Making the First Cut: Sizing New Technology

How can I possibly estimate the size of something I've never done before?

The modern computing environment poses many challenges. Foremost among them is addressing totally new technologies. Trying to figure out how "big" computer programs are has challenged software engineering since its inception and is further complicated by the aforementioned dynamic nature of technology.

Analysts have traditionally sized systems written in statement-oriented procedural languages expressed largely as text (large stacks of cards or reams of tractor-feed paper). Current technologies now take the form of more abstract representations such as diagrams, objects, spreadsheet cells, database queries, and Graphical User Interface (GUI) widgets.

The secret to making progress in sizing these new environments is to identify the "unit of human thought" in the abstraction being used. Next, the organization must go through a calibration process, starting with projects for which the actual size can be determined in terms of that "unit of human thought". The goal of calibration is to establish productivity as a function of actual size, actual time, and actual effort for completed projects. This newly-established productivity relationship can then be used to fine-tune the sizing process and to forecast time and effort on a new project. Once the project is complete, another iteration of the calibration process can be done. This cycle, repeated numerous times, yields sizing and forecasting methods that exhibit a high degree of accuracy with minimal variation.

From Concept to Countable Entities

Sizing is one of the hardest things a development organization does, and the earlier in the life cycle it is done, the harder it is to do.

Historically, statements (Source Lines of Code or SLOC) have been used for sizing systems, sometimes with poor results due to the difficulty of making the mental leap across the "abstraction chasm" from operational capability to programming language constructs. SLOC, however, is an excellent measure of the "work" done by the development process and is most effectively used in process productivity metrics. Advantages: 1) it can be unambiguously defined for a given language; 2) measuring the size of an existing product is automatable; 3) most of the world's historical data contains SLOC as the sizing measure. Disadvantages: 1) the notion of SLOC becomes ambiguous when dealing with non-textual abstractions; 2) the measure has little meaning to the customer / end user.

Function Points (FP) offer a way of narrowing the "abstraction chasm" by providing a level of abstraction between operational capability and programming language constructs. FP is an excellent measure of the "value" produced by the development process and is most effectively used in "bang for the buck" type metrics. Advantages: 1) the customer / end user can likely relate to the entities being counted; 2) there are networks of people (e.g., IFPUG) dedicated to standardizing and improving the counting process. Disadvantages: 1) FP are limited to application domains for which their countable entities make sense (typically mainframe business applications); 2) the process of counting the number of FP in a finished product is not automatable, in fact many big FP shops do quick-and-dirty estimates, using shortcuts such as "backfiring" (back-calculating FP as a function of language and size in SLOC).

A major qualifier on the use of FP as the sizing measure in a productivity relationship is the fact that FP do not

directly relate to development process “work” and must be scaled as a function of the programming language used (an additional source of complexity and variability in the relationship).

If all this isn’t enough to complicate the selection of sizing measures, consider that many new development methodologies employ abstractions that are neither textual nor do their components fit within the set of FP counting entities.

Case Study

Answering the challenge of the above points, Barbara Bilodeau, metrics manager at SRA International, speaks of a custom sizing model they developed to measure productivity for new fourth generation language (4GL) technology. While the organization had a historical database of third generation projects, new development was predominantly in Oracle Developer/2000 and PowerBuilder. Therefore, it appeared that earlier SLOC measurements didn’t apply.

The organization needed a way to predict size early to provide input to SLIM. It also wished to monitor size throughout the life cycle, and measure the actual, completed size of the product. An additional requirement was using the same unit of measure both to predict and to measure.

SRA’s GUI sizing model uses the concept of “design objects” to identify visible requirements and design items. The complexity of each design object is defined, and a gearing factor for each type and complexity of design object aggregates them to a single unit of measure. Before deciding on the custom sizing model, Bilodeau said they looked at function points. However, they determined that neither their software designers nor the functional experts were trained in the terminology and concepts of function points. Not only did the GUI model have simpler terminology than FPs, but it specifically addressed their particular type of business and system. In addition, the GUI model would directly measure the product. Thus there would be no possibility of variation or error based on the person doing the counting, as there is with FP counting.

The GUI sizing model detail was based on the Oracle Developer/2000 environment. The estimate of the total system size was determined from the screen design, the data design, and a combination of all the design factors. These included screens, tables/ entities, reports, external interfaces, and commercial off the shelf software (COTS) interfaces. The hardest thing in developing the models was to identify a common unit of measure. If they chose SLOC, for example, which they had used for a long time, they had to determine how the measure would apply in a GUI system. In sizing third generation languages, text can be easily measured by SLOC. However, the SLOC challenge for 4GLs is in sizing a visual development environment, which can’t be measured simply in SLOC.

Happily, Oracle Forms provides text export, in the form of generated SLOC. Therefore in developing the models, they exported the applications to text (that is, generated code), ran code counters against the text files, and counted every design object they could think of that went into the application, contributing to size. They looked for trends to find which design objects most affected the generated code size, and defined categories for the complexity of the design objects. Then they calculated an average gearing factor and standard deviations for each complexity category.

According to Bilodeau, they have estimated at least 24 projects using this model, which has changed the way they approach certain projects. For one, they learned to develop and use complexity definitions, rather than relying on “instinct” regarding complexity. In gauging complexity, they depend on someone familiar with the model to challenge the assumptions. In the future, Bilodeau says they will repeat thestudy for environments that allow text export, such as PowerBuilder and Web-based application projects. They will also collect data and

refine estimates based on reports and COTS interfaces, and collect more data on defects. In addition, they intend to study the effect of reuse on GUI models.

The Secret to Success

As the GUI sizing project illustrates, people shouldn’t be afraid to try sizing new environments. The first time out, the techniques won’t be perfect, but they can be refined. Therefore, people should roll up their shirt sleeves and get going.