Software Process Improvement
(Impacting the Bottom Line by using Powerful "Solutions")

by David F. Rico

Abstract

This paper examines only just a few, but extremely impressive examples of "successful Software Process Improvement (SPI)," a highly controversial and much disputed field.

SPI is the discipline of characterizing, defining, measuring, and improving software management and engineering processes, leading to successful software engineering management, higher product quality, greater product innovation, faster cycle times, and lower development costs, simultaneously.

The case studies, examples, information, and data examined in this paper were the result of a notion called "using powerful solutions." Powerful SPI solutions are examined here and others introduced, in order to lead the way and encourage others that have not been successful with SPI, or have yet to try SPI, to use high leverage strategies as methods of making quantum leaps forward in bottom line organizational performance.

This paper represents a significant departure from traditional SPI methods, in that it simply advises organizations to use universal SPI solutions that are guaranteed to work.

Traditional SPI methods direct unskilled and inexperienced individuals to embark on long and indefinite journeys to invent homegrown and highly individualized solutions, having no chance to succeed.

Introduction

SPI is highly controversial because the technology called "software," our mathematical, engineering, and scientific understanding of it, our ability to manage its development successfully, and the state-of-the-practice are yet in their early infancy. It is software's infancy that results in the exact opposite outcome of which is desired:

- Uncontrollable development.
- Low quality.
- High costs.
- Lack of innovation.

Unfortunately, the overwhelming majority of software development practitioners believe that software development will always be a craft industry, a practice of highly skilled and highly individualized artists, artisans, and artistry. In addition, the majority also believe that software development management is unmeasurable, and thus uncontrollable.

This paper illuminates, introduces, and examines a systematic series of indisputable evidence, examples, and case studies, proving that software and software development management are indeed measurable, and thus extremely manageable and controllable.

Furthermore, this paper represents indisputable evidence that an extremely sound, stable, and scientific understanding of software and software development, indeed does exist, and has existed for some time, nearly three decades.

This paper also asserts the notion that software is truly an engineering discipline, though practiced and taught as a craft.

While this paper is largely devoted to a quantitative examination of history, that is the past, it will offer a highly unique, tantalizing, and prophetic glimpse into the future of software engineering that few have seen. For it is only by examining history that the future can be clearly seen. Ironically, it is often said that the past must be forgotten, in order to create new and innovative computer programs. Maybe that's why software is still in its infancy, because we refuse to learn from the past, in fact we forbid it.

Organization

This paper is organized and structured around eight important topics depicting "successful Software Process Improvement (SPI)":

- Benefactors.
- Alternative SPI Models.
- Powerful SPI "Solutions."
- How much does SPI Cost ???
- How do you measure Quality ???
- Myths & Misconceptions.
- Model Corporate Culture.
- Use Powerful "Solutions."

Benefactors

This section highlights real, important, and impressive SPI results from the likes of Motorola, IBM, Hewlett Packard, Raytheon, and NEC. Most of these organizations are icons of world class organizational performance, product innovation, and more importantly SPI.

- Motorola Capability Maturity Model® (CMM®) Results: Motorola is a world leader in applying Statistical Process Control (SPC) to hardware design and manufacturing processes. Only recently has Motorola's software development management began to match their world renown quality management performance in hardware, electronics, and communications [1]. A Motorola division in Scottsdale, Arizona achieved Level 5 of the Software Engineering Institute's Capability Maturity Model for Software (CMM) [2], in December of 1996 [3]. Motorola has shown that CMM Level 5 organizations perform an order of magnitude better than Level 1 organizations, in terms of productivity, quality, and cycle time.
- Motorola Personal Software Process℠ (PSP℠) Results: Once again, Motorola leads the way with the successful industrial implementation of the Personal Software Process (PSP). A Motorola Division in Boynton Beach, Florida used the PSP to achieve zero defects in use over 18 software projects, removing over 76% of their defects before testing began [4].
- IBM NASA Space Shuttle Program: While Motorola has garnered much of the credit for successful SPI, it is IBM that pioneered the techniques used by Motorola to achieve their success. IBM in Houston, Texas began using the Software Inspection Process [5] in 1986, following the explosion of the Space Shuttle Challenger. Within 3 years IBM had achieved CMM Level 5 and near zero defect levels [6].
- IBM Quality Estimation Accuracy: During the 1970s and 1980s IBM pioneered and perfected a software life cycle reliability model based on the Rayleigh model. IBM had achieved the ability to predict, manage, and deliver a specific software quality target within a tenth of a defect per thousand lines of code, in the worst case [7].
- IBM Defect Prevention Results: Following the publication of Philip Crosby's book, "Quality is Free," an IBM division in Research Triangle Park, North Carolina successfully pioneered, mastered, and used the Software Defect Prevention Process, achieving 50% quality improvement the first time used, and up to 99% quality improvement in other instances, without product appraisal activities, that is Inspection and Test [8].
- Hewlett Packard Software Inspection Process Results: Hewlett Packard [9] has saved more than $350 million dollars in software development expenses from 1989 to 1998 by using the Software Inspection Process [5], pioneered by Michael Fagan of IBM in 1972.
- Raytheon Productivity Improvement: Raytheon has achieved an order of magnitude improvement in software productivity from 1988 to 1996 by using CMM style software process improvement [10].
- NEC Defect Prevention Results: This example is extremely profound. In this example, NEC of Tokyo, Japan achieved a hundred to one defect prevention improvement from 1984 to

® Capability Maturity Model and CMM are registered in the U.S. Patent and Trademark Office.
℠ Personal Software Process and PSP are service marks of Carnegie Mellon University.

1993, corporate wide, by using a software defect prevention process [11]. NEC's approach is a homegrown process based on large-scale participation in quality circles. NEC also uses a sophisticated yet simple method of process improvement through process simplification.

- CMM Level 5 Organizations: While it is still commonly held that CMM Level 5 is a dream state that was never meant to be achieved, at least seven organizations worldwide have achieved CMM Level 5, or something very close to it, dispelling the myth of an unachievable level of perfection:
  - NEC Tokyo (notional) [11].
  - IBM Rochester (notional) [12].
  - IBM Houston [6].
  - Motorola India.
  - Boeing Seattle [13].
  - Motorola Scottsdale [3].
  - Lockheed Owego [14].

Alternative SPI Models

The most frequently asked question is invariably, "what is the formula for successful SPI?" There was a time when the answer to that question was as elusive as the hope of a software engineering discipline, within our lifetime.

A heavy dose of hands on software development, hands on experience with highly structured and measurable processes, listening to the masters, and carefully examining successful case studies, as they emerge in increasing numbers, leads directly to the answer.

"The" answer is to use proven, structured, universal, portable, measurable, and powerful software process "solutions."

Three distinct approaches have emerged from the ashes and radiated from the pedestals of champions:

- Indefinite SPI Models: Indefinite SPI models are not "solutions" at all. Indefinite approaches offer tools to novices in a vain attempt to aid in the invention of new solutions. Examples of indefinite SPI models include, Kaizen [15], ISO 9000 [16], the Experience Factory [17], the Goal Question Metric (GQM) paradigm [18], Total Quality Management (TQM) [19], the CMM [2], and Business Process Reengineering [20]. Lowell Jay Arthur gives an excellent exposition of the fallacy and ineffectiveness of indefinite approaches [21].
- Vertical Process SPI Models: The Vertical Process Strategy has been advocated for many decades in various forms, which involves not the invention of a process, but the exploitation of an existing process technology. This approach involves identifying, selecting, and improving but a single software development process, which is believed to positively affect the bottom line. A conservative favorite among the software engineering community is the Software Configuration Management (SCM) Process. NTT in Tokyo, Japan spent nearly a decade perfecting the Testing Process. While these efforts have yielded impressive results like 2,000 changes successfully managed in 2 years, or 90% test efficiency in 10 years, these processes just aren't powerful enough to positively affect the bottom line. However, the Software Inspection Process has proved to be powerful enough to improve the bottom line, and even directly lead to achieving CMM Level 5 [5, 6, 7, 9, 12, 13, 22, 23, 24, 25]. In fact, the overwhelming majority of successful SPI participants, cite the Software Inspection Process as "the" key to successful SPI. The evidence is overwhelming for proponents of the Vertical Process Strategy.
- Vertical Life Cycle SPI Models: The Vertical Life Cycle Strategy, like the Vertical Process Strategy, involves the exploitation of a proven process technology or solution. But, instead of a single process, it involves the use of an entire life cycle. Fewer organizations have been successful with this approach, because it involves more experience and personal maturity than that of exploiting only a single process. In fact, in the history of software engineering only one organization had been successful with this approach until recently, IBM in Rochester, Minnesota [7, 12, 25, 26, 27]. IBM Rochester created an entire life cycle from scratch, largely based on the Software Inspection Process, but including much more, and actually deployed it

to develop a next generation computer and operating system. IBM Rochester went on to generate billions in revenue, win the Malcolm Baldrige National Quality Award, and become ISO 9000 certified, while the rest of IBM came tumbling down. Watts Humphrey did one better by creating the Personal Software Process (PSP), a scaled down version of what IBM Rochester used, and many are becoming quite successful using this approach [4, 28, 29, 30, 31, 32]. Many more so now will benefit from the PSP as a Vertical Life Cycle Strategy, and the Software Inspection Process as a Vertical Process Strategy will rapidly fade from view.

Powerful SPI "Solutions"

While it has been already established that the Vertical Process Strategy and the Vertical Life Cycle Strategy are powerful SPI "solutions," there are a few more that deserve attention. The first solution, Design Management, is a brand new and rapidly emerging SPI solution that has yet to be fully established and quantified. The next two are a recap of the two most powerful Vertical Life Cycle Strategies. And finally, the last four are a quick review of Vertical Process Strategies that must not be ignored by fledgling SPI enthusiasts:

- Design Management: Design Management is a scientific discipline which involves the quantitative study, comprehension, propagation, and exploitation of proven and functionally valid software designs (and their variations) within vertical domains, product classes, and market sectors. Literature is slowly emerging which examines this SPI approach [33, 34, 35, 36, 37, 38, 39].
- Defect Removal Model Life Cycle: The Defect Removal Model Life Cycle is the process of using a Rayleigh life cycle reliability model in conjunction with the Software Inspection Process to predict and manage software development and software quality management [7, 12, 25, 26, 27].
- Personal Software Process (PSP): The PSP is a precisely defined, measurement intensive, portable, and individualized Defect Removal Model Life Cycle [4, 28, 29, 30, 31, 32].
- Defect Prevention: Defect Prevention is a process of capturing and studying defects and defect trends, with the intention of preventing them from happening again [8, 11, 40, 41]. Defect Prevention popularly takes the form of educating design programmers of common mistakes not to repeat. Defect Prevention more effectively takes the form of eliminating, automating, and simplifying defect prone activities, in order to eliminate defect injection into software code. Unlike product appraisal activities, like the Software Inspection Process, Test, and even parts of the PSP which may cost as much as 90% of the project resources, Defect Prevention costs about 1.5% of project resources and is more effective.
- Defect Classification: Defect Classification is a process of analyzing and precisely categorizing software defects, in order to facilitate the Defect Prevention Process [42, 43, 44]. Objectively studying software defects in highly structured ways, eliminates all subjectivity associated with problems that occur, and enables fast and sharply focused defect prevention and even SPI.
- Statistical Process Control (SPC): SPC is the process of quantitatively determining process capability based upon the standard deviation of a sample of process characteristics [45, 46, 47]. Several samples may be taken to ensure that the process capability measurement is correct. Once the process signature has been determined, it is not necessary to continue sampling, or sample every data point, a common fallacy. If the process signature is acceptable, no further action is required. However, if it is not, a process change may be enacted and the process resampled to determine the effectiveness of the process change. Designing processes with desired defectiveness levels enables the elimination of expensive and inefficient product appraisal.
- Software Inspection Process: The Software Inspection Process is a precisely defined and highly measurable team review of software, designed for optimal defect identification [5, 22, 23, 24]. It is not a design review, but a defect review. Only defects may be noted. The facili-

tator, called a moderator, strictly prohibits discussion of design alternatives, style, or other excursions. In addition to being a good tool for identifying, and thus eliminating software defects less expensively and more effectively than test, it is also an excellent team building, communication enhancing, educational, empowering, and information democratizing tool.

How much Does SPI Cost ???

This paper has examined how to achieve SPI successfully and quickly. This paper has also bestowed the benefits of SPI and SPI "solutions." This paper has also examined the cost reducing, productivity enhancing, and cycle time reducing properties of SPI. Yet, common questions are still asked, such as: "How much does SPI cost?"; "Does SPI cost more than the way business is done now?"; "Are the benefits of SPI worth the investment and trouble?" While this paper has already answered each of these questions in rather great detail, these questions will be directly addressed head on:

- PSP Cost Model: The PSP costs about 50 work days, or 2.5 months to execute producing 10,000 lines of code, and result in zero defects, repeatable project performance, and software engineering professionalism. The cost model used is source lines of code divided by 25. This is a custom cost model derived from a study by the Software Engineering Institute (SEI) [31]. This is more than economical for 10,000 source lines of code.
- Software Inspection Process Cost Model: The Software Inspection Process costs about 472 hours to inspect 10,000 source lines of code by a team of four, or about three weeks in elapsed time. This is a custom cost model derived from Software Inspection Process experience and literature [5, 22, 23, 24]. The Software Inspection Process can reach an efficiency of about 99%, while PSP reviews have an average efficiency of 66% on the high side. While the Software Inspection Process may be effective for strategic software elements, it may be impractical for large-scale software development. Proponents still advocate 100% coverage for source code. This quickly becomes an economic impracticality and a misguided priority. The necessity of the Software Inspection Process can easily be minimized by use of Design Management, Defect Prevention, and SPC.
- Hewlett Packard SPI Cost Model: This SPI cost model was extracted from "Successful Software Process Improvement," by Robert Grady. It clearly indicates that Design Management (reuse) and the Software Inspection Process offer the greatest return on investment (ROI). Remember, Hewlett Packard has reclaimed $350 million dollars in development costs by using the Software Inspection Process.
- SPR SPI Cost Model: This SPI cost model by Capers Jones of Software Productivity Research indicates that the cost of SPI is rather negligible [48]. This model indicates that it costs a 1,000 person firm about $17 thousand to achieve industry leadership. The PSP can easily be implemented for that cost, and can result in world class leadership.
- SPR SPI Effort Model: This SPI effort model by Capers Jones of Software Productivity Research indicates that the time to achieve SPI is also rather negligible [48]. This model indicates that it takes just over 2 years to achieve industry leadership. Once again, the PSP can easily be implemented in less than that amount of time, and can result in world class leadership.
- Rome Labs SPI Cost Model: This SPI cost model by Rome labs has a similar construction to SPR's SPI cost model [49]. It clearly indicates an average of about a five to one return on investment for each class of SPI achieved.
- SEI SPI Cost Survey: The SEI conducted a survey of SPI costs and published the results [50]. The SEI survey, like the Rome Labs study, also indicates an average of about a five to one return on investment (ROI) for the surveyed SPI efforts.

How do you measure Quality ???

Many of the powerful SPI "solutions" advocated by this paper, and successfully exploited by the SPI benefactors, are based on the "Defect Removal Model" methodology. The "Defect Removal Model" is an extremely powerful, yet extremely

simple paradigm. The "Defect Removal Model" is based upon Defect Density Metrics and "Process Metrics" [1, 5, 7, 31].

Defect Density Metrics are extremely powerful, yet utterly simple (involving elementary arithmetic). Ironically, very few people have heard of them, understand them, or know how to apply them. In fact, most of the people that have heard of them, reject them as absurd, useless, and too simplistic.

The Defect Removal Model and Defect Density Metrics are used in conjunction with "Process Metrics" to successfully manage software development.

There are other more popular forms of metrics called "Product Metrics," "Design Metrics," or more appropriately "Structural Design Metrics." Examples of popular Product Metrics include Universal Design Metrics [7], Object Oriented Design Metrics [51], and Relational Database Design Metrics [52].

Very few software professionals use any metrics at all. The few that do, use Product Metrics. Proponents of Product Metrics are either oblivious to Defect Density Metrics and Process Metrics, or reject their value, usefulness, and applicability altogether.

- Defect Density: Defect Density Metrics take the simple arithmetic form of: Number of Defects per Thousand Source Lines of Code. Defect Density Metrics can be computed by anyone at any time and measured immediately before, during, and after each quality enhancing activity. Since they are so easy to use, and very accurate, Defect Density Metrics are excellent indicators of progress on day one. Defect Density Metrics dispel the myth that quality can't be measured, or that SPI is a long journey.
- Design Metrics: Design Metrics are a measure of source code structural style. How tall is it? How wide is it? How modular is it? How complex is it? Is it spaghetti code? Is it well structured?

  There are Design Metrics for procedural third generation programming languages, generally called "Universal Design Metrics." There are Design Metrics for object oriented programming languages, generally called "Object Oriented Design Metrics." There are even Design Metrics for fourth generation programming languages, generally called "Relational Database Design Metrics."

  Design Metrics can be used in conjunction with the Defect Removal Model and Defect Density Metrics. All violations of Design Metrics will be rolled up into a cumulative Defect Density Profile. However, it's important to remember that the Defect Removal Model is not dependent upon Design Metrics.

  The term "Design Metrics" is somewhat of a misnomer. Design Metrics only address source code structure. Design Metrics don't measure product desirability, design desirability, or customer satisfaction. Misunderstanding of this fact causes much confusion, misunderstanding, and misdirection of resources devoted to ultimately satisfying customers.

  The Defect Removal Model and Defect Density Metrics also struggle from positive correlation with customer satisfaction. The emerging field of Design Management corrects this dilemma.

SPI Myths & Misconceptions

Despite the overwhelming evidence for the benefits of SPI exhibited by this paper, only a small, and almost insignificant fraction of the world's software development community are represented by these experiences and results. It is with great regret that the majority of software professionals believe software development to be hopelessly confusing, and thus many of these myths, misconceptions, dogmas, and doctrines are accepted as irrefutable.

- High quality is too expensive: The overwhelming majority of software developers believe that if high software quality were possible, it would cost too much to be practical, or desirable. Many professionals believe this myth is true.
- SPI & high quality are for NASA & DoD: There is still a propensity for software professionals to believe that it costs an enormous amount of money and time to produce high quality software (something only NASA could afford or aspire too).
- Faster cycle times result in lower quality: Unfortunately, this myth is still common among

the uninitiated, that high quality requires a lengthy and expensive schedule, while faster cycle times must mean corners are being cut, and quality is being sacrificed.

- Software is purely creative thought stuff: It is also commonly believed that software development is spontaneous and unpredictably creative activity that can't be measured or repeated.
- Process improvement is a long journey: Among those that might be willing to believe that SPI is possible, it is commonly believed that it takes decades to achieve respectable results.
- CMM Level 3 is good enough: Level 3 is erroneously perceived to be average, and thus a perfectly acceptable goal to achieve and state to desire. Level 3 requires neither measurement nor improvement. Why would anyone want to be at Level 3?
- CMM Level 5 is a utopian state: Sigh. Level 5 is commonly believed to be a conceptual state of perfection that was never meant to be achieved. Sigh.
- CMM Level 5 costs more than Level 1: While most don't believe Level 5 is achievable, if it were, it is said, it would be cost prohibitive and offer no return on investment (ROI). Only NASA need worry about being in this state.
- SPI is just a fad: While SPI quickly became popular in the early 1990s, many have relegated SPI and the CMM to fad status. Unfortunately, nothing could be further from the truth. Great SPI breakthroughs are being rediscovered and even greater ones have yet to be invented.
- SPI doesn't affect the bottom line: Processes and process improvement are perceived to be ineffective when it comes to affecting an organization's profit and loss statement.
- Metrics are too hard & irrelevant: Mention the work "metric" and everyone runs for the hills. Software cost estimation is assigned to a cost estimation group, and metrics are perceived to be beyond the skill level of computer programmers. Powerful metrics involve only elementary arithmetic for the most part.
- Quality can't be measured: Many confuse design desirability and software quality, believing design desirability to be unquantifiable. Most powerful SPI methods use defects as a basic unit of measurement and software quality. Many refuse to accept this notion.
- SPI is too expensive: SPI is perceived to offer no ROI, take many years, and require large software process improvement staffs. Three strikes and you're out.
- CMM is too heavy & complex: While the CMM is a bit overwhelming at first, the CMM is designed around utter simplicity. Write a project plan, inspect your code, measure the process, and make an improvement next time. The SEI needs to work on this one.
- CMM certification without change: SPI professionals are often relegated to writing software process documents for CMM auditors to see, while the organizations make no effort to reform software development practices.
- ISO 9000 certification without change: Quality professionals are also tasked to construct a quality plan for ISO 9000 auditors, while the organization makes no attempt to reform the quality system.
- Pervasive myth of partial implementation: This is really aimed at misuse and misapplication of the CMM. Many believe that the CMM is designed to describe five different levels of sophistication, each having merits and degrees of acceptability. The CMM doesn't do that at all. The CMM describes the stages to achieving a single, unified, and functional state, Level 5. Many believe that it's perfectly acceptable and respectable to be at Levels 2 and 3, seeing no reason at all to aspire to Levels 4 and 5. The so called lucky few that have achieved Level 3 are perfectly content to stay there. Level 3 involves no useful measurement, statistical analysis, and process improvement.
- What part of the house can you live without ?: It is not at all acceptable to stop at Level 2, 3, and 4, or aspire to these Levels. The goal is to become a fully functional and improving software organization.

Model Corporate Culture

Organizational commitment is the key to success. If organizational participants believe that SPI affects the bottom line, then everyone will stop at nothing to achieve SPI. If all of an organization's leaders and staff aren't personally involved in SPI, especially the executives, that organization isn't committed to SPI, and won't achieve SPI organizationally. The leaders and staff of a problematic Motorola division didn't believe in SPI, and that division didn't understand why SPI couldn't be achieved. It takes two seconds to determine whether an organization will be successful at SPI. Are all of these people committed to SPI?

- CEO committed to SPI.
- Executives committed to SPI.
- Senior management committed to SPI.
- Line management committed to SPI.
- Staff committed to SPI.
- Organization hires people with SPI skills.
- Organization cultivates SPI skills.

If key organizational personnel state and adamantly believe that SPI doesn't affect the bottom line, it's time to pack up and leave.

Use Powerful "Solutions"

Finally, when an organization decides to try SPI, these guidelines can be an important roadmap to success. Though they seem rather simple and superficial, they are in fact deeply profound:

- Learn that SPI affects the "bottom line": This is the first step in beginning a SPI program. If SPI affects the bottom line, then everyone will want SPI completed today, not tomorrow.
- Aggressively achieve SPI: SPI is like pinball machine, it has to be rocked until the ball hits the right contacts to rack up the points. SPI must be worked until it yields noticeable results.
- Make SPI #1 goal (Ichiban): SPI must be the number one priority. Not an individual's number one goal, but everyone's. If it isn't your president's and management staff's number one goal, it's just not going to happen on an organizational scale.
- Choose powerful SPI "solutions": Here's the key. Don't waste your time with low leverage improvements. Examine the powerful "solutions" outlined in this paper and start with one of them. Once again, use powerful SPI "solutions." Use powerful SPI "solutions."
- Create SPI vision, strategy, & tactical plan: The organization's executives must construct a written project plan for identifying and rolling out powerful SPI "solutions" on a schedule.
- Manage directly to SPI vision: Manage all organizational activity to the SPI plan. Don't hire people that don't know SPI, to do SPI. Train project personnel in the SPI "solution." Measure the results. Make organizational adjustments. It only takes a few minutes and hours to have your first results.
- Reap benefits inexpensively on day one: If SPI is true, then SPI yields benefits. Measure, determine the benefits, and correct if necessary. SPI will begin paying for itself on day one.

Conclusion

This paper has examined SPI benefactors and benefits, SPI strategies and approaches, powerful SPI "solutions," and the costs of SPI. It has also clearly identified SPI myths to overcome. It has even provided a method to gauge the temperature of the SPI environment. Choose a powerful SPI "solution" and run with it. This isn't rocket science.

References

1. Daskalantonakis, M., "A Practical View of Software Measurement and Implementation Experiences Within Motorola," IEEE Transactions on Software Engineering, November 1992.

2. Paulk, M., "The Capability Maturity Model: Guidelines for Improving the Software Process," Addison Wesley, 1995.

3. Diaz, M., "How Software Process Improvement Helped Motorola," IEEE Software, September/October 1997.

4. Ferguson, P., “Results of Applying the Personal Software Process,” IEEE Computer, May 1997.

5. Fagan, M., “Design and Code Inspections to Reduce Errors in Program Development,” IBM Systems Journal, Volume 15, Number 3, 1976.

6. Billings, C., “Journey to a Mature Software Process,” IBM Systems Journal, Volume 33, Number 1, 1994.

7. Kan, S., "Metrics and Models in Software Quality Engineering,” Addison Wesley, 1995.

8. Mays, R., “Experiences with Defect Prevention,” IBM Systems Journal, Volume 29, Number 1, 1990.

9. Grady, R., “Successful Software Process Improvement,” Prentice Hall, 1997.

10. Haley, T., “Software Process Improvement at Raytheon,” IEEE Software, November 1996.

11. Kajihara, J., “Learning from Bugs,” IEEE Software, September 1993.

12. Kan, S., “AS/400 Software Quality Management,” IBM Systems Journal, Volume 33, Number 1, 1994.

13. Yamamura, G., “SEI CMM Level 5: For the Right Reasons,” Crosstalk, August 1997.

14. "LMFS Owego Attains Highest Level for Software Development Capability," Lockheed Martin News, 1998.

15. Imai, M., “Kaizen: The Key to Japan’s Competitive Success,” McGraw Hill, 1989.

16. Tingey, M., “Comparing ISO 9000, Malcolm Baldridge, and the SEI CMM for Software: A Reference and Selection Guide,” Prentice Hall, 1996.

17. Basili, V., "Experience Factory," Encyclopedia of Software Engineering Volume 1, Wiley and Sons, 1994.

18. Van Latum, F., “Adopting GQM-Based measurement in an Industrial Environment,” IEEE Software, January/February 1998.

19. Moriguchi, S., “Software Excellence: A Total Quality Management Guide,” Productivity Press, 1997.

20. Rockefeller, B., “Using SAP R/3 Fi: Beyond Business Process Reengineering,” Wiley & Sons, 1998.

21. Arthur, L., “Quantum Improvements in Software System Quality,” Communications of the ACM, June 1997.

22. Fagan, M., “Advances in Software Inspections,” IEEE Transactions on Software Engineering, July 1986.

23. Russell, G., “Experience with Inspection in Ultralarge-Scale Developments,” IEEE Software, January 1991.

24. Weller, E., “Lessons Learned from Three Years of Inspection Data,” IEEE Software, September 1993.

25. Sulack, R., “A New Development Rhythm for AS/400 Software,” IBM Systems Journal, Volume 28, Number 3, 1989.

26. Kan, S., “Modeling and Software Development Quality,” IBM Systems Journal, Volume 30, Number 3, 1991.

27. Kan, S., “Software Quality: An Overview from the Perspective of Total Quality Management,” IBM Systems Journal, Volume 33, Number 1, 1994.

28. Humphrey, W., “Using a Defined and Measured Personal Software Process,” IEEE Software, May 1996.

29. Humphrey, W., “Introduction to the Personal Software Process,” Addison Wesley, 1996.

30. Humphrey, W., “A Discipline for Software Engineering,” Addison Wesley, 1995.

31. Hayes, W., “The Personal Software Process (PSP): An Empirical Study of the Impact of PSP on Individual Engineers,” CMU/SEI-97-TR-001.

32. Humphrey, W., “Three Dimensions of Process Improvement Part II (The Personal Process: Solid data on the effectiveness of the Personal Software Process,” Crosstalk, March 1998.

33. Macala, R., “Managing Domain-Specific Product-Line Development,” IEEE Software, May 1996.

34. “Product-Line Reuse Delivers a System for One-Fifth the Cost in One-Half the Time,” Crosstalk, August 1996.

35. Dikel, D., “Applying Software Product-Line Architecture,” IEEE Computer, August 1997.

36. “Investment Analysis of Software Assets for Product Lines,” CMU/SEI-96-TR-010.

37. “A Case Study in Successful Product Line Development,” CMU/SEI-96-TR-016.

38. “Product Line Practice Workshop Report,” CMU/SEI-97-TR-003.

39. “A Concept of Operations for the ESC Product Line Approach,” CMU/SEI-96-TR-018.

40. Jones, C., “A Process-Integrated Approach to Defect Prevention,” IBM Systems Journal, Volume 24, Number 2, 1985.

41. Gale, J., “Implementing the Defect Prevention Process in the MVS Interactive Programming Organization,” IBM Systems Journal, Volume 29, Number 1, 1990.

42. Bhandari, I., “In-Process Improvement through Defect Data Interpretation,” IBM Systems Journal, Volume 33, Number 1, 1994.

43. Chillarege, R., “Orthogonal Defect Classification - A Concept for In-Process Measurements,” IEEE Transactions on Software Engineering, November 1992.

44. Bhandari, I., “A Case Study of Software Process Improvement During Development,” IEEE Transactions on Software Engineering, December 1993.

45. Burr, A., "Statistical Methods for Software Quality,” Thompson, 1996.

46. Owen, M., "SPC and Business Improvement," IFS Limited, 1993.

47. Wheeler, D., “Understanding Variation: The Key to Managing Chaos,” SPC Press, 1993.

48. Jones, C., “Activity Based Costs: Polishing the Software Process,” Software Development, December 1997.

49. McGibbon, T., “A Business Case for Software Process Improvement,” Contract F30602-92-C-0158, September 1996.

50. Herbsleb, J., “Benefits of CMM-Based Software Process Improvement: Initial Results,” CMU/SEI-94-TR-013.

51. Bansiya, J., "Automated Metrics and Object Oriented Development," Dr. Dobbs Journal, December 1997.

52. Siqueira, L., "Who is Minding your RDBMS Application Store?" DBE Software, 1997.