Is There Magic Associated with Software Benchmarks

Donald J. Reifer, Manager
Reifer Consultants LLC

1. Introduction

Benchmarks have proven to be a useful tool for determining whether or not software cost, productivity and quality estimates are realistic and make sense. When used properly, benchmarks can be a helpful tool especially during negotiations because they set realistic expectations. While many may questions the numbers, benchmarks that are produced with rigor and care can serve as a basis for comparison. When done haphazardly, they can be very dangerous.

This paper summarizes over three decades of experience with benchmarks. We both generate and use them in our practice which is aimed at producing numbers that our clients use to assess their relative performance versus peers and determine whether or not they are competitive in their industries. We started producing software cost and productivity benchmarks in the 1980's when we assisted the now defunct ITT (International Telephone & Telegraph) Software Center as they produced their blue book, their compilation of benchmarks for member companies. We next generated cost and productivity benchmarks as a byproduct of our efforts to develop and market a commercial cost model for software named SoftCost. During the early 1990's, we generated a set of benchmarks for the telecommunications industry as a work for hire. As part of this effort, we expanded our benchmarking work to include quality and factor analysis. Currently, we generate software cost, quality and productivity benchmarks as a product and sell them on a subscription basis to a variety of commercial, defense and government clients, all of whom are interested in comparing their performance against numbers that they trust.

2. What are Benchmarks?

According to Wikipedia, benchmarking is the process of comparing one's business processes and performance metrics to industry's best and/or best practices from other industries. Items of interest to those managing software organizations typically measured by benchmarks are cost, quality and productivity. Improvements gleaned from these comparisons led to doing things quicker, better and cheaper. They also permit competitive assessments to be conducted.

Cost, quality and productivity benchmarks are typically based on the following ***four core measures*** each of which is aimed at providing insight into these performance indicators:

- Cost in labor hours
- Time in days
- Quality in terms of defects
- Size in either SLOCs or Function Points

The measures are generated at either the enterprise level or over the course of the project to quantify relative performance using the following three ***indicators***:

- Cost benchmark – cost ($)/ SLOC or function point
- Quality benchmark – number of defects/KSLOC or function point or defect rates in terms of number of defects/time period
- Productivity benchmark – SLOCs or function points/staff-hour or staff-month

While other measures have been used, these are the most common in the literature.

Of course, the scope of each of these measures and contents needs to be pinned down in order to make any sense of them. So do the definitions for indicators because it is easy to excel using them just by changing definitions.

3. Benchmarking Process

The process that we used to develop our benchmarks is illustrated in Figure 1. This process consists of a series of steps each of which is discussed along the rules, checks and conventions adopted to ensure quality in the following paragraphs.

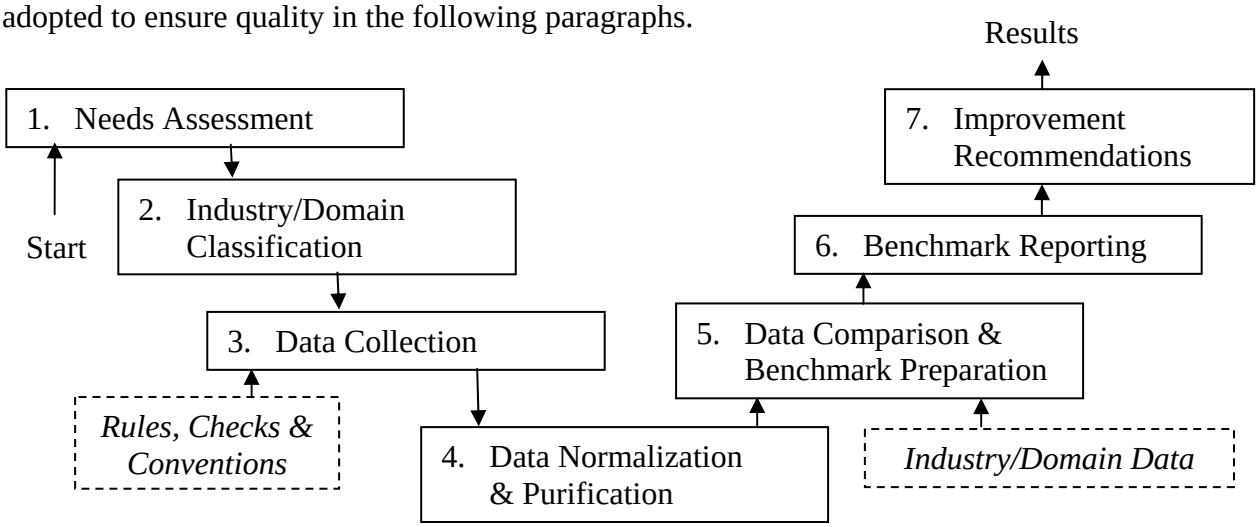

Figure 1: Benchmarking Process

Step 1 - Needs assessment

The first step in the benchmarking process is aimed at determining what is to be benchmarked and what questions need to be answered, by whom, when and how. For general benchmarks, the questions typically answered are similar to the following:

- Am I generating products and services in line with those offered by my main competitors?
- Are my costs, productivity and quality on par with those organizations that are considered "best in class" in my industry?
- How competitive am I versus my rivals?
- Are my investments yielding improvements on par with those being made in my industry?
- Are all of my operating divisions making improvements on par with my expectations?

The results of this step may get more difficult. General answers to questions may not be enough to satisfy senior management's quest for information. For instance, seniors may wish to know how they compare to a specific competitor whom they are bidding against for a specific job.

Step 2 – Industry/domain classification

The second step in the benchmarking process is directed towards determining to whom or what will we compare results. We need to be able to characterize the products and services that are being generated in order to determine which of our benchmarks fits it best. In some cases, generic benchmarks will have to be employed. In others, domain-specific benchmarks can be used. Our experience with benchmarking indicates that domain-specific answers are most meaningful. For example, most of those we have worked with would rather compare their performance on banking application against others in this area rather than some generic number.

This leads us to our first lesson learned when generating benchmarks, which is:

> **Lesson 1** – Domain-specific benchmarking results are preferred by clients because they tend to be more accurate.

Step 3 - Data collection

The third step in the benchmarking process is aimed at collecting the data we need in order to develop answers to the questions posed with our benchmarks. The process we use to complete this all important task is as follows:

- **Establish Framework** – Once we establish a data sharing agreement, we work with our partners to establish a framework that they can use for data collection. This involves identifying what tasks and labor are within scope and what data we need to collect. We may run a pathfinder project through the process to iron out the rough spots.
- **Survey –** Partners next survey each project using a questionnaire that is on-line to capture the data of interest. These data adhere to the framework we established to ensure consistency. Data are captured via questionnaire at the beginning, during the mid-term, at the conclusion and at times that maintenance releases are delivered to the customer.
- **Fact-finding** - Periodically, we visit our partners to review their results. This is typically done bi-annually once the process is up and running and results are being generated.
- **Update** – We also update our process and data collection instruments (questionnaires) periodically. This refresh is accomplished primarily to address issues identified by partners.

The data model we used for collecting our data was built around that developed for the COCOMO-II cost estimating model. We selected COCOMO-II for the following reasons:

- The model is the cost estimating package most used in the world.
- The model is public domain which means all of its mathematics and data collection conventions are available readily either from open publications [1] or the university publisher's web site (http://usc.sunset.edu).
- The partners in our data exchange program were familiar with the model and were collecting data in an attempt to use it to estimate their software costs.

The primary lesson that we have learned using this collection process was as follows:

> **Lesson 2** – It is worth the effort to check the data as it is being submitted. Your results will be statistically better because the data will it be more coherent, complete and consistent.

Step 4 – Data normalization and purification

The fourth and most difficult step in the benchmarking process is directed towards normalizing and purifying the data. Once project data flows, we test it to ensure that it is compatible with what is already in our databases. We first check it for completeness, consistency and accuracy. Often, we find that key inputs are missing and that some of the data just does not make sense when viewed in context. Next, we look at outliers and determine what to do with them. Then, we run a number of statistical tests to determine errors in the data and ranges of variation. We also perform a series of tests to search for biases in the data and determine if our results will change if we include this project in our database. Finally, we enter the data into our databases and test them for homogeneity, goodness of fit and other statistical properties that we believe are important because they provide us added confidence in our results.

The lesson that we have learned using this process on thousands of project data points is:

> **Lesson 3** – Normalizing and purifying the data that you have collected takes a great deal of time and effort because you have to make sure that it hangs together statistically and makes sense from an engineering point-of-view.

Step 5 – Benchmark preparation

The fifth step in the benchmarking process is aimed at preparing the benchmark and checking it for reasonableness. We incrementally add new data to our databases and then test them to determine the impact. If it is within expectations, we incorporate the data. If not, we try to determine why not by performing some form of root cause analysis. Often, it is difficult to ascertain the cause. There are many reasons for this situation. For instance, the data may reflect a new technology, process or technique that when applied results in more than expected cost and/or productivity improvements. As an example, our analysis inferred that agile methods might have resulted in productivity improvements twice as much as what we expected when we first experienced their use. In this case, we need to determine whether the improvements were lasting or were the result of some form of Hawthorne effect [2]. However, root cause analysis indicated that this was a trend rather than a one-time event.

> **Lesson 4** – When data does not fall within expectations, look for the "root" cause. This search may reveal facts that might otherwise be overlooked.

Step 6 – Benchmark reporting

The sixth step in the benchmarking process is directed towards packaging the benchmark so that it can be easily digested by its target audience. These audiences vary as do the questions they pose. Seniors are interested in whether or not their investments are yielding benefits. Middle managers tend to be more interested in costs and the cost of quality. Finally, technical personnel tend to be interested in the details associated with the numbers. For example, they want to understand defect densities and trends and how their work practices impact them. These packaging lessons lead us to the next lessons learned which follows:

> **Lesson 5** – Package information for different audiences using the same core data (effort in hours, size in SLOCs or function points, etc.). For example, balanced scorecards [3] appeal to seniors because a lot of information can be communicated in just a few charts. Middle managers tend to like dashboards because they can visualize the details in a single glance.

Step 7 – Improvement recommendations

The seventh and final step in the benchmarking process is aimed at analyzing the results to glean improvement recommendations from the industry data. Benchmarks provide the baselines that progressive organizations can use to plot an evolutionary path aimed at using successive sets of benchmarking data to make incremental improvements. Having the data provides many advantages. Primary among these is the fact that the data does not lie. You can believe your "best-in-class" cost-wise. But, if the benchmarks do not support this, it would be hard to prove.

Trends are our recommended way to understand how you compare to our baselines. We suggest that you plot your performance using a chart like that illustrated in Figure 2 versus the benchmarks to determine whether or not you are converging on "best-in-class" software performance. If you are not, determine why not by again conducting a root cause analysis.

Engage your stakeholders (customers, users, performers, etc.) in the discussions to get their take on the situation. You will be glad you did because those closest to the situation often have an unobstructed view of cause and effect. Once you have a handle on the situation, act swiftly and smartly. Talking an issue to death does not solve it. Action does. It also reinforces your position because it shows through deeds not words that you are listening to your stakeholders.

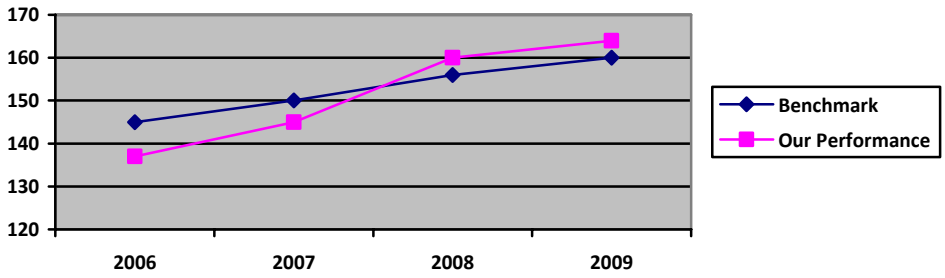

Figure 2: Productivity Performance versus the Benchmarks (SLOC/staff-month)

3. Benchmark Organization

There are many ways to organize benchmarks. We have found it best to summarize findings by industrial sector and application domain because the data we have gathered cluster nicely when plotted in this manner. The three industry sectors we use to classify our benchmarks include:

- **Commercial** – firms within this category include those profit-making organizations that design, develop, field, maintain and/or sustain Information Technology (IT) products and services. In some cases, these organizations may manufacture and sell such systems commercially. In others, they may integrate and use products supplied by others to automate their systems and procedures (production control, finance and accounting, travel, etc.).
- **Defense** – firms within this category include those profit-making organizations that design, develop, field, maintain and/or sustain systems used by the military for a full range of weapons system (fire control, sensor data processing, etc.) and support (medical, supply chain management, etc.) applications.
- **Government** – organizations within this category include government groups that design, develop, field, maintain and/or sustain Information Technology systems for a wide range of applications including support (payroll, travel, etc.), critical infrastructure support (i.e., banking and finance, power grid management, emergency response systems, etc.) and support of sensitive missions (intelligence, homeland defense, etc.).

We have further classified our benchmarks using data from one thousand projects into the twenty-three applications domains listed in Table 1. This list represents an expansion of the domains that we used in earlier classifications such as those that appeared in Crosstalk about a decade ago [4]. The expansion is purely data-driven. We have been successful in getting over eighty organizations to contribute data on the thousand projects noted, none of which is over ten years old. These domains are used in the benchmarks to which we currently sell subscriptions.

We have also found that data within an application domain cluster by software size in SLOC or function points. In other words, projects do not share the same tendencies relative to cost, quality and productivity primarily because of the economies of scale that are available to larger projects. We segment by application size using the following guidelines:

- **Small projects** = those applications under 50 KSLOC or 500 function points
- **Medium projects** = those applications between 50 and 250 KSLOC (500 to 2,500 function points)
- **Large projects** = those applications over 250 KSLOC or 2,500 function points

Sector	Domain	No. of Projects	Notes
Commercial	Automation	65	Mostly factory automation.
(710)	Command and Control	45	
	Information Technology – Banking	47	
	Information Technology – General	37	
	Information Technology - Insurance	25	
	Medical	34	
	Process Control	53	Mainly manufacturing sites.
	Scientific Systems	45	Number crunching usage.
	Software Tools	98	
	Telecommunications	62	
	Test Systems	44	Many remote diagnostics.
	Training/Simulation	31	
	Web Business	124	Primarily client/server sites.
Defense	Military – Airborne	41	Many embedded sensors.
(217)	Military – Ground	45	
	Military – Information Technology	33	
	Military - Medical	31	
	Military – Missile	24	
	Military – Space	26	Both ground and spaceborne.
	Military – Trainers	17	
Government	Information Technology	32	Civilian sector systems.
(73)	Infrastructure	30	
	Sensitive Systems	11	
	TOTAL	1,000	

Table 1: Number of Projects by Domain and Industrial Sector

These experiences lead us to the following additional two lessons learned when developing benchmarks using a large data set:

Lesson 6 – When organizing benchmarks, let the data drive the taxonomy you use. Perform cluster analysis and statistical goodness of fit as your primary tools.

Lesson 7 – A good way to look at the data is by domains. Define these in terms of the applications your users relate to. In addition, use size as a secondary classification.

4. Project Characteristics

The number of projects in each application domain by industrial sector is listed in Table 1. Each project represents a data point that has been normalized using definitions that we have developed

for that purpose. Data compiled was verified as valid via site visits. For completeness, the following characteristics are offered to help you understand what constitutes a project.

- ***Has a Scope*** - Projects start with requirements in terms of a problem-statement, functional specification, feature list or user story. Using requirements as a starting point, software engineers then architect and implement solution using a software development paradigm of their choosing (agile, etc.). They next integrate the solution and test it to ensure requirements are satisfied and then deliver it to an end user for further integration (in the case where software is a component of a larger system that is being delivered) or use (in the case where software is the end-product of the development). Documentation is generated as a by-product of the effort. Support services like version control are included as part of the scope.
- ***Delivers a Product*** – Projects deliver a software product and potentially a number of related services (when the product is sold, services such as training and consulting may be sold along with it on either a subscription or purchase basis). Software developed incrementally may also be delivered incrementally. As a result, a single development effort may generate multiple projects (each increment, version or build that is delivered and sustained is considered a project by this characterization).
- ***Requires Staff and Effort to Accomplish*** – To satisfy their goals, projects require a staff to perform the effort. This effort then must be estimated, budgeted, allocated and managed. Once budgets are approved, the project must be staffed to satisfy these expectations and expenditures must be tracked to ensure that they do not exceed allocations.

5. Project Cost and Productivity

When looking at project costs, we recommend using staff-months of engineering effort as your measure. A staff-month is defined as 152 staff hours of labor. Labor included is scoped to include from the start of software requirements analysis to the completion of software integration and test. It includes the following activities:

- Software requirements analysis
- Software architectural design
- Software implementation
- Software integration and test
- Software task management
- Software documentation
- Software version control
- Software peer reviews
- Software quality engineering
- Integrated product team participation

All directly charged labor is included in this tabulation. For example, labor hours for project leads who charge the project directly are included. In contrast, labor hours for executive management who charge overhead or other accounts are excluded. Things get more complicated when contractors, consultants and part-time employees are involved. Rules will have to be devised to handle these and other labor that does not fit into these definitions.

When looking at productivity, we define it as an economic measure that divides outputs produced by the project by the inputs used to generate them. Inputs are labor in staff-months. For outputs, we prefer to use equivalent new logical source lines of code (ESLOC). This measure permits us to equate modified, reused, auto-generated, and code carried over without modification from one build or increment to another to new source lines. It also allows us to convert function point estimates and actuals to source line counts using industry accepted conversion factors. This leads to the additional lesson learned:

Lesson 8 – While not perfect, equivalent source lines of code (ESLOCs) are a useful size measure. However, to use them effectively, understand how they were derived.

If actual size counts are available, we use them as the next lesson highlights.

Lesson 9 – When calculating cost or productivity, use the actual size of the software not an estimate.

In addition, many organizations determine actual new, modified, reused and carry-over code source line counts by using file comparators that count changes that occur between versions of a release or build. Auto-generated is harder to tally because code to be generated is counted as new code, while the auto-generated code is handled separately because of rules have to be developed to determine how much code they are equivalent to.

As these discussions highlight, there are many issues that make counting these inputs and outputs hard. These complications motivated us to develop a set of standard counting conventions to operate under. These definitions, which for the most part were developed for use with the COCOMO-II cost model, are summarized as follows:

- The scope of all projects starts with software requirements analysis and finishes with completion of software testing.
  - For commercial systems, the scope extends from approval of project startup until sell-off. Web systems scope extends from product conception to customer sell-off.
  - For defense systems, the scope extends from software requirements review until handoff to the system test bed for hardware and software integration and testing. It includes the specification effort necessary to take the requirements allocated from the systems specification to software and refine them to the point where they detail the software functions, interfaces and performance goals that will be implemented.
- Projects employ a variety of methods and practices that range from simple to sophisticated, depending on the need and the maturity of the software organization involved.
- Effort includes all chargeable engineering, management and support labor in the numbers.
  - It includes programming, task management and support personnel (configuration management, network administration, etc.) who normally charge to project.
  - It typically does not include independent quality assurance, system or operational test, and beta test personnel who charge their time to accounts other than software. It does include the software effort needed to support those who charge these accounts.
  - As an example, the software effort includes version control. However, it does not include the software effort needed to assess the impact of a major change to the system if this is funded through a Program Office account and charged separately.
  - As another example, the software effort includes responding to a quality assurance group's independent assessment of their product if the quality group had a separate charge number which they charged to.
- SLOC is defined by Park [5] in his original work to be a logical source line of code using the conventions published by the Software Engineering Institute (SEI) as altered by current practice. ESLOC are further defined by Reifer in his benchmarking reports to take into account reworked, modified, generated and reused code counts. All SLOC counts adhere to

the SEI counting conventions with the exception of deleted source lines which were included in the counts and were not disregarded as recommended by the referenced standard.

- The average number of hours per staff month was 152 (takes holidays, etc. into account).
- Function point sizes are defined using current International Function Point Users Group (IFPUG) standards and counting guidelines as contained in their current counting practices manual [6].
- Function point sizes were converted to SLOC using ranges for backfiring factors determined by QSM in 2009 based on their and the David Group analysis, on the following web site: http://www.qsm.com/?q=resources/function-point-languages-table/index.html. The nominal value was used for the most part. In cases where COCOMO II files were provided, conversions were made per the recommendations of the Center for Software and Systems Engineering (CSSE) at the University of Southern California (USC) in Los Angeles, CA.
- Projects used many different life cycle models and methodologies. For example, web projects typically followed an agile process and used lightweight methods, while defense projects used more classical methods like the waterfall and incremental development paradigms. Commercial projects mostly used some object-oriented design methodology, while military projects used a broader mix of conventional and object-oriented approaches.
- Projects used a variety of different languages. Many used the Unified Modeling Language (UML) for design. For programming, commercial projects tended to employ C#, Java J2EE, Perl and Visual C, while military projects used predominately C/C++ and Ada for legacy.
- The cost assumed per staff month (SM) of effort varies by sector. It assumes an average labor mix and includes all direct labor costs plus applicable overhead, but not general & administrative (G&A) costs and profit. The mix assumes that the average staff experience across the team also varies by sector. It assumes that staff has the skills and experience needed with the application, methods and tools used to develop the products.
- Many of the organizations surveyed were trying to exploit commercial off-the-shelf, open source and legacy components to reduce the volume of work involved.

These assumptions lead us to develop yet another lesson learned which is summarized below:

Lesson 10 – When looking at benchmarks, understand exactly what the data means because you have to be careful when using the numbers.

6. Software Quality

Gains in productivity and reductions in cost are meaningless without parallel gains being made in quality. When cost and productivity are equal, clients view quality as the discriminator. Quality in this sense relates to the product and is measured in terms of defect densities or rates. Defect densities relate to the number of defects as a function of size in either equivalent KSLOC or function points. Defect rates look at defects as a function of time, either calendar, or test or run-time. Both measures are useful when comparing actuals to benchmarks devised for that purpose.

Process quality also is important to measure. Many firms use some form of rating systems (either ISO or CMMI) to appraise the maturity of the processes that they use for both software development and maintenance. Studies have shown that those with mature processes can develop their products quicker, cheaper and better than their competitors [7]. The gains achieved

seem to be derived from that infrastructure that firms put into place when institutionalizing their processes. For example, small projects now can achieve some economies of scale because they can share processes with others. This sharing allows them to avoid reinvention and cut implementation costs to affordable levels.

Typically, cost and productivity are measured in terms of some normative value assigned for quality. For example, a productivity number of 100 ESLOC/staff-month may not be as good as one of 120 ESLOC/staff-month if the quality rating of the former number is fifty percent less than that later number. This leads us to the following next lesson learned pertaining to quality:

Lesson 11 – Quality is king when looking at cost and productivity numbers. To truly understand the benchmarks, relate your numbers to quality norms and rank them accordingly.

7. Is There Magic Associated with Benchmarks?

By now, you are probably coming to the conclusion that there is no magic associated with benchmarks. The numbers that they revolve around can be put to gainful use when developed by a trusted source in a trusted manner. In addition, benchmarks are potent tools when building business cases that justify change [8]. They arm you with the empirical evidence that is needed to show that the changes that are proposed make both good technical and business sense. They help you view the world of business from a best-in-class vantage point and win arguments based on hard data and competitive advantage. This leads us to the following lesson learned:

Lesson 12 – While there is no magic associated with benchmarks, they serve as a useful tool when actual performance can be compared against trusted numbers.

As a warning, you need to realize that no matter what the benchmarking numbers say there will be those who argue against them. This is especially true when they prove your opposition's arguments wrong especially when money is at stake. When faced with opposition, ask them for their evidence. Discount opinion and supposition offered because these are not based on facts. Do not back down when things get tough because the numbers used and the facts portrayed by benchmarks are rarely wrong. This leads to our final lesson learned which is as follows:

Lesson 13 – Trust benchmarks when arguing for change because the picture that they paint relative to performance is rarely wrong.

8. Summary and Conclusions

Cost, quality and productivity benchmarks have been shown to be a valuable tool for assessing competitive performance and making factual comparisons. They allow firms to compare their accomplishments against numbers that have been shown to be sound and trustworthy.

This paper shares thirteen lessons that we have learned during the past few decades when developing and using software cost, quality and productivity benchmarks. Our goal is that those reading this paper will take advantage of these lessons to improve the practice of benchmarking. Our hope is to stimulate more people in the field to use benchmarks as part of their practice. Their value as a performance assessment tool cannot be underestimated.

REFERENCES

[1] B. Boehm, C. Abts, W. Brown, S. Chulani, B. Clark, E. Horowitz, R. Madachy, D. Reifer and B. Steece, ***Software Cost Estimation with COCOMO II***, Prentice-Hall, 2000.

[2] E. Mayo, ***Hawthorne and the Western Electric Company, The Social Problems of an Industrial Civilization***, Routledge, 1949.

[3] R. Kaplan and D. Norton, "Using the Balanced Scorecard as a Strategic Management System, ***Harvard Business Review***, January-February 1996, pp. 4 to 13.

[4] D. Reifer, "Let the Numbers do the Talking," ***Crosstalk***, March 2002, pp. 4 to 8.

[5] R. Park, ***Software Size Measurement: A Framework for Counting Source Statements***, CMU/SEI-92-TR-20, Software Engineering Institute, Pittsburgh, PA, 1992.

[6] International Function Point Users Group (IFPUG), ***Counting Practices Manual***, Release 4.3.1, January 2010.

[7] D. Reifer, ***Impact of Process Maturity on Software Cost, Quality and Productivity***, Special Report, Reifer Consultants, Inc., 2010.

[8] D. Reifer, ***Making the Business Case: Improvement by the Numbers***, Addison-Wesley, 2002.