Testing, Reviews, and Inspections

Daniel M. Berry

Syllabus

I. Background and Theory of Inspections
II. In-class Inspection of Instructor's Document
I. Group Inspection of Document Provided by One Member

Outline -1

- Sources of information
- Background
- Bugs come early and stay
- Finding bugs with reviews
- Psychological impediments

Outline -2

- Procedures for doing reviews
- Experimental evidence
- Why inspections work
- Starting an inspection program
- Evaluating effectiveness of program

Sources of Information -1

M.E. Fagan, *Design and Code Inspections and Process Control in the Development of Programs*, Technical Report IBM-SSD TR 21.572, IBM Corporation, December, 1974

M.E. Fagan, "Design and Code Inspections to Reduce Errors in Program Development", *IBM Systems Journal* 15:3, pp. 182–211, 1976

Sources of Information -2

T. Gilb and D. Graham, *Software Inspection*, Addison-Wesley, Wokingham, UK, 1993

R.G. Ebenau and S.H. Strauss, *Software Inspection Process*, McGraw Hill, New York, 1994

Background

- **History**
- **Definitions**
- **Pithy Quotes**

History -1

Inspection technique was developed by Michael E. Fagan at IBM Kingston.

Fagan was a certified quality engineer and studied the methods of Deming and Juran.

He used inspection on a SW project he was managing in 72–74, in effect applying industrial hardware quality methods to SW.

History -2

It was *very* successful!

He reported results in a now famous 1976 paper.

The method became very popular in IBM, although there was some resistance.

History -3

AT&T Bell Labs started using technique in 1977.

In 1986, a major Bell Labs SW development organization with 200 people reported its experience with inspections:

History -4

- **14% productivity increase for a single release**
- **better tracking and phasing**
- **early defect density data improved 10-fold**
- **staff credited inspection as an ‘‘important influence on quality and productivity’’**

Definition -1

ANSI/IEEE Standard 729-1983 *IEEE Standard Glossary of SE Terminology* defines inspection as

“... a formal evaluation technique in which software requirements, design, or code are examined in detail by a person or group other than the author to detect faults, violations of development standards, and other problems....”

Definition -2

The ANSI/IEEE Standard 1028-1988 *IEEE Standard for Software Reviews and Audits* defines the objectives of SW inspection as

“to detect and identify software elements defects. This is a rigorous, formal peer examination that does the following:
(1) Verifies that the software element(s) satisfy its specifications.

Definition -3

(2) Verifies that the software element(s) conform to applicable standards.
(3) Identifies deviation from standards and specifications.
(4) Collects software engineering data (for example, defect and effort data).
(5) Does not examine alternatives or stylistic issues.”

Pithy Quotes -1

Mostly from Gilb & Graham:

Gilb & Graham’s Prevention Principle:
Prevention is better than cure.
or
An ounce of prevention is worth a pound of cure.

Pithy Quotes -2

Santayana’s Principle:
Those who do not remember the past are condemned to relive it.

The Sewing Principle:
A stitch in time saves nine.

Bugs Come Early and Stay

- **Recall the myth**
- **Basic reality of bugs in design**
- **Costs of finding bugs**
- **Bugs probably required**
- **Requirements are difficult**

Recall the Myth

Bug

The word itself is a myth.

It implies that a bug is something that an otherwise healthy program gets,

maybe as a result of contagion from sitting in the same memory with other buggy programs?!

Basic Reality of Bugs in Design -1

If a bug exists in a design, then if that design is implemented, the bug will be in the code, inevitably.

’Tis only logical!

Basic Reality of Bugs in Design -2

So the issue is not *if* there is a bug or *if* it will be found, but ...

“By whom and when will the bug be found?”

- **by developers before delivery**

OR

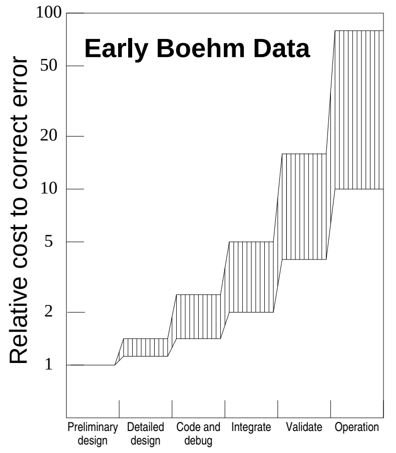
- **by customer after delivery**

Former is better both for costs and company’s image and customer good will.

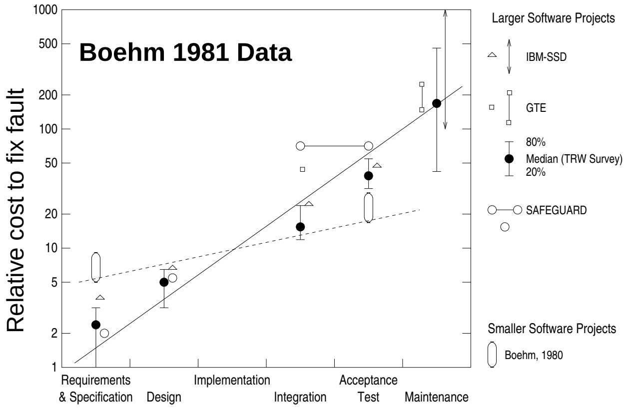
Costs of Finding Bugs

Graphs on the next slides show that the latter costs one to two orders of magnitude less.

Same and worse is true of hardware!!



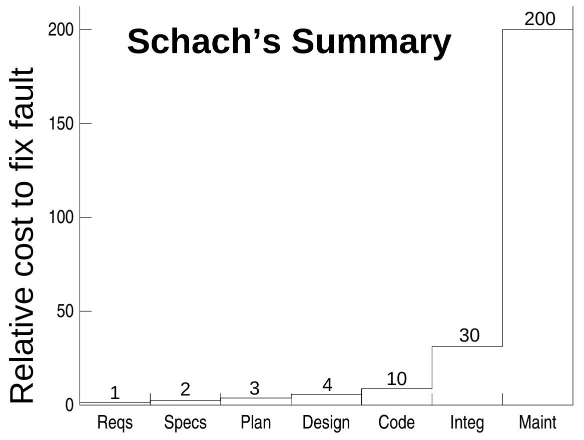
Phase in which error is detected



Phase in which fault was detected and fixed

Boehm's 1981 Numerical Data

Cost	**Stage**
1	**requirements & specification**
3–6	**design**
10	**implementation**
15–40	**integration**
30–70	**acceptance test**
40–1000	**operation/maintenance**
	(82 is IBM average)



Phase in which fault is detected and fixed

Bugs Probably Required

The bug that shows up after delivery to the client was probably *required* into the software, with probability 85%, according to Barry Boehm's data, and with probability 56% according to DeMarco's data.

Why?

Requirements are Difficult -1

Martin & Tsai's study of requirement errors:

They conducted an experiment to identify lifecycle stages in which requirement errors are found.

An experienced user produced a polished 10-page requirements document for a centralized railroad traffic controller.

Requirements are Difficult -2

Ten 4-person teams of software engineers were given the requirements document in order to find errors in it.

The user believed that the teams would find only 1 or 2 errors.

Requirements are Difficult -3

92 errors, some very serious, were found!

The average team found only 35.5 errors, i.e., it left 56.5 to be found downstream!

Many errors were found by *only* one team!

The errors of greatest severity were found by the fewest teams!

Requirements are Difficult -4

CONCLUSIONS: Requirements are *hard* to get right!

Errorer

This is a new term I invented to describe the entire class of causes of errors. It includes all the reasons that errors are put into programs, ranging from sloppiness, through writing something not intended, through not understanding the phenomena involved, through not understanding the user, to just not being able to see the full picture.

Finding Bugs with Reviews

- **Finding bugs earlier**
- **Sad fact**
- **True cost of reviews**
- **Panicky manager**
- **Pressman quotes Machiavelli**

Finding Bugs Earlier -1

So the question is: “How do you find the bugs earlier?”

Answer: The same way as you find them later, by *testing*! Nu?!

But but ... this means that you cannot find them earlier than the implementation/coding phase because you cannot run test cases before you have code to run! Nu!?

Finding Bugs Earlier -2

Ah ... so let us redefine testing so that it can be done on non-executable products like requirements and designs

Testing includes reviewing (later we will distinguish walkthroughs from inspections).

Let the good ol’ human keppele test the requirements and designs.

Finding Bugs Earlier -3

But hey!!! Why not do this kind of testing on executable products too?

After all, humans can see and understand much more of a global picture than can a computer, whose vision is very local and stupid.

OK, OK, but what do you review against?

Finding Bugs Earlier -4

You review against the same thing that you test against, i.e., an acceptability criterion, i.e,

- **test data and expected results derived from requirements, OR,**
- **that the product meets its requirements**

Finding Bugs Earlier -5

So reviews are considered the testing of non-executable documents.

One should do reviews and traditional testing, if possible, on all deliverable products of the lifecycle, including test plans!

Finding Bugs Earlier -6

For test plans, the inspection is for coverage and conformance

- **that the test case input *covers* the input domain**
- **that the expected output for each test case *conforms* to the requirements**

Deliverable products are not considered delivered unless they have passed inspection and tests as applicable.

Sad Fact -1

There is never enough time to do it right, but there is always enough time to fix it or to do it over.

However, it always takes more time to fix it than to have done it right or to do it over (not even counting the fact that you have done it twice).

When you fix it, it is always flaky and never quite fixed.

Sad Fact -2

“We don’t have time to do reviews | walkthroughs | inspections!”

If so, then you don’t have time to finish the project at all,

because without the reviews, you will send out something unfinished if you send anything out at all!

Sad Fact -3

What is doing it right?

It is testing or reviewing every product of the lifecycle at the time it is produced, *before* going on to the next step.

It is fixing the bugs found in the reviewed product *before* going on to the next step.

Sad Fact -4

Why?

Because, as shown in the graphs a few slides back, the cost to fix a bug grows dramatically with any delay in its discovery or its fixing.

Contributing to this cost is the cost of redoing anything mistakenly done as an implication of the buggy part, i.e., of redoing anything that is undone by the fix.

True Cost of Reviews -1

Introducing reviews, in all likelihood, decreases the total cost of a project.

Reviews find errors that are there already, so no matter what, any error a review finds earlier would have to be dealt with at some time, possibly even after delivery of the software, if there were no review.

True Cost of Reviews -2

But, fixing the error now is considerably cheaper than fixing it later (recall graph).

The question, then, is which is greater, the time to review and fix it earlier or the time to fix it later?

True Cost of Reviews -3

A five-person review costs five person-days.

Consider an error that takes one person-day to fix if found during the requirements phase:

Review + Fix Reqs	Review + Fix Impl	Fix after Delivery
5+1	5+2	10

True Cost of Reviews -4

If *n* errors are found,

Review + Fix Reqs	Review + Fix Impl	Fix after Delivery
5+n	5+2n	10n

Fix-after-delivery costs grow faster!

Panicky Manager -1

There is a story of a Panicky Manager who agreed to initiate inspections on the basis of glowing predictions of improved productivity and reduced errors such as on these slides delivered by a slick software methods salesperson ....

Panicky Manager -2

The software was estimated to require 50 KLOC.

The basic level of COCOMO estimates for an embedded organization that 50 KLOC will require about 60PM to build.

The IBM rule of thumb is that inspection adds 15% to the resources required, i.e., an additional 9PM.

Panicky Manager -3

"Nine PM! We don't *have* that much available! The development budget is for 60PM and not a PSecond more!"

So where is the panicky manager going to get the additional PM?

Panicky Manager -4

The problem is that the panicky manager is taking the inspection PMs out of the wrong budget!

It should come out of the post-development maintenance budget!

Panicky Manager -5

Data show that it takes 1.58 PHour to find a single defect in inspections.

There are 1188 PH in 9PM, so one can expect to find 752 defects in 9PM.

It is known that it typically takes 9PH to fix a defect found after delivery.

So these 752 defects will require 51PM to fix, almost as long as it took to build the software.

Panicky Manager -6

If the same defects are found earlier by inspection, say no later than coding stage, then it will take 1PH to fix each.

Therefore with inspections, these 752 defects will require 5.7PM to fix.

Panicky Manager -7

So,

W/O Inspection	**60+51 = 111PM**
W Inspection	**60+9+5.7 = 74.7PM,**

a savings over the lifecycle of 32%.

Panicky Manager -8

This does not always solve the problem of where to get the additional resources, because in some organizations, maintenance is *not* budgeted; rather, developers are pressed into service, borrowed from their next projects.

Pressman quotes Machiavelli -1

“ *... some maladies, as doctors say, at the beginning are easy to cure but difficult to recognize ... but in the course of time ... become easy to recognize but difficult to cure.*”

Pressman quotes Pressman

Indeed! But we just don’t listen when it comes to software. Every shred of evidence indicates that formal technical reviews (for example, inspections) result in fewer production bugs, lower maintenance costs, and higher software success rates. Yet we’re unwilling to plan the effort required to recognize bugs in their early stage, even though bugs found in the field cost as much as 200 times more to correct.

What Evidence?

We'll look at some later, but first let's attack the psychology of resistance to reviews, and then describe the procedures of doing reviews in detail.

Psychological Impediments -1

- **Heard during reviews**
- **Necessity of reviews and tests**
- **Implication of finding bug**
- **Definition of testing**
- **Implication of definition**
- **Happiness at discovering a bug**

Psychological Impediments -2

- **Shame in discovering bugs**
- **Team ownership**
- **Exposing errors to superiors**
- **Adobe's attitude**
- **Bug-finding tools**
- **A real tragedy**

Heard During Reviews

Complaining about the level of detail required in preparation of reviewed documents:

"But that's *so* easy to (fix | deal with)!"

True, but if so, then how come it's not already (fixed | dealt with)?

Necessity of Reviews and Tests

An old adage:

The work of the person or team that insists the loudest that no review or test is needed is most in need of review or tests.

Implication of Finding Bug -1

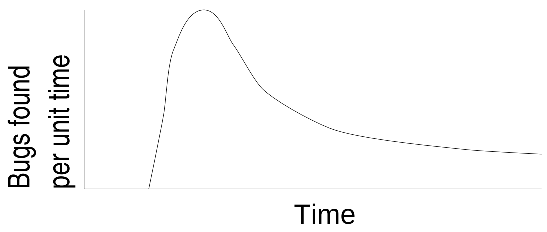
Mills says:

“The best way to know that you have found the last bug is never to find the first bug.”

Any bug, even a tiny one, is a sign of an errorer in the development, and an errorer leads to lots of bugs, never just one.

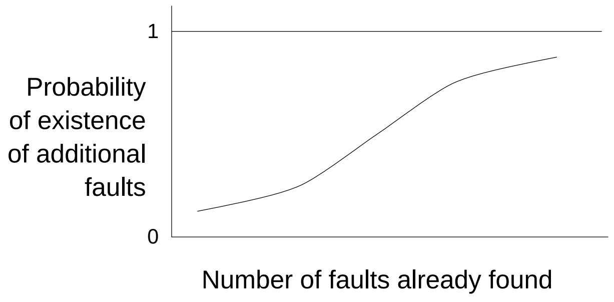
Implication of Finding Bug -2

Studies have shown the following bug arrival graph over the testing of one release.



That is, they tend to arrive in bunches if they arrive at all.

Implication of Finding Bug -3



Implication of Finding Bug -4

Finding one bug is a symptom of an errorer.

There is no reason to expect that even a single errorer causes only one bug.

This applies both to semantic and syntax errors!

In particular, a syntax error is a symptom of a semantic error.

Definition of Testing -1

Often hear:

> **Testing is confirming that program works.**
>
> **or**
>
> **Testing is demonstrating that errors are *not* present.**

Definition of Testing -2

Nonsense! wrong! bubbe meises!

Already know that:

> **Program testing can be used to show the presence of errors but never their absence**
> **— E.W. Dijkstra**

Definition of Testing -3

∴ The proper definition of testing is:

> **Testing is executing a program with the intention of finding errors. — G. Myers**

Implication of Definition

These quotes suggest a certain mind set for testers.

They should be trying to find errors rather than trying to show that there are none.

Goals have a bad habit of turning into reality!

Psychology of Testing -1

A program is its programmer’s *baby*!

Thus, trying to find errors in one’s own program is like trying to find defects in one’s own baby.

∴ It is best to have someone other than the programmer doing the testing.

Psychology of Testing -2

Tester must be highly skilled, experienced professional.

Helps if he or she possesses a diabolical mind

“Heh ... Heh ... Heh!” — Count Dracula

Psychology of Testing -3

It is well known that what is achieved in any endeavor depends a lot on what are the goals.

Myers says:

If your goal is to show absence of errors, you will not discover many.

If your goal is to show presence of errors, you will discover large percentage of them.

Psychology of Testing -4

If you are trying to show the program correct, your subconscious will manufacture safe test cases.

∴ Tester should be someone other than the programmer, who just *loves* bugs.

Happiness at Discovering a Bug

While it is painful to realize that your own code has a bug, in terms of total effect, you should be happier to find them before delivery than after,

because, in any case, they *will be* found, if not by you, but by your customer!

Shame in Discovering Bugs -1

There should be no shame in finding bugs in your code during review.

The shame is releasing the code with the same bugs.

I consider myself to be a good programmer; I release only good code.

Shame in Discovering Bugs -2

But, I will let you in on a secret (sh! sh!)

When I am modifying previously written code, fully 50% of the lines that I write have bugs.

Shame in Discovering Bugs -3

It's very hard to do modify existing code right because of unanticipated ripple effects.

Therefore, I use defense mechanisms, reviews and line-by-line changing/testing.

I end up releasing mostly bug-free code.

Knuth has No Shame! -1

Donald E. Knuth wrote TEX starting in May 1977.

By Sept 1988, it had grown to 14 KLOC in Pascal, and he sent in a final version of a "The Errors of TEX" to be published in *Software, Practice & Experience*, July 1989.

Knuth has No Shame! -2

He proudly reported 867 errors, including bad requirements.

"But I see no harm in admitting the horrible truth of my tendency to err, when such details might shed light on the problem of writing large programs. (Besides I am lucky enough to have a secure job.)"

Knuth has No Shame! -3

Knuth had published every version of the source program and had a whole world of TEX hackers as inspectors.

He paid a small (≈ $20) monetary reward to the first person who found any particular error; thus he had motivated, "professional" inspectors!

All of this is described in open publications!

Team Ownership

It helps to change the philosophy of code ownership.

The *team* owns the code, and while individuals write parts, it is the *team* that will release and stand behind the code.

This is a case of the team being more than the sum of its individuals.

When is a bug a bug?

In line with the team ownership philosophy, a bug should not be considered a bug *unless* it is in released code.

A bug inserted prior to inspections and removed by inspection is considered the part of the normal process of software development.

Only bugs in released code are logged!

Exposing Errors to Superiors -1

But I don’t want my code reviewed because then my manager will know that the bugs found came from *me*!

First, it is necessary to adopt the team ownership of code and bugs philosophy.

Exposing Errors to Superiors -2

Second, it is necessary that bugs found before and during inspection not be considered bugs, just hiccups!

Thirdly, the inspection procedure is adamant in insisting that the line manager of the author of the inspected product *not* attend the inspection.

Adobe’s Attitude -1

I found the following pages in the manual for Adobe’s Acrobat 2.0.

Adobe's Attitude -2

Notice that:

- **the two biggest groups are Development (Engineering) and the Quality Assurance**
- **the sizes of D and QA are about the same,**
  (In fact, QA has one more than D!)
- **D and QA have no members in common**
- **D and QA get essentially equal and top billing**
- **Adobe products *are* good!**

Bug-Finding Tools? -1

Why not let tools find defects in the code for us?

First it's not clear that tools can find many errors beyond syntactic or so-called static semantic (the good 'ol halting problem rears its ugly head!)

Bug-Finding Tools? -2

The Gilb & Graham Tools for Fools Principle:

> **Automated tools *should* be used to find problems, but not if you must delay detection until the fixing costs outweigh the advantage of automation.**

A Real Tragedy -1

At this company I know, among the dozen projects they had going, only *one* project met its final deadline.

Nearly all of these deadlines had been slipped from earlier impossible deadlines.

The one project that met its deadline was the one that I helped out by moderating inspections for it.

A Real Tragedy -2

I had been called in by the Marketing VP to use inspections as a means to help a project make a one-month deadline for a trade show, at which they had committed to announce and demonstrate the product.

Against the wishes of the President, the VP of R&D, the Chief Programmer, and all the programmers, I instituted an inspection of the product.

A Real Tragedy -3

Even though the Chief Programmer and the programmers were dragged kicking and screaming into the inspection, the first inspection found a whole lot of bugs, including one that would have killed the product and would have cost an unspecifiable amount of time to find and fix (so said a development consultant observing the project).

A Real Tragedy -4

A second inspection failed to find any new problems and the project finished on time, the product was announced and demonstrated at the trade show, *and it won an award at the trade show!*

Sadly, the President, the VP of R&D stood behind the Chief Programmer, who was opposed to inspection despite the success, and refused my offer to start a regular inspection program.

A Real Tragedy -5

The company never met another deadline and is in complete doldrums, because it has not been able to meet the demands of the market before its competitors did.

A real tragedy.

Procedures for Doing Reviews

Two kinds of reviews

- **walkthroughs**

- **inspections**

Latter more formal than first.

Outline of Discussion

- **Common elements**
- **Differences**
- **Walkthroughs**
- **Inspections**

Common Elements

Both:

- **Review is an in-depth examination of some work product by a team of reviewers.**

- **Product is anything produced for the lifecycle, i.e., requirements, plans, design, code, test cases, documentation, manuals, everything!**

- **No meeting is more than 2 hours.**

Common Elements

- **The focus is on finding errors in the product,**
  - **not on correcting them, and**
  - **not in finding fault in producer of product.**

- **It is essential that it *not* be used for employee performance evaluation, either of producer or reviewers.**

Not for Appraisal

This last point is critical.

If reviews are as effective in finding errors as will be shown, then any manager that uses them for evaluating producer or reviewers will kill the goose that lays the golden egg!

Differences -1

Atmosphere:

W: informal

I: formal

Differences -2

Reviewers:

W: experts in the domain

I: trained, professional inspectors

Inspectors Not Domain Experts? -1

But, but, ..., how can someone who does not understand the program be expected to find anything?

This is *such* waste of time, making someone who knows nothing about the program examine it carefully when someone who knows it can find the errors *much much* quicker.

Inspectors Not Domain Experts? -2

Well, the data, time and time again, do show that inspections, as defined above, *are* significantly more effective in finding errors than walkthroughs, as defined above.

So, evidently, the lack of domain knowledge of the inspectors is not a problem.

Inspectors Not Domain Experts? -3

There is other evidence that in some circumstances, a person who does not know the domain is more effective than someone who does, because the former is less likely than the latter to fall into the tacit assumption tar pit.

An every-day occurrence of this phenomenon is that someone who has never seen an article finds more errors in proof-reading an article or book than does the author.

Inspectors Not Domain Experts? -4

I know from one experience I had participating in the inspection of requirements and design of some networking software that this phenomenon applies to inspection.

I came to the meeting prepared with a 20 page list of possible problems in the document, while the members of the development team participating in the inspection came with none (they thought there were no problems).

Inspectors Not Domain Experts? -5

In this list were 5 really serious problems which the development team members recognized instantly when I asked seemingly innocent questions like "Why is this variable used here, but not here in this other module that #includes the module defining the variable?"

In other words, I found errors that had been just under the noses of the development team.

Differences -3

Subject of review:

W: correctness of product, as seen by experts

I: correctness of product, according to checklist of items to be examined

Differences -4

Leader of discussion and session controller:

W: producer of product

I: official moderator of review team

Walkthroughs

The producer presents product (if code, then also its documentation) and the reviewers comment on the correctness of the product (if code, also consistency with documentation).

Inspections

- **Roles**
- **Checklists**
- **Steps**
- **Users at Inspections**
- **Caveats**

Roles

Inspection team members have specific roles:

- **moderator**
- **secretary (can be same as moderator)**
- **inspectors (2)**
- **producer(s)**
  - **designer**
  - **programmer**

Checklists -1

Product is examined relative to specific checklists, e.g.,

- **compliance to standards**
- **domain correctness**
- **consistency with requirement items**
- **language usage**
- **use-declaration type consistency**

Checklists -2

- **invocation interface consistency**
  - **formal-actual correspondence**
  - **return value**
- **use of correct dimensions (e.g. °C vs °F)**
- **comment-code consistency**
- **avoidance of errors in historical list (frequency ranked)**

Steps -1

5 Steps:

1. **Overview given by producers to reviewers; at end of session, the relevant documents are distributed to reviewers (could be combined with a walkthrough!)**

Steps -2

2. **Reviewers prepare individually for inspection, try to understand the product in detail; should be aided by checklists; ranking lists by importance allows focusing on most important potential errors first. (Maybe each inspector can have a different checklist.)**

Steps -3

3. **Inspection session itself: moderator walks through the product line by line, asking inspectors for observed problems (sometimes hearing another's fault reports triggers discovery of more); within one day, moderator produces written report on all faults found**

Steps -4

4. **Rework: the producer fixes the product according to list of faults in report**

The producer might ask for assistance in solving the problem.

Gilb & Graham suggest brainstorming as a way to get free flowing of ideas that might provide a solution.

Steps -5

5. **Follow up: the moderator makes sure that all faults have been fixed and calls another inspection if more than 5% of the product has been modified**

Steps -6

In my opinion, we must always do one more inspection than might seem necessary, i.e., until inspection yields no new errors.

It is not safe to skip this last inspection, even if there was only one itty-bitty, tiny little ol’ error; its fixing might introduce a great big, ferocious error!

Users at Inspections -1

It may also be useful to have a user as part of the inspection team, especially if a requirements document is being inspected, but also even if code is being inspected.

The first Airbus crash resulted from the computer shutting down the engine while the plane was in the air, as a result of an inconsistent state.

Users at Inspections -2

The fix was not to allow the engine to shut down if the plane is still in the air.

A later crash came when the pilot could not turn off the engine in a plane that had just landed!????

Users at Inspections -3

The way the software detected that the plane was on the ground is by seeing that the wheels are spinning; if the wheels are not spinning, the plane is still in the air.

The crash happened when the plane landed on a wet runway; the wheels were not spinning and the engine would not turn off.

Users at Inspections -4

So the software was changed to use pressure on the wheels to determine if the wheel is on the ground and the definition that the plane is on the ground if both wheels are on the ground.

The next slanted landing resulted in a crash because the pilot could not turn off the engine soon enough.

Users at Inspections -5

All of these problems could have been avoided by having users, i.e., experienced pilots, in the inspection teams.

Caveats

- **Team members must read product before meeting.**
- **Meeting must not be longer than two hours.**
- **Author's boss cannot be present at the meeting.**
- **Inspectors must not consider solutions during meeting.**
- **Inspectors must not attack author; they can criticize product.**

Experimental Evidence -1

Experiments over several projects have shown that inspections can be 4 times more effective than traditional testing and 2 times more effective than walkthroughs in finding errors.

Experimental Evidence -2

- Effectiveness in finding bugs
- Effectiveness & needed resources
- Cost reduction
- Time reduction
- Defect reduction
- Increased productivity
- Unmentioned in ALL Reports
- JPL study
- Inspection meetings
- Programmers’ surprise
- Reeve’s rule of thumb

Effectiveness in Finding Bugs -1

According to experiments by Fagan reported in ’76, ...

67% of all errors eventually found for a particular system, were found by code inspection before unit testing, and

for a similar system, informal walkthroughs were used instead of inspection; the inspected system had 38% fewer errors in the first seven months of operation than the walkedthrough system.

Effectiveness in Finding Bugs -2

Jones reported in 1977, that ...

He examined the history of 10 million lines of code.

Inspection removed 85% of the total errors found.

No other technique, walkthrough, testing, etc. found even 50%.

Effectiveness in Finding Bugs -3

Jones reported in 1978, that ...

Inspection removed 70% of the total errors found in another set of projects.

Effectiveness in Finding Bugs -4a

Kitchenham reports that at ICL, in one project 57.7% of the defects were found in inspections and only 4.1% of them were found in in-house use. The normal rates at the time was for 11.2% of the defects to be found in in-house use.

The cost of finding one defect in design inspection was 1.58 work-hours. The cost of finding one defect without inspection was 8.47 work-hours, more than 5 times more.

Effectiveness in Finding Bugs -4b

The total proportion of the development effort devoted to inspections was only 6%.

Kitchenham feels that these figure would have been better if better training in inspection had been given.

Effectiveness in Finding Bugs -5a

Grady reported in 1994, HP's data over several years and projects:

The average effectiveness of code reading/code inspections was 4.4 times better than that of any other test technique.

Effectiveness in Finding Bugs -5b

Testing Technique	Effectiveness (Defects found per hour)
Regular Use	0.210
Black Box	0.282
Glass Box	0.322
Reading/ Inspections	1.057

Effectiveness & Needed Resources

In another later experiment, Fagan found that ...,

82% of all errors eventually detected in another particular system were found by design and code inspection before unit testing.

He determined that 25% less programmer resources were needed than were estimated by a technique that had been tuned to the organization.

Cost Reduction -1

Crossman reported in 1982, that ...

Code inspections led to a 95% reduction in maintenance costs, that were predicted by an experience-tuned estimation procedure.

Cost Reduction -2a

Gilb & Graham report:

A production planning system consisted of 800 programs.

In the middle of development, inspection was stopped after 400 of them had been inspected, because people did not see the effect as a result of not keeping data.

Cost Reduction -2b

One year later, the maintenance cost of the programs was measured, and ...

Cost Reduction -2c

The maintenance cost of the 400 inspection-managed programs was between 0.6 and 0.7 minutes per line of code per year.

The maintenance cost of the 400 non-inspected programs was 7 minutes per line of code per year.

A 10 fold decrease with inspection!!

Cost Reduction -3

Gilb & Graham report:

A bank's software development department compared the maintenance costs of 88,000 lines of inspected COBOL code with those of 900,000 lines of non-inspected COBOL code:

The inspected code cost 28 times less to maintain than the non-inspected code.

Cost Reduction -4

Don O'Neill, in "National Software Quality Experiment: Results 1992-1995" *Proceedings of the 8th Annual Software Technology Conference*, 1996, reports that companies that use inspections have had a return on investment ranging from four to eight dollars saved for every dollar spent.

Time Reduction -1

Gilb & Graham report that typical net savings for project development using inspections are 35% to 50%.

In 1976, Fagan reported a 25% reduction of schedule plans from a predicted 62 programmer days to 46.5 programmer days including the inspection. (and if, as usual, estimates normally err on the optimistic side, this is even more amazing)

Time Reduction -2

In 1975, Rodney Larson reported that inspecting test plans, designs and cases saved 85% of the time normally needed do tests.

Defect Reduction -1

In 1980 IBM had the 11 development stages of a large, ½M line networked operating system inspected.

Their expectation at the time was for about 800 bugs in trial site operation.

They got only 8.

Defect Reduction -2

The 500K-line, inspected space shuttle software produced by IBM FSD had zero mission defects for the last 6 of the 9 1985 missions.

Moreover, this SW is tailored for the next mission in between missions.

Thus inspection helped eliminate ripple effects of maintenance changes.

Increased Productivity -1

Fagan’s original article reports a 23% increase in coding productivity as a result of inspection.

He later reports a 10% productivity increase with informal moderator training, a 25% increase with formal moderator training and design change control, and a 40% increase by adding code change control, inspection of tests and fixes, and test defect tracking.

Increased Productivity -2a

IBM FSD collects 14 pages of parameters after each project.

In a 1977 report by Waltston and Felix, 30 FSD projects using inspection were compared to about 30 similar FSD projects using the older structured walkthrough technique.

Increased Productivity -2b

The median net delivered lines of non-comment code productivity rate per work-month was:

- **300 for the 30 inspection-managed projects**
- **144 for the 30 structured-walkthrough-managed projects**

JPL Study -1

JPL is funded by NASA to conduct its unmanned interplanetary space program.

Inspections were introduced in 1987 to improve software quality by detecting errors as early in the development lifecycle as possible.

JPL Study -2

JPL basically followed the Faigin process:

- **inspections on all documents: software requirements, architectural design, detail design, source code, test plans, and test procedures**
- **a checklist tailored to JPL's needs was produced for each document type**

JPL Study -3

One mahor enhancement, a third hour to the inspection meeting, after the standard two-hour search for problems, to discuss possible solutions and clear up issues raised in the search meeting.

JPL Study -4

They also introduced data collection to gather data which is the basis for this study.

J.C. Kelly, J.S. Sherif, and J. Hops "An Analysis of Defect Densities Found During Software Inspections", *Journal of Systems and Software*, 17:111-117, 1992.

Data were collected on 203 inspections performed on 5 software-intensive projects between February, 1987 and April, 1990.

JPL Study -5

Nearly all team members were trained in a 1.5 day course.

Although projects used Ada, C, and Simula, on 16% of the inspections were performed on code.

JPL Study -6

1. **Increasing the number of pages inspected at once decreases the number of defects found.**

JPL Study -7

2. **Significantly more defects were found per page at the earlier phases of the lifecyle. The highest density was observed during requirements inspections. Fitting data shows:**

   $$\text{Defects/Page} = 3.19\, e^{-0.61t}$$

   where $t = 1, 2, 3, 4$ for SWR, AD, DD, SC respectively.

JPL Study -8

3. **The cost in staff hours to find and fix defects was consistently low across all types of inspections, on average 1.1 hours to find a defect and .5 hours to fix and check it, compared with 5-17 hours required to fix defects found during formal testing.**

JPL Study -9

4. **A variety of defects are found through inspections, with defects in clarity, logic, completeness, consistency, and functionality being the most prevalent.**

Inspection Meetings -1

One problem that a lot of places report is that of project delay due to difficulties scheduling the inspection meeting.

A. Porter, H. Siy, and L. Votta report in a *ICSE* ’97 paper that the inspection interval, i.e., the calendar time needed to complete the inspection, is significantly longer than the work time needed to complete the inspection, leading to significant delays in projects.

Inspection Meetings -2

In any busy place, it is difficult to find a common unscheduled two-hour slot for the inspection meeting.

Anywhere, Anytime Inspections -1

Perich, Perry, Porter, Votta and Wade in their *ICSE* ’97 paper suggest the use of the WWW to allow asynchronous inspection meetings that seem to generate enough synergy between the inspectors to match the effectiveness of inspection meetings.

Instead of having meetings, each inspector adds his comments to a Web page maintained on the company’s intranet.

Anywhere, Anytime Inspections -2

The synergy happens because each inspector gets to read previously added comments and to build on them. Each inspector is urged to visit the Web site more than once so that all inspectors get to read the comments of all others.

Anywhere, Anytime Inspections -3

Perich, Perry, Porter, Votta and Wade report that the cost savings from just the reduction in paper work and the time savings from the reduction in distribution interval of the inspection package (sometimes involving international mailings) have been substantial.

Anywhere, Anytime Inspections -4

Since there is no need to schedule a common meeting time, the asynchronous approach reduces the inspection interval significantly, leading to even more cost savings.

They also report that the new process is at least as effective in terms of defect detection effectiveness as synchronous inspection meetings.

Value of Inspection Meetings

Indeed, experiments by Johnson and Tjahjono reported in *ICSE* '97 failed to show that inspection meetings are more effective than having inspectors file their independently written private reports.

So with the cost reduction and inspection interval reduction coming from not having inspection meetings, many places are beginning to look at substitutes for inspection meetings.

Unmentioned in ALL Reports

What about the remaining 33%-15% errors not found by inspections?

The customers found the rest! sigh!

Programmers' Surprise

Gilb & Graham report:

One inspection user, with a very large systems programming project, had finished three weeks before the official deadline. They didn't believe it themselves. So, they spent two and a half weeks checking to see what they had missed. They found no defects (indeed there were virtually none, as it turned out) and handed [the program] over officially three days early.

Reeve's Rule of Thumb

As a rule of thumb, each major defect found at inspection will save nine-hours of downstream correction effort.

The reported ranged of average effort downstream to repair defects that could have been caught by inspection is from 4 (Gilb 1988) to 30 hours (Doolan 1992),

and the average effort downstream to repair defects caught by inspection is 9.3 hours (Reeve 1991)

Why Inspections Work

These are my feelings as to why they work!

- **Why is inspection so good?**
- **Inspection before compilation**
- **The power of water coolers**
- **Time estimation and testing**
- **Sources of inspection savings**
- **Summary of direct savings**
- **Indirect benefits to management**
- **Costs of inspection**

Why is Inspection So Good? -1

Why are inspections better than traditional testing?

- **People think.**
- **Test cases do not.**

People see implications of even supposedly good code.

People see reasons for errors that they find.

Why is Inspection So Good? -2

A test case exposes an error without exposing reasons; a person must then try to find the reasons and has to jump into the middle of things rather than having had the gentle introduction of having examined every line of the code.

Also, multiple heads are better than one!

Most test case examination is done by one person, privately in his or her office!

Why is Inspection So Good? -3

Tony Hoare put it nicely.

"It's much easier to find bugs in a line of reasoning than in a line of code."

Why is Inspection So Good? -4

Also examining code 4 times is bound to find more errors than 1 time.

No Inspection	Inspection
writing	writing
	walkthrough
	private reading
	public meeting

Inspection before Compilation -1

Some insist on inspections before submission to a compiler.

Why not just let the compiler find most errors?

Each syntax error can be a symptom of a serious semantic error.

The compiler finds the syntax error but not the semantic error.

Inspection before Compilation -2

Only people can find semantic errors.

If you fix syntax errors with the aid of a compiler without seriously considering semantic implications, you can end up totally overlooking a serious semantic error.

Ah, but why not just have the person running the compiler think?

Again, multiple heads are better than one.

Inspection before Compilation -3

Also, there is a strong temptation to rush through the compilation, reducing the effectiveness of the thinking.

The Power of Water Coolers -1

Did you ever have the experience?

You’ve been running test cases and have been trying to track down this one nasty bug for days.

You’re exhausted and are standing next to the water cooler kibbitzing with your fellow programmers and bitching about your stupid program.

The Power of Water Coolers -2

Someone says, “What’s the problem?”

You start explaining it, and bingo, eureka, you or the someone else sees the solution!

The Power of Water Coolers -3

There’s a story told of a manager that tried to put an end to everyone’s goofing off by the water cooler, by having the water cooler removed.

The manager saw the incidence of errors in delivered software go up!

The system staff started complaining about a sudden increase in a demand for assistance, an increase that they could not handle.

The Power of Water Coolers -4

An efficiency expert identified that the water cooler was *not* really for goofing off, but was an essential debugging tool!

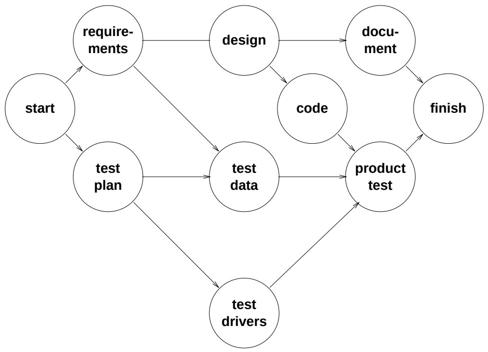
After the water cooler was brought back, things went back to normal.

Time Estimation and Testing

We now consider the problem of allocating sufficient time for testing in a project plan.

A Typical Pert Chart

A simple SW development Pert chart:



Criticality of Slippage -1

Look at Pert Chart!

What step is most critical to start on time to finish on time?

What step is most critical to estimate its duration accurately?

That is, for which step is a schedule slippage impossible to recover from?

Criticality of Slippage -2

product test!

Because it is one of the last two and the other one is not critical to the delivery on time.

Surprised?

Criticality of Slippage -3

Having reviews throughout the lifecycle reduces the risk of slippage inherent in having a single or very few testing phases, as is the case with having only unit and integration testing.

When inspections remove most of the defects, testing goes more smoothly, as there are fewer defects to find.

Sources of Inspection Savings -1

Savings comes from avoiding following costs:

- **downstream defect detection costs, i.e., module tests, integration tests, field tests, etc**
- **overtime costs during frantic last days/weeks before deadlines**
- **re-testing necessitated by defect fixes**

Sources of Inspection Savings -2

- **field service to deal with defects found by customers**
- **costs due to high turnover of overworked, burned-out professionals, i.e., recruitment, moving, training, etc.**
- **public relations and marketing costs to compensate for poor product quality**

Summary of Direct Savings

- 30%–100% net productivity increases
- 10%–30% net timescale reductions
- 5 to 10 times reduction in test execution costs and timescale
- order of magnitude reduction in maintenance costs
- nearly automatic improvement of SE quality and product quality
- early or on-time delivery of systems

Indirect Benefits to Management -1

Management gets what it inspects, not what it expects! (Gilb)

- It gets fuller data about defects and product quality and the progress of the development.
- It gets satisfaction of ISO 9000 (and other) quality standards.

Indirect Benefits to Management -2

- It is able to encourage developers to take more responsibility for the quality of their own work,

  partially because it will not have to put out as many fires with customers.

Costs of Inspection -1

Start up costs include costs of

- buy in
- training
- making checklists
- getting used to doing inspections

Costs of Inspection -2

Ongoing Inspection Costs:

- **Inspections add 10–15% to the costs of development.**
- **Besides the inspections themselves, people will end up spending more time preparing their documents for inspections, but that will pay off in the long run.**
- **Also time must be allocated to allow people to do inspections properly.**

Starting an Inspection Program

"OK, OK, you've convinced me! So *now* what do we do?"

Must start an inspection program!

- **Steps**
- **Checklists**
- **What to inspect**
- **Deal that cannot be refused**
- **Best thing about inspection**

Steps -1

1. **Selling and buy-in**

2. **Training**
   - **aimed at management, programmers, and inspectors**
   - **explaining each's role**
   - **explaining procedures**
   - **explaining checklists**
   - **supervised practice inspections of**
     - **instructor's product**
     - **each other's product**

Steps -2

3. **Recruit inspectors, people whose job definitions include all that is necessary to carry out inspections, including preparation for and attendance of meetings, code reading, etc.**

4. **Do it!**

5. **Inspect and evaluate the inspection process for improvement.**

6. **Improve it!**

Steps -3

Doing it involves

- **deciding what products (documents) are to be inspected,**
- **making checklists for these products, and**
- **making the 5 inspection steps part of the schedule for each product, together with supporting memos, signoffs, and whatever else the organization requires for any other milestone.**

Steps -4

Don O'Neill, in "Issues in Software Inspection" *IEEE SW* January, 1997, reports that once you adopt software inspection and complete training, your organization can detect 50% of the defects present. To achieve expert practice, a detection rate of 60 to 90 percent, can take from 12 to 18 months. After 10 years of practice, IBM reported an 83% and AT&T reported a 92% defect detection rate.

Checklists -1

For example, if code is to be checked, the checklist can include items such as

- **compliance to standards**
- **domain correctness**
- **consistency with requirement items**
- **language usage**
- **use-declaration type consistency**

Checklists -2

- **invocation interface consistency**
  - **formal-actual correspondence**
  - **return value**
- **use of correct dimensions (e.g. °C vs °F)**
- **comment-code consistency**
- **avoidance of errors in historical list (frequency ranked)**

Checklists -3

The Ebenau and Strauss book is chock full of good checklists for a wide variety of documents.

The Gilb and Graham book have some too.

You will no doubt find others.

Checklists are the Key -1

It has become clear in retrospect that the checklists are the key!

An inspection is as good as its checklist, whether explicit or implicit (i.e., in the inspector's mind).

Sometimes, how powerful inspections can be becomes apparent only after seeing the checklists.

Checklists are the Key -2

Thus, it is critical to turn these implicit checklists into explicit.

It's sort of like codifying expert knowledge from a human expert for an expert system.

To produce a checklist for inspecting modified code for ripple effects, I thought about how I desk check for this problem and wrote down the danger signs I look for and the questions I ask as I find all uses of any identifier participating in modified code.

Checklists are the Key -3

But then the question naturally comes up!

Why bother with inspections?

Why not just give the programmers the checklists and have them inspect the code themselves?

Checklists are the Key -4

It's the same as proofreading a book or article.

Most authors know all the grammar and language rules.

Most authors know what they are trying to say.

But in all the busy worrying about content and everything else that goes into making good writing, authors make mistakes.

Checklists are the Key -5

When it comes to proofreading, the worst person to do it is the author.

He or she sees what he or she has learned to expect and not what is actually there.

Only another person can do an adequate job of proofreading.

So inspection by another party is necessary.

Checklists are the Key -6

However, the best procedure is for the checklist to be known by all.

The developers strive their damndest, through discipline and desk checking, to give products to inspection for which the inspection fails (i.e., the inspectors find *no* errors!).

The inspectors strive their damndest to succeed.

Checklists are the Key -7

Sometimes the developers win, sometimes the inspectors win, and at all times, the team wins!

Inspections work best when everyone is trying to make them unnecessary

Unfortunately, the real threat of inspection appears to be necessary to make people work like this.

Checklist Effectiveness -1

Which is better?

- **Inspectors have no checklist.**
- **Inspectors have checklist.**

With a checklist one is less likely to overlook an item on the list, but with a checklist, one might overlook something not on the list in focusing on the list.

Checklist Effectiveness -2

Which is better?

- **Each inspector has responsibility for specific classes of faults.**
- **All inspectors search for whatever faults they can find.**

With specific domains covered, faults in those domains are less likely to be missed, but with specific domains covered, faults not in any anticipated domain are more likely to be missed.

Checklist Effectiveness -3

The three main approaches that are use are combinations:

1. **Ad Hoc: no checklist and all inspectors search for whatever they can find.**
2. **Checklist: checklist and all inspectors search for whatever they can find.**
3. **Scenario: checklist and each inspector has responsibility for specific classes of faults.**

Checklist Effectiveness -4

A study by A. Porter and L. Votta has shown

1. **The fault detection rate when using Scenarios was superior to that obtained using Ad Hoc or Checklist methods, by roughly 35%**
2. **Scenarios helped reviewers focus on specific fault classes and, in comparison to Ad Hoc or Checklist methods, did not compromise their ability to detect other classes of faults.**

Checklist Effectiveness -5

3. **The Checklist method, the industry standard, was no more effective than the Ad Hoc method.**

4. **On the average, collection (inspection) meetings contributed nothing to fault detection effectiveness.**

What to Inspect -1

In principle, any product can be inspected, *even* an inspection plan-and-checklist itself; all you need is a checklist of what to look for based on some definition of quality or correctness of the product.

What to Inspect -2

Any product that can give you trouble can be subjected to inspection.

For example, if you know that you have trouble correcting bugs without introducing new bugs, so initiate inspections of bug correction strategies, i.e., of the modifications to the whole program that purport to fix a bug, and get multiple minds helping to find ripple effects.

But *how* do you inspect *that* ? -1

Sometimes people have difficulty seeing how to inspect a deliverable that does not represent a complete solution, but is just, say a preliminary design.

I claim that *any* deliverable product is inspectable relative to the very criteria by which it is decided whether or not to deliver it at this time.

But *how* do you inspect *that* ? -2

If the author of the product can identify a point at which he or she knows the product is ready to be seen by others and before that point it was not, then the unspoken criteria, the basis for his or her pride upon delivery of a good product, are the criteria by which the product is to be inspected.

But *how* do you inspect *that* ? -3

When I produce a preliminary design, even though it is purposely incomplete, there is a certain point at which it is complete enough to serve its purpose as a preliminary design. Whether the preliminary design meets its purpose can be judged by others and is thus inspectable.

The checklist is a verbalization of these unspoken criteria, of whether the product meets its purpose.

Deal that Cannot Be Refused -1

Gilb & Graham tell the story of one project manager that pulled a neat trick to get people to buy into inspection and to defuse potential arguments that people were not given enough time to do proper inspections.

He simply declared all deadlines extended by 15% for any project that agreed to do inspections.

Deal that Cannot Be Refused -2

15% was simply the historical gross cost at IBM of doing full inspections.

He ended up winning more than he gave away, as they used only 4–5% additional time!

Best Thing About Inspection -1

The best thing about instigating an inspection program, as a means to improve software quality and productivity is that:

It can be introduced without having to change anything else you're doing.

You can piggyback it on your current process and methods.

Best Thing About Inspection -2

You will notice an immediate improvement anyway.

Furthermore, as you build up a friendly competition to deliver for inspection only products that cause the inspection team to fail, you will begin to notice other ways to improve your process and methods.

Only, now the motivation to try new processes and methods will come from within.

Evaluating Program's Effectiveness

There are two ways to evaluate effectiveness of your inspection program:

- **feelings**
- **data**

Feelings

Programmers *know* when they are doing better!

After all, students *know* how they've done on their bagrut exams, baccalaureate exams, Regent's exams, SATs, GREs, etc. even before they get the results, in fact even before they leave the room.

But feelings can be deceiving, so ... get DATA!!!

Data -1

Since the purpose of inspections is to spend a little extra development resources to significantly reduce defects and total lifecycle resources, the important data to collect for each project are:

- **resource expenditures**
- **defect reports (but *only* after inspection!), particularly after delivery**

Data -2

A primary concern is to show pay-off, i.e., ROI, and to show it as soon as possible.

Most organizations do keep these items of data even if they are not collecting them to track inspections, just for budgeting and accountability purposes.

So the organization must start collecting these data *now*, and it must try to dig these data up for past projects.

Data -3

If it can find the data for the past projects, then it will be able to demonstrate effectiveness sooner.

Conclusion -1

We covered:

- **Sources of information**
- **Background**
- **Bugs come early and stay**
- **Finding bugs with reviews**
- **Psychological impediments**

Conclusion -2

- **Procedures for doing reviews**
- **Experimental evidence**
- **Why inspections work**
- **Starting an inspection program**
- **Evaluating effectiveness of program**