

Measurement for Successful Projects

Authored by Michael Harris and David Herron

Software Project Managers have many responsibilities ranging from managing customer expectations to directing internal resources towards a successful software build. That said, their easiest path to a quiet life is to design, develop and deploy software on time, on budget and with a high degree of quality. Why, then, do industry publications continue to unveil stories about failed projects and the financial impact they have had on the organization? Of course, not all projects fail. When we do read about successes we learn that these projects were well managed and effectively controlled. So what is missing or what is the gap between project success and project failure?

The one attribute that remains constant in all successful projects is measurement. This is the key to successfully managing a project. Equipped with the right set of measures, software project managers can properly set expectations and maintain greater control over their deliverables. The old adage – ‘you can’t manage what you don’t measure’ is sage advice that all too often gets ignored. But it is not simply the act of measuring that leads to success. It is having the right measures that will provide the information necessary to keep a project on time and on budget.

As we explore how projects are being measured today we will look at measures currently being used and share one key measure that is often missing from a project manager’s tool kit.

To better understand the importance and impact of having the right measures in place to manage a project it is essential that we acknowledge the role of the software project manager in today’s environment. So what challenges do project managers’ face today? The list of responses is of course rather long depending upon each unique situation but there are some common problems or challenges that all PMs have to struggle with. High on that list would be the shifting priorities around managing scope and schedule changes. Even in an ideal situation with everyone cooperating and communicating effectively, changes in scope will occur that have an impact on schedule and cost. From there the software project manager has to focus on managing the customers’ expectations with regard to changes to schedule and cost. As the project progresses through the development lifecycle issues of performance and resource management may serve to further challenge the manager. The list goes on.

So how does the project manager gain better control of their project? We know that all customers want their software solutions delivered quicker (time to market), better (high functional quality) and cheaper (lowest possible cost). Given these parameters, it is understandable that projects tend to measure schedule, cost, quality and product deliverables.

These are perfectly good elements to be measuring. Assuming that these four measures can properly address time to market (schedule), high functional quality (quality and

product deliverable) and lowest possible cost, lets look at how these elements are measured:

Schedules are usually created in the form of an initial timetable with noted milestones. This “planned” schedule is then monitored and actuals are compared to the plan periodically throughout the lifecycle. When the plan and the actuals differ then decisions are made as to the cause and agreed corrections are made to the plan. The plan vs. actual analysis cycle continues on throughout the lifecycle. The initial plan is usually based on the scope of the project. The analysis of how long it will take is usually done using a bottom up or top down approach. Various individuals may weigh in with their levels of experience and an overall schedule is developed that everyone feels comfortable with. Cost is analyzed and tracked in a similar way.

Quality is commonly measured using defect analysis. Throughout the lifecycle defects are collected, analyzed, and corrected. The number of defects found is used as a measure of the quality of the deliverable. In a process mature organization defect analysis includes a measure of defect removal efficiency. Quality may also be measured, in part, through the use of Customer Satisfaction Surveys.

Finally, the functionality being developed and deployed is measured by comparing the original user requirements to the actual functions being provided by the software solution. This most often occurs during testing.

Clearly some organizations have been successful using these forms of measurement. But what about the more common scenario whereby an organization has all of these measures in place and yet the project still comes in late, it is over budget and the quality is lacking. What is missing? Where did they and do they go wrong?

Here is the first moment of truth – Nine out of ten projects that fail have not been properly sized. Size is the key to effectively managing software projects. Size is the missing element from the measures we noted. Here is a simple example showing some measures for a set of four enhancement projects on the same system.

We have the following data points for four projects.

Project	Cost (\$000's)	Quality (Major Defects Released)
PO Special	500	12
Vendor Mods	760	18
Pricing Adj	80	5
Store Sys	990	22

Assume that these projects came in on time and within budget. Which of these projects were successful? All of them? How could you compare them?
Now add a size indicator.

Project	Size	Cost (\$000's)	Unit Cost (\$)	Quality (Major Defects Released)	Defect Density
PO Special	250	500	2000	12	.048
Vendor Mods	765	760	993	18	.023
Pricing Adj	100	80	800	5	.050
Store Sys	1498	990	660	22	.014

Forget about what the measure of size is for the moment (we will address that shortly). Assume for now that size is a measure of the functionality (or unit of value) being delivered to the user. We have calculated a “Unit Cost” based on size and it is expressed as a ‘cost per unit of work’. We have also defect density based upon the number of defects and the size of the deliverable.

The Store Sys. Project has the lowest cost per unit of work and it has the lowest defect density. In contrast, the PO Special project has an extremely high cost per unit of work and an equally poor defect density level. The truth is revealed.

Here is the second moment of truth – size does matter!! Size is the key element that actually adds value to measures that are already in place and being used.

A good sizing measure should have the following attributes. It should be -

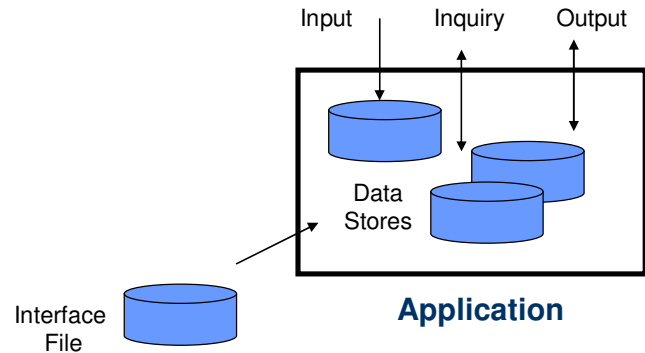
- Meaningful to the software developer and to the user
- Consistent and follow a documented methodology
- Easy to learn and apply
- Accurate and statistically based
- Available when needed early in the lifecycle

If it is possible that cross-industry comparison will be necessary now or in the future, it is also essential that the sizing measure is:

- Defined and recognized as a standard in industry

The most effective sizing technique used today for software is Function Point Analysis. Function Point Analysis (FPA) is an industry accepted sizing technique and has been adopted worldwide. The methodology is supported by a user group, The International Function Point Users Group (IFPUG), which maintains the defined FPA methodology, supports the current counting practices and certifies professional counters. The advantages of FPA are: statistically demonstrable repeatability, speed of implementation, availability of expertise, etc.

The Function Point method is dependent upon the identification of five elements; inputs, outputs, inquiries, internal stores of data and external references to data. The definition of these elements is logical and therefore aligned to the users requirements. The next step in the methodology requires a detailed examination each of the individual elements to determine a 'true value' of its size.



Each element carries a specific 'complexity value as determined by a number of variables. To calculate a total value all of the elements are evaluated, assigned a value and then totaled to derive a total function point size. The size of a project varies from the very small enhancement (50 – 100 function points) to the very large application (1000s of function points).

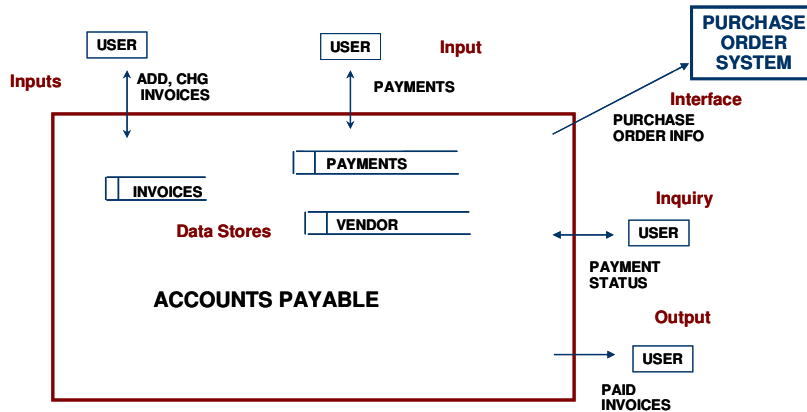
As the project manager begins their project, a function point size is calculated based upon the user's functional requirements. Using the calculated size of the planned project and several other known attributes about the project a project manager can then more realistically estimate the cost, duration and quality of the deliverable. The 'reality' or 'accuracy' of this estimate is based on having available a knowledge base of information relating to past performance of projects based on size, technical environment and a myriad of other attributes. This knowledge base of information is often created by developing a performance baseline of recently completed projects. As projects are completed the baseline is updated with information regarding the project size, cost, duration, technical environment, etc. This database then serves as an ongoing repository of knowledge from which new projects are estimated.

When a project is sized and an estimate is created it may become apparent that the current attributes of the project are such that the predicted cost, duration or quality will not be acceptable. Armed with this information and insight, a project manager can take steps to manage expectations and ideally make recommendations that will result in a more satisfactory outcome.

To demonstrate this approach the following example is presented.

The user is requesting a small series of enhancements to the accounts payable system (see figure below). The request calls for the modification of an input transaction that affects the processing of adding and changing invoices and modifying the input transaction for Payment information. Thus far three inputs have been identified. In addition the request calls for an interface with the Purchase Order system for processing of a payment and so now there is one interface. The status of a Payment is of interest to the user and so they want the ability to view the payment record. This accounts for one inquiry. And there is one output transaction to account for dealing with how invoices are being paid. Finally,

three data stores that are being impacted by our enhanced changes are taking into consideration.



This is a fairly simple series of changes and all of the transactions and data stores are assessed to be of average complexity. Note – we have not shown the complexity evaluation in this example. Using this information we would calculate a functional value as follows.

Components:	Complexity			Total
	Low	Avg.	High	
Data Stores	— X 7	3 X 10	— X 15	30
Interfaces	— X 5	1 X 7	— X 10	7
Inputs	— X 3	3 X 4	— X 6	12
Outputs	— X 4	1 X 5	— X 7	5
Inquiries	— X 3	1 X 4	— X 6	4
				58

Function Point Size

Using the calculated function point size of 58 the project estimator would then scan their baseline database for a project of a similar size. Within a targeted size range that would include a project of 58 function points a variety of other attributes would be used to ‘fine tune’ the results. For example, what is the programming language? What is the technology being used? How skilled is the development team? What processes are being used to design, develop and deploy the solution? These attributes make up what is called a performance profile and allow the project estimator to focus in on the most relevant responses.

Now the project estimator has found a corresponding project that is of similar size and complexity with a matching profile. The historical data base reveals that a project of this

size and with this type of profile has typically resulted in a delivery rate of 10 function points per person month. The estimator can then make the simple calculation that this particular project will require about 5.8 person months of effort. Additional calculations for cost, schedule, resource loading, etc. can then be made.

To further underscore the value of using size in conjunction with an historical database of performance data the Software Engineering Institute at Carnegie Mellon states requirements for good estimating practices to include:

- Corporate historical database
- Structured processes for estimating product size and reuse
- Mechanisms for extrapolating benchmark characteristics of past projects
- Data collection and feedback processes foster correct data interpretation

The functional size measure reflects the business functionality being delivered to the customer. The size value is then available for use in conjunction with other common project measures to quantify and to evaluate software delivery and performance.

- Development Cost per Function Point
- Support Cost per Function Point
- Delivered Defects per Function Point
- Function Points per Staff Month
- Project Delivery Rate (Hours per Function Point)
- Portfolio Function Points supported by one FTE
- Speed of Delivery or Function Points per Elapsed Month

In summary we have learned that -

- Size makes standard project measures more meaningful
- Function Point analysis is an effective sizing technique
- Performance profiles can be developed from a baseline measure
- Size and baseline performance reference points represent best practices