How to Avoid Software Inspection Failure and Achieve Ongoing Quality, Cost, & Schedule Benefits

A Stewart-Priven Group White Paper

The objectives of this white paper are to:

- examine why software inspections are not used more widely,
- identify the issues contributing to their lack of use, 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 paper, an inspection is defined as a pre-emptive 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.

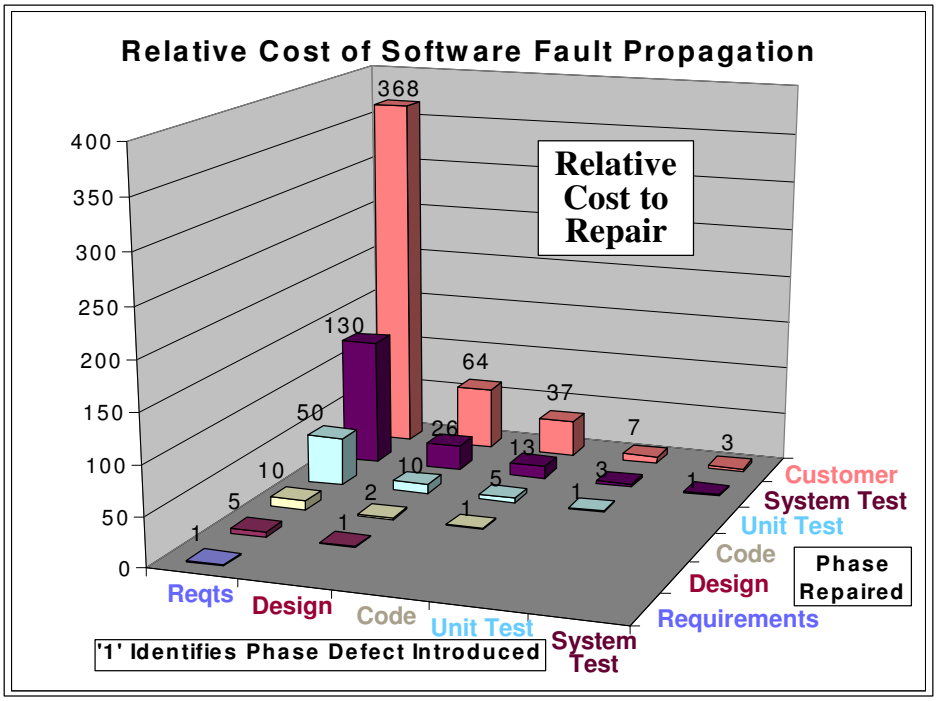
I. Background

According to the National Institute of Standards and Technology (NIST) study [1] - "*The Economic Impacts of Inadequate Infrastructure for Software Testing*" - **the problem of continued delivery of bug-ridden software** is costing the U.S. economy an estimated $59.5 billion each year. Some examples of high impact software failures are:

- during the first Gulf War, an American Patriot Missile battery in Dharan, Saudi Arabia, failed to track and intercept an incoming Iraqi Scud missile. The Scud struck an American Army barracks, killing 28 soldiers and injuring about 100 others.
- the misdirection of the NASA Mars climate orbiter
- the shutdown of the air-traffic control system in the LA airport
- the Northeast blackout
- the long delay in the completion of the Denver airport baggage-handling system

The study also found that: "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%) 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."

Figure 1 shows a typical relationship between the cost 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.



Defects inserted ('1' on the graph), and not discovered and fixed at their point of insertion, are much costlier to fix later in the Software Development Life Cycle. For example, in the graph:

Defects inserted during Requirement specification could be 5 times more costly to fix during Design, or 10 times more costly to fix during Coding. Defects inserted during Design could be 26 times most costly to fix during System Test.

Figure 1 – Cost of fixing a defect [2]

What is the evidence that inspections address the cost and quality issues described above 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.' *Steve McConnell - "IEEE Software Jan/Feb 2000, Best Influences on Software Engineering over past 50 years"*

- '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 - "IEEE Software Jan/Feb 2000, Best Influences on Software Engineering over past 50 years"*

- '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.' *Karl Wiegers - "The More Things Change, Better Software, Oct. 2006"*

- 'Formal inspections can raise the [defect] removal efficiency to over 95%. But part of the problem here is that not a lot of companies know how to use these things.' *Capers Jones, Chief Scientist, SPR – "Computer Aid Inc. July 2005"*

- 'The software community has used Inspections for almost twenty eight 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.' *Ron Radice - "High Quality Low Cost Software Inspections," 2002 Paradoxicon Publishing*

- '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.' *Ed Weller - "Jan. 2002, Calculating the Economics of Inspections"*

The evidence is clear – 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, Roger Stewart and Lew Priven each spent over 20 years developing projects that used inspections and for the past 8 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 - for example: critical inspection execution deviations are introduced and/or ineffective 'shortcuts' are employed. This results in inspection benefits being compromised, leads to limited use or discontinuation, and allows too many defects to escape into test and customer use.

II. Back to basics

In order to deal with the problem of inspections not being widely used (or 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" pseudo-quality at the end of the production line.

What naturally followed was the idea of applying quality control techniques to the software development life-cycle as if it were a production line. For example: sample the product periodically (detect defects), make adjustments as defects are found (fix defects and improve the development process), and predict the shipped product quality based on the results of the sampling

Application of the quality control techniques described above 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 [3], co-author of this paper.

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.

III. How have inspections evolved?

In 1972, Lew 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. The management system was based on a well defined development process – which satisfied the need for a "production line" as described above. 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 [3][4][5][6]. The IBM (Fagan) Inspection Process then evolved as a critical component of the end-to-end software development life cycle. Over the years, that integration with the software development life cycle has been lost as the **focus on the inspection process turned to execution details and inspections came to be viewed as a stand alone quality process**. However, the supporting infrastructure of a software development life cycle is critical to successfully implementing inspections.

IV. 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, be actively supportive of the inspection process. A life-cycle view is needed because the cost and schedule impact are primarily borne by the design and implementation components of the organization, while the resulting benefits of reduced cost, higher quality and improved schedule primarily accrue to testing, customer use and the overall project.

V. Theory good, but why aren't inspections embraced?

In addition to being viewed as a stand alone 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 are:

- inspections are time consuming;
  yes, they add up-front development time (e.g., requirements, design) but the payoff in improved project cost and quality can be quantified and the benefits shown. The problem is that time for inspections are added to the up-front

development phase schedules and the benefits, which accrue to the project, are not visible to these groups. Rather than being viewed as a problem, the additional up-front requirements, design or code time for inspections should be viewed as an investment in obtaining the quality, cost and schedule benefits for the project.

- 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 in an on-going project. An approach to prioritizing what work products to inspect needs to be intelligently applied by management.

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 paper will focus on dealing with those 'issues' which we will later refer to as '*Inspection Pitfalls*'.

VI. A realistic approach …

A realistic approach to inspections is to formulate a set of 'selection criteria' to guide the identification of those areas of the product most critical to success, or where problems are most likely to occur, and inspect those areas. 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% of defects are injected in requirements and 28% are injected in design. [7]

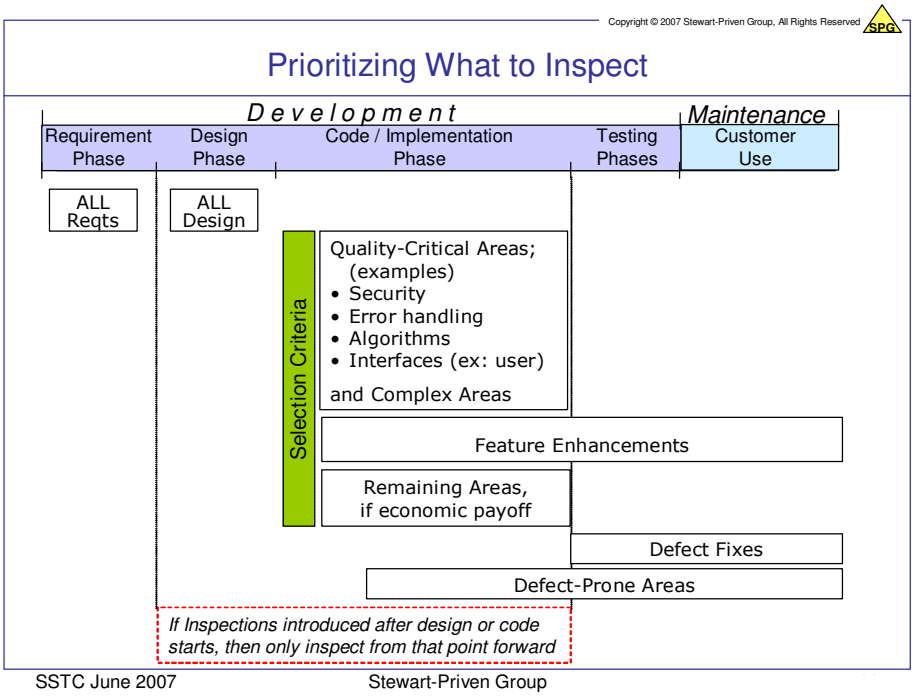


Figure 2 – Prioritizing what to inspect

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

VII. Development Infrastructure to support inspections

There is a lot of guidance on the structure of inspections such as IEEE standard 1028, and how to conduct an inspection, but little guidance on:

- how to select what to inspect (see Figure 2),
- how to develop an appropriate software development life cycle infrastructure that provides the necessary framework for successful implementation of inspections, or
- 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). See section VIII.

  For example, data collection is too often left to the discretion of the inspection teams and therefore data needed to evaluate the effectiveness, savings, etc. of inspections is not available to management.

Successful inspection implementation requires a software development life cycle 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 [8], "Although the CMM® provides a powerful improvement framework, its' focus is necessarily on 'what' organizations should do – not 'how' they should do it."

There are 4 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, then 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.

VIII. How the Stewart-Priven Group's (SPG) 'Inspection Methodology' reinvigorates 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. SPG's methodology 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 show the SPG Life Cycle Assessment Methodology.

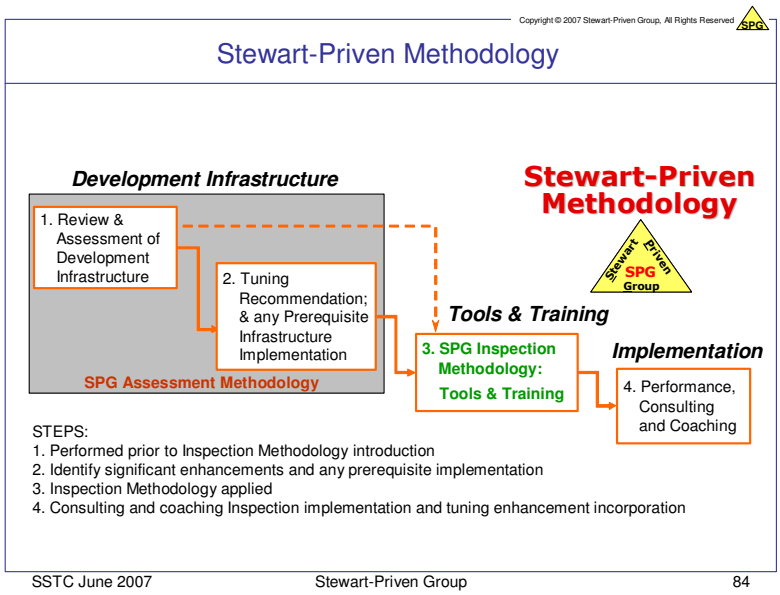


Figure 3 – SPG Assessment Methodology

Once the development infrastructure is in place, what else needs to be done? Based on our experience in training over 5,000 inspectors in many companies at over 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.

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

#	PITFALL	RISKS
1	Lack of Supportive SDLC Infrastructure	• Immature practices for planning, data collection, reporting, monitoring & tracking • Leads to Pitfalls #4, 6, 10
2	Poor Management Understanding of the Inspection Process, its' benefits, and their responsibilities.	• Leads to Pitfalls #4, 6, 8, 9
3	No Computerized Management Planning Tools	• Inadequate schedule time (Pitfall #4) • No savings appreciation, leads to no inspections or too few inspections
4	Too Little Schedule Time for Inspections	• Defects escape to more costly phases to fix • Inspections not correctly executed • Leads to Malicious Compliance
5	No Computerized	• Inconsistency, compromising shortcuts

	Inspector Tools	• Defects escape to more costly phases to fix
6	Inadequate Monitoring of Inspection Execution and Results	• Inspection process execution deteriorates • Defects escape to more costly phases to fix • Employees lose interest when savings summaries not periodically shared
7	No Post-Class Practitioner Refresher	• Process misunderstood, compromising shortcuts introduced, defects escape
8	No Inspection Facilitator / Project Champion	• Inspection process issues not addressed or coordinated • Inconsistent or incorrect inspection execution • Little useful feedback to management
9	Slow Inspection Implementation by Project Teams	• Ineffective start or no start occurs • Inspection training forgotten; Incorrect execution
10	No Inspection Process Capture	• Process misunderstood, inconsistent execution, defects escape • No repository for project lessons-learned

Table 1 – Risks associated with inspection pitfalls

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

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

SPG distinguishes between the development life-cycle infrastructure - within which inspections fit (see section VII), and the inspection infrastructure – which enables proper inspection execution. An enabling inspection infrastructure must address all ten pitfalls and would consist of:

1. Computerized tools for management use in planning inspections and predicting the overall project costs and savings from applying inspections: (see APPENDIX I for an inspection tool overview)
2. Computerized tools to aid inspectors in correctly and consistently performing inspections, gathering inspection related data, and identifying how future inspections can be improved

3. Monitoring and Analysis computerized tools for management post-inspection evaluation of individual inspections
4. Computerized management tools for analyzing inspection ROI and an aggregate calculator for assessing the actual project savings from multiple inspections
5. An inspection process that allows for options based on the project's target environment and on team knowledge.
6. Ability to customize your training material, incorporating your terminology, and is based on your needs
7. Rapid training of project personnel by a two-day comprehensive training course with significant focus on requirements and design documentation
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 your implementation successful both initially and long-term.
10. An inspection process capture tool to enable inspections to become an integral part of a company's SDLC infrastructure

Table 2 shows how SPG's Inspection Methodology solves the 10 inspection pitfalls

#	PITFALL	SOLUTION
1	Supportive SDLC Infrastructure	• Assessment Methodology – step 1 assess client SDLC – step 2 recommends any changes
2	Management Understanding	• Student Feedback from Insp. Class • Upper Management Overview • Management Performance Class
3	Management Planning & Savings Tools	• Planning-Counter Tool • Savings/Cost Estimator Tool
4	Scheduled Time for Inspections	• Inspection Planning-Counter Tool • Management Performance Class
5	Inspector Tools	• Preparation Tool • Inspection Meeting Tool • Analysis Tool, Effectiveness Tool
6	Monitoring & Assessment Tools	• ROI Calculators for Text and Code • Analysis Tool, Effectiveness Tool • Aggregate Results Tool
7	Student Refresher	• Seminar for Previous Students • Inspection Reference Card

		• Inspection Product Checklist Kit
8	Inspection Facilitator / Champion	• Upper Management Overview • Management Performance Class
9	Rapid Project Implementation	• 2-Day Inspector Training Class • Multiple Classes /Week (100+ students)
10	Company Inspection Process Capture	• Course Material Tailoring • Inspection Process Capture Tool

Table 2 – SPG's Inspection Methodology solves the pitfalls

IX. The Roadmap to Success

The pitfall solution roadmap shown in Figure 4 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 form an Inspection Methodology for achieving a lasting and successful inspection program.

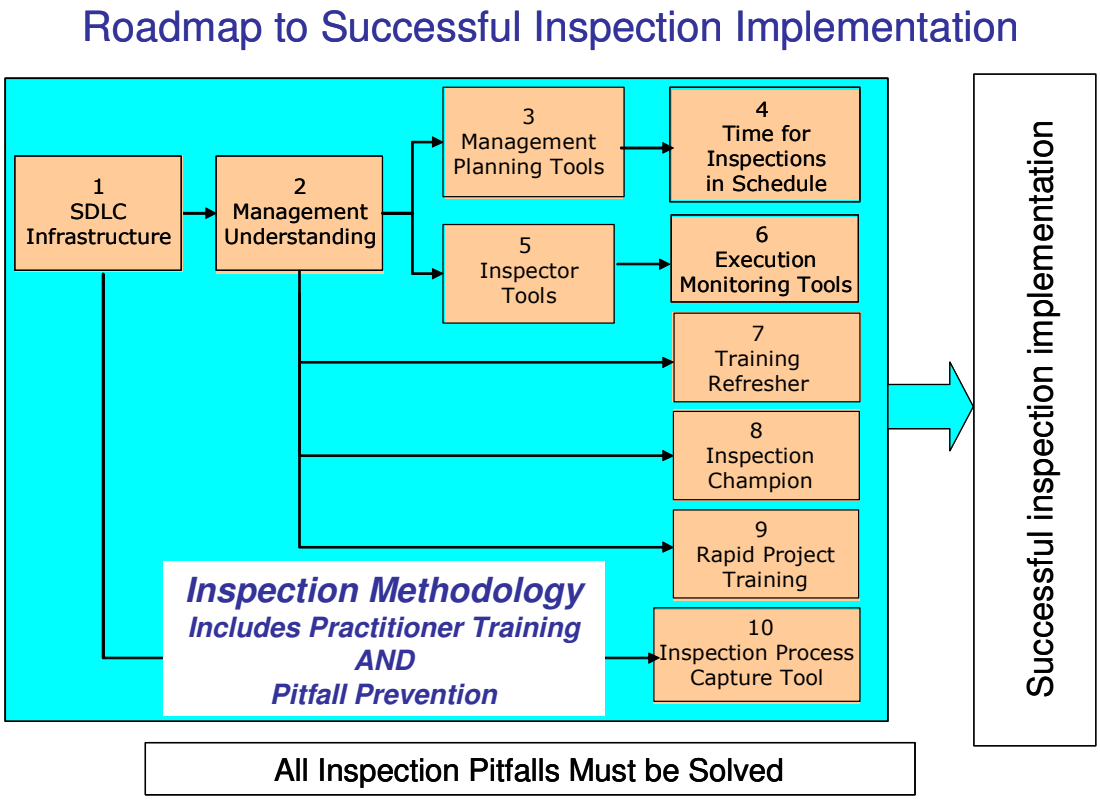


Figure 4 – Inspection Infrastructure

X. Summary

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

- they are integral to a well defined software development life cycle infrastructure supported by management
- they are flexible in determining what to inspect
- computerized tools are available to guide the inspection teams
- computerized tools are available to assist management in planning and evaluating inspections
- management monitors inspection execution and tracks results
- project personnel are provided with the proper training and follow-up support.

APPENDIX I

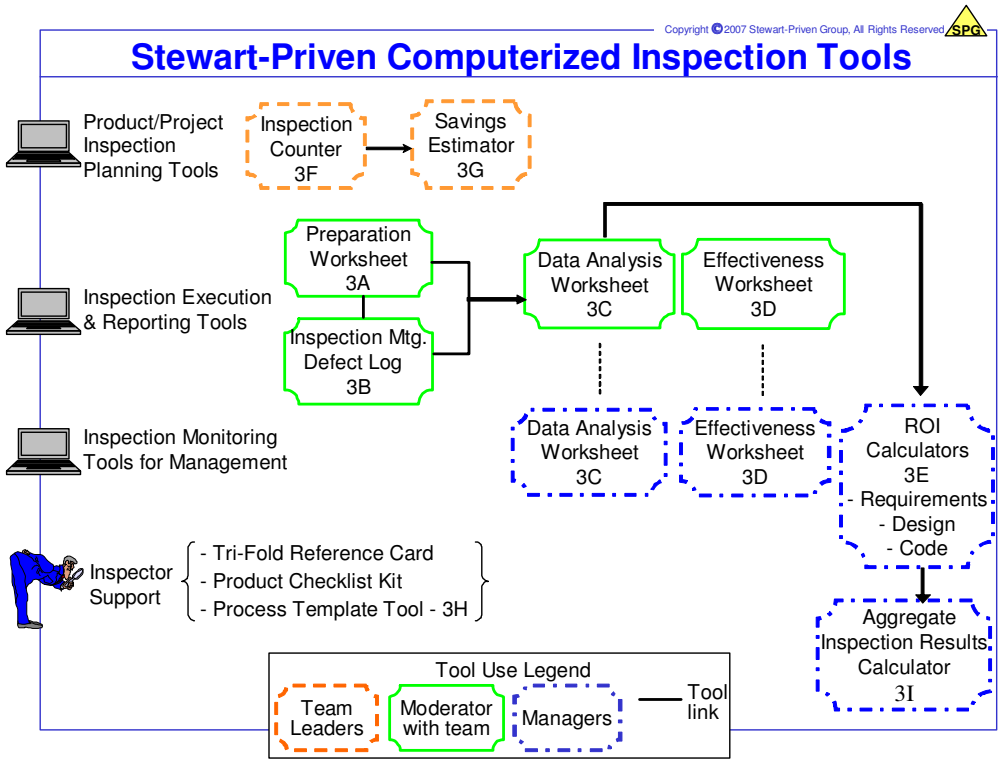


Figure 5 – SPG Computerized Inspection Tools

References

[1] NIST Planning Report 02-3 The Economic Impacts of Inadequate Infrastructure for Software Testing May 2002

[2] Bennett, Ted L. and Paul W. Wennberg. "Eliminating Embedded Software Defects Prior to Integration Test." Sidebar page 16 "Economics of Fault Finding." CROSSTALK Dec. 2005

[3] Radice, Ron. High Quality Low Cost Software Inspections. Andover, MA: Paradoxicon Publishing, 2002

[4] Priven, L.D. "Managing the Programming Development Cycle." IBM Technical Report 21.463 March 1972

[5] Fagan, Michael E. "Design and Code Inspections and Process Control in the Development of Programs." IBM Technical Report 21.572 December 1974

[6] Priven, L. and F. Tsui. "Implementation of Quality Control in Software Development." AFIPS Conference Proceedings, 1976 National Computer Conference 45, (1976):443-449

[7] McGraw, Gary. Making Essential Software Work. Cigital, Inc. March 2003 <http://citigal.com/whitepapers>

[8] Humphrey, Watts. "Three Dimensions of Process Maturity" CROSSTALK Feb. 1998

About the Authors

Roger Stewart is co-founder and Managing Director of the Stewart-Priven Group. He is an experienced Lead Systems Engineer and Program Manager in government and commercial system development – including Systems Engineering, Software Development, System Integration, System Testing, and Process Improvement.

Previously, Stewart taught the Fagan Defect-Free Process for Michael Fagan Associates (8 years) after spending 30 years with IBM's Federal Systems Division, (now part of Lockheed-Martin) managing and developing systems for Air Traffic Control, Satellite Command & Control, On-Board Space Shuttle, Light Airborne MultiPurpose System (LAMPS Helicopter); and in Commercial Banking, Telecommunication and Networking systems.
Roger has a BS in Mathematics from Cortland University.

Lew Priven is co-founder and Managing Director of the Stewart-Priven Group. He is an experienced executive with management and technical background in system and software development, software quality training, management development training and human resource management.

Previously, Priven managed the IBM team that developed the inspection process, taught the Fagan Defect-Free Process for Michael Fagan Associates (8 years), and was Vice-President of Engineering & Application Development at General Electric Information Services, Vice President of Application Development for IBM's Application Systems Division, Director of Operations & Development for the IBM Information Network, Vice President of Information Technology & Human Resources for Satellite Business Systems.
Lew has a BS in Electrical Engineering from Tufts University and an MS in Management from Rensselaer Polytechnic Institute.

The Stewart-Priven Group can be contacted at:

7962 Old Georgetown Road, Suite B, Bethesda MD 20814
Office: 865-458-6685
Fax: 865-458-9139 Fax
Web: www.stewart-priven.com
Email: spgroup@charter.net

