Reviews and inspections

Michael Fagan
President, Michael Fagan Associates
Palo Alto, California, USA
Michael@mfagan.com

IBM T.J. Watson Research Lab: senior technical staff member

IBM Quality Institute: Co-founder

Corporate Achievement Award from IBM

University of Maryland: Visiting Professor

Major contribution: software inspection process

Current interests: improving the Fagan Defect-Free Process

Michael Fagan

A History of Software Inspections

Inspections are now thirty years old and they continue to improve software quality and maintainability, reduce time to delivery, and lower development costs!

The concept of finding defects as early as possible in the software development process to save time, effort and money seems intuitive in 2001. Many companies employ variations of the software inspections that I created as part of their development process in order to do just that. However, even thirty years after its creation, it is often not well understood and more often, poorly executed – yielding results that are positive, but well below their potential.

This paper will explore the history and creation of the software inspection process by Michael Fagan. Some readers will find a great deal of similarity between the development environment that led to the creation of this process and the one in which they are currently involved. The process itself has proven to be as effective and necessary today as when it was first created.

The Genesis of Inspections

My professional career began in hardware engineering and manufacturing. This shaped my thinking in terms of the cost of rework because when errors are cast in silicon, there is no recourse but to scrap the erroneous pieces, fix the design and rerun the lot. In short, defect rework was obvious, measurable and expensive. Every effort was made to find defects BEFORE production began.

In my area, at least, testing did not find sufficient defects before release to production, so I formed teams of engineers to intellectually examine designs AFTER exhaustive testing and before release to production. The results were startling: the engineers found many defects that testing had missed, significantly shortening the time needed to manufacture good product.

In 1971, on the advice of well-meaning friends, I made the switch from hardware development to the burgeoning world of software development. In this world, everything was new to me and posed quite a challenge. Development was chaotic, and, what was worse, no one seemed to have a way to get it under control.

There were no useful measurements that shed any light on what was going on or how to do better. There seemed to be an "everything is new under the sun" approach where each new project was undertaken as though no development had ever been done before. There was very little learning from project to project. I quickly realized that in order to be effective (and to survive), I had to really understand my new environment and then find a way to make order out of chaos.

It was obvious, even without measurement, that reworking defects was a very significant component of the software development effort. It was seen as being a completely natural accompaniment to creating design or writing code. Of course, it was and is. However, no one knew or focused attention on its relative size or how it may be examined in its own right as a consumer of vital software development resource.

It became clear to me that the magnitude of the rework effort was large enough that it needed to be better understood. I decided to measure it by applying a similar method that I had employed in hardware engineering. The idea was to intellectually examine software design and code in an organized manner and then measure the resultant effects on defect rates and defect rework during Integration and Testing. These human examinations reduced defect rework out of all proportion to the effort expended on them. This sparked the development of software inspections.

Introducing Inspections into the Development Process

Creating a methodology and persuading others to use it are two very different things. Indeed, during the early years of inspections, even with results that looked convincing to me, I encountered everything from derision to exasperation, but very little acceptance from my peers. Even when third parties, who had no stake in the methodology, reported how it had helped them, some people were reluctant to try using the inspection process. Resistance to changing working habits was (and continues to be) an impediment to spreading the use of inspections.

Conventional software development wisdom at that time was to follow a traditional life-cycle model in which there were a series of steps that were approached in a logical, sequential order (later called the Waterfall Process, by Winston Royce.) But this was not what I saw happening. Instead of one approach, there were several versions of the approach going on simultaneously. The product was developed in a series of cycles, similar to what we call "iterative development" or "spiral development" today.

Managing development of software (or anything) – making a plan, measuring accomplishment against the plan as the trackable elements were developed, and controlling the outcome to meet project objectives – as we tried to do in weekly "Build Meetings," required reconciling interpersonal communications among many people, who made perfect sense to themselves, but often not to one another.

I often left these meetings with more unanswered questions than answered ones: What were the actual requirements for the product we were trying to build, and were they clear and unambiguous to all the people who were responsible for satisfying them? Completion criteria for the Requirements Specification amounted to it being "signed off." However, being "signed off" did not necessarily signify that it was complete, correct, and non-ambiguous, because the signatories would often sign-off when the deadline arrived, but rarely with a clear understanding of the contents of the specification. Development would proceed based upon these requirements nevertheless.

Was there really an accurate plan of how the components of a system really fit together *before* system integration? Did all the players have the same understanding of the definition of each of the components of a system? What were the trackable units in the system build plan, and was their relative granularity known? Measuring and tracking the degree of completion of design and code entities that comprised a function in the development phases between "start of design" and "end of integration" was simply infeasible.

Absence of commonly agreed upon, measurable completion criteria of the activities of Requirements, Design, Code, and the other deliverables were a significant cause of mis-understandings, caused a good deal of rework, and did not provide any framework for managing software deve-

lopment. Without well-defined exit criteria for each stage of development, it was surprising to find how many different perspectives can be found within a development team about precisely when a particular type of work product (i.e., requirements or design) was "finished." For instance, are requirements finished when "signed off," when there is a testable product or feature defined, or when design, the next operation, accepts the document as valid input and begins work? This really was subject to a great deal of interpretation and was often more time (schedule) dependent than criteria dependent. Understandably, the confusion arising from this range of interpretations had to be resolved, and it was – in favor of meeting schedule dates, which are explicitly defined (even if it is unclear exactly what is to be delivered on a particular date!).

This brought up another question – how could the testing schedule be set when no one knew how much defect rework was involved during testing? Worse yet, immediately after a product was shipped, work would start on the next release. Unfortunately, developing the new release was frequently interrupted in order to fix the defects found by users in the previous release. Reworking defects in the previous release obviously subtracted from the effort committed to the new release and often caused it to ship late, or to ship on time with reduced functionality (i.e., fewer requirements satisfied), and / or to ship with more defects than it should. In addition, there was always a backlog of defects to be reworked and incorporated as a part of developing the new release. Supporting additional releases of legacy product compounded the problem.

[Note: The foregoing pattern is often found in young, start-up organizations. They get started with little or no formal process methodology and are successful until they must handle their own legacy software.]

However, it should be noted that if the percentage of total development effort that is devoted to fixing defects has an upward trend over the course of several releases, it is easy to extrapolate to a time when more effort will be spent on defect rework, with a diminishing percentage being available to develop new functionality. Since customers purchase products for their new functionality, continuation of this trend will lead to the point at which new functionality can only be added by employing more people – most of whom must rework defects in current and legacy products.

Answers to these and many other problems were not consistent and varied from case to case, without sufficient qualification of each case to enable extraction of any useful generalizations. The prevailing management climate was not sympathetic to dealing with these questions for the purpose of improving planning for future development. The thrust was to "ship this function now!" So, a lot of very good people worked very hard and wasted a lot of time dealing with miscommunications and crossed purposes.
We shipped products – time after time without much improvement to the development process between releases. We made software with (please forgive the expression) brute force – intellectual, of course – and heroic effort. Today, we would recognize this as an SEI CMM Level 1 organization.

In short, I found myself in a software development world that I found largely unmanageable, with too high a potential for failure. (Many others confided the same concerns to me, so I did not feel too lonely.) Therefore, to help our ability to manage in the face of the prevailing problems endemic in creating innovative software, and improve the quality of software delivered to customers, I decided that I would have to modify the development process so as to make it more manageable – at least in my area.

Since I was a manager and it was my job to deliver high quality software on-time and within budget, I took the liberty and the attendant risk to implement an inspection process in my area. I did not receive much support – in fact, I was ridiculed, counseled and otherwise told to stop the nonsense and get on with the job of managing projects the way everyone else was doing it.

However, I was given just enough rope to hang myself with (many people were hoping I would) and I was able to implement inspections on my project – part of an operating system. Through perseverance and determination, my team and I learned that by applying inspections, we could improve the outcome of the project. Further, the more effective inspections were at finding defects, the more likely the project was to meet its schedule and cost objectives. This led me to ignore my critics and continue experiments and improve the inspection method and its application within the development process to get even better results.

One of the first steps taken was the creation of measurable exit criteria for all the operations in the development process. Explicit, measurable, succinct exit criteria for the operations which created the work products that would be inspected (i.e., requirements, design, code, etc.) were the first building blocks needed to support the inspection process. This eliminated many mis-communications and bad handoffs between development phases.

Another issue that needed to be addressed was the pressure to reduce the number of defects that reached customers. This dove-tailed with the need I saw to minimize defect rework, which was such a large and uncontrolled component of development effort. Estimates of defect rework as a percentage of total development effort ranged from 30% to 80%. The two most obvious means to satisfy these needs were to reduce the number of defects injected during development, and to find and fix those that were injected as near to their point of origin in the process as possible. This led to formulation of the dual objectives of the inspection process:

- Find and fix all defects in the product, and
- Find and fix all defects in the development process that give rise to defects in the product (namely, remove the causes of defects in the product.)

My development team and I modeled, measured, and implemented changes in the initial inspection process that we used for creating real live software products. The procedure we followed was to make changes in the inspection process, including adjusting the number of inspectors and their

individual roles, measuring the results of executing the process with these changes installed, and then interviewing all the people involved in the inspection and in development. We analyzed the results of each change and made refinements and repeated the execution-refinement cycle up to the point when we experienced no further improvements. (This procedure was retained and periodically executed for monitoring and enabling continuous improvement of both the inspection and development processes.) After numerous cycles through this procedure, we reached a stage at which the inspection process was repeatable and efficiently found the highest number defects when it was executed by trained developers.

This initial work was done over a period of three and a half years. Many products and releases were involved, and several hundreds of developers and their managers participated – with varying degrees of willingness. The inspection methods employed did not delay any shipment, and customers noticed an improvement in the quality of products they received. Both the proponents and critics – and the latter abounded – were very helpful in making sure that we "turned over every stone." They helped us more than they know (or sometimes intended), so that by the time I wrote the paper, "Design and code inspections to reduce errors in program development," in the IBM System Journal, in 1976, I was confident that the process had been well wrung out in actual development and produced repeatable results. This paper provides a partial overview and useful highlights of the process, but was too short to fully describe the entire inspection process.

Over my next several years with IBM as a development manager of increasingly larger organizations, I continued measuring and repeating the execution – refinement cycle described above. After the process had been proven beyond a shadow of a doubt to be an effective means of reducing customer-reported defects, improving product quality as well as productivity, IBM asked me to promote this process within its other divisions. Again, I had to convince non-believers and change the work habits of development organizations. Eventually, I was even asked to share this process with a few large and key IBM customers.

For the creation of the software inspection process and the fact that it saved untold millions of dollars in development cost, IBM awarded me its largest corporate individual award at that time.

Implementation of Inspections in the Software Development World

The Inspection process we created proved to be an extremely effective means of reducing the number of defects users find in software, as well as increasing development productivity. This was achieved by using inspections to find defects and remove them from all the work products that are created in the course of developing software, namely, in requirements specifications, design, coding, test plans and test cases, and user documen-

tation. In addition, these inspections identified and caused to be fixed those defects in the development process that gave rise to defects in the products made using the process.

Over time, the list of noteworthy benefits has grown and now includes:

- Reduction in user reported defects;
- Increased customer satisfaction;
- Increased development productivity, which materializes in shipping more function in a given time or reduction in time to delivery;
- Improvement in meeting committed schedules;
- Rapid cross-training of developers and maintainers on new products;
- Continuous process improvement through removal of systemic defects (which are the cause of defects in the product);
- Authors rapidly learned to avoid creating defects through participating in inspections that find defects in their own work products and in the work products of others;
- Team building; and
- Inspections have, in some cases, eliminated the need to unit test code.

With this list of benefits, it is hard to understand why the use of the Fagan Inspection Process is not "de rigueur" in all development organizations. In fact, supporters claim that all "world-class" product development groups are using inspections. This is not what I see today. There are still many organizations that are not using inspections. Additionally, there are precious few organizations who have been trained to execute the process fully and in a way that will provide the complete set of benefits of the process.

One reason that is expressed by developers and managers who are launched on tight delivery schedules for not employing the inspection process, is their dominant fear that inspections will take too long, lengthen the schedule and delay shipment (this fear has stalked inspections throughout their history, and it continues). They note that inspections use 20 - 30% of the effort during the first half of product development and, they fear, this will add to the development cycle. Of course, what they overlook is the net reduction in development effort that results from inspections finding and fixing defects early in the life-cycle at a very much lower cost (in terms of time and money) than the effort that is expended on fixing defects that are found during testing or in the field. This reduction in development effort often leads to shortening the time to delivery. See Figure 1. Although experience has shown this concern to be a myth, fear of delaying shipment often causes managers to resist adopting inspections.

Additionally, mis-labelling of process operations continues to this day in the software industry. If we insist on truth-in-advertising, "Testing" would be called "Defect Rework" because in most cases much less than 50% of the effort expended during "Testing" operations is actually used to verify that the product satisfies its requirements, while more than 50% of effort is consumed doing defect rework. Exercising the product under test with one pass of the test cases is all that should be needed to verify that the product meets its requirements. This simple act of renaming would

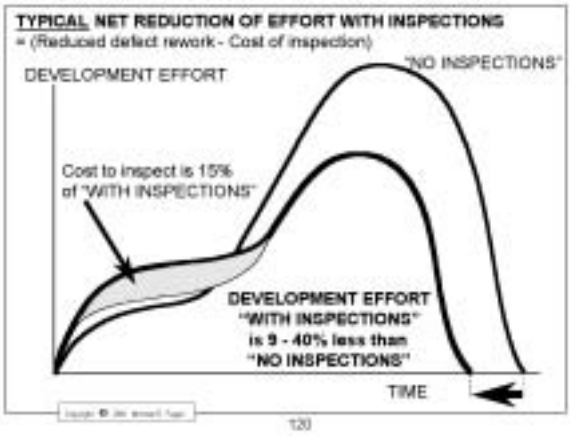

Fig. 1.

focus the attention of management [and financial auditors] on what is really going on, leaving testing to verify that the product satisfies its requirements.

Code inspections are more often discussed and receive a lot more attention than the inspection of requirement or design specifications, although the latter can produce even larger benefits. This could be because code is easier to inspect: since requirements are the foundation on which products are built, but are often improperly specified, or reflect some other process deficiency, making them more difficult to inspect. Similarly, discussion of inspections of hardware and software systems design, logic design, test plans and cases, and user documentation tend to be under reported, and / or under utilized.

Since their inception, inspections have had the dual objectives of finding defects in the product and in the process used to create the product. Finding defects in the product is so immediate and satisfying that most often the added step of identifying defects in, and removing them from, the development process is overlooked. Skipping this second step is very short-sighted as it does not enable the reduction of defect injection in all future development, an essential element in "putting oneself out of the defect generation business."

Inspections are a team sport. There are two aspects to this: First, the organization of effort to accomplish the inspection meetings and, secondly, the conduct of the inspection meetings themselves. The team members, each playing their appointed role in the inspection process, include Managers (all levels of line managers), Project Managers, Requirements Specifiers, Hardware and Software Designers, Coders, Integrators, Testers, Quality Assurance, Writers of User documents, Users of the product, and Marketing People. Like any team endeavor, successful inspections depend upon the individual capability of each team member and how well they work in inspection teams. Successful execution is an obvious, but frequently overlooked, determinant in effective inspections – see Effectiveness Factors, below.

There is often confusion of the relative roles of reviews, walkthroughs, and inspections. The fact is that both reviews and walkthroughs (taking these two activities as being essentially of the same type) and inspections are both important in the development of work products. Reviews / walkthroughs are conducted during development to demonstrate and refine approaches, solicit and compare viewpoints, develop improvements, and identify problems and create solutions to them. Inspections, on the other hand, are conducted when a work product is asserted to be complete. The objective of inspection is to find defects* only, not to undertake the activities of reviews / walkthroughs. (* - A defect is defined as an instance in which the work product fails to satisfy the exit criteria of the operation in which it was created, including operational defects that could be reported by customers.)

The reason for maintaining this exclusive objective for inspections is that experimentation has shown that when review / walkthrough activities are included in the inspection or, indeed, even one more objective is included, there is significant degradation in the defect detection effectiveness of inspections. Occasionally, reviews / walkthroughs are spoken of as if they are competing activities, and at other times as if one has evolved into the other. In practice, combining their activities creates an amalgam of disparate objectives and appears to lower their individual and combined effectiveness. The fact is that reviews / walkthroughs and inspections come into play during different times in the development of a work product, and each adds unique value. When they are both used separately, each one contributes according to its unique objectives in the development of software that is well thought out and contains very few defects.

The Inspection Process Today

After I left IBM, the incubator of the software inspection process, I have continued to study and make additional enhancements to the process through my company, Michael Fagan Associates.

The early inspection process included checklists of conditions to look for during preparation and the inspection meeting. Many of these checklists included items that could be found using tools and did not require the human brain. As the inspection process evolved, we eliminated checklists and learned to concentrate human intelligence on finding those defects that testing with computers could only detect with a lot of effort, and to let computers find mechanical type defects. For example, compilers flag syntax errors and LINT-type compiler tools are good at finding procedure and standard violations. Using these tools relieves human inspectors from mechanical checklist tasks, and allows them to be much more effective by concentrating their attention on finding operational type defects – those that would otherwise be found in test or by the customer.

Other changes have included creating a set of “Effectiveness Factors” to quantify the relative quality of the inspections themselves. We know from history and analysis of compiled data which things have the largest

positive and negative impact on the defect-finding effectiveness of an inspection. These factors can be used to manage the inspection process to produce its maximum set of benefits.

These and other recent enhancements have not been discussed in any other papers to date. They have been disseminated only to our clients through a course which provides training on the full inspection process.

Unfortunately, the terms "inspection," and even "Fagan Inspection," have become almost generalized to refer to things that do not produce the consistent and repeatable results that I consider necessary to good process management. Since the terms have come to mean something other than what I intended and implemented, I now speak of the "Fagan Defect-Free Process" which includes Fagan Inspections, as well as the other components needed to make this process successful and enduring.

The Fagan Defect-Free Process includes all of the necessary and interwoven components that are required to make software inspections successful. The three main ones include:

- Formal Process Definition, i.e., ensuring each member of the team is conversant in the objectives, function and entry and exit criteria of each process phase;
- Inspection Process – the seven-step process used to find defects;
- Continuous Process Improvement – removing systemic defects from the development process.

The name "Fagan Inspection Process" has been misused to denote just the seven-step inspection process originally created by me. It has also been applied to the variations created and promoted by others without a real mastery of what makes the process work successfully. Unfortunately, many organizations have had comparatively modest to poor results without understanding that their implementation of the process was incomplete. Thus, while inspections have been very successful and have proven their value over many years and many projects, there are those who would rather not include inspections in their development processes because of mediocre results, due to a mediocre implementation.

As may be expected, cases of partial or improper implementation of the inspection process, which are incorrect executions of the process, often produced inferior results and confused the issue. Thus, although it should be obvious, experience has shown that it must be stated that to get the best results from the inspection process, it is necessary to execute this process precisely and consistently. Similarly, experience has shown that all variations of inspection processes do not produce similar results.

Summary

The "Fagan Inspection Process" is as relevant and important today as it was thirty years ago, perhaps even more so. With systems growing ever more complex and a shortage of resource with which to develop them, its benefits of are even more attractive. Applied and executed as intended, it produces significant improvements to the software development process, including schedule and cost reduction, productivity improvements, and fewer customer-reported defects.

Nothing will be more satisfying in the future than to find the "Fagan Defect-Free Process" including Fagan Inspections, in increasingly wide-spread use not only in the software world, but also in other areas in which more diverse products and services are created. I have, and will continue to work towards that goal.