

Niels Malotaux:

*In my experience the
'zero defects' attitude
results in 50% less defects
almost overnight*

Examples of how to move towards Zero Defects

Niels Malotaux

Niels Malotaux



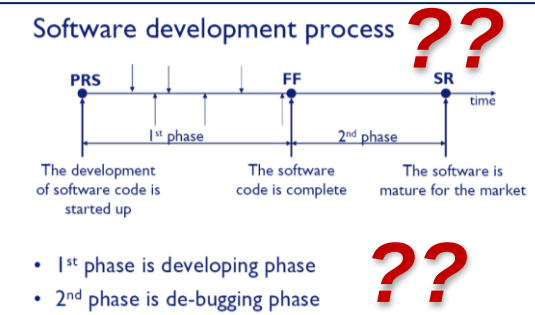
- Independent Engineering and Project Coach
- Expert helping teams and organizations quickly to become
 - More effective - doing the right things better
 - More efficient - doing the right things better in less time
 - More predictable - delivering as needed
- Project rescue (helping solving 'impossible' deadlines)
- Embedded Systems architect (electronics/firmware)
- Project types
electronic products, firmware, software, space, road, rail,
telecom, building automation, industrial control, parking system

Is software without defects possible ?

- How many defects are acceptable ?
- Requirements specify a certain number of defects ?
- Check 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 see repeated issues ?
- How much time is spent on defect prevention ?



Requirement:

- Deliver 7 bugs in every sprint

User Story:

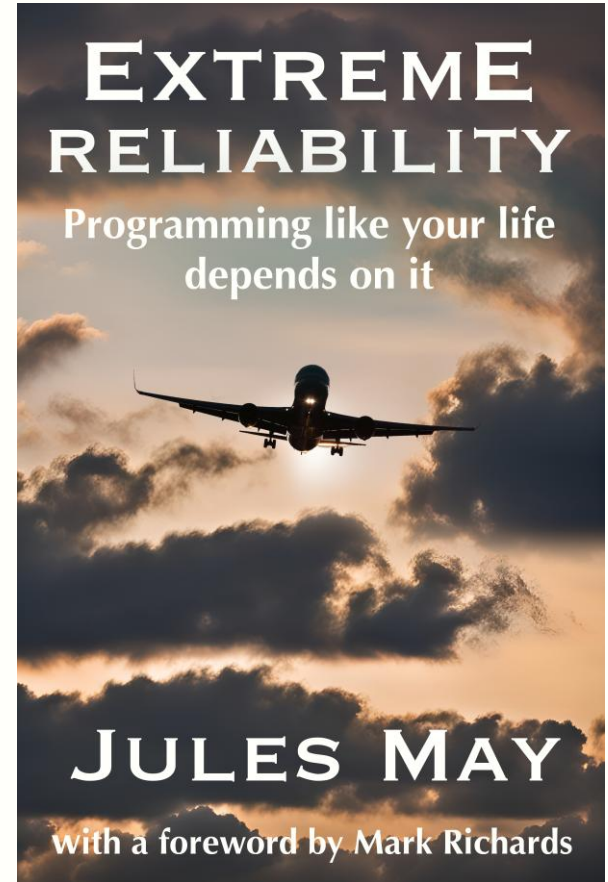
- As a customer,
I like to experience some bugs,
so that I regularly can complain

Better quality costs less

Jules May: Extreme Reliability

All my experience has convinced me
that it takes less time to code it right first time
than knowingly doing it wrong

It's not just a long-term benefit:
it's quicker in the short term as well



New QA manager at a large bank

No defects in the first two weeks of use

What is a defect ?

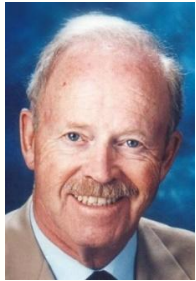


A defect is the cause of a problem
experienced by any of the stakeholders
while relying on our results

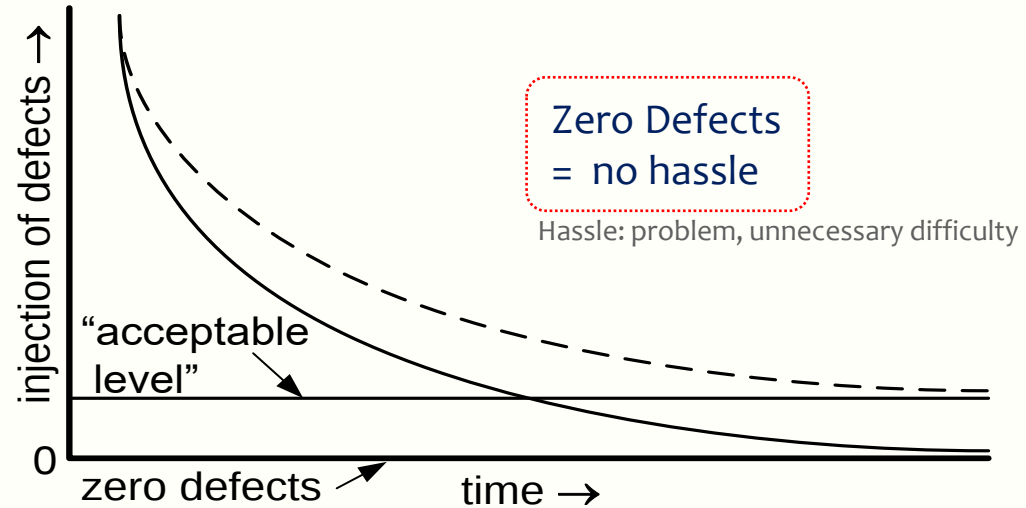
What is Zero Defects ?

We aren't perfect,
but the customer shouldn't find out

- Zero Defects is an *asymptote*



Niels Malotau:
In my experience the
'zero defects' attitude
results in 50% less defects
almost overnight



- 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

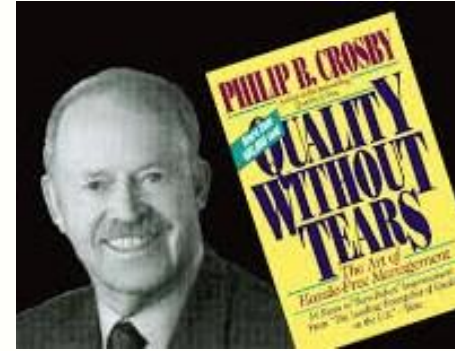
Crosby (1926-2001) - Absolutes of Quality

- 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



Universal goal

Quality on Time

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

- 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

Feeding prevention: Root Cause Analysis

- Is Root Cause Analysis routinely performed - every time ?
- What is the difference between *Cause* and *Root Cause* ?
- **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

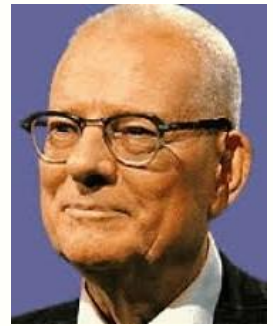
cause

solution

root cause

root solution

Doesn't Testing and QA 'assure' quality ?



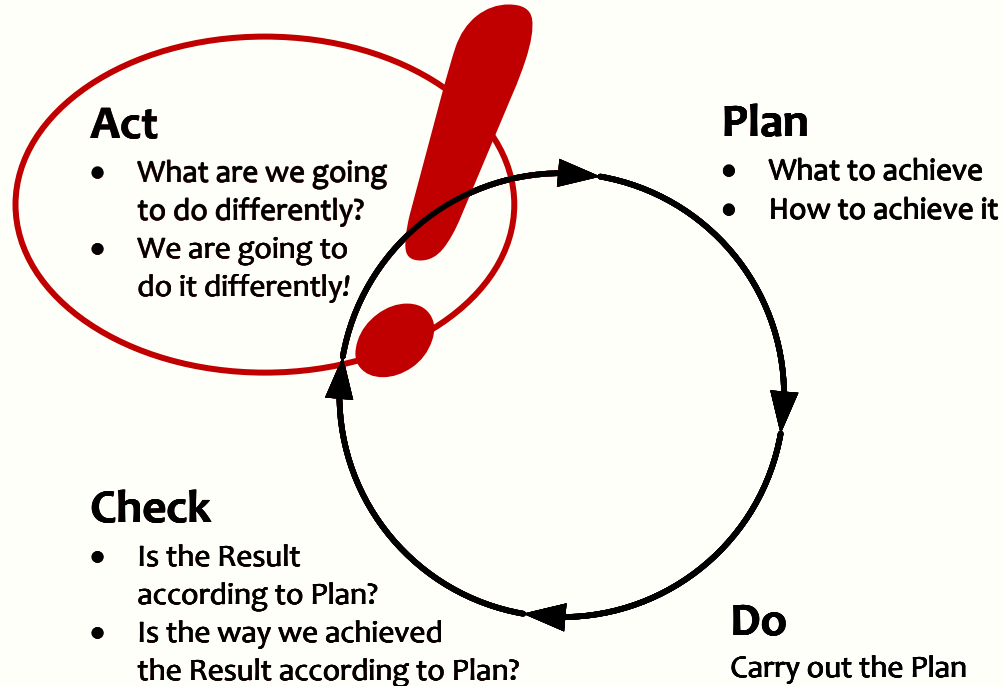
Deming
(1900-1993)

- **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 are the main customers of Testing and QA ?**
- **What to deliver to these customers ?**
What are they *waiting for* ?
- **Testers and QA are *consultants* to development**

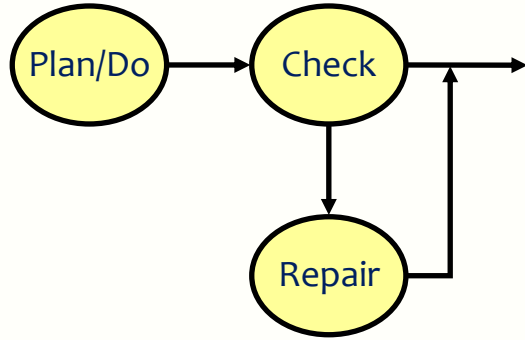
- **Providing the customer with**
 - what they need
 - at the time they need it
 - to be satisfied
 - to be more successful than without it
- **Constrained by (win - win)**
 - what the customer can afford
 - what we mutually beneficially and satisfactorily can deliver
 - in a reasonable period of time

The essential ingredient: the PDCA Cycle

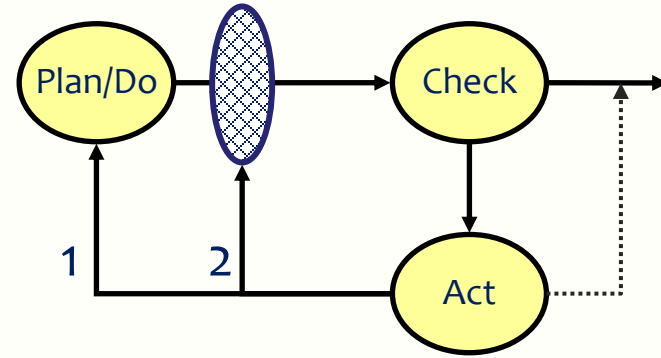
(Shewhart Cycle - Deming Cycle - Plan-Do-Study-Act Cycle - Kaizen)



How can we prevent defects ?



What we often see



What we should expect

1. How can we prevent this ever happening again ?
2. Why did our earliest sieve not catch this defect ?

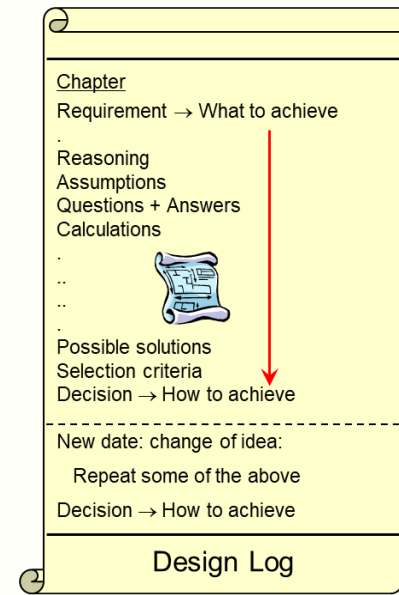
Some Examples

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

Design techniques

Cleanroom

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



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

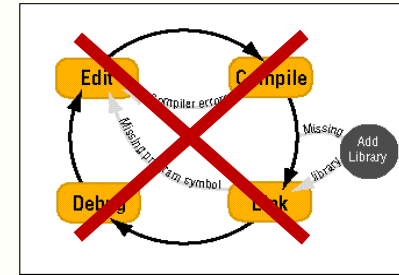
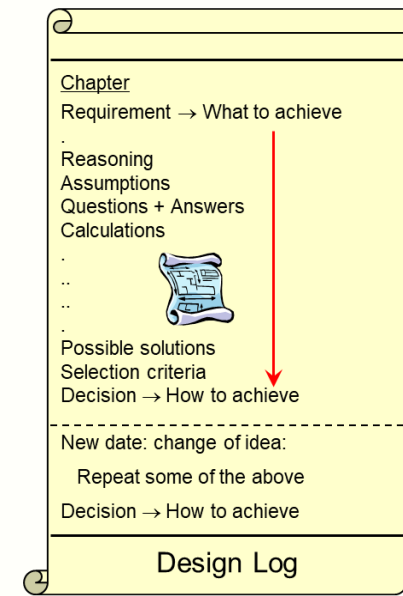
"We don't have time!" - "We've only 7 days!"

James insisted

We designed, found the problem, corrected it, cleaned up the mess

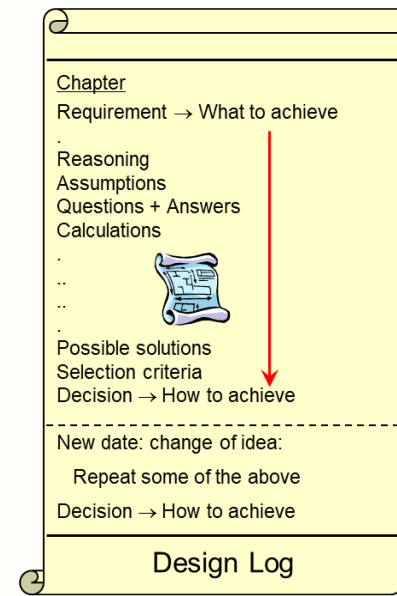
Done in less than 7 days

Thank you!



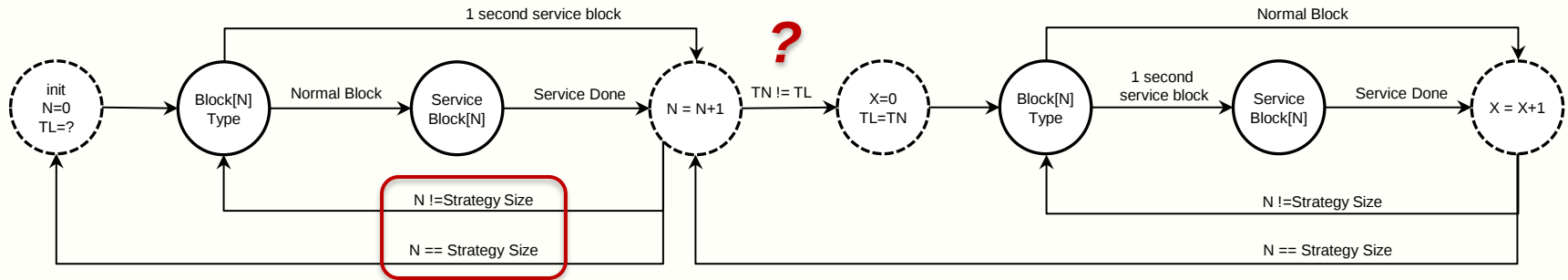
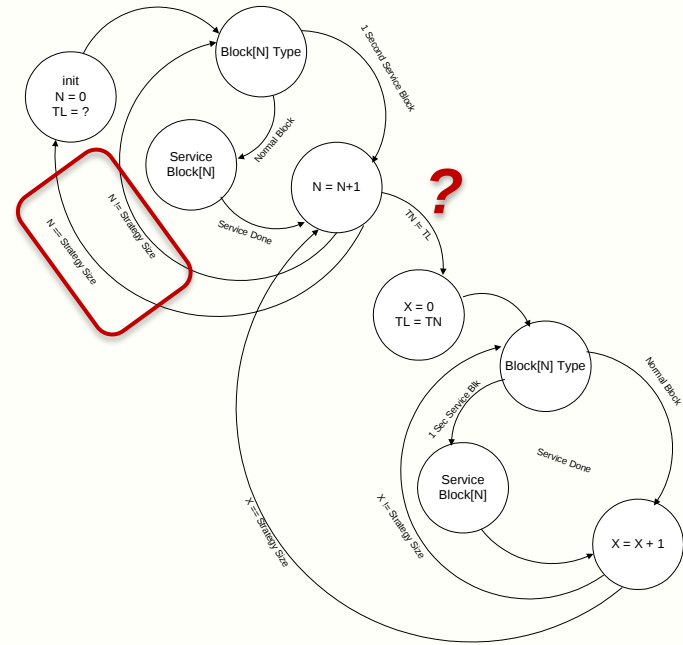
What James told me afterwards

- I gave the design to two colleagues for review
- Louise corrected some minor issues
- It went into a ‘final’ review, with another colleague
- Based on his expertise, *the solution was completely reworked*
- Actually, two features were delivered and deployed
 - The one that was design and code reviewed had no issues after deployment
 - The other one was the source of quite some defects
- This success has proved instrumental in buy-in for DesignLogs, which are now embedded in the development process



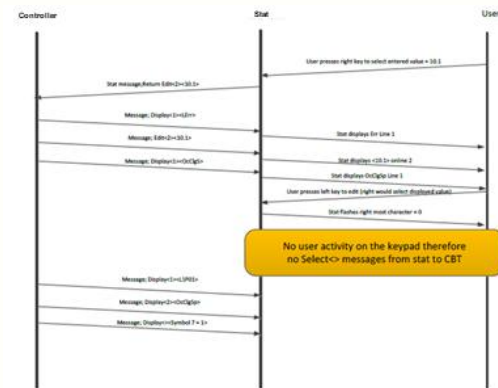
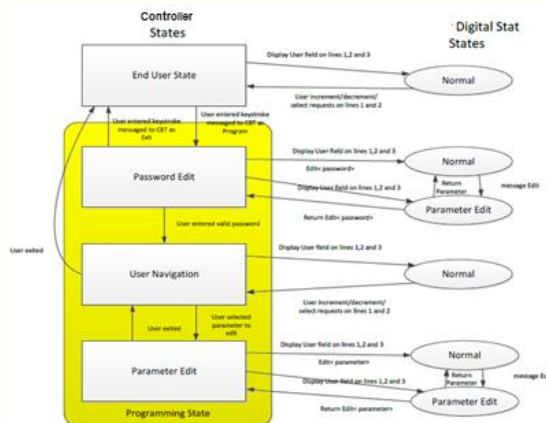
There are many ways to represent a design

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



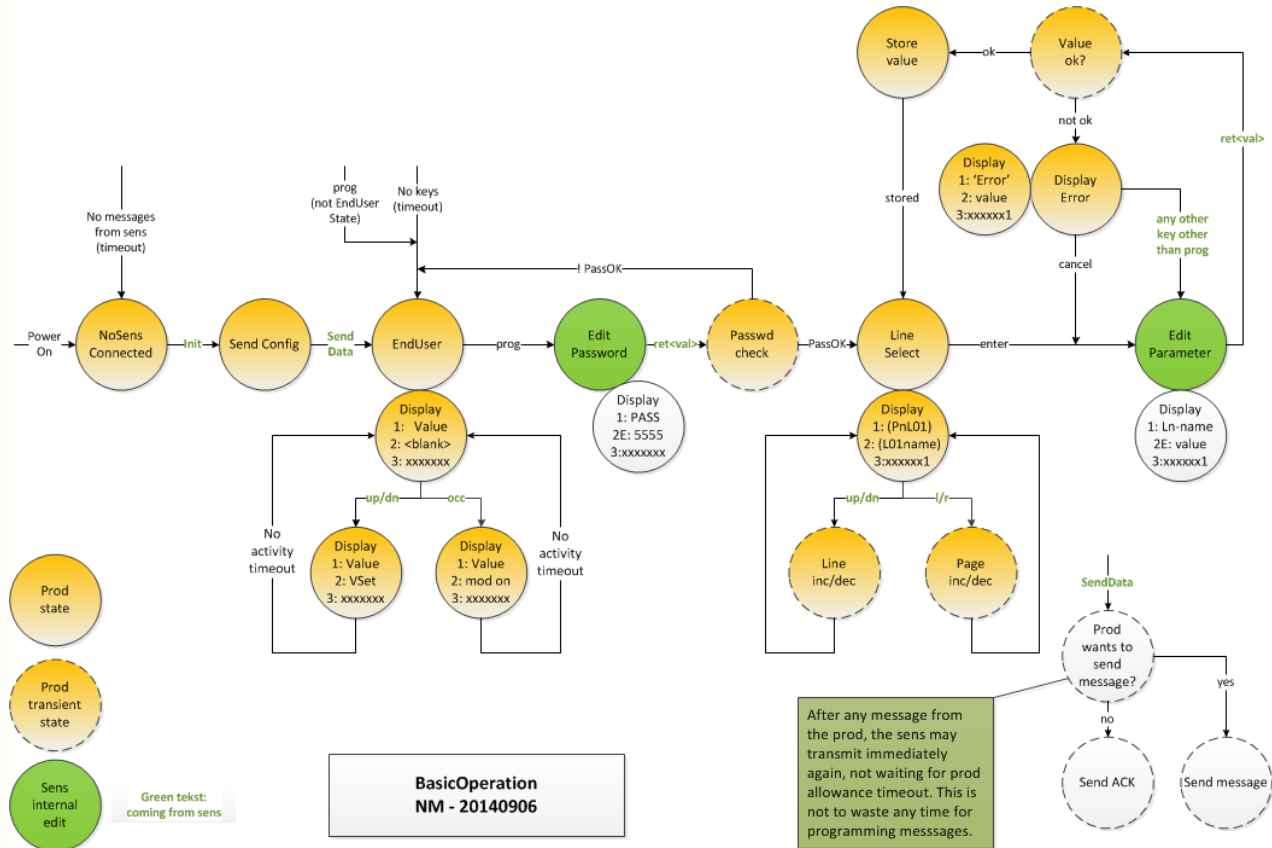


Useful design ?

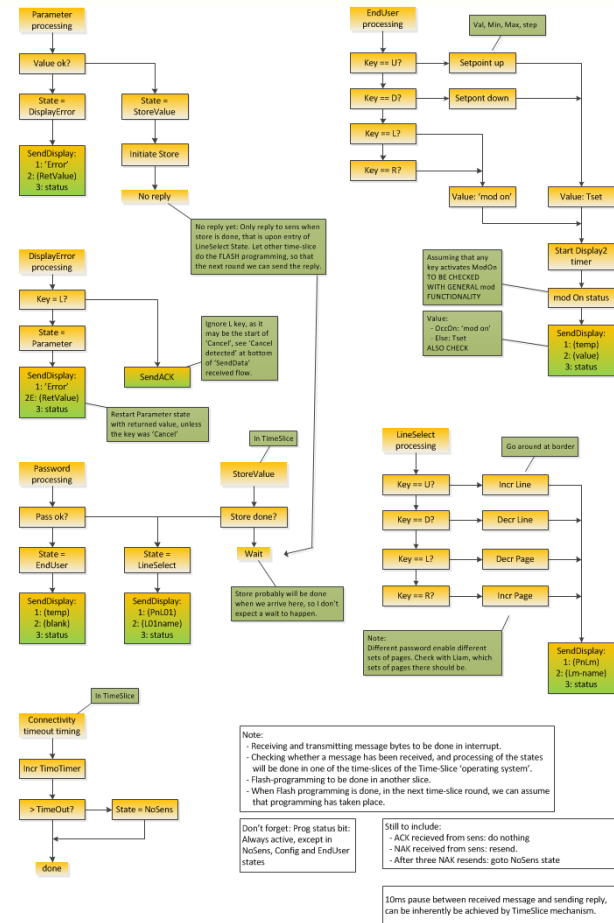


Choose the appropriate design

47 pages documentation
condensed into one page

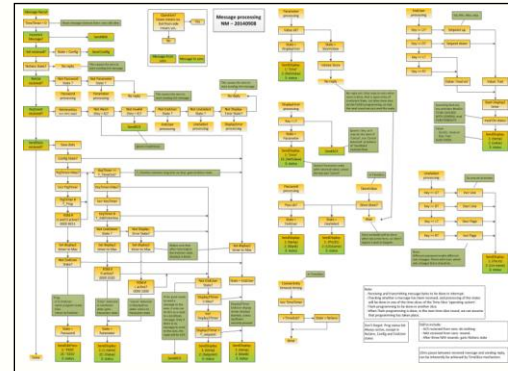
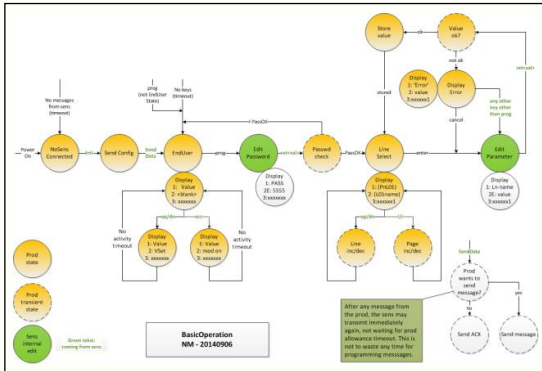


Malotaux - Zero Defects



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 ?



```

85 #! keybind $mod+p pseudotitle toggle
86 #! $contract
87
88 # some keybindings like to change on different hooks.
89 herbstclient -i
90 while read line;do
91   case $line in
92     # remove the gap
93     nogap ) herbstclient chain ; set frame_gap -1 ;\
94       set window_gap $(window_gap-1) ; keybind $mod-0 emit_hook "\
95       set frame_border_width ${frame_border_width-1} ;\
96       pad 0 $pad ;\
97       pad 1 $pad ;\
98       # add the gap
99       gap+ ) gap=$(line/gap /); aPad=( $pad )
100       For (( i=0; i < ${#aPad[@]}; i++));do
101         aPad[$i]=$(( ${aPad[$i]} + $gap - 1))
102       done
103       herbstclient chain ;\
104       set frame_gap "$gap" ;\
105       set window_gap $gap ; set frame_border_width 0 ;\
106       pad 0 ${aPad[@]} ;\
107       pad 1 ${aPad[@]} ;\
108       keybind $mod-0 emit_hook nogap ;\
109       expand) herbstclient $expand ;\
110       contract) herbstclient $contract ;\
111     esac
112   done&
113
114 # switch to different layouts directly
115 #! keybind $mod-Alt-w set_layout vertical
116 #! keybind $mod-Alt-h set_layout horizontal

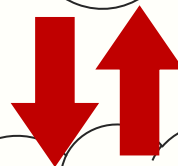
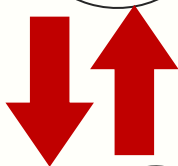
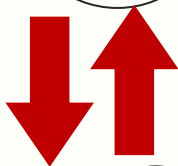
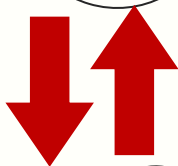
```

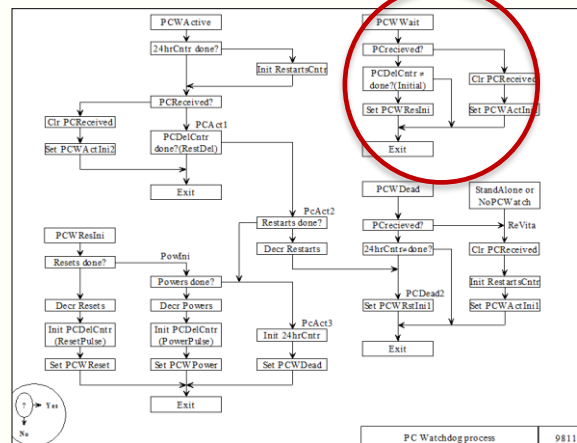
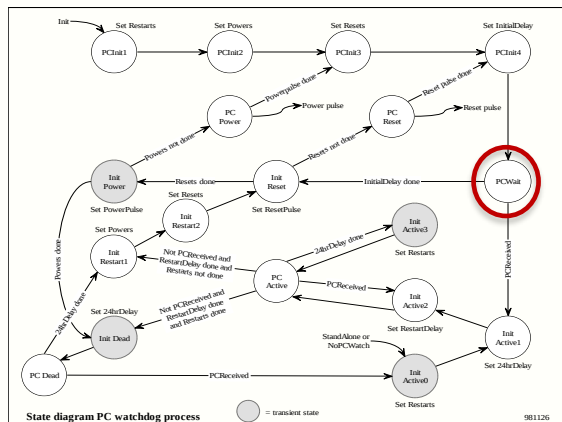
[illegible][illegible]

```

1 #!/bin/bash
2 # This script started it's life as demm_run
3
4 source $HOME/.demmurc
5
6 # If we have a directory to run in, we'll use it.
7 if [ -d "$SPATH" ]; then
8     echo "Running in $SPATH"
9     cd "$SPATH"
10 else
11     echo "No directory specified, running in $HOME"
12     cd "$HOME"
13 fi
14
15 # If we have a command to run, we'll run it.
16 if [ -n "$CMD" ]; then
17     echo "Running command: $CMD"
18     eval "$CMD"
19 else
20     echo "No command specified, running in interactive mode"
21 fi
22
23 # If we have a list of files to add to the directory, we'll add them.
24 if [ -n "$FILES" ]; then
25     echo "Adding files: $FILES"
26     for file in $FILES; do
27         cp "$file" "$HOME"
28     done
29 fi
30
31 # If we have a list of programs to launch, we'll launch them.
32 if [ -n "$PROGRAMS" ]; then
33     echo "Launching programs: $PROGRAMS"
34     for program in $PROGRAMS; do
35         eval "$program"
36     done
37 fi
38
39 # If we have a list of commands to run, we'll run them.
40 if [ -n "$COMMANDS" ]; then
41     echo "Running commands: $COMMANDS"
42     for command in $COMMANDS; do
43         eval "$command"
44     done
45 fi
46
47 # If we have a list of files to delete, we'll delete them.
48 if [ -n "$FILES_TO_DELETE" ]; then
49     echo "Deleting files: $FILES_TO_DELETE"
50     for file in $FILES_TO_DELETE; do
51         rm "$file"
52     done
53 fi
54
55 # If we have a list of directories to create, we'll create them.
56 if [ -n "$DIRECTORIES_TO_CREATE" ]; then
57     echo "Creating directories: $DIRECTORIES_TO_CREATE"
58     for directory in $DIRECTORIES_TO_CREATE; do
59         mkdir "$directory"
60     done
61 fi
62
63 # If we have a list of files to move, we'll move them.
64 if [ -n "$FILES_TO_MOVE" ]; then
65     echo "Moving files: $FILES_TO_MOVE"
66     for file in $FILES_TO_MOVE; do
67         mv "$file" "$HOME"
68     done
69 fi
70
71 # If we have a list of files to copy, we'll copy them.
72 if [ -n "$FILES_TO_COPY" ]; then
73     echo "Copying files: $FILES_TO_COPY"
74     for file in $FILES_TO_COPY; do
75         cp "$file" "$HOME"
76     done
77 fi
78
79 # If we have a list of files to rename, we'll rename them.
80 if [ -n "$FILES_TO_RENAME" ]; then
81     echo "Renaming files: $FILES_TO_RENAME"
82     for file in $FILES_TO_RENAME; do
83         mv "$file" "${file%.old}.new"
84     done
85 fi
86
87 # If we have a list of files to delete, we'll delete them.
88 if [ -n "$FILES_TO_DELETE" ]; then
89     echo "Deleting files: $FILES_TO_DELETE"
90     for file in $FILES_TO_DELETE; do
91         rm "$file"
92     done
93 fi
94
95 # If we have a list of directories to create, we'll create them.
96 if [ -n "$DIRECTORIES_TO_CREATE" ]; then
97     echo "Creating directories: $DIRECTORIES_TO_CREATE"
98     for directory in $DIRECTORIES_TO_CREATE; do
99         mkdir "$directory"
100     done
101 fi
102
103 # If we have a list of files to move, we'll move them.
104 if [ -n "$FILES_TO_MOVE" ]; then
105     echo "Moving files: $FILES_TO_MOVE"
106     for file in $FILES_TO_MOVE; do
107         mv "$file" "$HOME"
108     done
109 fi
110
111 # If we have a list of files to copy, we'll copy them.
112 if [ -n "$FILES_TO_COPY" ]; then
113     echo "Copying files: $FILES_TO_COPY"
114     for file in $FILES_TO_COPY; do
115         cp "$file" "$HOME"
116     done
117 fi
118
119 # If we have a list of files to rename, we'll rename them.
120 if [ -n "$FILES_TO_RENAME" ]; then
121     echo "Renaming files: $FILES_TO_RENAME"
122     for file in $FILES_TO_RENAME; do
123         mv "$file" "${file%.old}.new"
124     done
125 fi
126
127 # If we have a list of files to delete, we'll delete them.
128 if [ -n "$FILES_TO_DELETE" ]; then
129     echo "Deleting files: $FILES_TO_DELETE"
130     for file in $FILES_TO_DELETE; do
131         rm "$file"
132     done
133 fi
134
135 # If we have a list of directories to create, we'll create them.
136 if [ -n "$DIRECTORIES_TO_CREATE" ]; then
137     echo "Creating directories: $DIRECTORIES_TO_CREATE"
138     for directory in $DIRECTORIES_TO_CREATE; do
139         mkdir "$directory"
140     done
141 fi
142
143 # If we have a list of files to move, we'll move them.
144 if [ -n "$FILES_TO_MOVE" ]; then
145     echo "Moving files: $FILES_TO_MOVE"
146     for file in $FILES_TO_MOVE; do
147         mv "$file" "$HOME"
148     done
149 fi
150
151 # If we have a list of files to copy, we'll copy them.
152 if [ -n "$FILES_TO_COPY" ]; then
153     echo "Copying files: $FILES_TO_COPY"
154     for file in $FILES_TO_COPY; do
155         cp "$file" "$HOME"
156     done
157 fi
158
159 # If we have a list of files to rename, we'll rename them.
160 if [ -n "$FILES_TO_RENAME" ]; then
161     echo "Renaming files: $FILES_TO_RENAME"
162     for file in $FILES_TO_RENAME; do
163         mv "$file" "${file%.old}.new"
164     done
165 fi
166
167 # If we have a list of files to delete, we'll delete them.
168 if [ -n "$FILES_TO_DELETE" ]; then
169     echo "Deleting files: $FILES_TO_DELETE"
170     for file in $FILES_TO_DELETE; do
171         rm "$file"
172     done
173 fi
174
175 # If we have a list of directories to create, we'll create them.
176 if [ -n "$DIRECTORIES_TO_CREATE" ]; then
177     echo "Creating directories: $DIRECTORIES_TO_CREATE"
178     for directory in $DIRECTORIES_TO_CREATE; do
179         mkdir "$directory"
180     done
181 fi
182
183 # If we have a list of files to move, we'll move them.
184 if [ -n "$FILES_TO_MOVE" ]; then
185     echo "Moving files: $FILES_TO_MOVE"
186     for file in $FILES_TO_MOVE; do
187         mv "$file" "$HOME"
188     done
189 fi
190
191 # If we have a list of files to copy, we'll copy them.
192 if [ -n "$FILES_TO_COPY" ]; then
193     echo "Copying files: $FILES_TO_COPY"
194     for file in $FILES_TO_COPY; do
195         cp "$file" "$HOME"
196     done
197 fi
198
199 # If we have a list of files to rename, we'll rename them.
200 if [ -n "$FILES_TO_RENAME" ]; then
201     echo "Renaming files: $FILES_TO_RENAME"
202     for file in $FILES_TO_RENAME; do
203         mv "$file" "${file%.old}.new"
204     done
205 fi
206
207 # If we have a list of files to delete, we'll delete them.
208 if [ -n "$FILES_TO_DELETE" ]; then
209     echo "Deleting files: $FILES_TO_DELETE"
210     for file in $FILES_TO_DELETE; do
211         rm "$file"
212     done
213 fi
214
215 # If we have a list of directories to create, we'll create them.
216 if [ -n "$DIRECTORIES_TO_CREATE" ]; then
217     echo "Creating directories: $DIRECTORIES_TO_CREATE"
218     for directory in $DIRECTORIES_TO_CREATE; do
219         mkdir "$directory"
220     done
221 fi
222
223 # If we have a list of files to move, we'll move them.
224 if [ -n "$FILES_TO_MOVE" ]; then
225     echo "Moving files: $FILES_TO_MOVE"
226     for file in $FILES_TO_MOVE; do
227         mv "$file" "$HOME"
228     done
229 fi
230
231 # If we have a list of files to copy, we'll copy them.
232 if [ -n "$FILES_TO_COPY" ]; then
233     echo "Copying files: $FILES_TO_COPY"
234     for file in $FILES_TO_COPY; do
235         cp "$file" "$HOME"
236     done
237 fi
238
239 # If we have a list of files to rename, we'll rename them.
240 if [ -n "$FILES_TO_RENAME" ]; then
241     echo "Renaming files: $FILES_TO_RENAME"
242     for file in $FILES_TO_RENAME; do
243         mv "$file" "${file%.old}.new"
244     done
245 fi
246
247 # If we have a list of files to delete, we'll delete them.
248 if [ -n "$FILES_TO_DELETE" ]; then
249     echo "Deleting files: $FILES_TO_DELETE"
250     for file in $FILES_TO_DELETE; do
251         rm "$file"
252     done
253 fi
254
255 # If we have a list of directories to create, we'll create them.
256 if [ -n "$DIRECTORIES_TO_CREATE" ]; then
257     echo "Creating directories: $DIRECTORIES_TO_CREATE"
258     for directory in $DIRECTORIES_TO_CREATE; do
259         mkdir "$directory"
260     done
261 fi
262
263 # If we have a list of files to move, we'll move them.
264 if [ -n "$FILES_TO_MOVE" ]; then
265     echo "Moving files: $FILES_TO_MOVE"
266     for file in $FILES_TO_MOVE; do
267         mv "$file" "$HOME"
268     done
269 fi
270
271 # If we have a list of files to copy, we'll copy them.
272 if [ -n "$FILES_TO_COPY" ]; then
273     echo "Copying files: $FILES_TO_COPY"
274     for file in $FILES_TO_COPY; do
275         cp "$file" "$HOME"
276     done
277 fi
278
279 # If we have a list of files to rename, we'll rename them.
280 if [ -n "$FILES_TO_RENAME" ]; then
281     echo "Renaming files: $FILES_TO_RENAME"
282     for file in $FILES_TO_RENAME; do
283         mv "$file" "${file%.old}.new"
284     done
285 fi
286
287 # If we have a list of files to delete, we'll delete them.
288 if [ -n "$FILES_TO_DELETE" ]; then
289     echo "Deleting files: $FILES_TO_DELETE"
290     for file in $FILES_TO_DELETE; do
291         rm "$file"
292     done
293 fi
294
295 # If we have a list of directories to create, we'll create them.
296 if [ -n "$DIRECTORIES_TO_CREATE" ]; then
297     echo "Creating directories: $DIRECTORIES_TO_CREATE"
298     for directory in $DIRECTORIES_TO_CREATE; do
299         mkdir "$directory"
300     done
301 fi
302
303 # If we have a list of files to move, we'll move them.
304 if [ -n "$FILES_TO_MOVE" ]; then
305     echo "Moving files: $FILES_TO_MOVE"
306     for file in $FILES_TO_MOVE; do
307         mv "$file" "$HOME"
308     done
309 fi
310
311 # If we have a list of files to copy, we'll copy them.
312 if [ -n "$FILES_TO_COPY" ]; then
313     echo "Copying files: $FILES_TO_COPY"
314     for file in $FILES_TO_COPY; do
315         cp "$file" "$HOME"
316     done
317 fi
318
319 # If we have a list of files to rename, we'll rename them.
320 if [ -n "$FILES_TO_RENAME" ]; then
321     echo "Renaming files: $FILES_TO_RENAME"
322     for file in $FILES_TO_RENAME; do
323         mv "$file" "${file%.old}.new"
324     done
325 fi
326
327 # If we have a list of files to delete, we'll delete them.
328 if [ -n "$FILES_TO_DELETE" ]; then
329     echo "Deleting files: $FILES_TO_DELETE"
330     for file in $FILES_TO_DELETE; do
331         rm "$file"
332     done
333 fi
334
335 # If we have a list of directories to create, we'll create them.
336 if [ -n "$DIRECTORIES_TO_CREATE" ]; then
337     echo "Creating directories: $DIRECTORIES_TO_CREATE"
338     for directory in $DIRECTORIES_TO_CREATE; do
339         mkdir "$directory"
340     done
341 fi
342
343 # If we have a list of files to move, we'll move them.
344 if [ -n "$FILES_TO_MOVE" ]; then
345     echo "Moving files: $FILES_TO_MOVE"
346     for file in $FILES_TO_MOVE; do
347         mv "$file" "$HOME"
348     done
349 fi
350
351 # If we have a list of files to copy, we'll copy them.
352 if [ -n "$FILES_TO_COPY" ]; then
353     echo "Copying files: $FILES_TO_COPY"
354     for file in $FILES_TO_COPY; do
355         cp "$file" "$HOME"
356     done
357 fi
358
359 # If we have a list of files to rename, we'll rename them.
360 if [ -n "$FILES_TO_RENAME" ]; then
361     echo "Renaming files: $FILES_TO_RENAME"
362     for file in $FILES_TO_RENAME; do
363         mv "$file" "${file%.old}.new"
364     done
365 fi
366
367 # If we have a list of files to delete, we'll delete them.
368 if [ -n "$FILES_TO_DELETE" ]; then
369     echo "Deleting files
```

[illegible]





```

MovLW    ReInPC           ; Select next phase
MovWF    PCPhase          ; (See EndPCX)
Goto     EndPCX           ; Exit PC
;
;
; Phase Restart init1 PCW
;
PCRIin1   Call    EtoOPCRP      ; Init powers counter
            MovLW    Rln2PC      ; Select next phase
            MovWF    PCPhase      ; (See EndPCX)
            Goto     EndPCX       ; Exit PC
;
;
; Phase Restart init2 PCW
;
PCRIin2   Call    EtoOPCRP      ; Init resets counter
            MovLW    ReInPC      ; Select next phase
            MovWF    PCPhase      ; (See EndPCX)
            Goto     EndPCX       ; Exit PC
;
;
; Phase Active init 1 PCW
;
PCAIin1   Call    Pre24h        ; Init 24h counter
            MovLW    Ain2PC      ; Select next phase
            MovWF    PCPhase      ; (See EndPCX)
            Goto     EndPCX       ; Exit PC
;
;
; Phase Wait PCW
;
PCWait1   BTFSS    PCStat,PCRCvd ; PC received?
            Goto     PCWait1      ; Branch if not
            BCF     PCStat,PCRCvd ; Acknowledge PC received
            MovLW    Ain1PC      ; Select next phase
            MovWF    PCPhase      ; (See EndPCX)
            Goto     EndPCX       ; Exit PC
;
;
PCWait1l  MovF     PCDntnr,f      ; Check delay counter (initial delay)
            SkipIf   fCounter done (=zero) ; Skip if counter done (=zero)
            Goto     EndPC        ; Exit PC if not yet done
;
;
            MovLW    ReInPC      ; Select next phase
            MovWF    PCPhase      ; (See EndPCX)
            Goto     EndPCX       ; Exit PC
;
;
;
; Phase Reset PCW
;
PCRes     BSF     PCStat,ReaPls   ; Reset pulse on
            MovF     PCDntnr,f    ; check delay counter (reset pulse)
            SkipIf   fCounter done (=zero) ; Skip if counter done (=zero)
            Goto     EndPC        ; Exit PC if not yet done
;
;
            BCF     PCStat,ReaPls ; Reset pulse off
            MovLW    Ini4PC      ; Select next phase
            MovWF    PCPhase      ; (See EndPCX)
            Goto     EndPCX       ; Exit PC
;
;
;
; Phase Power PCW

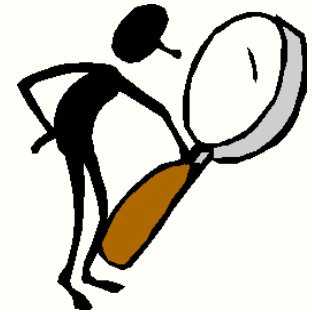
```

Case: Scrum Sprint Planning

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

Demo ??

- Give the delivery to the stakeholders
- Zip your mouth
- Keep your hands handcuffed on your back
- and o-b-s-e-r-v-e what happens
- Seeing what the stakeholders actually do provides real feedback
- Then we can 'talk business' with the stakeholders
- Is this what you do ?



The 'Demo'

Concurrent database record update

No questions – no issues !

Customer site



Demo room



Delivery Strategy Suggestions

- 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 just by us
- The users determine success, and whether they want to pay (we don't have to tell them, but it should be our attitude)
- Would you dare to deliver no-cure-no-pay ?

Some techniques shown

- Design
- Drawings
- DesignLog
- Review
- No Questions - No Issues

A Zero Defects attitude makes an immediate difference

Basic approach

- Improve the requirement
- Review
- Design implementation
- Review
- Implement (code ?)
- Review
- Test doesn't find issues (because they're not there)



Iterate fast, as needed

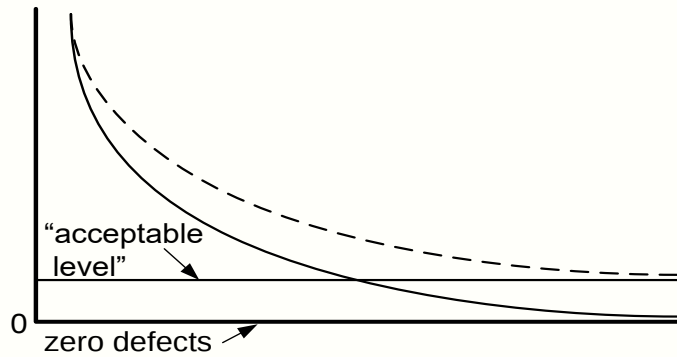
Do we deliver Zero Defect software ?

Better quality costs less

- 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 projects

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



Approaching Zero Defects is Absolutely Possible

If in doubt, let's talk about it

Niels Malotaux

Inquiries: niels@malotaux.eu

More

www.malotaux.eu/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 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 (2012) - 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/inspections

Inspection pages

- **Plan-Do-Check-Act**
 - The powerful ingredient for success

- **Business Case**

- Why we are going to improve what *Why*

- **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

- **Weekly TaskCycle**

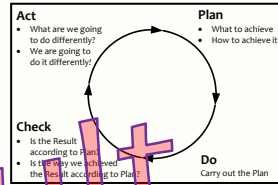
- Short term planning
 - Optimizing estimation
 - Promising what we can achieve
 - Living up to our promises

- **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



What
How much
Are we done

How

Check and learn
as early as possible

Zero
Defects
Attitude

Evolutionary Planning - Niels

Efficiency
of what we do

Effectiveness
of what we do

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