POSITIVE AND NEGATIVE FACTORS THAT INFLUENCE SOFTWARE PRODUCTIVITY

October 15, 1998
Version 2.0

Abstract

Software productivity is a complex topic with at least 100 known factors that can influence the outcome of software projects. Some of these factors can have a positive influence and raise productivity. There are also a significant number of factors which can have a negative impact and degrade productivity. The influence of the negative productivity factors is not as well covered in the software literature as the influence of the positive factors. This article summarizes the impacts of both positive and negative factors against a background of "average" results derived from 2000 software projects examined between 1993 and 1998.

Capers Jones, Chief Scientist
Artemis Management Systems

Software Productivity Research, Inc.
(An Artemis company)
1 New England Executive Park
Burlington, MA 01803-5005

Phone	781 273-0140 X-102
FAX	781 273-5176
Email	Capers@SPR.com
CompuServe	75430,231
Web	http://www.spr.com

POSITIVE AND NEGATIVE FACTORS
THAT INFLUENCE SOFTWARE PRODUCTIVITY

INTRODUCTION

Software productivity is a complex phenomenon with at least 100 known factors at play that can influence the outcomes of software projects. The SPR knowledge base of roughly 8500 software projects spans a range which runs from a low of 0.13 function points per staff month to a high of about 150 function points per staff month.

Figure 1 shows the approximate distribution of 2000 software development projects from the years 1993 through 1998 from the SPR knowledge base:

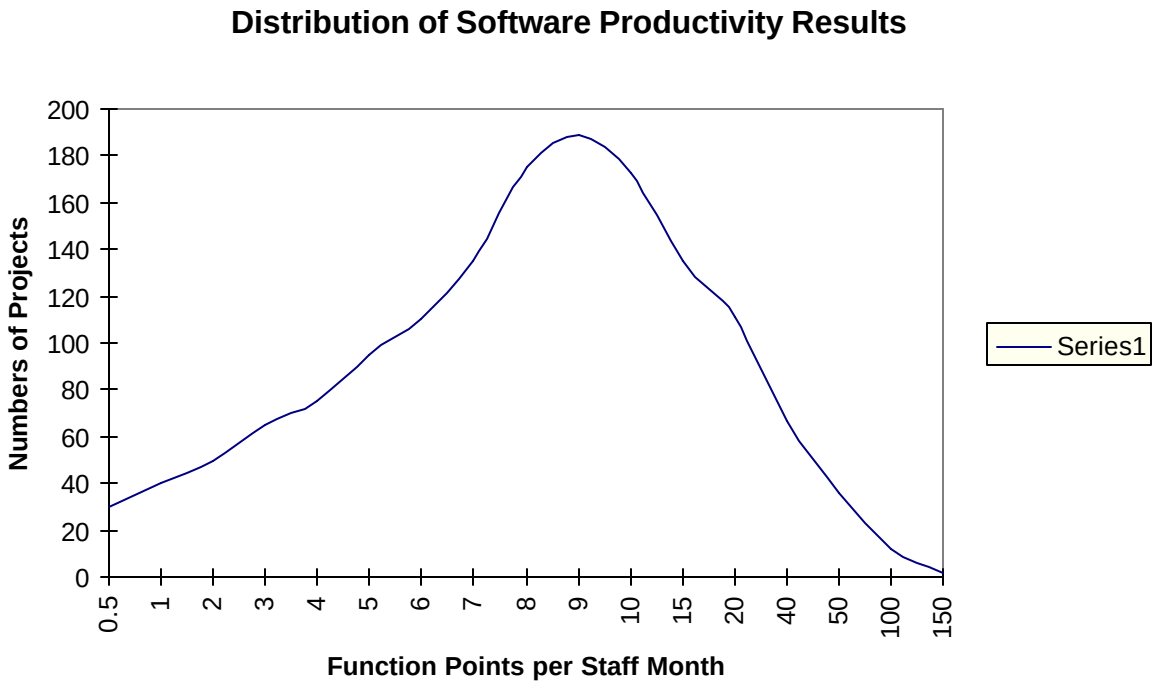

Figure 1: Distribution of Productivity Rates of 2000 Software Projects

As can be seen from figure 1, the range of software productivity results is very broad. This brings up a key question: What causes productivity to be significantly better or worse than average results?

Some of the factors that influence software productivity results are outside the control of the software project team. Although these factors are important, there is little ability to change them. For example:

?? Large systems of more than 10,000 function points in size have lower productivity rates than small projects of less than 100 function points in size.

?? Military software projects that follow the older Department of Defense standards such as DoD 2167 have lower productivity than similar civilian projects because of the huge volume of required paperwork.

There is little value in discussing factors that are outside the control of the software development team. What this article is concerned with are the factors over which there is a measure of control, such as the choice of tools, programming languages, and development processes.

Before discussing the positive and negative factors, it may be of interest to show a relatively typical "average" project such as a COBOL application of 1000 function points in size in order to give a context of why changes can occur in both positive and negative directions:

Table 1: Example of an Average COBOL Application of 1000 Function Point

Application class	Information system
Programming language(s)	COBOL
Size in function points	1000
Source lines per function point	100
Size in source code statements	100000
Average monthly salary	$5,000
Burden rate percent	100%
Fully burdened compensation	$10,000
Nominal work hours per month	160
Effective work hours per month	132

Activity	Assgn. Scope	Prod. Rate	Hours per FP	Staff	Effort Months	Sched. Months	Burdened Cost	Cost per Func. Pt.	Cost per LOC	Percent of Total
Requirements	333	95.00	1.39	3.00	10.53	3.51	$105,263	$105.26	$1.05	8.42%
Prototyping	500	60.00	2.20	2.00	1.67	0.83	$16,667	$16.67	$0.17	1.33%
Design	250	75.00	1.76	4.00	13.33	3.33	$133,333	$133.33	$1.33	10.66%
Design Inspections	200	150.00	0.88	5.00	6.67	1.33	$66,667	$66.67	$0.67	5.33%
Coding	200	25.00	5.28	5.00	40.00	8.00	$400,000	$400.00	$4.00	31.99%
Code inspections	165	150.00	0.88	6.06	6.67	1.10	$66,667	$66.67	$0.67	5.33%
Change control	1000	175.00	0.75	1.00	5.71	5.71	$57,143	$57.14	$0.57	4.57%
Testing	200	60.00	2.20	5.00	16.67	3.33	$166,667	$166.67	$1.67	13.33%
User documents	1000	140.00	0.94	1.00	7.14	7.14	$71,429	$71.43	$0.71	5.71%
Project management	1500	60.00	2.20	0.67	16.67	25.00	$166,667	$166.67	$1.67	13.33%
Net Result	199	8.00	18.49	5.02	125.05	24.93	$1,250,501	$1,250.50	$12.51	100.00%

Since table 1 is being used for illustrative purposes, it simplifies a number of topics and uses rounded data in order to arrive at a net productivity result of exactly 8 function points per staff

month which is a very representative average value for COBOL applications on mainframe computers.

Table 1 also uses a ratio of 100 source code statements per function point for COBOL even though the real-life results for COBOL applications average about 107 statements per function point. The main point to be gained from examining table 1 is that for typical software projects coding may be the largest single activity, but it constitutes only a relatively small percentage of total effort and expense. This means the productivity improvement or degradation can affect some activities more than others, or may affect only one activity.

Factors Which Influence Software Development Productivity

Table 2 shows a number of factors that exert a positive impact on software productivity and raise results higher than average values:

Table 2: Impact of Positive Adjustment Factors on Productivity (Sorted in order of maximum positive impact)

New Development Factors	Plus Range
Reuse of high-quality deliverables	350%
High management experience	65%
High staff experience	55%
Effective methods/process	35%
Effective management tools	30%
Effective technical CASE tools	27%
High level programming languages	24%
Quality estimating tools	19%
Specialist occupations	18%
Effective client participation	18%
Formal cost/schedule estimates	17%
Unpaid overtime	15%
Use of formal inspections	15%
Good office ergonomics	15%
Quality measurement	14%
Low project complexity	13%
Quick response time	12%
Moderate schedule pressure	11%
Productivity measurements	10%
Low requirements creep	9%
Annual training of > 10 days	8%
No geographic separation	8%
High team morale	7%
Hierarchical organization	5%
Sum	800%

The three most influential factors for elevating software productivity are the use of high-quality reusable materials, and the experience levels of both the managers and technical staff in building similar kinds of applications.

Let us now consider some of the factors that can reduce or degrade software productivity, and cause it to lag average values:

Table 3: Impact of Negative Adjustment Factors on Productivity (Sorted in order of maximum negative impact)

New Development Factors	Minus Range
Reuse of poor-quality deliverables	-300%
Management inexperience	-90%
Staff inexperience	-87%
High requirements creep	-77%
Inadequate technical CASE tools	-75%
No use of inspections	-48%
Inadequate management tools	-45%
Ineffective methods/process	-41%
No quality estimation	-40%
High project complexity	-35%
Excessive schedule pressure	-30%
Slow response time	-30%
Crowded office space	-27%
Low level languages	-25%
Geographic separation	-24%
Informal cost/schedule estimates	-22%
Generalist occupations	-15%
No client participation	-13%
No annual training	-12%
No quality measurements	-10%
Matrix organization	-8%
No productivity measurements	-7%
Poor team morale	-6%
No unpaid overtime	0%
Sum	-1067%

What is most interesting about table 3 is that the same factor, software reuse, which can exert the largest positive impact on improving software productivity can also exert the largest negative impact on reducing productivity. How is it possible for software reuse to exert such a large influence in both directions?

The critical difference between the positive and negative influences of software reuse can be expressed in one word: Quality. The positive value of software reuse occurs when the reusable artifacts approach or achieve zero defect levels.

The negative value of software reuse, on the other hand, will occur if the reusable materials are filled with errors or bugs. Imagine the result of reusing a software module in 50 applications only to discover that it contains a number of high-severity errors which trigger massive recalls of every application!

Note that software reuse encompasses much more than just source code. An effective corporate reuse program will include at least these five reusable artifacts:

1. Reusable requirements
2. Reusable designs
3. Reusable source code
4. Reusable test materials
5. Reusable user documentation

In order to gain the optimum positive value from software reuse, then each major software deliverable should include at least 75% reused material, which is certified and approaches zero-defect levels.

Another interesting aspect of table 3 is that the cumulative results are much larger than those of table 2. Essentially this means that it is far easier to make mistakes and degrade productivity than it is to get things right and improve productivity.

In general, there is often a lack of symmetry between the positive influences and the negative influences. For example, a good development process will exert a moderate positive influence on software productivity, but a really bad development process can exert a very severe negative impact on productivity.

Consider a topic unrelated to software for a moment: use of tobacco. Heavy use of tobacco products exerts a severe negative impact on human tissues and causes a number of medical problems. On the other hand, not using tobacco will not make a person healthy, but simply minimizes the risk of becoming unhealthy. Here too there is a lack of symmetry between positive and negative impact. Use of tobacco has a severe negative impact, but not using tobacco has no particular positive impact, as might regular exercise or a diet high in vegetables.

Factors Which Influence Software Maintenance Productivity

The word “maintenance” is highly ambiguous in the software domain. The common meaning for the term “maintenance” includes both defect repairs and also enhancements in response to new requirements. Although this definition is imperfect, it will serve to show the positive and negative factors which influence the modification of existing software applications.

Table 4 illustrates a number of factors which have been found to exert a beneficial positive impact on the work of updating aging applications:

Table 4: Impact of Positive Factors on Maintenance Productivity (Sorted in order of maximum positive impact)

Maintenance Factors	Plus Range
Maintenance specialists	35%
High staff experience	34%
Table-driven variables and data	33%
Low complexity of base code	32%
Year 2000 search engines	30%
Code restructuring tools	29%
Reengineering tools	27%
High level programming languages	25%
Reverse engineering tools	23%
Complexity analysis tools	20%
Defect tracking tools	20%
Year 2000 specialists	20%
Automated change control tools	18%
Unpaid overtime	18%
Quality measurements	16%
Formal base code inspections	15%
Regression test libraries	15%
Excellent response time	12%
Annual training of > 10 days	12%
High management experience	12%
HELP desk automation	12%
No error prone modules	10%
On-line defect reporting	10%
Productivity measurements	8%
Excellent ease of use	7%
User satisfaction measurements	5%
High team morale	5%
Sum	503%

Because software reuse is not a factor in either defect repairs or adding features to existing applications, the overall positive impacts in the maintenance domain are not as strong as those for new development projects.

The top three factors which exert a positive influence are those associated with the use of full-time maintenance specialists, with having extensive experience in the application being updated, and with the use of tables for holding variables and constants rather than embedding them in the source code itself.

Let us now consider some of the factors which exert a negative impact on the work of updating or modifying existing software applications. Note that the top-ranked factor which reduces maintenance productivity, the presence of error-prone modules, is very asymmetrical.

Table 5: Impact of Negative Factors on Maintenance Productivity (Sorted in order of maximum negative impact)

Maintenance Factors	Minus Range
Error prone modules	-50%
Embedded variables and data	-45%
Staff inexperience	-40%
High complexity of base code	-30%
No Year 2000 search engines	-28%
Manual change control methods	-27%
Low level programming languages	-25%
No defect tracking tools	-24%
No year 2000 specialists	-22%
Poor ease of use	-18%
No quality measurements	-18%
No maintenance specialists	-18%
Poor response time	-16%
Management inexperience	-15%
No base code inspections	-15%
No regression test libraries	-15%
No HELP desk automation	-15%
No on-line defect reporting	-12%
No annual training	-10%
No code restructuring tools	-10%
No reengineering tools	-10%
No reverse engineering tools	-10%
No complexity analysis tools	-10%
No productivity measurements	-7%
Poor team morale	-6%
No user satisfaction measurements	-4%
No unpaid overtime	0%
Sum	-500%

Error-prone modules were discovered in the 1960's when IBM began a methodical study of the factors which influenced software maintenance. It was discovered that errors or bugs in IBM software products such as the Information Management System (IMS) data base and the multiple virtual storage or MVS operating system tended to "clump" in a very small number of modules which were extremely buggy indeed.

In the case of MVS, about 38% of customer reported errors were found in only 4% of the modules. In the case of IMS, an even more extreme skew was noted. There were 300 zero-

defect modules out of a total of 425, and 57% of all customer-reported errors were against only 31 modules.

Although IBM first discovered the existence of error-prone modules, they are remarkably common and have been found by dozens of companies and government agencies too. Fortunately, error-prone modules are a completely curable problem, and the usage of formal design and code inspections has the effect of completely immunizing software projects against these troublesome entities.

Patterns of Positive and Negative Factors

My company, Software Productivity Research, Inc. collects quantitative and qualitative data on about 50 to 70 software projects every month, and has been doing so for about 12 years. Software projects and software organizations tend to follow relatively normal bell-shaped curves. There are comparatively few projects and companies that are good in almost every aspect of software, and also few that really bad in almost every aspect of software.

Table 6 illustrates a typical pattern of responses on the part of our clients to a set of nominally 100 questions that we use in our software process assessment studies:

Table 6: Patterns of Software Factors Noted During SPR Studies

	Excellent	Good	Average	Marginal	Poor	Total
Best in Class (top 5%)	15.00%	30.00%	50.00%	4.00%	1.00%	100.00%
Better than average	10.00%	25.00%	50.00%	10.00%	5.00%	100.00%
Average (central 50%)	5.00%	15.00%	50.00%	20.00%	10.00%	100.00%
Worse than average	3.00%	12.00%	50.00%	23.00%	12.00%	100.00%
Worst in Class (bottom 5%)	1.00%	5.00%	50.00%	29.00%	15.00%	100.00%
Average	6.80%	17.40%	50.00%	17.20%	8.60%	100.00%

Note that both "best in class" companies and "worst in class" companies are average in about 50% of all the topics that we examine. However, the "best in class" organizations rank as good or excellent in roughly 45% of the topics, and only lag in about 5% of the topics.

By contrast, the "worst in class" organizations lag in about 44% of all topics, and are only good or excellent in about 6% of the topics.

This pattern of being average in many factors but good or bad in some is not unique to software. Similar patterns can be found in professional sports, in scientific discoveries, in artistic creation, in musical composition, and in many knowledge-based occupations.

The most common pattern which we encounter are projects and companies where the technical work of building software in terms of design and coding are reasonably good, but project management factors and quality control factors are fairly weak. This combination of factors is a reasonable characterization of the information systems domain, which numerically is the most common form of software development.

A variant of information systems projects, those produced by outsource contractors, have a somewhat better chance of having good project management methods than do the other forms of information systems development. However, not all outsource projects use good project management tools and methods either.

For a variety of reasons, the systems software domain has a greater likelihood of having fairly good quality control as well as fairly good development skills. Here too, project management is often the weak link. Quality control is better in the systems software domain because the hardware devices which the software controls (switching systems, computers, aircraft, etc.) need stringent quality control in order to operate.

The military software domain, like the systems software domain, is often characterized by rather good development methods and fairly good quality control methods, but marginal or deficient project management methods.

The large commercial software vendors tend to have better than average software development methods and much better than average maintenance methods. Here too, quality control and project management are the weaknesses noted most often in assessment and benchmark studies.

Among the organizations whose productivity rates are in the top 5% of our client results, there is a very strong tendency for most of these factors to be better than average:

?? Project management tools and methods
?? Quality control tools and methods
?? Maintenance tools and methods
?? Development tools and methods

Conversely, among the clients who bring up the rear in terms of productivity, there is a strong tendency for this pattern to be noted:

?? Poor project management tools and methods
?? Poor quality control tools and methods
?? Poor maintenance tools and methods
?? Adequate development tools and methods

On the whole, the software technical community of analysts and programmers seem to be better trained and equipped for their jobs than do the software project management community.

Summary and Conclusions

Software productivity is a very complex topic. There are many wrong ways to go about building and maintaining software and only a few ways of getting it right. As a result, average productivity and quality levels for software projects are not particularly impressive.

There is a fairly extensive literature on methods that might improve software productivity, although empirical data is somewhat sparse. However, the factors which degrade and reduce software productivity are not explored nor published very often.

This article summarizes a number of factors that have been found to exert either positive or negative impacts on the outcomes of software projects. Few projects are at extreme ends of the productivity spectrum, but it is interesting to study polar opposites since much can be learned from projects which are either total disasters on one hand, or set new records for productivity and quality on the other hand.

ADDITIONAL READINGS ON SOFTWARE FACTORS

Austin, Robert d:; Measuring and Managing Performance in Organizations; Dorset House Press, New York, NY; 1996; ISBN 0-932633-36-6; 216 pages.

Boehm, Barry Dr.; Software Engineering Economics; Prentice Hall, Englewood Cliffs, NJ; 1981; 900 pages.

Bogan, Christopher E. and English, Michael J.; Benchmarking for Best Practices; McGraw Hill, New York, NY; ISBN 0-07-006375-3; 1994; 312 pages.

Jones, Capers; Assessment and Control of Software Risks, Prentice Hall, Englewood Cliffs, NJ; ISBN 0-13-741406-4; 1994; 619 pages.

Jones, Capers, Patterns of Software Systems Failure and Success, International Thomson Computer Press; Boston, MA; 1-850-32804-8; 1995; 292 pages.

Jones, Capers; Applied Software Measurement - 2nd Edition; McGraw Hill, New York; ISBN 0-07-032826-9; 1996; 618 pages.

Jones, Capers, Software Quality - Analysis and Guidelines for Success, International Thomson Computer Press; Boston, MA; 1-85032-867-6; 1997; 492 pages.

Putnam, Lawrence H.; Measures for Excellence -- Reliable Software On Time, Within Budget; Yourdon Press - Prentice Hall, Englewood Cliffs, NJ; ISBN 0-13-567694-0; 1992; 336 pages.

Putnam, Lawrence H and Myers, Ware.; Industrial Strength Software - Effective Management Using Measurement; IEEE Press, Los Alamitos, CA; ISBN 0-8186-7532-2; 1997; 320 pages.

Rubin, Howard; Software Benchmark Studies For 1997; Howard Rubin Associates, Pound Ridge, NY; 1997.