

# POSTIVE AND NEGATIVE INNOVATIONS IN SOFTWARE ENGINEERING

Version 3 – December 24, 2006

## Abstract

The software engineering field has been a fountain of innovation. Ideas and inventions from the software domain have literally changed the world as we know it. Today in 2007 we have the internet, vast data bases of knowledge, sophisticated search engines, and computerized controls of almost all complex operating equipment. The number and importance of innovations from the software industry are equal to any other industry in human history. But we also have viruses and spyware, which rank among the most harmful innovations in human history.

For software development, we have a few proven innovations. The way software is built remains surprisingly primitive. Even in 2007 major software applications are cancelled, overrun their budgets and schedules, and often have hazardously bad quality levels when released.

There have been many attempts to improve software development, but progress has resembled a drunkard's walk. Some attempts have been beneficial, but others have been either ineffective or harmful.

This article puts forth the hypothesis that the main reason for the shortage of positive innovation in software development methods is due to a lack of understanding of the underlying problems of the software development domain. A corollary hypothesis is that lack of understanding of the problems is due to inadequate measurement of quality, productivity, costs, and the factors that affect project outcomes.

Capers Jones, Chief Scientist Emeritus  
Software Productivity Research LLC

Phone           401 789 7662  
Email           cjonesiii@cs.com  
Web             <http://www.spr.com>

Copyright © 2006-2007 by Capers Jones. All Rights Reserved.

## **Positive and Negative Innovation in Software Engineering**

### **Introduction**

There are two kinds of innovations that are important to the software world: product innovations and process innovations. Product innovations involve developing new or improved products that will excite customers. Process innovations, involve developing new or improved methods of development that can shorten development times, reduce costs, or improve quality.

Innovations can be either positive or negative. Positive innovations are those that add value and have clearly defined benefits. Negative innovations are those that make situations worse, or which add to expense levels but not to perceived value.

In the software domain external product innovations and internal process innovations are at differing levels of sophistication. Even in 2007 very sophisticated and complex pieces of software are still constructed by manual methods with an extraordinary labor content and very distressing quality levels.

Another example of an imbalance between product innovations and process innovations can be seen in the migration of technology jobs from the United States to India, China, and other countries with low labor costs. Many sophisticated products designed in the United States are now being manufactured abroad because the U.S. has not been able to introduce internal manufacturing innovations in sufficient quantities to stay cost competitive.

### **External Product Innovation**

The software domain has created scores of innovative products that have changed the way the world does business. Some examples of positive software innovations in alphabetic order include compilers, data base software, embedded software, graphical user interfaces, the internet, medical software, search engines, spreadsheets, web browsers, and word processors.

Unfortunately there have also been negative or harmful innovations. The most harmful of the external innovations from the software domain are computer viruses and spyware. Other negative innovations include pop-up ads, spam, phishing, and browser hijackers. The negative innovations have caused enormous problems and expenses for all users of computers. The negative innovations have also caused the creation of a sub-industry of anti-virus and anti-spyware companies.

Other forms of negative innovation include hacking and theft of corporate records. Identify theft is yet another form of negative innovation, and one which is becoming more and more common.

The term “phishing” has become prominent and refers to highly innovative methods of deceiving computer users into going to fraudulent web sites. The most common reason for phishing attacks is to lure users into revealing financial data, passwords, and confidential information by tricking them into thinking they are actually at the web sites of banks or government agencies when in fact the sites are merely replicas designed to look like the original, but created to route private data into the hands of identity thieves.

At the corporate level, a major negative innovation has been finding clever methods of hiding losses and exaggerating profits, as seen in the case of Enron, WorldCom, Arthur Anderson, Global Crossing and other large corporations. In recent years, negative innovations in finance and accounting have cost many billions of dollars.

Guarding against negative innovation is a major issue in the 21<sup>st</sup> century, and is likely to become even more important.

The measures and metrics for external innovation are standard business measures that include:

- Patents issued to employees
- Invention disclosures
- Technical publications
- Research and development spending
- Morale surveys of technical workers
- Market share
- Market growth
- Profit and revenue growth
- Customer focus group results
- Customer survey results
- Trade show editorial reviews
- Successful completion of government tests
- Loss or gain of research and development jobs
- Loss from stolen corporate data, phishing attacks, viruses, etc.
- Security costs for guarding against viruses, spyware, and hacking
- Costs of compliance with Sarbanes-Oxley legislation

The success of the software world in product innovation can also be measured by the number of billionaires and Fortune 500 companies created as a direct result of these innovations. Apple, Cisco, IBM, Microsoft, Oracle, and SAP are excellent examples of companies built around successful product innovation.

The bankruptcy and collapse of Enron and the convictions of its senior executives is a sober example of the hazards of negative innovation, as are the similar failures of more than a dozen large corporations in recent years.

Another example of the economic costs of negative innovation is the huge expense devoted to security of computer systems. In the modern computer security world of 2007, there is a race going on between the perpetrators of negative innovations and the defenders of data bases and stored information.

### **Internal Process Innovation**

Although the software domain has been remarkably effective in external product innovation, it has not been as successful in process innovation. In fact measures of software productivity, quality, schedules, and failure rates have stayed comparatively flat between 1977 and 2007.

The following list of 30 issues is taken from the 2<sup>nd</sup> edition of the author's book Estimating Software Costs. Although the list was assembled for the 2007 edition, it would have been about the same in 1977 as it is today based on data published at approximate 10 year intervals (Jones 1977; Jones 1986; Jones 1996; Jones 2000; Jones 2006, Jones 2007. See also Brooks 1974; Boehm 1981; Kan 2003; Strassmann 97; and Yourdon 1997):

### **Thirty Software Engineering Issues that have stayed constant for 30 years**

1. Initial requirements are seldom more than 50% complete.
2. Requirements grow at about 2% per calendar month during development.
3. About 20% of initial requirements are delayed until a second release.
4. Finding and fixing bugs is the most expensive software activity.
5. Creating paper documents is the second most expensive software activity.
6. Coding is the third most expensive software activity.
7. Meetings and discussions are the fourth most expensive activity.
8. Most forms of testing are less than 30% efficient in finding bugs.
9. Most forms of testing touch less than 50% of the code being tested.
10. There are more defects in requirements and design than in source code.
11. There are more defects in test cases than in the software itself.
12. Defects in requirements, design, and code average 5.0 per function point.
13. Total defect removal efficiency before release averages only about 85%.
14. About 15% of software defects are delivered to customers.
15. Delivered defects are expensive and cause customer dissatisfaction.
16. About 5% of modules in applications will contain 50% of all defects.
17. About 7% of all defect repairs will accidentally inject new defects.
18. Software reuse is only effective for materials that approach zero defects.
19. About 5% of software outsource contracts end up in litigation.
20. About 35% of projects > 10,000 function points will be cancelled.
21. About 50% of projects > 10,000 function points will be one year late.
22. The failure mode for most cost estimates is to be excessively optimistic.
23. Productivity rates in the U.S. are about 10 function points per staff month.
24. Assignment scopes for development are about 150 function points.
25. Assignment scopes for maintenance are about 750 function points.

26. Development costs about \$1200 per function point in the U.S.
27. Maintenance costs about \$150 per function point per calendar year.
28. After delivery applications grow at about 7% per calendar year during use.
29. Average defect repair rates are about 10 bugs or defects per month.
30. Programmers need about 10 days of annual training to stay current.

There is an interesting dilemma associated with the fact that measurable progress in software productivity and quality has remained essentially flat for 30 years. Dozens of new development approaches have been created over the past 30 year period. Examples of these include Agile development, the capability maturity model (CMM), the capability maturity model integration (CMMI), CASE tools, clean-room development, CRYSTAL development approach, dynamic system development method (DSDM), extreme programming (XP), incremental development, ISO 9000-9004 standards, iterative development, object-oriented development, pattern-based development, personal software process (PSP), rapid application development (RAD), reusability, SCRUM, six-sigma for software, spiral development, team structured process (TSP), total quality management (TQM), and the unified modeling language (UML). With all of these new methods available, why are productivity and quality results in 2007 almost the same as those in 1977?

As of 2007, the software industry has more than 600 programming languages in use. We have more than 40 different methods of designing applications. We have 38 different kinds of size metrics. We have some 26 named development methods. There are about 25 international standards that affect software. There are at least 18 different kinds of testing, and four different kinds of review and inspection method. Yet software quality and productivity levels in 2007 are hardly different from 1977.

Also in 2007 millions of programmers are locked into tasks involving the maintenance and support of millions of aging legacy applications. As time passes, the global percentage of programmers performing maintenance on aging software has steadily risen until it has become the dominant activity of the software world.

The plethora of development methodologies is a sign that none of them have actually addressed or solved all 30 of the problems shown earlier in this article. If any of the methods had solved all 30 of the problems, then it would have gradually become the standard approach for building software.

From carrying out benchmark studies in Fortune 500 companies between 1977 and 2007, a curious phenomenon has been noted. For several years after adoption of a new software development approach productivity and quality levels do tend to improve. But when the same company is revisited after about 10 years, it often happens that the new method has been abandoned and productivity and quality results have declined back to the levels before the improvement program started. In fact one of the reasons for revisiting the same companies is that new management wants to start a new software process improvement program, since data on the earlier improvement programs has vanished or is no longer viewed as relevant.

The fact that corporations may abandon even successful process improvement methods in less than five years, and forget about them in less than 10 years, tends to occur often enough so that the phenomenon should receive additional research. What causes this “corporate memory loss?”

Another kind of innovation problem occurs from time to time in large corporations. It sometimes happens that two or more process improvement programs begin in different operating units at the same time. It often happens that each unit selects a different improvement model. For example one unit might choose the object-oriented approach, another might choose extreme programming (XP), and a third may adopt the six-sigma approach. The result is sometimes a political battle between the units, with each one striving to have its choice adopted as a corporate standard.

As of 2007, some companies are improving productivity and quality but other companies are retrogressing and losing ground. Still other companies are doing both at the same time, with improvements in some locations and regressions in others. The short-term improvements among companies upgrading their development methods are balanced by the regression of companies that are abandoning the same techniques.

There is no definitive answer for why this wave pattern of progress followed by regression occurs, but it may be related to the fact that companies don’t measure software productivity or quality levels with enough detail to prove that improvements are cost justified.

In the absence of accurate measurements, organizations tend to follow the path of least resistance and abandon approaches that require continuous training. In other words, the teams that adopt new approaches eventually retire or change jobs, and their replacements are not brought up to speed in successful methods.

This brings up the related question as to why companies don’t measure software productivity and software quality well enough to prove success? The first problem with accurate measurement is selecting accurate metrics.

In terms of sizing and measurement approaches, the industry is overburdened with more than a dozen function point clones, including backfired function points, Cosmic function points, engineering function points, feature points, Mark II function points, NESMA function points, object points, use-case points, and web-object points. Since there are no conversion rules among all of the disparate function point clones, there is no effective way of comparing productivity and quality between projects measured with different variations.

If you have productivity rates of 10 function points per staff month with IFPUG function points, 10 with Cosmic function points, 10 with Mark II function points, 10 with NESMA function points, 10 with web-object points, and 10 with use case points are these

results the same or different? In terms of real economic productivity, which one is highest? Which one is lowest? As of 2007 there is no good answer.

The only function point method with as many as 10,000 projects measured and 30 years of historical data is that of the International Function Point Users Group (IFPUG). Even here the counting rules changed several times between 1977 and 2007. However conversion rules were established with each change. Enough projects have been measured with this metric to see that national productivity and quality levels have remained fairly constant for more than 30 years (Garmus and Herron 1995).

This is not to say that there are no differences between projects and companies. Indeed the best quality and productivity levels are more than 10 times better than the worst. There is solid evidence that organizations at or higher than level 3 on the CMM have better quality and productivity rates than those at levels 1 and 2. There is also evidence that the Agile methods have improved productivity rates for applications below 1000 function points in size (Jones 2000; Krasner 1997). But at national levels, the averages for both productivity and quality are fairly constant.

Even for the older “lines of code” metric there are too many choices. From reviewing the software literature (IEEE Software, IEEE Computing, IBM Systems Journal, etc.) there was no consistency in the usage of lines of code metrics. About one third of the published data is based on counts of physical lines of code, about one third is based on counts of logical statements, and the remaining third uses “lines of code” without bothering to state whether physical or logical code was counted. The difference in size between physical lines and logical statements can top 500%, so this is too big a difference to ignore. There are no true international standards on source code counting.

If an application written in Objective C has a productivity rate of 750 physical lines of code per staff month while a similar application written in C++ has a productivity rate of 750 logical statements per staff month, are these two equal in terms of real economic productivity? As of 2007 there is no good answer.

Even for lines of code measures that use the same approach, the LOC metric has a major economic flaw in that it penalizes modern high-level languages and gives the false impression that older low-level languages such as assembly are more productive than they really are. This paradox was illustrated in Ed Yourdon’s former American Programmer magazine (Jones 1994) in a study which showed productivity rates measured with both LOC metrics and function point metrics for 10 versions of an application written in 10 different programming languages.

The LOC metrics incorrectly showed the highest productivity rates for the lowest-level languages, and penalized modern object-oriented languages. The reason was because with modern programming languages, more than half of the work of software development goes to paperwork production or non-coding tasks. These tend to act as fixed costs, and cause LOC metrics to be inaccurate as measures of true economic

productivity. Function point metrics, on the other hand, do measure economic productivity and highlight the value of modern languages.

Another aspect of the lack of solid measurements is the fact that selecting a suitable development method is a difficult task. There are scores of competing methods. There are conflicting and ambiguous measurement methods for proving results. This leads to a paucity of empirical data. In the absence of solid data, companies tend to select the “methodology du jour” which may or may not be appropriate for the kinds of applications they build. This brings up the topic of “negative process innovation” or methods that do harm rather than adding value.

It often happens that a specific methodology was created to solve a certain kind of problem for a certain kind of software. For example the Agile approaches and extreme programming (XP) were developed to speed up the development of small projects below about 1000 function points or 100,000 logical source code statements in size. This is the size range where small teams in face to face contact are quite effective.

The Agile methods are not yet effective on large systems of 10,000 function points and 1,000,000 logical source code statements in size. In this large size range there are hundreds of developers, sometime located in different cities or even different continents. Thus the Agile approaches are a positive innovation for small projects, but sometimes negative for large systems.

The capability maturity model (CMM) on the other hand was designed specifically for large systems in the 10,000 function point or 1,000,000 source code statement size range. But the infrastructure that comes with the CMM is too massive for small projects below 1,000 function points in size. The CMM has proven to be a positive innovation for large and complex systems, but due to its extensive overhead, the CMM can be a negative innovation for small and simple programs.

This brings up the question of how should process innovations be measured? Internal innovations are subtle and sometimes difficult to measure. Because they often involve improvements in development techniques compared to prior approaches, it is necessary to have long-range measurements that cover both the “before” and “after” methodologies.

The standard economic definition of productivity is: “goods or services produced per unit of labor or expense.” One of the main goals of internal innovation is to improve economic productivity. Thus a tangible improvement in economic productivity is a useful measure of internal innovation.

In the modern hi-tech world a major component of economic productivity centers on warranty repairs, recalls, and repairing defects in technical products. Therefore it is apparent that quality and reliability are also critical for successful process innovations. Some of the measures and metrics that can highlight internal innovations include:

- Time to market: inception to delivery

- Manufacturing cycle time
- Baseline studies showing improvement over time
- Benchmark studies against similar companies
- Cost of Quality measures
- Six-Sigma Quality measures
- Total cost of ownership measures
- Defect removal efficiency measures
- Scrap and rework during manufacture
- Manufacturing work hours per unit
- Manufacturing cost per unit
- Warehouse and storage cost per unit
- Distribution cost per unit
- Warranty repairs per unit
- Defect reports per unit per time period
- Defect repair rates
- Recalls of defective units
- Morale surveys of manufacturing staff
- Morale surveys of support and maintenance staff
- Litigation alleging defective products
- Government withdrawal of safety certificates
- Internal Quality Assurance reviews
- Inspection reports
- Test reports
- Customer surveys dealing with quality and reliability
- Customer surveys dealing with service and support
- Loss or gain of manufacturing jobs

As can be seen, the measurement of internal innovations includes measures of cost, measures of time to market speed, measures of quality, and measures of customer satisfaction. Success in all of these is necessary.

### **Summary and Conclusions**

The software world has been a major source of exciting product innovations for more than 50 years. Unfortunately, negative innovations such as viruses and spyware have also come to light.

However the software world has continued to build large and complex applications by brute force with a high content of manual labor. Thus it is clear that the software world has not been as successful with internal process innovation as it has been with external product innovation.

A key reason for lack of tangible improvement in software development methods is because most companies have not been effective in measuring software productivity and

quality. This is partly due to the competing and conflicting metrics available, and the lack of conversion rules from one metric to another.

Without solid measures that show the economic benefits of various software methods, companies have trouble selecting methods that are effective. Companies also have trouble with keeping effective methods active. This is due to lack of training of new employees in effective methods as older employees retire or change jobs.

The overall goal of corporate innovation is to have a good balance of both external and internal innovations. New and exciting products need to be coupled with efficient and reliable manufacturing steps. The software community has excelled in producing a stream of external innovations, but lags in the creation of proven internal process innovations.

The software industry needs to be much more cautious than it has been in avoiding the harmful consequences of negative innovation. All companies need to be careful about viruses, spyware, hacking, and corporate malfeasance.

In addition, the software industry needs much better proof of the effectiveness of various software development methods. Since development methods are not panaceas and are not necessarily suitable for every kind of project, the software industry needs better warnings about the limitations of various development approaches.

That brings up the final point that the software industry needs to standardize software metrics and measurement practices. The presence of scores of incompatible metrics without any conversion ratios from metric to metric is itself an example of negative innovation.

## References

- Boehm, Barry Dr.; Software Engineering Economics; Prentice Hall, Englewood Cliffs, NJ; 1981; 900 pages.
- Brooks, Fred; *The Mythical Man-Month*, Addison-Wesley, Reading, Mass., 1974, rev. 1995.
- Garmus, David & Herron, David; Measuring the Software Process: A Practical Guide to Functional Measurement; Prentice Hall, Englewood Cliffs, NJ; 1995.
- Jones, Capers; Program Quality and Programmer Productivity; IBM Technical Report TR 02.764, IBM San Jose, CA; January 1977.
- Jones, Capers; Programming Productivity; McGraw Hill, New York; ISBN 0-07-032811-0; 1986.
- Jones, Capers; "Estimating and Measuring Object-Oriented Software"; American Programmer; 1994.
- Jones, Capers; Applied Software Measurement, 2<sup>nd</sup> edition; McGraw-Hill, New York, NY, 1996; 457 pages.
- Jones, Capers; Software Assessments, Benchmarks, and Best Practices; Addison Wesley Longman, Boston, MA, 2000; 659 pages.
- Jones, Capers; Conflict and Litigation Between Software Clients and Developers; Version 6; Software Productivity Research, Burlington, MA; June 2006; 54 pages.
- Jones, Capers; Estimating Software Costs; McGraw Hill, New York; 1998; 724 pages; ISBN 0-07-913094-1; (2<sup>nd</sup> edition due in Spring of 2007).
- Kan, Stephen H.; Metrics and Models in Software Quality Engineering, 2<sup>nd</sup> edition; Addison Wesley Longman, Boston, MA; ISBN 0-201-72915-6; 2003; 528 pages.

Krasner, Herb; "Accumulating the Body of Evidence for the Payoff of Software Process Improvement – 1997;" Krasner Consulting, Austin, TX.

Strassmann, Paul; The Squandered Computer; Information Economics Press, Stamford, CT; 1997.

Yourdon, Ed; *Death March—The Complete Software Developer's Guide to Surviving "Mission Impossible" Projects*, Prentice Hall PTR, Upper Saddle River, N.J., ISBN 0-13-748310-4, 1997.

## Readings on Related Topics

Ambler, S.; Process Patterns – Building Large-Scale Systems Using Object Technology; Cambridge University Press; SIGS Books; 1998.

Artow, J. & Neustadt, I.; UML and the Unified Process; Addison Wesley, Boston, MA; 2000..

Beck, K; Extreme Programming Explained: Embrace Change; Addison Wesley, Boston, MA; 1999.

Boehm, Barry; A Spiral Model of Software Development and Enhancement; Proceedings of the Int. Workshop on Software Process and Software Environments; ACM Software Engineering Notes, Aug. 1986, pp. 22-42.

Booch Grady, Object Solutions: Managing the Object-Oriented Project; Addison Wesley, Reading, MA; 1995.

Booch, Grady; Jacobsen, Ivar, and Rumbaugh, James; The Unified Modeling Language User Guide; Addison Wesley, Boston, MA; 2<sup>nd</sup> edition 2005.

Capability Maturity Model Integration; Version 1.1; Software Engineering Institute; Carnegie-Mellon Univ.; Pittsburgh, PA; March 2003; <http://www.sei.cmu.edu/cmmi/>

Charette, Robert N.: *Software Engineering Risk Analysis and Management*, McGraw-Hill, New York, ISBN 0-07-010719-X, 1989.  
———: *Application Strategies for Risk Analysis*, McGraw-Hill, New York, ISBN 0-07-010888-9, 1990.

Chidamber, S.R. & Kemerer, C.F.; A Metrics Suite for Object-Oriented Design; IEEE Trans. On Software Engineering; Vol. SE20, No. 6; June 1994; pp. 476-493.

Cockburn, Alistair; Agile Software Development; Addison Wesley, Boston, MA; 2001.

Cohen, D. Lindvall M. & Costa, P. An Introduction to agile methods; Advances in Computers, pp. 1-66; 2004; Elsevier Science, New York.

Cohn, Mike; Agile Estimating and Planning; Prentice Hall PTR, Englewood Cliffs, NJ; 2005; ISBN 0131479415.

Cohn, Mike; User Stories Applied: For Agile Software Development; Addison Wesley, Boston, Ma; 2004; ISBN 0-321-20568-S.

Feature Driven Development; [http://en.wikipedia.org/wiki/Feature\\_Driven\\_Development](http://en.wikipedia.org/wiki/Feature_Driven_Development).

Gack, Gary; Applying Six Sigma to Software Implementation Projects; <http://software.isixsigma.com/library/content/c040915b.asp>.

Gamma, Erich; Helm, Richard; Johnson, Ralph; Vlissides, John; Design Patterns: Elements of Reusable Object Oriented Design; Addison Wesley, Boston MA; 1995.

Garmus, David & Herron, David; Function Point Analysis; Addison Wesley Longman, Boston, MA; 2001; ISBN 0-201-69944-3; 363 pages.

Hallowell, David L.; Six Sigma Software Metrics, Part 1.; <http://software.isixsigma.com/library/content/03910a.asp>.

Highsmith, Jim; Agile Software Development Ecosystems; Addison Wesley, Boston, MA; 2002.

Humphrey, Watts; TSP – Leading a Development Team; Addison Wesley, Boston, MA; 2006.

Humphrey, Watts; Managing the Software Process; Addison Wesley, Reading, MA; 1989.

IFPUG Counting Practices Manual, Release 4, International Function Point Users Group, Westerville, OH; April 1995; 83 pages.

IFPUG; IT Measurement: Practical Advice from the Experts; Addison Wesley Longman, Boston, MA; 2002; 759 pages.

International Organization for Standards; ISO 9000 / ISO 14000; <http://www.iso.org/iso/en/iso9000-14000/index.html>.

- Jeffries, R. et al; Extreme Programming Installed; Addison Wesley, Boston; 2001.
- Jones, Capers; Patterns of Software System Failure and Success; International Thomson Computer Press; Boston, ISBN 1-850-32804-8; 1995.
- Jones, Capers; Software Quality – Analysis and Guidelines for Success; International Thomson Computer Press, Boston, MA; ISBN 1-85032-876-6; 1997; 492 pages.
- Jones, Capers; “Sizing Up Software”; Scientific American, New York, NY, December 1998; Vol. 279 No. 6; pp 104-109.
- Jones, Capers; *Why Flawed Software Projects are not Cancelled in Time*; Cutter IT Journal; Vol. 10, No. 12; December 2003; pp. 12-17.
- Jones, Capers; *Software Project Management Practices: Failure Versus Success*; Crosstalk, Vol. 19, No. 6; June 2006; pp4-8.
- Kemerer, C.F.; “Reliability of Function Point Measurement - A Field Experiment”; Communications of the ACM; Vol. 36; pp 85-97; 1993.
- Larman, Craig & Basili, Victor; *Iterative and Incremental Development – A Brief History*; IEEE Computer Society; June 2003; pp 47-55.
- Love, Tom; Object Lessons; SIGS Books, New York; 1993.
- McConnell; Software Estimating: Demystifying the Black Art; Microsoft Press, 2006
- Mertes, Karen R.; Calibration of the CHECKPOINT Model to the Space and Missile Systems Center (SMC) Software Database (SWDB); Thesis AFIT/GCA/LAS/96S-11, Air Force Institute of Technology (AFIT), Wright Patterson AFB, Ohio; September 1996; 119 pages.
- Mills, H.; Dyer, M. & Linger, R.; *Cleanroom Software Engineering*; IEEE Software; 4, 5 (Sept. 1987); pp. 19-25.
- Park, Robert E. et al; *Software Cost and Schedule Estimating - A Process Improvement Initiative*; Technical Report CMU/SEI 94-SR-03; Software Engineering Institute, Pittsburgh, PA; May 1994.
- Park, Robert E. et al; *Checklists and Criteria for Evaluating the Costs and Schedule Estimating Capabilities of Software Organizations*; Technical Report CMU/SEI 95-SR-005; Software Engineering Institute, Pittsburgh, PA; January 1995.
- Perry, William E.: *Handbook of Diagnosing and Solving Computer Problems*, TAB Books, Blue Ridge Summit, Pa., ISBN 0-8306-9233-9, 1989.
- Pressman, Roger; Software Engineering – A Practitioner’s Approach; McGraw Hill, NY; 6<sup>th</sup> edition; 2005.
- 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.
- Rapid Application Development; [http://en.wikipedia.org/wiki/Rapid\\_application\\_development](http://en.wikipedia.org/wiki/Rapid_application_development)
- Roetzheim, William H. and Beasley, Reyna A.; Best Practices in Software Cost and Schedule Estimation; Prentice Hall PTR, Saddle River, NJ; 1998.
- Stapleton, J.; *DSDM - Dynamic System Development Method in Practice*; Addison Wesley; Boston, MA; 1997.
- Stephens M. & Rosenberg, D.; Extreme Programming Refactored; The Case Against XP; APress L.P., Berkeley, CA; 2003.
- Strassmann, Paul; Information Productivity; Information Economics Press, Stamford, Ct; 1999.
- Strassmann, Paul; Information Payoff; Information Economics Press, Stamford, Ct; 1985.
- Strassmann, Paul; Governance of Information Management: The Concept of an Information Constitution; 2<sup>nd</sup> edition; (eBook); Information Economics Press, Stamford, Ct; 2004.
- Stukes, Sherry, Deshoretz, Jason, Apgar, Henry and Macias, Iona; Air Force Cost Analysis Agency Software Estimating Model Analysis; TR-9545/008-2; Contract F04701-95-D-0003, Task 008; Management Consulting & Research, Inc.; Thousand Oaks, CA 91362; September 30 1996.
- Symons, Charles R.: *Software Sizing and Estimating—Mk II FPA (Function Point Analysis)*, John Wiley & Sons, Chichester, U.K.,

ISBN 0-471-92985-9, 1991. Wellman, Frank, *Software Costing: An Objective Approach to Estimating and Controlling the Cost of Computer Software*, Prentice Hall, Englewood Cliffs, NJ, ISBN 0-138184364, 1992.