Determining Return on Investment Using Software Inspections

Don O'Neill
Consultant

This article examines the defined measurements used to form the derived metric for return on investment. These measurements involve additional cost multiplier, defect detection rate, cost to repair, and detection cost. This article further examines the behavior of these measurements and metrics for various software product-engineering styles using data collected in the National Software Quality Experiment.

To deliver superior quality, many organizations have made commitments to initiatives on the Software Engineering Institute's (SEI) Capability Maturity Model® (CMM®), ISO 9001, or Six Sigma. Each of these initiatives has one thing in common: software inspections.

Managers are interested in knowing the return on investment to be derived from software process improvement actions. The software inspection process gathers some of the data needed to determine this [1]. Software inspections are structured to serve the needs of quality management in verifying that a software artifact complies with its standard of excellence for software engineering. The focus is on verification, on doing the job right. The software inspection is a formal review held at the conclusion of a life-cycle activity and serves as a quality gate with exit criteria for moving on to subsequent activities [2].

The National Software Quality Experiment (NSQE) is a mechanism for obtaining core samples of software product quality. The NSQE includes a micro-level national database of product quality populated by a continuous stream of samples from industry, government, and military services. The centerpiece of the experiment is the software inspection lab where data collection procedures, product checklists, and participant behaviors are packaged for operational project use. The NSQE is providing valuable insights on the practice of software inspections through its database of thousands of software inspection sessions from dozens of organizations containing tens of thousands of defects along with the pertinent information needed to pinpoint their root causes [3].

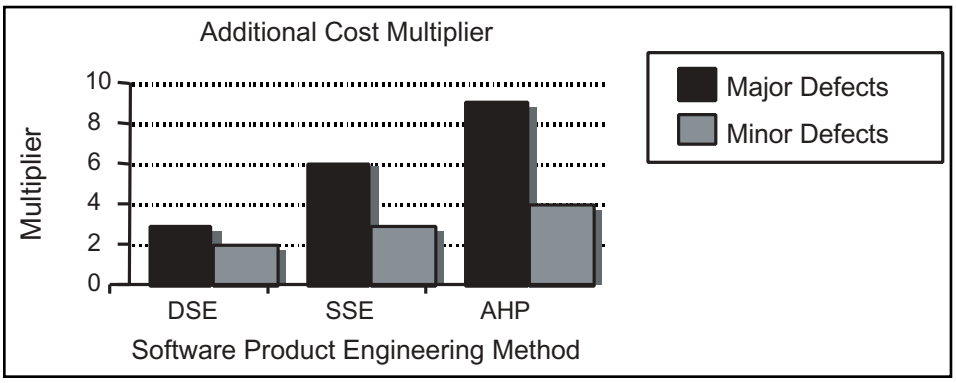
> *"The NSQE is providing valuable insights on the practice of software inspections through its database of thousands of software inspection sessions from dozens of organizations containing tens of thousands of defects along with the pertinent information needed to pinpoint their root causes."*

To review NSQE description and data summaries, please visit the Web resource listed in [4].

The model in this article for return on investment bases the savings on the cost avoidance associated with detecting and correcting defects earlier rather than later in the product evolution cycle. It is defined as *net savings* divided by *detection cost*, where net savings is *cost avoidance* less *cost to repair now*; detection cost is the cost of preparation effort and the cost of conduct effort. Savings result from early detection and correction, which avoids the increased cost multiplier associated with detection and correction of defects later in the life cycle.

A major defect that leaks from development to test may cost two to 10 times more to detect and correct than if done earlier. Some of these defects leak further from test to customer use and may cost an additional two to 10 times to detect and correct. A minor defect may cost an additional two to four times to correct later. The defined measurements collected in the software inspection lab may be combined in complex ways to form the derived metric for return on investment. These involve an additional cost multiplier, defect detection rate, cost to repair, and detection cost.

Figure 1: *Additional Cost Multiplier*



Software Product Engineering Method

The values for these complex parameters revolve around the Software Product Engineering (SPE) key process area being practiced, which is a CMM Level 3 key process area. Three levels of achievement of SPE are identified as follows:

1. Ad-hoc programming is characterized by a code-and-upload life cycle and a hacker coding style. This is common in low software process maturity organizations, especially those facing time-to-market demands.
2. Structured software engineering employs structured programming, modular design, and defined programming style, and pays close attention to establishing and maintaining traceability among requirements,

specification, architecture, design, code, and test artifacts. This is the minimum expectation for CMM Level 3 [5].
3. Disciplined software engineering is more formal and might be patterned after cleanroom software engineering, Personal Software Process, Team Software Process, and extreme programming techniques [6, 7, 8]. This is the expectation for SEI CMM Level 4 and 5 organizations [5].

Additional Cost Multiplier

Since savings result from avoiding the increased cost multiplier associated with detection and correction of defects later in the life cycle, the question of the cost multiplier must be answered to determine the return on investment.

Some set the additional cost multiplier for finding and fixing a defect detected after delivery at 100 times earlier detection [9]. Others have measured it more precisely and found it to be 10 times more for each life-cycle activity. IBM Rochester, Rochester, Minn., winner of the Malcolm Baldrige National Quality Award in 1990, reported that defects leaking from code to test cost nine times more to detect and correct, and defects leaking from test to the field cost 13 times more to detect and correct [10].

Why Is There a Multiplier?

An example may help illustrate why a leaked defect costs more. A code defect that leaks into testing may require multiple test executions to confirm the error and additional executions to obtain debug information. Once a leaked defect has been detected, the producing programmer must put aside the task at hand, refocus attention on correcting the defect and confirming the correction, and then return to the task at hand. The corrected artifact must then be reinserted into the SPE or product release pipeline and possibly into user operations.

Keep in mind that software changes experience high defect rates. In addition, the span of impact from defects introduced during different activities of the life cycle varies. The number of source lines affected by a requirement defect might exceed 100 lines; by a design defect, 10 to 100 lines; by a coding defect, less than 10 lines; and by a clerical defect, one line.

What Is the Multiplier?

It is reasonable to expect the additional cost multiplier to be linked to the SPE method practiced. Figure 1 portrays the additional cost multiplier by SPE method.

1. Ad-hoc programming (AHP) is likely to experience a multiplier of eight to 10 times in detecting and correcting major defects in spaghetti-bowl coding that lacks order and consistency. The multiplier for minor defects is likely to be four times.
2. Structured software engineering (SSE) is likely to experience a multiplier of five to seven times in detecting and correcting major defects in the production of well structured, consistently recorded components with organized relationships among modules and traceability among life-cycle artifacts. The multiplier for minor defects is likely to be three times.
3. Disciplined software engineering (DSE) with its formal focus on quality may experience a multiplier of two to four times in detecting and correcting major defects. The multiplier for minor defects is likely to be two times.

What Effect Does the Multiplier Have?

In summary, an undetected major defect that leaks to the next phase of the life cycle may cost two to 10 times more to

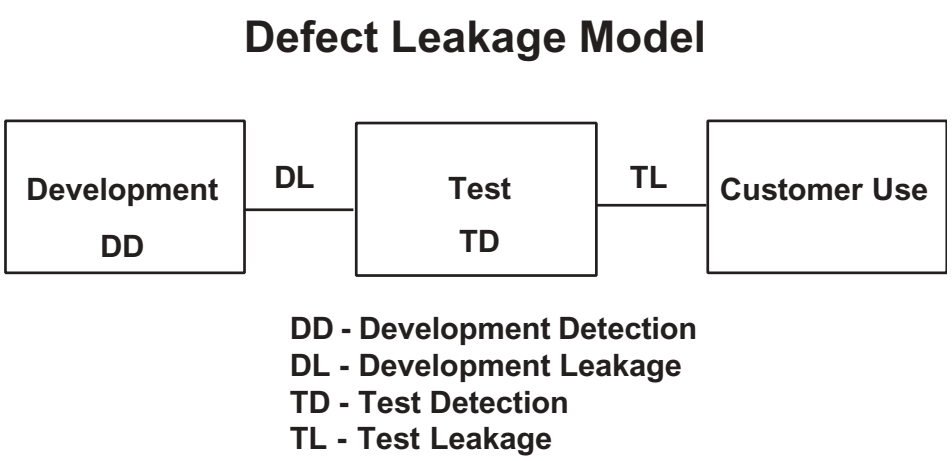


Figure 2: *Defect Leakage Model*

Figure 3: *Development Detection*

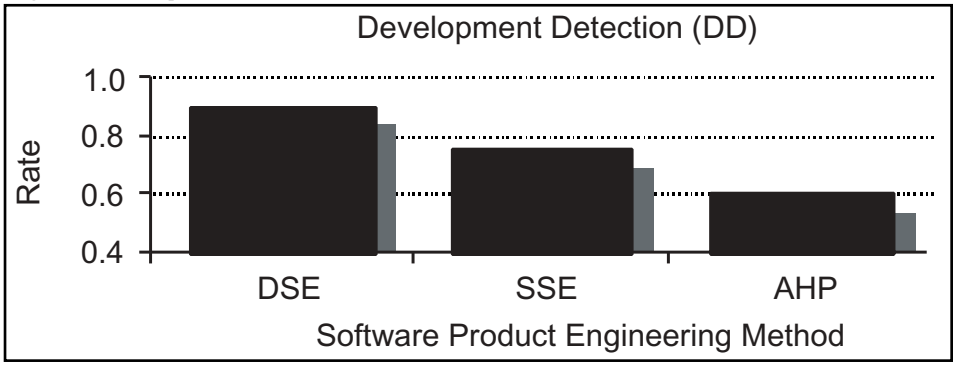
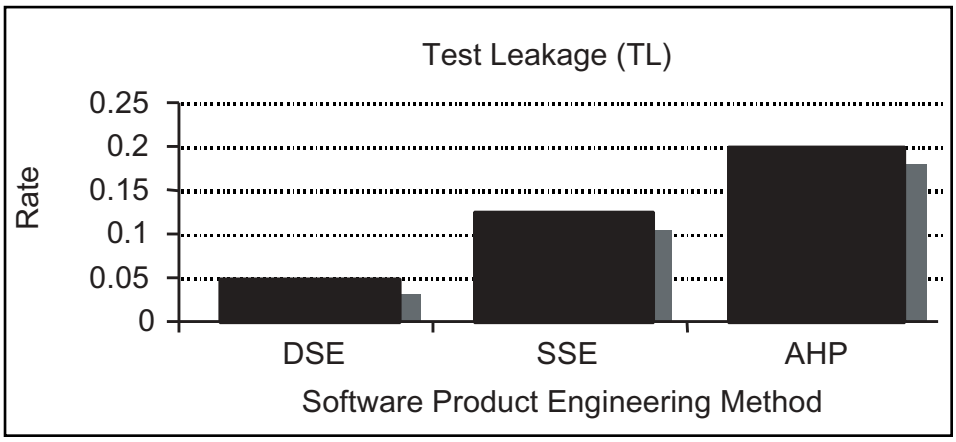


Figure 4: *Test Leakage*



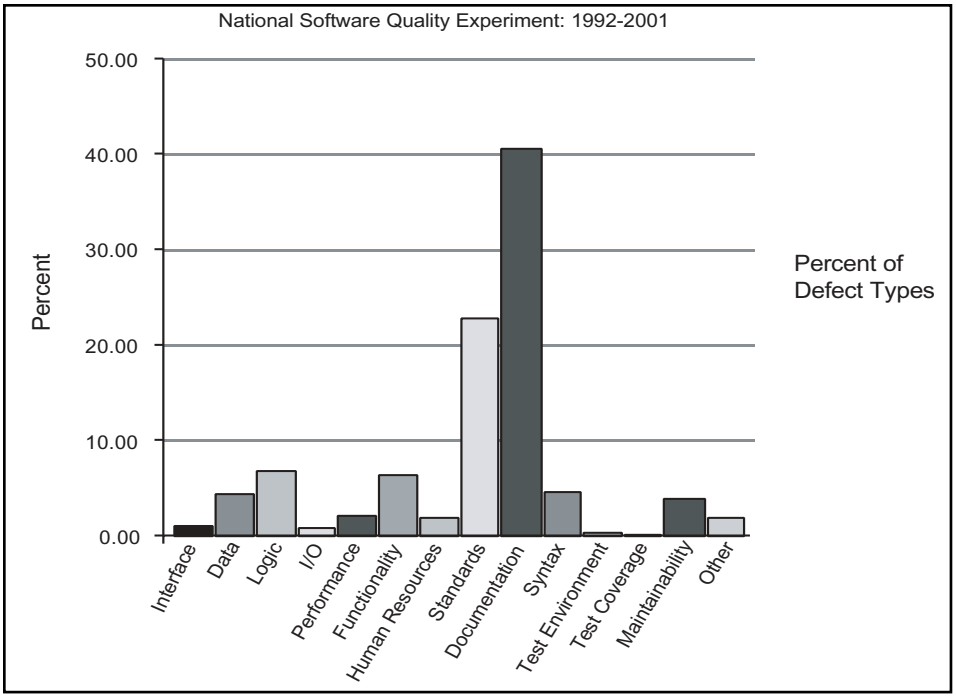


Figure 5: *Defect Type Distribution*

detect and correct. A minor defect may cost two to four times more to detect and correct. The resulting net savings then may be up to nine times for major defects and up to three times for minor defects.

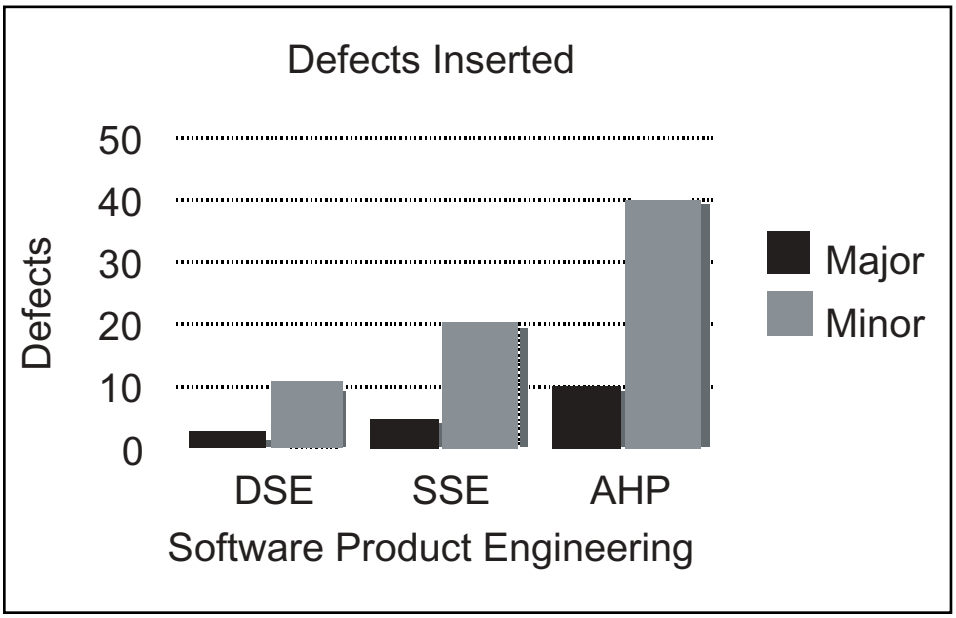
Defect Detection Rate

The model shown in Figure 2 illustrates that defects are detected in development (DD) and test (TD), defects leak from development (DL), and defects leak from test (TL). Defect detection rate equals the number of defects detected divided by the number of defects present.

It is reasonable to expect the defect detection rate to be linked to the SPE method practiced, including the software inspection process followed. Figures 3 and 4 illustrate DD and TL using empirically derived values for the defect leakage model factors of each SPE method. While the defect DD rates are based on the results of the NSQE, the expected TD uses a notional value in order to complete the analysis.

1. AHP is likely to experience a defect DD rate in the range of 0.50 to 0.65. While the TL depends on the adequacy of the test process, AHP is likely to experience TL in the range of 0.175 to 0.25 based on an expected TD of 0.50.
2. The SSE is likely to experience a defect DD rate in the range of 0.70 to 0.80, and a TL in the range of 0.1 to 0.15 based on an expected TD of 0.50.
3. DSE may experience a defect DD rate in the range of 0.85 to 0.95, and a TL in the range of 0.025 to 0.075 based on an expected TD of 0.50 [4].

Figure 6: *Defects Inserted Per Thousand Lines*



Cost to Repair

The cost to repair a defect detected in the life-cycle activity in which it was inserted depends on the SPE method practiced and the business environment in which it is operating. This must be supplied by the organization based on its actual cost history and the superior knowledge of its personnel.

In determining the cost to repair, the organization needs to obtain this cost-by-defect type. During the software inspection lab session, each detected defect is assigned a type, including interface, data, logic, input/output, performance, functionality, human factors, standards, documentation, syntax, maintainability, and others. The defect type distribution revealed by the NSQE is shown in Figure 5 [3, 11, 12].

For purposes of the software inspections' return-on-investment analysis, the cost-to-repair factor is included in the expression for net savings discussed later. For analysis here, the cost to repair is set at one hour for a major defect and one hour for a minor defect.

Major defects are those that affect execution; minor defects do not affect execution but may still be important. Some practitioners mistakenly assign a major classification to defects that require extensive rework, and a minor classification to defects that require little rework. When this mistake is made, the major rework metric will systematically exceed the minor rework metric.

In actual practice, as measured by the NSQE during the past 10 years, certain major defects such as eliminating magic numbers or inserting a *when others clause* require one or two line changes in the code. Also, some minor defects such as lack of traceability or adding prologue and version history commentary may have more pervasive impact.

During software inspections, many defects are detected; many are trivial and require little cost-to-repair effort. Others may be more complex and require substantial effort. Where the organization has superior knowledge of the cost-to-repair metric, it should use that informa-

tion. Where the organization lacks measured results, using one hour for cost to repair is an initial value that many have found valid.

Defect Detection Cost

The cost of defect detection includes the participants' efforts to prepare and conduct the software inspection. Time to conduct the inspection includes the actual physical time consumed by the software inspection meeting. Effort to conduct the inspection includes the actual time it takes to complete the inspection multiplied by the number of participants. Factors used to determine detection cost include the size of the artifact being inspected, the number of defects inserted, and the relationship between the effort to prepare and conduct the inspection.

It is reasonable to expect the defect detection cost to be linked to the SPE method practiced, including the software inspection process followed. Figure 6 illustrates the following defect insertion rates by the SPE method.

1. AHP may experience a preparation effort divided by a conduct effort ratio of approximately 0.60 in inspecting artifacts of 400 to 600 lines of code, as experienced by CMM Level 1 organizations in the NSQE [6, 8, 10]. These organizations may experience a defect insertion rate of 40 to 60 defects per thousand lines of code.
2. SSE may experience a preparation effort divided by a conduct effort ratio of approximately 0.80 in inspecting artifacts of 200 to 400 lines of code, as experienced by CMM Level 3 organizations in the NSQE [6, 8, 10]. These organizations may experience a defect insertion rate of 20 to 30 defects per thousand lines of code.
3. DSE may experience a preparation effort divided by a conduct effort ratio of approximately 1.0 in inspecting artifacts of less than 200 lines of code. These organizations may experience a defect insertion rate of 10 to 15 defects per thousand lines of code.

Reasoning About ROI

Software inspections' return on investment is equal to net savings divided by detection cost. Evaluating the following expression assists in reasoning about return on investment:

ROI = Net Savings/Detection Cost

Reasoning About Net Savings

Net savings is equal to cost avoidance minus cost to repair now. Evaluating the following expression assists in reasoning about net savings:

Net Savings =
Cost Avoidance - Cost to Repair Now

Cost avoidance results from avoiding the higher costs that occur from deferred detection and correction. The additional cost multiplier comes into play in the following ways:

- M1 is the additional cost-to-repair multiplier for development to test major defect leakage.
- M2 is the additional cost-to-repair multiplier for test to customer-use major defect leakage.
- M3 is the additional cost-to-repair multiplier for minor defect leakage.

Evaluating the following expression assists in reasoning about cost avoidance:

Cost Avoidance =
Major Defects x {(M1 x DD) + (M1 x DD) x (M2 x TL) x C1} + Minor Defects x M3

The cost to repair now, simply the cost of defect correction, is subtracted from cost avoidance to yield net savings. Evaluating the following expression assists in reasoning about net savings:

Net Savings = Major Defects x {(M1 x DD) + (M1 x DD) x (M2 x TL) x C1 - C1} + Minor Defects x (M3 - C2)

Simplifying the expression results in the following:

Net Savings =
Major Defects x {C1 x [(M1 x DD) x (1 + (M2 x TL))] - 1} + Minor Defects x (M3 - C2)

Where:

- M1: (2 - 10) Additional Cost-to-Repair Multiplier for Development to Test Major Defect Leakage.
- M2: (2 - 10) Additional Cost-to-Repair Multiplier for Test to Customer Use Major Defect Leakage.
- M3: (2 - 4) Additional Cost to Repair for Minor Defect Leakage.
- DD: (0.5 - 0.95) Defect Detection Rate for Development to Test.
- TL: (0.05 - 0.5) TL Rate for Test to Customer Use.
- C1: Average Cost to Repair Major Defect (in hours of effort).
- C2: Average Cost to Repair Minor Defect (in hours of effort).

Reasoning About Detection Cost

Detection cost is equal to preparation effort plus conduct effort. Evaluating the following expression assists in reasoning about detection cost:

Detection Cost =
Preparation Effort + Conduct Effort

Preparation effort is the total minutes of preparation effort. Conduct effort is the minutes of conduct time multiplied by the number of participants. Substituting the resulting expression is the following:

Detection Cost =
{Minutes of Preparation Effort + (Minutes of Conduct Time x Participants)}/60

Where:

- Participants: (4-6) Number of participants.
- 60 minutes per hour.

A Worked Example

The return on investment is determined by using the expression for net savings specified above and setting the parameters for cost-to-repair multiplier, defect detection, and defect leakage. For example, to determine the expression for return on investment to be used in a project spreadsheet, the following example is offered:

1. Setting the parameters: M1=5, M2=10, M3=2, DD=0.6, TL=0.25, C1=1, and C2=1.
2. Using the expression:

   Net Savings = Major Defects x {(M1 x DD) + (M1 x DD) x (M2 x TL) x C1 - C1} + Minor Defects x (M3 - 1)

3. Substituting for the values of the worked example:

   Net Savings = Major Defects x {(5 x .6) + (5 x .6) x (10 x .25) x 1- 1} + Minor Defects x (2-1)

4. The following expression for Net Savings results:

   Net Savings = 9.5 x Major Defects + Minor Defects

The result of the worked example is

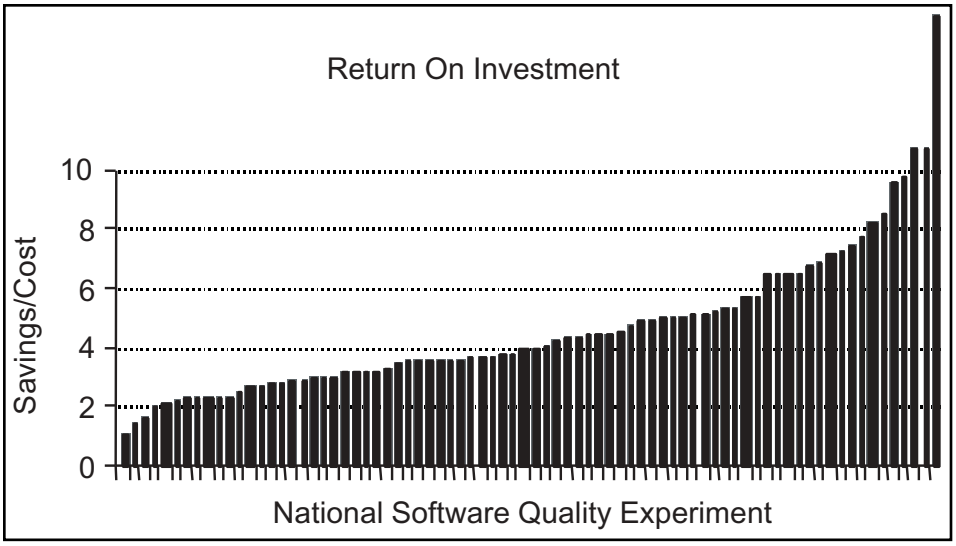


Figure 7: *Return on Investment*

a simplified expression for net savings of the type used to derive the return-on-investment metric in the NSQE. Figure 7 illustrates the range of practice for return on investment.

Selecting Parameter Values

Where an organization possesses superior knowledge of its software operation, it should utilize the parameter values that best reflect this understanding. Candidate parameter values for each SPE method are shown in Table 1 for DSE, SSE, and AHP.

Computing Return on Investment

Software process improvement goals involve both cost and quality. The achievement of these goals varies according to the SPE method practiced, and these variations are illustrated in the application of the selected parameter values (see Table 2). AHP practitioners derive substantial net savings and return on investment, but a high incidence of defect leakage into customer use. The SSE practitioners experience attractive net savings and return on investment, and a reduced defect leakage into customer use. DSE practitioners barely recoup the investment but achieve a very low incidence of defect leakage into customer use.

Transition From Cost to Quality

In using software inspections, the goals vary with the SPE method used, transitioning from cost to quality.

By necessity, the focus of AHP practitioners is on reducing cost by detecting as many defects as possible. With 40 to 60 defects inserted per thousand lines of code, a defect detection rate of 0.5 to 0.65, and an additional cost multiplier of eight to 10, the result is a net savings of 234.8 to 285 labor hours and a defect leakage expectation of 8.75 to 12.5 per thousand lines of code, numbers that promote a focus on cost. For this group, finding defects is like finding free money, and there are always more defects to find; however, managers struggle to meet cost and schedule commitments.

The SSE focus is split between reducing cost and improving quality. With 20 to 30 defects inserted per thousand lines of code, a defect detection rate of 0.70 to 0.80, and an additional cost multiplier of five to seven, the result is a net savings of 65 to 85.23 labor hours and a defect leakage expectation of 2.5 to 3.75 per thousand lines of code, numbers that promote an attraction to both goals. For this group, there is constant dithering between cost and quality.

Without question, the focus of DSE practitioners is on eliminating every possible defect even if defect detection costs exceed net savings and the return on investment falls below the break even point. With 10 to 15 defects inserted per thousand lines of code, a defect detection rate of 0.85 to 0.95, and an additional cost multiplier of two to four, the result is a net savings of 12.49 to 18.55 labor hours and a defect leakage expectation of 0.3125 to 0.9375 per thousand lines of code, numbers that promote a focus on quality. For this group, every practitioner is riveted on achieving perfection.

Table 1: *Candidate Parameter Values*

	M1	M2	M3	Major Per K	Minor Per K	DD	TL	Prep Min	Conduct Min	Participants
DSE	2-4	2-4	2	2.5	10	0.95-0.85	0.025-0.0075	500	120	4
SSE	5-7	5-7	3	5	20	0.70-0.80	0.075-0.150	400	120	4
AHP	8-10	8-10	4	10	40	0.50-0.65	0.175-0.250	300	120	4

Table 2: *Computing Return on Investment*

	DD	M1	TL	M2	M3	Net Savings	Detection Cost	ROI Per K	Leaks
DSE Disciplined Software Engineering	0.95	2	0.025	2	2	12.49	16.33	0.76	0.3125
	0.90	3	0.050	3	2	15.26	16.33	0.93	0.6250
	0.85	4	0.075	4	2	18.55	16.33	1.14	0.9375
SSE Structured Software Engineering	0.80	5	0.100	5	3	65.00	14.67	4.43	2.5000
	0.75	6	0.125	6	3	74.38	14.67	5.07	3.1250
	0.70	7	0.150	7	3	85.23	14.67	5.81	3.7500
AHP Ad Hoc Programming	0.65	8	0.175	8	4	234.80	13.00	18.06	8.7500
	0.60	9	0.200	9	4	261.20	13.00	20.09	10.0000
	0.55	10	0.225	10	4	288.75	13.00	22.21	11.2500
	0.50	10	0.250	10	4	285.00	13.00	21.92	12.5000

Summary

In studying the issues associated with the new realities of the workplace and the new software engineering responses to the issues, the answer lies in first understanding what the enterprise is trying to do, and then in how the enterprise does it.

What the enterprise is trying to do revolves around the management of commitments and the drive toward product perfection. The management of commitments is primarily associated with time to market but also performance to budget. The drive toward product perfection is associated with satisfying and delighting the customer with a continuous stream of the right capabil-

ities and features packaged in a defect-free container.

How the enterprise does this revolves around its software product engineering practice. The three modes of practice in software product engineering include AHP, SSE, and DSE.

What is actually occurring is a competition among stresses, not all of which can be satisfied. In reasoning about new software engineering, it is important to explicitly acknowledge that choices must be made. The drive toward perfection clashes with the drive to achieve time to market. In the short-term environment of today, the successful enterprise makes the strategic selection and accepts any collateral damage.

When an organization has superior knowledge of the parameter values for software inspections return on investment, it is able to derive its own return-on-investment metric. To perform this computation, simply visit the tool at <http://members.aol.com/ONeillDon/nsqe-roi.html>.◆

References

1. McGibbon, T. "A Business Case for Software Process Improvement." Rome Laboratory DACS Report, 30 Sept. 1996.
2. O'Neill, Don. "Peer Reviews." Encyclopedia of Software Engineering. Wiley Publishing, Inc., Jan. 2002.
3. O'Neill, Don. "National Software Quality Experiment: A Lesson in Measurement 1992-1997." CrossTalk 11.12 (Dec. 1998) <www.stsc.hill.af.mil/crosstalk/frames.asp?uri=1998/12/oneill.asp>.
4. O'Neill, Don. National Software Quality Experiment Description and Data Summaries <http://members.aol.com/ONeillDon/nsqe-results.html>.
5. Paulk, Mark C. The Capability Maturity Model: Guidelines for Improving the Software Process. Reading, MA: Addison-Wesley, 1995: 270-276.
6. Prowell, Stacy J., Carmen J. Trammell, Richard C. Linger, and Jesse H. Poore. Cleanroom Software Engineering: Technology and Process. Addison Wesley Longman, 1999: 17, 33-90.
7. Humphrey, Watts. Introduction to the Personal Software Process. Reading, MA: Addison-Wesley, 1997.
8. Wells, J. Donovan <www.extremeprogramming.org>.
9. Basili, Vic, and Barry Boehm. "Top Ten Defect Reduction List." IEEE Software Jan. 2001.
10. Lindner, Richard J., and D. Tudahl. Software Development at a Baldrige Winner. Proc. of ELECTRO '94, Boston, MA, 12 May 1994: 167-180.
11. O'Neill, Don. "National Software Quality Experiment: Results 1992-1999." Software Technology Conference, Salt Lake City, UT, 1995, 1996, and 2000.
12. O'Neill, Don. National Software Quality Experiment: A Lesson in Measurement 1992-1997. First International Software Assurance Certification Conference. Chantilly, VA, 1 Mar. 1999: 1-14.

About the Author

Don O'Neill is a 43-year veteran software engineering manager and technologist currently serving as an independent consultant. He conducts defined programs for managing strategic software improvement, including implementing an organizational software inspections process, directing the National Software Quality Experiment, implementing software risk management on the project, conducting the Project Suite Key Process Area Defined Program, and conducting Global Software Competitiveness Assessments. O'Neill is a founding member of the Washington, D.C.-based Software Process Improvement Network and the National Software Council and serves as the executive vice president of the Center for National Software Studies. He has a bachelor's of science degree in mathematics form Dickinson College, and has completed a three-year residency at Carnegie Mellon University's Software Engineering Institute.

9305 Kobe Way
Montgomery Village, MD 20886
Phone: (301) 990-0377
E-mail: oneilldon@aol.com