

Software inspection Process: Integration into a Software Development Cycle

By Puthyrak Kang, September 19, 2009

I Introduction

Software inspection is a powerful tool to improve the quality and productivity of a software process. It was invented by M. Fagan at IBM in 1976 [6]. Software inspection was successfully adopted in procedural programming paradigm [6] to detect and fix defects early in the process. However, in Objected-Oriented programming, benefits of software inspection have been skeptical and sometimes not cost-effective. Software inspection involves a variety of software artifacts such as requirement documents, design documents, test documents, and code. These inspections contribute to the quality and production of the software process.

The scope of this search focuses only the code inspection (hereafter used interchangeably by inspection), discussing its costs and benefits, and analyzing how it is adopted to a real software development process (hereafter used interchangeably with development process).

The paper centers on: summary of literature reviews, summary of discussion with an experienced practitioner, personal experiences, discussion and conclusion. The literature review provides academic insights into a software inspection's process, techniques, costs and benefits. The interview section discusses practical feedback of software inspection in a real project. The personal experience section shares the author's experience in a process to integrate a software inspection into a project. The discussion section centers on the pros and cons of the software inspection, and also focuses on how an inspection process can be adopted to software development process. The conclusion section summarizes of the paper.

2 Literature Review

2.1 Definition of Software Inspection

A formal software inspection is defined as a formal tool, process, or method to detect and correct errors in work products (design document, testing document, code, etc) in each phase of software development process. The inspection shall be formal, efficient and economical [6].

2.2 Objective of Software Inspection

The primary objective of the software inspection is to detect and correct defects early before they leak into subsequent phases. This improves the quality and productivity of a software development. Generally, a

software inspection is used as an exit criterion for coding [6]. Coding is not considered complete unless it passes a code inspection process. A software inspection may also act as a mean to ensure that appropriate parties are technically agree on the work, and the work meets pre-defined standards or conventions [6].

2.3 Inspection Process

Fagan [6] defines an inspection process a set of five activities: *overview*, *preparation*, *inspection meeting*, *rework* and *follow-up*. In the *overview*, an owner goes over a work product under inspection, providing inspectors understanding and background of the work product. In the *preparation*, each inspector reviews a work product on his own, and lists down all questions and defects discovered. At the beginning of the *inspection meeting*, moderator collects lists of defects and questions found in the preparation, and hands over then to a owner of the work product. During the inspection meeting, a reader reads the code, and moderator instructs inspectors to perform inspections. A recorder records all defects and questions. When the meeting is over, the owner collects all questions and defects for reworks. Each session of the inspection meeting is recommended for less than 2 hours. In the *rework*, the owner goes back and resolves all problems found by the inspection. After all problems are resolved, the moderator decides if a *follow-up* session is needed.

Ronald A. Radice [6] adds to Fagan's process the following activities: *planning & scheduling*; *analysis meeting*; *prevention meeting*; *inspection process monitoring*; and *data recording and reports*. The *inspection monitoring* and *data recording and reporting* are performed throughout the inspection process.

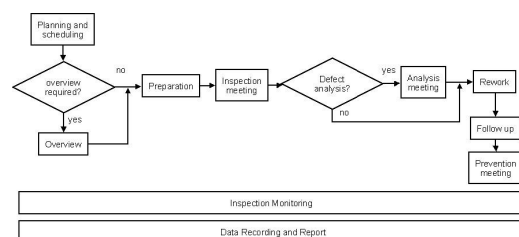


Figure I – Inspection Process; Adapted from Radice [6]

In the *Inspection Planning*, the Project Lead is responsible for determining what to be inspected, and estimating resources for inspections allocate budgets. In *Inspection Scheduling*, The Project Lead is responsible for requesting, selecting, or assigning Moderators when a work product approaches Inspection readiness. A notification

about inspections is sent and inspection meeting date and schedule are determined. The *Analysis Meeting* is intended to provide inspectors a time period to analyze defects found and their causes. The *Prevention Meeting* is optional and held periodically after a series of inspections have been completed. The Prevention Leader for the Prevention Meeting will record result of this meeting, and propose an action plan to the management. The *Inspection Monitoring* is help concurrently with other activities and after inspections. The Inspection Process Coordinator or SEPG is responsible for monitoring and suggesting improvement. The *Data Recording and Reporting* is performed during the *Overview, Inspection Meeting*, and optional *Analysis Meeting*. The Recorder records data about the defects and the conduct of the inspection, using templates, database, etc.

2.3.1 Procedure for each activity of the inspection process

Radice [6] suggests that inspectors follow a procedure for each activity in an inspection process to gain knowledge and perform the activity effectively. This procedure includes such sections as: *introduction, responsible individual, other roles, entry criteria, tasks to be performed, validation/verification, and exit criteria.*

The *introduction* section gives a brief explanation of the corresponding activity and its related procedure. The *responsible individual* section describes who is the primary person performing this activities. The *other roles* section describes other individuals or roles involved in this activity. The *entry criteria* section lists pre-conditions that need to be complete before performing this activity. The *tasks to be performed* section lists each of the key tasks to be perform within this inspection activity. The *validation/verification* section lists the relevant validation/verification tasks to be performed to assure control of the tasks to be performed. The *exit criteria* section lists all conditions to be satisfied before this activity is considered completely performed.

2.3.2 Techniques for Code Inspection

Dumsmore et al. [6] discusses three techniques used by inspectors to inspect code in Object Oriented (OO) paradigm. The three techniques include: *checklist-driven, use-case driven, and abstraction-driven.*

The *checklist-driven* technique has been used by software inspectors since the beginning of the inspection era in the late 1970s. This is considered as a static technique. It provides a series of specific questions on common sources of defects as for “where to look,” and “how to detect.” Its principle is to have inspectors to look into areas such as: inheritance, class constructor, data referencing, object messaging, selection, iteration, method behavior, and method overriding.

The *use-case driven* technique supports inspection of the dynamic execution of the system. It devises scenarios from use-cases and scrutinizes how an object under inspection responses to each scenario. It seeks to verify the

correctness and consistency when methods are invoked, decisions are made, and states are changes. Its aim is to force an inspector to consider the context in which the object is used.

The *Abstraction-Driven* technique focuses on creating abstract specification of classes for the system under inspection. It checks interdependencies in classes and methods (coupling) across the system and dependent libraries. The fundamental idea is to enforce an inspector to create an abstract specification from the code, which benefits current and future inspections. This technique is attempting to address an inspection problem – the resolution of frequent nonlocal references – caused by the concept of delocalization. The abstraction localizes the “reference delocalization.”

2.4 Benefits of Inspection

2.4.1 Code inspection improves code quality

Don O’Neill [6] quotes that a code inspection detects and removes 50 percent of the defects present early in development phase for an organization with adequate training in an inspection process. For organization with expert practices (IBM, AT&T, etc.), detection rate is up to 83 percent (for IBM) and 92 percent (for AT&T). This early detection and removal of defects improves the code quality before the code is delivered to next phase.

An inspection also pays enormous dividends in quality of software maintenance. Watts S. Humphrey [6] quotes that 55 percent of one-line maintenance changes were in error before the introduction of inspections and, immediately after, the error rate dropped to 2 percent.

Additionally, an inspection increases the system reliability. Humphrey quotes that the inspection in software maintenance reduces the number of production crashes by 77 percent.

Design quality is also improved by a code inspection. The inspection techniques improve for design quality by removing: redundant logic, unused variables, incorrect Boolean tests, comments that don’t match the code, and other un-handle exception. The inspection also provides opportunity for refactoring to get better quality of code and design.

2.4.2 Code inspection increases productivity

Humphrey [6] quotes that development productivity increases by 14 percent of a project in AT&T Bell Laboratory. This project of approximately 200 professionals instituted several changes, including inspections.

2.4.3 Code inspection reduces costs

Early detection and correction of defects reduce significant number of reworks and associated costs, and also shorten software lifecycle. Don O’Neill [6] asserts that a defect leaking to the test costs 9 times to detect and correct

than if it is found and fixed in the coding phase, and it costs 13 times more if it leaks to the fields.

2.4.4 Code inspection provides return on investment

Don O'Neill [6] also indicates that a software inspection provides a return on investment (ROI). Don O'Neill conducted a study in 1991 gathering data on defects and inspection practice in dozens of companies organized by software process maturity level, organization type, product type, programming and global region. The study has shown that companies who use inspections have a return on investment ranging from 4 to 5 dollars saved from every dollar spent.

2.4.5 Code inspection promote team relationship and education

Inspection process enable developers share their works and expertise across the team, and encourages developers to work together to deliver quality code. Discussions and defects analysis are conducted on the base of professional and team morale, not personal motivations. By integrating developers from other project, the inspection promotes relationship and close communication within a project and cross projects. Developers get to know each other and increase their relationship.

It is recommended to include one or two novices in an inspection. This enables novices learn from experienced developers, and also understand more about the project. The novices get familiar with the code quality standards and inspection process, and improve their productivity.

2.4.6 Code inspection promotes a culture of quality

Software inspection promotes a quality-oriented culture. According to Humphrey, code inspection encourages programmers to work more carefully either to avoid being embarrassed by sloppy mistakes or through the pride of exhibiting a quality work product. By enlisting others in identify their errors, developers actually end up doing better work themselves. The inspection makes it clear that quality code is what the organization expects.

2.5 Cost of Code Inspection

Although recognized as a technical best practice, inspections are slowly adopted into the software development processes. According to Don O'Neill [6], only 22 percent of organization assessed by the Software Engineering Institute practiced software inspections. This is due to the facts that software inspection is associated with costs. The following sections summarized some inspection costs observe red by practitioners.

2.5.1 Inspection direct costs

Bob McCann [6] defines a total direct cost of an inspection as a sum of costs for fixes, regression tests,

preparation, meeting, and reworks. He gives the following equation:

$$\begin{aligned} \text{Total Cost} = & (\text{inspection fixed cost}) \\ & + (\text{regression fixed cost}) \\ & + (\text{inspection preparation plus meeting cost}) \\ & + (\text{inspection rework cost}) \\ & + (\text{test rework cost plus regression cost}) \end{aligned}$$

Each cost item in the equation includes quantified metric of times and efforts used by each activities in the inspection process. Don O'Neill [6] estimates that it costs on average two person-months to establish a functional database for storing an inspection data.

2.5.2 Inspection indirect cost

One of an indirect cost of an inspection is training moderator and participants in the inspection process. According to Don O'Neill [6], to acquire the knowledge, skills and behaviors cost about 12 hour per participant. Managers are also trained in how to initiate inspections and interpret the data they produce. This takes about four hours.

The inspection is a time consuming and labor intensive process. An inspection involves at least six or six people which include talent and experience developers. Based on scope of work products and coverage, inspection may take multiple 2-hour sessions. To build up experiences in the process, it is time. Don O'Neill [6] asserts that it can take from 12 to 18 months to achieve expert practice which attains a defect detection rate of 60 to 90 percent.

2.5.3 Cost of an ineffective inspection

According to Humphrey [6], factors contributing to an ineffective inspection include inadequate training, lack of experiences, insufficient resources available, management inattention, and schedule pressure. Either the preparation was not enough, too many people were involved, the wrong people attended, or too much material was covered at one time.

Ineffective inspection may send misleading message to developers. Developers may consider an inspection as a mean for performance evaluation. Low-experienced developers are not comfortable to expose their works, and sensitive to an inspection report. An inspection may be turned to an offense-defense game. Inspectors are pointing out defects while the owner is defending his works. An inspection can become a pro forma exercise in which the inspection effort is expended but the desired results are not obtained [6].

3 Toward a formal inspection [personal experience]

My current project develops and supports a home-grown scheduling application. The project follows "iterative process" for the software development. The project consists of a team of 16 developers, which includes release

developers and maintenances. The team is very integrated. A developer can be either part of release development team or maintenance team depending on the release schedule. We don't have dedicated expert testers. We depend totally on unit test (called IT testing) and functional test (performed by liaisons) to ensure the quality of the software. The software features have grown from release to release. Each release has a development cycle of 3 months. Given the delivery constraint, the project's code is exposed to the possibility of high coupling, loose cohesion and structural corruption. This is a known drawback of iterative process. It happens when practitioners do not have proper and effective plans for refactoring.

The team agrees that this potentially affects the quality of the software and increase the maintenance costs. The team considers exploring code review techniques to address this concern. However, due to lack of experience we do not *how to do inspections*. To get it starting, the maintenance team of four developers includes code reviews in the bug fix process.

The maintenance team started with walk-through. Each developer presented his fixes to the team during a 2-hour review session, and resolved any errors detected by the review after the session. The developer re-presented his works in the follow-up session depending on the severity of the changes. This simple review process significantly improved the quality of the code and minimized potential side-effect of maintenance fixes. Code standard and style are enforced. Runtime exceptions (e.g. NullPointerException, ClassCastException) were detected and corrected. The team started with a simple checklist which included: checking for compliance to the code standard and style, detecting runtime exception, and analyzing exception handling. Defects and problems found during the code review were analyzed to determine its causes and effects. The review checklist was updated to prevent defects from reoccurring. As implication of the code review, developers spent more time and efforts during fixing to analyze a defect. Developers used flowcharts to present their analysis, and assess risks associated with their fixes. Developers also looked for opportunity for code refactoring.

After implementing for about a year, the team faced new challenge. Developers became familiar with coding convention and style. Their works met the team's standard requirements. Participants in the review had minor things to suggest. The review session turned to less interesting, besides sharing information about bug fixes. To booster the effectiveness and efficiency, the team added more features to the process adopted from a more formal software inspection. However, it was still a baby-step. A bug fix was required to be signed off by one or more developers depending on the size of the fix. Developers were assigned to be a sign-off for a fix one week before the review session. The sign developers were expected to perform an in-depth inspection of the fix, and present questions or suggestions to the owner before the review session. The main purpose was

to have someone look into other fixes more seriously. This was very beneficial to the quality of the fix, minimizing its risk of injecting new unknown defects.

Currently, the code review process of our team is somewhat formal. An inspection checklist was crated. Each developer follows the checklist to review peers' code. The review session is 2 hours every Wednesday and Thursday. Defects and problems found in the review session are analyzed to find the root cause and how to prevent them from reoccurring. The checklist is updated accordingly if necessary. The team tries to keep the check list simple, precise and intuitive, even though it has been modified and growing. We are moving forward to improve the review process, and extend the review coverage beyond maintenance team.

4 Discussion & what learned

4.1 Lesson learned

What I learn from the literature reviews and the interview is that a software inspection provides more benefits than costs to an organization. Benefits increase and costs decrease when the organization gains more and more experiences in practicing a software inspection process. Inspection costs are largely due to ineffectiveness in preparing, conducting and monitoring an inspection. This is caused by lack of experiences, management inattention, and management commitments. Other factors contributing to an ineffective inspection process include: organizational culture, maturity level, size of project, and resource availability.

However, I think any organization can adopt an inspection to its software process regardless size and maturity of projects. Inspection costs can be reduced by such factors as: training, a realistic scope, proper preparation, effective performance of inspection meeting, monitoring, and evaluation

An organization which is new to the inspection process can start with a basic process—code walk through—and then gradually moves toward more formal inspection process, as needed. The purpose is to get developers involved in the inspection process. When they are comfortable with sharing their code with peers, a project team can move to improve the quality of the code. To avoid frustration and resistances from developers to the inspection process, the inspection can focus on bug fix and maintenance works which do not involve a large amount of code. The organization needs to create an inspection process and coding standards to meet its own needs. The next section 5.2 proposes a process which can help a project to integrate an inspection process to a software development process.

4.2 Process Road Map toward a formal inspection process

A process for integrating an inspection is discussed as follows:

Stage 1: get start with "Hello World"

- Involve all project members to create standards, styles and coding conventions used for the project or organization
- Involve all member to create an inspection checklist (the simple and short), focusing on standards, styles and coding conventions.
- Start with a simple code review (walkthrough)
- Confine the scope to bug fix and maintenance works.
- Conduct an review session of two hours every week
- Maintenance team leads the code review
- Include at least 4 developers in the maintenance team
- As entry criteria, code fix is checked in to a control version system (e.g. CVS), and problem descriptions or requirements are written and available
- Inspection is not used as an exit criteria for development work

Stage 2: involve new developers

- Include one or two developers from other teams in the code review session
- Analyze defects or problems found during the review, and determine how to prevent them in future
- Update the inspection check list as needed
- Assign a developer to review bug fixes before the meeting. This person is expected to review the fixes thoroughly.

Stage 3: expand inspection scope

- Expend inspection to include larger code (with less than 2000 line of codes) which is not bug fix or maintenance works.
- Use the techniques (*checklist-driven*, *use-case driven*, and *abstraction-driven*) discussed in section 2.3.2 for code reading
- Not every code for non-bug fixes needs to be inspected
- Provide training to developers in the current inspection process, and in a formal inspection process.

Stage 4: formal inspection

- Adopt a formal inspection process in section 2.3.

My experience revealed that moving from one stage to another may take a year or longer. It requires management support and motivations. My current team is moving toward stage 2 after practicing a code review for bug fixes about more than a year. My previous small team in IBM stayed in stage 1. One main reason is cost effectiveness of the process.

5 Conclusion

Software inspection is a technical best practice and should be adopted by organizations to improve the quality and productivity of a software development process. The inspection also reduces number of reworks and associated costs. Additionally, it implicitly promotes team morale and a quality oriented culture in the organization. However, there are costs of the inspection. It is a time-consuming and labor-intensive process. To achieve an expert level of inspection practices, the organization needs to provide adequate training, acquire experienced developers, and allocate resources. Ineffective inspections will have significant impacts on individuals and team morale.

No all organizations have adopted a code inspection into their software development process. This is because organizations do not have enough skills and experiences in an inspection process, and may be not able to justify the inspection's cost effectiveness against business. However, I believe that an organization or a project can adopt an inspection process if it wants to. It does not need to go with a big bang approach. An organization should start with a low level inspection (walk through), and gradually moves toward a formal inspection process.

6 REFERENCES

- 1 Dumsmore, M. Roper, and M. Wood, "The Development and Evaluation of Three Diverse Techniques for Object-Oriented Code Inspection," *IEE Trans. Software Eng.*, vol. 29, no. 8, August 2003.
- 2 Fagan, M. E. "Design and code inspections to reduce errors in program development," *IBM System Journal*, val 15, no. 3, 1976.
- 3 Watts S. Humphrey, *Managing the Software Process*, 1989
- 4 Alexandrov, et al., "Impact of Software Review and Inspection," <http://hal.archives-ouvertes.fr/docs/00/01/34/58/PDF/democrite-00009850.pdf>
- 5 Ronald A. Radice, "High Quality Low cost Software Inspections,"
- 6 Don O'Neill, "Issues in Software Inspections," *IEE Xplore*, May 3, 2009 (<http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=00566420>)
- 7 Bob McCann, "When Is It Cost Effective to Use Formal Software Inspections?"

(<http://www.stsc.hill.af.mil/crosstalk/2004/03/0403mccann.pdf>)