

Industry Software Cost, Quality and Productivity Benchmarks

Donald J. Reifer, Reifer Consultants, Inc.

Abstract: This article provides software cost, quality and productivity benchmarks in twelve application-oriented domains that readers can use to determine how well their organizations are performing relative to industry averages. In addition to answering common questions raised relative to these benchmarks, the article summarizes the return on investments firms are realizing as they try to harness new technology for a variety of reasons.

Keywords: Software Benchmarks, Software Cost, Software Productivity, Software Quality

Introduction

For years, I have heard my friends complain about the lack of software cost and productivity benchmarks in the open literature. They wanted to use the benchmarks to identify how they stacked up with others within their industry in terms of their performance. In March 2002, I tried to do something about this lack of data by publishing an article which put cost and productivity numbers that I had been compiling for more than two decades into the public domain along with a call for others to join me in publishing numbers [1]. Unfortunately, nobody has answered the call. As a consequence, most cost and productivity numbers continue to remain confidential.

This paper tries to rectify the current state of affairs by providing readers with a revision of my original paper. I have updated my cost and productivity numbers and added quality benchmarks. The new data is important because I believe that cost and productivity should be measured from a quality point of view. It seems counterproductive to me to increase productivity by cutting quality. Yet, there firms that will do anything to improve their numbers.

Needless to say, numbers are important. Besides giving you insight into what the costs and benefits are for a given alternative, numbers can be used to establish industry benchmarks that firms can use to compare their performance with others. When push comes to shove, such numbers are what really matters when you are seeking management approval for funds. "If there isn't some benefit, why spend money?" management asks. Questions like these seem to permeate the atmosphere when management is asked to spend money on software improvements;

- What are the costs and what are the benefits?
- What are the returns on this investment?
- Why should I spend money on this investment rather than on alternatives?
- What would happen if I did nothing (actually my boss's favorite question)?

Getting management to spend money isn't easy. These same managers will use rules of thumb that they have developed over the years to determine if the numbers you are pitching are reasonable. For example, they might surprise you by asking the following question if your numbers don't make sense: "How can you suggest that we will more than triple our productivity if your proposal is accepted?" You need to be prepared to respond to such a question. Having benchmark data that you can use to answer their question could make or break your proposal.

While helpful, the use of such benchmarks is often fraught with danger. The primary reason for this is that people take numbers and use them out of context to justify what they are after. Often, management sees right through such tactics and turns down their request. To avoid failing to the numbers must make sense test, they must be thoroughly defined and be used properly.

Looking at Software Productivity Numbers

Software productivity refers to the ability of an organization to generate outputs (software systems, documentation, etc.) using the inputs or resources it has at its disposal (people, money, equipment, tools, etc.). Using this definition, an organization can increase its productivity by either focusing on reducing the input or increasing the output.

To improve productivity using my definition, organizations could focus on either the input or output strategy. For an input-based productivity improvement strategy, they would focus on increasing workforce productivity through efficiencies gained by inserting better methods, tools, processes, facilities and equipment into the production mix. In contrast, an output-based productivity improvement strategy would place emphasis on increasing the amount of output generated under equivalent circumstances by using components, architecture-centric reuse and product line tactics. Both strategies strive to produce more output using less input.

Within many industries, productivity is commonly expressed as either equivalent source lines of code (ESLOC)/staff month (SM) or as function points (FP)/SM. While other measures may be used, these two tend to predominate especially for medium to large scale projects. Of course, the measures ESLOC, FP and SM must be carefully scoped and consistently defined for these metrics to convey a common meaning (see Notes under Tables for applicable definitions). In addition, the many factors that cause these measures to vary must also be identified. These measures, called cost drivers, must be normalized when defining the terms. Because my firm's databases are primarily ESLOC-based, I use this metric as the basis for my analysis. For those interested in how we handle other measures, we backfire FP data to convert them to ESLOC using the language conversion factors supplied from the International Function Point Users Group (e.g., one FP is expressed as so many lines of C or Java using such backfiring tables).

Table 1 summarizes the results of our analysis of twelve application domains for which my firm has collected data that seem to be of interest to readers. The numbers in Table 1 were derived by taking a 600 project subset of our 2,000+ project experience database and performing statistical analysis. In addition, there are no foreign projects in our database to distort conclusions. The average productivity in our foreign databases for Europe (Belgium, England, Finland, France, Germany, Ireland, Italy, Spain and Sweden) and Asia (India, Korea, Japan and Singapore) are summarized in Table 2. This Table shows how the average productivity for these regions compares to the United States and tries to explain why they are different. Summaries for Australia, Canada and South American numbers are not included in our analysis because the number of completed projects in our databases is too small to be statistically significant. The box under each of the Tables provides notes about the contents. Data were validated using standard statistical means. When anomalies in the data were observed, site visits were made to address concerns. An acronym list is also provided below to define terms used in the Tables.

Acronym List for Tables			
IOC	Initial Operational Capability	PRR	Product Readiness Review
IPT	Integrated Product Team	SAR	Software Acceptance Review
IRR	Internal Requirements Review	SDR	System Design Review
KESLOC	Thousand Source Lines of Code	SETD	Systems Eng. & Technical Direction
LCA	Life Cycle Architecture (review)	SM	Source Lines of Code
LCO	Life Cycle Objectives (review)	SRR	Software Requirements Review
MBASE	Model-Based Software Engineering	STR	Software Test Review
PDR	Preliminary Design Review	UTC	Unit Test Review

Application Domain	Number Projects	Size Range (KESLOC)	Avg. Productivity (ESLOC/SM)	Range (ESLOC/SM)	Example Application
Automation	55	25 to 650	245	120 to 445	Factory automation
Banking	30	55 to 1,000	270	155 to 550	Loan processing, ATM
Command & Control	45	35 to 4,500	225	95 to 350	Command centers
Data Processing	35	20 to 780	330	165 to 500	DB-intensive systems
Environment/Tools	75	15 to 1,200	260	143 to 630	CASE, compilers, etc.
Military -All	125	15 to 2,125	145	45 to 300	See subcategories
▪ Airborne	40	20 to 1,350	105	65 to 250	Embedded sensors
▪ Ground	52	25 to 2,125	195	80 to 300	Combat center
▪ Missile	15	22 to 125	85	52 to 175	GNC system
▪ Space	18	15 to 465	90	45 to 175	Attitude control system
Scientific	35	28 to 790	195	130 to 360	Seismic processing
Telecommunications	50	15 to 1,800	250	175 to 440	Digital switches
Test	35	20 to 800	210	100 to 440	Test equipment, devices
Trainers/Simulations	25	200 to 900	225	143 to 780	Virtual reality simulator
Web Business	65	10 to 270	275	190 to 985	Client/server sites
Other	25	5 to 1,000	182	65 to 481	All others
Totals	600	10 to 4,500		45 to 975	

Table 1: Software Productivity (ESLOC/SM) by Application Domains

Notes for Table 1

- The 600 projects are the most recent projects taken from my database of more than 1,800 projects. These projects were completed within the last seven years by any of 40 organizations (each organization is kept anonymous due to the confidentiality of the data). A project is defined as the delivery of software to system integration. Projects include builds and products that are delivered externally, not internally. Both delivery of a product to market and a build to integration fit this definition.
- The scope of all projects starts with software requirements analysis and finishes with completion of software testing.
 - For military systems, the scope extends from software requirements review until handoff to the system test bed for hardware and software integration and testing.
 - For banking and ADP systems, the scope extends from approval of project startup until sell-off.
 - For web systems, the scope extends from product conception to customer sell-off.
- Projects employ a variety of methods and practices ranging from simple to sophisticated.
- Analysis includes all chargeable engineering, management and support labor in the numbers.
 - It includes programming, task management and support personnel who normally charge to project.
 - It does not include quality assurance, system or operational test, and beta test personnel.
- The average number of hours per staff month was 152 (takes holidays, vacation, etc. into account).
- SLOC is defined by Florac and Carleton [2] to be logical source line of code using the conventions published by the Software Engineering Institute in 1993. ESLOC are defined by Boehm to take into account reworked and reused code [3]. All SLOC counts adhere to the SEI counting conventions.
- Function point sizes are defined using current International Function Point Users Group (IFPUG) standards.
- Function point sizes were converted to SLOC using backfiring factors published by IFPUG in 2000, as available on their web site.
- Projects used many different life cycle models and methodologies. For example, web projects typically followed a Rapid Application Development process and used lightweight methods, while military projects used more classical processes and methods. Commercial projects used object-oriented methodology predominantly, while military projects used a broader a mix of conventional and object-oriented approaches.
- Projects used a variety of different languages. For example, web projects employed Java, Perl and Visual C while military projects used predominately C/C++.

	United States	Europe	Asia	
Automation	245	200	310	Use software generation approaches more in Asia (e.g., generate applications using high level scripting languages)
Banking	270	260	300	Use software generation approaches more in Asia (see above for explanation)
Command & Control	225	-	-	
Data Processing	330	310	325	About the same
Environment/Tools	260	300	270	More small startups in Europe, e.g., more flexible and innovative.
Military -All	145	-	-	
▪ Airborne	105	-	-	
▪ Ground	195	174	-	Few projects, little discrimination.
▪ Missile	85	-	-	
▪ Space	90	85	-	Few projects, little discrimination.
Scientific	195	-	-	
Telecommunications	250	240	255	About the same
Test	210	210	200	About the same
Trainers/Simulations	225	-	-	
Web Business	275	225	230	More flexibility to vary from process in U.S., e.g., using XP and lightweight methods
Other	182	200	-	

Table 2: Productivity Comparisons between U.S., Europe and Asia by Application Domain

Note:

The datasets involved in Table 2 for Asia and Europe are much smaller than those compiled in the United States.

What about Cost?

While cost and productivity are related, they should be considered separately. The reason for this is simple to understand, cost and productivity concerns focus on different factors. For example, cost is very sensitive to labor and burden rate factors. The numbers can be easily manipulated by varying rates. In contrast, organizations trying to improve productivity focus on efficiencies that arise from automation, teamwork and improvement of their process infrastructure. For productivity, the focus is on getting more output with a standard input.

When analyzing our database, assuming rates are normalized, we find that software cost tends to be related to both labor costs and language level. I used \$15,000 as the standard cost for a staff-month (SM) of effort to reflect the average cost of labor across industries exclusive of profit and general and administrative charges, as applicable. Of course, labor rates vary greatly by industry and \$15,000 may not be enough in some (aerospace, telecommunications, etc.) and is too much in others (data processing, web developments, etc.). Language levels are a function of the methods, tools, and language technology used to develop the software product. For simplicity, we define the following three language levels for the purpose of expressing relative cost [4]:

- **Third Generation Languages (3GL)** – A high level procedural programming languages like Basic, C, C++, Java and/or Pascal designed to facilitate the development of software products by programming professionals.
- **Fourth Generation Languages (4GL)** – A programming language close to human languages that makes computing power available to non-programmers and is employed typically by users to access databases, generate scripts, support financial modeling and/or generate reports (Access, PERL, Sequel, etc.).

- **Fifth Generation Languages (5GL)** - A programming language that incorporates the concepts of knowledge-based systems, visualization, animation and/or natural language processing to facilitate the generation of software by a variety of professionals, some of which are outside of the programming arena..

Table 3 shows the dollar cost per ESLOC by application domain that we have developed to serve as benchmarks. The cost numbers vary widely and are sensitive to size, team experience and many factors. Although the Table shows that the cost/ESLOC goes down as language level goes up, this is not true. Firms using fourth and fifth level languages typically use them as part of a mix. The effects illustrated in the Table are therefore the result of using languages from more than one language level each selected to perform the job it was designed to accomplish.

Application Domain	3GL	4GL	5GL	Norm	Notes
Automation	50	*	*	50	Must implement ladder nets.
Banking	40	30	*	35	
Command & Control	85	*	*	85	
Data Processing	40	30	*	35	Many have moved from COBOL to Java and visual languages
Environment/Tools	40	25	*	35	
Military -All	145	*	*	145	While most still use 3G, many have moved to object-oriented languages. Ada on the decline.
▪ Airborne	200	*	*	200	
▪ Ground	90	*	*	90	
▪ Missile	225	*	*	225	
▪ Space	210	*	*	210	
Scientific	90	*	*	90	
Telecommunications	75	50	*	75	Most continue to use C/C++ and Unix
Test	50	*	*	50	
Trainers/Simulations	65	*	*	65	
Web Business	50	35	25	45	Most use Java, HTML, PERL, etc.

Table 3: Software Cost (\$/ESLOC) by Predominate Language Level by Application Domain

Notes for Table 3

- Dollars used to determine cost are assumed to be constant year 2003 dollars.
- The cost assumed per staff month (SM) of effort of \$15,000 assumes an average labor mix and includes all direct labor costs plus applicable overhead. The mix assumed that the average staff experience across the team in the application domain was three to five years. It assumed that the staff was moderately skilled and experienced with the application and languages, methods and tools used to develop the products.
- The scope of the estimate extended from software requirements analysis through completion of software integration and test.
- The scope of the labor charges included all of the professionals **directly** involved in software development. The people involved included software engineers, programmers, task managers and support personnel.
- Many of the organizations reporting were trying to exploit commercial off-the-shelf packages and legacy systems to reduce the volume of work involved.
- The military organizations reporting were all CMM Level 3 or greater, while most commercial firms were not [5]. However, the processes that these organizations used were mostly ISO certified.
- Finally, most projects used more than one language on their projects. The Table shows the cost per SLOC for the predominate language. However, the reader is cautioned that this metric may be misleading when used out of context.

How is this Effort Distributed?

Distribution of effort and schedule is a function of the life-cycle paradigm or model selected for the project. For the following three popular life-cycle models, the allocations of effort and duration are shown in Tables 4, 5 and 6:

- Waterfall [6],
- Model-Based Software Engineering (MBASE) [7],
- Rational Unified Process [8].

Formats used for the Tables vary widely because the numbers are based on slightly modified public references. In some cases, the allocations of effort and duration are shown as ranges. In other cases, they are normalized so that they sum to 100 percent. In yet other cases, the sums are not normalized and therefore are equal to more than 100 percent.

Tables 4 and 6 clearly show the effort and duration to perform a particular activity excluding requirements synthesis (inception) and system test tasks (transition). Table 6 reflects the effort and duration to perform tasks that are part of the Rational Unified Process (RUP). It is interesting to note that both the MBASE and RUP paradigms can be tailored to support agile methods and extreme programming practices [9]. Do not infer that it takes MBASE 18 percent longer than RUP to do equivalent tasks from these Tables. That is not the case because different life cycles embody different activities that make comparisons between them difficult and misleading. If you are interested in more detailed comparisons between life cycles, see the approaches outlined in [6], which provides the most complete coverage that I have seen to date.

These allocation tables are interesting because they tell us that software people do much more work than what is normally considered by most to be software development. This workload typically starts with requirements analysis and ends with software integration and test. They get involved in requirements analysis (averages 7 percent additional effort and 16 to 24 percent more time) and system integration and test support (12 percent more effort and 12.5 percent more time). As you would expect, component design, coding and testing are just a small part of the overall effort and overall schedule.

Phase (end points)	Effort %	Duration %
Plans and Requirements (SDR to SRR)	7 (2 to 15)	16 to 24 (2 to 30)
Product Design (SRR to PDR)	17	24 to 28
Software Development (PDR to UTC)	52 to 64	40 to 56
Software Integration and Test (UTC to STR)	19 to 31	20 to 32
Transition (STR to SAR)	12 (0 to 30)	12.5 (0 to 20)
Total	107 to 131	116 to 155

Table 4: Waterfall Paradigm Effort and Duration Allocations

Phase (end points)	Effort %	Duration %
Inception (IRR to LCO)	6 (2 to 15)	12.5 (2 to 30)
Elaboration (LCO to LCA)	24 (20 to 28)	37.5 (33 to 42)
Construction (LCA to IOC)	76 (72 to 80)	62.5 (58 to 67)
Transition (IOC to PRR)	12 (0 to 20)	12.5 (0 to 20)
Totals	118	125

Table 5: MBASE Paradigm Effort and Duration Allocations

Phase (end points)	Effort %	Duration %
Inception (IRR to LCO)	5	10
Elaboration (LCO to LCA)	20	30
Construction (LCA to IOC)	65	50
Transition (IOC to PRR)	10	10
Totals	100	100

Table 6: Rational Unified Process (RUP) Effort and Duration Allocations

What About the Other Support Costs?

In addition to development, software organizations spend a lot of time supporting other organizations. For example, they are often called upon during system test to fix systems or hardware problems discovered during bench, system and/or operational testing. That's where software shines. It is used to fix problems that would be difficult to resolve via any other means.

There is a great deal of controversy over how much additional effort is required to provide needed software support. Based on experience [10], Table 7 illustrates the extra effort consumed in support of others expressed as an average and a range. The items in this Table are not part of the scope of the effort involved in software development per my and others' definitions [6]. To use the percentages in this Table, you would decrease the productivity or increase the cost appropriately. For example, my life cycle with requirements analysis, not synthesis. This means that the software contribution to the IPT in a military project allocating requirements to software would not be within the current scope. This is pretty standard practice in large military software organizations when systems engineering is tasked to allocate requirements to hardware, software and procedures using a systems specification and operational concept document as their basis. To take the nominal 10% effort into account, you could divide the productivity figure or inflate the cost figure in my Tables accordingly.

It should be noted that many of these additional efforts are application domain specific. For example, Independent Verification and Validation (IV&V) and System Engineering Technical Direction (SE/TD) represent activities conducted in support of military contracts. Because independent teams get involved in oversight and testing, additional work is required on the part of the development contractor. Many times this is true because these third-party contractors feel that they must find something wrong, else their worth can not be justified. Yet, such teams add value when properly staffed and directed towards contributing to release of the software product. The trick in keeping IV&V and SE/TD costs down is to make sure that efforts are directed at making the product better, not finding problems just for the sake of justifying one's existence.

For commercial jobs, alpha and beta testing requires analogous additional support. Development staff must be allocated to address problems identified by testers and an elaborate test management scheme must be employed by the developers to avoid reworking fixes that have already been made. Teams must be coordinated to minimize communications problems. In other words, a lot of extra effort is involved. Many ask me why military projects appear so costly and unproductive. The reason for this is simple. When warfighter lives are at stake, commercial quality is not good enough. As we will show when we look at software quality, the added cost proves worthwhile because the latent defect rates in military systems are much lower than those in their commercial counterparts.

Support Cost Category	Effort (% of software development costs)	Notes
Requirements synthesis and IPT participation	10 (6 to 18)	Participation in system definition and specification activities
System integration and test	30 (0 to 100)	Supporting problem resolution activities as the system is integrated and tested
Repackaging documentation per customer requirements	10 (0 to 20)	Repackaging required to meet some customer preference instead of following normal best practice
Formal configuration management	5 (4 to 6)	Support for system level CM boards and activities
Independent software quality assurance	5 (4 to 6)	Quality audits done by an independent organization
Beta test management support	6 (4 to 10)	Development organization manages bug fixes as they are discovered by outside organizations during beta testing activities
Independent Verification & Validation (IV&V) or SETD support contractor	6 (4 to 10)	Development organization support to an independent contractor hired to provide technical oversight and direction
Subcontract management	10 (8 to 16)	Providing technical oversight to those subcontractors tasked with developing all or part of the software deliverables
Total	82 (30 to 186)	Depending on what tasks are applicable

Table 7: Typical Software Support Costs

Notes for Table 7

- Percentage allocations for effort are shown as both an average and a range in parentheses.
- Version control of software deliverables is included in our normal numbers as is normal quality assurance efforts. The additional effort shown in Table 7 refers to formal activities undertaken at the project-level that the software organization must support (e.g., project-level change control boards).
- If large amounts of commercial off-the-shelf software will be used, additional effort must be provided by the software group to support evaluation, glue code development, tailoring and other life-cycle tasks.
- Percentage expressed uses software development costs as their basis. For example, the chart states that the cost on average quoted for a software job should be increased as much as 186 percent of the base (the base plus 86 percent) to cover the additional costs associated with the activities identified in Table 7 should all of these costs be applicable.
- These support costs can vary greatly based on the manner in which the organization is organized and the work allocated. For example, some firms combine their CM and SQA support in a single Product Assurance organization to take advantage of economies of scale.
- System integration and test support does not include operational test and evaluation for military projects and beta testing for commercial projects. These activities can require even more support than identified depending on the application domain. For example, I have seen aircraft projects that have gone through extensive flight testing burn thousands of hours of software support during these time periods.
- Many firms have adopted defined processes at the organizational level. As such, they are rated as at least a CMM Level 3 organization. When true, these firms generate documentation as part of their normal way of doing business. However, customers can ask for different documentation for their users. When this occurs, repackaging costs for the documentation must be added to the costs because these costs represent divergence to the way they conduct their business.

Looking at Software Quality

Achieving high productivity at the expense of quality is the wrong approach. Yet, many organizations have tried to do this in the commercial world at the customer's expense. To prevent this from occurring, we must view productivity from a quality point-of-view. To accomplish this feat, we must view productivity by normalizing the quality of the products generated upon delivery to the field using metrics like so many software errors per thousand ESLOC. While there have been many papers published on quality metrics and measurement, few of them have put such error baselines into the public domain. There are many reasons for this omission. First, there is a great deal of confusion over just what an error is. For example, do documentation discrepancies count the same as code anomalies? Second, what is the scope of an error? For instance, do you count defects throughout the life cycle or just start counting once you enter software integration and test? Do you mix development errors and operations errors within the same count? Third and finally, what can you do with the error data once you've collected it? Can you feed the data into models and use the predictions about error rates to make decisions on whether or not you have tested enough?

Table 8 portrays error rates by application domain. Errors in this Table are software anomalies discovered and fixed during software integration and testing. The scope of the effort involved starts when integration begins and terminates with delivery of the software. Using this definition, requirements, design, coding and repair problems all constitute an error. However, errors discovered during testing and were not repaired when the product was delivered are not counted. They are fixes pending that will be treated as latent defects in the code that will be repaired at a later time during either system test or maintenance.

The term normative error rate is used to communicate that we are using KESLOC rather than KSLOC as our basis. This is important consideration because it allows us to take legacy and reused software into account as we compute our averages. Such definitions are consistent with the pioneering work in error analysis that IBM and Motorola has completed in the area of orthogonal defect classification [11] (see current updates on-line on the IBM web site located at <http://www.research.com/softeng/ODC>).

Application Domain	Number Projects	Error Range (Errors/KESLOC)	Normative Error Rate (Errors/KESLOC)	Notes
Automation	55	2 to 8	5	Factory automation
Banking	30	3 to 10	6	Loan processing, ATM
Command & Control	45	0.5 to 5	1	Command centers
Data Processing	35	2 to 14	8	DB-intensive systems
Environment/Tools	75	5 to 12	8	CASE, compilers, etc.
Military -All	125	0.2 to 3	< 1.0	See subcategories
▪ Airborne	40	0.2 to 1.3	0.5	Embedded sensors
▪ Ground	52	0.5 to 4	0.8	Combat center
▪ Missile	15	0.3 to 1.5	0.5	GNC system
▪ Space	18	0.2 to 0.8	0.4	Attitude control system
Scientific	35	0.9 to 5	2	Seismic processing
Telecommunications	50	3 to 12	6	Digital switches
Test	35	3 to 15	7	Test equipment, devices
Trainers/Simulations	25	2 to 11	6	Virtual reality simulator
Web Business	65	4 to 18	11	Client/server sites
Other	25	2 to 15	7	All others

Table 8: Error Rates upon Delivery by Application Domain

Notes for Table 8

- Errors are anomalies discovered during software integration and testing by comparing actual functionality and behavior to that specified. Requirements, design, code and repair errors are all applicable to the counts in the Table per this definition.
- Error rates in military systems are much smaller than in their commercial counterparts because lives are affected and there can be severe financial consequences when errors occur (a jet crashing into a neighborhood damaging life and property).
- While error rates in commercial products are improving, they are still viewed as too high by consumers. This is especially true when licenses limit the potential legal recourse from damages.
- High error rates typically occur when systems are fielded prematurely. Considerable rework is required to correct anomalies.
- Errors discovered, but not fixed prior to delivery are not included within the counts. They are treated as latent defects that are shipped to the field.
- Errors rates post-delivery tend to be cyclical. As versions are released, error rates soar. As versions mature, error rates stabilize around 1 to 2 errors per KSLOC in systems whose useful life exceeds 5 years. Web business does not fall in this category because such systems have a life which extends at most 2 years before they are replaced.

Because I do not have comparative data for Asia and Europe, I have not done a comparison. The little data that I do have indicates that the error trends in the United States and abroad are in the same ballpark for similar applications domains.

I felt it was important to include the quality data. The reason for this revolves around the assumption that you don't want to improve productivity at the cost of quality. To use this data, I suggest that you compare productivity when similar defect rates are present. This will allow you to compare apples with apples because you would be assuming a quality norm for comparison purposes.

Have We Made Progress During the Past Decade?

It is also interesting to look at productivity, cost and quality trends during the past decade. Based on the data I have analyzed, the normative improvements we are realizing across all industries ranges from 8 to 12 percent a year. When compared to hardware gains through new chipsets, these results are dismal. However, when compared against productivity trends in service industries, software shines. When productivity gains of 4 to 6 percent are reported for service industries nationally, stock markets shoot up and the media goes wild. It can be argued that comparing software improvements to hardware gains is like comparing apples with oranges.

It is also interesting to compare my findings to those generated for DOD by their Cost Analysis Improvement Group (CAIG). My figures for military systems according to DOD reviewers of this article about 50 percent better. I really don't know why. I suspect the reasons deal with scope of the data and definitions. How they collect and validate their data also could be the root cause. It would be interesting to find out why.

Table 9 illustrates the net gain that you would realize in a 500 person shop when you accelerated productivity gains from a nominal 4 percent to 12 percent a year. It assumes that you begin with a nominal benchmark productivity of 200 ESLOC/SM as your starting point. The net result is that you would save about \$3.8 million if your improvement strategy was capable of achieving this gain. Looking at results of process improvement initiatives alone, realizing these gains over a five year period seem reasonable [12][13][14].

	Year 1	Year 2	Year 3	Year 4	Year 5
Current productivity (ESLOC/SM) (4% nominal gain)	208	216	225	234	243
Accelerated gain (12%)		224	251	281	315
Additional number of ESLOC that can be generated via acceleration assuming 500 engineers		4,000	13,000	23,500	36,500
Cost avoidance (\$50/ESLOC)		\$200K	\$650K	\$1,175K	\$1,825K
Cumulative cost avoidance		\$200K	\$850K	\$2,025K	\$3,850K

Table 9: Savings Attributed to Accelerating Productivity from 4 to 12% Annually

Notes for Table 9

- A 4% rate was selected to represent the nominal cost of money borrowed from a bank at the current prime rate.
- The analysis was done using constant year dollars. The returns were not discounted to take the cost of money into account.
- No gains were assumed during the first year of the initiative. This is typical because workforces have to be hired and there are lots of startup costs and issues.
- The compound effect (costs and benefits) of multiple improvement programs being undertaken in parallel was not taken into account. Often, firms embark on initiatives that address many needs in parallel (train the workforce, improve the process, automate the methods, etc.).
- A similar, but more detailed example illustrating many of these points is in [15].

Summary and Conclusions

This article was to provide software cost, quality and productivity benchmarks in twelve application-oriented domains that readers can use to determine how well their organizations are doing relative to industry averages. Besides publishing these numbers, one of my goals was to stimulate others to publish their benchmark data as well. I therefore encourage those who disagree with my numbers to publish theirs. But don't stop there. If you publish, tell the community what your assumptions are, how the numbers were derived, what their source is, and how they were normalized. Just putting numbers out without explaining them would be counterproductive. If you want to help people, give them the details.

I really encourage those of you who do not have numbers to develop them. Throughout my career, I have used numbers to win the battles of the budget, acquire investment dollars, improve my decision-making abilities, and, most importantly, to win management trust. I gained credibility with management at all levels of the enterprise by discussing both the technical and business issues associated with my projects and proposals. I was successful when I emphasized business goals and showed management my ideas made both good business and technical sense. It is not surprising that I rely on the numbers daily. I encourage you to follow suit.

References

- [1] D. J. Reifer, "Let the Numbers Do the Talking," **CrossTalk**, March 2002, pp. 4-8.
- [2] W. A. Florac and A.D. Carleton (Editors), **Measuring the Software Process**, Addison-Wesley, 1999.
- [3] B. W. Boehm, **Software Engineering Economics**, Prentice-Hall, 1981.
- [4] IEEE Software Engineering Standards, **Volume 1: Customer and Terminology Standards**, IEEE, 1999.

- [5] M. C. Paulk, C. V. Weber, B. Curtis and M. B. Chrissis, **The Capability Maturity Model: Guidelines for Improving the Software Process**, Addison-Wesley, 1995.
- [6] B. W. Boehm, C. Abts. A. W. Brown, S. Chulani, B. Clark, E. Horowitz, R. Madachy, D. Reifer and B. Steece, **Software Cost Estimation with COCOMO II**, Prentice-Hall, 2000.
- [7] W. Royce, **Software Project Management: A Unified Framework**, Addison-Wesley, 1998.
- [8] P. Kruchten, **The Rational Unified Process**, Addison-Wesley, 1978
- [9] B. W. Boehm and R. Turner, **Balancing Agility and Discipline: A Guide for the Perplexed**, Addison-Wesley, Fall 2003.
- [10] D. J. Reifer, **A Poor Man's Guide to Estimating Software Costs**, 9th Edition, Reifer Consultants, Inc., 2002.
- [11] R. Chillarage, I. S. Bhandari, J. K. Char, M. J. Halliday, D. S. Moebus, B. K. Ray and M-Y Wong, "Orthogonal Defect Classification – A Concept for In-Process Measurements," **IEEE Transactions on Software Engineering**, Vol. 18, No. 11, November 1992, pp. 943-956.
- [12] D. J. Reifer, D. Walters and A. Chatmon, "The Definitive Paper: Quantifying the Benefits of SPI," **The DoD SoftwareTech**, Vol. 5, No. 4, November 2002, pp. 12-16.
- [13] B. K. Clark, "Quantifying the Effects on Effort of Process Improvement, Software, IEEE Computer Society, November/December 2000, pp. 65-70.
- [14] T. McGibbon, **Final Report: A Business Case for Software Process Improvement**, Data & Analysis Center for Software, Contract F30602-92-C-0158, Kaman Sciences Corp., Utica, NY, 1996.
- [15] D. J. Reifer, **Making the Software Business Case: Improvement by the Numbers**, Addison-Wesley, 2002.
- [16] R. F. Solon and J. Statz, "Benchmarking the ROI for SPI," **The DoD SoftwareTech**, Vol. 5, No. 4, November 2002, pp. 6-11.

About the Author

Donald J. Reifer is a teacher, change agent, consultant, contributor to the fields of software engineering and management and author of the popular IEEE Computer Society Tutorial on Software Management, 6th Edition. He is President of Reifer Consultants, Inc. and serves as a visiting associate at the Center for Software Engineering at the University of Southern California. Contact him at d.reifer@ieee.org.