

NATIONAL SOFTWARE QUALITY EXPERIMENT A LESSON IN MEASUREMENT 1992-1997

KEY WORDS

Analysis Bins
Common problems
Core samples
Defect types
Experiment participants
Software Inspection Lab
Software process maturity level
Standard of excellence
Return on investment

PROLOGUE

The nation's prosperity is dependent on software. The nation's software industry is slipping, and it is slipping behind other countries. The National Software Quality Experiment is riveting attention on software product quality and revealing the patterns of neglect in the nation's software infrastructure.

ABSTRACT

In 1992 the DOD Software Technology Strategy set the objective to reduce software problem rates by a factor of ten by the year 2000. The National Software Quality Experiment is being conducted¹ to benchmark the state of software product quality and to measure progress towards the national objective.

The National Software Quality Experiment is a mechanism for obtaining core samples of software product quality. A micro-level national database of product quality is being populated by a continuous stream of samples from industry, government, and military services. This national database provides the means to benchmark and measure progress towards the national software quality objective and contains data from 1992 through 1997.

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 uniform application of the experiment and the collection of consistent measurements are guaranteed through rigorous training of each participant. Thousands of participants from dozens of organizations are populating the experiment database with thousands of defects of all types along with pertinent information needed to pinpoint their root causes.

To fully understand the findings of the National Software Quality Experiment, the measurements taken in the lab and the derived metrics are organized along several dimensions including year, software process maturity level, organization type, product type, programming language, and industry type. These dimensions provide a framework for populating an interesting set of analysis bins with appropriate core samples of software product quality.

¹ The National Software Quality Experiment is an entrepreneurial activity.

EXPERIMENT MOTIVATION AND ORGANIZATION

Overview

Participants are attracted to the experiment as a place where they can calibrate their software quality against appropriately selected industry core samples. Here they can jump-start the organization's quality measurement program on the shoulders of uniform Software Inspection Lab procedures. These procedures are operationally packaged for project use and include well defined processes, industrial strength product checklists, participant roles and behaviors, and standard forms and reports.

The National Software Quality Experiment provides the framework to pose important quality questions. Its micro-level national quality database provides the measurements to answer them. Similarly, the extent of certain common risks can be quantified. As a participant in the experiment, an organization can characterize the effectiveness of its software quality process. At the industry level, progress towards the national software quality objective can be benchmarked.

Participants in the experiment benefit in several ways. They are able to characterize the maturity of their software quality process. With this understanding, they are able to establish goals for improving the process and to set priorities for immediate action. Beyond that, these organizations are able to promote a vision for excellence in their software products and to calibrate their progress towards the national software quality goal.

Motivation

The Department of Defense Software Technology Strategy was drafted for the Director of Defense Research and Engineering in December 1991 [DOD STS 91]. Three important national objectives were established to be achieved by the year 2000:

1. Reduce equivalent software life-cycle costs by a factor of two
2. Reduce software problem rates by a factor of ten
3. Achieve new levels of mission capability and interoperability via software

Every software organization should treat the national objective to improve software product quality by a factor of ten as a wake-up call. Are organizations planning to reduce software problem rates by a factor of ten? Do they know what these rates are now?

Measurement Best Practice

Although measurement is needed to derive effective policy governing acquisition, development, and operations, there is not yet an industry consensus on the wisdom of creating a national database for software engineering. The issue centers on the use of the data, not on its collection. The worry is that the industry is not ready to use the database appropriately. Clearly the industry can learn to use the database appropriately once it exists. If there are national goals set for software engineering, there must also be a national measurement program and database to track progress and refine goals. Carnegie Mellon University's Software Engineering Institute produced "A Concept Study for a National Software Engineering Database" in July 1992 [Van Verth 92]. The study points out that there are many users for such a database, but few suppliers. The study offers the following observations and advice on establishing a national database:

1. Wide variance may exist in the collection process
2. Common data definitions are needed
3. Goals and questions should precede data collection
4. Confidentiality of the data must be protected

In designing the experiment, it is recognized that the prescription for achieving lasting value in measurement depends on the successful integration of measurement concepts, operationally defined and packaged processes, effective technology transition including the training of participants and the dissemination of results, and hands-on oversight of the experiment. The prescription for lasting value in measurement revolves around four driving measurement concepts. First, measurement must be aligned with business needs. Second, metrics must be carefully pinpointed and rigorously defined. Third, measurement activities must be built into the normal operation of the organization. Finally, extraordinary steps must be applied to obtain consistency and uniformity in data collection.

Finally, Dr. Vic Basili of the University Maryland provides the following additional guidelines [Wallace 97]:

1. Establish the goals of the data collection
2. Develop a list of questions of interest
3. Establish data categories
4. Design and test the data collection form
5. Analyze data

Nature and Role of Experiment

In the practice of software engineering, managers are guided more by myth than by measurement. The experiment provides the framework for measuring critical aspects of software product quality practice. The framework supplies the ingredients needed to install a uniform and consistent measurement methodology. These are described in the Software Inspections Mechanism. The predictability of the measurements taken in conducting the experiment provides the basis for assessing the validity of a hypothesis. This is discussed in Experiment Results. Some of the questions asked and answered in the experiment are:

1. To what extent is there a continuing stream of requirements changes?
2. What are the leading types of errors?
3. Are errors traced to people or process?
4. Is a standard development process followed?
5. To what extent are wrong software functions being developed?
6. To what extent are there shortfalls in real time performance?
7. Is gold plating a problem?

Software inspections are an essential ingredient in fact-based software management. They utilize a reasoning process for conducting a fine-grained, deep-probing evaluation. When combined with automation-based quick-look evaluations, the best balance between efficiency and insight can be obtained. Once installed in the organization, the software inspection process yields core samples of software product quality. These can be used to benchmark problem rates by defect type among major product areas within the organization. With the benchmark measurements in place, the software inspections process provides a stable, uniform, and persistent mechanism for measuring improvement progress toward the software product goals of the organization.

SOFTWARE INSPECTIONS MECHANISM

Setting the Standard of Excellence

Software products reveal the standard of excellence in software engineering applied in their production. In improving software product quality, an industrial strength standard of excellence must be set, and the software operations within the organization must be disciplined to meet that standard. This is done by measuring actual practice using the strongly preferred indicators from the national standard of excellence spanning completeness, correctness, style, rules of construction, and multiple views.

Completeness

Completeness is based on traceability among the requirements, specification, design, code, and test artifacts. Completeness analysis reveals what predecessor artifact sections have not been satisfied as well as the inclusion of extra fragments.

Correctness

Correctness is based on reasoning about programs through the use of informal verification and correctness questions derived from the prime constructs of structured programming and their composite use in proper programs. Input domain and output range are analyzed for all legal values and all possible values. Adherence to project specific disciplined data structures is analyzed.

Style

Style is based on project specified style guidance based on block structured templates. Semantics of the design and code are analyzed for correspondence to the semantics used in the requirements, specifications, and design. Naming conventions are checked for consistency of use; and commentary, alignment, upper/lower case, and highlighting use are checked.

Rules of Construction

Rules of construction are based on the software architecture and its specific protocols, templates, and conventions used to carry it out. For example, these include interprocess communication protocols, tasking and concurrent operations, program unit construction, and data representation.

Multiple views

Multiple views are based on the various perspectives required to be reflected in the product. During execution many factors must operate as intended including initialization, timing of processes, memory use, input and output, and finite word effects. In generating the software, packaging considerations must be coordinated including program unit construction, program generation process, and target operations. Product construction disciplines of systematic design and structured programming must be followed as well as interfaces with the user, operating system, and physical hardware.

EXPERIMENT RESULTS

Experiment Participants

The participants of the National Software Quality Experiment have been trained in the Software Inspections Course and Lab. Experiment results are drawn from these Inspection Lab sessions. The participating organizations span government, DOD industry, and commercial sectors and represent a wide range of application domains.

- Accounting, personnel, administration
- Administrative and management decision support
- Aircraft jet engine diagnostics, logistics, and maintenance
- Artillery fire control system
- Avionics flight on-board control
- Control devices for avionics applications
- Credit card application
- Department of State embassy support
- Electronic commerce
- Electronic warfare
- FAA communications
- Factory line support
- Financial services
- Global positioning system user sets
- Insurance and medical information
- International banking
- Joint Chiefs of Staff support
- Medical information system
- Naval surface weapons system
- Pre and post flight space application
- Telecommunications

Results Summary

Ralph Waldo Emerson said, "The years teach us things the days never knew". The National Software Quality Experiment has been accumulating a steady stream of core samples for its micro-level national database. These results provide a benchmark of software product quality measurements useful in assessing progress towards the national software quality objective for the year 2000. These results are highlighted below in the discussion of the common problems pinpointed, defect category and severity data summary, Inspection Lab operations, the predictability of certain measurements, and the ranking of defect types.

Common Problems

Analysis of the issues raised in the experiment to date has revealed common problems that reoccur from session to session. Typical organizations which desire to reduce their software problem rates should focus on preventing the following types of defects:

1. Software product source code components are not traced to requirements.
 - *As a result, the software product is not under intellectual control, verification procedures are imprecise, and changes cannot be managed.*
2. Software engineering practices for systematic design and structured programming are applied without sufficient rigor and discipline.
 - *As a result, high defect rates are experienced in logic, data, interfaces, and functionality.*
3. Software product designs and source code are recorded in an ad hoc style.
 - *As a result, the understandability, adaptability, and maintainability of the software product are directly impacted.*
4. The rules of construction for the application domain are not clearly stated, understood, and applied.
 - *As a result, common patterns and templates are not exploited in preparation for later reuse.*

Defect Category and Severity

The defect severity metric revealed that 14.27% of all defects were major, and 85.73% minor. Defect category distinguishes missing, wrong, and extra. For major defects, 7.44% were missing, 5.95% wrong, and .88% extra. For minor defects, 49.76% were missing, 27.63% wrong, and 8.32% extra.

Defect Severity and Category Summary

	Missing	Wrong	Extra	Total
Major	7.44	5.95	.88	14.27
Minor	49.76	27.63	8.32	85.73
Total	57.20	33.60	9.20	100.00

Inspection Lab Operations

Through 1997 there have been 112 Inspection Labs in which 2317 participants were trained and conducted inspection sessions. A

total of 788,459 source lines of code have received strict and close examination using the packaged procedures of the lab. There have been 142,306 minutes of preparation effort and 52,196 minutes of conduct time expended to detect 11,375 defects.

Of these 11,375 defects, 1854 were classified as major, and 9521 as minor. A major defect effects execution; a minor defect does not. It required 12.51 minutes of preparation effort on the average to detect a defect. To detect a major defect required 76.76 minutes of preparation effort on the average. On the average, .906 thousand source lines of code were examined each inspection conduct hour. There were 2.35 major defects detected in each thousand lines, and 12.08 minor defects. There were 4.91 defects detected per session with a return on investment of 4.48.

INSPECTION LAB OPERATIONS					
Sessions	Prep Effort	Conduct Time	Major Defects	Minor Defects	Size in Lines
2317	142,306	52,196	1854	9521	788,459
Metrics:					
1.	12.51	minutes of preparation effort per defect			
2.	76.76	minutes of preparation effort per major defect			
3.	2.35	major defects per KSLOC			
4.	12.08	minor defects per KSLOC			
5.	906	lines per conduct hour			
6.	4.91	Defects per session			
7.	4.48	Return on Investment			

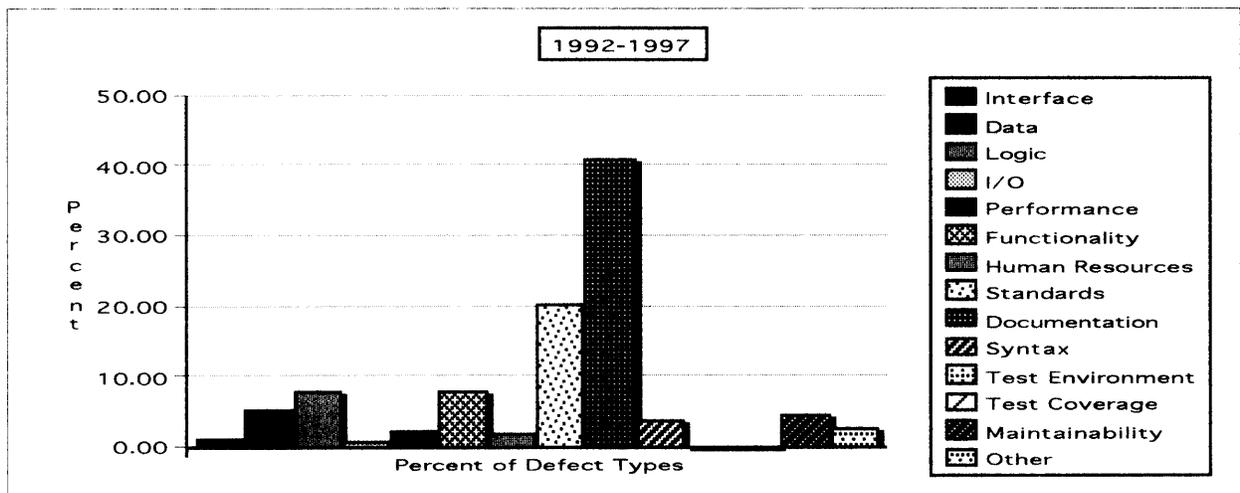
Questions Answered in the Lab

The micro-level national database on software product quality can be used to answer important software engineering questions. When appropriately selected core samples are accumulated in the Report Summary Form and the probability of occurrence is computed for each defect type, defect severity, and defect category, these probabilities can be used to construct answers to questions. Five of Boehm's top ten risks are answered below using the 1992-1997 data from the experiment:

Defect Type Ranking

The foremost defect types that accounted for 90% of all defects detected are:

Documentation	40.86%	error in guidance documentation
Standards	20.39%	error in compliance with product standards
Functionality	7.95%	error in stating or meeting intended
Logic	7.86%	error revealed through informal correctness questions
		function
Data	5.36%	error in data definition, initial value setting, or use
Maintainability	4.73%	error in good practice impacting the supportability
		and evolution of the software product
Syntax	4.02%	error in language defined syntax compliance



To what extent were the wrong software functions being developed?

Functionality errors accounted for 7.95% of all errors.

To what extent were the wrong user interfaces developed?

Interface errors accounted for 1.05% of all errors.

Human Factors accounted for 1.79% of all errors.

To what extent was there gold plating?

9.20% of all errors were classified as extra.

To what extent was there a continuing stream of requirements changes?

Documentation errors accounted for 40.86% of all errors.

To what extent was there a shortfall in real time performance?

Performance errors accounted for 2.39% of all errors.

Questions Not Yet Answered

It is useful to keep in mind that defects detected do not equal defects inserted. Defects may go undetected and leak into downstream activities. Consequently there is interest in defect leakage and ways to measure and reason about it. The Software Inspection Lab includes a mechanism to collect data on defect leakage and to reason about the results. This reasoning process crosses over into defect prevention.

Defect leakage was introduced into the National Software Quality Experiment in 1995, and the data on this is starting to build up. The defect leakage data needs to populate each analysis bin in sufficient quantity before these results are usable. With this data it will be possible to conduct special studies on defect leakage to augment the core analyses done continuously.

Questions asked but not yet answered include:

1. To what extent is defect leakage occurring?
2. What is the frequency distribution of defect types that leak?

The mechanism used to gather defect leakage involves identifying the life cycle activity for each software inspection and the defect origin for each defect. Each software inspection is considered an exit criteria for a software product engineering activity. Each defect is characterized by category, severity, type, ... and defect origin. Defect origin is the software product engineering activity where the defect was inserted. Where defect origin does not match the software product engineering activity for which this inspection serves as an exit criteria, defect leakage has occurred.

Measurement Results By Analysis Bin

The findings of the National Software Quality Experiment are organized along several dimensions which provide a framework for populating an interesting set of analysis bins with appropriate core samples of software product quality. The analysis bins are used to organize the findings into collections of data that reveal distinctions and may suggest interesting trends. The types of bins selected are year, software process maturity (level 1,2,3), organization type (commercial, DOD industry, government), product type (embedded, organic), programming language (modern, old style), and industry type (defense, financial, manufacturing, medical, telecommunication, transportation). As data for each year is collected, the overall results become more interesting, and the population of analysis bins becomes more robust.

Return On Investment

Managers are interested in knowing the return on investment to be derived from software process improvement actions. The Software Inspections Process gathers the data needed to determine this.

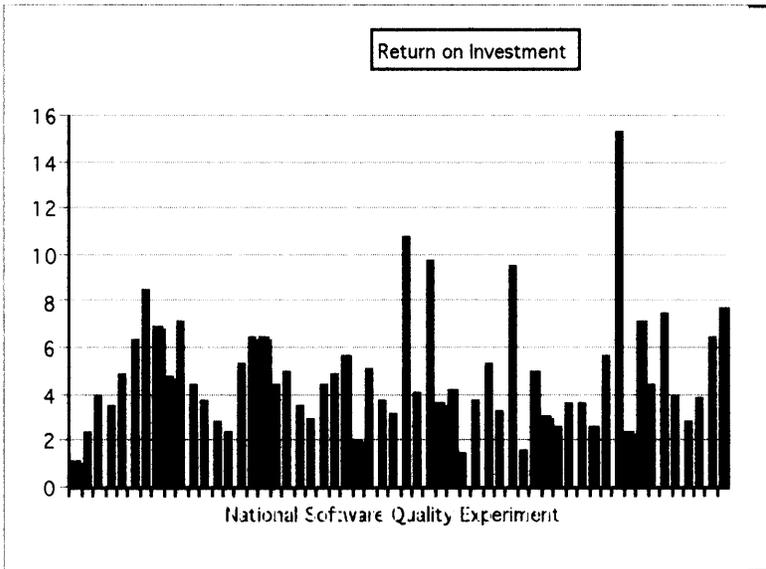
The defined measurements collected in the Software Inspections Lab may be combined in complex ways to form this derived metric. The Return on Investment for Software Inspections is defined as:

Savings/Cost, where:

$$\text{Savings} = (\text{Major Defects} * 9) + \text{Minor Defects}$$

$$\text{Cost} = (\text{Minutes of Preparation Effort} + (\text{Minutes of Conduct Time} * 4)) / 60$$

This model 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. A Major Defect that leaks into later phases may cost ten hours to detect and correct. Ten hours to fix later minus one hour to fix now results in the constant nine (9) applied to Major Defects. A Minor Defect may cost two hours to fix later minus one hour to fix now resulting in a constant of one (1) applied to Minor Defects. To convert the Minutes of Conduct Time to effort, the average number of participants (4) is applied. The constant 60 minutes is applied to convert minutes to hours.



The graph showing the Return on Investment for each organization participating in the National Software Quality Experiment suggests that the Return on Investment for software inspections ranges from 4:1 to 8:1. For every dollar spent on software inspections, the organization can expect to avoid 4-8 dollars on higher cost rework.

CONCLUSION

Closing Observations

In closing it needs to be stated that the data does not suggest progress towards the Year 2000 goal to reduce software problems by a factor of ten. Hunting for defects in software is a target rich opportunity. The harder the project looks for errors, the more it finds. The way to look harder is to reduce the volume of product inspected in each session.

The data suggests that increased software process maturity results in increased defect detection, with the result perhaps being lower defect leakage into the field. At level 1 the project lacks a shared vision for a standard of excellence for software engineering products. At level 2 attention is paid to establishing a standard of expectation, a standard of excellence, and so more defects are identified. At level 3 the standard is set and the well defined, fined grained processes for software product engineering are in place and in practice with software inspections operating as the exit criteria for each activity of the life cycle.

The data also suggests that defect density decreases with program size. As stated earlier, all programs contain a beginning, an end, and a context for operation within the larger system. Starting, finishing, and fitting in are all more error prone than the body of the program which gives it size.

In addition the data suggests that the organization's neglect of its software process exceeds the poor workmanship of individual programmers as the source of errors. Documentation and standards defect types account for nearly two-thirds of all defects, and these are the responsibility of the organization and its process.

Software products are not well connected to the requirements or business case that inspired their creation. Much of the documentation type defect detection results from the lack of traceability from the code to the design to the specification to the requirements.

Field Measurement Lessons

In conducting the National Software Quality Experiment, valuable lessons in field measurement are being learned. These lessons are forming the prescription for obtaining lasting value in measurement:

1. Measurement must be aligned with business and performance needs. These activities must be built into the normal operation of the organization. To do this, the goals to be met and questions to be answered in management, engineering, and operations must precede the collection of data.
2. Metrics must be carefully pinpointed and rigorously defined. Extraordinary steps must be applied to obtain consistency and uniformity. Without a well defined process for data collection and analysis, the variance in the measurement process itself impacts the accuracy of results.
3. Attention must be paid to the confidentiality of results. The opportunity for improvement is increased when the measured results are made more widely available. However, individuals and groups naturally resist having their shortcomings made public. If ignored, this resistance will defeat the measurement program. The organization must strike a balance between public and private data.

Next Steps

The National Software Quality Experiment is a demonstrated mechanism for collecting uniform and consistent measurements of software product quality. It provides the vantage point for software product quality and the field experience in measurement needed to jump start the practice of fact-based software management.

As the centerpiece of the experiment, the Software Inspection Labs have been installed in software factories around the country. The National Experiment collects, organizes, and packages core samples of software product quality. These measurements are increasing the understanding of the state of the practice and how to measure it.

The usefulness and success of the National Software Quality Experiment depends on sustaining a continuous stream of core samples. Organizations from industry, government, and the military are invited to participate and enrich this national database resource.

BIBLIOGRAPHY

- [DOD STS 91] Department of Defense Software Technology Strategy, draft prepared for the Director of Defense Research and Engineering [DDR&E], December 1991
- [Ebenau 94] Ebenau, Robert G. and Susan H. Strauss, "Software Inspection Process", McGraw-Hill, Inc., 1994
- [Fagan 76] Fagan, M., "Design and Code Inspections to Reduce Errors in Program Development", IBM Systems Journal, 15, 3, 1976, 182-211
- [Florac 92] Florac, William B., "Software Quality Measurement: A Framework for Counting Problems and Defects", CMU/SEI-92-TR-22, September 1992
- [Freedman 90] Freedman, D.P., G.M. Weinberg, "Handbook of Walkthroughs, Inspections, and Technical Reviews", Dorset House Publishing Co., Inc., 1990
- [Gilb 93] Gilb, Tom and Dorothy Graham, "Software Inspection", Addison Wesley Longman Limited, 1993
- [Linger 79] Linger, R.C., H.D. Mills, B.I. Witt, "Structured Programming: Theory and Practice", Addison-Wesley Publishing Company, Inc., 1979
- [Humphrey 89] Humphrey, Watts S., "Managing the Software Process", Addison-Wesley Publishing Company, Inc., 1989
- [O'Neill 88] O'Neill, Don and Albert L. Ingram, "Software Inspections Tutorial", Software Engineering Institute Technical Review 1988
- [O'Neill 89] O'Neill, Don, "Software Inspections Course and Lab", Training Offering for Practitioners, Software Engineering Institute, 1989
- [O'Neill 92] O'Neill, Don, "Software Inspections: More Than a Hunt for Errors", CrossTalk, Issue 30, January 1992
- [O'Neill 94] O'Neill, Don, "National Software Quality Experiment", International Conference on Software Quality, Washington DC, 1994
- [O'Neill 95,96] O'Neill, Don, "National Software Quality Experiment: Results 1992-1995", Software Technology Conference, Salt lake City, 1995 and 1996
- [O'Neill 97] O'Neill, Don, "Issues in Software Inspection", IEEE Software, Vol .14 No 1., January 1997
- [O'Neill 97] O'Neill, Don, "Setting Up a Software Inspection Program", CrossTalk, The Journal of Defense Software Engineering, Vol. 10 No. 2, February 1997
- [O'Neill 97] O'Neill, Don, "National Software Quality Experiment: A Lesson in Measurement 1992-1996", Quality Week Conference, San Francisco, May 1997 and Quality Week Europe Conference, Brussels, November 1997
- [O'Neill 98] O'Neill, Don, "Software Inspections and the Year 2000 Problem", CrossTalk, The Journal of Defense Software Engineering, Vol. 11 No. 1, January 1998
- [Paulk 95] Paulk, Mark C., "The Capability Maturity Model: Guidelines for Improving the Software Process", Addison-Wesley Publishing Company, 1995
- [Van Verth 92] Van Verth, Patricia B., "A Concept Study for a National Software Engineering Database", CMU/SEI-92-TR-23, July 1992
- [Wallace 97] Wallace, Dolores R., Laura M. Ippolito, and Herbert Hecht, "Error, Fault, and Failure Data Collection and Analysis", <http://hissa.ncsl.nist.gov>, Quality Week, San Francisco, May 1997

AUTHOR: Don O'Neill

Don O'Neill is a seasoned software engineering manager and technologist currently serving as an independent consultant. Following his twenty-seven year career with IBM's Federal Systems Division, Mr. O'Neill completed a three year residency at Carnegie Mellon University's Software Engineering Institute (SEI) under IBM's Technical Academic Career Program. There he developed a blueprint for charting software engineering evolution in the organization including the training architecture and change management strategy needed to transition skills into practice.

As an independent consultant, Mr. O'Neill conducts defined programs for managing strategic software improvement. These include implementing an organizational Software Inspections Process, implementing Software Risk Management, and conducting Global Software Competitiveness Assessments. Each of these programs includes the necessary practitioner and management training.

In his IBM career, Mr. O'Neill completed assignments in management, technical performance, and marketing in a broad range of applications including space systems, submarine systems, military command and control systems, communications systems, and management decision support systems. He was awarded IBM's Outstanding Contribution Award three times:

1. Software Development Manager for the Global Positioning Ground Segment (500,000 source lines of code) and a team of 70 software engineers within a \$150M fixed price program.
2. Manager of the FSD Software Engineering Department responsible for the origination of division software engineering strategies, the preparation of software management and engineering practices, and the coordination of these practices throughout the division's software practitioners and managers.
3. Manager of Data Processing for the Trident Submarine Command and Control System Engineering and Integration Project responsible for architecture selections and software development planning (1.2M source lines of code).

Mr. O'Neill served on the Executive Board of the IEEE Software Engineering Technical Committee and as a Distinguished Visitor of the IEEE. He is a founding member of the National Software Council and the Washington DC Software Process Improvement Network (SPIN). He is an active speaker on software engineering topics and has served as the Program Chairman and Program Committee member for several conferences. He has numerous publications to his credit. Mr. O'Neill has a Bachelor of Science degree in mathematics from Dickinson College in Carlisle, Pennsylvania.

Contact Information

Don O'Neill
Independent Consultant
9305 Kobe Way
Montgomery Village, Maryland 20886

Phone: (301) 990-0377
email: ONeillDon@aol.com
<http://members.aol.com/ONeillDon/index.html>

word count: 4,581