current release. This scope creep (or in some cases "scope leap") increases the functional size of the application, and if not managed they can cripple even the best planned project. Sizing the scope creep using function points accomplishes two goals: we can derive new work effort estimates to complete

[1] Contact the author for further information about these and other high level function point estimating methods.

the existing project based on the new size, and we can use it as an "expansion factor" when we are extimating the FP size of future projects.

3. Many "bells and whistles" requirements arise after users view a prototype and realize what functionality is available to them. These requirements count as functional user requirements (and thus count as additional function points) as long as they represent functionality that the user has specifically requested and received. In other words, functionality that the user is willing to pay for. It is important to note that the ease of implementation does not affect the Function Point size because function points are independent of how the requirements are delivered. The impact of adding (or changing) these new requirements however, increases the FP size and translates into a larger work effort.

In all of these cases, it is important to track the increase in FP size, and use the growth rates when estimating future projects. (For example, if the function point size of an application doubles between requirements to implementation, the next time the work effort is estimated on a project, an estimate should also be done initially taking into account the possibility of it too doubling in size). History is a much better predictor of the future than theoretical models.

User Requirements at the Analysis and Design Phases

Seasoned systems professionals know that the further along one proceeds in the development life cycle, the better the requirements are defined. In addition, as requirements are formalized and documented on paper, users introduce further requirements. In the words of Putnam and Myers "Accept the reality that requirements develop over time"[iii]. This poses a challenge as the software size and estimates of work effort are usually calculated at only a few milestones along the development process, not whenever a new requirement is identified. In traditional waterfall development this occurs typically at the end of requirements analysis and at the end of the functional design. (Milestone points for other development approaches vary with the methodology). Once the functional design is complete, it is akin to the system "blueprint" after which time, new requirements and rework cycles impact cost and schedules exponentially.

With this in mind, it is critical that the requirements (and the corresponding software size) at these milestone points be as accurate and as complete as possible. The more complete the requirements are, the easier it is to obtain the software size, and the more accurate the work effort estimate can be. Joint application design (JAD) sessions, user walkthroughs and user cross training about development life cycles are widely documented as valuable tools

to getting the requirements right. User involvement and accountability to define the system functional requirements are as critical in systems development as the involvement of a home buyer in designing a home. To borrow a phrase from Capers Jones "It should be considered professional malpractice" for users to abdicate their responsibilities of defining requirements to system developers.

Once user requirements are documented (and a function point count generated), their completeness should be reviewed and tested prior to estimating the work effort. One test for requirements completeness can be done by comparing the project's function point profile (function point percentage breakdown across the five function types) to the function point profile of similar existing applications. Discrepancies often highlight incomplete requirements.

Impact of incomplete or incorrect requirements

Because function point size is based on the functional user requirements, it is a given that if the requirements are incomplete or incorrect, the function point size will also be incorrect. Users need to know how profound the impact of incorrect or unspecified requirements are on work effort estimates. The cost and effort of rework and redesign attributable to these requirements problems can often exceed the entire cost of developing the system if the requirements were originally correct. Requirements that evolve throughout the project are difficult to estimate in the early stages of system design, and can wreak havoc on the development effort and schedules. New estimates of work effort because of newly discovered or corrected requirements are not a result of function point miscalculations, but rather due to an increase in the software's function point size.

Summary

Software estimation is a complex issue for which many estimation models have been proposed. None of the models, however, can estimate accurately if the user requirements are vague, including those based on function points. Well documented functional user requirements are a pre-requisite to sound application development practices. Their absence impedes not only function point counting, but also accurate software estimation.

It is critical for user management to know that correct and complete user requirements are mandatory pre-requisites to both software sizing and to accurate (work effort) estimating. Without solid user requirements on which to base their estimates, software managers are like builders estimating construction time without knowing the size of the house.

==========

Carol A. Dekkers is the President of Quality Plus Technologies, Inc. a management consulting firm specializing in quality initiatives, software metrics, and process improvement. She is a frequent presenter and trainer at both U.S and international quality and measurement conferences and is credentialled as a Certified Management Consultant (CMC), a Certified Function Point Specialist (CFPS), a professional engineer (Canada) and an Information Systems Professional (ISP). She is the Vice President and formerly the Director of Counting Standards on the International Function Point Users Group (IFPUG) Board of Directors, and is a project editor within the ISO Functional Size Measurement workgroup (ISO/IEC/JTC1/SC7 WG12).

---

[i] David N. Card in *The Role of Measurement in Software Engineering*, July 1988

[ii] Putnam L. and Myers, W., Familiar Metrics Management: "The Software Cost Estimation Problem is Solved", *ITMS* Volume III, No.7, July 1997, p.10.

[iii] Ibid, p.10

