Software Inspections: 
**Cost Effective Quality Assurance**

**What are Software Inspections?**

Software Inspections are a formalized, structured form of *peer reviews*. They are an extremely cost-effective quality assurance technique that can be applied to any type of software project deliverable, such as Requirements documents, Design documents, Code, and other items such as test plans and user documents. Our course is fully compliant with the requirements of IEEE Std. 1028-2008 (Software Reviews and Audits) and fully consistent with the requirements of the Verification & Validation Process Area of CMMI.

For most software organizations, Software Inspections are **the most important single process improvement**. As illustrated by the attached article, if you don’t do them, you are throwing money away! The quotes below (among hundreds we could cite), demonstrate the effectiveness of this technique in increasing productivity, reducing cost, and enhancing quality. Even better, Inspections can be made effective quickly with proper training. In our course workshops you will inspect your actual deliverables and will find significant numbers of defects during the course!

“... **formal design and code inspections rank as the most effective methods of defect removal yet discovered ...** (defect removal) *can top 85%, about twice those of any form of testing.*”


“Rework to fix defects accounts for between 40 and 50% of total development costs. Formal reviews (i.e., inspections) typically find 80% of defects as they happen (walkthroughs typically find 60%). When effectively used, formal reviews can make an enormous difference to program cost, schedule and quality.”

- Program Manager’s Guide to Software Acquisition Best Practices

“Experience has shown that the cost of executing software tests to catch and correct problems is at least an order of magnitude greater than if such items are found and corrected earlier by using Inspection.”

- Tom Gilb & Dorothy Graham, *Software Inspection*

“An 8 work-year development, delivered in 5 increments over 9 months for SEMA Group found 3512 defects in Inspection, 90 in testing, and 35 (including enhancements) in the field. After two evolutionary deliveries, unit testing was dropped because it was no longer cost-effective.”

- Dennis Leigh, presented at British Computer Society
Getting Started:

PROCESS-FUSION offers a “non-denominational” software inspections training program. This program is a completely up-to-date synthesis that reflects our extensive experience blended with the best ideas of leading experts, including Radice, Wiegers, Fagan, Ebenau, Gilb, and others. The PROCESS-FUSION program is based on an exhaustive review and integration of published and unpublished materials.

Our comprehensive software inspections “start-up” package includes:

- Pre-course consultation and planning – designed to ensure you are well prepared to get maximum benefit from the training provided.
- A 2-day “hands-on” course that incorporates actual inspection of 2 work products by each team, including any combination of Requirements, Design, and/or Code. The course addresses both management and practitioner aspects of software inspections. Up to six teams of 4 inspectors each can be accommodated for a single fixed fee.
- Extensive reference materials are included in the course fee. These include copies of all course materials and a number of articles.
- An inspections data collection and analysis tool based on MS Access. Data collection, and subsequent analysis of that data, is a crucial element of making inspections effective – it provides the essential feedback loop that aids organizational learning and process improvement. So far as we are aware, no other training provider includes such a tool.
- No-cost 3-month and 6-month implementation follow up review and feedback to ensure effective deployment.

Intended Audience:

This course is intended primarily for software development practitioners – i.e., programmers, business analysts and development team leaders. The course is structured to include management in the first hour and last two hours of the course.

Software Inspections are a peer review process designed to significantly improve the quality of inspected products by finding defects at a very low cost compared to other quality assurance techniques such as testing. It is essential that all participants are knowledgeable about the items being inspected. Participants are expected to be competent in the language(s) used in the code to be inspected and/or in the notation and standards used in requirements and design documents to be inspected. This course is not intended for managers (except for the opening and closing sessions) or others who are not principally concerned with actual preparation of software requirements, design and development deliverables.

Inspections are conducted by teams of 4 persons (optimal size team, sometimes 3 are used); one of whom is usually the author (or maintainer) of the deliverable being inspected. Ideally, these teams are composed of people who normally work together. The course schedule is designed to accommodate a maximum of 6 teams of 4 persons each. Each team will inspect two deliverables during the course – any desired combination of requirements documents,
design document and segments of code. Each selected deliverable will be approximately 200-250 lines in length and will be selected prior to the course.

Structure of teams is typically rotated between assignments to give more individuals experience as inspection leaders. Details of team membership and assignments to be inspected are determined prior to start of the class.

Course Overview:

During the introductory portion, practitioners are given an overview of the course, expected benefits are described, and expectations are established for all parties.

During the closing portion of the course, management receives feedback from practitioners on what was accomplished, and their assessment of the value and effectiveness of the course. Next steps are discussed and any open questions or issues are addressed. This section includes a discussion of implementation issues and support required of management to ensure success.

The course is divided into three major parts – the first part (morning of day 1) instructs attendees on the details of the Inspection process, while the second part (days 1 and 2) consists of workshops designed to provide practical experience in use of the Inspection process, using real deliverables recently prepared by the participants. When the class is completed, significant practical experience will have been gained – this is NOT a “theory” course. The third section (afternoon of day 2) is a wrap-up that includes de-brief and summary for course participants and managers.

In one recent course, participants found **42 major defects** in a single requirements document, and **36 major defects** in 250 lines of code. Results vary, but significant numbers of defects are discovered in most deliverables inspected. The number of defects found nearly always surprises participants.

Software Inspections are applicable to every industry and every technology – they work extremely well regardless of the languages or applications being developed. Every deliverable – requirements, design, code, test plans, user manuals - can be cost-effectively improved by formal Inspections. Software Inspections are effective for both new development efforts and for maintenance of legacy systems. They work with any development methodology from traditional waterfall approaches to iterative, agile, and object-oriented techniques. Whether large or small, your organization will benefit from use of this technique.

Agenda:

- Day 1, morning – Inspection Process Tutorial
- Day 1, afternoon – Workshop #1 – participants inspect work products they developed
- Day 2, morning – Workshop #2 - inspect work products they developed
- Day 2, afternoon – review results, debrief with management, summary remarks

For additional information, including cost and scheduling, please contact Gary Gack at (904) 579-1894, or via email, ggack@Process-Fusion.net.
The Value of Software Inspections

a case study

by Gary Gack

The Client’s Problem

“We have contracted for a large custom-built software system which is critical to our business objectives. The project is now 18 months late and $ 20,000,000 over budget. Should we cancel or continue? If we continue, what should be done to ensure success?”

The Assignment

“Perform an in-depth assessment of the development processes being used and of the product itself. Recommend actions to be taken and provide a business case justification to support those recommendations.”

Outcome

Although there were many aspects of the recommendations we developed in performing this assignment, I will focus here on the quality assurance issues, and specifically on the impact of formal Software Inspections.

At the time this assignment was undertaken, the system in question had ostensibly been in system test for about 10 months. I say ostensibly because the programming was not entirely complete (hence certain elements could not be fully tested). Indeed, even certain requirements were not fully agreed, let alone designed. In spite of these shortcomings, significant testing had been done.

An elaborate automated testing process was in place, thousands of test cases existed, and comprehensive defect tracking records existed. Up to the time of the assessment over 3300 “major” (i.e., important enough to be fixed immediately) defects had been discovered and most had been repaired. Based on headcount assigned to the relevant activities, we estimated about 76,000 (!) hours, roughly 40 person years of effort had been charged to testing and defect repairs up to that time. This works out to about 23 hours per defect to find and fix.

We noted that the defect discovery rate had not materially declined over time, leading us to suspect that substantial numbers of defects remained to be discovered. As one of the steps to verify or reject that suspicion, we used a code analyzer to determine the cyclomatic complexity levels in the existing code. This analysis revealed that approximately 20% of the approximately 7000 C and C++ modules exceeded the generally accepted standard of cyclomatic complexity <= 10. As expected, these excessively complex modules accounted for 60% of the defects identified up to that time.

In order to develop an estimate of the number of defects remaining, we selected a small sample of complex functions and subjected them to careful unit testing in which coverage was assured by generating test cases corresponding to the cyclomatic complexity of each module. (See Structured Testing: A Testing Methodology Using the Cyclomatic Complexity Metric, A. Watson & T. McCabe, National Institute of Standards & Technology Publication 500-235 for details.) This testing process revealed an average of 2 defects / thousand lines of code (KLOC), and we estimated 2.4 defects /
kloc in the non-complex code. Overall, we estimated 3000 defects remained to be discovered. By way of comparison, the project team had based their plans and schedules on the assumption that 1000 defects remained. (Here we are referring to the number expected to be found by testing – not to those left for the customer to find.)

Hence, if the team’s estimate of 1000 remaining defects were correct, another 23,000 hours of testing would be needed to find them all. If our estimate were correct, 69,000 hours would be required. (Of course, in neither case would we realistically expect to find all defects through testing – many will get to the customer.) Given the lack of any meaningful level of unit testing, and no inspections or walkthroughs at all, the best we could reasonably hope for would be that 70% of the defects would be removed before delivery. Assuming the team’s estimate of 1000 remaining to be discovered was correct, how many defects would likely be delivered to the customer? If 4300 were discovered (3300+1000), and the removal rate was 70%, we conclude there were 6100 defects in the system (i.e., 4300/.7), which means 1800(!) would be delivered (6100-4300). If our estimate of 3000 remaining is correct …. Good grief!

To make a long story short, this analysis (and more) persuaded the customer and the vendor to accept our recommendation to introduce formal Software Inspections in order to achieve acceptable quality at an acceptable cost. The development team was given a training course that included “hands-on” inspection of a number of deliverables, including requirements, design and code. Very significant numbers of defects were discovered – sufficient to convince a very skeptical and resistant team that Inspections really work.

In this real life situation I had the opportunity to compare actual costs of finding and fixing defects by inspection in comparison to system testing. The actual results from that analysis, stated in terms of defects discovered and fixed per hour of effort, were as follows:

<table>
<thead>
<tr>
<th>Method Used</th>
<th>Defects Found &amp; Fixed / Hr.</th>
<th>Relative Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Design Inspections</td>
<td>.94</td>
<td>1</td>
</tr>
<tr>
<td>Code Inspections</td>
<td>.12</td>
<td>7.8</td>
</tr>
<tr>
<td>System Testing</td>
<td>.038</td>
<td>24.7</td>
</tr>
</tbody>
</table>

If we were Engineers, subject to professional regulation, it is safe to say that failure to use Inspections would constitute malpractice! Regardless of the actual numbers in your organization, one thing is VERY clear – Inspections are the most cost effective intervention you can make in most organizations. Without exception, every software development organization should be using Inspections, particularly as they can be made effective, with proper training, very quickly – usually within a few weeks.
What IF ??

While it is never possible to know for certain what would have happened if things had been done differently, perhaps it is instructive to speculate a little in this instance. What might have happened if a "best practices" approach had been followed from the outset?

Suppose that there really were 6100 defects inserted into this system during development, and that the defects found and fixed/hour rates given above are valid under the best practices scenario.

Suppose formal design Inspections had been done. Further suppose that 50% of the defects existed at that point in time (i.e., 3050 defects), and that Inspection would remove 60% of them. (60% is generally consistent with industry experience for team new to the process – up to 80% for more experienced teams.) Given these assumptions, design Inspections would find and fix 1,830 defects (3050*.6) at a cost of 1,947 hours (1830/.94).

Suppose formal code Inspections were also done, and that they also remove 60% of the remaining 4270 defects (6100-1830). Given these assumptions, code inspections would find and fix 2,562 defects (4270*.6) at a cost of 21,350 hours (2562/.12).

The above scenario means that we will enter system test with 1,708 defects yet to be discovered (i.e., 6100-1830-2562). It is likely that 1,196 would be found by system testing (i.e., 1708*.7). (70% is generally consistent with industry experience). Given these assumptions, system testing would find and fix 1,196 defects at a cost of 31,463 hours (1196/.038).

The following tables summarizes and compares this best practices scenario to the actual project:

<table>
<thead>
<tr>
<th></th>
<th>Actual Project</th>
<th>Best Practices</th>
</tr>
</thead>
<tbody>
<tr>
<td>Design Inspections</td>
<td>Not done</td>
<td>1,947 hours</td>
</tr>
<tr>
<td>Code Inspections</td>
<td>Not done</td>
<td>21,350 hours</td>
</tr>
<tr>
<td>System Testing</td>
<td>99,000 hrs (est.)</td>
<td>31,463 hours</td>
</tr>
<tr>
<td><strong>Total Hours</strong></td>
<td>99,000</td>
<td>54,760</td>
</tr>
<tr>
<td>Defects Delivered (est.)</td>
<td>1800</td>
<td>512</td>
</tr>
</tbody>
</table>

In this analysis the Best Practices approach would probably have improved total project productivity by at least 10% while delivering a dramatically better product that reduces delivered defects by 72%. The final schedule would have probably been reduced by about 6 months!

For more information about Inspections and related process improvements, see www.Process-Fusion.net