How to Improve the Result of Reviews and Inspections

Niels Malotaux

niels@malotaux.eu
www.malotaux.eu/?id=conferences

Niels Malotaux

Result Management

- Independent Team, Project, Organizational Coach
- Expert in helping optimizing performance

- Helping projects and organizations very quickly to become
  - More effective – doing the right things better
  - More efficient – doing the right things better in less time
  - Predictable – delivering as predicted

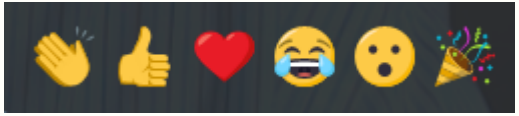
- Project rescue

- Sometimes actually developing an electronic product (hardware, firmware)

Who is doing what ? (use Chat to answer)

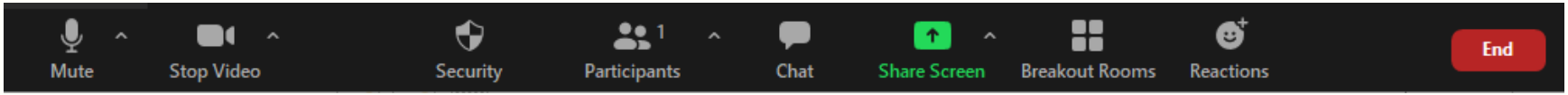
1. Tester ?
2. QA ?
3. Developer ?
4. Systems Engineer ?
5. Architect ?
6. Project Manager ?
7. Product Owner ?
8. Scrum Master ?
9. Customer ?
10. Manager ?
11. Consultant ?
12. Coach ?

Just in case …



yes no

(stays on for about 10 sec)



use Spacebar to unmute shortly

use Chat for answers (English please !)

Schedule

10:00-11:30	1:30
break	0:10
11:40-13:00	1:20
lunch	1:00
14:00-15:20	1:20
break	0:10
15:30-17:00	1:30

For me that is ☺

9:00-10:30
break
10:40-12:00
lunch
13:00-14:20
break
14:30-16:00

The top level requirement for our work

Quality on Time

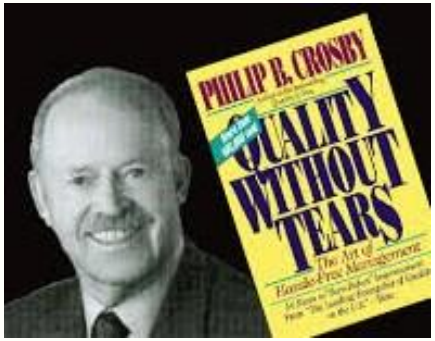
- Delivering the Right Result at the Right Time, wasting as little time as possible

- Providing the customer with
  - what they need
  - at the time they need it
  - to be satisfied
  - to be more successful than without it
- Constrained by (win - win)
  - what the customer can afford
  - what we mutually beneficially and satisfactorily can deliver
  - in a reasonable period of time

Is there a Quality On Time problem ?

- What made you decide to attend ?
- Do your projects produce the Right Results ?
- Do your projects deliver the Right Results at the Right Time ?
- What could we do about it ?
- Can Reviews and Inspections help delivering better quality in less time?
- What is
  - Better quality ?
  - On Time ?

Crosby (1926-2001) - Absolutes of Quality



- Conformance to requirements
- Obtained through prevention
- Performance standard is zero defects
- Measured by the price of non-conformance (PONC)

Philip Crosby, 1970

- The purpose is customer success (not customer satisfaction)

Added by Philip Crosby Associates, 2004

- Providing the customer with
  - what they need
  - at the time they need it
  - to be satisfied
  - to be more successful than without it
- Constrained by (win - win)
  - what the customer can afford
  - what we mutually beneficially and satisfactorily can deliver
  - in a reasonable period of time

Conformance to requirements

- We meet the agreed requirements

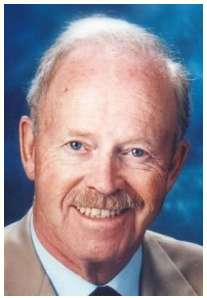
or

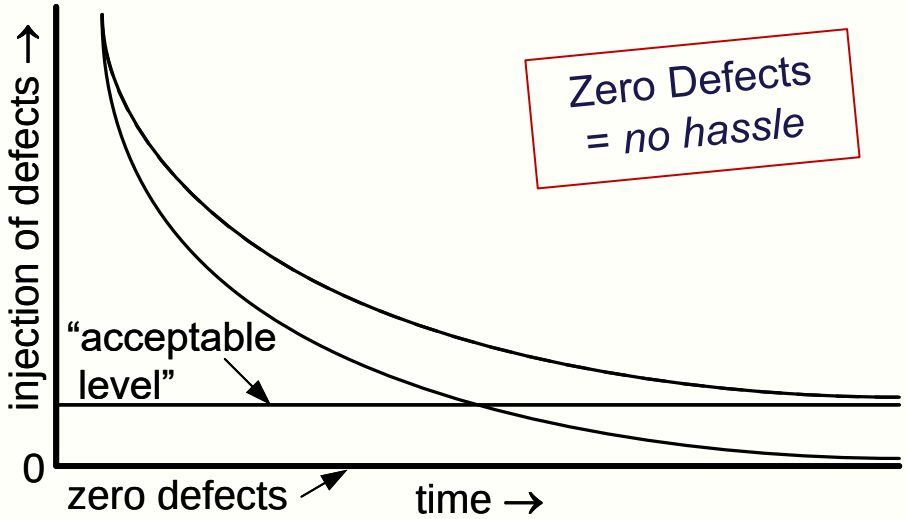
- Have the requirements changed to *what we and the customer really need*
- We create requirements with care and we meet them with care

Philip Crosby

Is Zero Defects possible ?

- Zero Defects is an asymptote





- When Philip Crosby started with Zero Defects in 1961, errors dropped by 40% almost immediately
- AQL > Zero means that the organization has settled on a level of *incompetence*
- Causing a hassle other people have to live with

Hassle: problem, unnecessary difficulty

Attitude

How to move towards Zero Defects
www.malotaux.eu/?id=conferences
(video also with Russian translation)

- As long as we think Zero Defects is impossible, we will keep producing defects

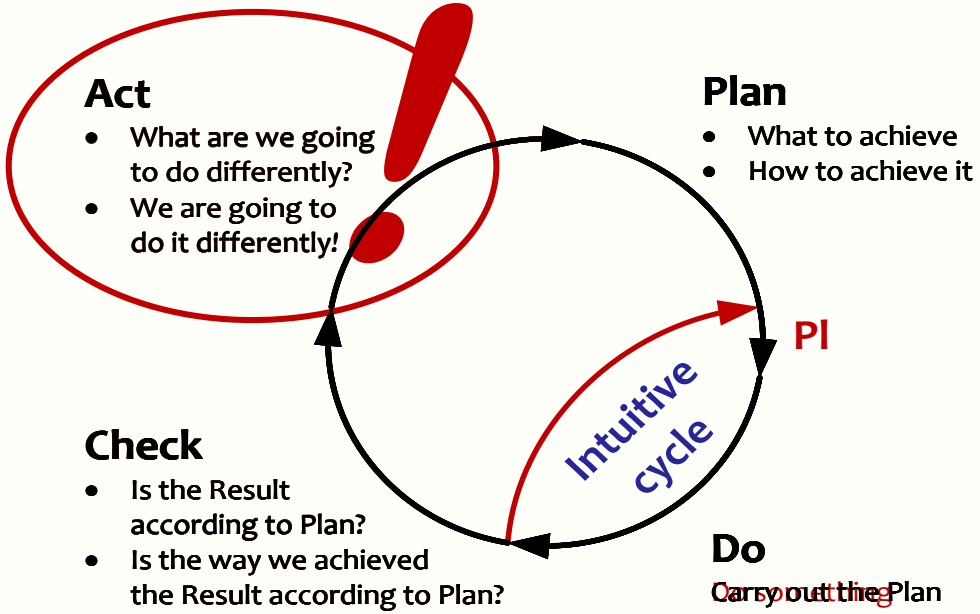
- From now on, we don't want to make mistakes any more
- We feel the failure (if we don't feel failure, we don't learn)
- If we deliver a result, we are sure it is OK
  and we'll be highly surprised when there proves to be a defect after all
- We do what we can to improve (continuous improvement)

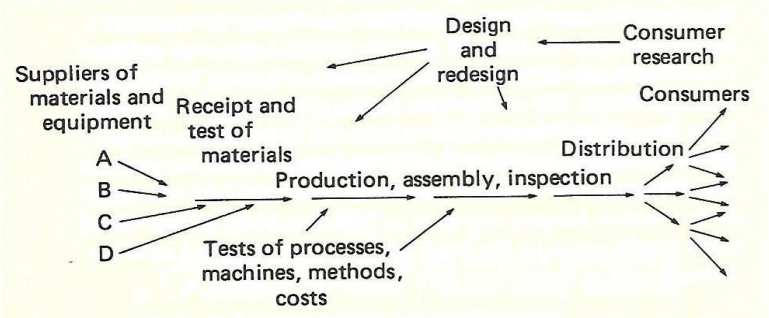
What is a defect ?

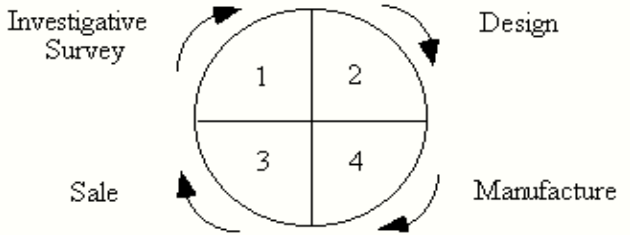
- A defect is the cause of a problem experienced by the users
- Making the customer more successful implies *no defects*
- *Mantra: "What we deliver simply works"*
- Are we delivering results without defects ?

The essential ingredient: the PDCA Cycle
(Shewhart Cycle - Deming Cycle - Plan-Do-Study-Act Cycle - Kaizen - Continuous improvement)

Deming (1900-1993)



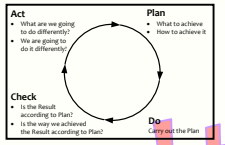




Deming talking to Japanese Top Management in 1950

Evolutionary Project Management elements (Evo)

(Tom Gilb)



- Plan-Do-Check-Act
  - The powerful ingredient for success
- Business Case
  - *Why* we are going to improve *what*
- Requirements Engineering
  - What we are going to improve and *what not*
  - How much we will improve: *quantification*
- Architecture and Design
  - Selecting the *optimum compromise for the conflicting requirements*
- Early Review & Inspection
  - Measuring quality while doing, learning to prevent doing the wrong things

Why

What
How much
Are we done

How

Check as early as possible

Zero Defects Attitude

Right Result

Quality On Time

Right Time

- Weekly TaskCycle
  - Short term planning
  - Optimizing estimation
  - Promising what we can achieve
  - Living up to our promises
- Bi-weekly DeliveryCycle
  - Optimizing the requirements and checking the assumptions
  - Soliciting feedback by delivering Real Results to *eagerly waiting Stakeholders*
- TimeLine
  - Getting and keeping control of Time: Predicting the future
  - Feeding program/portfolio/resource management

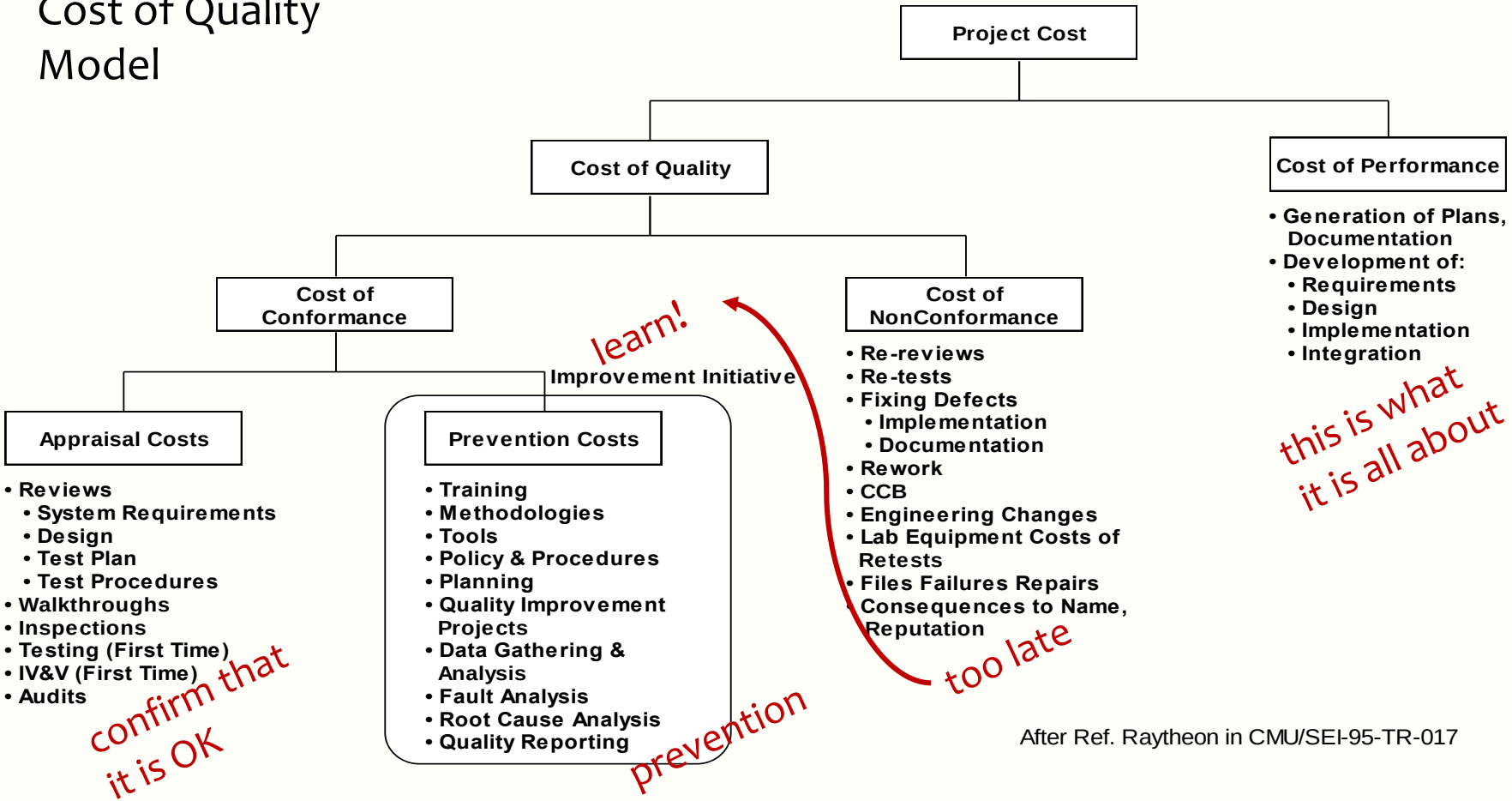
Evo Project Planning - Niels

Efficiency of what we do

Effectiveness of what we do

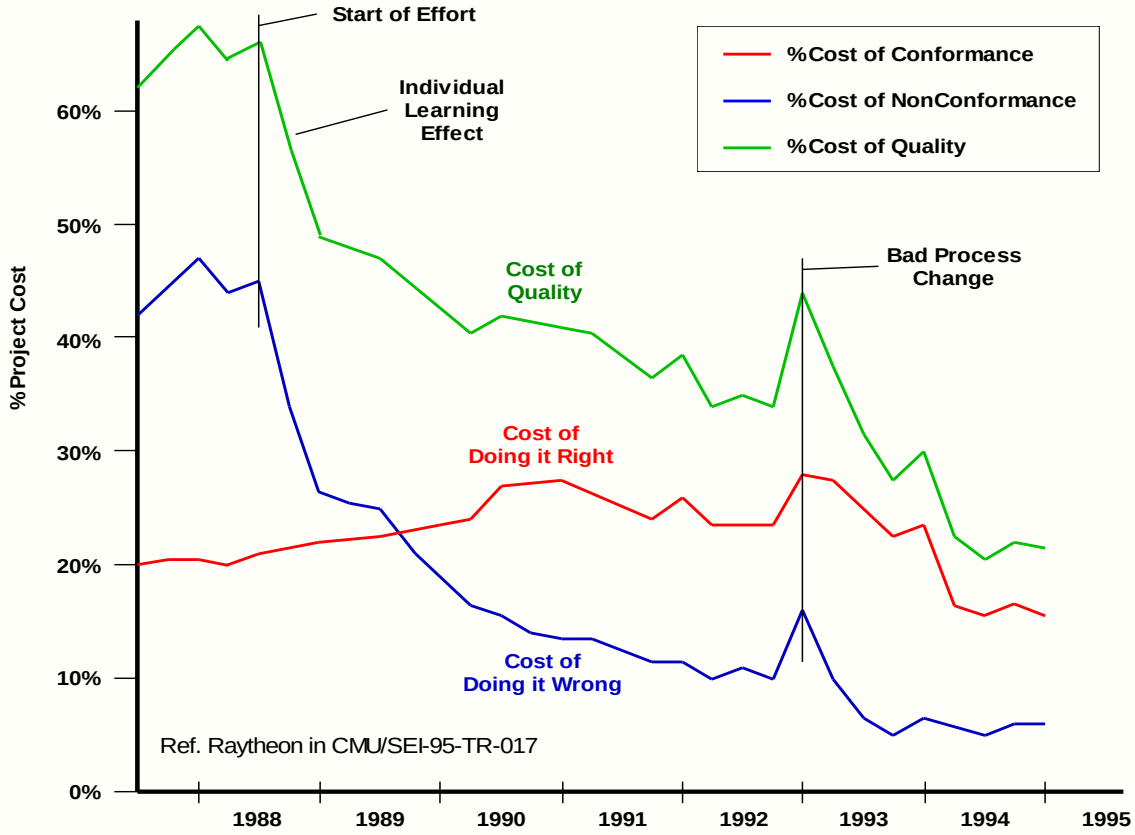
What will happen, and what will we do about it?

Cost of Quality Model

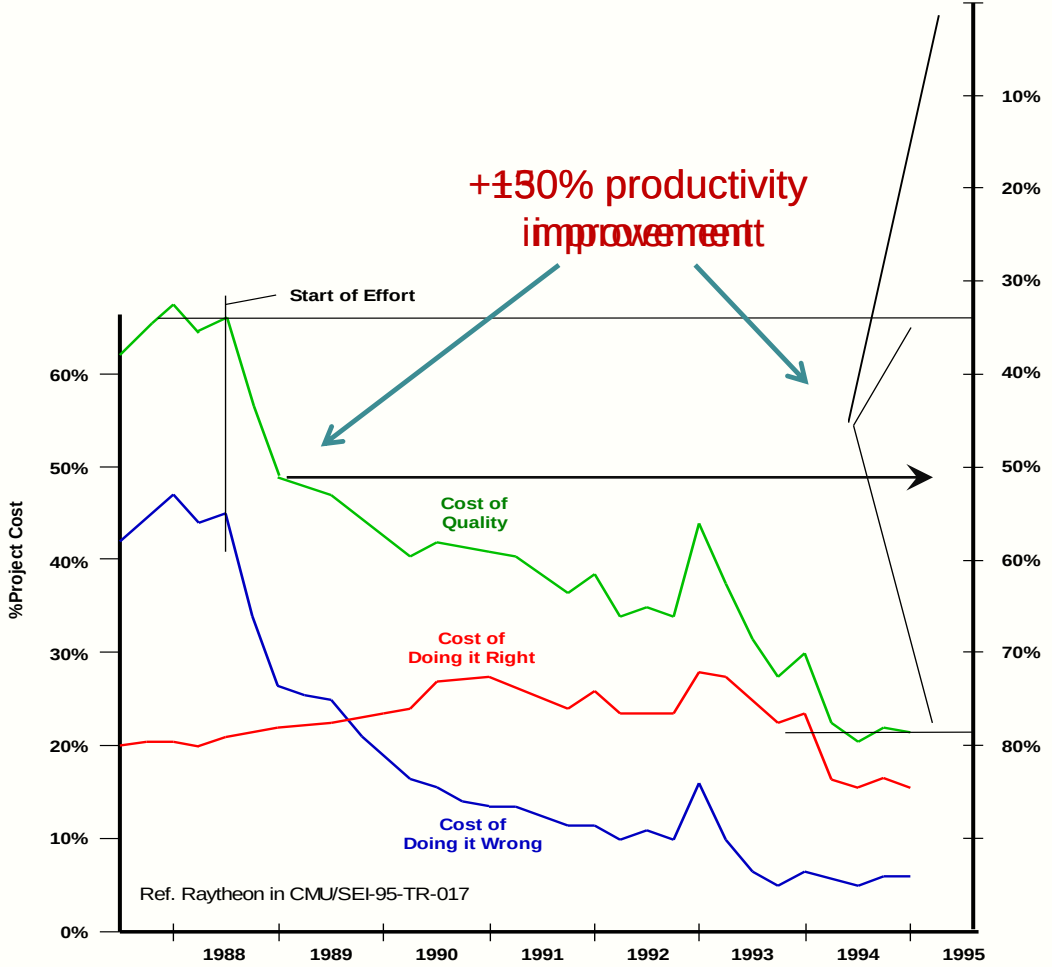


After Ref. Raytheon in CMU/SEI-95-TR-017

Cost of Quality (introducing inspections)



How much productivity gain?



Could deliver 2.3 x as much in the same time

Quality On Time

The most effective way of improving software productivity and shortening project schedules is to *reduce defect levels*

Capers Jones

Both *Quality* and *On Time* is improved if we work on reducing defect levels

Why are testers so obsessed to find defects, where we should have *no* defects

Better quality costs less

Are all of your documents always reviewed ?

- If your product is tested, how do you know it's correct ?
  (testing hardly *proves* anything)
- Reviews are for
  - early detection
  - quick learning
  - *prevention*
- Without proper education reviews are not very effective
- Inspections are a special kind of review

The process of defect injection

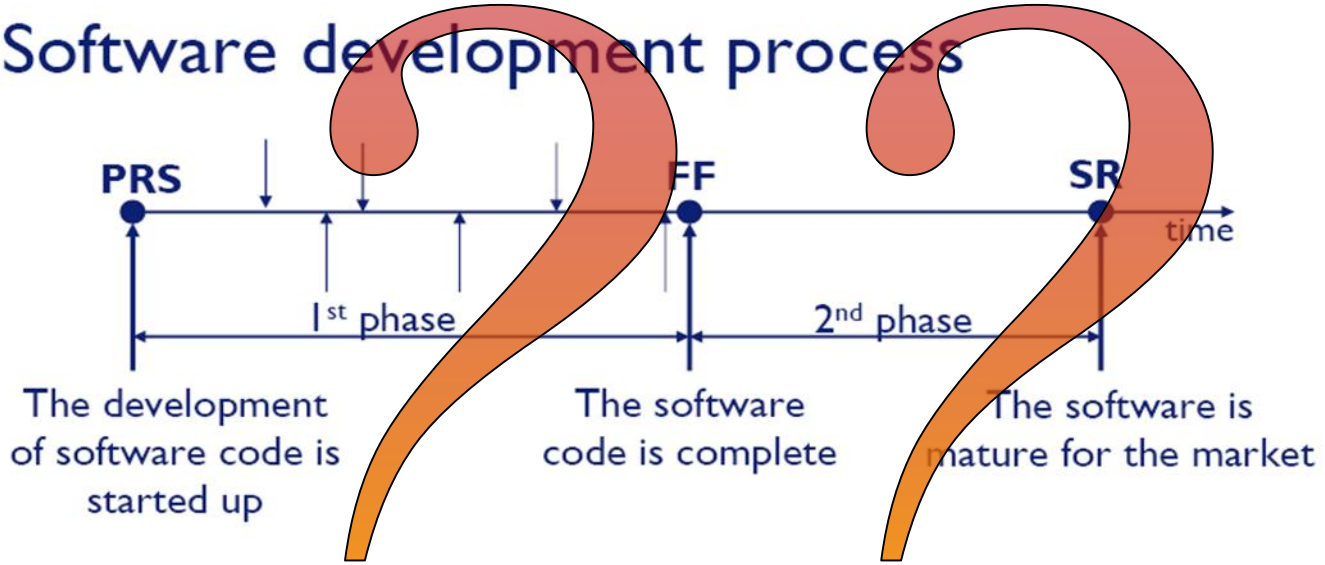
Conventional software development:

1. Development phase: inject bugs
2. Debugging or Testing phase: find bugs and fix bugs

Are we doing better ?

Does anybody mind ?

Software development process



- 1st phase is developing phase
- 2nd phase is de-bugging phase

Let's do a lot of testing !

Dijkstra (1972):

It is a usual technique to make a program and then to test it

however:

Program testing can be a very effective way to show the presence of bugs but it is hopelessly inadequate for showing their absence

- Conventional testing:
  - Pursuing the very effective way to show the *presence* of bugs
- The challenge is, however:
  - Making sure that there are no defects
  - And how to show their *absence* if they're not there

Bugs or Defects ?

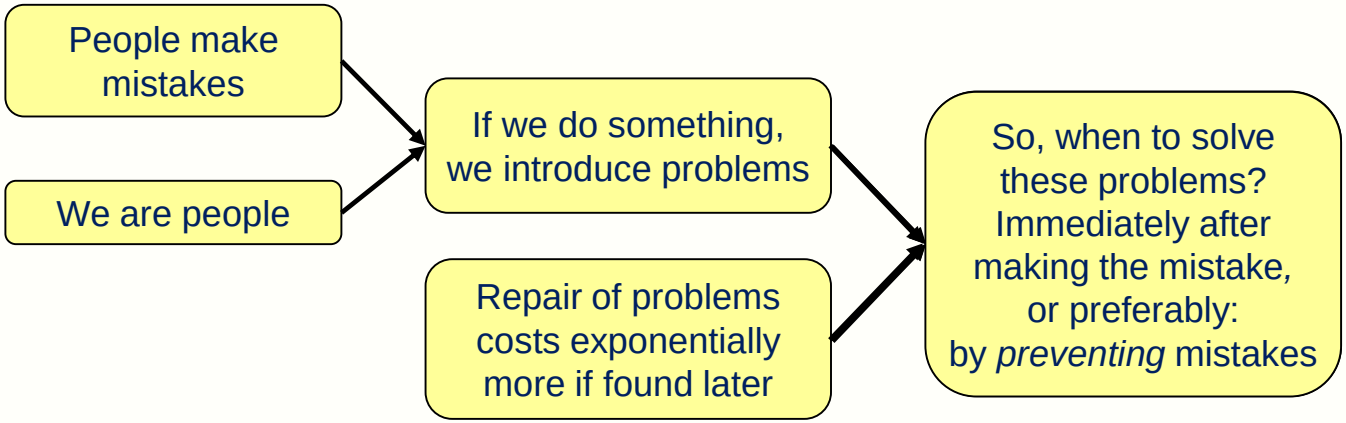
- A design does not have bugs, it has defects
- Defects do not emerge
- People make errors, causing defects, causing problems

Do you ever make a mistake?

- People make mistakes
- We are people
- We are making mistakes

If we think we are done
there are still defects

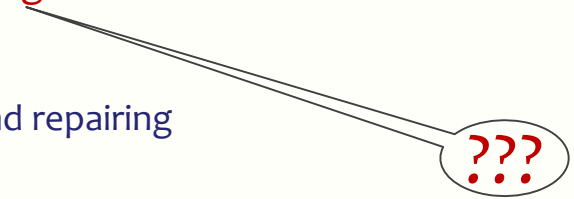
We are people



Prevention costs much less than inject → find (?) → repair (?)

The Problem

- Still defects experienced by your users ?
- Apparently
  - Still defects generated by developers
  - Still defects remaining undiscovered
- However, there is a lot of knowledge how to reduce the generation and proliferation of defects in the first place
- How much of your project will be spent on finding and fixing defects ?
- There is a large budget to do something about it:
  - Some 50% of project time is consumed by all kinds of testing and repairing
  - About 50% of developed software is never used
  - Over 50% of delivered software is never used



Let's move

from

Fixation to Fix

to

Attention to Prevention

- If we don't deal with the root,
  we will keep making the same mistakes over and over
- Toyota Production System: "Stop the Line"

- Without feedback, we won't even know
- With quick feedback, we can put the repetition to a halt

*

Who is regularly doing Reviews and/or Inspections?

Prerequisites - did you bring with you ?

- Available on paper (not just on screen!)
- A few representative pages of documentation of your current project, preferably of a (customer) requirements document
- You will identify the quality of your document
  Warning: after your review, you may decide to discard this document due to its unacceptable quality. However, you at least now know why, and what to do about it.
- If your document isn't too confidential, invite some others to help reviewing the usefulness of your document
- Did you bring a pen as well ?

Case: City of Amsterdam

- Can you teach Inspections ?
- We have a request for proposal to send to potential suppliers
- You'll throw away the document after the course !
- Ha ha
- Of course they did
- They even killed the project

Baseline: Let's check your document → exercise

- Take one page
- Would you invite others to review your document as well ?
- How much time shall we spend (chat) ?
  (show 👍 when you think you're ready)

- Did you find any issues ?

*

What did we find ?

Let's use some Rules

ref Tom Gilb

- Unambiguous
  Every word and phrase should be unambiguous to all potential intended readers
- Clear to test
  Every word and phrase should be clear enough to allow objective test
- Quantified quality
  All qualities (things we want to improve) shall be expressed quantitatively
  (element of unambiguousness)
- No design in requirements
  Specify ***what*** has to be achieved, not ***how*** it should be achieved

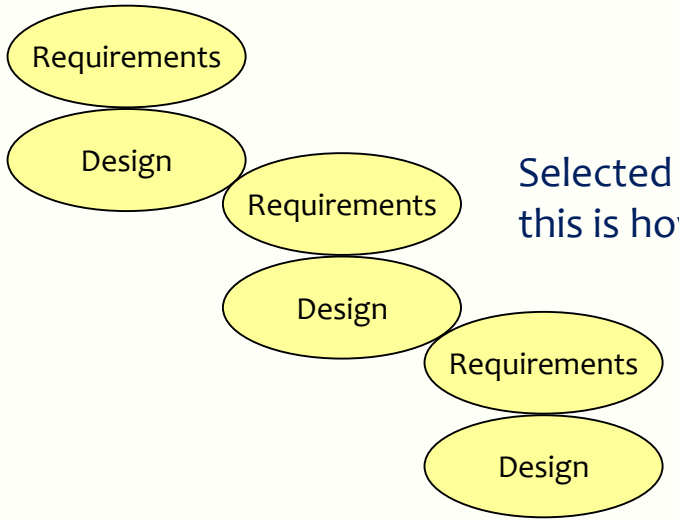
Tom Gilb quote on *quantification*

- The fact that we can set numeric objectives, and track them, is powerful, but in fact it is *not the main point*
- The main purpose of quantification is to force us to *think deeply*, and *debate exactly*, what we mean
- So that others, later, *cannot fail* to understand us

No Design in the requirements, but …

Needs: what do we need

Options: how can we do it



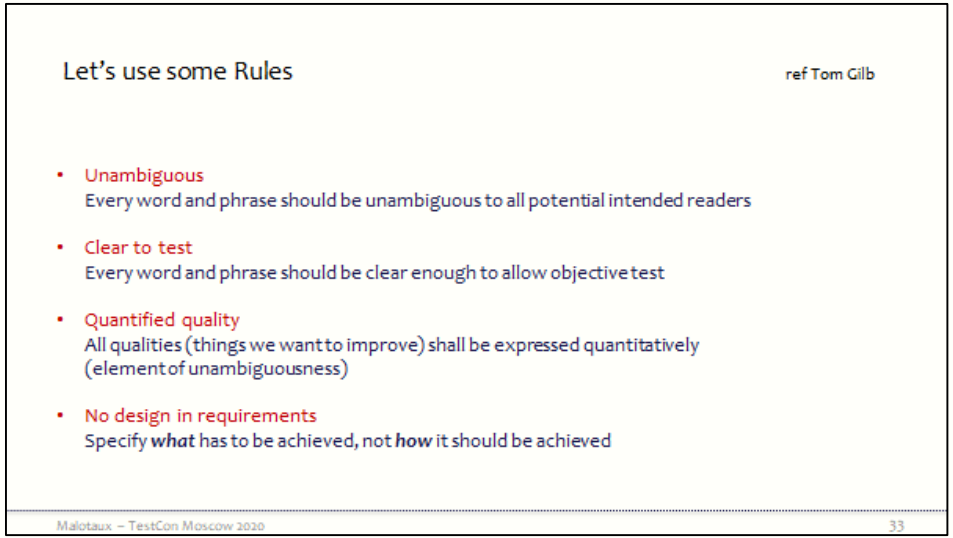
Selected solution:
this is how we are going to do it

Requirement: What the acquirer cares about: 'how good it should be'
Design: Set of decisions made by development: 'how to be good'

Design provides the Requirements for the next level

Let's check again

- Take the same page
- Would you find anything differently ?



*

What did we find ?

Defects found are symptoms of deeper lying problems

Repairing defects creates risks:

- Repair is done under pressure
- We think the problem is solved
- We introduce scars
- We keep repeating the same problems

→ Do Root Cause Analysis and make sure it *never happens again*

Prevention: Root Cause Analysis

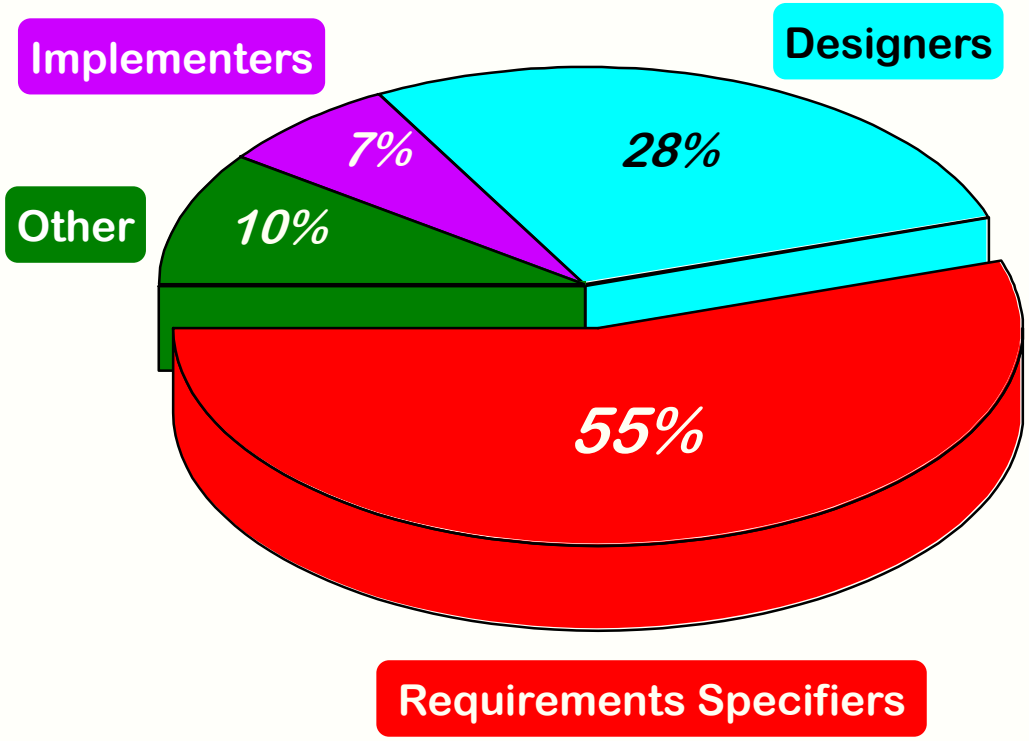
- Is Root Cause Analysis routinely performed – *every time* ?
- What is the Root Cause of a problem ?

- Cause:
  The error that caused the problem
- Root Cause:
  What caused *us* to make the error that caused the problem

- Without proper Root Cause Analysis, we're *doomed to repeat the same errors*

What to look for ?

Typical Defect Injectors (cost breakdown)



After Bender Associates, *1996*

Let's focus on requirements

- Are your requirements clear ?
- What’s the point in designing, implementing, and testing based on unclear requirements ?
- Working on a great solution for the wrong problem ?
- First develop the problem, then the requirements, then the design, only then the implementation
- What’s your experience ?
- Don’t believe anything I say

*

Do you have requirements at all ?

Defects typically overlooked (can test find these ?)

- Functions that won't be used (superfluous requirements)
  - Why to repair defects in the implementation of these requirements ?
  - The only defect is that it has been implemented
- Nice things (not checked, not paid for)
  Shouldn't be there in the first place
- Missing quality levels (should have been in requirements)
  Checking the implementation
  of the documented requirements won't help
- Missing constraints (should have been in requirements)
  Product could be illegal
- Unnecessary constraints (not required)
  What would testing say about these ?

How to check this (usually not specified)?
- 20% of the software is there to make the computer do what it should do
- 80% is there to make the computer *not* do what it should *not* do

Ever seen such requirements ?

- The system should be extremely user-friendly
- The system must work exactly as the predecessor
- The system must be better than before

- It shall be possible to easily extend the system's functionality on a modular basis, to implement specific (e.g. local) functionality

- It shall be reasonably easy to recover the system from failures, e.g. without taking down the power

- Do you know other examples ?

‘Weak words’ in requirements

- e.g. (is it a requirement or not ?)
- etc. (could be anything)
- and (probably two requirements)
- or (can I choose which one ?)
- includes (what more ?)
- such as (is it a requirement or not ?)
- specific conditions (but not specified)
- essentially the same (how much is essentially ?)
- information *may be* shown (may also be not shown ?)
- all possible data (that's a lot !)

- well, better
- much, more
- fast, faster
- high, higher
- easy
- reasonable
- …

3 or 7 ?

Basic Types of Requirements

- Functional *binary*
  - Determine the scope of the project:
  - What are we working on to improve
- Quality/performance *scalar*
  - To enhance the performance of the selected functions
  - This is the essence of development work
- Constraints *binary / scalar*
  - What should we not do, be aware of, be limited by

Example using Planguage

ref Tom Gilb

SMART

Specific

Measurable

Attainable

Realizable

Time

Traceable

Definition:

RQ27: Speed of Luggage Handling at Airport

Scale: Time between <arrival of airplane> and first luggage on belt

Meter: <measure arrival of airplane>, <measure arrival of first luggage on belt>, calculate difference

Benchmarks (Playing Field):

Past: 2 min [minimum, 2018], 8 min [average, 2018], 83 min [max, 2018]

Current: < 4 min [competitor y, Jan 2020] ← <who said this?>, <Survey April 2020>

Record: 57 sec [competitor x, Jan 2017]

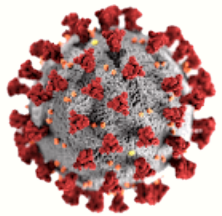
Wish: < 2 min [2022Q3, new system available] ← CEO, 19 Jan 2020, <document ...>

Requirements:

Tolerable: < 10 min [99%, Q4] ← SLA

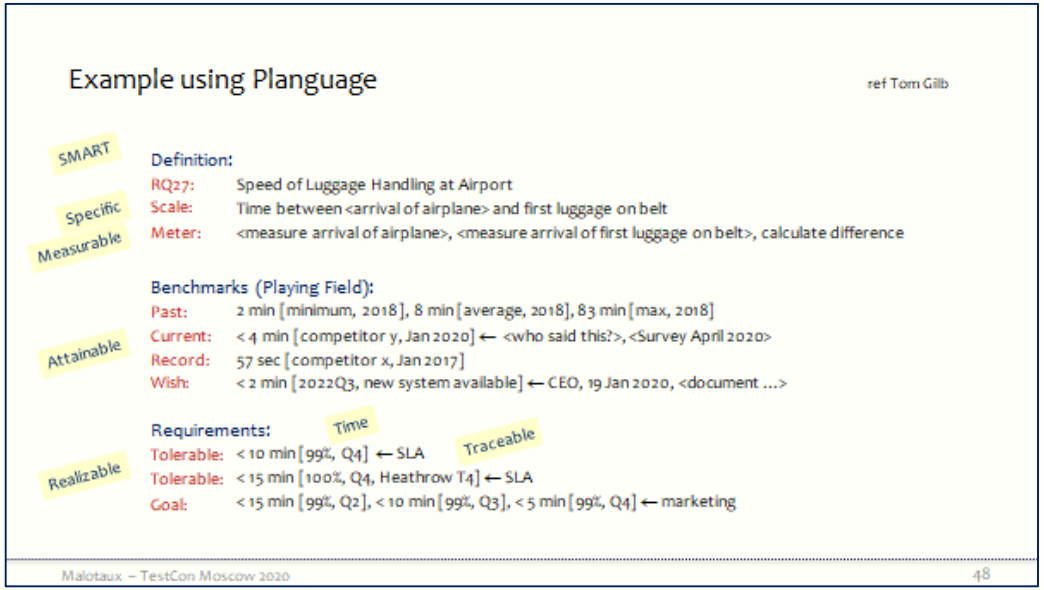
Tolerable: < 15 min [100%, Q4, Heathrow T4] ← SLA

Goal: < 15 min [99%, Q2], < 10 min [99%, Q3], < 5 min [99%, Q4] ← marketing



Exercise

- Think of the most important improvement goal of your work
- Or use one of the requirements from your document

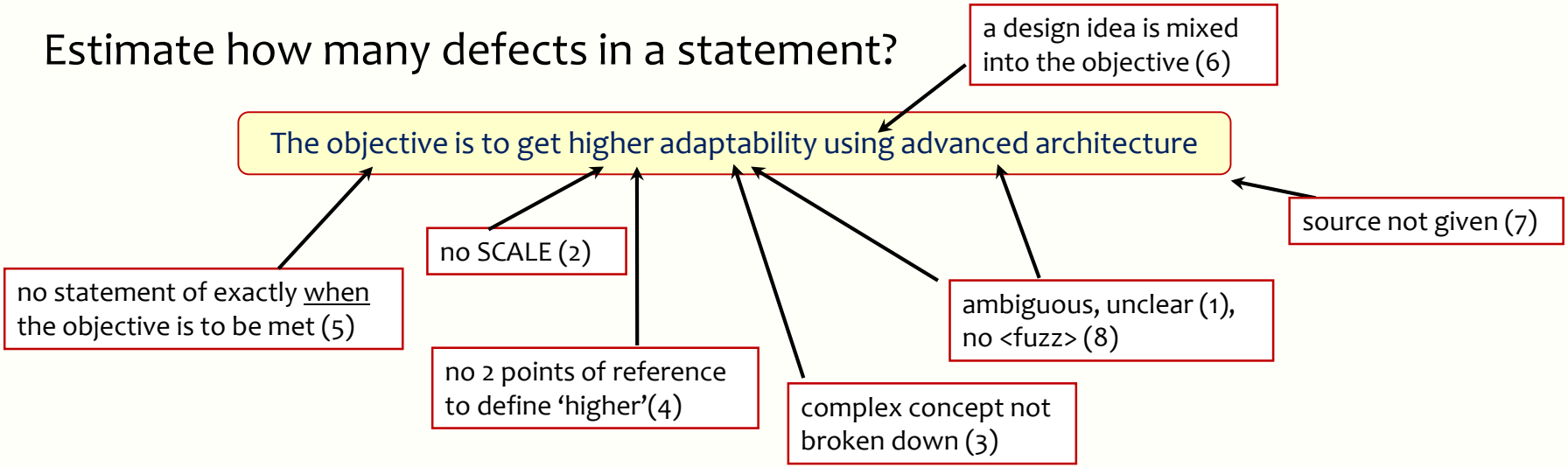


- Can you improve on it using this as an example ?

*

Results ?

Estimate how many defects in a statement?



Rules

1. Unambiguously clear to the intended reader
2. SCALE of measure
3. Complex concepts should be broken down into a set of measurable *elementary* concepts
4. To define 'relative' terms like 'higher' there should be at least two points of reference on the defined SCALE
5. Specify when a quality level is to be available
6. Not mixing design ideas in objectives/requirements
7. Specifying the source (like contract, standard, marketing plan)
8. Fuzzy unclear concepts shall be marked with <angle brackets> for improvement

How many issues can you find ?
Unambiguous, Clear to Test, Quantified, No Design

- The system should be extremely user-friendly
- The system must work exactly as the predecessor
- The system must be better than before

- It shall be possible to easily extend the system's functionality on a modular basis, to implement specific (e.g. local) functionality

- It shall be reasonably easy to recover the system from failures, e.g. without taking down the power

Some requirements

(perhaps your requirements are clearer, or… ?)

- REQ 4010 The storage of the [system] shall store diagnostic information, excluding sensor information, for a period of at least 4 months
- REQ 3776 Recorded data shall be stored and available for transfer for at least 2 months
- REQ 1503 The [system] shall record all diagnostic data in a non-volatile memory
- REQ 5037 Deactivation of a failure by the [system] shall be only allowed when the [system] detects that the failed function is working correctly again in the same state as the failure was activated
- REQ 4758 The [system] shall provide the other diagnostic data (sensor values, performance and usage counters and other possible data) to the service interface for transmission to the wayside within other time intervals

Can we develop based on Management Poetry ?

- Nice input, to be taken seriously
- We write back the requirements, don't we ?
- This is what we plan to do, if you let us continue

- Are we better at requirements ?
  - Unambiguous, Clear to Test, Quantified, No Design

Is this a Requirement ?
or ‘nice input’, to be taken seriously ?

Design

Need

“Create a new ‘Price Sentinel’ component that can detect if the bank’s published customer quotations go off-market, and then to immediately cancel all current quotations.”

How ‘off’ ?

How ‘immediately’ ?

Need

Ref http://rsbatechnology.co.uk

Using 5 Whys

Why do you need a “Price Sentinel” ?

1. To prevent publishing off-market tradable prices
2. To prevent trading loss
   (having to buy at a higher price than the bank offered to the customer)
3. To demonstrate to senior management that e-trading business can safely (no unexpected loss) manage customer trading
4. To ensure that senior management will agree to expand e-trading business in the future to other customer segments and business areas, based on current business performance
5. To meet business medium / long-term financial targets

Ref http://rsbatechnology.co.uk

First try

New ‘Price Sentinel’ component:

- detect if the bank’s customer quotations go off-market
- then immediately cancel all current quotations

- Off-market
  - Our margin less than 0.1%
- Immediately (<happening>?)
  - Scale: seconds after <detection>
  - Current: 600 sec (= 10 min)
  - Goal: 1 sec

Prioritized solutions by Impact Estimation

(Don't immediately go for the first solution that comes to mind !)

	Kill button	Price Sentinel
Cancel 600 → 1 sec	10.5 sec (note) 98%	1 sec 100%
Cost	1 day	30 day (6 sprint)
Note: 10 sec human recognition time, 0.5 sec cancel time		

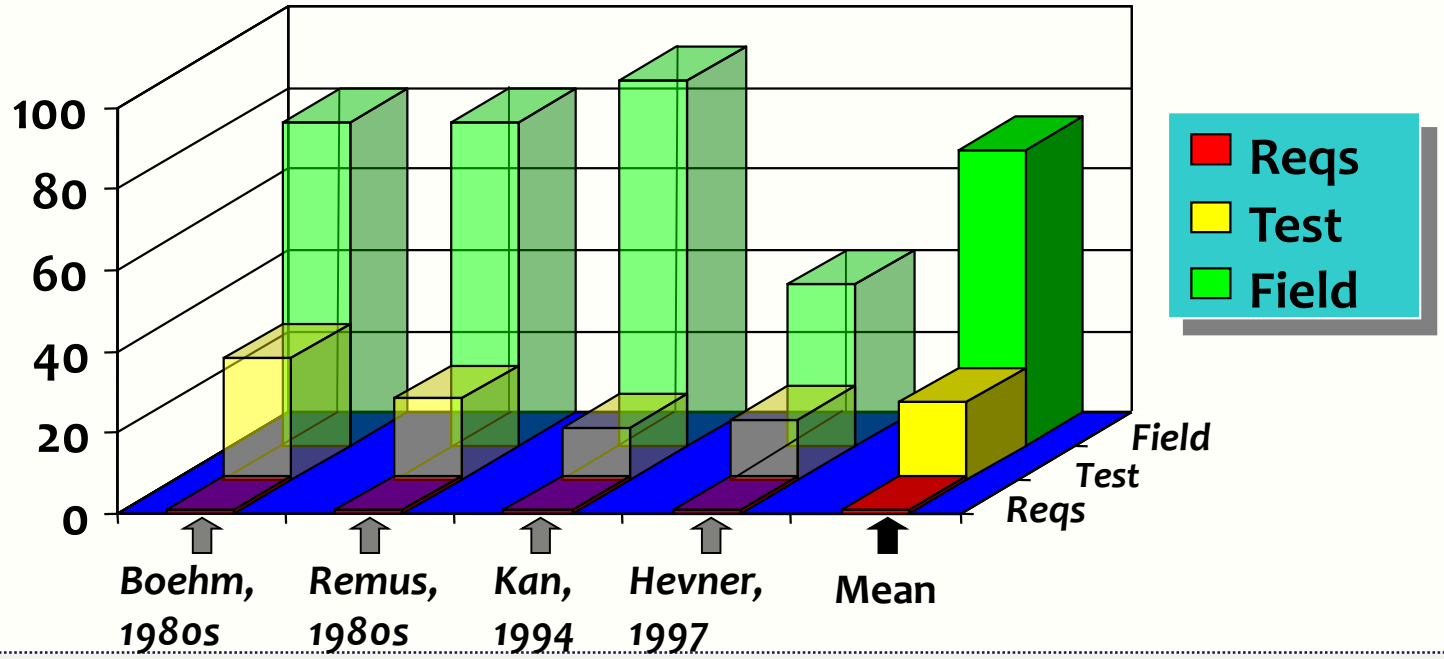
Reviews & Inspections

Costs of defects

The longer a defect stays in the system,
the more it costs to find and repair

Cost of Requirements Defects

The longer a defect stays in the system,
the more it costs to repair

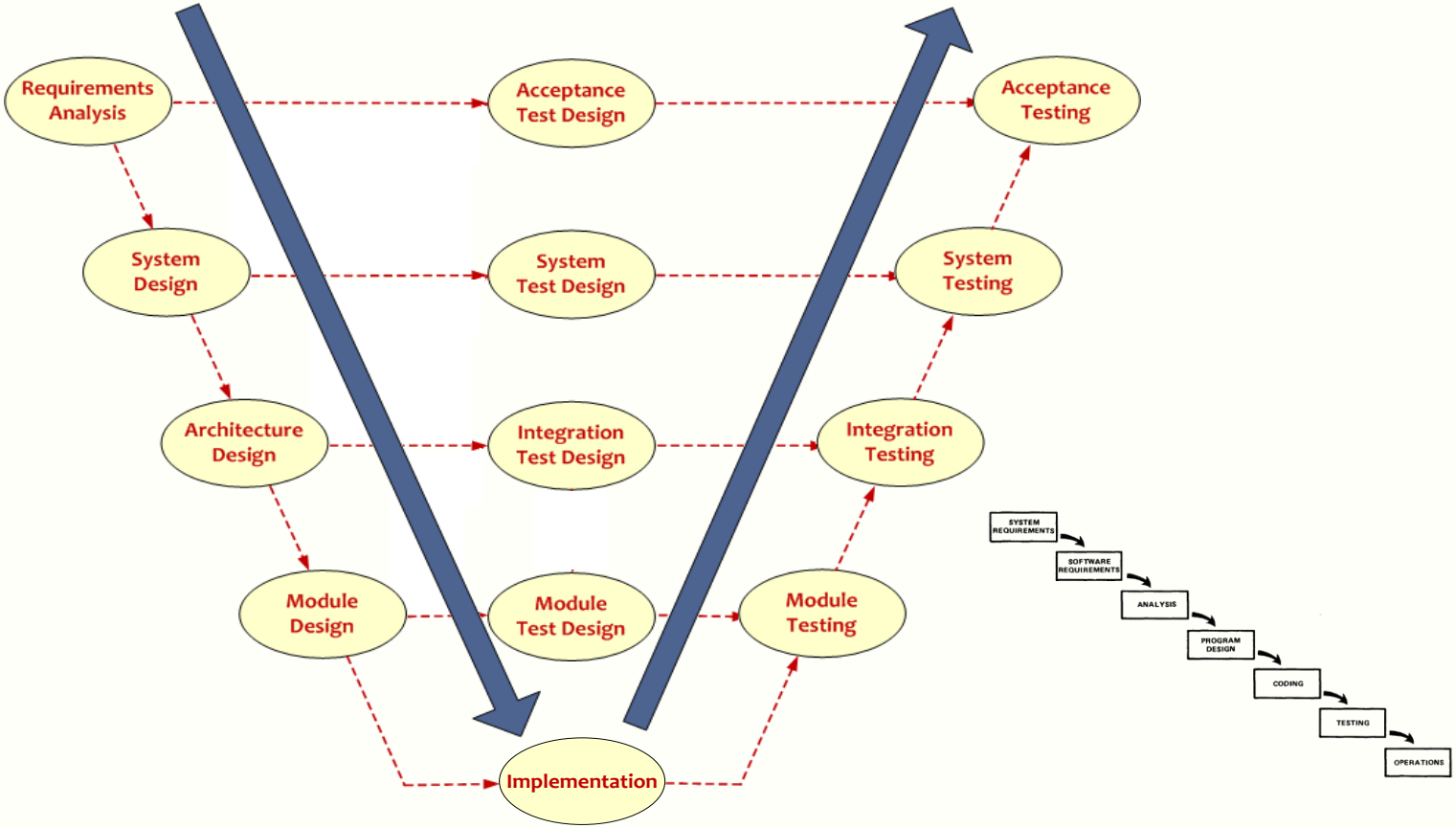


Testing vs Reviews & Inspections

- If you find an issue during Test, you still have to find the origin
- If you find an issue during Review or Inspection, you're on top of it

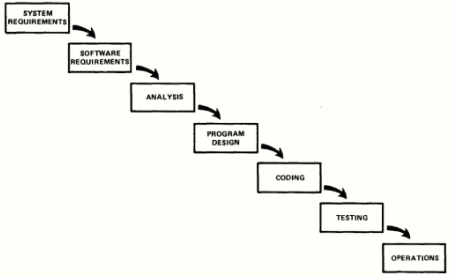
- Testing means running the system
- Review / Inspection means verifying a document

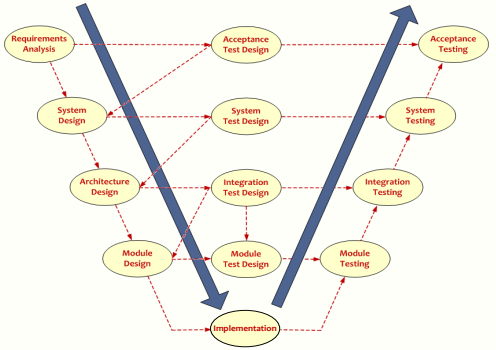
V-Model

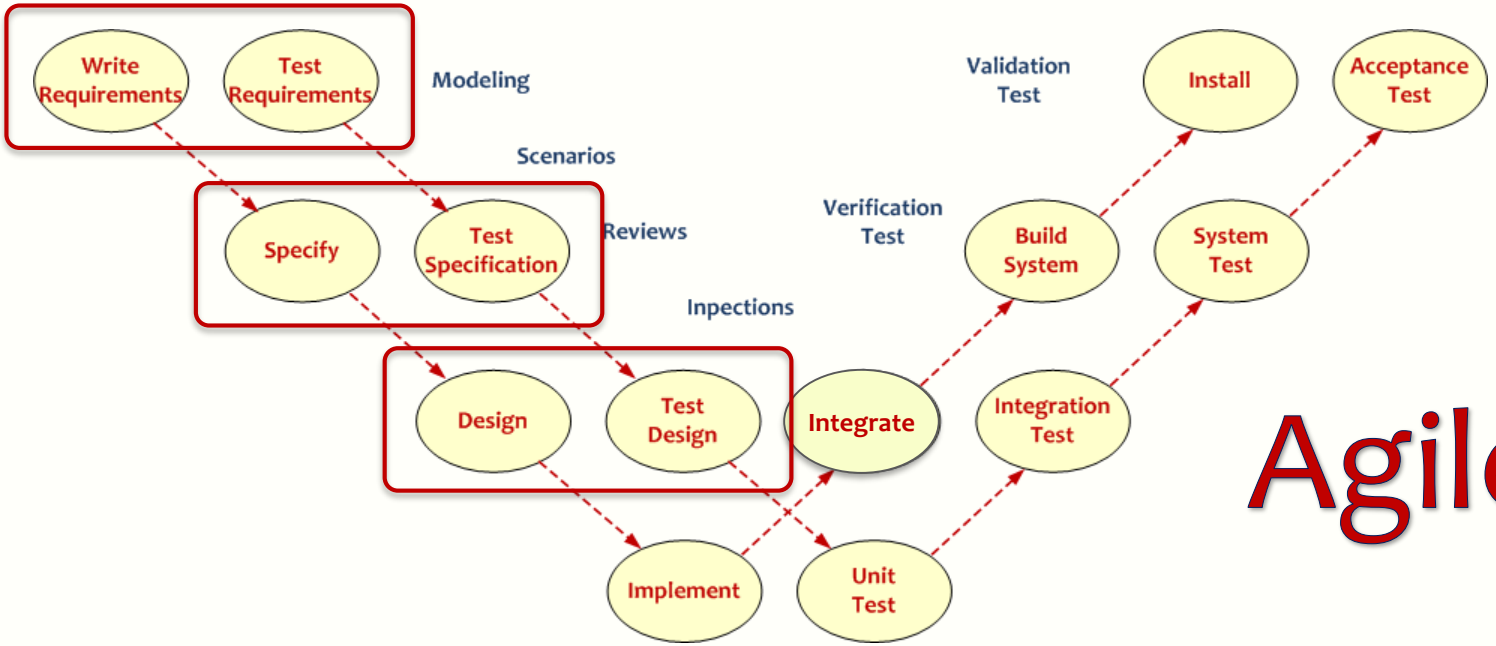


Remember: All models are wrong …, some are useful

W-model



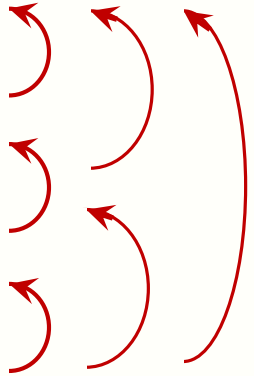




Agile ??

Don’t pollute the next stage

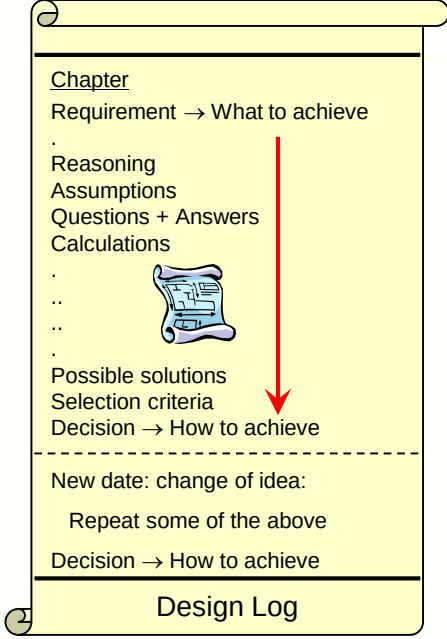
- Requirements
- Review
- Design
- Review
- Code
- Review



Iterate as needed

- Test (no questions, no issues)
- If issue in test: no Band-Aid: start all over again:
  Review: What’s wrong with the design ?
- If there is no design: Reconstruct the design !
- QA to review the DesignLog for more efficiently helping the developers:
  Ask "Can we see the DesignLog ?"

Cleanroom



Many types of Review to choose from

- Informal Review
- Pair Programming
- Mob Programming
- Technical Review
- Walkthrough
- Formal Inspection (Fagan type)
- Cleanroom Inspection
- Formal Inspection (Gilb/Graham type)
- Agile/Extreme/Lean/Early Inspection
- Gate Review
- Unit Test
- Debugging
- Test

Techniques

- Can you look at this ?
- Over the shoulder
- Pair Programming
- E-mail
- Tool
- On Screen
- Projector
- On Paper
- Formal process

Have you been looking at the document ?

Did you check my car ?

We have looked at it on the bridge !

What I think

What they mean

Formal Reviews (vs Ad-Hoc)

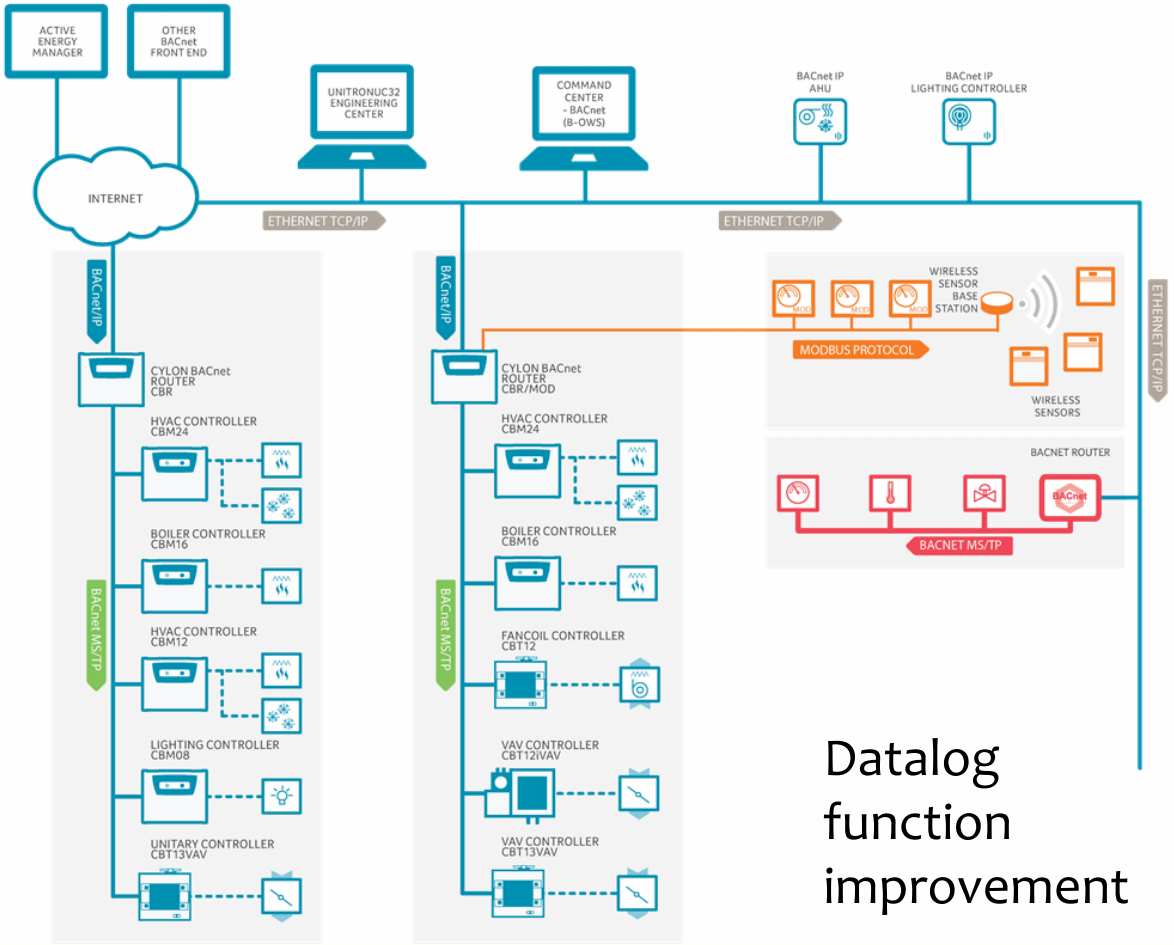
- Defined, repeatable process
- Measures effectiveness
- Continuous improvement
- Rules/checklists
- Feeds prevention process

What to review ?

• Wish specification	Thank you, nice input, to be taken seriously
• Contract	This is what I'll take you to court with
• Business Case	Why are we doing it
• Requirements	What the project agrees to satisfy
• Design	Selecting the 'optimum' compromise
• DesignLog	How we arrived at this decision
• Specification	This is how we are going to implement it
• Implementation	Models, code, schematics, plans, procedures, hardware, software, documentation, training

Case: Can you teach Inspections ?

- Short intro
- Are you regularly reviewing ?
- Let’s do it: baseline
  - Take a document
  - Reproduce one page
  - Do review
  - No issues



Datalog function improvement

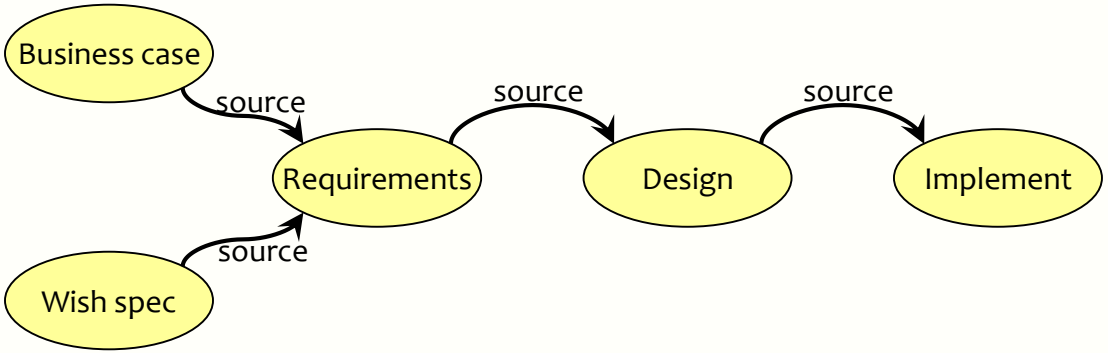
Simple Rule for Reviews

One Rule:

'source': "We don't review unless there is a source document"

Review again…

Many issues

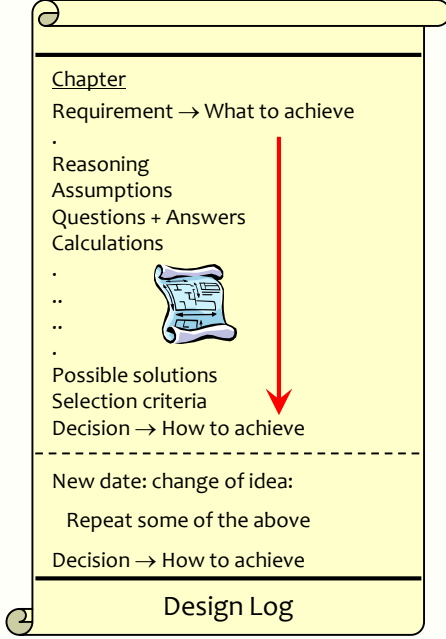


Consequence

- No code until design-log is reviewed
- You're delaying my project !
- Example
- Solution
- Thanks, you saved my project

- Did I do the same ?
  - Sometimes, all we can is to review ourselves …

- Telling people to change: resistance
- Using an Inspection to let people change themselves …



From the DesignLog

A number of Firmware based methods of removing the glitches from the datalog reading process have been investigated,

but it has been decided to go with a mechanism implemented in the external system reading the datalog to remove the glitches.

Case: In the pub

James:

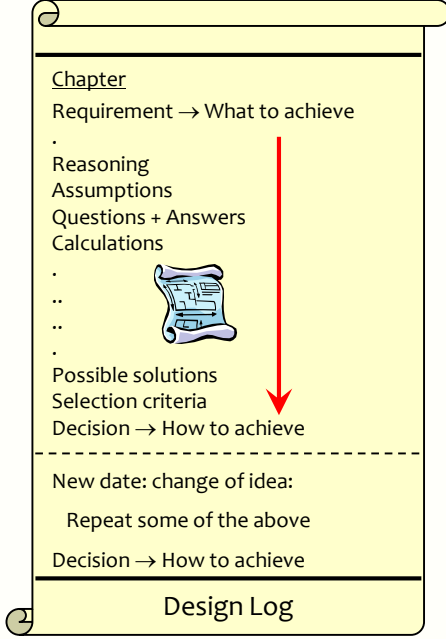
Niels, this is Louise

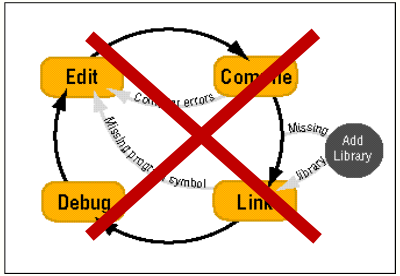
Louise, this is Niels, who taught me about DesignLogging

Tell what happened

Louise:

- *We had only 7 days to finish some software*
- *We were working hard, coding, testing, coding, testing*
- *James said we should stop coding and go back to the design*
- *"We don't have time!" - "We've only 7 days!"*
- *James insisted*
- *We designed, found the problem, corrected it, cleaned up the mess*
- *Done in less than 7 days*
- *Thank you!*

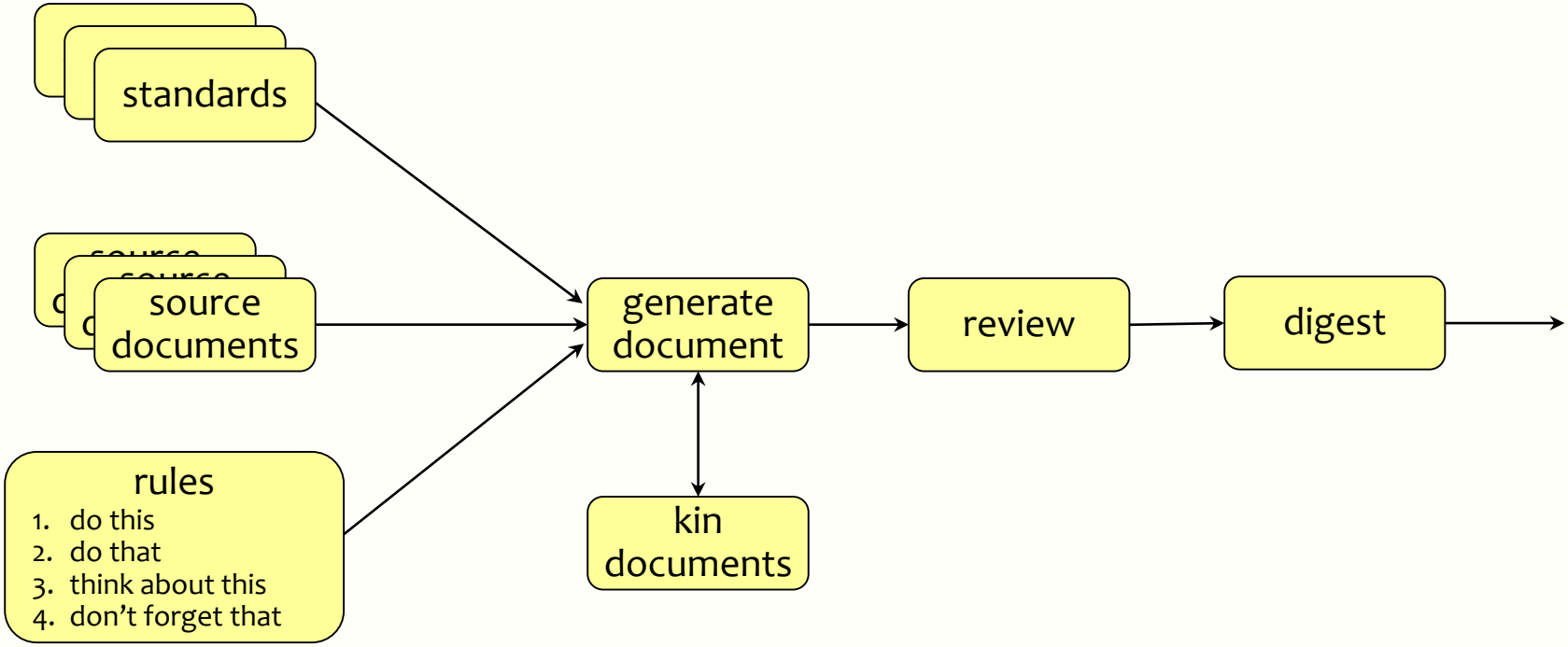




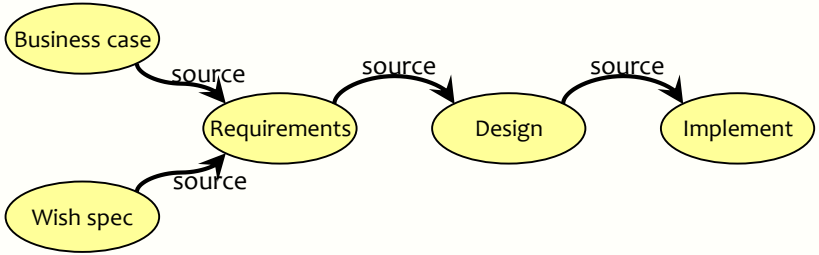
What James told me later

- I gave the design to two colleagues for review
- Louise corrected some minor issues
- It went into a ‘final’ review, with another colleague
- Based in his expertise, the solution was *completely reworked*
- Actually, two features were delivered and deployed
  - One that was design and code reviewed had no issues after deployment
  - Other one, was the source of quite some defects
- From now on we use DesignLogs, to be reviewed before coding

Document generation



‘Sources’ rule



- Any work product will be reviewed against
  - Itself
  - Kin documents
  - Source documents

  If we don’t have the source, how can we judge the work product?

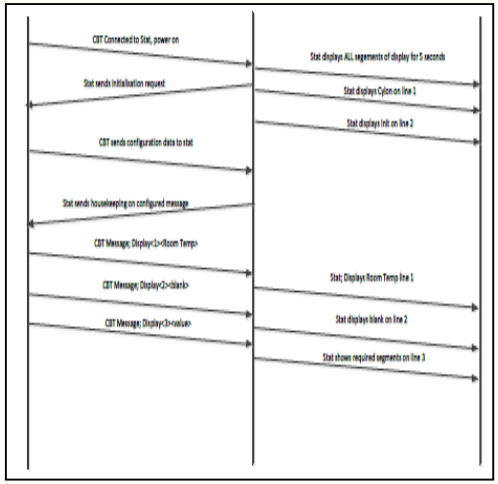
- We always update the source document first before changing the work product(s)
  - First change the Requirement, then the Design, then the Code, and the Test (as needed)

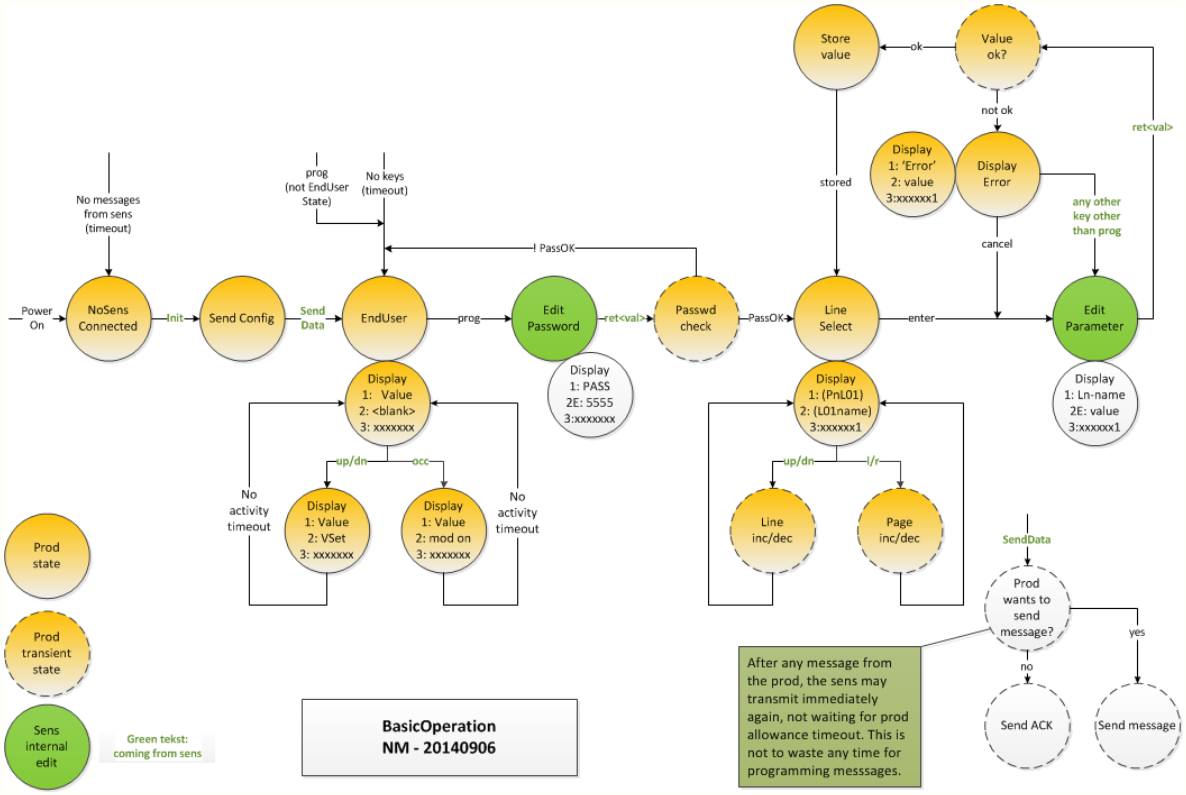
Make Documents Reviewable

- If not, they're probably not very useful

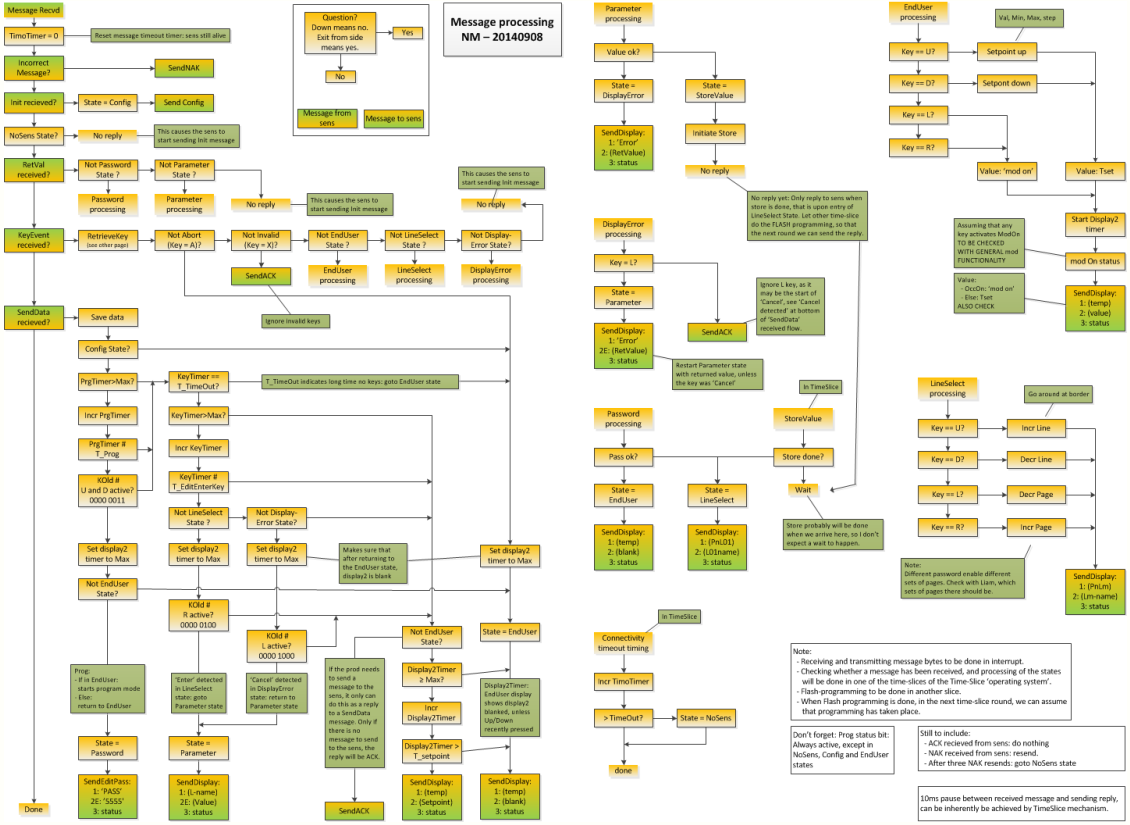
  Unambiguous, Clear to Test, …

Design example
47 pages documentation condensed into one page



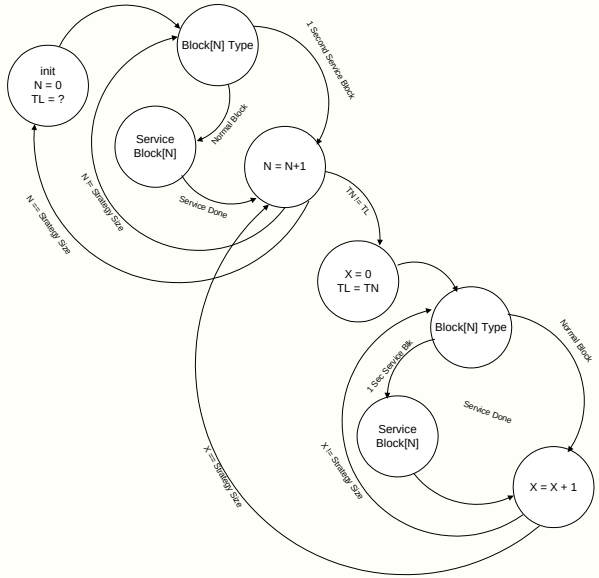


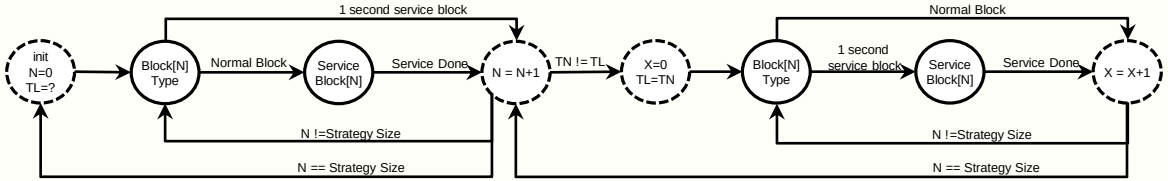
Design example



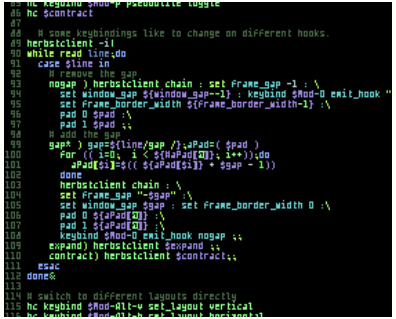
There are many ways to represent a design

- Only few are useful
- Don't waste reviewer's time



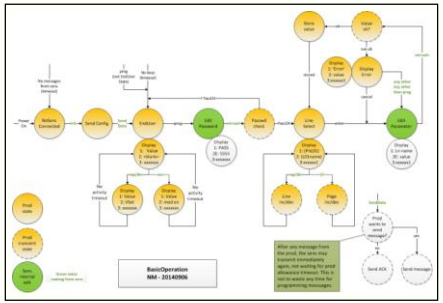


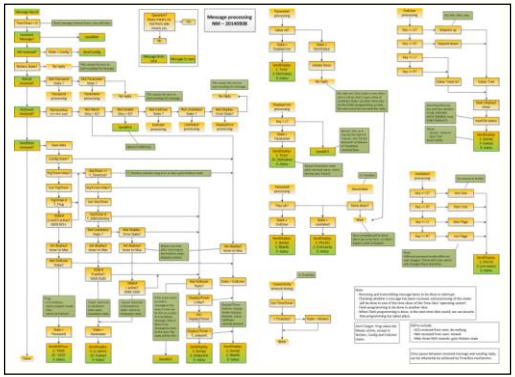
What is better than reviewing code ?



- Do you ever review software ?
- What do you review ?

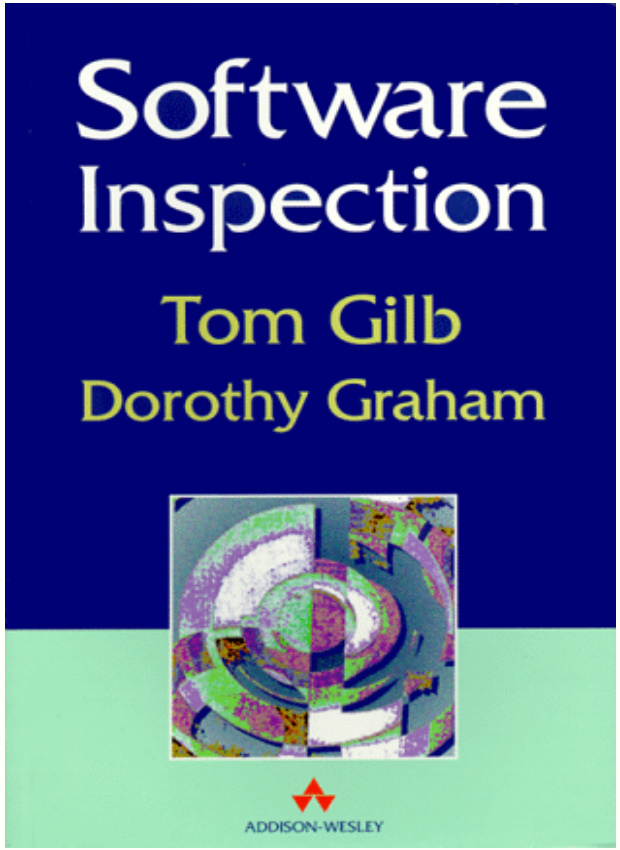
- What is better than reviewing code ?
  - May I review the design first ?





Inspection

- Most rigorous form of review
- Pioneered by Fagan (IBM) (paper 1976)
  - Locating all the defects in a work product, focus on code
- Inspection economics: Gilb/Graham (Software Inspection, 1993)
  - Quantifying the defect density of a work product
    and preventing poor quality work from moving downstream
- Early Inspection
  - Not waiting until the whole waterfall of the document is completed
- Is not the same as Review
- Use:
  - Walkthroughs for training
  - Technical Reviews for consensus
  - Inspections to improve the quality of the document and its process
  - Gate Reviews to decide what to do with it
- Would you base further work or decisions on a document of unknown quality ?



A ready to use recipe …

A typical Review …

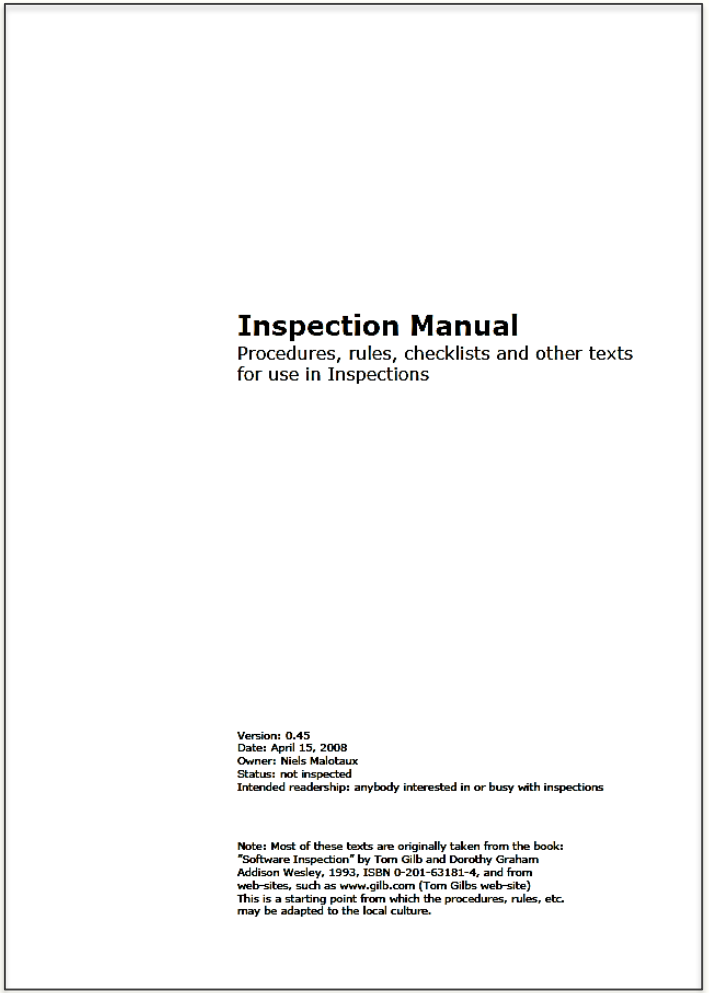
- The document to be reviewed is given out in advance
- Typically dozens of pages to review
- Instructions are "please review this"
- Some people have time to look through it
- Review meeting often lasts for hours
- Typical comment: "I don't like this"
- Much discussion, some about technical approaches, some about trivia
- Don't really know if it was worthwhile, but we keep doing it
- Next document reviewed will be no better

Inspection is different

- The document to be reviewed is given out in advance
  - *not just product - rules to define defects, other docs to check against*
- Typically dozens of pages to review
  - *chunk or sample*
- Instructions are "please review this"
  - *training, roles*
- Some people have time to look through it
  - *entry criteria to meeting, may be not worth holding*
- Review meeting often lasts for hours
  - *2 hr max*
- Typical comment: "I don't like this"
  - *Best Practice rules - Rules are objective, not subjective*
- Much discussion, some about technical approaches, some about trivia
  - *no discussion, highly focused, anti-trivia*
- Don't really know if it was worthwhile, but we keep doing it
  - *exit criteria - continually measure costs and benefits*
- Next document reviewed will be no better
  - *most important focus is improvement in processes and skills*

16 page Inspection Manual

www.malotaux.eu/?id=inspections



Inspection Manual

Procedures, rules, checklists and other texts for use in Inspections

Version: 0.45
Date: April 15, 2008
Owner: Niels Malotaux
Status: not inspected
Intended readership: anybody interested in or busy with inspections

Note: Most of these texts are originally taken from the book:
"Software Inspection" by Tom Gilb and Dorothy Graham
Addison Wesley, 1993, ISBN 0-201-63181-4, and from
web-sites, such as www.gilb.com (Tom Gilbs web-site)
This is a starting point from which the procedures, rules, etc.
may be adapted to the local culture.

- Home
- Services+
- Conferences
- Courses/Workshops+
- **Reviews & Inspections+**
  - Review/Inspection workshop
- Project management+
- Engineering+
- Human behaviour+
- Texts+
- Booklets / downloads
- Glossary
- Aphorisms
- Quotations
- Mantras
- Books
- Contact
- Search
- Use the Sitemap to find your way

Twitter:
Tweet
Follow

Reviews and Inspections

Formal Inspection was pioneered in the seventies by M.E. Fagan at IBM. He published about it in IBM Systems Journal in 1976: Vol 38, nr. 2&3 (*Turning Points in Computing 1962~1999*): *Design and code inspections to reduce errors in program development* (.pdf; 1.5Mb). Since then, a lot of experience has been gained on the techniques of Inspections. NASA has a very useful "Software Formal Inspections Guidebook" and a "Software Formal Inspections Standard". A very useful guide to Inspections (with a big I) is the book by Tom Gilb en Dorothy Graham "Software Inspection" (1993). The word "Software" could have been omitted: Inspections are by no means limited to software. Without the "software" the book would however not sell as well and let's be frank, in the field of software, Inspections are probably most needed. The authors cover in great detail all steps which make the Inspection process so valuable. Several chapters describe the experience of Inspections in various organisations. After having read this book, you will ask yourself how you ever could have done without Inspections. The question is not any more *whether* Inspections should become a standard part of your development process, nor *when*. Only *how*. Answering this question with you is our job!

Invariably I have found that if we use the following sequence in any development activity, we will deliver better and faster: A great technique to move towards Zero Defects!
(see for example DesignLog-case#2)

- Design, followed by Review (repeat as needed)
- Implement/Code, followed by Review (repeat as needed)
- V&V/Testing doesn't find issues
- User doesn't find issues.

In the Gilb/Graham book you will find a complete set of forms for Inspections. You can order it from Amazon.com.

For a description of the Review/Inspection workshop, see the workshops page.

Latest versions of Inspection documents for Document Inspections:

Download **16-page Inspection Manual** (pdf, 183 kb), V0.45
Download **Inspection Master Plan** (.doc, 47kb)
Download **Data Summary**, logging sheet, brainstorm sheet (.xls, 72 kb)
Download **ReviewInspCourse** (pdf, 2504 kb), as used at a client in October 2007

Early Inspections
Even where Tom and Dorothy introduced the "Economy of Inspections" in 1993, since then we have learnt a lot more, and nowadays we are using an even more light-weight process that can be used with many types of documents, producing even better results at even much less investment.

Comments or discussion: niels@malotaux.nl

this page original date: 2011-02-12
this page last update: 2019-04-22

Inspection goals and effects

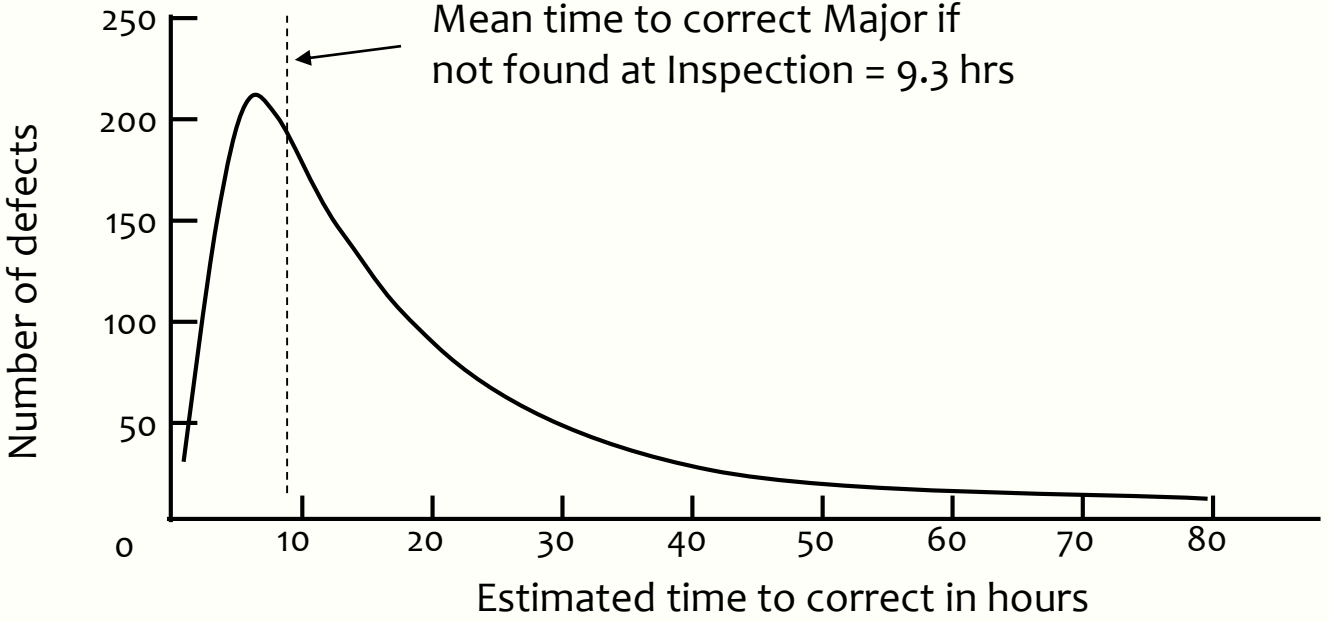
- Identify and correct major defects
- Most important:
  Identify and remove the source of defects
- Consequence:
  Education and interaction:
  How should we make documents in the first place?
- Interesting side-effect:
  People get to know each others documents efficiently

Defect classes

- Major defect
  - Defect probably has significantly increased costs to find and fix later (test, field)
    - 10 engineering hours lost extra
    - Average time in work-hours to find, log and fix a major defect by Inspection is 1 hour (observed by many sources)
- Minor defect
  - Not major (no significant impact on result)
- Super-major/critical
  - Order of magnitude more costly than major
  - Project threat

Cost of Repair

ref Software Inspections, fig 14.6, p315



Rules

- Rules are the law for documents
- Defect = Rule violation
  not: “I think this is wrong”, or “I don’t like it”, or “I know better”
- Rule:
  All quality requirements must be expressed quantitatively
- Typical requirements found:
  - The system should be extremely user-friendly
  - The system must work exactly as the predecessor
  - The system must be better than before

Generic Specification Rules

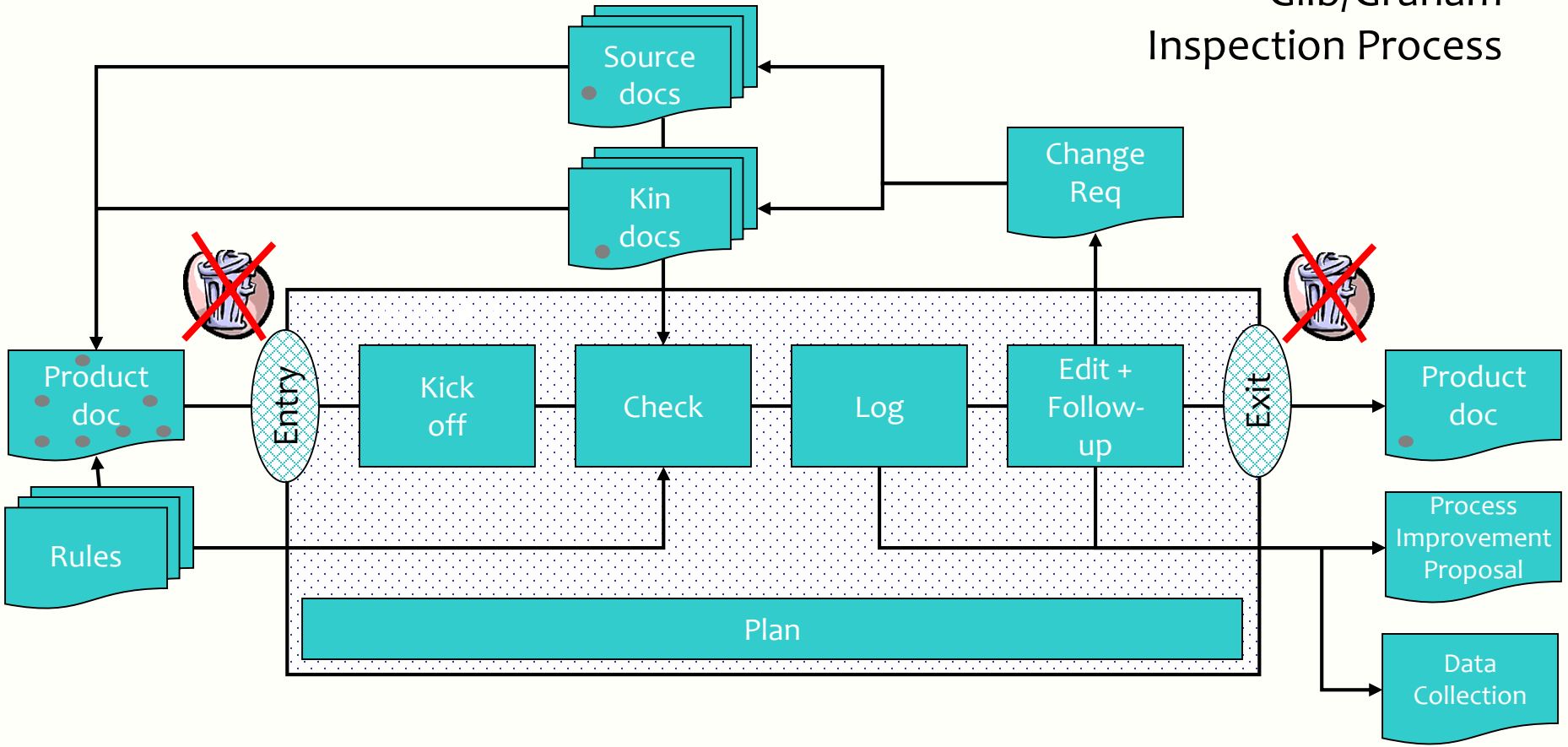
(see Inspection Manual)

GE0	(def)	Generic engineering specification rules apply to all engineering documents as required best practices
GE1	(relevant)	All statements should be relevant to the subject
GE2	(complete)	There should not be any significant omissions
GE3	(consistent)	Statements should be consistent with other statements in the same or related documents
GE4	(unambiguous)	All specifications should be unambiguous to the intended readership
GE5	(note)	Comments, notes, suggestions, not official part of document shall be clearly marked (“”, *ital*, /**/)
GE6	(brief)	All specifications shall be as brief as possible, to support their purpose, for the intended readership
GE7	(clarity)	All specifications shall result in clarity to the intended readership regarding it’s purpose or intent (the burden is on author, not the reader) *Note: It is not enough that statements are unambiguous. They must contain clarity of purpose: why is it there?*
GE8	(elementary)	Statements shall be broken into their most elementary form *Note: This is so that they each can be cross-referenced externally (Traceability)*
GE9	(unique)	Specifications shall have a single instance in the entire project documentation
GE10	(source)	Statements shall have source info (spec ← source)
GE11	(risk)	The author should clearly indicate any information which is uncertain or poses any risk to the project, using indications like: {<vaguely defined>, ?, ??, 70% ±20, suitable comments or notes}
GE12	(verifiable)	All statements should be verifiable
GE13	(true?)	The statement is simply not true

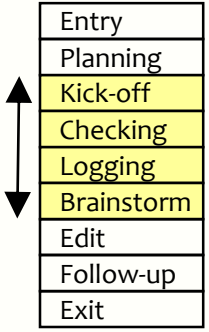
Check Lists

- Checklists contain interpretations of Rules to help reviewers to find more issues
- Rules are “The Law”
- Checklists provide “Jurisprudence”

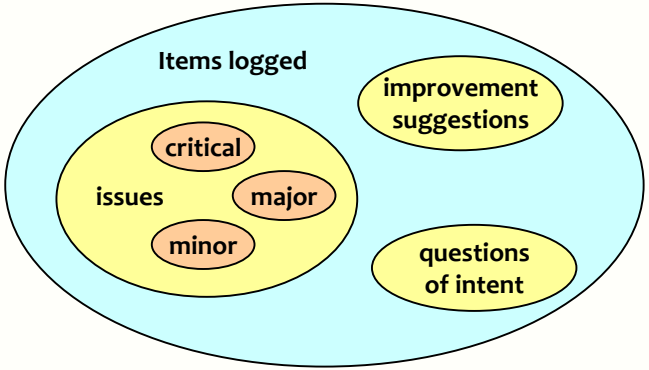
Gilb/Graham Inspection Process



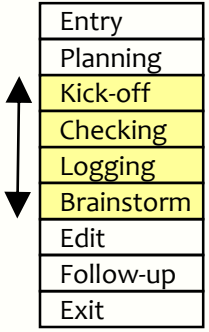
6 hour initial Inspection process



- 2 hr Kickoff
  - Why
  - How
  - What
- 2 hr Individual checking
  - 1 hr Whole document / relevant chapter
  - 1 hr 2 selected pages
- 2 hr Logging meeting
  - 1 hr Logging issues
  - ½ hr Discussion about Inspection process
  - ½ hr Discussion about what should have been in the document



4 hour mature Inspection process



- ½ hr Kickoff
  - Why
  - How
  - What
  - 2 hr Individual checking
  - 1 hr Whole document / relevant chapter
  - 1 hr 2 selected pages
  - 1½ hr Logging meeting
  - 1 hr Logging issues
  - ½ hr Discussion about Inspection process
  - ½ hr Brainstorm

What do you need

- Trained Inspection leader (process, psychology)
- Inspection Manual
  - Rules, Procedures
- Document + owner
- Checkers
- Inspection Master Plan (one page)
  - Who, What, Where
- Presentation for the Kick-off meeting
  - Why, How, What
- Inspection metrics template
  - Data collection
  - Issue collection
  - (Brainstorm - fruits collection)
  - Verdict

Inspection Master Plan

Owner: Niels Malotaux – Version 1.01 – 23 Nov 2001

Inspection no. 7784-RMU28_1 **Date requested:** Nov 29, 2001

who	name	init	tel	e-mail	role	scan	time	min/ page	check	time	min/ page	rule set
Leader	Maarten	mvl	-		Leader	Product document	½ hr	3 min	Ch 3.1 + 3.2	1½ hr	~30	GE
Author	Rudy				Author	Product document	½ hr	3 min	Ch 1 - 3.(0)	1½ hr	~30	GE
Checker	Frank				-	Product document	½ hr	3 min	Ch 1 - 3.(0)	1½ hr	~30	GE
Checker	Raf				-	Product document	½ hr	3 min	Ch 3.3 + 3.4	1½ hr	~30	GE
Checker	Vova				-	Product document	½ hr	3 min	Ch 3.3 + 3.4	1½ hr	~30	GE
Checker					-							
Checker					-							

doc	owner	init	tel	e-mail	docname	date	ver	Location Project\software\documents\	insp status	maj/ page
Product	Rudy				Eco Product Configurations SD7784-RMU28	2001-11-23	0.1	configuration management	For inspection	
Reference	Niels Malotaux	nma		niels@malotaux.nl	InspectionManual	2001-11-20	0.42	Q:\Inspections\CoursenspMan.doc	Not inspected	
Source	Jan Hollevoet				Branching Strategy	2001-09-17	1.0		Not inspected	
Source	Rudy				Eco Merging Strategy SD7784-RMU27	2001-11-23	0.2		Not inspected	
Source	Jan Hollevoet				Software Build Instructions ThisProduct	2001-11-19	1.4		Not inspected	
Source									Not inspected	

meeting	date	location	start	end
KickOff	2001-11-29	here		
Logging	2001-12-06	same		

Individual checker data collection To be filled in by each checker, ***before*** logging meeting	**Checker:**	
	scan	**check**
Time spent (X.X hrs)		
Pages studied		
Majors		
Super majors (project threat)		
Minors		
Process Improvements		
Questions		

Instructions

Inspection goals: Getting the product exited
Learning Inspections

Strategy to meet goal: Do Inspection, find as many issues as possible
Note: The brainstorm will initially be replaced by:
- 30 min. discussion about what you think of this inspection process
- 30 min. Just In Time Training on the subject of the document

Optimum checking rate: 60 min per page
At first Inspections we will use about 30 min per logical page

Exit condition: < 2 major defects remaining per page

Assignment for this Inspection:

Please check the sheets against all source document and rule set GE. See Inspection Manual. In this manual you can also find the procedure for checking (Procedure for Checker during Checking: CC). Read this procedure to know what to do during checking.

Inspection statistics

Data summary

prepare | fill in | changeable | calculated | assumed | results

Owner: Niels Malotaux - Version 1.01 - 23 Nov 2001

InspectionID	2	Date	29-nov-01	Leader	Niels Malotaux	e-mail	niels@malotaux.nl	
Product document	Eco Product Configurations SD7784-RMU28					Pages	9	Chck 3

Individual checking data (to be reported during the entry process for logging meeting)

Checker report	Pages studied		Time spent (x.x hrs)		Major + SM issues		minor issues		Improve-ments		Questions of intent		Check rate hr per page		Majors per hour		Majors per page	
	Scan	Chck	Scan	Chck	Scan	Chck	Scan	Chck	Scan	Chck	Scan	Chck	Scan	Chck	Scan	Chck	Scan	Chck
Author	9,0	3,0	0,5	1,0	9	4	4	1			2	1	0,05	0,33	20,0	4,0	1,0	1,3
Checker 1	9,0	3,0	0,5	1,5	2	0	1	4					0,06	0,50	4,0	0,0	0,2	0,0
Checker 2	9,0	3,0	0,5	1,0	3	4	1	2		1		1	0,06	0,33	6,0	4,0	0,3	1,3
Checker 3	9,0	3,0	0,5	1,3	1	1	19	2	0	1		1	0,06	0,42	2,0	0,8	0,1	0,3
Checker 4	9,0	3,0	1,0	2,0	19	30							0,11	0,67	19,0	15,0	2,1	10,0
Checker 5																		
Total checking hours			9,7	wrkhrs			Average team checking rate						0,07	0,45	10,2	4,8	0,8	2,6

optimum checking rate is 1,00 hr per page

Logging meeting summary

	Major + SM issues		minor issues		Improve-ments		Questions of intent		Total items	
	Scan	Chck	Scan	Chck	Scan	Chck	Scan	Chck	Scan	Chck
Unique found during checking	21	21	13	12	2			1	36	34
New found in meeting									0	0
Total	21	21	13	12	2	0	0	1	36	34

Final findings as reported by editor

	Scan	Chck	Total
Major + SM issues	21	21	42
minor issues	13		
Change Reports	2		

Edit time		wrkhrs
Follow-up time		wrkhrs
Exit time		wrkhrs

Follow-up and exit time: author + leader

Exit results

Did the Inspection Process meet the Exit Criteria? (yes/no)		date
comment		

Assumptions

Average time to find and fix later	9,3 hrs/major
% causing defects	50% of found in Inspection
Insp effective-ness	50% % Maj found per page
Repair efficiency	5/6 (1 - fraction not repaired correctly)

Preparation

Planning time	2,0	wrkhrs
Entry time	1,0	wrkhrs
Kickoff, no of people	7	people
Kickoff, time	50	min

Planning and entry time: author + leader

Logging meeting data

(fill in at the end of logging meeting)

Number of people	7	people
Item logging time	90	min
Discussion time		min
Checking time		min
Pages chckd in meeting		pages
Brainstorming time		min
Items logged in meeting	36	
Logging time	10,5	wrkhrs
Item logging rate	0,40	items/min
Meeting checking rate	0,00	hr/page

Calculations

Total checking time	9,7	wrkhrs
Checking time before and in meeting		
Detection time	29,0	wrkhrs
Planning+Entry+Kickoff+Checking+Logging		
Control time	8,8	wrkhrs
Planning+Entry+Kickoff+Followup+Exit		
Defect removal time	29,0	wrkhrs
Detection+Edit+Followup+Exit		
Efficiency	1,4	Maj/wrkhr

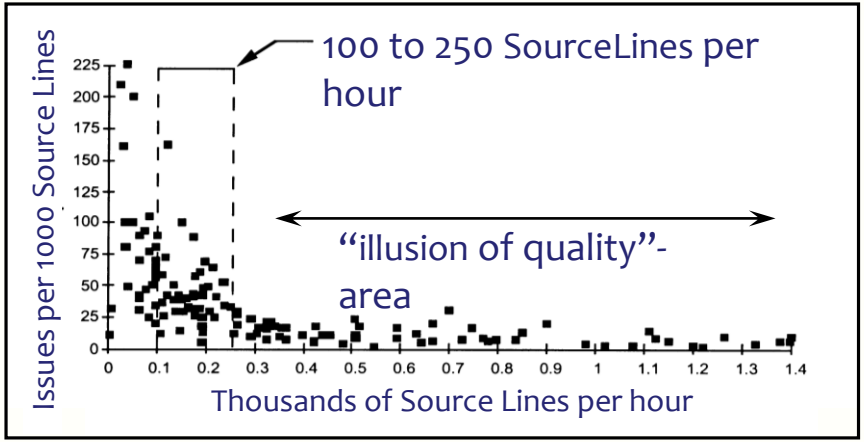
Time saved

Net time saved	134	hrs saved
by using	29	hrs used
Relative cost of Inspecting	18%	used/would

Results in document

Majors per page found	7,0	Maj/page
Maj per page remaining	8,2	Maj/page
Majors remaining in doc	73,5	Majors

Optimum Checking Rate

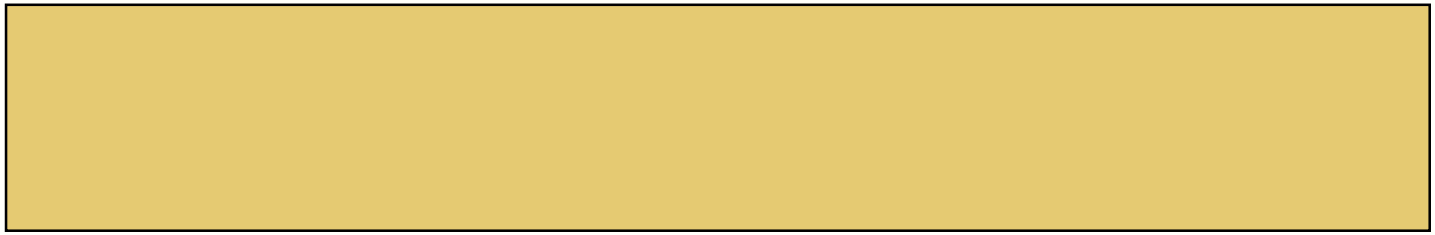


Raytheon, CMU/SEI-95-TR017

- The most effective individual speed for 'checking a document against all related documents' in page/hr
- Not reading speed, but rather correlation speed
- Failure to use it, gives 'bad estimate' for 'Remaining defects'
- 100~250 SLoC per hour
- 1 page of 300 words per hour ("logical page")

Optimum checking rate

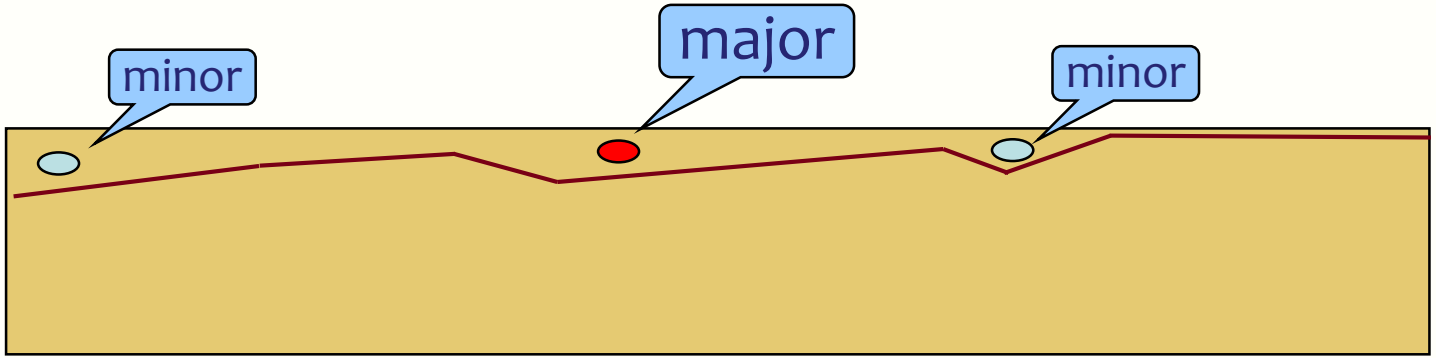
Ref. Dorothy Graham



Here's a document: review this (or Inspect it)

Review “Thoroughness”?

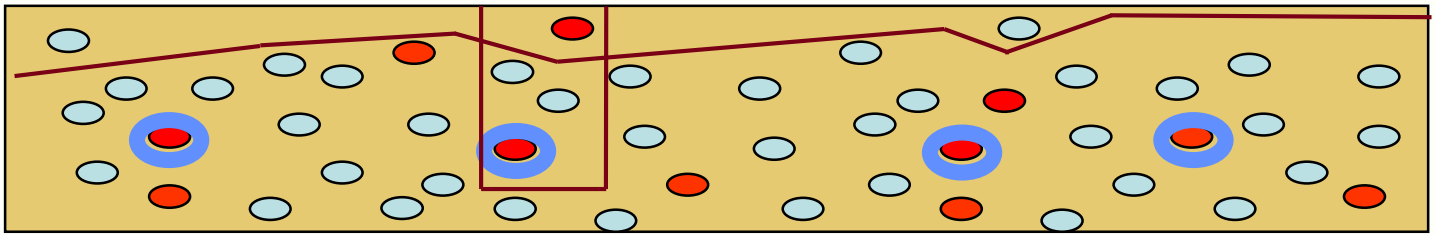
Ref. Dorothy Graham



- Ordinary review
  - Find some defects, one Major
  - Fix them
  - Consider the document now corrected and OK …

Inspection Thoroughness

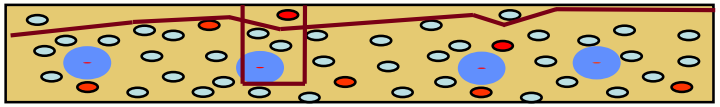
Ref. Dorothy Graham



- Inspection can find deep-seated defects
- All of that type can be corrected
- Needs optimum checking rate

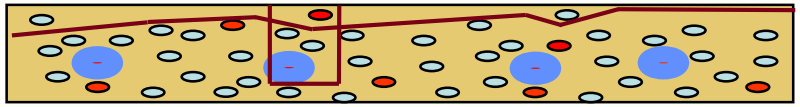
- In the above case we are clearly taking a sample
- In the "shallow" case we were also taking a sample, *however, we didn't realize it !*

Gilb/Graham Inspection

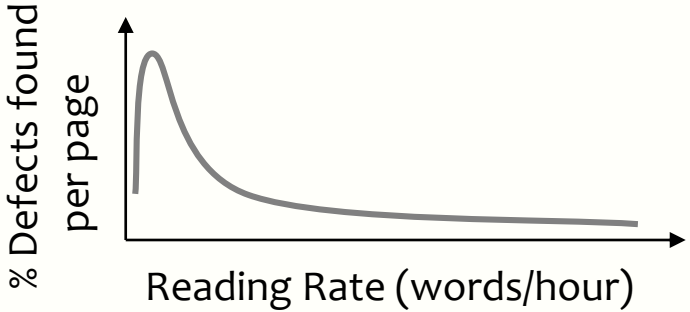


- Gilb/Graham inspection differs from other types of inspection in some or all of these ways:
  - Purpose:
    Quantifying quality, not searching for all defects
  - Controlled reading rate:
    The material being inspected is read very thoroughly in order to identify as many defects as possible (deep vs shallow sample)
  - Sampling:
    Only samples are inspected to optimize time and effort investment while maintaining the reading rate
  - Entry/Exit Criteria:
    Quantified entry and exit criteria used to guide the inspection effort
  - Rules:
    Written rule sets used to locate and classify defects

Gilb/Graham Concepts
Reading Rate



- Default recommended reading rate is one logical page per hour, 'slower' than in many other inspection methods
- This ensures adequate time to locate the vast majority of latent defects in the specification
- Supporting documents, rules, etc. can be read at any speed



Read too fast and you will miss most of the defects!

Exit criteria:
estimating remaining majors (after fixing)

Entry
Planning
Kick-off
Checking
Logging
Brainstorm
Edit
Follow-up
Exit

- You are about to Inspect your own document
- What is acceptable exit level?
  - 1000 estimated Major defects remaining per page ?
  - 100 ?
  - 10 ?
  - 1 ?
- What exit criteria will you use today?
  - I will accept no more than _____ estimated remaining major defects per page
- How much %% of defects do you think you'll find?
  - I will find _____ % of the defects

Undetected defects

Entry
Planning
Kick-off
Checking
Logging
Brainstorm
Edit
Follow-up
Exit

- Defects present but not yet detected by Inspection

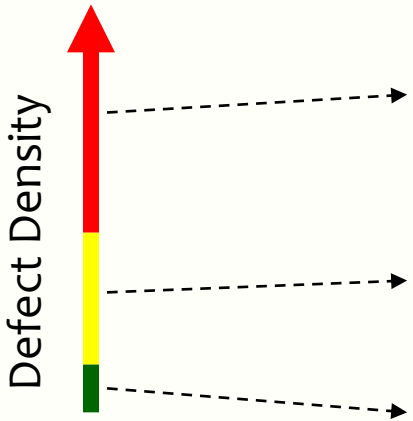
Mature Inspection process	undetected defects	yield	99% yield
Pseudo code	20%	80%	
Module and interface design	12%	88%	
Source code	40%	60%	7 x
Immature Inspection process			
All documents	70%	30%	12 x

(Lindner 1992)

Exit Criteria

Entry
Planning
Kick-off
Checking
Logging
Brainstorm
Edit
Follow-up
Exit

Once the quality level of a document is known, there are three possible paths forward:



Well above exit criteria: Process failure! Recreate after training or process improvement

Somewhat above exit criteria: Rework or enlarge inspection sample

Meets exit criteria: Success! Exit

The F-check - How many times do you see the letter f ?

Federal Funds are the result of years of scientific study combined with the experience of years

(Deming)

Summary (so far)

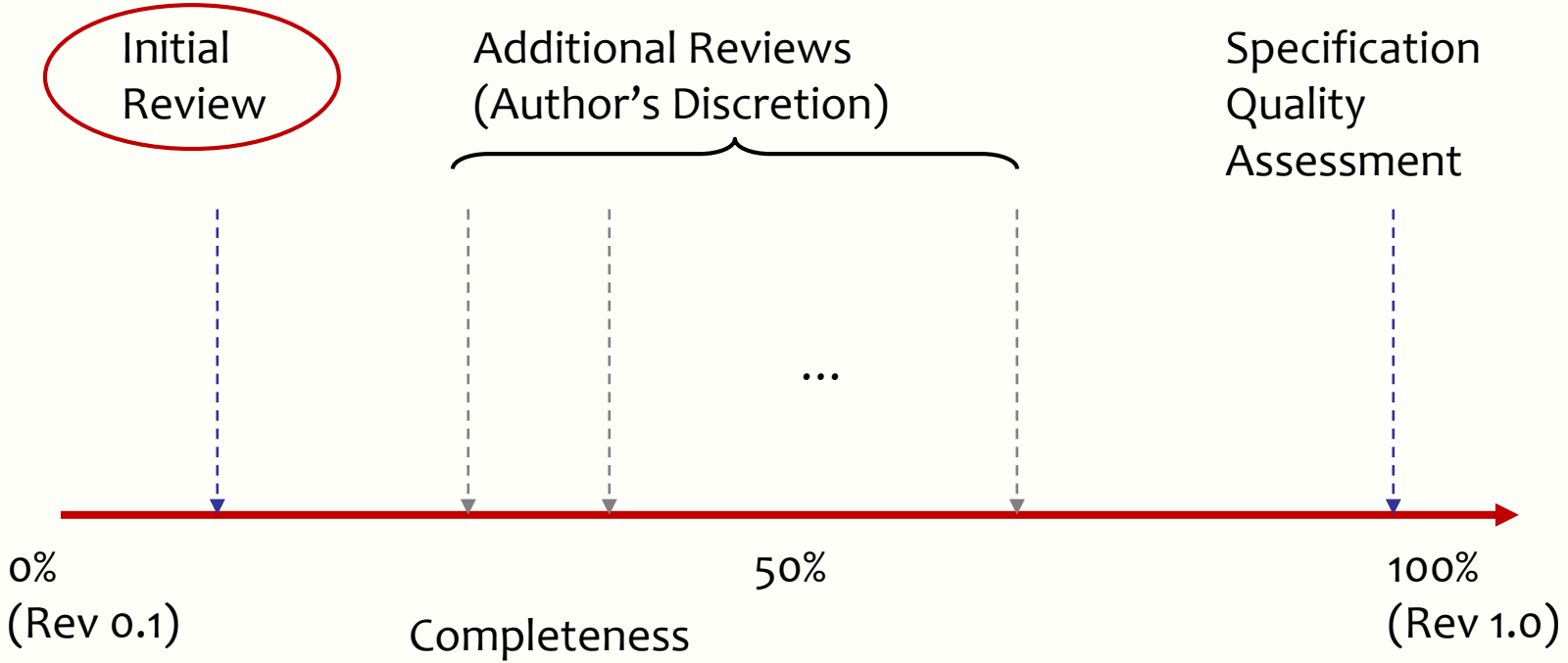
- Rules are the laws for documents
- Optimum checking rate
- Sampling
- Types of defects
- Exit criteria
- Measuring the benefit

- Isn’t that a heavy process ?

Early Inspections

Early Inspection

Prevention costs less than Repair



Initial Review

Purpose: Locating mistakes and tendencies that could lead to injecting major defects if not corrected

When: As soon as the author has completed a small representative portion of the specification, typically a few pages or 600-1200 words (e.g. few requirements)

Who: Individual or small team (1 or 2)

- Expertise in the subject matter
- Expertise in generic principles (such as requirements engineering, design, specific language)

What: Detailed review of the specification against rules and checklists for known error conditions and dangerous tendencies; formal inspection may be used

Duration: Because the sample is small, the initial review takes only 1-2 hr

The earlier it's reviewed, the more defects we can *prevent*

Initial Review Checklist

- ✓ Use a small team of experienced reviewers
- ✓ Schedule the review to minimize author waiting time
- ✓ Focus on issues that are or will cause major defects
- ✓ Avoid elements of style
- ✓ Be constructive at all times
- ✓ Focus on the work product, and never on the author
- ✓ Maintain confidentiality!
  The review is for the author's benefit

Reviewers: Your job is to make the author look like a hero

Case Study 1 - Situation

- Large e-business integrated application with 8 requirements authors, varying experience and skill
  - Each sent the first 8-10 requirements of estimated 100 reqs per author (table format, about 2 requirements per page including all data)
  - Initial reviews completed within a few hours of submission
  - Authors integrated the suggestions and corrections, then continued to work
  - Some authors chose additional reviews; others did not
  - Inspection performed on document to assess final quality level

Case Study 1 - Results

Average major defects per requirement in initial review	8
Average major defects per requirement in completed document	3

- Time investment: 26 hr
  - 12 hours in initial review (1.5 hrs per author)
  - About 8 hours in additional reviews
  - 6 hours in final inspection (2 hrs, 2 checkers, plus prep and debrief)
- Major defects prevented: 5 per requirement in ~750 total
- Saved 5 x 750 x 10 hr = 37500 hr / 3 = 12500 x $50 = $625000

Why Early Inspection Works

- Many defects are repetitive and can be prevented
  - Early review allows an author to get independent feedback on individual tendencies and errors
  - By applying early learning to the rest (~90%) of the writing process,
    many defects can be prevented
  - Reducing rework in both the document under review and all downstream work products

Testers can use it as well !

Case Study 2 - Situation

- A tester's improvement writing successive test plans:
  - Early Inspection used on an existing project to improve test plan quality
  - Test plan nearly "complete", so simulated Early Inspection
  - First round, inspected 6 randomly-selected test cases
  - Author notes systematic defects in the results, reworks the document accordingly (~32 hrs.)
  - Second round, inspected 6 more test cases; quality vastly improved
  - Test plan exits the process and goes into production
  - The author goes on to write another test plan on the next project…

Case Study 2 - Results

First round inspection	6 major defects per test case
Second round	0.5 major defects per test case

- Time investment: 2 hours in initial review, 36 hours total in inspection, excluding rework (2 inspections, 4 hrs each, 4 checkers, plus preparation and debrief)
- Historically about 25% of all defects found by testing, were closed as "functions as designed", still 2-4 hrs spent on each
- This test plan yielded over 1100 software defects with only 1 defect (0.1 %) closed as "functions as designed"
- Time saved on the project: 500 - 1000 hrs (25% x 1100 x 2-4 hrs)

Defect Prevention in action: First inspection of this tester's next test plan: 0.2 major defects per test case

Early Detection vs. Prevention

Denise Leigh (Sema group, UK), British Computer Society address, 1992:

An eight-work-year development, delivered in five increments over nine months for Sema Group (UK), found:

- 3512 defects through inspection
- 90 through testing
- and 35 (including enhancement requests) through product field use

After two evolutionary deliveries,
unit testing of programs was discontinued because it was no longer cost-effective

Nice job! Early detection has big benefits - BUT…

How many of the 3512 defects found in end-of-line inspections could have been completely prevented by Early Inspection?

Cost-effective defect prevention is the bottom line

Preparation: 15 mins in groups of 3

- Which document(s) are you Inspecting ?
  - Are there any source documents ?
- Which Rules are you checking against?
  - Generic Rule set or just top 3 ?
  - Any specific Rule sets for this document ?
    - e.g. requirements ? new ones for today ?
- Which page(s) will each of you be checking ?
  - All checkers check the same (most important) page ?
    - "logical" page, not necessarily one physical page (300 words text, 100 lines of code)
- Exit criterion?
  - How many Defects remaining ?

Checking

Individual Checking
Working alone
(tends to be very quiet)

- Check against your chosen Rules
- Check against source documents (if available)
- Look for Major defects
  - Rule violations with potentially large impact
- Note down what you have found (use issue log)
  - Majors only

Analysis

Are these reasonable for you ?
Any you wish to change ?
Why ?

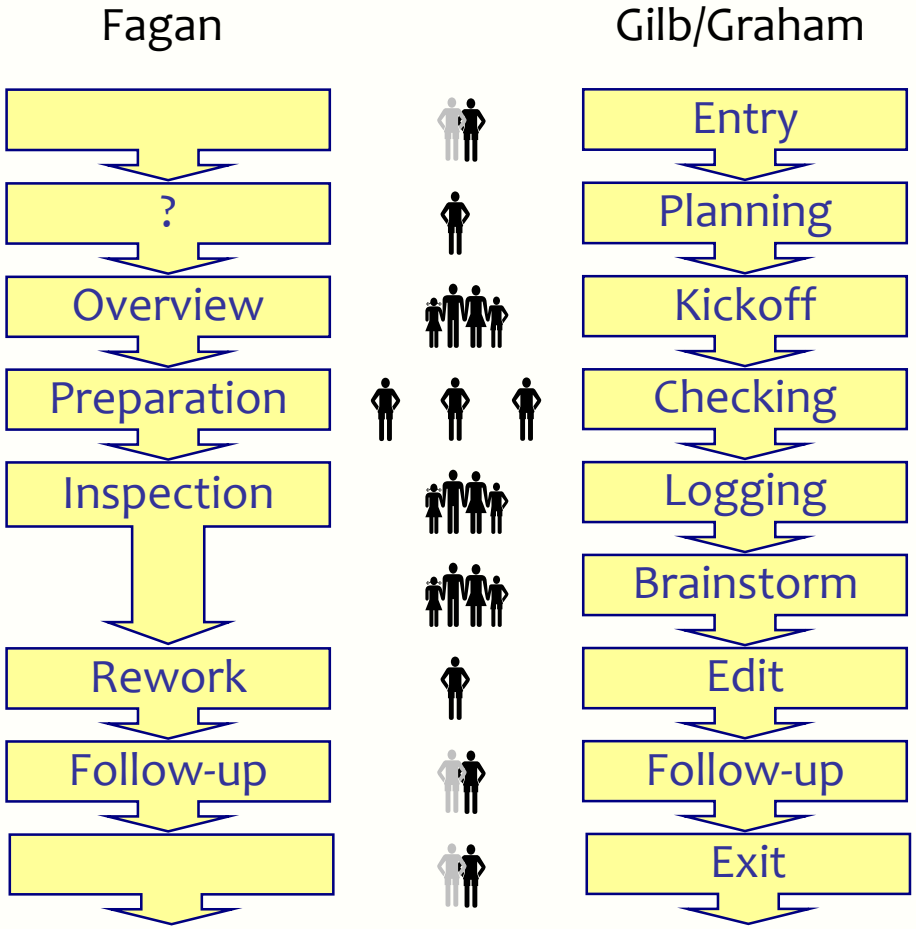
- Overlap of defects
  - Assume total = double maximum found by one
- Number fixed correctly
  - Assume 5 out of 6 will be fixed correctly
- Defects missed?
  - Assume we have found one third
    (based on observed effectiveness of new Inspectors)
- Chance of a defect causing a problem
  - Assume one third of defects will cause loss
- Average loss from a major defect
  - Assume nine hours

Report results

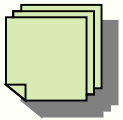
- Information from each group:
  - Type of document
    (e.g. requirements, functional specification, test plan, code)
  - Total size of document (in pages)
  - Number of pages Inspected (main focus)
    (i.e. number of words divided by 300)
  - Number of major issues found
    By each individual checker
  - Total unique major issues
  - Major issues remaining
  - Potential time saved
  - Potential money saved

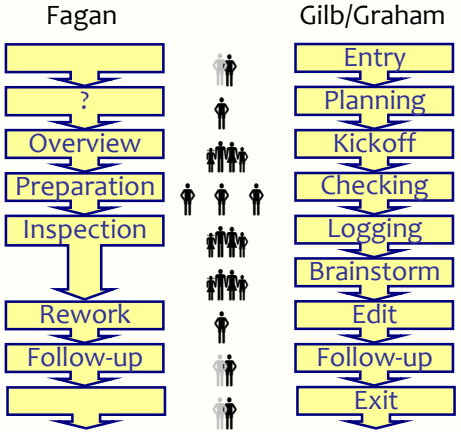
Fagan Inspections

Inspection Process Steps



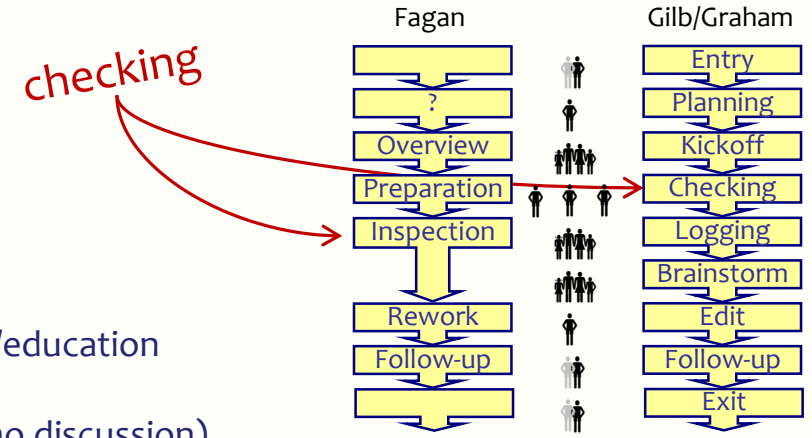
Fagan Inspections





- Objective: finding errors
- Based on publication in IBM Journal
- Emphasis on inspecting code
- If more than 5% reworked: 100% re-inspection
- If less than 5%: moderator decides
- All modifications better be inspected (even 1 line change)
- Most defects found during the meeting
- Typical defect list obtained used for prevention
- Typical defect list obtained used for next inspection
- Learn how to look for defects

Fagan Process

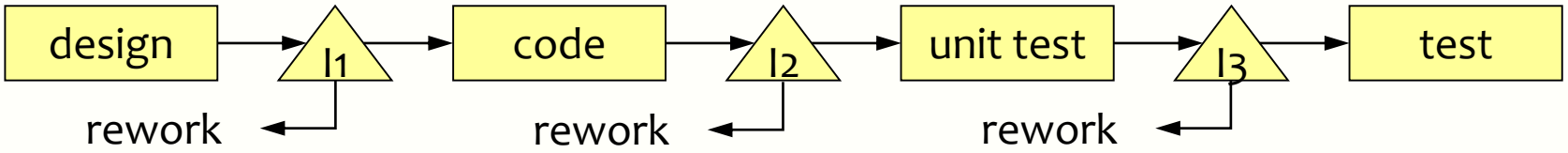


- Steps
  - Overview team Communication/education
  - Preparation individual Education
  - Inspection team Finding errors (no discussion)
  - Rework author Resolving errors and problems
  - Follow-up moderator Decision - analyse - process
- What to look for in Inspection
  Errors classified by type, ranked by frequency,
- How to look for presence of errors (education!)
- Analyse results for prevention

Fagan roles

- Moderator (specially trained)
- Designer (source document)
- Coder/Implementer (current document)
- Tester (testability)

Fagan experiment



Coding *productivity* change by Inspections:

- No Inspection: 100% (baseline)
- I1 only: 112%
- I1 and I2: 123%
- I3 had negative ROI, it was discarded

M.E. Fagan: *Design and Code Inspections to reduce errors in program development*
IBM Systems Journal, Vol15, No3, 1976

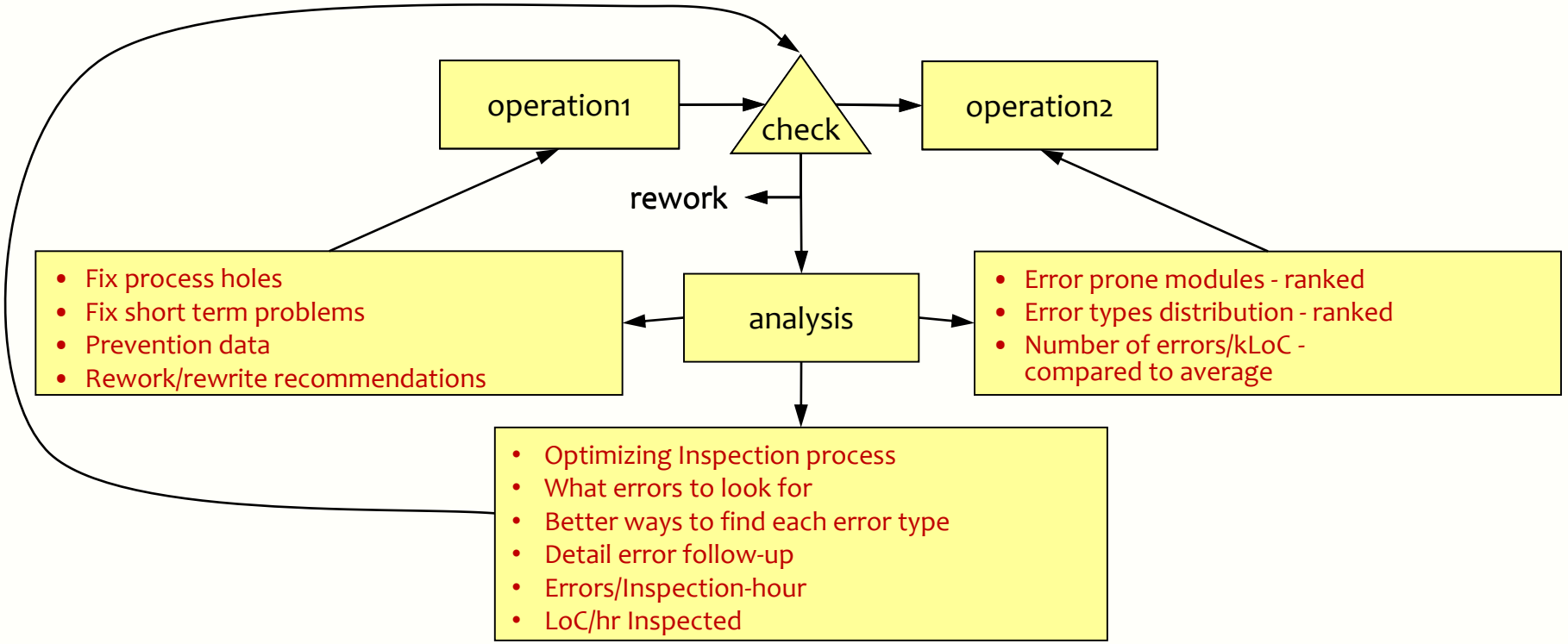
Errors found in Inspection and in Test

Table 1 Error detection efficiency

Process Operations	*Errors Found per K.NCSS*	*Percent of Total Errors Found*
Design		
I_1 inspection		
Coding	38*	82
I_2 inspection		
Unit test		
Preparation for acceptance test	8	18
Acceptance test	0	
Actual usage (6 mo.)	0	
Total	46	100

*51% were logic errors, most of which were missing rather than due to incorrect design.

Prevention and knowledge building (ref Fagan)



Cleanroom Inspections

Cleanroom expectations

NASA Satellite control system

- 40kLoC FORTRAN
- Testing found 4.5 defect/kLoC
- 60% of programs compiled successfully first time

IBM decision support program

- 107kLoC various languages, 50 person team
- Testing found 2.6 defect/kLoC
- 5 of 8 components: no defects found, no defects found in use

IBM tape drive controller, real time data stream control

- 86kLoC, C-code, 50 person
- Testing found 1.2 defect/kLoC

Ericsson Telecom operating system

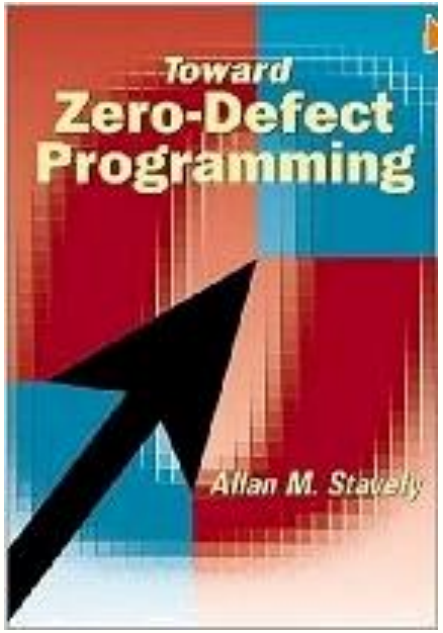
- 350kLoC, assembler and C, 70 person, 18 months
- Testing found 1 defect/kLoC

Cleanroom benefits

- Zero failures in field use
- Short development cycles
- Long product life

Quality is cheaper

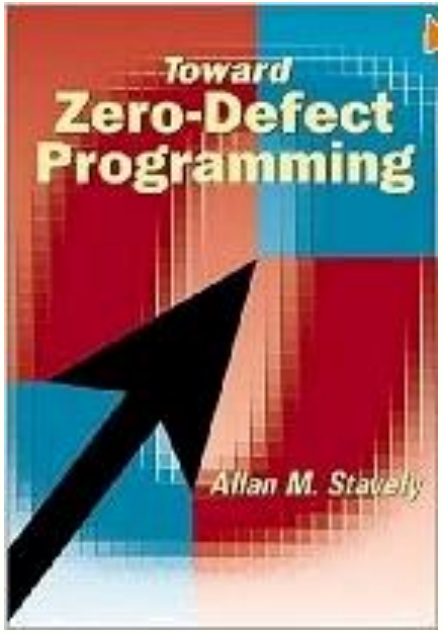
Cleanroom



Allan M. Stavely
Toward Zero Defect Programming

There are more books, but Stavely explains it very pragmatically

Cleanroom Software Development



- Design (Mathematical proof)
- Verification (by others)
- Implementation
- Verification (by others)
- No unit test
- Only Integration Test (by others)
  (Test is Running Code)

- *Verification* is for finding defects
- *Testing* is for *not* finding defects

Cleanroom fundamentals

- Design principle
  - Designers can and should produce systems free of defects *before* testing
- Testing principle
  - The purpose of testing is to *measure* quality
- Main development model
  - Incremental (Cleanroom) / Evolutionary (Gilb) / Cyclic (TSP)
    - Each increment is a working subset of the final product
    - Stable requirements for each increment
    - No eleventh hour integration

Cleanroom Principles

- Incremental development
  - User verifiable increments
- Team organisation
  - 4~8 people
- Formal methods of specification and design
  - Level of formalism varies even within project
- Intense review
  - Mathematical proof of correctness
  - Verifying individual control structures
- No unit test
  - No testing infinite number of paths, infinite combination of data
- Statistical testing as reliability measurement
  - Testing is not suitable for bug-hunting

Cleanroom Inspections

- The purpose of Inspection is to *eliminate* defects
- Exit criterion for design:
  - One design statement materializes as 3 to 10 code statements
- Checklists of typical errors we make
  - Listed in order of frequency
- No Unit Test - Developer does not 'try' software !
- Testing:
  - Finding as many of the remaining defects as possible
    - Too many errors discovered
      → previous steps are not being done properly
      → redo previous steps (do not "repair")

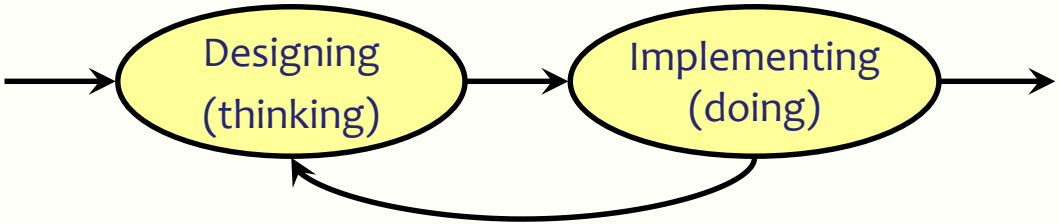
Cleanroom: *Slowest reviewer sets the pace*

- Wrong: Does anyone consider this incorrect? (dreamers won't answer)
- Better: Does everybody agree that this is correct? (attention is required)
- A team does not consider a verification condition proven until the slowest person to respond has expressed agreement

It is important to resist taking shortcuts here

Getting stuck somewhere ?

- Getting stuck in implementation? Back to the design !



- Getting stuck in Inspection? Back to the design !
- Getting stuck in Testing? Back to the design !

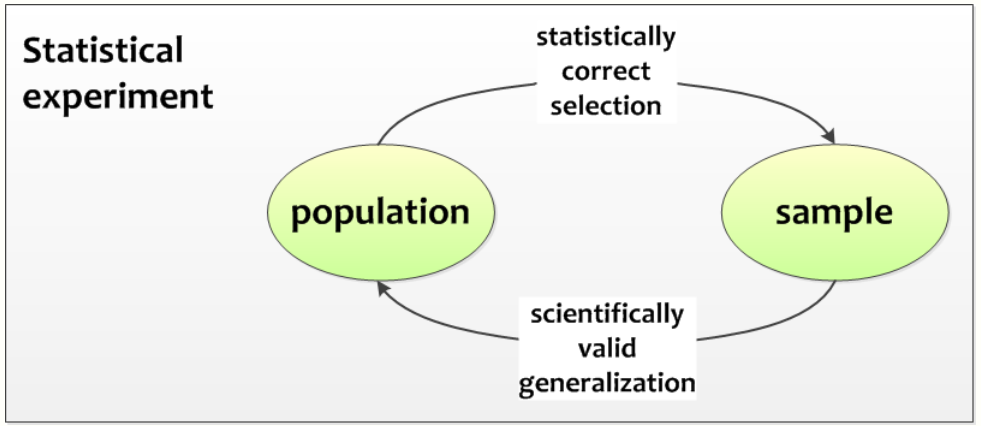
- *Why* do we get stuck ?
- Root cause analysis !

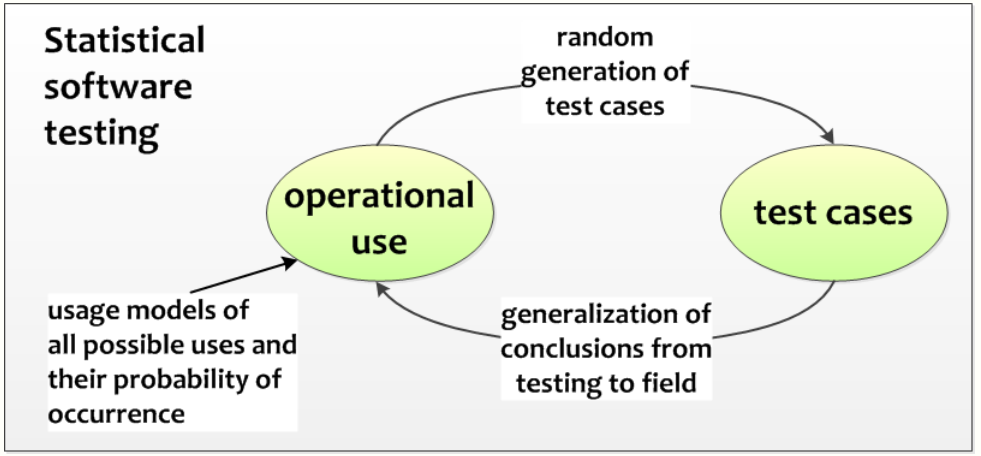
Testing in Cleanroom

- Testing is an important part of the process, but it is done only after verification (by Inspection) is successfully completed
- Testing is done:
  - Primarily to *measure* quality
  - Secondarily to find defects that escaped detection during verification
- Number of bugs per thousand lines of code <10 after verification, compilation and syntax checking
- Very good teams produce 2.3 defects per kLoC and reject code with 4 or 5 defects per kLoC
- *No attempt is done to try to salvage rejected code by debugging*
  - The code is sent back to the developers to be rewritten and reverified
  - Then it is tested as a completely new product
- Usage based testing – statistical testing
- Risk based testing – high risk, low probability will still be checked !

Statistical Testing

You need also other forms of testing !





No Unit Testing in Cleanroom

- We should avoid any kind of private testing, whether it is unit testing or some other kind
- We may experiment for various reasons,
  but we must resist the temptation to test our actual code

Philosophy behind Cleanroom

- To avoid dependence on costly defect-removal processes
- By writing code increments right the first time and
- Verifying their correctness *before* testing

(Linger, 1994)

Rules in Cleanroom

- Inspect also for attributes like: efficiency, simplicity, clarity, generality, portability, ease of verification, maintainability, ...
- People can make suggestions for improvement of any aspect of the program
  Valuable ideas will often emerge from the teams discussions
- The goal is to produce the best program possible: a program that can be verified with difficulty, but is more complicated than it needs to be, is not good enough
- If substantial revision appears necessary, the review process is stopped so that the team does not waste time verifying parts that will be changed anyway
- Usually, after some experience, this will rarely happen
- In a later meeting, the team will reverify the parts that were changed

Rules for Code

Tick the Code Rule Set

(Miska Hiltunen, 2007)

Extra baggage rules

DEAD	Avoid unreachable code
DRY	A comment must not repeat code
INTENT	A comment must either describe the intent of the code or summarize it
ONE	Each line shall contain at most one statement
UNIQUE	Code fragments must be unique

Tick the Code Rule Set

(Miska Hiltunen, 2007)

Missing info rules

DEFAULT	A ‘switch’ must always have a ‘default’ clause
ELSE	An ‘if’ always has an ‘else’
MAGIC	Do not hardcode values
PTHESES	Parenthesize amply
TAG	Forbidden: marker comments
ACCESS	Variables must have access routines
HIDE	Direct access to global and member variables is forbidden

Tick the Code Rule Set

(Miska Hiltunen, 2007)

Chaos-inducers

CALL	Call subroutines where feasible
NAME	Bad names make code bad
RETURN	Each routine shall contain exactly one ‘return’
SIMPLE	Code must be simple
FAR	Keep related actions together
DEEP	Avoid deep nesting
FOCUS	A routine shall do one and only one thing

Tick the Code Rule Set

(Miska Hiltunen, 2007)

Risky assumptions

CHECK-IN	Each routine shall check its input data
NEVERNULL	Never access a ‘NULL’ pointer or reference
NULL	Set freed or invalid pointers to ‘NULL’
CONST 1ST	Put constants on the left side in comparisons
ZERO	Never divide by zero
PLOCAL	Never return a reference or pointer to local data
ARRAY	Array accesses shall be within the array
VERIFY	Setter must check the value for validity

Tick the Code Speed

(Miska Hiltunen, 2007)

Rule	Call	Check-In	Dead	Deep	Default	Dry	Else
Ticks/hr	46	82	45	76	11	53	322
Rule	**Hide**	**Magic**	**Name**	**NeverNull**	**Tag**	**Unique**	
Ticks/hr	186	516	93	90	18	20	

- Average number of ticks found per hour per rule
- Software developers could find this many violations in one hour in the code they produce
- 144 developers checked for 108h to create the data

Draft Rule Set for Java

(Sybren Stüvel, 2007)

SIMPLE	Code should be as simple as possible, but not simpler
DOCUMENT	Documentation should be such that a developer who's unfamiliar with the code can still understand the reasoning behind it
CORRECT	Naming and documentation must be correct
CONDITIONAL CORE	Core functionality of a method should be outside any conditional block
EARLY	Return as soon as you can from a method. Assigning to RETURN a temporary variable and returning that variable usually results in overly complex code
EXCEPTIONS	Use exceptions to signal an error condition Don't return null to signify an error

Draft Rule Set for Java

(Sybren Stüvel, 2007)

REUSE	Use common library functions where applicable At least take a look at StringUtils and ListUtils (Spring framework) and ArrayUtils (Apache Commons) Use XStream for parsing and generating XML
EQUALS	To compare objects use their equals method
MAGIC	Define constants in one place, and use them
REFER	Use @see and @link in JavaDoc to refer readers to relevant other locations
READABLE	Ensure the code is easily readable
SENSIBLE TEST VALUES	Test values should be sensible
EARLY JAVADOC	Write a method's JavaDoc before writing actual code. This gives a method its scope
REVIEW TESTS	Start by reviewing the unit tests

MISRA C

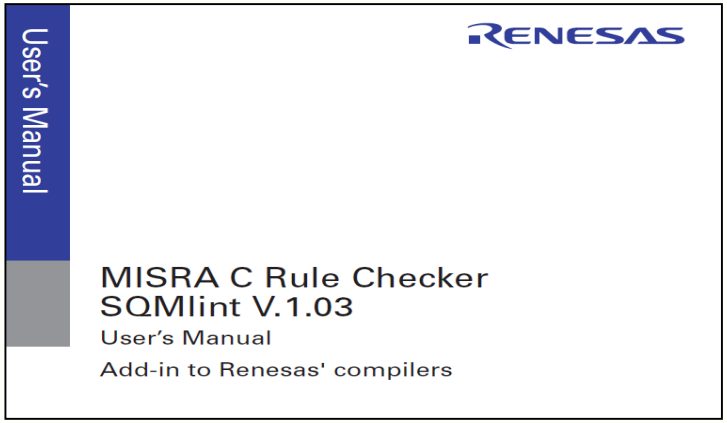
version	rules
MISRA C 1998	127
MISRA C 2004	142
MISRA C 2012	143
2012 Amendement 1 (2016)	156
2012 Amendement 2 (2020)	158

- MISRA: Motor Industry Software Reliability Association
- Providing a set of guidelines to restrict features in the ISO C language of known undefined or otherwise dangerous behaviour
- MISRA C:1998, 93 are required and the remaining 34 are advisory
  - Rule 104 (required): Non-constant pointers to functions shall not be used
- MISRA C:2012 Amendment 2
  - Rule 1.4: Emergent language features shall not be used
  - Rule 21.21: The Standard Library function system of <stdlib.h> shall not be used

MISRA C

Rule 59 (required): The statement forming the body of an "if", "else if", "else", "while", "do ... while", or "for" statement shall always be enclosed in braces

```
if (x == 0)
{
y = 10;
z = 0;
}
else
y = 20;
z = 1;
```



MISRA C

Rule 33 (required):
The right hand side of a && or || operator shall not contain side effects

```
if ((x == y) || (*p++ == z))
{
/* do something */
}
```

```
if (x == y)
{
doSomething = 1;
}
else if (*p++ == z)
{
doSomething = 1;
}

if (doSomething)
{
/* do something */
}
```

MISRA C

Motor Industry Software Reliability Association

```
a[i] = ++i;
```

happens once in every 7,000 lines in C

```
c == d;
```

```
if (c=d)
{
}
```

Put on checklist

CR - PR - RI

Database

CR/PR/RI Database

- Change Requests
  CR: customer pays
- Problem Reports
  PR: you pay
- Risk Issues
  RI: prevention → nobody pays !

- Where, what, when, who
- Urgency, severity
- Classification
- Status

Focus on "Repair"

Focus on Prevention

- Where caused and root cause
- Where should it have been found earlier
- Why not found earlier
- Prevention plan
- Analysis tasks defined and put on Candidate Task List
- Prevention tasks defined and put on Candidate Task List
- Check lists updated for finding issues easier, in case prevention doesn't work yet

More

www.malotaux.eu/?id=booklets

1 Evolutionary Project Management Methods (2001)
Issues to solve, and first experience with the Evo Planning approach

2 How Quality is Assured by Evolutionary Methods (2004)
After a lot more experience: rather mature Evo Planning process

3 Optimizing the Contribution of Testing to Project Success (2005)
How Testing fits in

3a Optimizing Quality Assurance for Better Results (2005)
Same as Booklet 3, but for non-software projects

4 Controlling Project Risk by Design (2006)
How the Evo approach solves Risk by Design (by process)

5 TimeLine: How to Get and Keep Control over Longer Periods of Time (2007)
Replaced by Booklet 7, except for the step-by-step TimeLine procedure

6 Human Behaviour in Projects (APCOSE 2008)
Human Behavioural aspects of Projects

7 Evolutionary Planning, or How to Achieve the Most Important Requirement (2008)
Planning of longer periods of time, what to do if you don't have enough time

8 Help ! We have a QA Problem ! (2009)
Use of TimeLine technique: How we solved a 6 month backlog in 9 weeks

9 Predictable Projects - How to deliver the right results at the right time

RS Measurable Value with Agile (Ryan Shriver - 2009)
Use of Evo Requirements and Prioritizing principles

www.malotaux.eu/?id=inspections

Inspection pages

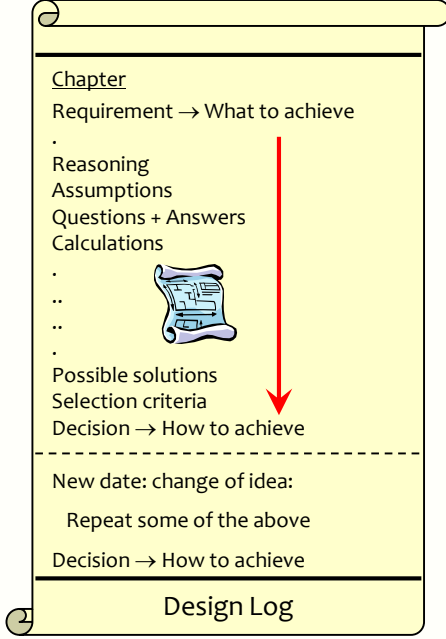
How to Improve the Result of Reviews and Inspections

Niels Malotaux

niels@malotaux.eu
www.malotaux.eu/conferences

Concept: DesignLog

- In computer, not loose notes, not in e-mails, not handwritten
  - Text
  - Drawings!
  - Chapter per subject
  - Initially free-format
  - For all to see
- All concepts contemplated
  - Requirement
  - Reasoning
  - Assumptions
  - Questions
  - Calculations
  - Possible solutions
  - Selection criteria
  - Choices:
    - If rejected: why?
    - If chosen: why?
- Implementation specification



Use the three rules on these Requirements

It shall be possible to easily extend the system's functionality
on a modular basis, to implement specific (e.g. local) functionality

It shall be reasonably easy to recover the system from failures,
e.g. without taking down the power

1. Unambiguous to the intended readership
   Two designers arrive at the same result
2. Clear enough to test
   Two testers get same result
3. Quantified quality
   All qualities shall be expressed quantitatively
   (element of unambiguousness)
4. No design mixed in requirements

Jet Case

The Jet Case

Introduce the following three rules for Inspecting a requirements document

Three Rules for Requirements:

1. Unambiguous to the intended readership
   Two designers arrive at the same result
2. Clear enough to test
   Two testers get same result
3. No design mixed in requirements (mark with 'D')
   Requirements: What the acquirer cares about: 'how good to be'
   Design: Set of decisions made by the developer: 'how to be good'

Defect

Explain the definition of a Defect

- A Defect is a violation of a Rule
- Note: If there are 10 ambiguous terms in a single requirement then there are 10 defects !

Severity

Explain:
- the definition of Major Defect
- the checkers must focus on finding Major Defects

- Major: a defect severity where there is potential of high loss later downstream (test, field)
- “10 lost engineering hours”

Exit?

Agree with the management team on a numeric exit condition:
Is 1,000 Majors per page OK ? 100, 10, 1 ?

- Exit Conditions:
  (document can go to next stage with little risk)
  Maximum 1 Major Defect / (Logical) Page

- Logical Page = 300 Non Commentary Words

The Job

- You have up to 15 minutes for checking
  One Requirements Logical page from the 82 pages document
- Count all Rule Violations → Defects
- Classify Major and minor

Report for Page 81

	Total	Major	Design
1.	24	15	5
2.	44	15	9
3.	55	20	4
4.	22	4	2

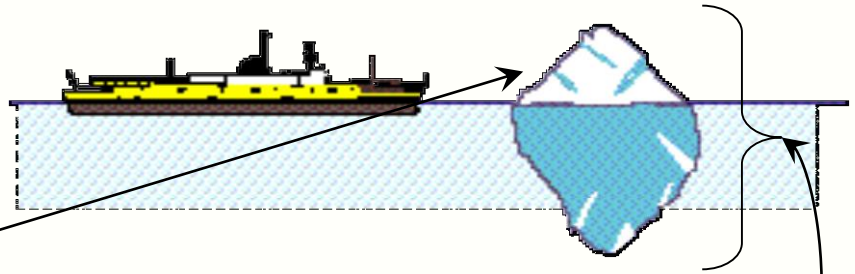
- Inspection results on requirements document, 4 managers

- Defect Density
  - Total for group 20 x 2 = 40 Majors assume are unique
  - If 33% effective, total in page = 3 x 40 = 120
  - Of which 2/3 or 80 were not yet found
  - If we attempt to fix the 40 found, and correctly fix 5/6, then 7 are failed fixes, so:
  - Total remaining after Inspection and editing = 80 + 7 = 87 Majors per page

Report for Page 82

	Total	Major	Design
1.	41	24	1
2.	33	15	5
3.	44	30	10
4.	24	3	5

- Inspection results on requirements document, 4 other managers



- Defect Density
  - Total for group 30 x 2 = 60 Majors assume are unique
  - If 33% effective, total in page = 3 x 60 = 180
  - Of which 2/3 or 120 were not yet found
  - If we attempt to fix the 60 found, and correctly fix 5/6, then 10 are failed fixes, so:
  - Total remaining after Inspection and editing = 10 + 120 = 130 Majors per page

Extrapolation to Whole Document

- Page 81: 120 Majors/page
- Page 82: 180 Majors/page
- Average 150 Majors/page x 82 page = 12300 Majors in the document.

- If a Major has 1/3 chance of causing loss (12300 / 3 = 4100)
  and each loss is ≈10 hours
  then total project Rework cost is about 41000 hours loss
- (This project was over a year late and expected one more year)
  - 1 year = 2000 hour x 10 people = 20000 hours