13

Business Case for Better Software Practices

"When you can measure what you are speaking about, and express it in numbers, you know something about it; but when you cannot measure it, when you cannot express it in numbers, your knowledge is of a meager and unsatisfactory kind."

— Lord Kelvin, 1893

Companies that invest in post-gold rush development practices have found that their investments pay off. In 1994, James Herbsleb reported that the average "business value" (roughly the same as return on investment) for 13 organizations that had taken on systematic improvement programs was about 5 to 1, with the best organizations realizing returns of 9 to 1.[1] In 1995, Neil C. Olsen reported similar returns for organizations that had made significant investments in staffing, training, and work environments.[2] In 1997, Rini van Solingen reported that the average ROI was 7 to 1 with the best organization realizing returns of 19 to 1.[3] In 2000, Capers Jones reported that the return on investment from process improvement could easily go into double digits (i.e., returns greater than 10 to 1).[4] A recent analysis by Watts Humphrey found that the ROI for improved software practices could be in the neighborhood of 5 to 1 or better.[5]

State of the Practice

Most people probably assume that software organizational effectiveness is distributed according to a typical bell curve—a few really poor organizations, a few exceptional organizations, and the majority somewhere in the middle. This is shown in Figure 13-1.

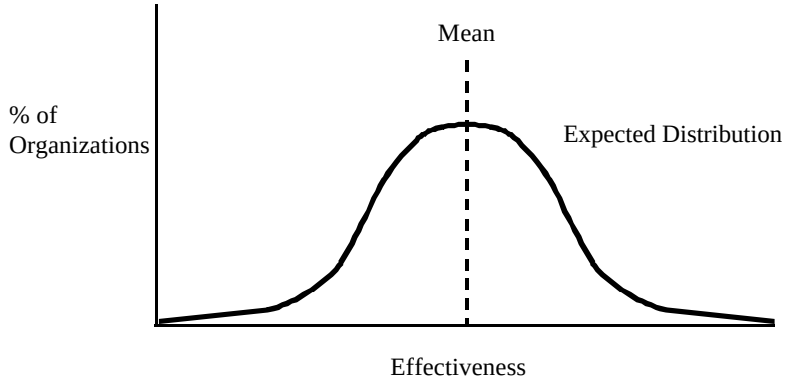

Figure 13-1
Most people expect organizational effectiveness to be symmetrically distributed, with an equal number of effective and ineffective software organizations.

Contrary to the expectation, the reality is quite different. Due to the slow uptake of effective software practices, as discussed in Chapters 1 and 2, only a handful of organizations are operating at the highest levels. Industry researchers have long observed a tremendous disparity between the best and worst organizations operating within the same industries—on the order of 10 to 1.[6] The average organization is much closer to the least effective organizations than to the best, as shown in Figure 13-2.

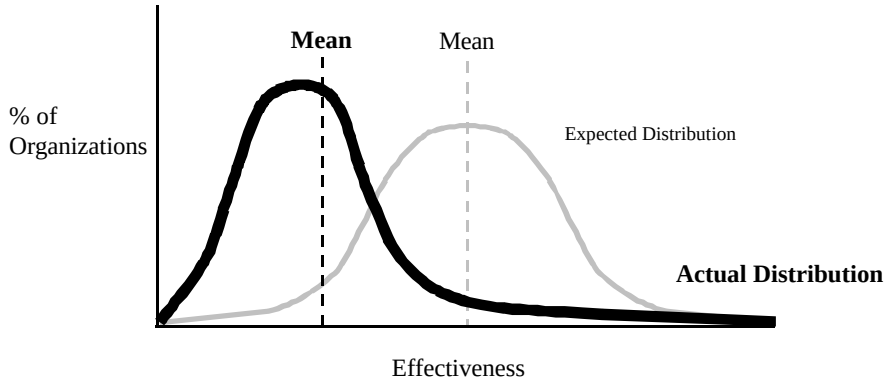

Figure 13-2
The actual distribution of software effectiveness is asymmetric. Most organizations perform much closer to the worst practice than to the best.[7]

The implication of this distribution is that most software personnel have never seen software development at its best. This gives rise to skepticism about whether things are really better anywhere. Even people who have worked in the software industry for 20 or 30 years might never have seen software development at its best. Most of them have spent their entire careers working in organizations to the left side of Figure 13-2. But, as the data in this chapter shows, the best organizations are indeed much better than the average.

Detailed Benefits of Improved Software Practices

An in-depth study of 13 organizations by the Software Engineering Institute found that the typical (median) organization engaged in systematic improvement experienced a productivity gain of 35 percent per year, schedule reduction of 19 percent per year, and reduction in post-release defect reports of 39 percent per year. These gains provided the underpinnings for the overall returns on investment. These results are summarized in Table 13-1.

Table 13-1. Results of software process improvement efforts.[8]

Factor	Median Improvement	Best Sustained Improvement
Productivity	35%/year	58%/year
Schedule	19%/year	23%/year
Post-release defect reports	39%/year	39%/year*
Business value of organizational improvement	5 to 1	8.8 to 1

**The "median" and "sustained" values here are the same because the highest defect-reduction results were short-term and not counted as "sustained" improvements.*

The gains experienced by the best organizations were even better. The organization with the strongest productivity gains improved 58 percent per year over a 4-year period, for a total compounded gain of more than 500 percent. The best schedule reduction was 23 percent per year for 6 years, for a total compounded schedule reduction of 91 percent. The best long-term quality improvement was a reduction in post-release defect reports of 39 percent per year for 9 years, for a total compounded defect reduction of 99 percent. Two organizations achieved short-term defect reductions of seventy percent or more in less than two years.

Organizations that are hooked on code-and-fix development tend to think there is a tradeoff between low defect count and productivity. But as I explained in Chapter 2, much of the cost on a code-and-fix project arises from unplanned defect-correction work. The results in Table 13-1 confirm that for most organizations no tradeoff exists between higher productivity and better quality. Organizations that focus on preventing defects also get shorter schedules and higher productivity.

As a percentage, the number of companies that have engaged in systematically improving their software practices is small. In raw numbers, hundreds of organizations have engaged in systematic improvement, and many have reported their results in professional journals, conference proceedings, and other publications. Table 13-2 summarizes the returns on investment reported by 20 organizations.

Table 13-2. Examples of software improvement ROI.[9]

Organization	Results
Boeing Information Systems	Estimates within 20%, $5.5 million saved in 1 year, ROI 775%
BDN International	ROI 300%
Computer Sciences Corporation	65% reduction in error rates
General Dynamics Decision Systems	70% reduction in rework; 94% defect rate reduction; 2.9x productivity gain
Harris ISD DPL	90% defect rate reduction; 2.5x productivity gain
Hewlett-Packard SESD	ROI 900%
Hughes	$2 million annual reduction in cost overruns, ROI 500%
IBM Toronto	90% reduction in delivered defects, 80% reduction in rework
Motorola GED	2-3X productivity improvement, 2-7X cycle time reduction, ROI 677%
Philips	ROI 750%
Raytheon	ROI 770%
Schlumberger	4X reduction in beta test bugs
Siemens	90% reduction in released defects
Telcordia	Defects $^{1}/_{10}$ industry average, customer satisfaction increased from 60% to 91% over 4 years
Texas Instruments – Systems Group	90% reduction in delivered defects
Thomson CSF	ROI 360%
U.S. Navy	ROI 410%
USAF Ogden Air Logistics Center	ROI 1900%
USAF Oklahoma City Air Logistics Center	ROI 635%
USAF Tinker Air Force Base	ROI 600%

ROIs for Selected Practices

Details of how organizations have achieved their greater software development effectiveness vary among different organizations. Nonetheless, some specific practices have been found to be generally effective. Table 13-3 shows the ballpark ROIs for a few selected practices.

Table 13-3. Return on investment for selected software practices.[10]

Practice	12-month ROI	36-month ROI
Formal code inspections	250%	1,200%
Formal design inspections	350%	1,000%
Long-range technology planning	100%	1,000%
Cost and quality estimation tools	250%	1,200%
Productivity measurements	150%	600%
Process assessments	150%	600%
Management training	120%	550%
Technical staff training	90%	550%

Insights from Software Estimation

The accuracy of an organization's estimates is a good indicator of how well it manages and executes its projects. A Standish Group survey of more than 26,000 business systems projects found that the average project overran its planned budget by more than 100 percent.[11] The level of estimation error reported in this survey is consistent with other industry findings.[12]

Figure 13-3 shows one finding from a study of U.S. Air Force projects at different levels of software practices. (This analysis is based on the SW-CMM, which I'll discuss more in Chapter 9.) Each point below 100 percent represents a project that finished under budget. Each point above 100 percent represents a project that overran its estimate.

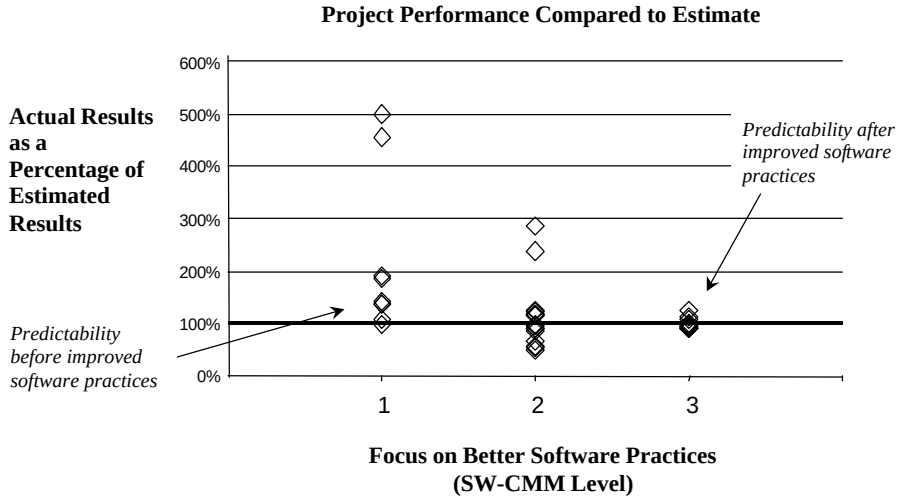

Figure 13-3
As organizations improve their software practices, they gain more control over their project estimates, which is generally indicative of better project control.[13]

As you can see from figure 13-3, the least sophisticated projects (SW-CMM Level 1) routinely overran their projects' budgets—in other words, they routinely underestimated their projects' costs. More sophisticated organizations (at SW-CMM Level 2) spread their estimation error more evenly between overestimation and underestimation, but the relative error of the estimates was still commonly 100 percent or more. For the most sophisticated organizations (those at SW-CMM Level 3), overruns and underruns become equally common, and the estimation accuracy became much improved.

Indirect Benefits of Improved Software Practices

The return on investment figures in the published literature have mostly been based on operational savings, that is, on reducing development cost per line of code written or per function point delivered. While these savings are impressive, the greater business benefit might arise from the significant *indirect* returns attributable to improved software practices. Better software practices improve cost and schedule predictability, reduce risk of cost and

schedule overruns, provide early warnings of problems, and support better management.

For a software products company, what is the business value of improving schedule estimation accuracy from ±100% to ±10%? What is the value of being able to make a commitment to customers 6 to 12 months in advance of a scheduled completion date with high confidence of delivering on that commitment? For a company that develops custom software, what is the business value of being able to provide a fixed-price bid with high confidence that the project will not significantly overrun the bid? For a retail sales organization, what is the value of being able to plan cutover to a new system with pinpoint accuracy? What is the value of knowing with confidence that cutover will occur October 1, as planned, with little risk of running into the holiday sales season?

Unlike the operational benefits that most of the industry literature has focused on, the indirect benefits of improved software practices open the door to additional revenue opportunities. For top decision makers, these indirect benefits may be more compelling than the direct, cost-savings benefit.

A View of the Best

Further examination of estimation practices holds out an interesting process-improvement carrot. Most organizations find that as their projects become larger, each team member becomes less productive. In contrast with the economies of scale experienced in other kinds of work, software projects usually experience *dis*economies of scale.

Organizations that use systematic estimation practices use formulas like this one to estimate their software project effort:[14]

$$\text{Effort} = 2.45 * \text{KSLOC}^{1.15}$$

Effort is the number of staff-months and *KSLOC* is the estimated number of lines of code in thousands. The numbers 2.45 and 1.15 are derived by calibration using data from projects that the organization has already completed. The values of 2.45 and 1.15 apply to typical

organizations. The specific value of the exponent (1.15) is significant because it means that larger projects require disproportionately more effort than smaller projects.

NASA's Software Engineering Laboratory (SEL) is a notable exception. The SEL was the first organization to receive the IEEE's award for software process achievement and is one of the most sophisticated software development organizations in the world. The SEL uses the following formula to estimate effort on its projects:[15]

$$\text{Effort} = 1.27 * \text{KSLOC}^{0.986}$$

In spite of its small type, the exponent of 0.986 points to a momentous difference between the SEL's formula and the formulas used by every other software organization. Every other published estimation model uses an exponent greater than 1.0. The fact that the SEL's exponent is less than 1.0 suggests that the SEL is achieving a slight *economy* of scale. The process improvement carrot this represents is that sufficiently mature organizations might be able to turn the large-project problem on its head. They might actually be able to improve per-person productivity slightly as their projects grow larger. Although rare, this is indeed a logical consequence of the specialization I will discuss in Chapter 16.

Challenge Is Organizational

Many organizations push responsibility for improved software practices down to the project level. As I was reviewing the "effort-multiplier" factors in the popular Cocomo II estimation model[16] recently, I was struck by how few factors are under the control of an individual project manager. Of the 22 factors Cocomo II uses to fine-tune a project's base effort estimate, only 3 in my judgment are typically under the control of an individual project manager (the factors of Level of Documentation, Architecture and Risk Resolution, and Development for Reuse). Numerous factors are dictated by the nature of a company's business (product complexity, required reliability, platform volatility, unprecedentedness of the software, and so on). These factors cannot easily be changed without changing businesses. The

remaining factors cannot readily be influenced by individual projects and must be addressed by the organization—staff capability, multi-site development, personnel continuity, process maturity, and other factors. These organization-level areas seem to be where the greatest leverage for improved software practices lies.

The Last Great Frontier

For a typical business-investment decision, the desirability of the investment is determined by weighing the return on investment against the cost of capital. An investment that produces a return greater than the cost of capital—all things considered—will be a good investment.[17] (This is a simplified explanation. See the references for more complete explanations.)

Cost of capital is typically around 10%. In many business contexts, an investment with a return of 15% or 20% would be considered compelling. Improved software practices, however, do not offer returns of 15% or 20%. *According to the examples in Table 13-2 (as well as studies cited at the beginning of the chapter), improved software practices provide returns ranging from 300% to 1900% and average about 500%.* Investments with these levels of returns are extraordinary—virtually unprecedented in business. These returns are higher than Internet stocks in the late 1990s. They're higher than successful speculation in the commodities markets. They're almost as good as winning the lottery, and they represent an unrivaled opportunity for any business that isn't already using these practices.

The reason for these exceptionally high returns is tied directly to the discussions in Chapters 1 and 2—improved practices have been available for decades, but most organizations aren't using them. Risk of adopting these practices is low; payoff is high. All that's needed is the organizational resolve to use them.

Ten Tough Questions

As Lord Kelvin wrote more than 100 years ago, decision-making is

difficult without quantitative data to support the decisions. Many organizations find they cannot answer basic questions about their software activities like these:

1. How much are you spending on software development?
2. What percentage of your projects are currently on time and on budget?
3. What is the average schedule and budget overrun for your projects?
4. Which of your current projects are most likely to fail outright?
5. What percentage of your project cost arises from avoidable rework?
6. How satisfied (quantitatively) are users of your software?
7. How do the skills of your staff compare to industry averages?
8. How do the capabilities of your organization compare to similar organizations?
9. How much (quantitatively) has your productivity improved in the past 12 months?
10. What is your plan for improving the skills of your staff and the effectiveness of your organization?

Organizations that cannot answer these questions almost certainly fall to the left side of Figure 13-2. Many organizations have never asked questions such as these and are only vaguely aware that they should be asking such questions. Considering that the business case for better software practices is so compelling, perhaps the 11th tough question should be, "What's keeping us from using better software practices now that we have seen this data?"

Notes

[1] Herbsleb, James, et al., "Benefits of CMM Based Software Process Improvement: Initial Results," Pittsburgh: Software Engineering Institute, Document CMU/SEI-94-TR-13, August 1994.

[2] Olsen, Neil C., "Survival of the Fastest: Improving Service Velocity," *IEEE Software*, September 1995, pp. 28–38.

[3] van Solingen, Rini, "The Cost and Benefits of Software Process Improvement," *Proceedings of the Eighth European Conference on*

Notes (continued)

Information Technology Evaluation, September 17-18, 2001.

[4] Jones, Capers, *Software Assessments, Benchmarks, and Best Practices*, Boston, MA: Addison-Wesley, 2000.

[5] Humphrey, Watts, *Winning with Software: An Executive Strategy*, Boston, MA: Addison-Wesley, 2001.

[6] Mills, Harlan D., *Software Productivity*, Boston, MA: Little, Brown, 1983.

[7] "Process Maturity Profile of the Software Community 2001Year End Update," Software Engineering Institute, March 2002.

[8] Data in this table is from Herbsleb, James, et al., *Benefits of CMM Based Software Process Improvement: Initial Results*, Pittsburgh: Software Engineering Institute, Document CMU/SEI-94-TR-13, August 1994.

[9] Adapted from Krasner, Herb, "Accumulating the Body of Evidence for the Payoff of Software Process Improvement – 1997," November 19, 1997. Unpublished paper; van Solingen, Rini, "The Cost and Benefits of Software Process Improvement," *Proceedings of the Eighth European Conference on Information Technology Evaluation, September 17-18, 2001*. Diaz, Michale, and Jeff King, "How CMM Impacts Quality, Productivity, Rework, and the Bottom Line," *CrossTalk*, vol. 15, no. 3 (March 2002), pp. 9–14.

[10] Jones, Capers, *Assessment and Control of Software Risks*, Englewood Cliffs, NJ: Yourdon Press, 1994.

[11] The Standish Group, "Charting the Seas of Information Technology," Dennis, MA: The Standish Group, 1994. Johnson, Jim, "Turning Chaos into Success," *Software Magazine*, December 1999, pp. 30–39 (Johnson is Chairman of The Standish Group).

[12] See, for example, Jones, Capers, *Patterns of Software Systems Failure and Success*, Boston, MA: International Thomson Computer Press, 1996.

[13] Lawlis, Dr. Patricia K., Capt. Robert M. Flowe, and Capt. James B. Thordahl, "A Correlational Study of the CMM and Software Development

Notes (continued)

Performance," *Crosstalk,* September 1995.

[14] See, for example, University of Southern California, *Cocomo II Model Definition Manual*, version 1.4, undated (circa 1997), which contains the coefficient of 2.45 as its baseline coefficient and the exponent of 1.15 as the nominal exponent.

[15] NASA Software Engineering Laboratory, *Software Engineering Laboratory (SEL) Relationships, Models, and Management Rules*, Document Number SEL-91-001, Greenbelt, MD: Goddard Space Flight Center, NASA, 1991.

[16] Boehm, Barry, et al., *Software Cost Estimation with Cocomo II*, Boston, MA: Addison-Wesley, 2000.

[17] Fetzer, Daniel T., "Making Investment Decisions for Software Process Improvement," DACS Software Tech News, November 2002, pp. 19–22. Also available at www.dacs.dtic.mil/awareness/newsletters. Reifer, Donald J., *Making the Software Business Case: Improvement by the Numbers*, New York: Addison-Wesley, 2001.