Reviews

Review What?

Review Everything

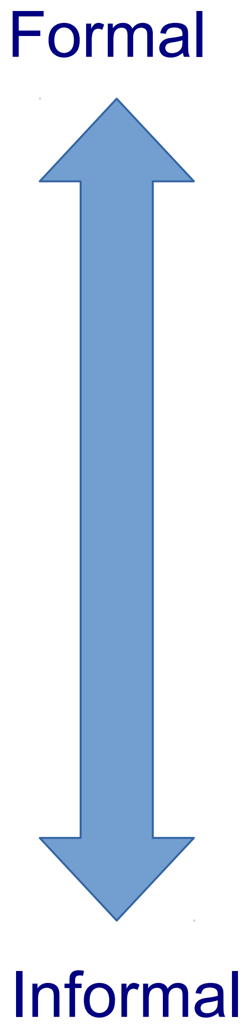
- Vision & Scope Document
- Requirements Specification
- Project Plan
- Design - High Level and Detailed Design
- Code
- Test Plan
- Documentation

Why Review?

1. Save time.
2. Save money.
3. Gain approval & increase sense of shared ownership.
4. Reviews find more defects than testing.
5. Share knowledge.
6. More ideas make better work products.

Kinds of Reviews

1. Inspection
2. Code Review - Inspection of Code
3. Walk-through
4. Desk check
5. Self-review



Which Review to Use?

Product	Technical Drivers - Complexity		
	Low	Average	High
Software Requirements	Walkthrough	Inspection	Inspection
Design	Desk check	Walkthrough	Inspection
Software Code and Unit Test	Desk check	Walkthrough	Inspection
Qualification Test	Desk check	Walkthrough	Inspection
User/Operator Manuals	Desk check	Desk check	Walkthrough
Support Manuals	Desk check	Desk check	Walkthrough
Software Documents, e.g. Version Description Document (VDD), Software Product Specification (SPS), Software Version Description (SVD)	Desk check	Walkthrough	Walkthrough
Planning Documents	Walkthrough	Walkthrough	Inspection
Process Documents	Desk check	Walkthrough	Inspection

Source: Prof. Claude Laporte, *U. of Quebec*, Dept of Software and IT Engineering

Inspection

The most formal kind of review.

Purpose: find defects.

How To:

1. Choose work product to inspect.
2. Choose 4-5 people, including a moderator
3. Prepare: Everyone reads the work product in advance and notes suspected defects.
4. Inspection meeting: confirm defects & log them

   Inspections may proposal correction (e.g. words in document)
5. Rework: author fixes defects from inspection log

Inspection Team

Author of document or work product

Project manager - for project documents

Representative of groups affected by the document, e.g. developers, management,

Inspectors should

- be familiar enough with project to understand problems and propose changes
- provide different perspectives on work product

Inspection Meeting

Moderator guides inspectors through work product.

Ask inspectors for defects.

Other inspectors (and author) confirm each defect, or explain why they disagree.

Inspectors agree on a fix (for document) or leave it to author to fix (code).

Record each defect in a written log.

Purpose is *not* for author to teach or explain.

After the Inspection

Rework: author fixes the work product.

Follow-up: inspectors individually review the revised work and approve or not approve it.

Acceptance: once all inspectors approve, the work product is accepted.

Inspection is NOT...

- Review of style
- Attempt to improve or optimize design*
- Evaluation of the author
- Subjective evaluation of quality

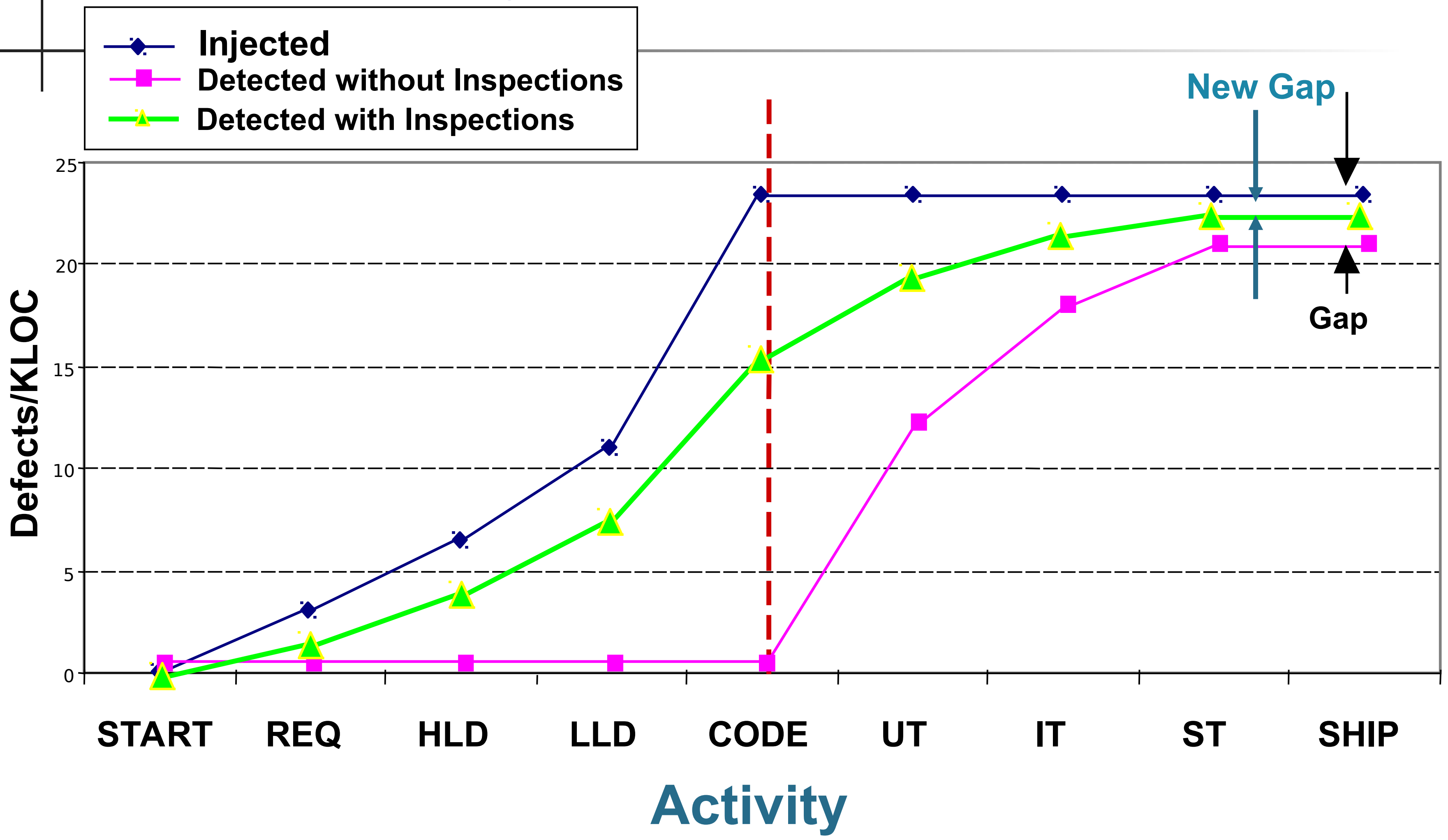
**Infosys: inspectors use a separate form to record comments, offer insights and ideas.*

Is Inspection Worth the Time?

Inspection involving 5 people takes 10-20 man-hours, about half the time is preparation.

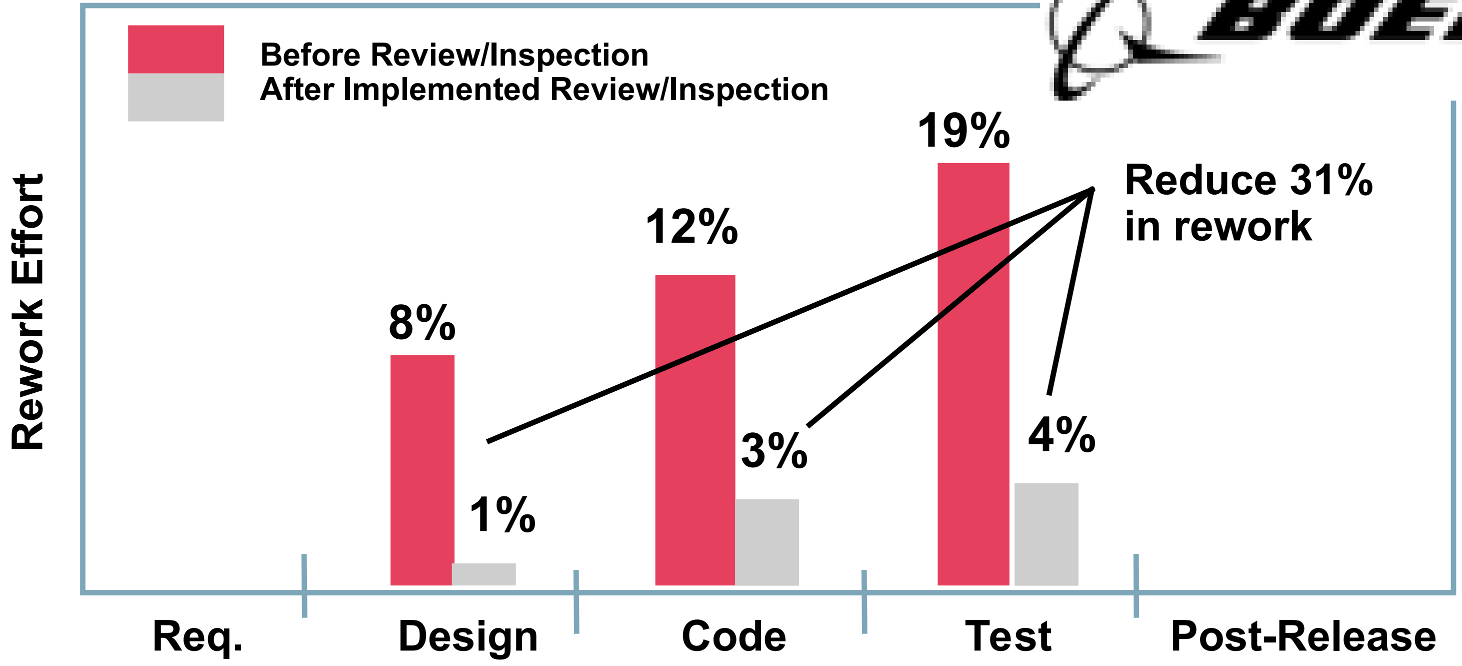
Inspection finds 5 - 10 defects, on average.

Defect Injection and Detection



Source: Ron Radice, 'Software Inspections: Past, Present, and Future.', Software Technology Conference, Salt Lake City, Utah, May, 2001

Cost/Benefit of Inspections



Formal Review/Inspection increased design effort by 4%

Decreased rework effort by 31%

Source: Vu, J., 'Software Process Improvement Journey', 8th Software Engineering Process Group Conference San Jose, California March, 1997.

Conclusion

Inspection Saves Time & Money

How Much Time Does it Take?

Inspection Type	Checking Rate	Logging Rate
Architecture	2 – 3 Pages Per Hr (PPH)	2 - 3 PPH
Requirements	2 - 3 PPH	2 - 3 PPH
Preliminary Design	3 – 4 PPH	3 – 4 PPH
Detailed Design	3 – 4 PPH	3 – 4 PPH
Source Code	100 – 200 LOC Per Hour (LPH)	100 – 200 LPH
Test Plan	5 – 7 PPH	5 – 7 PPH
Fixes and Changes	50 – 75 LPH	50 – 75 LPH
User Documentation	8 – 20 PPH	8 – 20 PPH

Source: Radice, '*High Quality Low Cost Software Inspections*', 2002.

Code Review

An inspection of code.

Similar to Inspection but more time is spent on **alternatives** and **qualitative** issues.

Before Review:

- choose the code to review (*you can't review everything this way) - see Stellman & Greene*

- choose moderator, reader, and inspectors

- choose a date/time and duration (60-90 minutes)

- everyone reviews code individually and makes notes of issues they find (paper or online notes)

Code Review Meeting

During Review:

- the "reader" walks through the code aloud -- by section (class, method, code block), not literally reading code.

- inspectors:
  - ask about anything they don't understand
  - question correctness of code
  - suggest "better" or more self-explanatory alternatives

- moderator: keep review on track. Don't get bogged down discussing particular design or code issues.

- recorder: writes down issues for follow-up

Code Review Follow-up

After Review:

- author addresses <u>all</u> issues, either revise code or explain to reviewer why he things no rework is needed
- do it promptly!
- gain agreement to close all issues

Code Review vs. Other Inspections

Code Reviews result in more open issues.

May *refactor the code* during the meeting... if it makes it easier to review.

More time spent proposing alternatives or improvements.

Follow-up and consensus can be done online.

Walk-through

Goals:

- find defects
- solicit feedback and ideas, other perspectives
- discover alternative solutions
- gain shared understanding of artifacts
- improve everyone's knowledge & skill

Author "walks" a small group through a work product.

More informal than code review, and led by author.

Procedure is more flexible than inspections.

Often applied to: code, use cases, software design

Desk Check

Purpose:

Individually review of code by another developer.

Usually done individually, with follow up discussion.

Procedure:

A developer asks another developer to review his work.

The reviewer (at his own desk) checks the work and reports defects, questions, and suggestions for alternatives or improvement.

Git Pull Request

Purpose:

Request review of work before incorporating it into a main dev branch or (if you dare!) master branch.

A kind of "desk check" using Github or Bitbucket.

Tutorial: https://yangsu.github.io/pull-request-tutorial/

Guide: https://help.github.com/articles/using-pull-requests/

Example (JQuery):

https://github.com/jquery/jquery/pull/1051#discussion-diff-2287441

Self-Review

Always review your own work

Obvious, but often not done!

How to:

- take a break before review. This is required.
- decide what criteria you are going to use (what are you checking for?)
- allocate sufficient time
- record DEFECTS you find

Scripts and Checklists

*Scripts and Checklists save **time** & make results more **consistent**.*

How Save time?

- don't re-discover what you did before

- focus on the creative, not the routine

Script - describe the activity, its purpose, desired result, important steps, and "exit criteria".

Checklist - concise list of particular things to do or inspect

Script

Purpose:	Find defects in code
Entry criteria:	Code specification and design
	Source code with tests that all pass.
	Goal for Code Review: review for what?
	Checklist
Steps:	1.
	2.
	3.
Exit criteria:	source code completely reviewed.
	all defects and open issues recorded

PSP Code Review Script

Purpose	To guide you in reviewing programs
Entry Criteria	- A completed and reviewed program design - Source program listing - Code Review checklist - Coding standard - Defect Type standard - Time and Defect Recording logs
General	Do the code review with a source-code listing; do not review on the screen!

Step	**Activities**	**Description**
1	Review	- Follow the Code Review checklist. - Review the entire program for each checklist category; do not try to review for more than one category at a time! - Check off each item as it is completed. - For multiple procedures or programs, complete a separate checklist for each.
2	Correct	- Correct all defects. - If the correction cannot be completed, abort the review and return to the prior process phase. - To facilitate defect analysis, record all of the data specified in the Defect Recording log instructions for every defect.
3	Check	- Check each defect fix for correctness. - Re-review all design changes. - Record any fix defects as new defects and, where you know the number of the defect with the incorrect fix, enter it in the fix defect space.

Exit Criteria	- A fully reviewed source program - One or more Code Review checklists for every program reviewed - All identified defects fixed - Completed Time and Defect Recording logs

Checklist

Reviews <u>should</u> use a checklist.

Contents of checklist depend on kind of thing being inspected!

Self-review and desk check are **more effective** if you use a checklist.

Example Checklist for Requirements Specification (RS)

- **RS 1 (TESTABLE) –** All requirements are verifiable (objectively)
- **RS 2 (TRACEABLE)** – All requirements must be traceable to a systems specification, contractual/proposal clause.
- **RS 3 (UNIQUE) –** Requirements must be stated only once
- **RS 4 (ELEMENTARY) –** Requirements must be broken into their most elementary form
- **RS 5 (HIGH LEVEL) –** Requirement must be stated in terms of final need, not perceived means (solutions)
- **RS 6 (QUALITY)** – Quality attributes have been defined.
- **RS 7 (HARDWARE)** – Is hardware environment is completely defined (if applicable).
- **RS 8 (SOLID) –** Requirements are a solid base for design

Source: Gilb, T., Graham, D., 'Software Inspection', Addison Wesley, 1993.

Example Checklist for C++ Code (CC)

- **CC1 (COMPLETE)** - Verify that the code covers all the design.
- **CC2 (INCLUDES)** - Verify that includes are complete.
- **CC3 (INITIALIZATION)** - Check variable and parameter initialization.
- **CC4 (CALLS)** - Check function call formats
- **CC5 (NAMES)** - Check name spelling and use
- **CC6 (STRINGS)** Check that all strings are ...
- **CC7 (POINTERS)** - Check that:
  - Pointers are initialized to NULL,
  - Pointers are deleted only after new, and
  - New pointers always deleted after use.
- **CC8 (OUTPUT FORMAT)** - Check the output format:
  - Line stepping is proper.
  - Spacing is proper.
- **CC9 (PAIRS)** - Ensure the { } are proper and matched.
- **CC10 (LOGIC OPERATORS)** - Verify that the proper use of ==, =, //, and so on.

Adapted from: Humphrey, W., ‘Introduction to the Personal Software Process’, Addison Wesley, 1997.

Another Code Inspection checklist

Fault class	**Inspection check**
Data faults	Are all program variables initialised before their values are used? Have all constants been named? Should the lower bound of arrays be 0, 1, or something else? Should the upper bound of arrays be equal to the size of the array or Size -1? If character strings are used, is a delimiter explicitly assigned?
Control faults	For each conditional statement, is the condition correct? Is each loop certain to terminate? Are compound statements correctly bracketed? In case statements, are all possible cases accounted for?
Input/output faults	Are all input variables used? Are all output variables assigned a value before they are output?
Interface faults	Do all function and procedure calls have the correct number of parameters? Do formal and actual parameter types match? Are the parameters in the right order? If components access shared memory, do they have the same model of the shared memory structure?
Storage management faults	If a linked structure is modified, have all links been correctly reassigned? If dynamic storage is used, has space been allocated correctly? Is space explicitly de-allocated after it is no longer required?
Exception management faults	Have all possible error conditions been taken into account?

PSP Checklist

This is worth studying.
He divides items into categories.

Humphrey's advise:

1. Keep your checklist simple and short.
2. Checklist must be complete.
3. Tailor to the programming languages you use.
4. Designed to address the kind of defects you inject.

Example Checklist for Java

Defect Type	Description
variable name	are names descriptive? correct case?
comments	Descriptive Javadoc method comments? In method: is complex logic explained?
exception handling	Are all reasonable exceptions caught and handled, or explicitly allowed to be thrown?
logging	Are security or unusual events being logged?
null pointers	Are any possible null values used? (Does NullObject pattern apply?)

Another Code Review Checklist

Applied Software Project Management, page 90.

1. parts of their list are too broad or vague (my opinion).

2. some items are outside the usual scope of Inspection.

For example:

Efficiency

Reusability (this can be a waste of time)

Summary

1. Review Everything - not just code

2. Choose an appropriate level of review

3. Reviews must produce a written result - not just talk
   - result is online where everyone can see it
   - open issues for specific items

4. Follow up & close all issues, answer all questions

5. Use tools to automate routine stuff (style checking ...)

6. Scripts and checklists make reviews more effective

Questions

1. Look at the PSP Code Review Checklist.

   what categories do not apply to Python?

   what categories can be done by automated tools?

References

Stellman & Greene, *Applied Software Project Management*, chapter 5 on Reviews.

- chapter 5 is available online.

Ship It! Item 13 - *Review all Code*

Practical advise for code reviews.

- *review only a small amount of code*
- *one or two reviewers at most*
- *review very frequently, often several times per day*

References

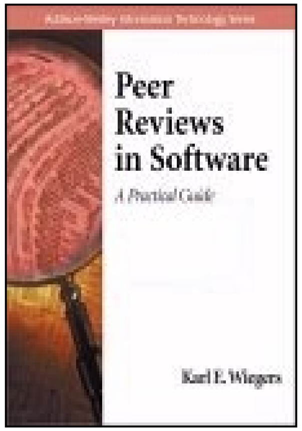
Karl Weigers, *Peer Reviews in Software - A Practical Guide*. Considered the "bible" on peer reviews.

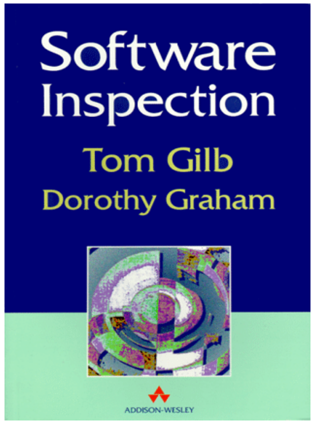
Karl Weigers, *Improving Quality Through Software Inspections* article online.

https://www.processimpact.com/articles/inspects.pdf

Tom Gilb, *Software Inspection*.

R. Radice, *High Quality Low Cost Software Inspections*.





Acknowlegement

Some of these slides are from a workshop at NECTEC by Prof. Claude Leporte, U. of Quebec.