IEEE Software Article Inspection: : Written by request of Anne Lear

Software Inspections are not for quality,

but for engineering economics

By TomGilb

Abstract (150 words or less=141 words)

Software Inspections were developed at IBM and published in 1974-76 [1] more than a quarter century has passed and it is time to reassess them. Most literature on the subject is rooted in the initial IBM practices. Most professionals have an incomplete understanding of the traditional inspection and why it succeeded at IBM and elsewhere. But time and experience have led to many new practices which make Inspection more economic and useful than originally envisaged. The bottom line is that I believe that it is more relevant to view Inspection as a way to control the economic aspects of software engineering, rather than a way to get 'quality' by early defect removal. Quality needs to be multidimensional, specified in quantified requirements and architected, engineered and designed into software products. Inspection needs to be used to monitor the entire software engineering process.

Keywords

Software Inspection, quality, design, economics

Introduction.

A warning to the reader. I am not going to attempt to cover all details and explain all concepts thoroughly. The curious reader is referred to the References for such detail. I am going to shoot straight and make main points in the hope that the reader will realize that they need to more seriously study Inspection, and Quality, and not take it as superficially as they may have done up to now.

What position does this essay take?

Inspection

Correct basic use

There are some basics which frequently are not learned or practiced.

OPTIMUM CHECKING RATES
Inspection checking activity must be conducted at the optimum 'coverage rate'. This can be found for any site and document type by simple experiments or extensive operational data. But it will be in the range of one Logical Page (300 non-commentary words of the product document being intensively checked) plus or minus 0.9 Logical pages/per hour per checker. This amount of time is necessary to permit the cross-checking of specification Rules, Checklists, Source Documents and Kin Documents (example Test Cases versus Code) with the process 'Product' document. Failure to determine and use the Optimum Checking Rate simply leads to a lower density of defects *found* per page (the rest of them 'hide'). This leads to false assumptions about product quality. This can lead to release of *Major Defect ridden* documents to the next software engineering stages. Lack of time to do this is an invalid excuse. You can use sampling to reduce the total time needed. And failure to get this accurate measurement of the document quality, and the implied process quality, will cost more than any such sample will cost.

EXIT CONTROL
Inspection should not be 'cleaning up' bad craftsmanship. It should be measuring that the engineering process is sound and that the 'product' document at hand is consequently sound. Management needs to establish a sound set of Process Exit Conditions before any document is allowed to be used by others in the organization. The foremost of these is a computation of *'probably remaining Major Defects'*, where the level remaining is economically acceptable. This computation is based on Majors-found density, Majors corrected, and average effectiveness (% of available Majors normally found by this process) of the process.

For example if 6 Majors are found per page, and we assume 80% defect finding effectiveness and 82% probability of correct correction: then there are 7.5 Majors/page of which 1.5 were not found. If we then try to fix all that we found, 5 of the 6 will be correctly fixed. So 2.5 Majors would remain on pages where we had identified Majors, and tried to correct them. If the Exit condition was a reasonable professional level of maximum 0.25/Page Majors remaining, then we are far from economic exit.
If the downstream consequence of Major defects is about 9 hours of software engineering work each [3. Page 315] then the consequences of premature release are calculable and normally far greater than the cost of dealing with the problem *before* we exit (costs are about 10% or less of the cost of moving on prematurely).

Advanced inspection use

Inspections can be conducted in many ways which were not described in the initial 1970's literature [1] even though the authors and company involved themselves have sometimes gone on to enlarge the practice in some of these directions.

Here are some examples:

SAMPLING: measurement not 'cleanup'.
Most industrial quality control relies on 'sampling'. IBM decided they wanted to Inspect the entire document, not sample. But you are not IBM and this is not 1970's, so this decision can be reconsidered in the light of your needs and economics.

Sampling 1 to 4 representative Logical Pages is usually more than enough to convince any project about the level of Major Defects present in a document of any length. If there is any doubt or discussion about the results, a further representative or random sample will always be sufficient to convince any skeptic about the average level of Major Defects present. You could follow strict academic rules of proper sampling regarding size and representative sample if needed, but this is not science, it is industrial engineering, and such precision is not usually demanded. It is important that all players agree to act on the results.

The cost of sampling is, by definition, a small percentage of the cost of '100% sampling'. It is typically 4 to 8 engineering hours total, irrespective of Product Document size.

If the sample shows that the Probably Remaining Major Defects/Logical Page are below the 'economically acceptable formal Exit threshold' (usually 3.0 for beginners, improving to 0.3 Majors/Page after a few years of experience at a site) then the entire Product Document is Exited (assuming it meets all other Exit Conditions too).

Of course you cannot hope to 'clean up' Major Defects in the entire document, based on the sample. Inspection is not in a clean up mode when we sample. We are measuring the Product document quality versus the defined specification Rules, especially for probable Major (costly downstream) defects. We are trying to determine if it would pay off to fight the defects at this stage (no Exit yet) or possibly fight the few remaining ones later (in future Inspections, tests and in the field).

If the sample shows a large number of defects (10 to 70 Majors per Logical Page is not uncommon before the process is brought under control, and before people follow their Rules (CMM Level 2), then these defects are usually a fair clue as to systemic injection of defects, and the information can be used by the Product document Author to systematically 'clean up' the entire document, before re-submitting to Inspection.

Something people do not generally know is that the most mature inspection processes (example at IBM, [3, 9]) do not have a defect-finding effectiveness greater than 60% for source code and 80-88% for pseudo code and interface specification. Consequently inspection can *never* operate as a 100% effective cleanup mechanism. No more than testing which has a similar, but lower effectiveness. Cumulative Inspections of 100% of documents can detect up to 95% of all existing defects [3, Sema, as reported at Eurostar Conference Proceedings, London October 1993, Denise Leigh], but this is usually not the smartest and cheapest way to get there, as we shall see.

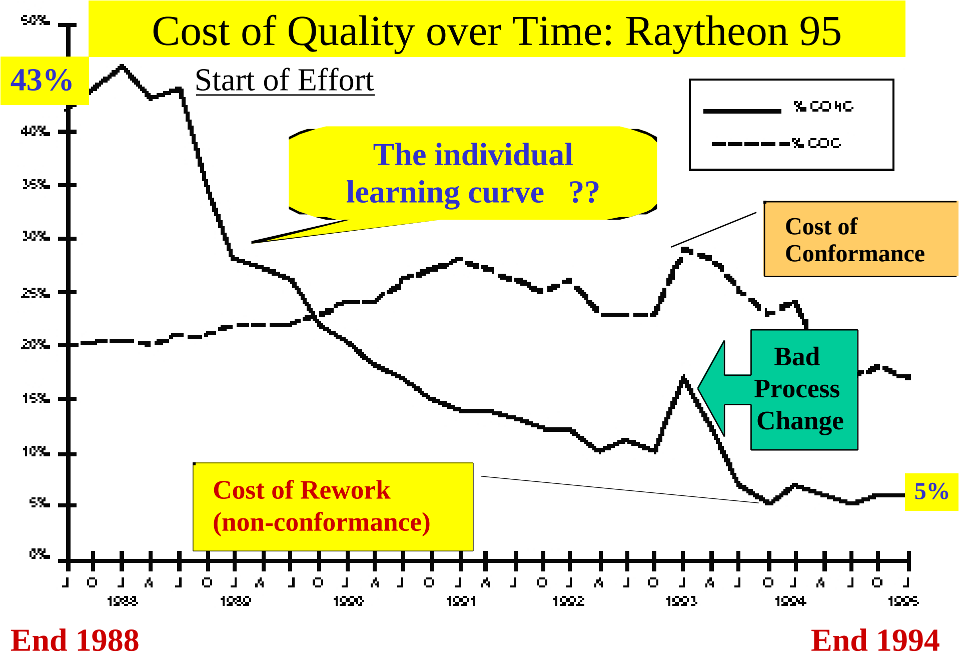
It is worth mentioning that for large documents (I have clients with 80,000 pages of requirements in a single project) *early sampling* long before the document is 'complete' is a basic defense against finding out the bad news too late!

Using Inspection to Reduce Defect Injection.

Inspection does have the ability to do two things which dramatically reduce the *injection* of Major Defects. It can systematically *teach* individual software engineers to follow the specification Rules. This in practice, within a few uses of Inspection reduces the individual defect injection by about two orders of magnitude (personal experience Douglas Aircraft 1989, and Ericsson 1997). This is I guess (Dion does not deny) probably responsible for the Raytheon drop from 43% (of total development costs) to about 27% rework costs within one year [8].

Then Inspection gives, combined with the Defect Prevention Process (DPP, CMM Level 5) the ability to *further* reduce defect injection, by improving the software engineering processes. This is responsible for the further drop in rework costs from 27% to 5% and less in the 1989 to 1994 time frame for Raytheon [8].

- **Source** Raytheon Report 1995, comments by Gilb
  - http://www.sei.cmu.edu/products/publications/95.reports/95.tr.017.html


Cost of Quality over Time: Raytheon 95

43%

Start of Effort

The individual learning curve ??

Cost of Conformance

Bad Process Change

Cost of Rework (non-conformance)

5%

End 1988

End 1994

Figure 8: Cost of Quality Versus Time

UPSTREAM QUALITY CONTROL

In spite of the fact that even the initial publications at IBM in 1974-5 by Fagan and Radice [1, 4] emphasized not only code inspections but 3 levels of design specification; and in spite of the fact that by the end of the 1970's IBM had documented that they were using Inspection for 12 different types of documentation, including user publications and test planning, I find it consistently gets perceived as 'Code Inspection'. In spite of the fact that TRW (Boehm, 62% Design error injection) and Bellcore (44% design error injection) [10] document that about half of all code bugs are not coding errors but are caused by bad info to programmers from the requirements-and-design documentation coders are given. Further, in both cases, the use of inspection immediately reduced this problem measurably. For example within 2 releases at Bellcore the percentage bugs due to bad design specs went from 44% to 30% [10].

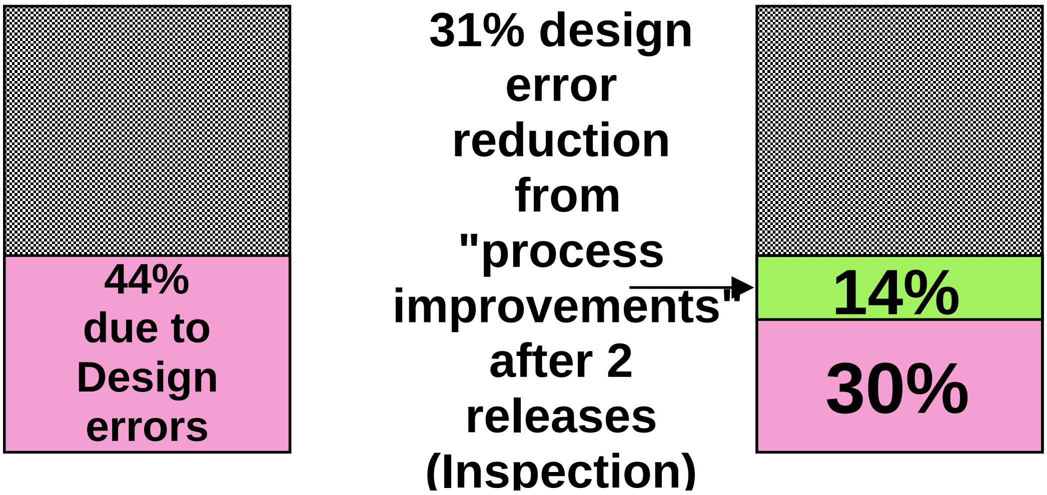


Figure illustrating the Bellcore experience. Initially 44% of all code bugs were due to bad specs given to coders. The use of inspection as a process improvement resulted in a 31% reduction in the % of defects due to bad specs. The lesson: fight code bugs upstream.

Inspection should be understood by managers to be something that should be used at management level on the critical 'upstream' paperwork of any project. This includes Sales proposals, Contracts (Philips UK), Contract Modifications (Nera ABB), Marketing Plans (HP Apollo), Systems level Requirements (Ericsson), System Architecture and all other project related documents. It has been used for these purposes [3, Reeve's Case Study] in many companies (some examples from my experience given above). In fact it can, is and should be used for any large system development where software is but one component.

As has been shown by Harlan Mills' Cleanroom method and by (recent re-calculations) Michael Fagan (the original champion of Inspections at IBM), if you are doing Inspection properly upstream, there will be so few Major Defects in the Code that it might not be cost-effective to use Inspection on source code for 'cleanup' purposes.

The apparent tremendous advantage of Code Inspections is often an illusion, created by allowing all Major Defect consequences to filter down to the Code Level, when they should have been prevented or detected at earlier stages. You might catch the defects, but your cost of correction will be unnecessarily higher than if you had dealt with them upstream.

DEFECT PREVENTION PROCESS (DPP)

IBM initially intended Inspection to exploit the principles of Statistical Process Control as taught by W. Edwards Deming and Joseph J. Juran. To learn from experience and to correct the process. This did not succeed; for subtle reasons (centralized high level control and analysis of statistics) which I did not appreciate at the time. In 1983 Robert Mays and C. L. Jones of IBM Research Triangle Park, North Carolina developed a practical way to learn from process errors, and to successfully systematically improve the software process, so as to reduce the systemic injection of Major Defects [3, Chapter 17 by Mays]. Robert Mays worked with Ron Radice [4] who developed the IBM version of CMM under Watts Humphrey. The DPP is the main model for CMM Level 5.

Statistically, DPP is generally capable of eliminating 50% of all normally injected defects in the first year of practice, about 70% within 2-3 years and about 95% after several years of maturity [8, and Mays , June 12-15 1995, his lecture 13th Wash DC, International Conference on Testing Computer Software IBM "Defect Prevention Process and Test"].

Because of the historical process development at IBM, Conventional Inspection and DPP became separate-but-linked processes at IBM, and consequently in the CMM Model (Level 3 and level 5). I believe it is time we re-married them, as initially intended. I have tried to do this in my version of Inspection [3, Chapter 7].

In addition to the systemic removal of Defect injection using DPP, it is well worth mentioning the 'Individual Training Effect' of inspections, mentioned earlier. The Individual Training Effect can reduce the level of Major Defects injected by an individual by two orders of magnitude, by simply giving individuals concrete feedback during Inspections as to what the Rules for avoiding defects are, for a particular documentation type.

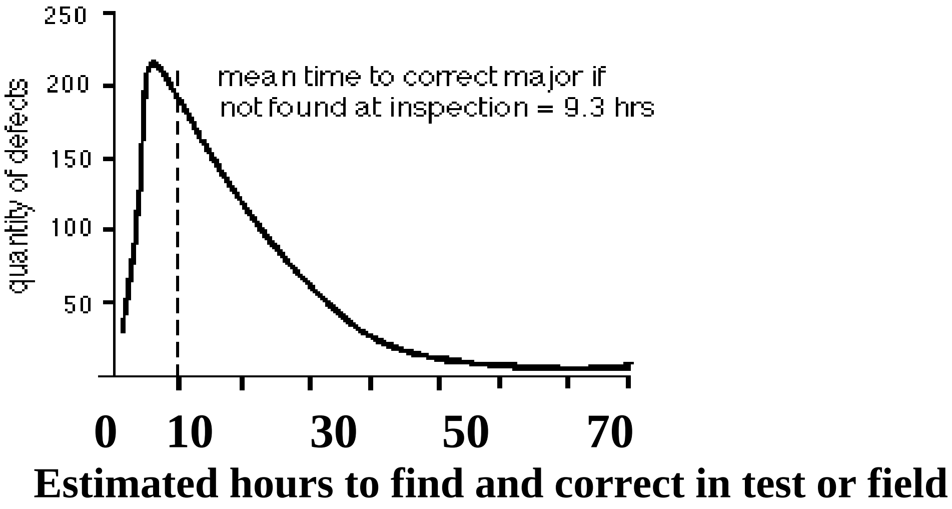
Both of these come under the heading of 'defect injection reduction'. They are by far the most important aspect of modern inspections. Simple defect removal, for example at the code inspection level is doomed to systematic failure.

The best large-scale ***Source Code*** Inspection Effectiveness (Defect detection % of all defects present) level I know recorded is 60% [3] at IBM Minnesota. It took years of good process improvement work to get there, and most good beginners are at a level nearer 30% defect-finding effectiveness. What this means is that 70% to (best case) 30% of all 'Bugs' will escape from Code Inspections, presumably to testing***. The order-of-magnitude of bugs released to testing will not be significantly different from those that entered the Code Inspection process!*** Cleanup mode doesn't work!

What you ***should*** be doing, if you ***must*** do Code Inspections, is to use it with ***sampling***, to determine the ***probably remaining Major Defects*** you would be releasing to testing. Inspection can give you the 'bad news' ***clearly***, so that you realize that ***Inspection*** has not 'failed' you - when all those bugs pop up in test stages. ***You*** have failed to use Inspection properly, to measure and control the engineering process.

If you examine the numbers [9] neither 'testing better', nor misusing Inspections in a 'Clean up' mode is going to give you the bug reduction you want, with resources you want to spend. Your only hope is to reduce defect injection, using Inspection to train and motivate individuals to 'follow the Rules'. You must also improve the engineering processes so that ***the 'workers are not forced to commit errors'*** (Deming's teachings). We have all heard the idea that ***'you don't get quality by testing it in, but by designing it in'.*** Why don't we act on that idea? Inspection in 'cleanup mode' is simply a cheaper way of 'testing it in'! But, still doomed to failure.

FOCUS ON MAJOR DEFECTS and Specification Rules to define them.
If you do not consciously manage the process, people will unconsciously spend 90% of their Inspection effort finding and fixing minor defects. Minor defects, using my definition have no value of saving time or effort downstream. They will not change the product in any customer-important way. The only valid use of Inspection is to find and measure the presence of Major Defects, defined as those which potentially will cost you interesting effort if first discovered downstream of Inspection (test, field use). One of my clients (3, p.315, Philips UK) studied 1,000 defects classified as 'Major'. They asked their own maintenance and test people what the cost downstream would be and the media answer was 9.3 hours. The average cost of finding and fixing them each was about one hour. The CEO and his team used this evidence to demand full use of Inspection.



There are a dozen or more specific tactics to ensure that your Inspection activity does not waste it's time with minor defects. [3, pp. 74-75 detailed list]. They include:

1. Defining defects exclusively with the help of a formal set of specification Rules [3, pp. 424 on], and then including only Rules which lead to Major defects, avoiding those which are primarily minor.
2. Giving aid to interpreting existing specification Rules using checklist questions which are directly supporting referenced specification Rules, and which particularly help Checkers find Major Defects.
3. Focusing all calculations on Major defects: for example the Exit levels of remaining Major Defects per Logical Page (300 Non commentary words).
4. Calculating the Optimum Checking rate for specific document types based on the peak observed ability to detect Major defects, and not including the minor defects.
5. Making it a serious Entry condition to any engineering process, including Inspection (for Source documents), that the Defect density, of documents used, does not exceed an economically sensible level of Major defects/Logical Page. For example initially Maximum 3.0 Majors/Logical Page and ultimately 0.3 Majors/Logical Page.
6. Insisting that all technical documents (as source code does naturally) make an agreed distinction between specs that count and background commentary which does not translate into product and economics. It is then easier to spot Majors.

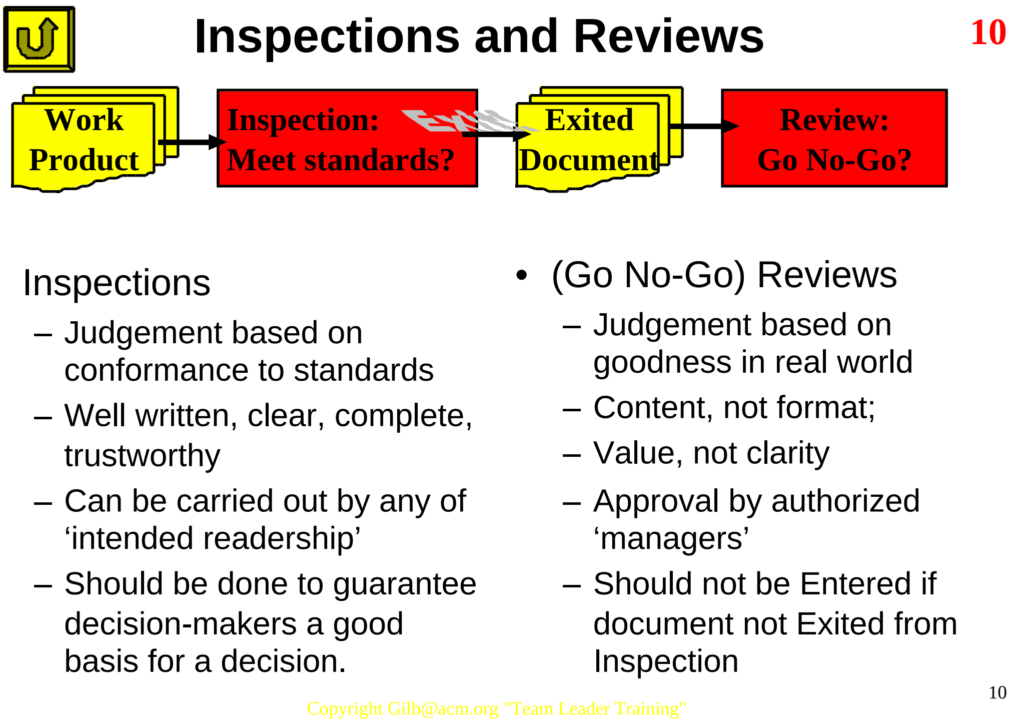
Avoid using it for Reviews:

Inspection should not be confused with other types of Reviews. The purpose of Inspection is to measure the degree of compliance with the specific specification process. There are many other partial objectives for using Inspection (I have noted over 20). But these do not include judging the 'objective quality of the ideas expressed', in relation to the real world needs. This I would call a Go-No Go Review (and there are many other names in circulation). Inspection is limited to determining if the people and process have complied with the current recommended best practices for a particular kind of specification (I assume these are codified in a continuously improved process standard, which I call 'Rules' (there are many other names for this).

If a large number of Major Defect type Rules (like "All Quality Requirements will be quantified and testable in practice") are violated per Logical Page of specification, then Inspection tells us the economic consequence (average about 8 - could be worse! - lost hours downstream per Major) of releasing the document to any other engineering process; including Technical or Management Reviews.

In fact Inspection should be a required process before any serious document is reviewed. The entry criteria to the Technical 'Go-No Go-type' review should be set conservatively (no more than 0.3 Majors maximum possibly remaining per Logical page) so as not to waste the time of the senior people involved, and to reduce the risk of dangerous decisions being made for, or against, details which are poorly specified.

Most companies I see, do not have a clear distinction between the use of *Inspection* to determine whether a *document* is economically safe to use, and Technical reviews to determine if the *ideas* in the document (Example "Marketing Requirements", or "Test Plan") are *sane economically* in that 'Jungle out there'. We must learn the distinction between *eloquent* and *relevant*. When I find that uncontrolled software engineering processes normally allow 20 or more Major Defects to remain per Page, then all other technical processes using those documents are bound to be frustrating and uneconomic; and to cause further chaos downstream. We are typically not aware of this. We just 'do our best'. Inspection makes us culturally aware of the dangers of 'rushing in where angels fear to tread'.



Inspections and Reviews

10

Work Product
Inspection: Meet standards?
Exited Document
Review: Go No-Go?

- Inspections
  - Judgement based on conformance to standards
  - Well written, clear, complete, trustworthy
  - Can be carried out by any of ‘intended readership’
  - Should be done to guarantee decision-makers a good basis for a decision.
- (Go No-Go) Reviews
  - Judgement based on goodness in real world
  - Content, not format;
  - Value, not clarity
  - Approval by authorized ‘managers’
  - Should not be Entered if document not Exited from Inspection

10

Copyright Gilb@acm.org "Team Leader Training"

Quality

Why Inspection is a 'loser' for quality

Most people have the illusion that Inspection is something they do for software quality. First part of this illusion is that they invariably have a simple-minded definition of quality: bug freeness. Quality in the eyes of our customers is much more than bug freeness.

And the second illusion is that they can use inspection to raise this narrow measure of 'quality'. This is immature.

As mentioned once earlier, it is an oft quoted notion that 'you cannot test quality into a system, you must design it in'. Software people do not seem to understand that paradigm. Using Inspection to remove defects is simplya cheaper way to 'test quality in'It works, poorly; and it costs too much. Its only saving grace is that if you are in such bad shape that you have injected too many bugs (which is the real problem, injection) then removing some of them by Inspection is an order of magnitude cheaper than removing them with testing.

But Inspection in a 'clean-up' mode is still a bad option. Reduction of injection, through design, through individual learning, and through process improvement are clearly the smart economic ways to go.

Of course, when all the bugs are gone, we have no more 'quality' problems? Right?

Wrong! We have lots of them. Some of them are far more important to your customers than infrequent bugs. They include

reliability (which isnot the same as bug density!), availability, usability, performance, installability, portability, security, customizability, extendibility, maintainability, recoverability; and when you begin to look at specific applications a very much longer list of quality and cost concerns for the customers and other 'stakeholders' (such as your corporation, your dealers, your help lines).

Inspection in the sense of 'bug removal' won't help these qualities significantly at all.

How to properly use inspection for quality

If you want to make use of Inspection to help get adequate and competitive levels of these qualities, you can. You should use Inspection *to quality control the entire chain of software engineering specification*, from Marketing Dreams, and Customer Contracts at the front end, to code and test cases at the other end. This will help all technical activity remain consistent with the dreams, expectations and the designs for quality. Inspection is a valid part of the engineering process. But it is *not* the most important part. It is merely *one* of the *many* quality control tools we will need to use.

How to get qualities into your products.

I am not going to be able to explain and argue this next proposition in the detail I know many would appreciate (see www.result-planning.com for that detail). But in order to make my main point about the role of Inspection, I have to at least sketch the position.

My assertion is very well known and understood in much of the civilized world of technology. You get the multiple qualities you want by specifying the numeric testable levels you want, together with cost and other constraints. This is called 'requirements' (not to be confused with software functional requirements). You then use a process of architecture/design/engineering to come up with the technology needed to reach the requirements levels [10]. This is not to be confused with structural decomposition of software, which some programmers consider 'design' work.

You can reach the quality levels you want if they are

- technologically achievable (inside state of art)
- consistent with any constraints, and
- consistent with the simultaneous achievement of the *other* quality levels, when considering a common set of constraints (such as deadline and budgets).

It should be obvious that the present culture of software engineering does not even begin to outline or teach the necessary disciplines for doing this. They are provably incapable of even specifying the quality requirements in a clear quantified way, let alone equipped with the disciplines necessary for solving the multidimensional design problem.

Sure, we do it, sometime through sheer brilliance and intuition. But we don't have a teachable design-engineering discipline. We do not even discuss its absence. Most people reading this might disagree, but *that* is a symptom of the problem. We don't even realize we have a problem. I cannot argue this further here, and I do not expect to convert this sluggish culture overnight. But I would be happy if some key intellectual leaders of our discipline would begin to move in the right direction. I personally intend to be there all the way, and I am pretty sure that is the rest of my life.

How does Inspection relate to overall software development ?

Inspection, properly used, is the best known tool for checking that written specifications are consistent with specification rules, and by extension with other related documentation upstream.

==

I am temporarily leaving these IEEE Outline things in place in case someone wants to move material there or force me to add comment. But I feel that I have adequately done 3 and 4 above. Tom Gilb 20 Jan 1999

=====================================

3. The Position

Explain the position.

Provide evidence to support it.

Justify it.

4. Common Practices

a) Explain the "opposite" camp(s).
b) Explain common practices.
c) How do practitioners currently address the issue?

======= end temporary outline retainment TG=================

Summary and Conclusions

Inspection is a general *systems* engineering process, highly applicable to software *engineering* in the broadest sense of that term. It applies to all known forms of technical and managerial documentation and specification. There are many variants of the Inspection process, but one common characteristic is that is a *rigorously applied quantified* discipline [1, 3], *not* an intuitive review.

Inspection should not normally be used to 'clean up' logic bugs, because that is neither effective nor cost-effective in that task. It is far more cost-effective to get required quality levels by means of *design*, by individual training, and by process improvement. Inspection should be used as a specification *quality control* tool.

The 'quality being checked' is the *specification consistency* with your *best practice standards*. Inspection should be used to determine that a specification *is economically safe* to release to other software engineering processes.

In particular Inspection should be used to determine the *probably remaining Major defects/Logical Page*, by using representative *sampling*, not the brute force 100% 'sampling' of conventional software inspections [1]. This *defect level determination* should be used to determine official 'Exit' of any specification from its creation or maintenance process.

Inspection should *only be used where it is profitable*, and profitability of the process should be *continuously monitored*. The use of the process should be *continuously justified*. In particular it should not become a mandated bureaucratic process, which cannot otherwise justify its existence amongst the rich selection of software engineering 'tools' for achieving the same ends. The engineers and project management should be given a choice of tools to achieve well-defined project targets.

Inspection is about the *economics* of software engineering. It is not a cost effective way to improve quality, or even bug-freeness, when used in the conventional *'clean up'* mode. The cost-effective route is to *reduce defect injection into specifications* through architecture, design, engineering processes, and individual professionalism. Inspection can play a useful role in training engineers in these processes, enabling them to improve the processes, and motivating them to do so.

References (NB not more than 12)

[1] Fagan, M. E. Design and Code Inspections: Dec 1974 Technical report Kingston, Later published in 1976 IBM Systems Journal. Republished in [5]. His website **http://www.mfagan.com/**:

[2] Susan Strauss & Robert Ebenau, **Software Inspection Process,**

Copyright 1994 (by AT&T) , November 1993, McGraw Hill, 363 Pages, $40

[3] Gilb, T, Graham Dorothy, Software Inspection, Addison-Wesley Longman, October 1993. Up to Date Inspection slides at www.result-planning.com.

[4] Ron Radice new book on Inspections/ His TR on Inspection 1973, stt@stt.com (Ron Radice Email, His website www.stt.com

[5] Bill Brykczynski and David Wheeler

Software Inspection an Industry Best Practice.
A compendium of Articles on Inspection . IEEE?? <tom has this at home but not on road>

[6] <was inadvertently blank, keep it for next one?>

[7] Grady, Robert B, and Slack, Tom Van, Hewlett-Packard

“Key Lessons in Achieving Widespread Inspection Use”

IEEE Software July 1994 pp. 46-58. Republished in [5]

[8] Technical Report
CMU/SEI-95-TR-017
ESC-TR-95-017

Raytheon Electronic Systems Experience in Software Process Improvement

Tom Haley
Blake Ireland
Ed Wojtaszek
Dan Nash
Ray Dion

- November 1995
  - http://www.sei.cmu.edu/products/publications/95.reports/95.tr.017.html
  - See also initial report

Raymond Dion, **Process Improvement and the Corporate Balance Sheet** (July 93)

IEEE Software, July 1993, pp. 28-35

[9] Capers Jones <book I have at home with all statistics. I asked TS to provide the reference> NB his def effectiveness opposite normal and that used here.
[10] Gilb, T., Principles of Software Engineering Management, Addison Wesley Longman, 1988

[10] J. L. Pete Pence and Samuel E. Hon III,

"Building Software Quality Into Telecommunications Network Systems",

Telecommunications Network Systems,

Bellcore Piscataway NJ
Quality Progress, Oct 1993 pp. 95-97

============end of paper
Author Email
Gilb@acm.org