I would like to talk about how we used Review and Inspection techniques to achieve less expected, but very interesting outcomes.

I will show some cases as illustration.

Niels Malotaux

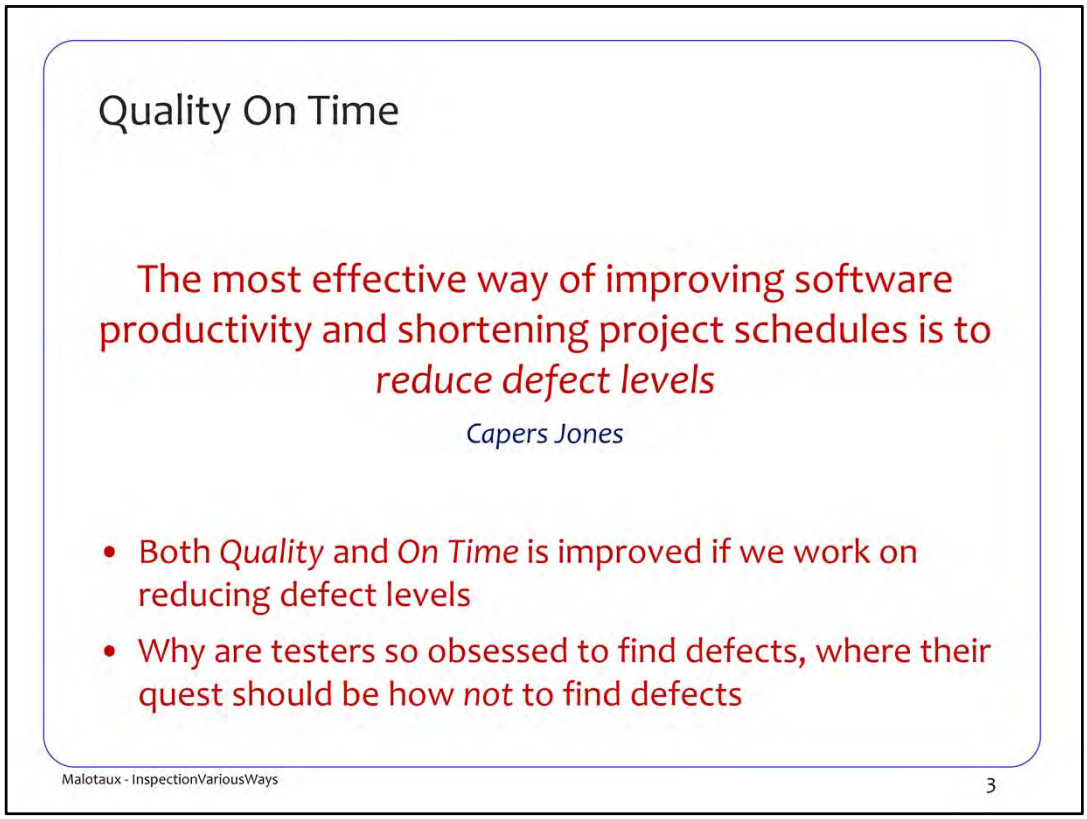
Graduated **Electronics** at Delft University of Technology in **1974**

Army service at the Dutch Laboratory for Electronic Developments for the Armed Forces, designing computer systems

Philips Electronics – Application support for microcomputer systems design

Malotaux - Electronic Systems Design - : developing electronic systems for clients products

Now: **N R Malotaux - Consultancy:** coaching projects to deliver successfully and much faster



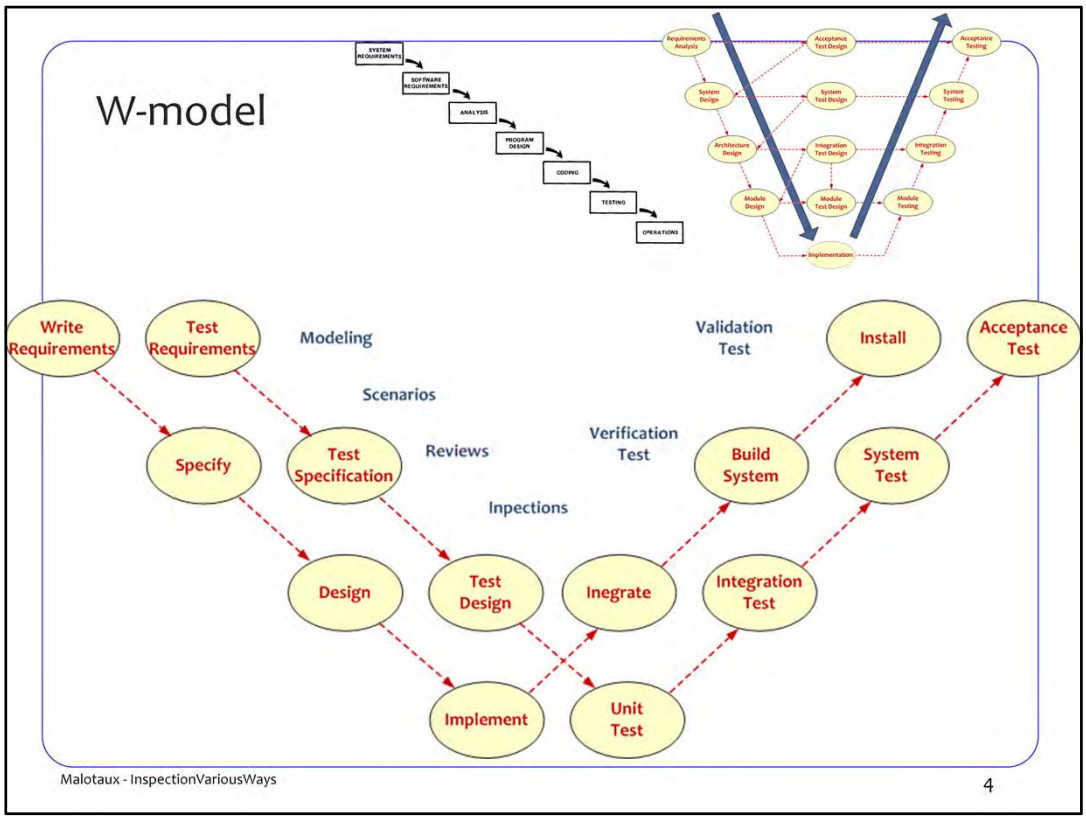
Capers Jones measures software productivity and many other software development metrics all his life.

The claim on this slide is not just an opinion. It's based on actually measuring the performance of thousands of software development projects, including Agile software development.

If Capers talks about 'software productivity', he doesn't just mean delivering software (as in the Agile Manifesto). He means delivering software that works at the required quality level.

So, how can we reduce defect levels?

Reviews and Inspections are known to be very useful techniques to quickly reduce the number of defects injected.



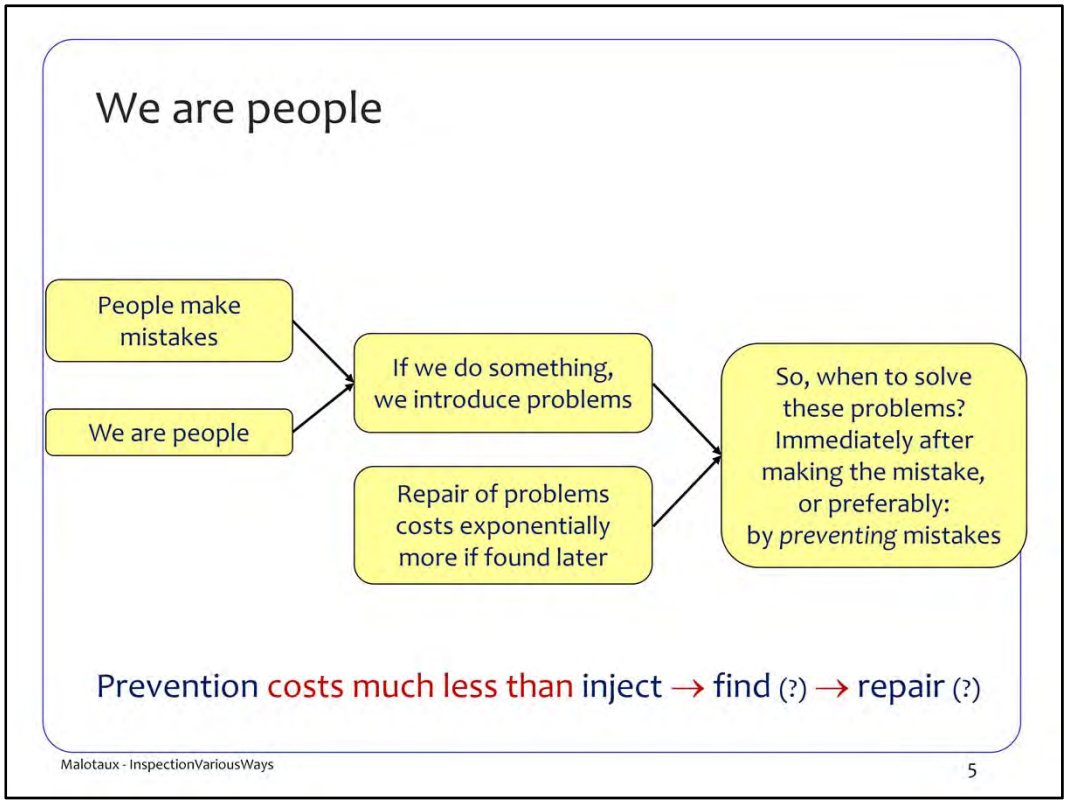
In the Agile world, the Waterfall and derived models are often seen as bad. All models are wrong, some are useful. If they're useful, we should use them.

These models are still applicable to every Sprint. After all, we have to determine what value we should deliver by the end of the Sprint (requirement), how we can and are going to realize it (design), how we implement it (coding), integrate it, test it, and deliver.

For QA the challenge is not to test only the code and find bugs, but to help development to prevent producing problems in the first place. By reviewing the requirement, the design, and the implementation, so that the final test can conclude that it simply works as it is supposed to work.

Inspection is a special form of review.

The V-model is actually a folded Waterfall, where the most expensive issues (requirements issues) are found at the end. The W-model shows that we should find any remaining issues as soon as they are created, rather than when they cause trouble later. The requirement, the design, the code: they're all different manifestations of the same product. However, only the code can be run to check that it does what it is supposed to do. That's what we usually call 'testing'. Before we have code, there are other techniques to check that the product is right, like Modelling, Scenarios, Reviewing, Inspecting. In these areas QA can prove its value as well.



Conventional wisdom says that "people make mistakes".

We are people, so if we are preparing some result, we're making mistakes, causing problems.

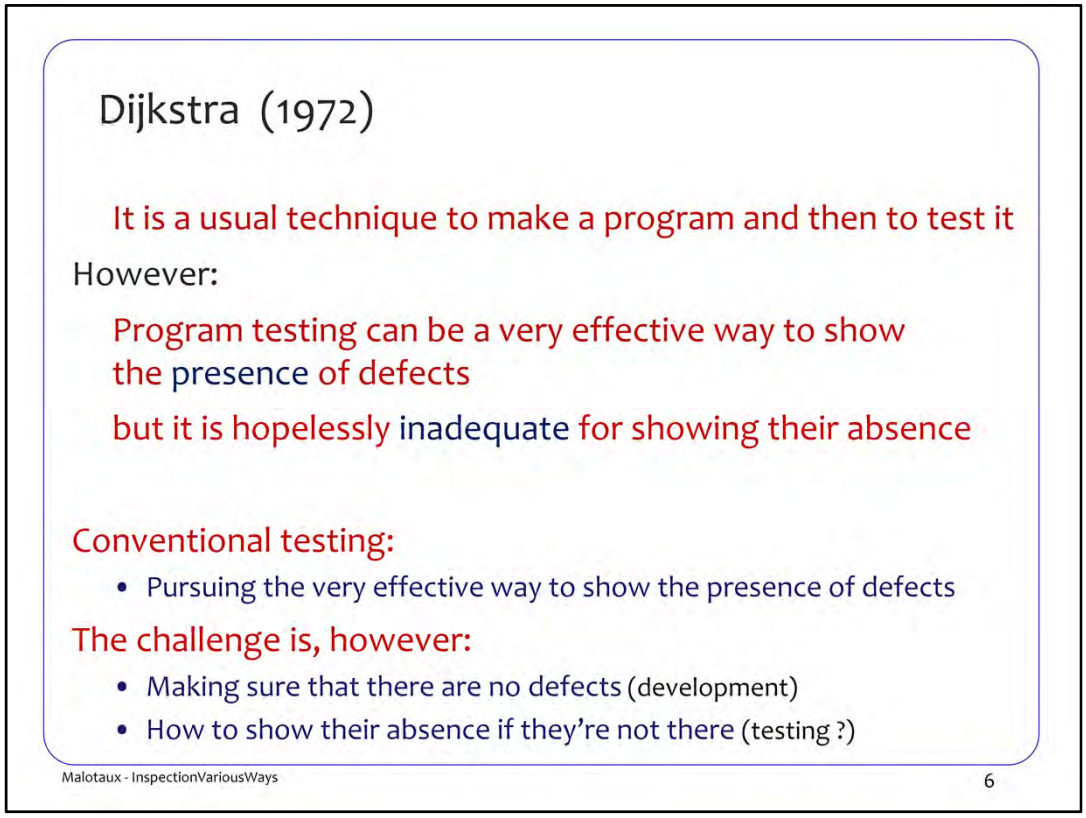
We also have learnt that the longer a problems stays in our result, the costlier it is to find and fix it, if we can and will fix it at all.

Combining these two issues, when should we find and solve the problems caused by the mistakes we, as human beings, are making during our work?

Answer: Immediately after injection of the defect, instead of much later at a 'final test', or even worse, after we caused a hassle to the users.

We can do even better: preferably we should *prevent* issues to be created in the first place: by not making the mistakes at all.

Prevention costs much less than first injecting a defect, then hoping to find it, and then hoping that we can repair it properly, while we know that not all defects are found (testers are also human, aren't they?) and while we know that not all defects found will be repaired properly (usually done under stress), or even cause other defects to emerge or to be introduced.



This is a quote from Edsger Dijkstra, who called himself the first programmer of the Netherlands.

As testing is 'highly inadequate for showing the absence of defects', it seems that testers decided to go for showing the presence of defects, as this is what they can do by testing very effectively. Why so much emphasis on defects while what we should pursue is *no defects*, as that is win-win: what the users are entitled to expect, and Capers Jones has measured to be less costly for us to produce.

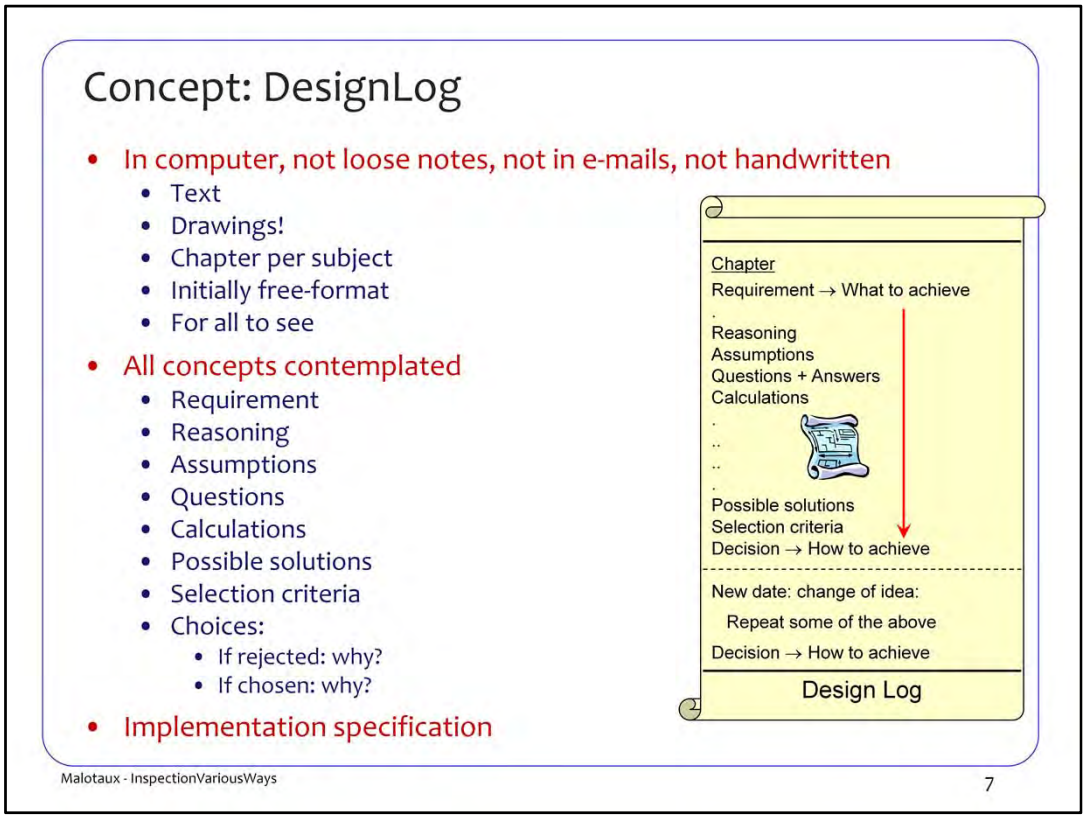
But what should the testers do if they only know how to show the presence of defects, once we have educated the developers not to produce defects anymore? Note: this is not as difficult as some people will try to convince you of!

That's the real challenge: how can we, together with development, make sure there are no defects. Isn't that what we should be after?

I assume that we don't *want* to deliver defects to the users of our software, do we?

Inspections catch issues much earlier, better and much less costly than testing.

Inspections also find different issues, issues that testing would hardly find at all.



For recording our design considerations and decisions, we use the concept of the 'DesignLog'.

When I started my career at Philips Electronics in 1976 (at the same time Philips started to sell its first microprocessor), we got a notebook to note our thoughts, experiments and findings chronologically. It was difficult, however, to retrieve an idea I had several weeks before, because it was buried in many pages of hardly readable handwriting.

Nowadays we can use a word processor, add pictures, organize by subject rather than chronologically, and search through the text. We log our thoughts in chapters, which start with what we have to achieve (requirement), end with how we think we will achieve it (implementation specification), with in between the reasoning, assumptions, questions and answers, possible solutions, decision criteria and the selected solution (design).

If I see design documentation, this often only shows what people decided to do, rather than also recording why and how they arrived at this decision.

The DesignLog should be reviewed to find possible issues before we start the implementation. Because the choices and design are well documented, and minimum time is lost in the maintenance of the software. Maintenance often being the largest portion of the cost of deployment!

People ask me: "How much detail do I need to put in the DesignLog?"

My answer is: "You'll have to find out yourself. One of the requirements for the quality of the DesignLog is: 'If someone has to change something in the software one year later, he should be up and running within a day.' "

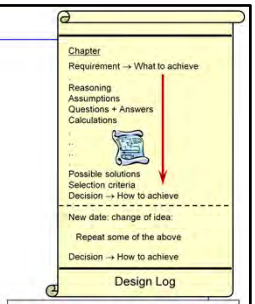
When QA asks development to review the DesignLog, if there is one they can review and also use this information to define and optimize the test-approach. If there is none, this is a good time to introduce the concept.

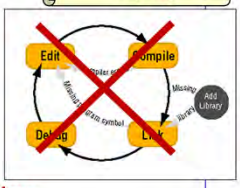
Case: In the pub

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!*

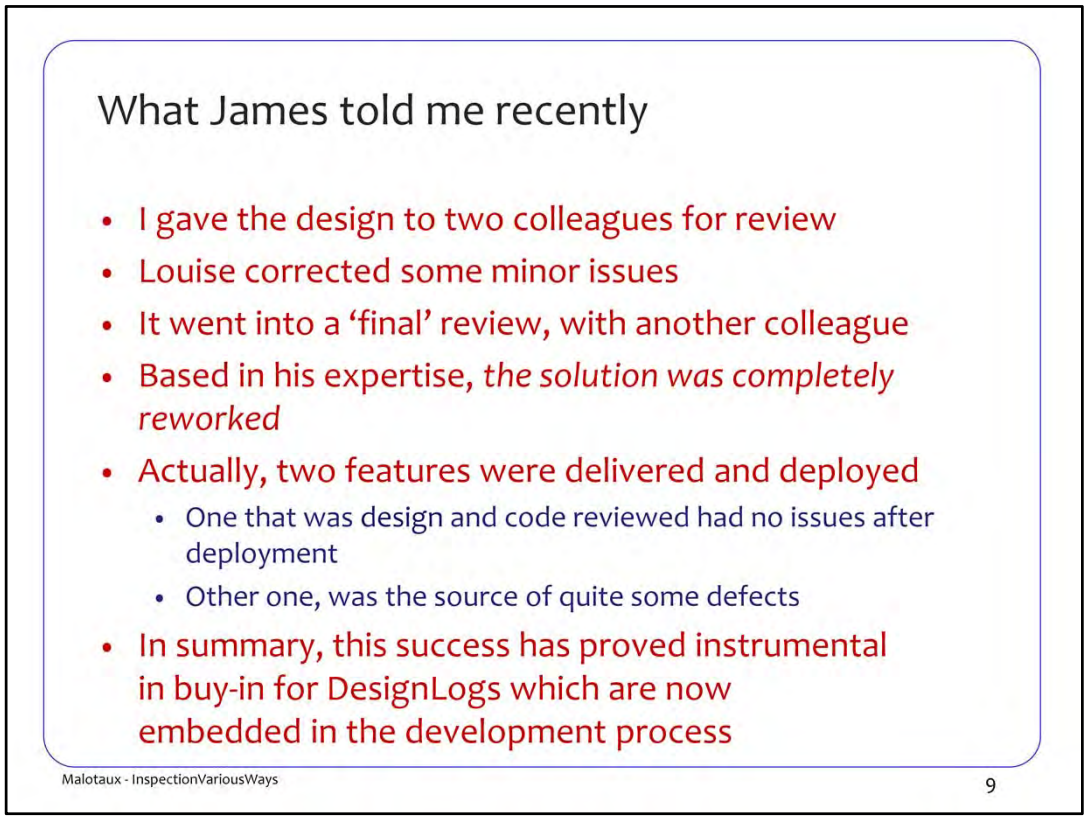
This case happened just a few months ago, see the text on the slide.

It's always nice to experience that the techniques that worked for me and for many others in the past, still work today. Many old techniques never get out of date.

We see, however, that it's not so easy to convince people to do something that seems counter-intuitive: going back to the design rather than grinding on in code and leaving a lot of dangerous scars in the process.

Delivering quality often needs counter-intuitive measures.

On the next slide we'll see how a review even caused the whole design to be changed!

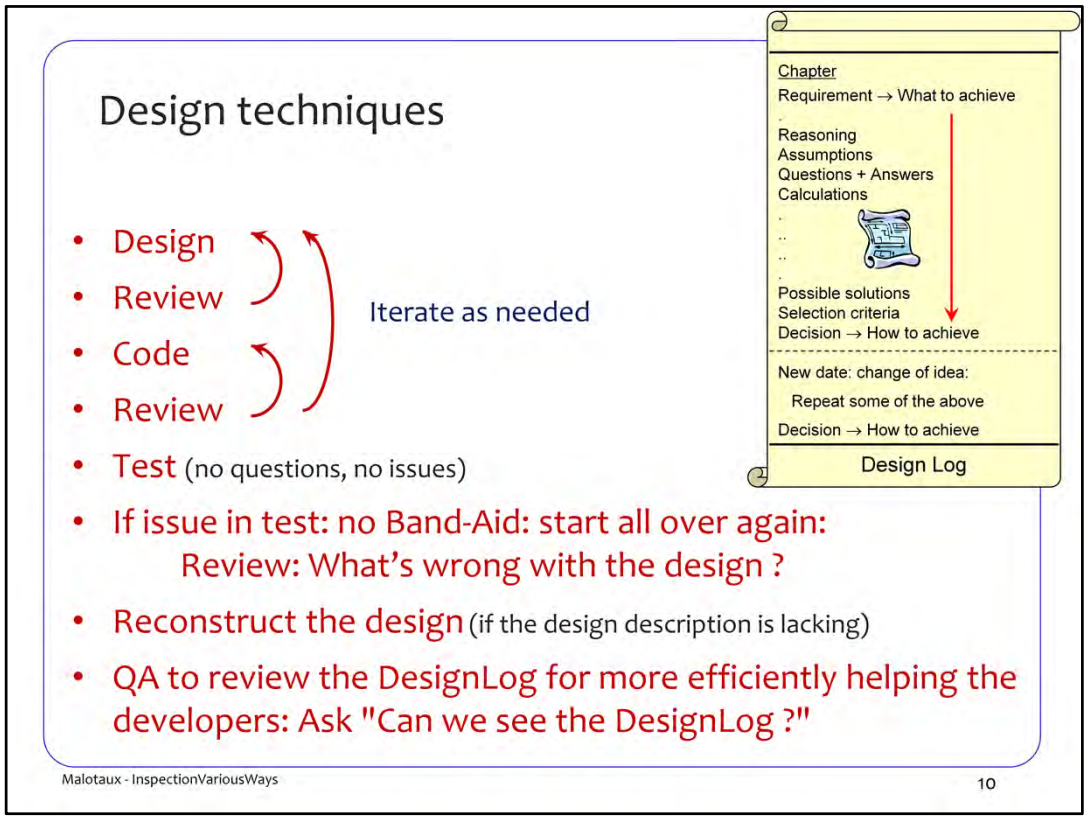


James explained me later more interesting details of this case:

- There were two features required for a release, one of which was on the critical path and placed the delivery at risk
- I saw an "opportunity" for a Design - "*prevention, rather than fixing*" and also an opportunity to encourage documentation
- Because Louise struggled a bit with the design (not many people in software have been educated in how to design), we Timeboxed the initial draft
- I emailed it to two colleagues to review: "please review, assuming you will code-review the implementation, and based on this DesignLog you know what the implementation you will have to code-review will be"
- Louise emailed me in a panic that if she knew it would be reviewed, she would have written something different. I said - "no, do nothing yet - review first and then update with your new understanding and feedback. Reason: the next draft will be better."
- For the next review, another colleague who was not previously available was invited.
  At the meeting where I expected the DesignLog to be approved with minor modifications and to get estimates for the work involved, the Design was *totally reworked*
- We agreed the new Design was better than the original ideas
- Actually, two features were delivered and deployed
    - The one that was designed, reviewed, coded, and code reviewed had no issues after deployment
    - Another one, which was done in the 'traditional' way, was the source of quite a few defects
- In summary, this success has proved instrumental in buy-in for DesignLogs which are now embedded in the development process

Using Inspection (Review) in various ways:

- The review caused them not to implement a bad solution
- If they would have implemented the original solution, they probably wouldn't have found out until much later
- The whole process allowed them to deliver well before the deadline rather than after.



In the previous case we saw the power of design – review – redesign – review – code – review – resulting in test and user not finding any issues.

For recording our design considerations and decisions, we use the concept of the 'DesignLog'.

When I started my career at Philips Electronics in 1976 (at the same time Philips started to sell its first microprocessor), we got a notebook to note our thoughts, experiments and findings chronologically. It was difficult, however, to retrieve an idea I had several weeks before, because it was buried in many pages of hardly readable handwriting.

Nowadays we can use a word processor, add pictures, organize by subject rather than chronologically, and search through the text. We log our thoughts in chapters, which start with what we have to achieve (requirement), end with how we think we will achieve it (implementation specification), with in between the reasoning, assumptions, questions and answers, possible solutions, decision criteria and the selected solution (design).

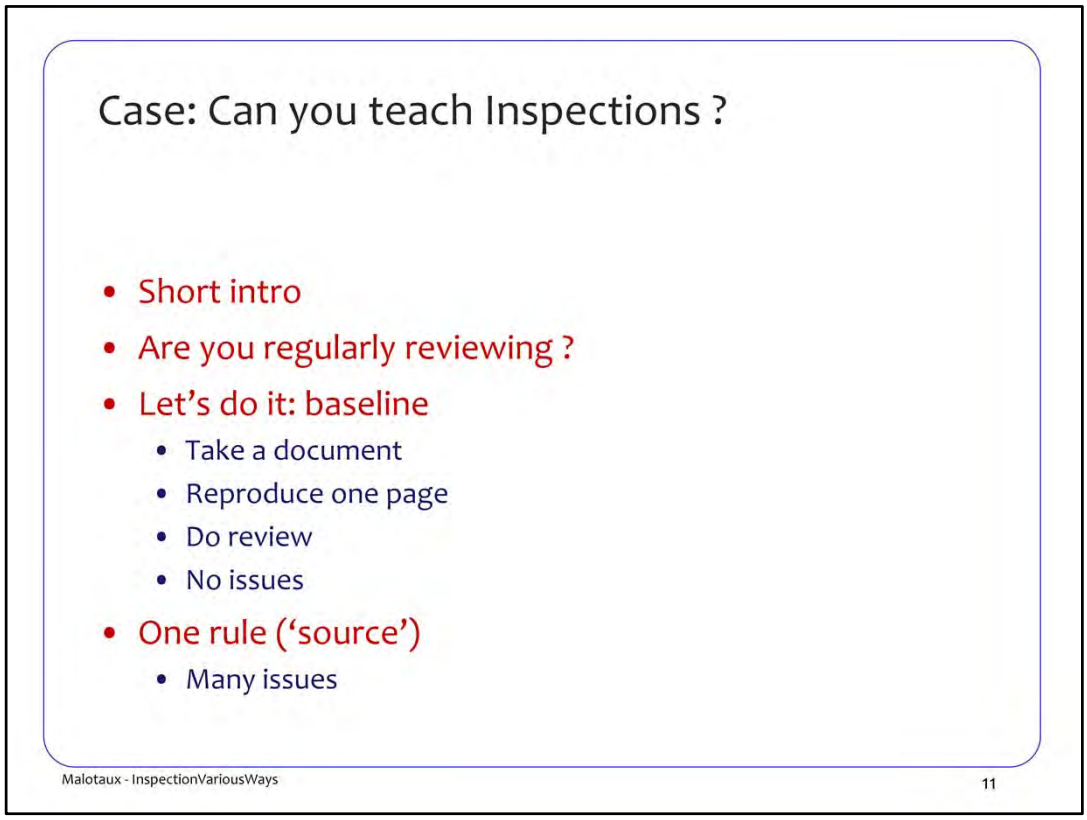
If I see design documentation, this often only shows what people decided to do, rather than also recording why and how they arrived at this decision.

The DesignLog should be reviewed to find possible issues before we start the implementation. Because the choices and design are well documented, and minimum time is lost in the maintenance of the software. Maintenance often being the largest portion of the cost of deployment!

People ask me: "How much detail do I need to put in the DesignLog?"

My answer is: "You'll have to find out yourself. One of the requirements for the quality of the DesignLog is: 'If someone has to change something in the software one year later, he should be up and running within a day.' "

When QA asks development to review the DesignLog, if there is one they can review and also use this information to define and optimize the test-approach. If there is none, this is a good time to introduce the concept.



In another case I was asked to teach Document Inspections to a group of developers. I gave a short introduction and then we did a baseline review. After all, most developers do reviews, don't they?

We selected a design document for some firmware datalog functionality in a controller, took one page from the document, made 20 copies of that page for everyone to review. They started reviewing and after some 10 minutes everyone seemed ready.
I asked about the issues found. Hardly any. Perhaps a typo or two.

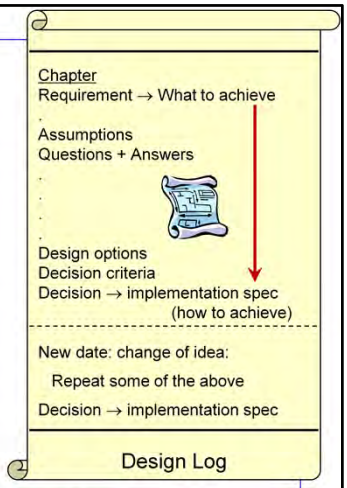
Then I introduced a 'rule' (In Inspections, we use 'rules', which are the 'laws' for making a useful document): "If we don't know the requirement of this design, how do we know that the design does what it should do and does *not* what it should *not* do?" I asked them to review once more.

After a short while, everyone's paper was full of remarks: This I cannot judge, that I cannot judge, because I don't know what was required and why this is the best solution. With the review they had found out *themselves* that there was a lack of knowledge what 'design' actually means. If I would have told them, they wouldn't have accepted it. Now they showed it to themselves. This was an interesting alternative use of the Inspection technique!

Explanation of the datalog environment

Results

- No code until design-log is reviewed
- You're delaying my project !
- Example
- Solution
- Thanks, you saved my project
- Now we can review to check the design before implementation
- Did I do the same ?
- Telling people to change: resistance
- How to let people change themselves …



We suggested the designer to make a DesignLog. Not to write even one line of code until the DesignLog was reviewed and found OK.

It took some time until the author understood how to make the DesignLog, but it led to an interesting conclusion: He decided not to implement the functionality in the controller firmware, because of some intricacies which could much better be solved in the PC software at the other side of the network, when analysing the datalog data.

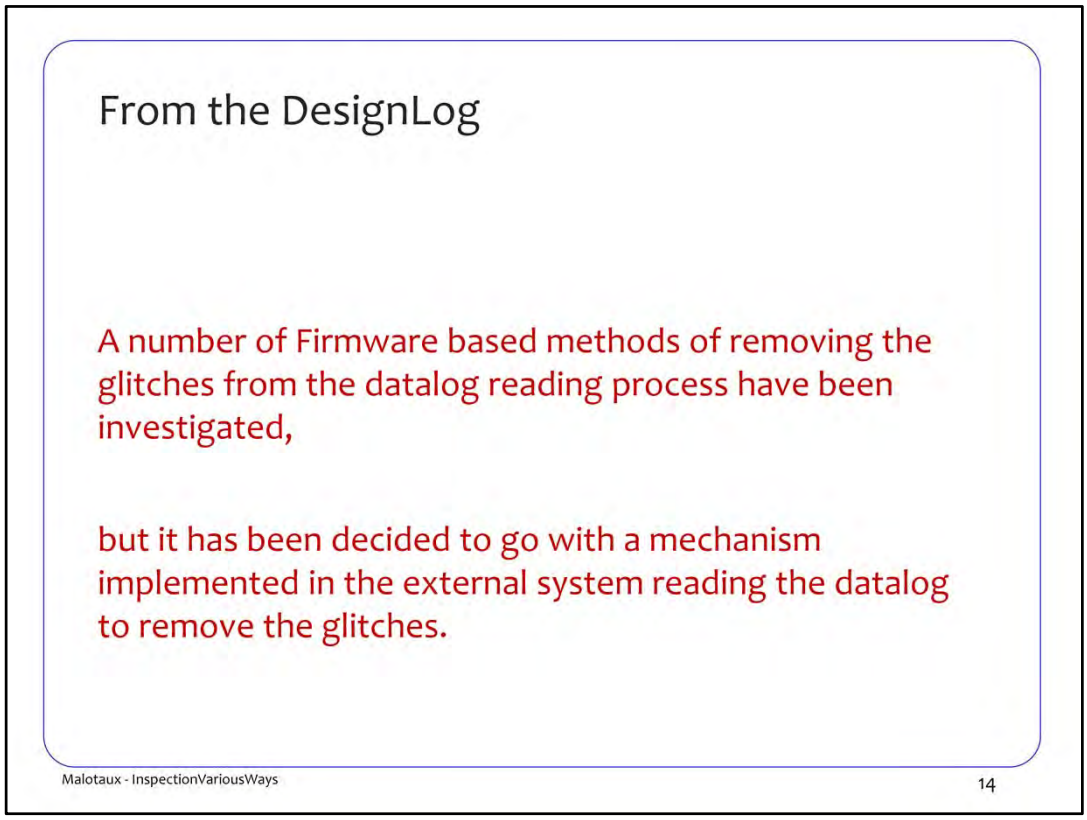
Imagine what would have happened if he had started coding already. Getting deeper and deeper into trouble, not wanting to stop, because having spent already so much time on coding.

First he had complained that I was delaying his project because he wasn't allowed to start coding. Later he said: Thank you, you saved my project!

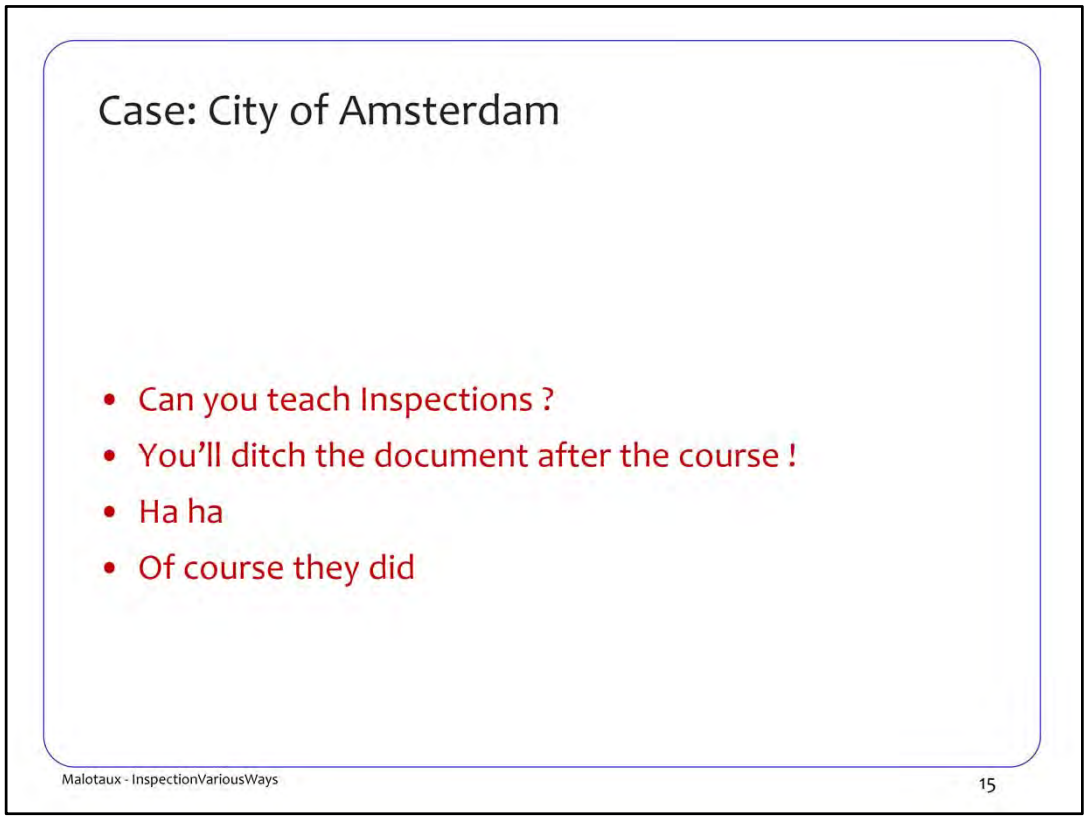
The design and the Inspection caused an unanticipated decision.

Using Inspection in an interesting way:

- If I would have suggested that the document wasn't right, he wouldn't have believed it.
- Now they *showed themselves* that there was a problem with design. That's much more powerful!
- Instead of proceeding further with Inspections, we first started working on what design was about.



I found the exact text in the DesignLog where he decided not to implement it in Firmware, but in PC software.



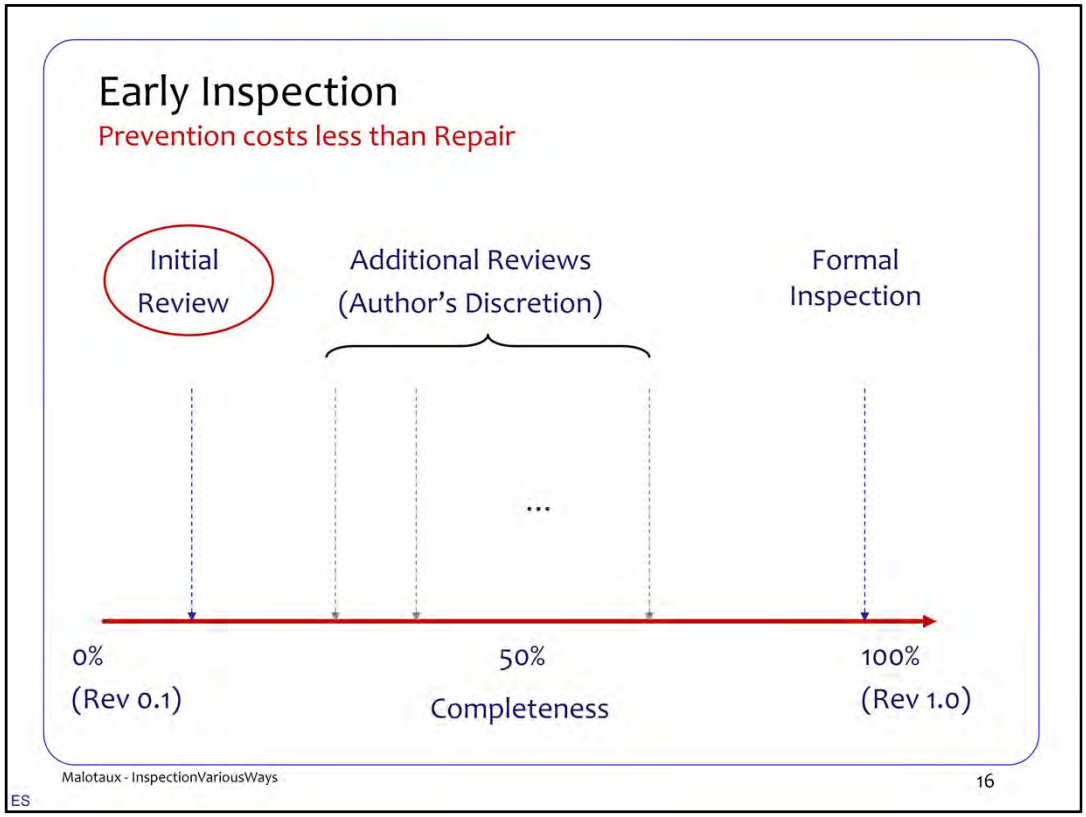
A team of the City of Amsterdam wrote a RfP (Request for Proposal) for some software development. Apparently they had some feeling that the RfP might have some issues, so they asked me to do an Inspection training with this document as training material.

When they called me for the training, I already warned them that after the training they probably would throw the document in the waste-bin, as that's my experience with first Inspections. They only laughed about this, not really believing me.

You can guess what happened at the end of the day: they ditched the document as useless. Within a few days, they even redefined the whole project, because of what they learnt by inspecting the document.

Using Inspection in various ways:

- If I would have suggested that the document wasn't right, they wouldn't have believed it.
- Now they showed themselves that it wasn't right. That's much more powerful!
- By doing the inspection, they learn so much about what their project should be about, that they redefined the project completely.
- Guess what would have happened if they would have sent out the RfP to prospective suppliers, who love those unclear documents, because it will allow them to sell many more hours than initially agreed. And apparently the City would have got a great solution for the wrong problem.



Early Inspection:

Why would we allow an author to complete the whole waterfall of a list of requirements, a design, a code module (for this Sprint), knowing that he is in the process of injecting a certain amount of defects?

How about after the author produced some 10%, reviewing this part, ploughing back the findings of the review immediately, so that the author can correct the issues in the first 10% and then prevent injecting similar defects in the remainder of his work?

I have a few case-studies (cannot mention the name of the large company), showing a huge Return on Investment of the Early Inspection technique. We always routinely calculate the RoI, to show that the time invested was worth spending. After all, we're always aware that we should spend our time on productive things, so we have to check that we are.

Because of time limitations, I'll show just one example (next slide).

Case: Early Inspection on Requirements

Large e-business application with 8 requirements authors

- Each sent the first 8-10 requirements of ~100 requirements 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 – what we did.

Results

Average major defects per requirement in initial review	8
Average major defects per requirement in final 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

Case study – results.

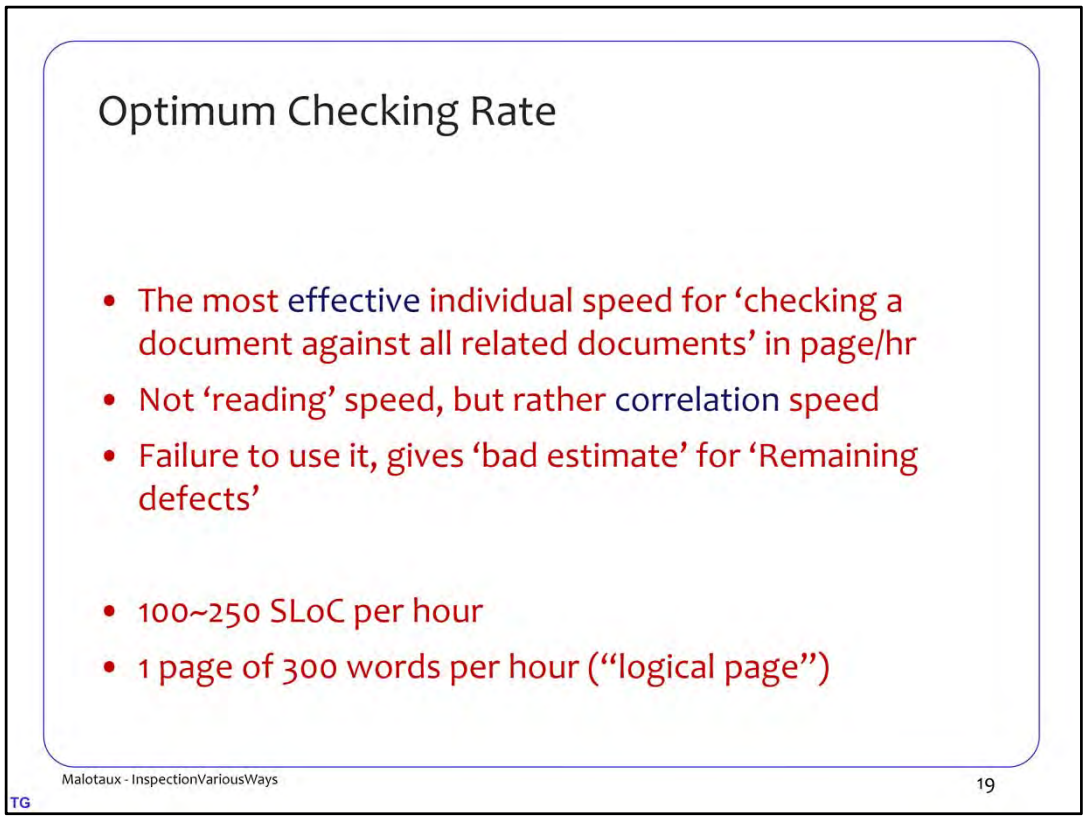
Major defects per requirement initially: 8

Major defects per requirement in final document: 3

Time saved in this case: some 37.500 hr (at least 1200 person-days).

Cost saved in this case: some $625.000.

I'll explain how we do these calculations.

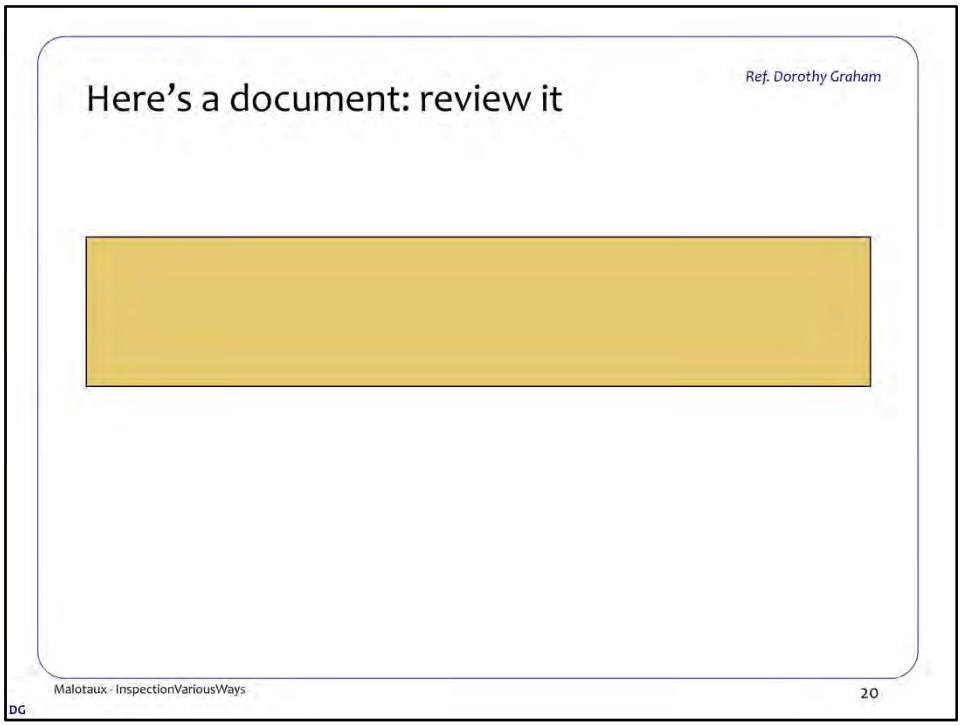


When I attended an Inspection Tutorial by Dorothy Graham, I had this 'Aha moment' about using sampling with Inspections.

When people first get the message that they should spend *one hour* per page of a document, they don't believe it (I didn't at first).

People who are used to doing reviews, but never had proper training, usually hardly find any issues. After some training they start understanding how to spot issues in a way they never contemplated before, and now start finding loads of issues on each page. You won't believe this until you experienced it yourself. I didn't believe it, until I tried it.

Actually, I found that when people start doing Inspections seriously, within some 20 min they already find so many issues on any page, that spending more time doesn't make sense.

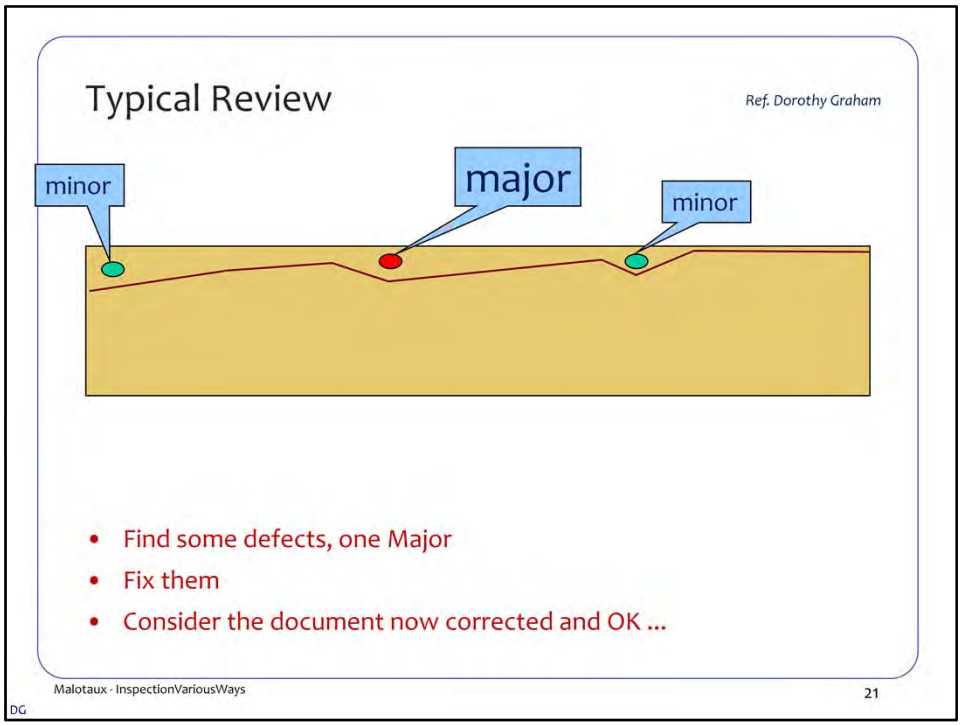


The following three slides are originally from Dorothy Graham. I had used these slides already many times in my own Inspection workshops.

When I had the opportunity to actually attend an Inspection Tutorial with Dorothy, I suddenly got an interesting 'Aha-moment'.

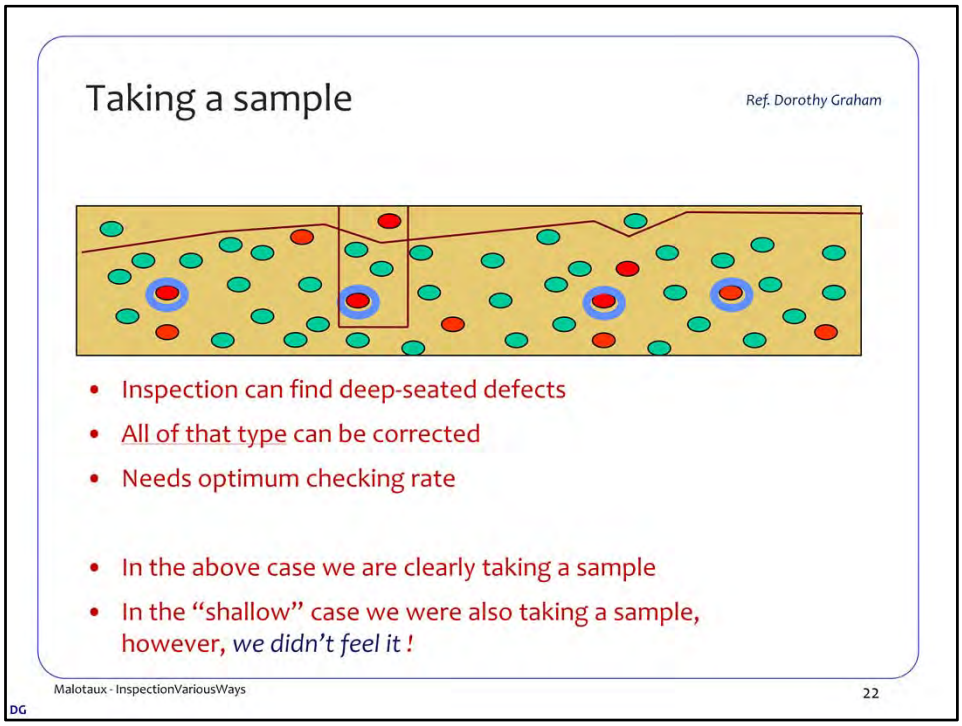
Normally, people are given a document for review: "Review this document".

In the datalog case we already found out what may happen then.



They tend to read the whole document (authors usually want us to 'review everything', in order to 'find all issues'), using a few minutes per page 'otherwise it will take too long'.

They find some issues, but by not following the 'optimum checking rate', they find only issues directly at the surface. They do not find the 'deeper seated' issues, which take more time for our mind to discover.



Because it takes some time before our mind detects the 'deeper seated' issues, the advice is to take a sample of only a few pages, and use the 'optimum checking rate' for these pages. That's not just reading the page many times, but also checking the correlation and consistency with other parts of the document as well as with other documents: How can we judge the correctness of a design, if we don't check what the design should accomplish (the requirement). Or: how can we judge the correctness of a code module, if we don't know the design the code is supposed to implement.

Most authors initially don't like us taking 'only' a sample. They want us to 'check the whole document'.

When for the first time hearing Dorothy herself explaining these principles, I suddenly got this 'Aha-moment':

When taking a 'vertical sample', it's clear that we are studying only a part of the document. If we are going through the whole document, as shown in the previous slide, we are, however, *also taking a sample*, namely a 'horizontal sample', but we're *not aware of the fact that we are also taking a sample*.

This was also the first time Dorothy realized this.

I would like to talk about how we used Review and Inspection techniques to achieve less expected, but very interesting outcomes.

I will show some cases as illustration.