How to Avoid Software Inspection Failure and Achieve Ongoing Benefits

Roger Stewart and Lew Priven
Stewart-Priven Group

The objectives of this article are to examine why software inspections are not used more widely, identify the issues contributing to their lack of use, identify why inspection benefits deteriorate for many companies, and recommend what can be done to address and solve these issues. The proven benefits of inspections are too significant to let them fall by the wayside!

For the purpose of this article, an inspection is defined as a preemptive peer review of work products – by trained individuals using a well defined process – to detect and eliminate defects as early as possible in the Software Development Life Cycle (SDLC) or closest to the points of defect injection.

Background

According to a National Institute of Standards and Technology (NIST) study, *the problem of continued delivery of bug-ridden software* is costing the U.S. economy an estimated $59.5 billion each year. The study also found the following:

> …although all errors cannot be removed, more than a third of these costs, or an estimated $22.2 billion, could be eliminated by an improved testing infrastructure [reviews, inspections, etc.] that enables earlier and more effective identification and removal of software defects. These are the savings associated with finding an increased percentage [but not 100 percent] of errors closer to the development stages in which they were introduced. Currently, over half of all errors are not found until 'downstream' in the development process (testing) or during post-sales software use. [1]

Figure 1 shows a typical relationship between the costs of repairing a defect in a given phase of the development cycle versus which phase the defect was introduced. This relationship gives rise to the development costs described in the NIST report.

The following testimonials answer the question: *What is the evidence that inspections address the cost and quality issues described earlier but are not widely used correctly to maximize defect detection and removal?*

- The data in support of the quality, cost, and schedule impact of inspections is overwhelming. They are an indispensable part of engineering high-quality software. [3]

- Inspections are surely a key topic, and with the right instrumentation and training they are one of the most powerful techniques for defect detection. They are both effective and efficient, especially for up-front activities. In addition to large-scale applications, we are applying them to smaller applications and incremental development (Chris Ebert). [3]

- Inspection repeatedly has been demonstrated to yield up to a 10-to-1 return on investment. . . . depressingly few practitioners know about the 30-year-old technique of software inspection. Even fewer routinely perform effective inspections or other types of peer reviews. [4]

- Formal inspections can raise the [defect] removal efficiency to over 95 percent. But part of the problem here is that not a lot of companies know how to use these things. [5]

- The software community has used inspections for almost 28 years. During this timeframe, inspections have consistently added value for many software organizations. Yet for others, inspections never succeeded as well as expected, primarily because these organizations did not learn how to make inspections both effective and low cost. [6]

- I continue to be amazed at the number of software development organizations that do not use this powerful method [inspections] to improve quality and productivity. [7]

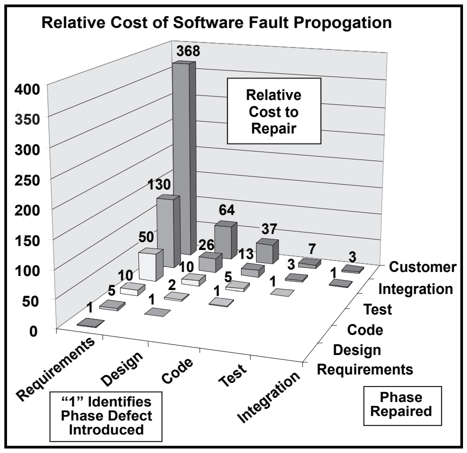
It is clear from these testimonials that inspections are the most effective way to improve the quality, schedule, and cost of developing software, but after all the years after their introduction, *why* are they not an integral part of all software development life cycles?

The authors of this article, Roger Stewart and Lew Priven each spent more than 20 years developing projects that used inspections and, for the past eight years, each has trained a wide variety of companies in the use of Fagan inspections. They consistently observed that soon after inspection training completes, *malicious compliance* sets in by critical inspection execution deviations being introduced and/or ineffective *shortcuts* being employed. This results in inspection benefits being compromised, leads to limited use or discontinuation, and allows too many defects to escape to later, more costly phases of test and customer use.

Back to Basics

In order to deal with the problem of inspections not being widely used (or not used correctly for the maximum benefit), we need to go back and look at the original approach. Inspections were an outgrowth of the quality message from gurus W. Edwards Demming and J.M. Juran to design in quality at the beginning of the development process, instead of *testing in*

Figure 1: *Cost of Fixing a Defect* [2]



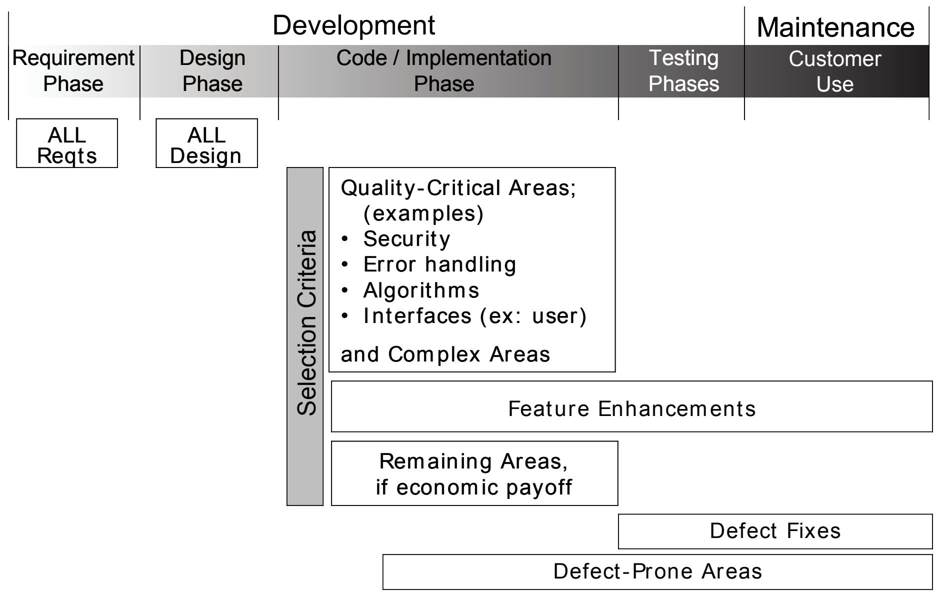


Figure 2: *Prioritizing What to Inspect*

pseudo-quality at the end of the production line.

What naturally followed was the idea of applying sampling quality control techniques to the software development life cycle as if it were a production line. Specifically, this involves sampling the product periodically (detect defects), making adjustments as defects are found (fix defects and improve the development process), and predicting the shipped product quality based on the results of the sampling.

Application of the sampling quality control techniques to the software development cycle led to the development of the software inspection process. The most widely known and practiced inspection process was introduced to the IBM software community in 1972 by a team led by Michael Fagan and managed by Lew Priven (co-author of this article) [6].

In the case of software, the development life cycle is the production line, and inspections are the sampling and prediction technique. Inspections are the vehicle to sample the product in the earlier phases of the development life cycle to detect and fix defects closest to the point of injection, and the data collected from inspections can be used as the basis for predicting the quality of the delivered product.

How Have Inspections Evolved?

In 1972, Priven published an IBM Technical Report which described a software development management system including *points of management control* using process monitors that evolved into inspections [8]. The management system was based on a well-defined development process – which satisfied the need for a production line as described earlier. With the *production line* in place, Priven hired Michael Fagan, a quality engineer with a hardware and manufacturing background, to work with the development team to find a way to improve the quality of delivered software [6-10]. The IBM (Fagan) Inspection Process then evolved as a critical component of the end-to-end software development life cycle. Over the years, the integration of inspections into the software development life cycle has been lost as the *inspection process came to be viewed as a standalone quality process with inspection execution becoming the prime focus*. However, the supporting infrastructure of a software development life cycle is still critical to successfully implementing inspections.

Why Is the Supporting Development Infrastructure Important?

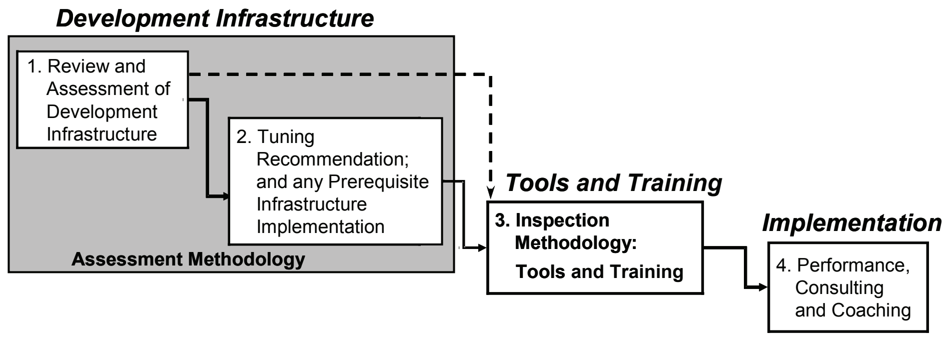
The supporting infrastructure of a well-defined development process is important because it requires management at all levels – and during all development phases – to actively support the inspection process. A life-cycle view is needed because the cost and schedule impact are primarily borne by the requirements, design, and implementation components of the organization. However, while these components also realize some of the reduced cost, higher quality, and improved schedule benefits, the majority of these benefits are primarily realized in testing and maintenance.

Theory Is Good, but Why Are Inspections Not Embraced?

In addition to being viewed as a standalone process, which lacks a life-cycle view of investment and associated savings, inspections have also been characterized by a number of myths. These myths discourage implementation. While there is a kernel of truth in each myth, each can be turned into a positive. Some examples follow:

- **Inspections are time consuming.** Yes, they add up-front development time to requirements, design and code. However, rather than being viewed as a problem, this additional up-front time for inspections should be viewed as an investment in obtaining the overall quality, cost, and schedule benefits over the project's life cycle.
- **Inspections are bureaucratic and one size fits all.** System engineers and software engineers, with support from management, need to have the flexibility to adjust their inspection process to the needs of the product under development. For example, the difference between inspecting software to control a jet fighter (where a defect could be a matter of life and death) and software that displays a Web form (where the impact of a defect may be an inconvenience). The former may require a broader comprehensive set of inspections while the latter could employ other visual analysis techniques to supplement a base set of inspections.
- **All work products must be inspected.** There is a lack of guidance on when, where, and how to start an inspection process. An approach to prioritizing what work products to

Figure 3: *Assessment Methodology*


Development Infrastructure
1. Review and Assessment of Development Infrastructure
2. Tuning Recommendation; and any Prerequisite Infrastructure Implementation
Assessment Methodology
Tools and Training
3. Inspection Methodology: Tools and Training
Implementation
4. Performance, Consulting and Coaching

inspect needs to be intelligently applied.

While these are myths that we typically hear about inspections, upon further examination they are symptoms of a much larger underlying set of issues. The remainder of the article will focus on dealing with those *issues* which we will later refer to as *inspection pitfalls*.

A Realistic Approach

Although complete inspection coverage may be ideal, a realistic approach to inspections is to formulate a set of *selection criteria* (see Figure 2). These criteria guide the identification of those areas of the product most critical to success or where problems are most likely to occur. At the least these areas should be inspected. This addresses the common complaint that there is not enough time to integrate inspections into tight schedules yet allows for using inspections for finding defects where they are most likely to cause problems.

Figure 2 addresses this no-time issue by showing the prioritization of *what to inspect* related to the development cycle phases of the project. There should be a strong focus on requirements and design, which are the most costly to fix when discovered later in the development cycle (see Figure 1). The focus on requirements and design is particularly important because our experience has shown that the largest numbers of defects are injected during these two phases of development. One example from a TRW study shows about 52 percent of defects are injected in requirements and 28 percent are injected in design [11].

The most successful implementations of inspections have been in organizations that have multi-level active management support of inspections and a well-defined development life cycle with pre-existing emphasis on planning, monitoring, and

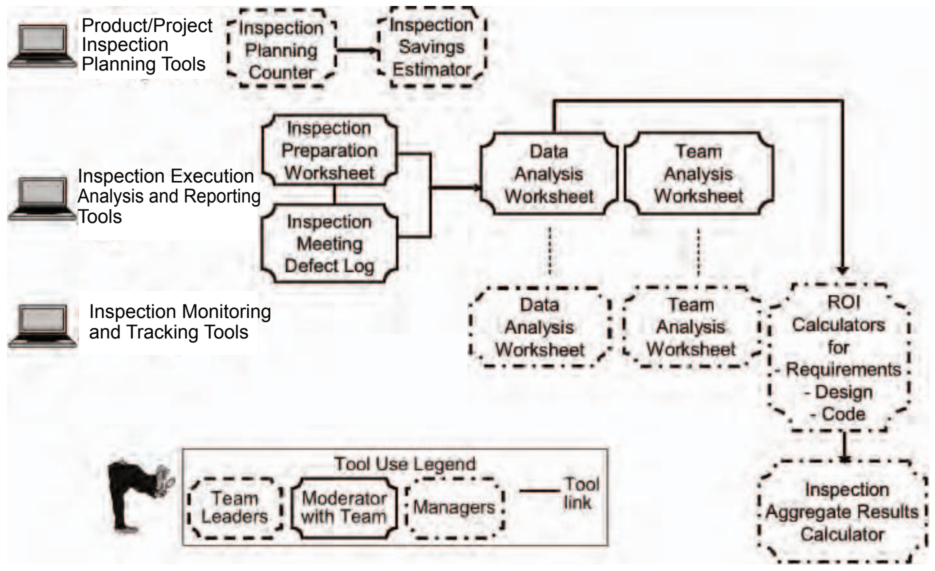

Product/Project Inspection Planning Tools
Inspection Planning Counter
Inspection Savings Estimator
Inspection Execution Analysis and Reporting Tools
Inspection Preparation Worksheet
Inspection Meeting Defect Log
Data Analysis Worksheet
Team Analysis Worksheet
Inspection Monitoring and Tracking Tools
Data Analysis Worksheet
Team Analysis Worksheet
ROI Calculators for Requirements - Design - Code
Tool Use Legend
Team Leaders
Moderator with Team
Managers
Tool link
Inspection Aggregate Results Calculator

Figure 4: *Computerized Inspection Tool Overview*

Table 1: *Risks Associated With Inspection Pitfalls*

No.	Pitfall	Pitfall Risks	Pitfall Solutions
1	Lack of supportive SDLC infrastructure	• Immature practices for planning, data collection, reporting, monitoring, and tracking • Leads to Pitfalls #3, 4, 6, 10	• Assessment methodology ◦ Step 1: Assess client SDLC ◦ Step 2: Recommend any changes
2	Poor management understanding of the Inspection Process, its benefits, and their responsibilities	• Leads to Pitfalls #4, 6, 8, 9 ◦ Inadequate inspection: schedules, tools facilitation, implementation	• Upper management overview • Management performance class • Student feedback from inspection class
3	No computerized management-planning tools	• Inadequate schedule time (Pitfall #4) • No savings appreciation, leads to no inspections or too few inspections	• Planning counter tool • Savings/cost-estimator tool
4	Too little schedule time for inspections	• Defects escape to more costly phases to fix • Inspections not correctly executed • Leads to malicious compliance	• Inspection planning counter tool • Management performance class • Criteria for prioritizing what to inspect
5	No computerized inspector tools	• Inconsistency, compromising shortcuts • Defects, escape to more costly phases to fix	• Preparation tool • Inspection meeting tool • Data analysis tool, team analysis tool
6	Inadequate monitoring of inspection execution and tracking of results	• Inspection process execution deteriorates • Defects escape to more costly phases to fix • Employees lose interest when savings summaries are not periodically shared	• ROI calculators for text and code • Data analysis tool, team analysis tool • Aggregate results calculator tool
7	No post-class practitioner refresher	• Process misunderstood, compromising shortcuts introduced, defects escape	• Seminar for previous students • Inspection role-reference card • Inspection product checklist kit
8	No inspection facilitator/ project champion	• Inspection process issues not addressed, coordinated, or resolution disseminated • Inconsistent or incorrect inspection execution • Little useful feedback to management	• Upper management overview • Management performance class
9	Slow inspection implementation by project teams	• Ineffective start or no-start occurs • Inspection training forgotten; incorrect execution	• Inspector training accommodating multiple classes, of 2 days or less, per week
10	No inspection process capture	• Process misunderstood, inconsistent execution, defects escape • No repository for project lessons learned	• Course material tailoring • Inspection process capture tool

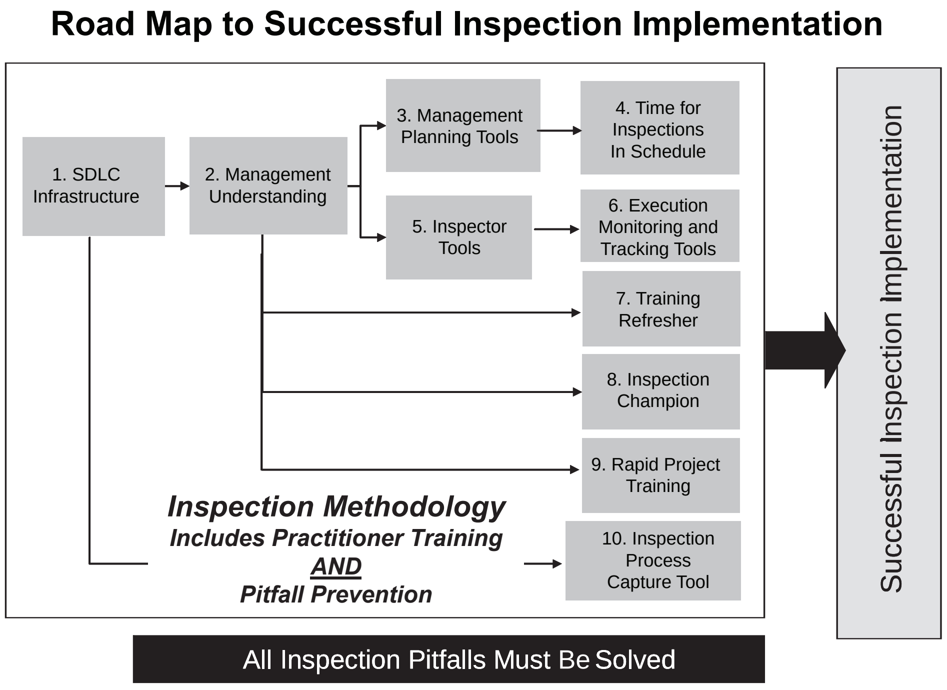

Road Map to Successful Inspection Implementation
1. SDLC Infrastructure
2. Management Understanding
3. Management Planning Tools
4. Time for Inspections In Schedule
5. Inspector Tools
6. Execution Monitoring and Tracking Tools
7. Training Refresher
8. Inspection Champion
9. Rapid Project Training
10. Inspection Process Capture Tool
Inspection Methodology
Includes Practitioner Training
AND
Pitfall Prevention
Successful Inspection Implementation
All Inspection Pitfalls Must Be Solved

Figure 5: *Inspection Infrastructure*

measurements use.

Development Infrastructure to Support Inspections

There is a lot of guidance on the structure of inspections such as the Institute for Electronics and Electrical Engineers (IEEE) Standard 1028-1997 [12] and how to conduct an inspection, but little guidance on the following:

1. How to select what to inspect (see Figure 2).
2. How to develop an appropriate software development life-cycle infrastructure that provides the necessary framework for successful implementation of inspections.
3. How to determine what computerized tools are needed to ensure proper inspection execution and management visibility into results, project savings, and return on investment (ROI). For example, because of the lack of inspection tools, data collection – which is too often left to the discretion of the inspection teams – is often not performed or is performed inconsistently. Therefore, data needed to evaluate inspection effectiveness is not easily available to management.

These three items will be discussed further in the next section.

Successful inspection implementation requires a software development life-cycle infrastructure that demands planning, data collection, reporting, monitoring, and tracking. Introducing inspections into a project culture that does not believe in and have a development infrastructure that actively supports these activities is fraught with risk.

Developing an appropriate infrastructure begins with selecting a framework upon which to build your development life cycle. A widely accepted framework is the Capability Maturity Model® (CMM®), and its successor CMM-Integration (CMMI®). However, as Watts Humphrey points out in [13], "Although the CMM [and CMMI] provides a powerful improvement framework, its focus is necessarily on what organizations should do – not how they should do it."

There are four key steps to filling out the development framework:

1. Select a development model (e.g., iterative, incremental, waterfall, spiral, agile).
2. Clearly define the development life cycle by identifying and recording for each process within the life cycle, its required inputs, the input's entrance criteria, the *what and how* of the process, the expected outputs, and the output's exit criteria.
3. Get process agreement by all components of the development organization (e.g., requirements generators, designers, coding/implementers, testers, etc.).
4. Determine which project tools will be used for planning, data collection, reporting, monitoring, and tracking (tool examples are critical path, earned value, etc.).

When these steps are completed, the introduction of inspections has the necessary framework (i.e., development infrastructure) for ongoing success and for inspections to be accepted as a very integral part of the end-to-end development life-cycle.

How an Inspection Methodology Can Reinvigorate Inspections

Inspections will only be successful long term if they are integral to a well-defined development process that has active management support in each phase of development. The methodology shown in Figure 3 (see page 25) starts with an assessment to ensure an adequate development life-cycle infrastructure is in place prior to inspection training. Steps 1 and 2 in Figure 3 are the assessment steps.

Once the development infrastructure is in place, what else needs to be done? Based on our experience in training more than 5,000 inspectors in companies at more than 50 locations, evaluating the data collected and observing the ongoing implementation or lack thereof, we have identified 10 inspection pitfalls, each of which inhibits inspection implementation. These inspection pitfalls must be resolved to achieve lasting benefits from inspections.

The 10 inspection pitfalls and associated risks are shown in Table 1 (see page 25). Note: The lack of a well-defined SDLC infrastructure, discussed earlier, is the first pitfall.

Table 1 identifies how each inspection pitfall leads to findable defects not being discovered with inspections, resulting in the following:

A. Development cost savings are not fully realized.
B. Quality improvements are not fully achieved.
C. Maintenance and support savings are not realized.
D. Inspections could become a total cost, not a savings.

We distinguish between the development life-cycle infrastructure – within which inspections fit (previous section), and the inspection infrastructure – which enables proper inspection execution for achieving the maximum benefit. An enabling inspection infrastructure must address all 10 pitfalls and would consist of the following:

1. Computerized management tools for use in planning inspections and predicting the overall project costs and savings from applying inspections (see Figure 4 on page 25 for an inspection tool overview example).
2. Computerized tools to aid inspectors

® CMMI is registered in the U.S. Patent and Trademark Office by Carnegie Mellon University.

in correctly and consistently performing inspections, gathering inspection related data, and performing analysis to identify how future inspections can be improved.
3. Monitoring and analysis computerized tools for management's post-inspection evaluation of individual inspections.
4. Computerized management tools for analyzing inspection ROI and an aggregate calculator for assessing and tracking the resulting project savings from multiple inspections.
5. An inspection process that provides flexibility for prioritizing what to inspect as shown in Figure 2.
6. Ability to have customized training material which incorporates your terminology and is based on your needs.
7. Rapid training (two days or less) of project personnel in a comprehensive training course with significant focus on requirements and design.
8. An overview briefing for upper management along with a more rigorous management performance course so upper managers and project leaders can fully understand the inspection process, its benefits, and their responsibilities
9. Follow-up practitioner refreshers to deal with any implementation problems – focused on making inspection implementation successful both initially and long-term.
10. An inspection process capture tool to enable inspections to quickly become an integral part of a company's SDLC infrastructure.

Table 1 shows how an *Inspection Methodology* can solve and prevent the 10 inspection pitfalls.

The Road Map to Success

The pitfall solution road map in Figure 5 shows the solution relationships that provide for successful software inspection implementation that will endure over the long term. The pitfall solutions provide the inspection infrastructure that, together with a comprehensive inspector training program, forms an inspection methodology for achieving a lasting and successful inspection program.

Summary

Our experience has shown us that inspections can live up to their potential and be embraced by the development community if the following happens:

- Inspections are integral to a well-defined software development life-cycle infrastructure supported by management in each phase of development.
- Inspections are flexible in determining what to inspect.
- Computerized tools are available to assist management in planning inspections and estimating project savings before commitment.
- Computerized tools are available to guide the inspection teams.
- Management tools are available for monitoring inspection process conformance (not an individual's performance) and tracking resulting inspection benefits.
- Project personnel are provided with the proper training and follow-up support.◆

References

1. NIST. "The Economic Impacts of Inadequate Infrastructure for Software Testing." NIST Planning Report 02-3. May 2002.
2. Bennett, Ted L., and Paul W. Wennberg. "Eliminating Embedded Software Defects Prior to Integration Test." CROSSTALK Dec. 2005.
3. McConnell, Steve. "Best Influences on Software Engineering Over Past 50 Years." IEEE Software Jan./Feb. 2000.
4. Wiegers, Karl. "The More Things Change." Better Software Oct. 2006.
5. Jones, Capers. "Interview." Computer Aid Inc. July 2005.
6. Radice, Ron. High Quality Low Cost Software Inspections. Andover, MA: Paradoxicon Publishing, 2002.
7. Weller, Ed. "Calculating the Economics of Inspections." StickyMinds Jan. 2002.
8. Priven, L.D. "Managing the Programming Development Cycle." IBM Technical Report 21.463 Mar.1972.
9. Fagan, Michael E. "Design and Code Inspections and Process Control in the Development of Programs." IBM Technical Report 21.572. Dec. 1974.
10. Priven, L. and F. Tsui. "Implementation of Quality Control in Software Development." AFIPS Conference Proceedings, 1976 National Computer Conference 45 (1976):443-449.
11. McGraw, Gary. Making Essential Software Work. Cigital, Inc. Mar. 2003 <http://citigal.com/whitepapers>.
12. Software Engineering Standards Committee of the IEEE Computer Society. "IEEE Standard for Software Reviews, Section 6. Inspections." IEEE Std. 1028-1997, 1997.
13. Humphrey, Watts. "Three Dimensions of Process Maturity." CROSSTALK Feb. 1998.

About the Authors

Roger Stewart is co-founder of the Stewart-Priven Group. Previously, he spent 30 years with IBM's Federal Systems Division managing and developing systems for air traffic control, satellite command and control, onboard space shuttle, LAMPS helicopter and in commercial banking, telecommunication and networking systems. Stewart has a bachelor's degree in mathematics from Cortland University.

The Stewart-Priven Group
7962 Old Georgetown RD
STE B
Bethesda, MD 20814
Phone: (865) 458-6685
Fax: (865) 458-9139
E-mail: spgroup@charter.net

Lew Priven is co-founder of the Stewart-Priven Group. He is an experienced executive with systems and software management and technical background. Priven was vice-president of engineering and application development at General Electric Information Services and vice president of application development for IBM's Application Systems Division. He has a bachelor's degree in electrical engineering from Tufts University and a master's degree in management from Rensselaer Polytechnic Institute.

The Stewart-Priven Group
7962 Old Georgetown RD
STE B
Bethesda MD 20814
Phone: (865) 458-6685
Fax: (865) 458-9139
E-mail: spgroup@charter.net