

How can QA help creating Zero Defects Software?

Niels Malotaux

www.malotaux.nl/conferences

Niels Malotaux:
»In my experience the
'zero defects' attitude
results in 50% less
defects almost
overnight.«

A colleague found this remark in one of my tutorials: "In my experience the 'zero defects' attitude results in 50% less defects almost overnight" and asked me to explain this to a project team he was coaching. I thought that my experience with Zero Defects might be interesting for more people than just this project team.

Niels Malotaux



- Independent Project and 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
- Getting projects on track

Result Management

Malotaux – Unicom Amsterdam 2016

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 (1976-1980)

Malotaux - Electronic Systems Design - : developing electronic systems for clients products (1980-1998)

Now: **N R Malotaux - Consultancy**: coaching projects to deliver successfully and much faster (1998-)

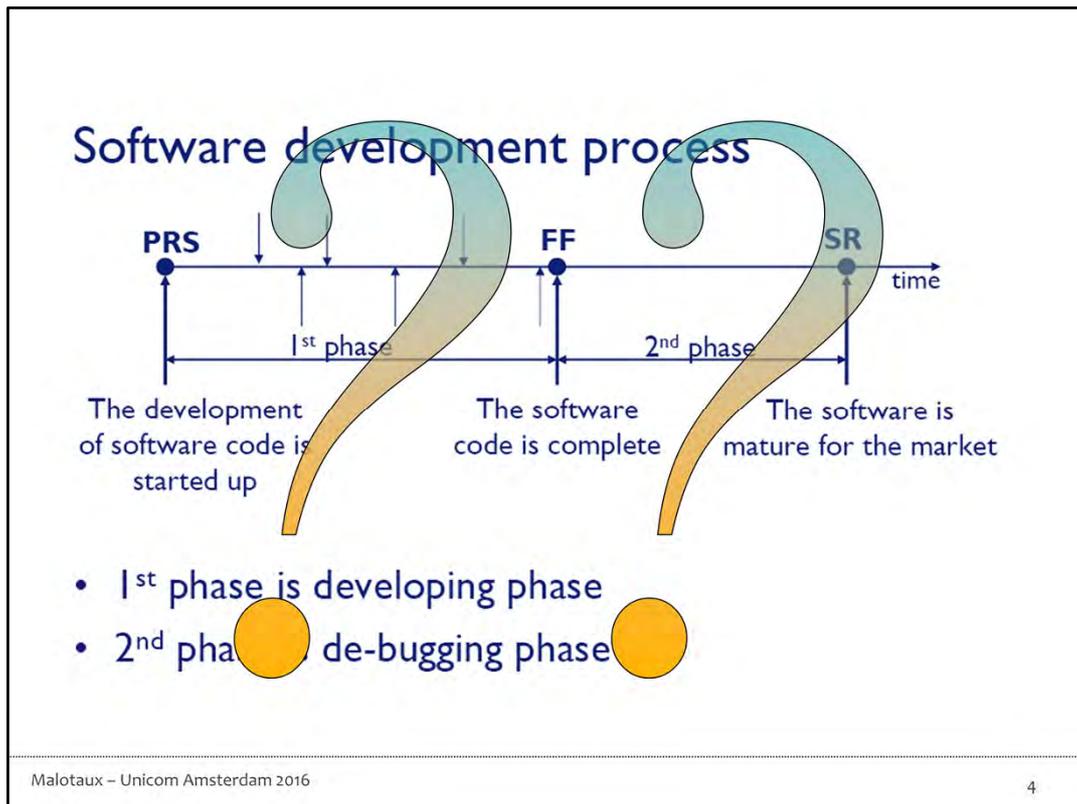
Do we deliver Zero Defect software ?

- How many defects are acceptable ?
- Do the requirements specify a certain number of defects ?
- Do you check that the required number has been produced ?

In your work

- How much time is spent putting defects in ?
- How much time is spent trying to find and fix them ?
- Do you sometimes get repeated issues ?
- How much time is spent on defect prevention ?

As many people think that even talking about ZD is useless, I'd like first to discuss some questions with the audience.



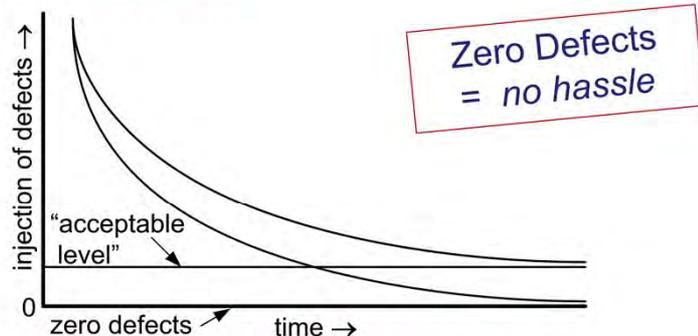
A University PhD student showed this picture as being the official development process at a well known large company in Holland. I've seen a similar picture in a presentation from a well known large software company in the US.

The 2nd phase usually takes 50(±30)% of the total time. How can we call it “Code Complete” if it's full of issues?

This is a very bad and costly process. However, because it's so widely practiced, many people think that this is how it should be. They should know better. Probably deficiency of the educational system, because the solution is known for decades.

What is Zero Defects

- Zero Defects is an *asymptote*



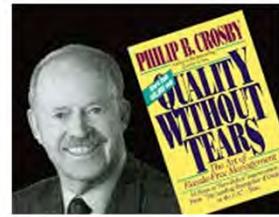
- When Philip Crosby started with Zero Defects in 1961, errors dropped by 40% almost immediately
- AQL > Zero means that the organization has settled on a level of incompetence
- Causing a hassle other people have to live with

When I actively started using the Zero Defects (ZD) concept in software projects, defects made decreased by at least 50% almost immediately. It took about 2 weeks before the developers understood that I was dead serious about it. Then the testers came to me saying: “Niels, something weird is going on: we don’t find issues anymore! It simply works!” I said: “Isn’t that exactly what we want to see? Now testing is becoming a real challenge, namely proving that there are no errors.”

So, even if you don’t believe that this can be true, if two people (Crosby and me) did it and showed a huge decrease of errors *made*, only by adopting the attitude, isn’t it at least worth a try, especially if you realize that about half of most projects is spent on finding and fixing defects. That’s a huge budget. Any savings on that is probably well worth trying.

"No Hassle" proved to be easier to use than ZD: Don't cause a hassle. No hassle to yourself, to your peers, to your organization, to your customers.

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



Philip Crosby defined the four 'Absolutes of Quality'. When I started as a coach in a company recently, I gave his book "Quality without tears" to the CEO for homework: "Next week I'll check that you read it!". He did and it immediately had an impact on his behaviour. He delayed a major release to first get rid of the hassles that we were going to deliver to the costumers. He also calculated the 'Price of Non-Conformance' (PONC), to be at least a quarter of a million Euro in the past year.

Phil Crosby's organization later added a 5th Absolute: Customer Success. I agree completely. But I don't agree with them adding "... not customer satisfaction". After all, the customer should be successful, but satisfied as well, as we'll see on the next slide.

Ultimate Goal of a What We Do

Quality on Time

Delivering the Right Result at the Right Time,
wasting as little time as possible (= efficiently)

Providing the customer with

- what he needs
- at the time he needs it
- to be satisfied
- to be more successful than he was 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

This is to me the top-level requirement for any project or any work we do.

- The customer is the entity that orders and pays. The customer, however, in many cases doesn't use the result of our project himself. He gets the benefit through the users of the result.
- What the customer says he wants is usually not what he really needs
- The time he needs it may be earlier or later than he says
- If the customer isn't satisfied, he doesn't want to pay
- If the customer isn't successful with what we deliver, he cannot pay
- If he's not more successful, why would he pay?
- What the customer wants, he cannot afford. If we try to satisfy all customer's wishes, we'll probably fail from the beginning. We can do great things, given unlimited time and money. But neither the customer nor we have unlimited time and money. Therefore: The requirements are what the Stakeholders require, but for a project: the requirements are what the project is planning to satisfy.
- The customer is king, but we aren't slaves. Both sides should benefit and be happy with the result.
- We will get the best result in the shortest possible time, but not shorter than possible. The impossible takes too much time.

Prevention: Root Cause Analysis

- Is Root Cause Analysis routinely performed – every time ?
- What is the Root Cause of a defect ?
- **Cause:**
The error that caused the defect
- **Root Cause:**
What *caused us* to make the error that caused the defect
- Without proper Root Cause Analysis ,
we're doomed to repeat the same errors

Years ago I suggested to add a box for the 'Root Cause' and for the 'Root Cause Suggested Solution' in a bug-tracking system. When I later checked how people were using this, I found that in the Root Cause box they documented the cause of the bug and in the Root Cause Suggested Solution box the suggestion how to repair the bug.

Apparently, they didn't see the difference between 'Cause' and 'Root Cause':

- The Cause of a defect is the error that caused the defect
- The Root Cause is what caused *us* to make the error that caused the defect

In another project I asked the project manager what they do with the results of the code reviews. "People repair the bugs" he said. I asked: "Don't you do Root Cause Analysis, in order to learn how to prevent this type of error from now on?" The response was: "On every issue we found??? We have no time for that!"

Apparently they have no time to learn to prevent, and rather spend a lot of time to find and fix(?). No wonder that projects take more time than they hoped for.

We're QA: What has this to do with us ?

- How does QA fit in ?
- What is the goal of QA in a software development project ?
- Do we still have a job when development produces no defects ?
- Who is our customer ?

Many people equate QA with Testing. Testing, however, is just one of the quality measuring instruments of QA and hence only a small part of QA. So, let's shortly discuss what QA actually is and who the customer of QA is.

Who is the (main) customer of Testing and QA ?

- **Deming:**
 - Quality comes not from testing, but from *improvement of the development process*
 - Testing does not improve quality, nor guarantee quality
 - It's too late
 - The quality, good or bad, is already in the product
 - You cannot test quality into a product
- **Who is the main customer of Testing and QA ?**
- **What do we have to deliver to these customers ?**
What are they waiting for ?
- **Testers and QA are consultants to development**
- **Testing and QA shouldn't delay the delivery - How ?**

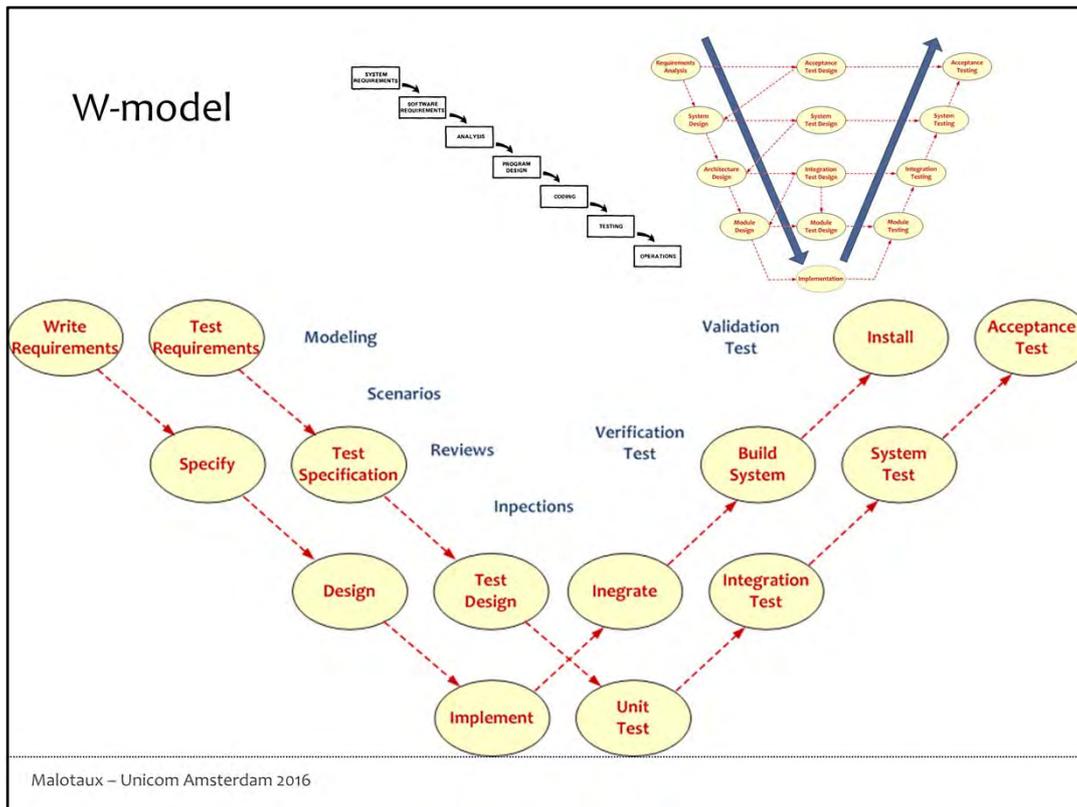


Deming
(1900-1993)

Malotaux – Unicom Amsterdam 2016

I experienced that to most testers this quote from Deming is quite a paradigm shift and usually comes as a shock. But usually it's a shock of recognition! It will change their attitude for the better forever.

Now let's see how we can optimize our contribution as consultants to development.



In the Agile world, the Waterfall and derived models are often seen as bad. However, these models are still valid for every sprint. After all, we have to determine what value we should deliver (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 the bugs 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.

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 the issues once they are created, rather than when they cause trouble later. All models are wrong, some are useful. If they're useful, we may use them.

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 'physically' 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.

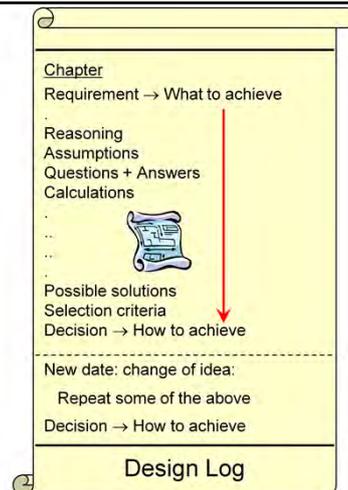
Some Examples

We're not perfect,
but the customer shouldn't find out

Let's discuss some examples of techniques that helped me and others to move towards Zero Defect deliveries.

Design techniques

- 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 ?
 - Reconstruct the design (if the design description is lacking)
 - What happens if you ask "Can I see the DesignLog ?"



There are many techniques known to approach ZD faster. One of them is what I call 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 usually only shows what people decided to implement, 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, in the maintenance phase (often a the largest portion of the cost of deployment of software!) minimum time is lost. One of the requirements for the DesignLog is: "If someone has to change something in the software one year later, he should be up and running within one or at most two days."

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

Case: In the pub

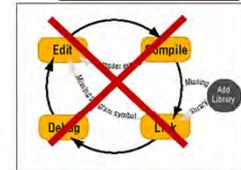
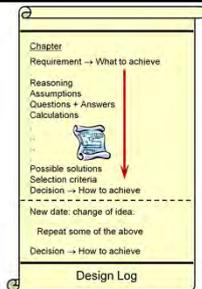
James:

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!

What James told me recently

- 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
- In summary, this success has proved instrumental in buy-in for DesignLogs which are now embedded in the development process

Malotaux – Unicom Amsterdam 2016

James explained me later more interesting details of this case:

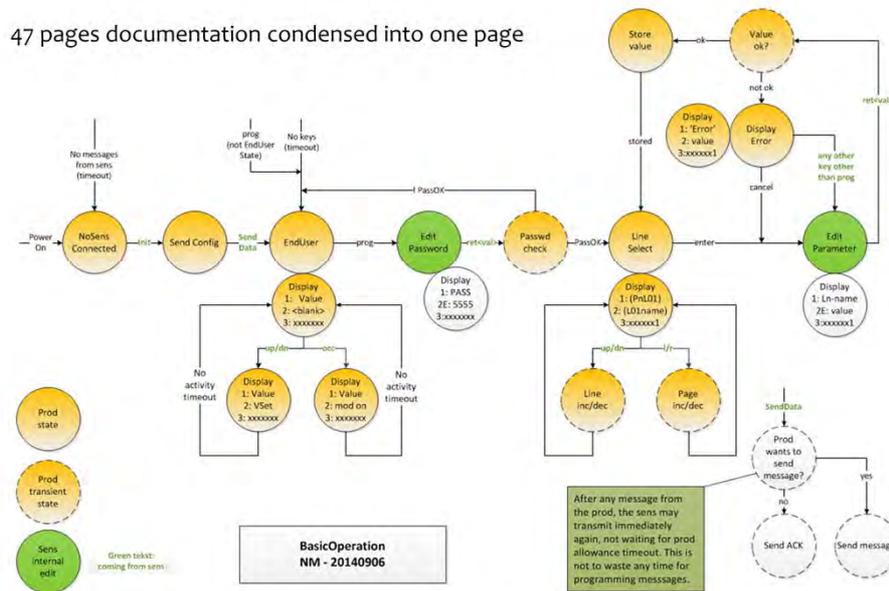
- There were two features required for a release, one of which was on the critical path and placed the delivery at risk
- I saw an "opportunity" for a Design - "*prevention, rather than fixing*" and also an opportunity to encourage documentation
- Because Louise struggled a bit with the design (not many 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.

Choose the appropriate design

47 pages documentation condensed into one page



If I see documentation at all, it is usually just text. Sometimes a lot of text. One of my mantra's is: "Where are the pictures?"

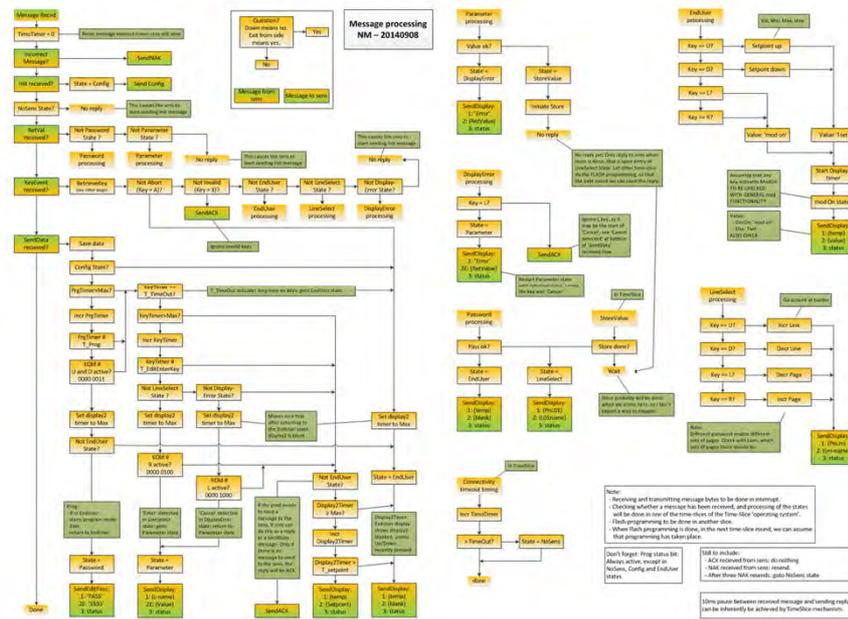
This and the next slide are an example of some design I made recently (anonymised). You don't have to check the text and what it actually does. It's just to show some examples of concisely documenting functionality in a way most people, with a bit of understanding, can follow immediately.

This slide shows a design of the communication between some controller and a remote user interface. It was documented in a 47 page document by an 'architect'. 47 pages of interface description is almost impossible to oversee by humans, hence it contained a lot of inconsistencies and the people who had to implement it actually ignored it.

Once I made this one page overview, we could discuss, ease out the inconsistencies, make decisions, agree, and everyone knew exactly what to do. Conclusion: just documenting isn't enough. We have to learn how to document for usefulness.

QA can ask a developer to explain how the interface should work. If the developer only shows code to review, we know we have a problem. If the QA person doesn't understand the explanation, the explanation apparently isn't clear enough, which is a big risk for the quality of the result. If it's only text, it won't work either.

Design example



This is how I implemented the communication design based on my discussions with the suppliers of the remote user interface. The design was made to be reviewed and then it could readily be implemented based on this design. If I see how much I moved and reshuffled before I was content that this was right, I cannot imagine how this could be done properly in code without having this design. Like in the Cleanroom Approach to Software Development I designed down to a level of some 3 lines of code per design element. Sorry, I have no time now to go into detail, but the Cleanroom Approach routinely delivered an order of magnitude less defects in shorter time. Making changes in the code is not allowed before we have updated the design. The code should always be derived from the current design. Reviews of code should always check that the code does what the design says

These were just examples. The challenge is every time again to find the right representation that is easiest to comprehend and review.

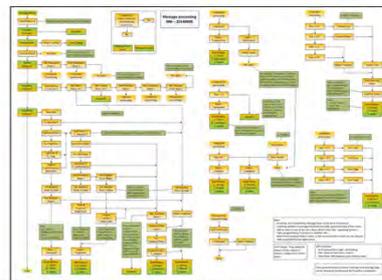
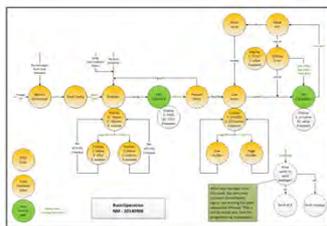
Of course the projects the audience is working in usually do these things properly. But I still see too often that the 'design' is only in the mind of the developer who writes the code, or just a rough sketch, with devastating effects in software quality and delivery time.

If as a QA person you encounter these effects, think what you should do about it.

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 ?

```
20 #c switch in different layouts directly
21 #c switch in different layouts directly
22 #c switch in different layouts directly
23 #c switch in different layouts directly
24 #c switch in different layouts directly
25 #c switch in different layouts directly
26 #c switch in different layouts directly
27 #c switch in different layouts directly
28 #c switch in different layouts directly
29 #c switch in different layouts directly
30 #c switch in different layouts directly
31 #c switch in different layouts directly
32 #c switch in different layouts directly
33 #c switch in different layouts directly
34 #c switch in different layouts directly
35 #c switch in different layouts directly
36 #c switch in different layouts directly
37 #c switch in different layouts directly
38 #c switch in different layouts directly
39 #c switch in different layouts directly
40 #c switch in different layouts directly
41 #c switch in different layouts directly
42 #c switch in different layouts directly
43 #c switch in different layouts directly
44 #c switch in different layouts directly
45 #c switch in different layouts directly
46 #c switch in different layouts directly
47 #c switch in different layouts directly
48 #c switch in different layouts directly
49 #c switch in different layouts directly
50 #c switch in different layouts directly
51 #c switch in different layouts directly
52 #c switch in different layouts directly
53 #c switch in different layouts directly
54 #c switch in different layouts directly
55 #c switch in different layouts directly
56 #c switch in different layouts directly
57 #c switch in different layouts directly
58 #c switch in different layouts directly
59 #c switch in different layouts directly
60 #c switch in different layouts directly
61 #c switch in different layouts directly
62 #c switch in different layouts directly
63 #c switch in different layouts directly
64 #c switch in different layouts directly
65 #c switch in different layouts directly
66 #c switch in different layouts directly
67 #c switch in different layouts directly
68 #c switch in different layouts directly
69 #c switch in different layouts directly
70 #c switch in different layouts directly
71 #c switch in different layouts directly
72 #c switch in different layouts directly
73 #c switch in different layouts directly
74 #c switch in different layouts directly
75 #c switch in different layouts directly
76 #c switch in different layouts directly
77 #c switch in different layouts directly
78 #c switch in different layouts directly
79 #c switch in different layouts directly
80 #c switch in different layouts directly
81 #c switch in different layouts directly
82 #c switch in different layouts directly
83 #c switch in different layouts directly
84 #c switch in different layouts directly
85 #c switch in different layouts directly
86 #c switch in different layouts directly
87 #c switch in different layouts directly
88 #c switch in different layouts directly
89 #c switch in different layouts directly
90 #c switch in different layouts directly
91 #c switch in different layouts directly
92 #c switch in different layouts directly
93 #c switch in different layouts directly
94 #c switch in different layouts directly
95 #c switch in different layouts directly
96 #c switch in different layouts directly
97 #c switch in different layouts directly
98 #c switch in different layouts directly
99 #c switch in different layouts directly
100 #c switch in different layouts directly
```

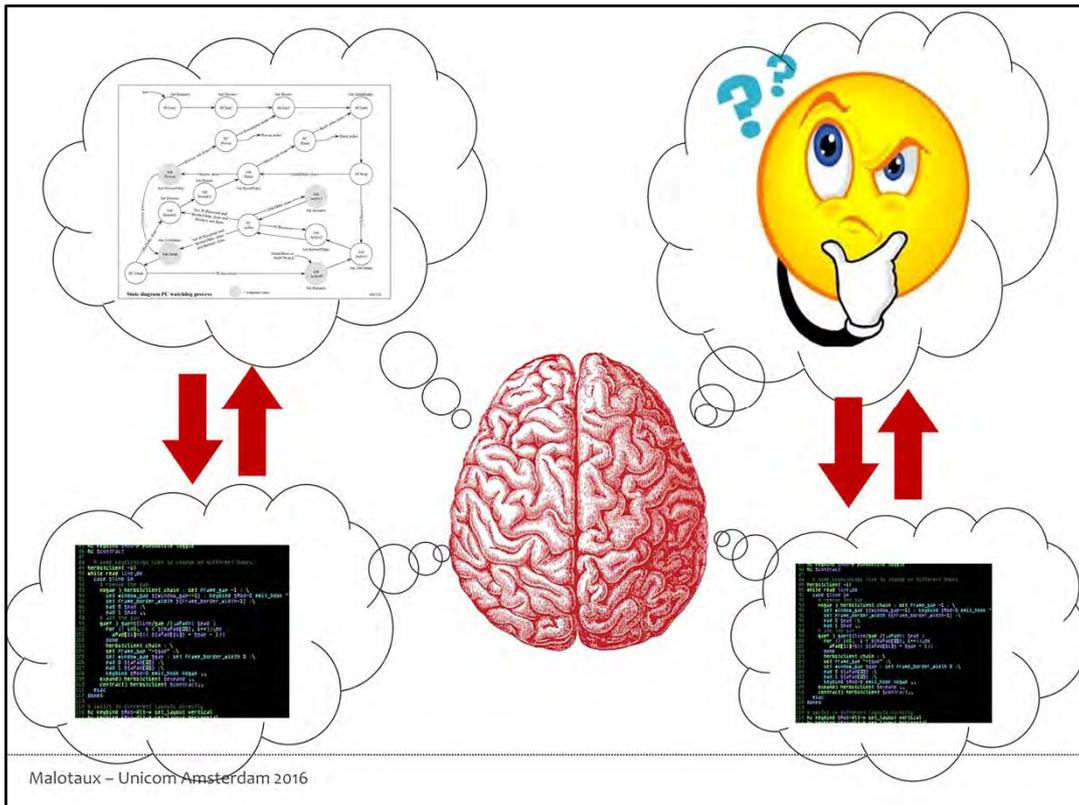


Malotaux – Unicom Amsterdam 2016

Many people do reviews, but have they been trained in reviewing and what is the best way to optimize the effectiveness.

Don't waste your time on reviewing code if you haven't reviewed the design. After the review of the design, the code will be changed anyway, so what's the point of first reviewing the code?

If there is only code and no design, reconstruct the design. If you work only in code, producing bugs is to be expected. Bugs shouldn't be there. Hence, don't work only in code.



If we don't have a proper design, the developer should still have a model of the system in his mind. A problem is that the model in the mind is fading, so we have to continuously refresh it. That's why you see developers moving and switching screens all the time, trying to keep the model in their mind current. At the same time they should in their mind correlate the model in their mind with the code they should produce or modify,.

If they would get the model of the system out of their mind e.g. on paper, or whiteboard, they don't have to continuously refresh it, they quickly see flaws in the design, others can even watch it and see possible issues, and they have their whole mind available to do the thinking and correlating.

Case: Scrum Sprint Planning

- What is the measure of success for the coming sprint ?
- “What a strange question !
We're Agile, so we deliver working software. Don't you know ?”
- Note: Users are not waiting for software:
they need *improved performance* of what they're doing
- How about a requirement for 'Demo': No Questions – No Issues
- How's that possible !!?
- They actually succeeded !

Malotaux – Unicom Amsterdam 2016

I came in an project of some 70 people, with 3 Scrum teams of some 12 people each. We know 12 is too many, but that's another story.

At a Sprint Planning meeting I asked one of the teams: "What would be the measure of success for this Sprint?"

They looked at me: "What a strange question. We're Agile, so we deliver working software. Don't you know?"

I asked: "Shouldn't we have a measure of success, to know that we really did a good job?" and suggested: "No questions, No Issues". That's easy to measure: one question or one issue and we know we failed. No question and no issue and we know we were successful.

Their first reaction was: That's impossible! Surely there will be some questions when we deliver and there are always some issues.

I suggested: "You find out how to do it. It's just a simple requirement: "No questions, No Issues".

Interestingly, they immediately started thinking how they could deliver according to this requirement.

For example, someone thought: "Ah. Perhaps halfway the Sprint we ask someone to check it out and to see whether he would have any questions?" I said: "You're on the right track. Just find out how to do it. The requirement is simple."

Actually, I didn't expect them to be successful in this first Sprint, perhaps after a few. Surprisingly, they were successful.

If we deliver

- Give the delivery to the stakeholders
- Keep your hands handcuffed on your back
- Keep your mouth shut
- and o-b-s-e-r-v-e what happens
- Seeing what the stakeholders actually do provides so much better feedback
- Then we can 'talk business' with the stakeholders

- Is this what you do ?



Malotaux – Unicom Amsterdam 2016

Scrum reviews usually only demo.

How about using the advice as shown here.

Is this what you do?

If not, why not?

The 'Demo'

Concurrent database record update

Customer site



Demo room



Delivery Strategy Suggestions (Requirements)

- What we deliver will be used by the appropriate users immediately, within one week not making them less efficient than before
- If a delivery isn't used immediately, we analyse and close the gap so that it will start being used (otherwise we don't get feedback)
- The proof of the pudding is when it's eaten and found tasty, by them, not by us
- The users determine success and whether they want to pay (we don't have to tell them this, but it should be our attitude)

This is my suggestion I gave the architect and the project manager to consider as Delivery Strategy.

How much legwork is being done in your project ?

- Requirements/specifications were trashed out with product management
- Technical analysis was done and
- Detail design for the first delivery

At the first delivery:

- James: *How is the delivery? (quality - versus expectation)*
- Adrian: *It's exactly as expected, which is absolutely unprecedented for a first delivery; the initial legwork has really paid off*

This case was an organization with extraordinary bright people. In many projects we have to explain things over and over again, but in this project people needed only half a word to understand and do things better.

James (their new QA person) told me this story. He asked them to prepare well, design properly, and then do the coding. The result: *"It's exactly as expected, which is absolutely unprecedented for a first delivery."*

He suggested it, they did it, and it worked. It's great for a QA person to work in such a fertile environment!

Basic approach

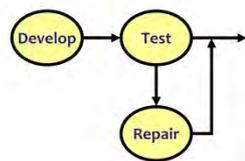
- Design the requirement
 - Review
 - Design implementation
 - Review
 - Implement (code ?)
 - Review
 - Test doesn't find issues (because they're not there)
- 
- Iterate fast, as needed
- The diagram illustrates an iterative process. It features a list of seven steps on the left. To the right of the first three steps (Design the requirement, Review, Design implementation) and the last three steps (Review, Implement (code ?), Review), there are red curved arrows pointing back to the previous step. A larger red curved arrow on the right side of the list encompasses the first three steps and the last three steps, indicating a broader iteration cycle. The text 'Iterate fast, as needed' is positioned to the right of the list.

The basic approach.

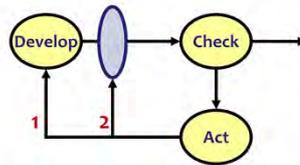
Some people simply call this: 'engineering'.

What's in it for QA ?

- Did we see much testing in the previous ?
- Testing shouldn't find anything (because there should be no issues)
- Did you ever find similar issues as you found before?
 - First time: Developers 'fault'
 - Second time: Testers 'fault'



What we often see



What we should expect

- QA to help developers to produce less and less defects

So, what's in it for QA?

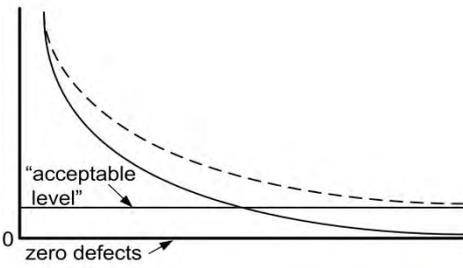
Do we deliver Zero Defect products ?

- How many defects do you think are acceptable ?
- Do the requirements specify a certain number of defects ?
- Do you check that the required number has been produced ?

In your projects

- How much time is spent putting defects in ?
- How much time is spent trying to find and fix them ?
- Do you sometimes get repeated issues ?
- How much time is spent on defect prevention ?
- Could you use “No Questions – No Issues” ?

What will you do next?



**Approaching
Zero Defects
is Absolutely Possible**

If in doubt, let's talk about it

Niels Malotaux

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

If in doubt, let's discuss.
To you it may be theory.
To me it's reality.