IKD training lente 2008
Succesvol Plannen van Softwareprojecten

20 en 27 mei 2008

Evolutionaire Project Management Methoden

Hoe realiseer je het beste resultaat in de kortst mogelijke tijd

Niels Malotaux

N R Malotaux - Consultancy
Bilthoven
030-2288868
niels@malotaux.nl
www.malotaux.nl

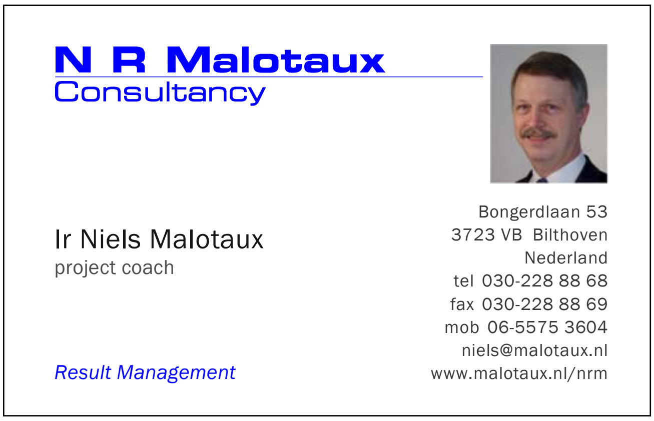
Niels Malotaux

IKD training lente 2008
Succesvol Plannen van Softwareprojecten

Niels Malotaux

Sinds 1998 treedt Niels Malotaux op als onafhankelijke consultant en project coach. Hij leert projecten en organisaties hoe ze voortaan Kwaliteit op Tijd kunnen leveren. Dat is het juiste op de juiste tijd, zonder excuses.

Niels heeft meer dan 30 jaar ervaring in het ontwikkelen van elektronische en software producten, eerst aan de TUDelft, vervolgens tijdens militaire dienst, toen bij Philips en tenslotte zo'n 20 jaar in zijn eigen systeemontwikkelings ingenieursbureau. Nu onderzoekt hij hoe menselijk gedrag het resultaat van projecten beïnvloedt en ontwikkelt hij methoden hoe we dit efficiënt en effectief kunnen verbeteren. Sinds 2001 onderwees en coachte Niels meer dan 100 projecten in 20+ verschillende organisaties in Nederland, België, Ierland, India, Japan, Roemenië en de VS, hetgeen een schat aan ervaring opleverde over wat beter en wat minder goed werkt. Hij spreekt regelmatig op conferenties en heeft 5 boekjes gepubliceerd. Kijk op www.malotaux.nl/nrm voor meer informatie.

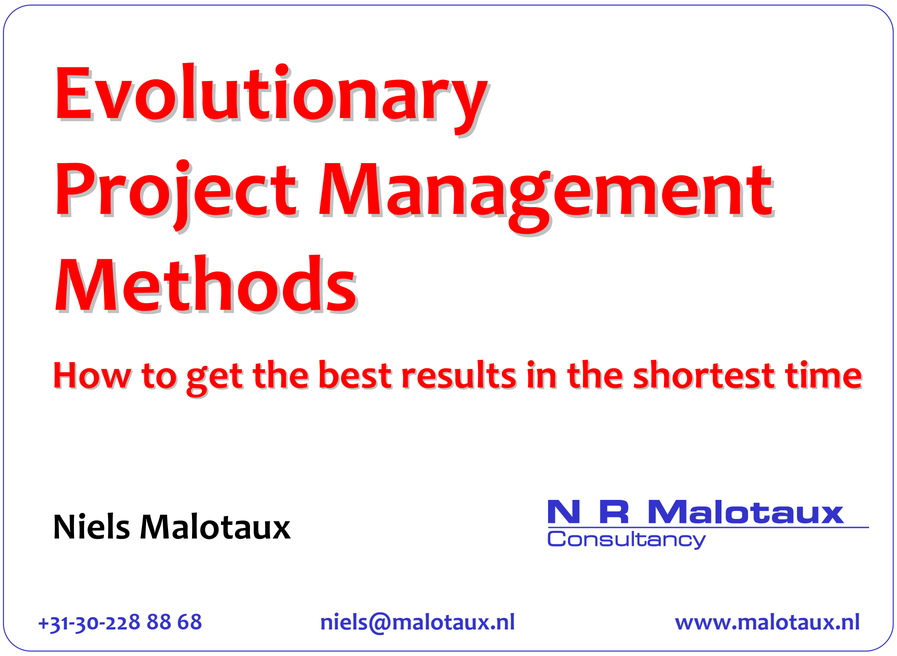


N R Malotaux
Consultancy

Ir Niels Malotaux
project coach

Bongerdlaan 53
3723 VB Bilthoven
Nederland
tel 030-228 88 68
fax 030-228 88 69
mob 06-5575 3604
niels@malotaux.nl
www.malotaux.nl/nrm

Result Management



Evolutionary Project Management Methods

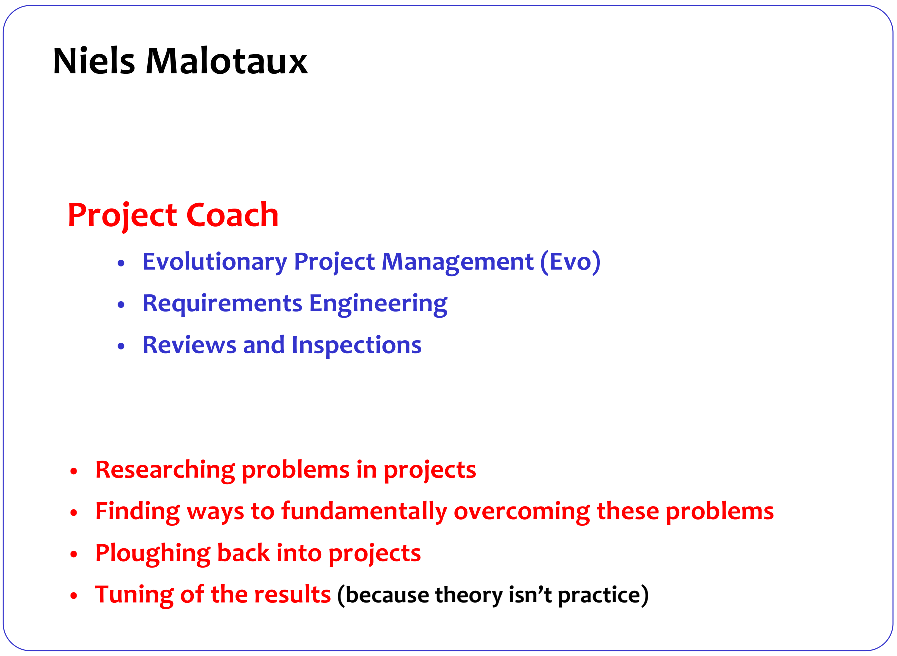
How to get the best results in the shortest time

Niels Malotaux

N R Malotaux
Consultancy

+31-30-228 88 68 niels@malotaux.nl www.malotaux.nl

1



Niels Malotaux

Project Coach

- Evolutionary Project Management (Evo)
- Requirements Engineering
- Reviews and Inspections

- Researching problems in projects
- Finding ways to fundamentally overcoming these problems
- Ploughing back into projects
- Tuning of the results (because theory isn't practice)

2

IKD training lente 2008

Dag 1:

- Inleiding, menselijke factoren, oefening in o.m. tijdschatten, basiselementen voor project planning
- Project planning in de praktijk, introductie TimeLine, eerste uitwerking van je individuele TimeLine
- Plannen van het werk voor de komende week

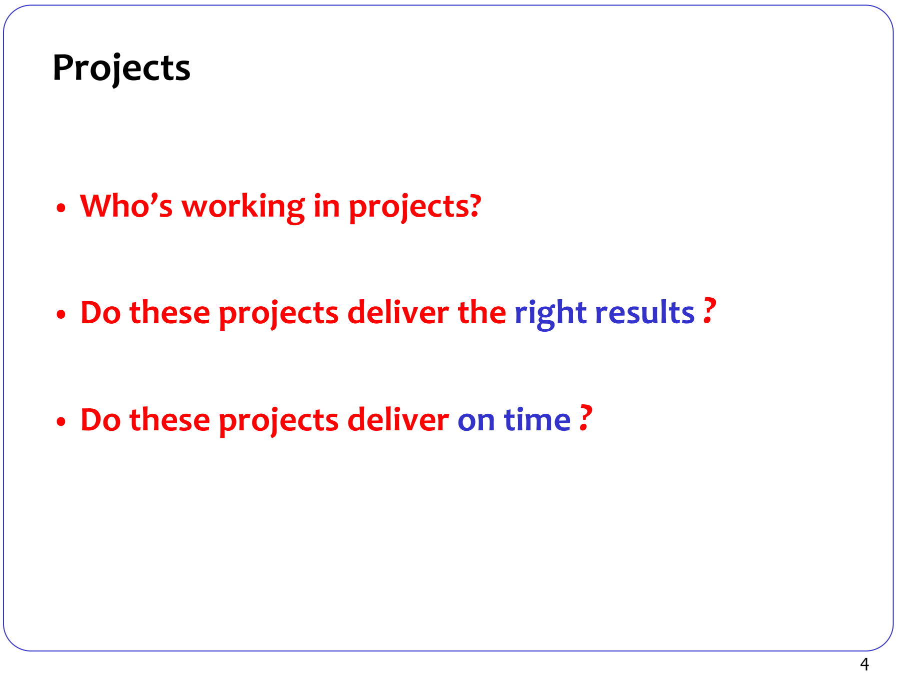
Huiswerk:

- Uitvoeren van je planning, bepalen wat je de volgende week denkt te moeten gaan doen, waarom, en hoeveel tijd dat gaat kosten. Verzamelen TimeLine materiaal

Dag 2:

- Analyse van de resultaten van de afgelopen week
- Requirements en Design
- Project risico’s, test en review technieken, inspectie van requirements document
- Discussie over het geleerde van de afgelopen weken
- Aanscherpen van Evo elementen naar aanleiding van de ervaring van de afgelopen weken
- Plannen van het werk voor de komende week

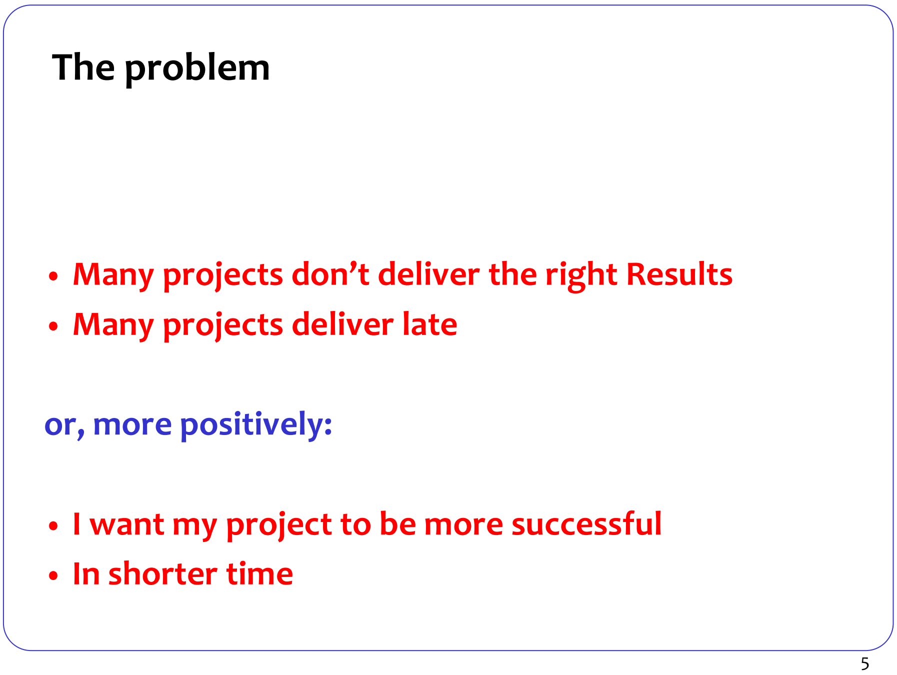
3



Projects

- Who’s working in projects?
- Do these projects deliver the right results ?
- Do these projects deliver on time ?

4



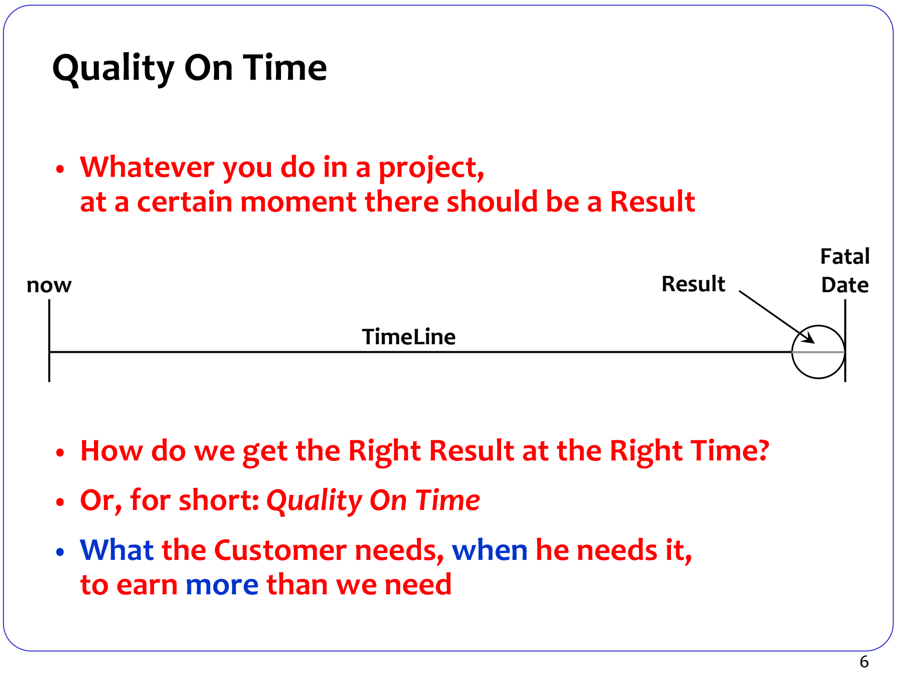
The problem

- Many projects don't deliver the right Results
- Many projects deliver late

or, more positively:

- I want my project to be more successful
- In shorter time

5



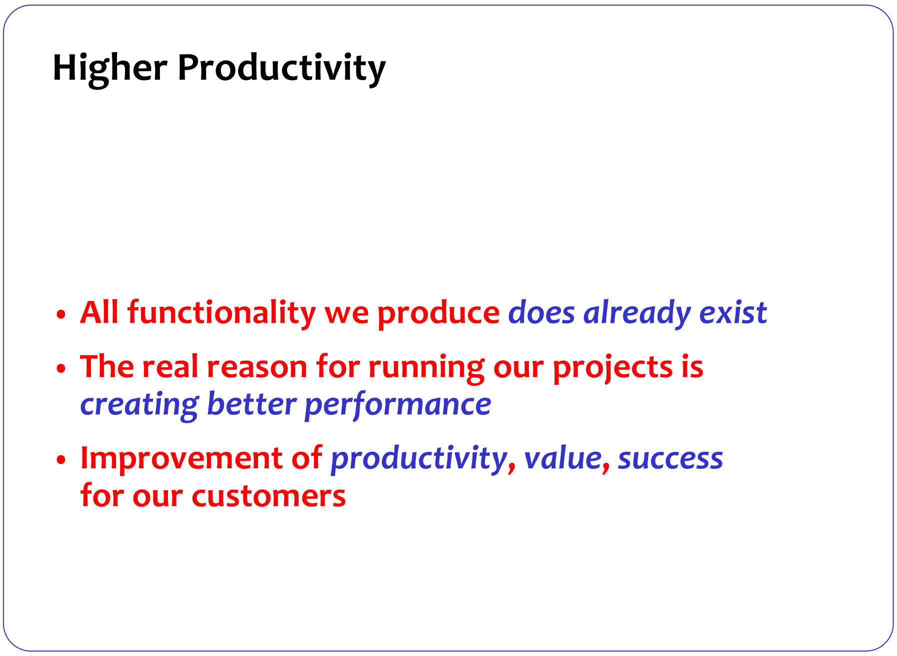
Quality On Time

- Whatever you do in a project,
  at a certain moment there should be a Result

now
TimeLine
Result
Fatal
Date

- How do we get the Right Result at the Right Time?
- Or, for short: *Quality On Time*
- What the Customer needs, when he needs it,
  to earn more than we need

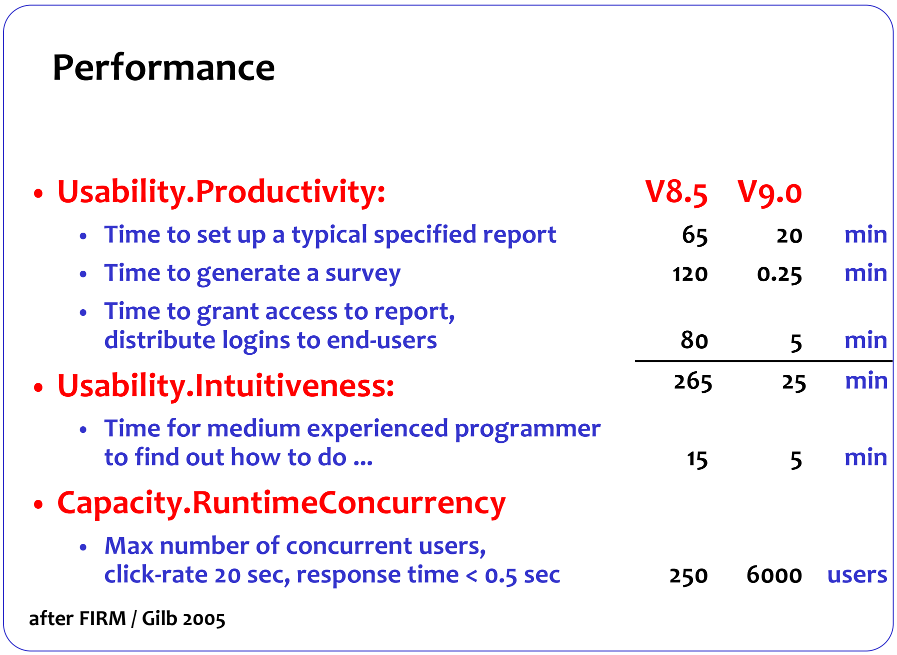
6



Higher Productivity

- All functionality we produce *does already exist*
- The real reason for running our projects is *creating better performance*
- Improvement of *productivity*, *value*, *success* for our customers

7

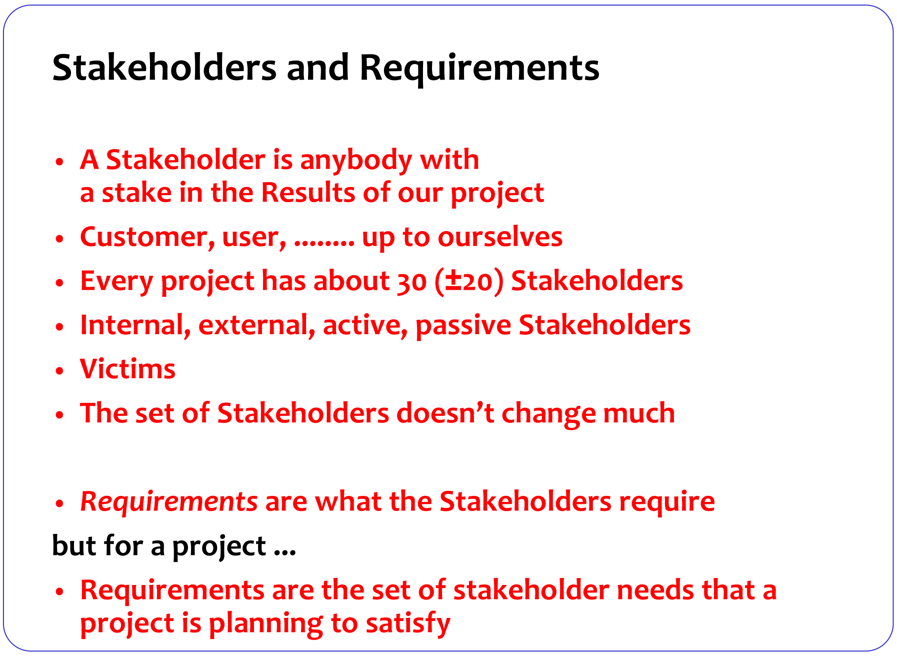


Performance

	V8.5	V9.0	
• Usability.Productivity:			
• Time to set up a typical specified report	65	20	min
• Time to generate a survey	120	0.25	min
• Time to grant access to report, distribute logins to end-users	80	5	min
	265	25	min
• Usability.Intuitiveness:			
• Time for medium experienced programmer to find out how to do ...	15	5	min
• Capacity.RuntimeConcurrency			
• Max number of concurrent users, click-rate 20 sec, response time < 0.5 sec	250	6000	users

after FIRM / Gilb 2005

8



Stakeholders and Requirements

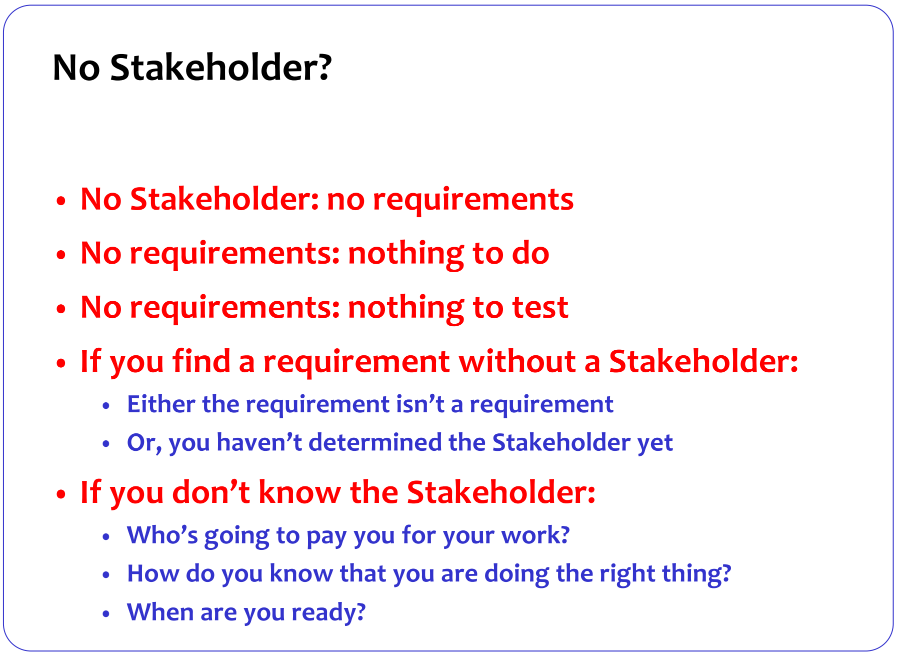
- A Stakeholder is anybody with a stake in the Results of our project
- Customer, user, ... up to ourselves
- Every project has about 30 (±20) Stakeholders
- Internal, external, active, passive Stakeholders
- Victims
- The set of Stakeholders doesn't change much

- *Requirements* are what the Stakeholders require

but for a project ...

- Requirements are the set of stakeholder needs that a project is planning to satisfy

9

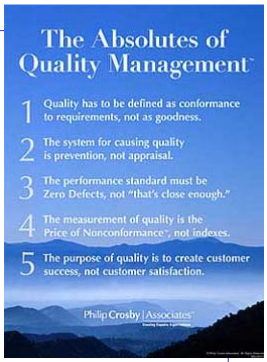


No Stakeholder?

- No Stakeholder: no requirements
- No requirements: nothing to do
- No requirements: nothing to test
- If you find a requirement without a Stakeholder:
  - Either the requirement isn't a requirement
  - Or, you haven't determined the Stakeholder yet
- If you don't know the Stakeholder:
  - Who's going to pay you for your work?
  - How do you know that you are doing the right thing?
  - When are you ready?

10

Absolutes of Quality



The Absolutes of Quality Management™

1 Quality has to be defined as conformance to requirements, not as goodness.

2 The system for causing quality is prevention, not appraisal.

3 The performance standard must be Zero Defects, not "that's close enough."

4 The measurement of quality is the Price of Nonconformance™, not indexes.

5 The purpose of quality is to create customer success, not customer satisfaction.

Philip Crosby | Associates™

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

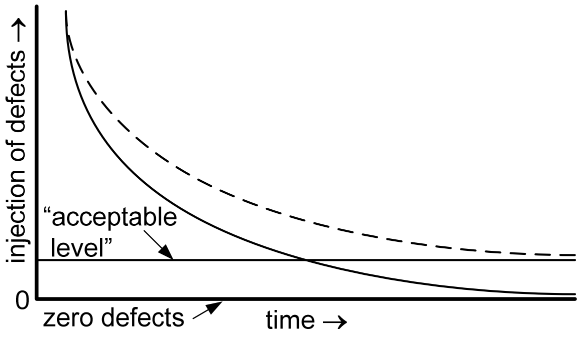
Philip Crosby, 1970

- The purpose is customer success
  (not customer satisfaction)

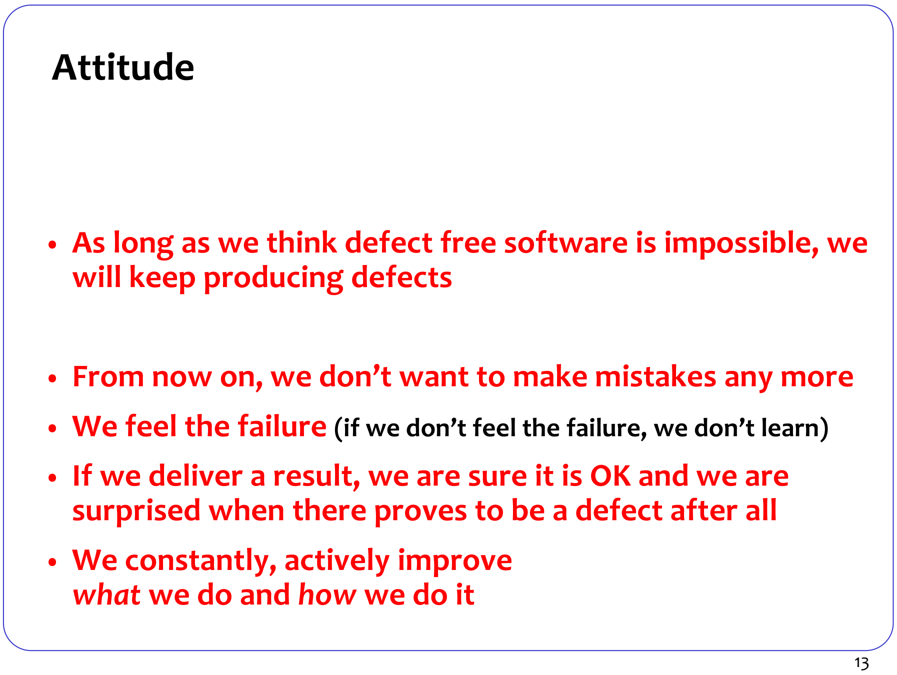
Added by Philip Crosby Associates, 2004

Is defect free software possible?

- Zero Defects is an asymptote



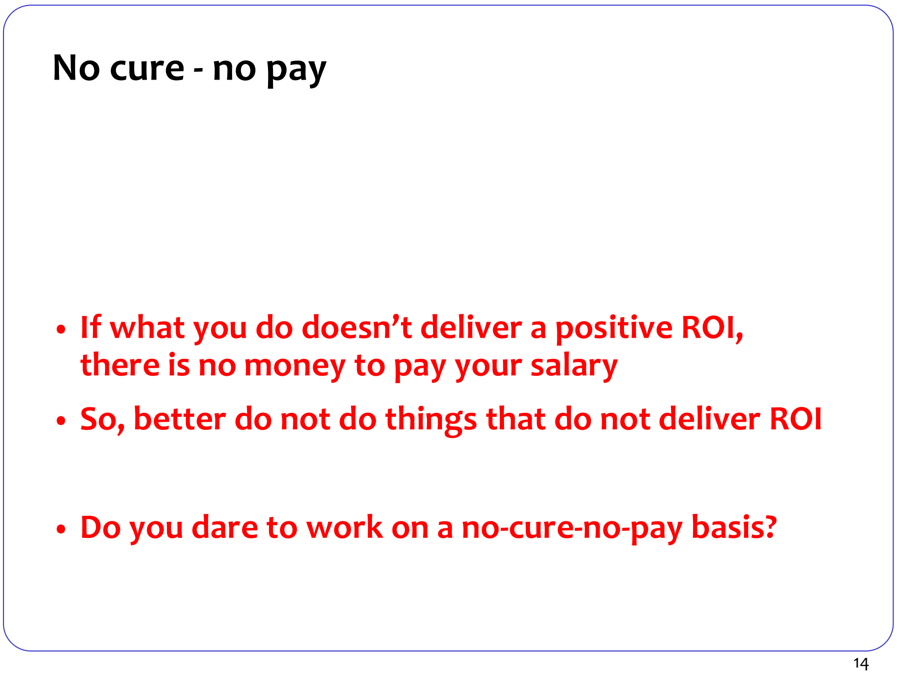
- When Philip Crosby started with Zero Defects in 1961, errors dropped by 40% almost immediately



Attitude

- As long as we think defect free software 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 the failure, we don't learn)
- If we deliver a result, we are sure it is OK and we are surprised when there proves to be a defect after all
- We constantly, actively improve *what* we do and *how* we do it

13



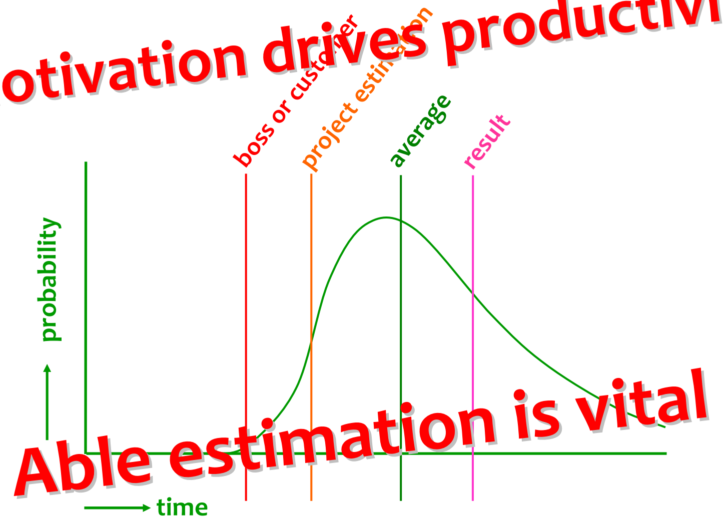
No cure - no pay

- If what you do doesn't deliver a positive ROI, there is no money to pay your salary
- So, better do not do things that do not deliver ROI
- Do you dare to work on a no-cure-no-pay basis?

14

Lead time

Motivation drives productivity



Able estimation is vital

Estimation Exercise

Are you an optimistic or a realistic estimator?

Let's find out !

Project:
Multiplying two numbers of 4 figures

How many seconds would you need to complete this Project?

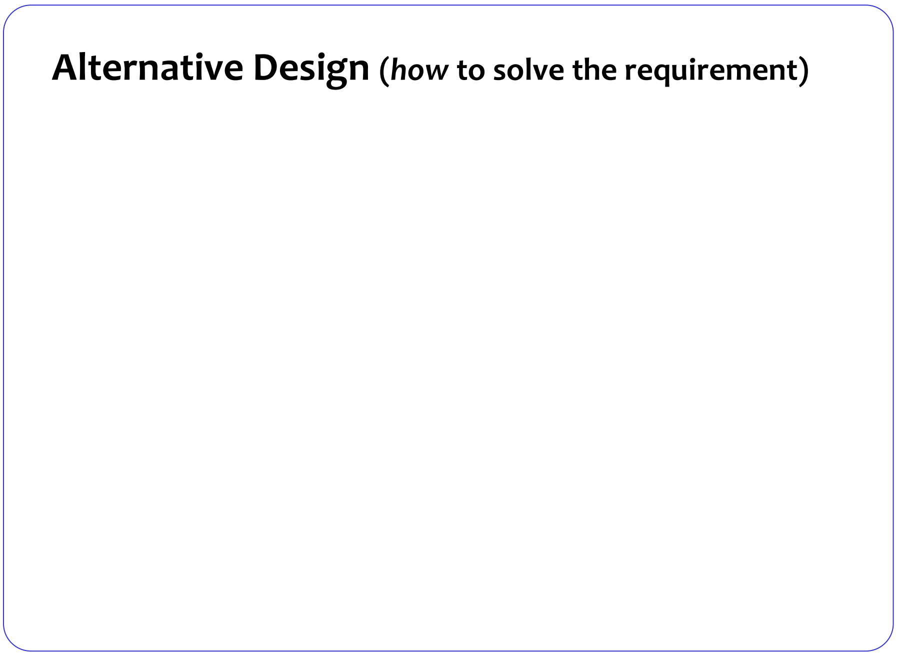
Is this what you did?

17

Defect rate

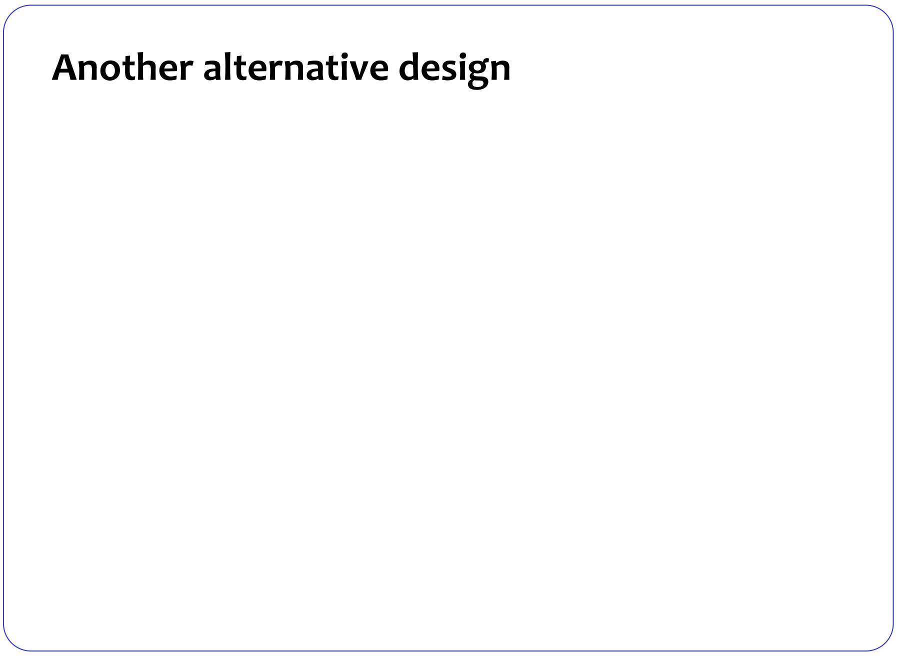
- Before test ?
- After test ?

18



Alternative Design (*how* to solve the requirement)

19



Another alternative design

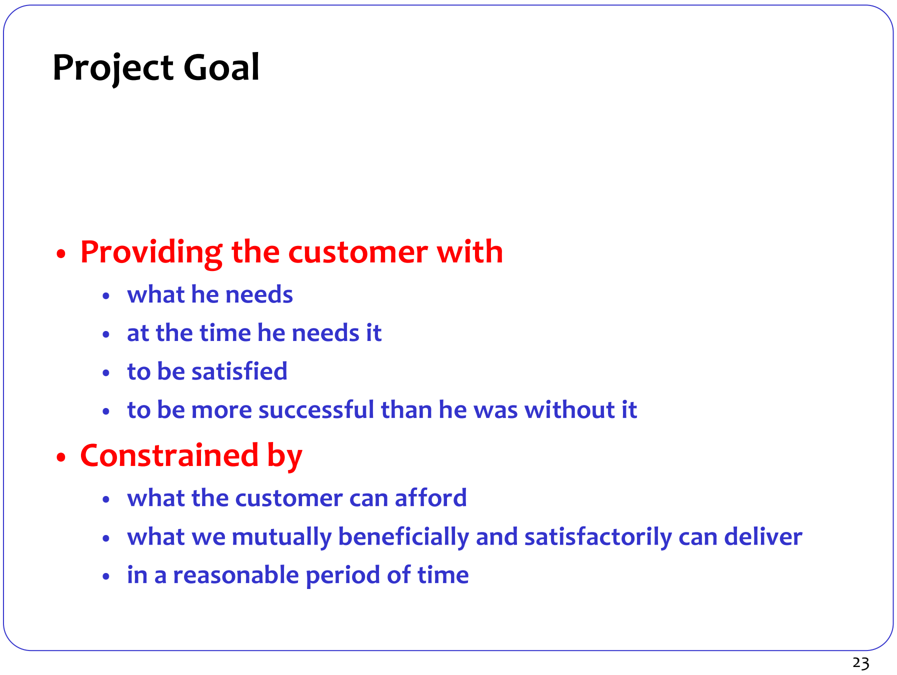
20

21

Elements in the exercise

- Estimation, optimistic / realistic
- Interrupts
- Test, test strategy
- Defect-rate
- Design
- Requirements
- Assumptions

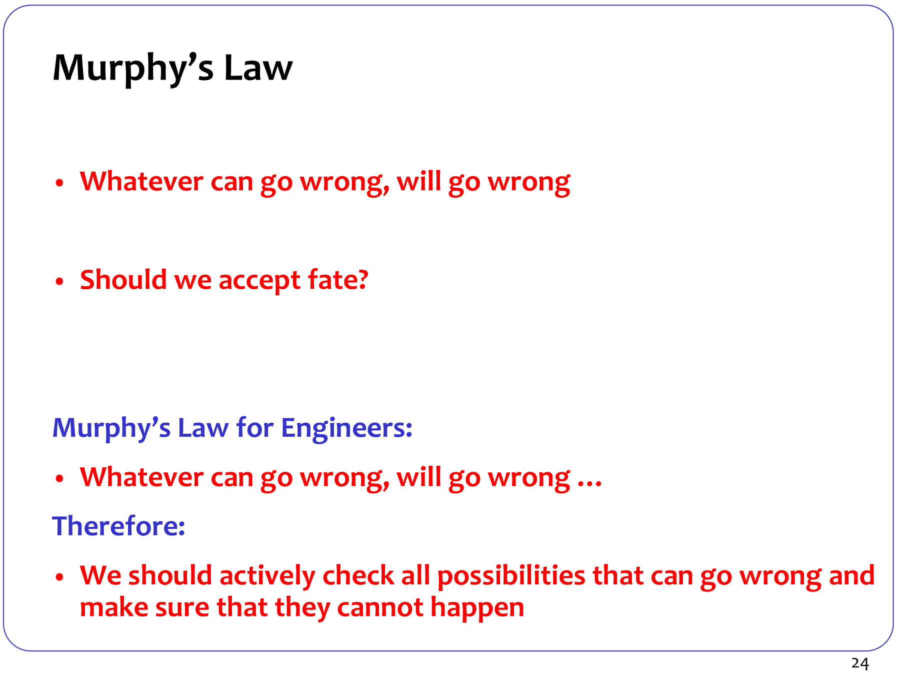
22



Project Goal

- 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
  - what the customer can afford
  - what we mutually beneficially and satisfactorily can deliver
  - in a reasonable period of time

23



Murphy’s Law

- Whatever can go wrong, will go wrong
- Should we accept fate?

Murphy’s Law for Engineers:

- Whatever can go wrong, will go wrong …

Therefore:

- We should actively check all possibilities that can go wrong and make sure that they cannot happen

24

Human Behavior

- Systems are conceived, designed, implemented, maintained, used, and tolerated *(or not)* by people
- People react quite predictably
- However, often differently from what we intuitively think

- Most project process approaches (as well as developers) *ignore* human behavior, incorrectly *assume* behavior, or decide how people *should* behave (ha ha)
- To succeed in projects, we must study and adapt to *real* behavior rather than *assumed* behavior
- Going against our genes is a lost battle

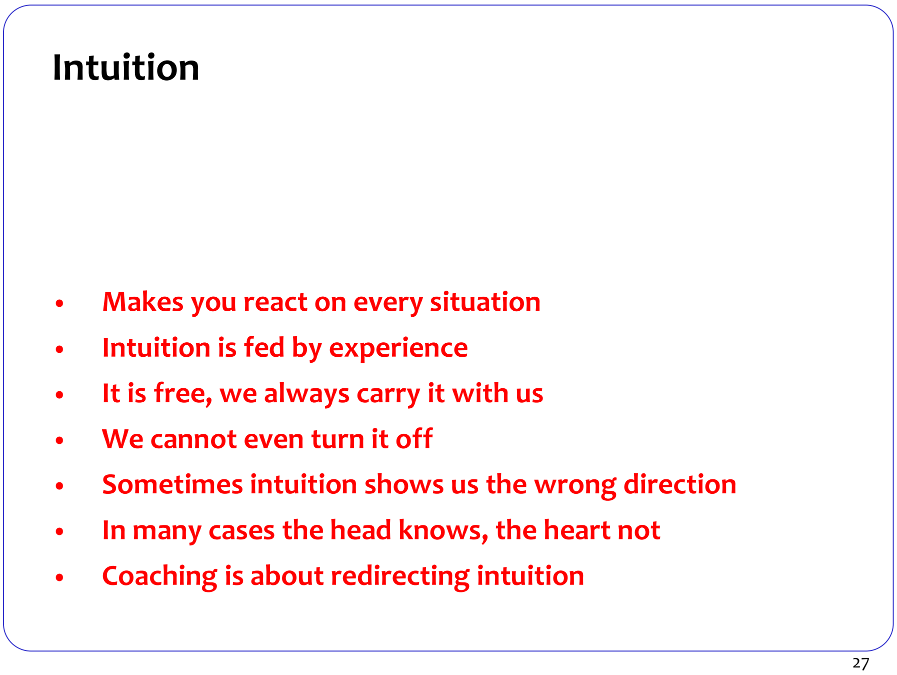
25

Discipline

- Control of wrong inclinations
- Even if we *know* how it *should* be done …
  (if nobody is watching …)
- Discipline is very difficult
- Romans 7:19
  - For the good that I would I do not …

→ We must help each other (watching over the shoulder)
→ Rapid success helps (within two weeks)
→ Making mistakes helps (if we immediately learn from them)

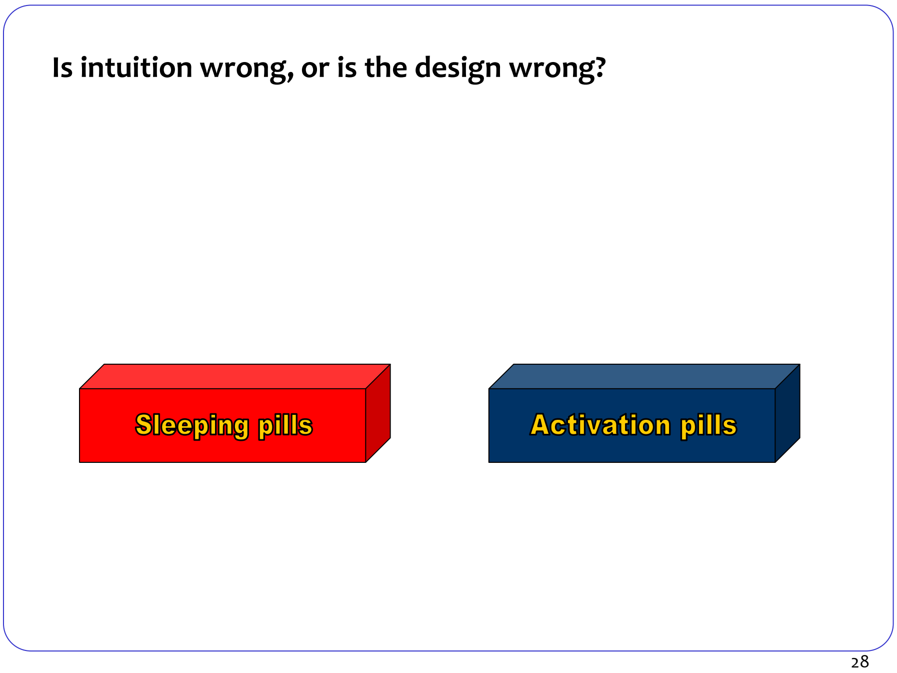
26



Intuition

- Makes you react on every situation
- Intuition is fed by experience
- It is free, we always carry it with us
- We cannot even turn it off
- Sometimes intuition shows us the wrong direction
- In many cases the head knows, the heart not
- Coaching is about redirecting intuition

27

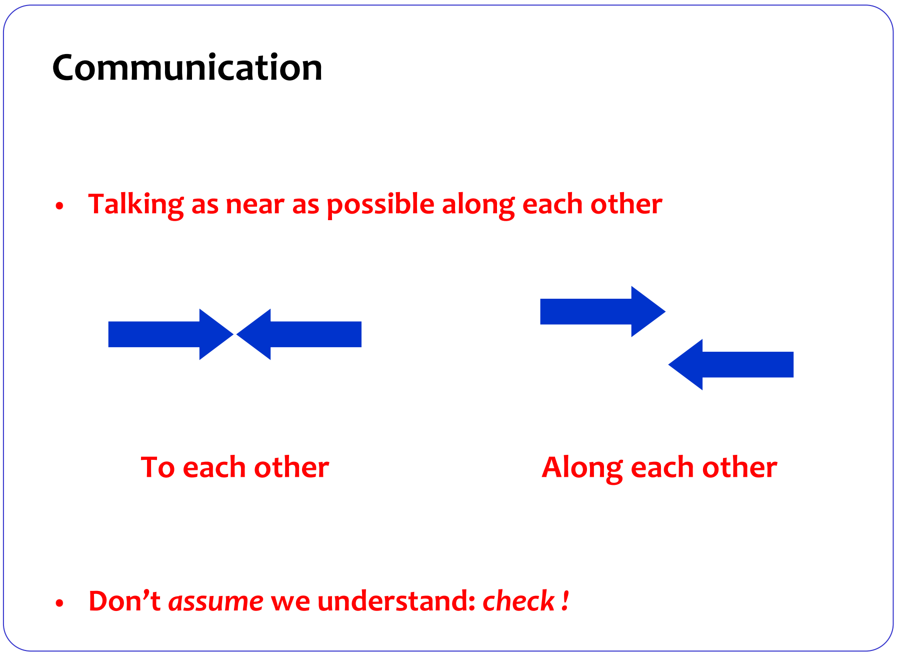


Is intuition wrong, or is the design wrong?

Sleeping pills

Activation pills

28



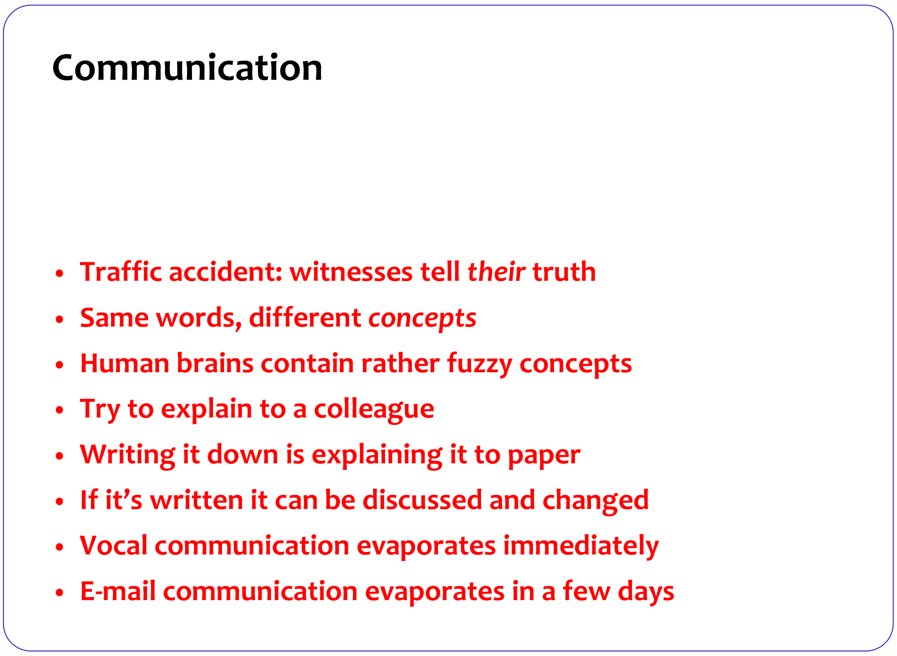
Communication

- Talking as near as possible along each other

To each other
Along each other

- Don't *assume* we understand: *check !*

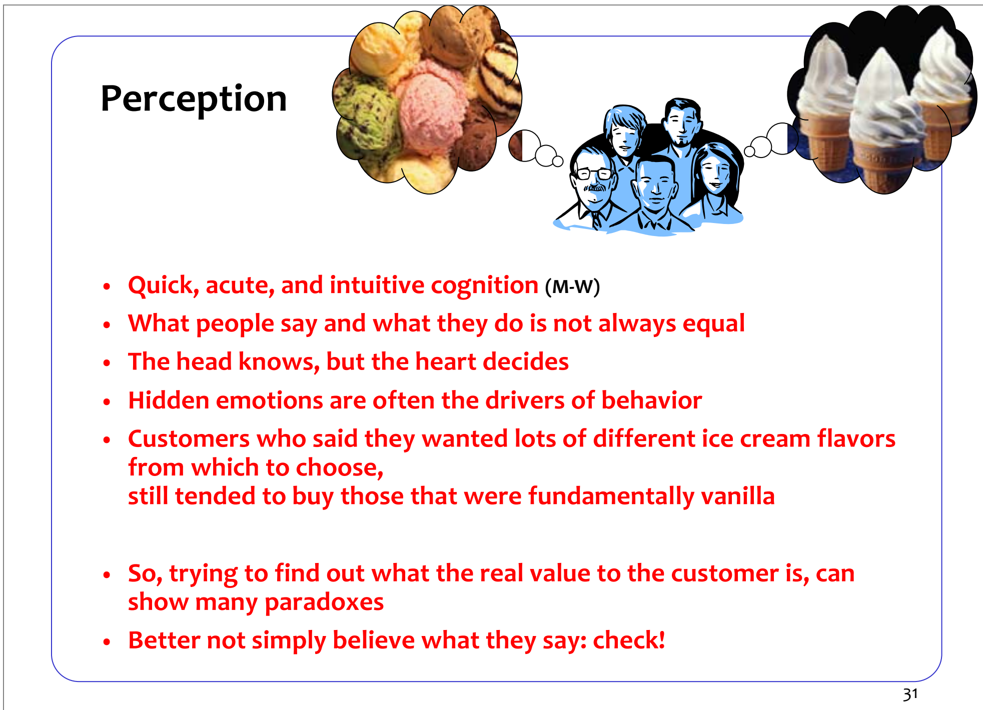
29



Communication

- Traffic accident: witnesses tell *their* truth
- Same words, different *concepts*
- Human brains contain rather fuzzy concepts
- Try to explain to a colleague
- Writing it down is explaining it to paper
- If it's written it can be discussed and changed
- Vocal communication evaporates immediately
- E-mail communication evaporates in a few days

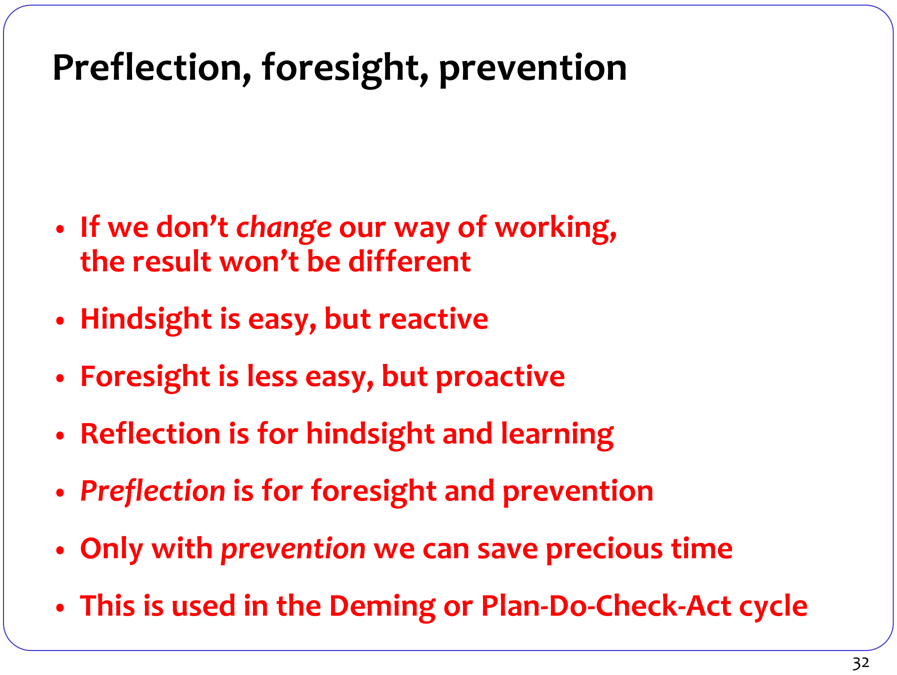
30



Perception

- Quick, acute, and intuitive cognition (M-W)
- What people say and what they do is not always equal
- The head knows, but the heart decides
- Hidden emotions are often the drivers of behavior
- Customers who said they wanted lots of different ice cream flavors from which to choose,
  still tended to buy those that were fundamentally vanilla
- So, trying to find out what the real value to the customer is, can show many paradoxes
- Better not simply believe what they say: check!

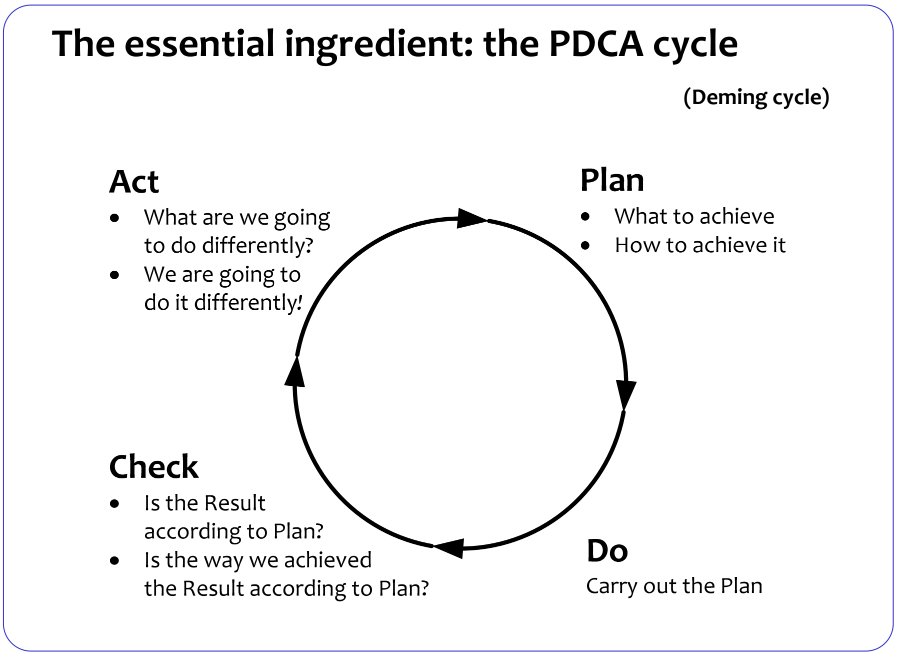
31



Preflection, foresight, prevention

- If we don't *change* our way of working,
  the result won't be different
- Hindsight is easy, but reactive
- Foresight is less easy, but proactive
- Reflection is for hindsight and learning
- *Preflection* is for foresight and prevention
- Only with *prevention* we can save precious time
- This is used in the Deming or Plan-Do-Check-Act cycle

32



The essential ingredient: the PDCA cycle

(Deming cycle)

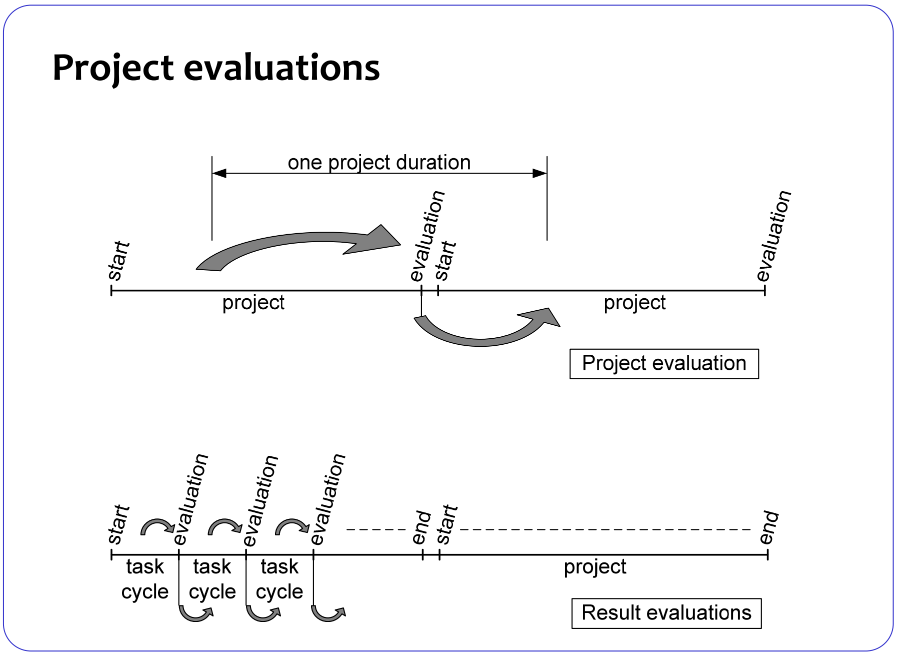
Act
- What are we going to do differently?
- We are going to do it differently!

Plan
- What to achieve
- How to achieve it

Check
- Is the Result according to Plan?
- Is the way we achieved the Result according to Plan?

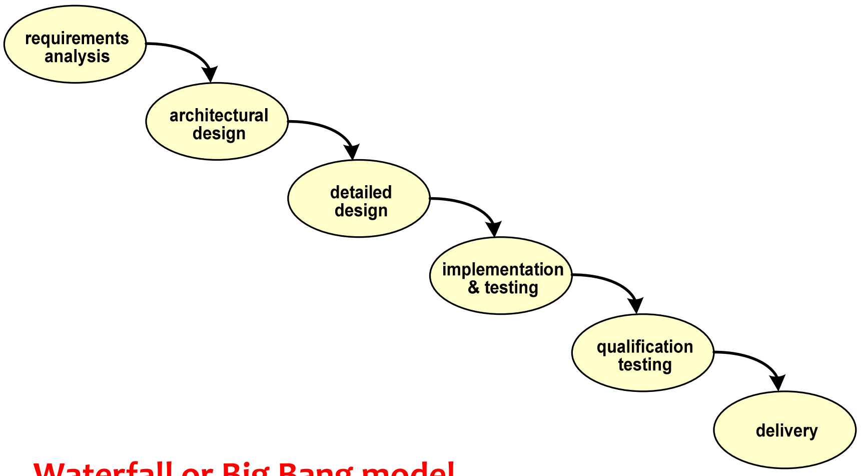
Do
Carry out the Plan

33



Project evaluations

34



requirements analysis

architectural design

detailed design

implementation & testing

qualification testing

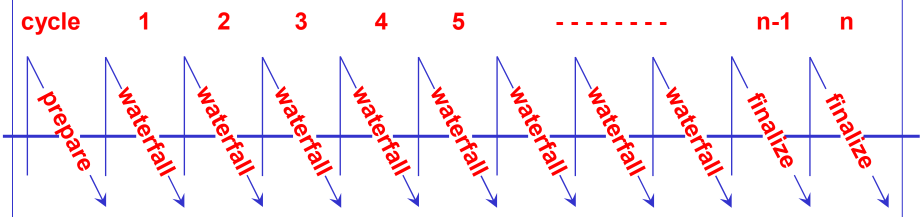
delivery

Waterfall or Big Bang model
= production
= fixed contract model (signed with blood)

35

Using many waterfalls

of growing functionality



cycle 1 2 3 4 5 - - - - - - - - n-1 n

prepare

waterfall

waterfall

waterfall

waterfall

waterfall

waterfall

waterfall

waterfall

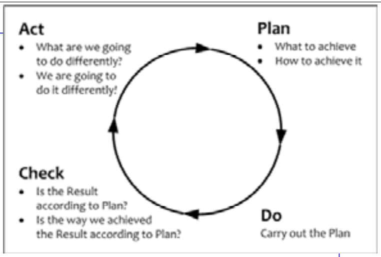
finalize

finalize

36

Knowledge

how to achieve the goal


Act
• What are we going to do differently?
• We are going to do it differently!
Plan
• What to achieve
• How to achieve it
Check
• Is the Result according to Plan?
• Is the way we achieved the Result according to Plan?
Do
Carry out the Plan

If we

- Use very short Plan-Do-Check-Act cycles
- Constantly selecting the most important things to do

doing the right things

then we can

- Most quickly learn what the real requirements are
- Learn how to most effectively and efficiently realize these requirements

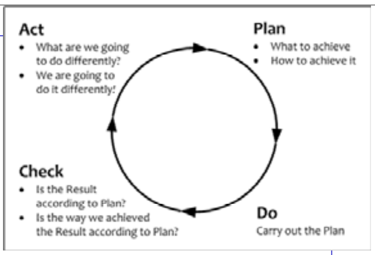
and we can

- Spot problems quicker, allowing more time to do something about them

doing the right things right

37

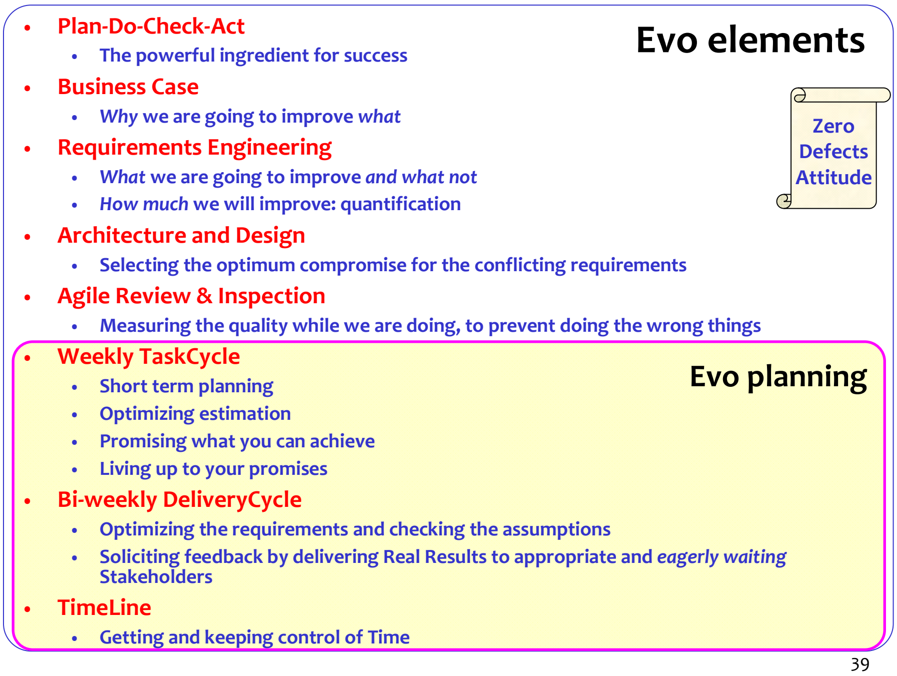
Evo


Act
• What are we going to do differently?
• We are going to do it differently!
Plan
• What to achieve
• How to achieve it
Check
• Is the Result according to Plan?
• Is the way we achieved the Result according to Plan?
Do
Carry out the Plan

- Evo (short for Evolutionary...) uses PDCA consistently
- Applying the PDCA-cycle actively, deliberately, rapidly and frequently, for *Product*, *Project* and *Process*, based on ROI and highest value
- Combining Planning, Requirements- and Risk-Management into *Result Management*
- We know we are not perfect, but the customer should never find out
- Evo is about delivering Real Stuff to Real Stakeholders doing Real Things

"Nothing beats the Real Thing"

38



Evo elements

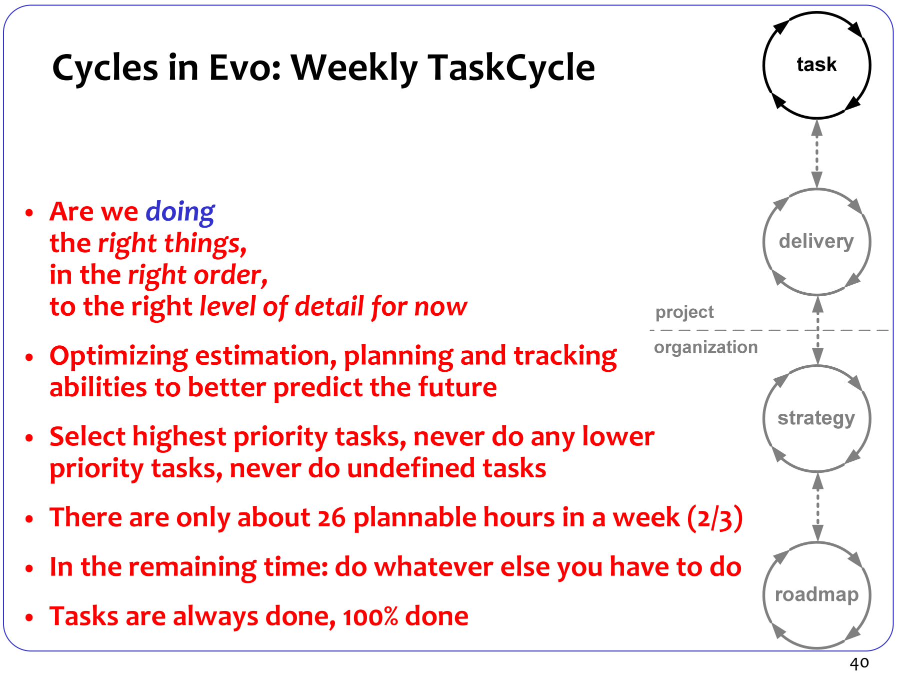
- Plan-Do-Check-Act
  - The powerful ingredient for success
- Business Case
  - *Why* we are going to improve *what*
- Requirements Engineering
  - *What* we are going to improve *and what not*
  - *How much* we will improve: quantification
- Architecture and Design
  - Selecting the optimum compromise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

Zero Defects Attitude

Evo planning

- Weekly TaskCycle
  - Short term planning
  - Optimizing estimation
  - Promising what you can achieve
  - Living up to your promises
- Bi-weekly DeliveryCycle
  - Optimizing the requirements and checking the assumptions
  - Soliciting feedback by delivering Real Results to appropriate and *eagerly waiting* Stakeholders
- TimeLine
  - Getting and keeping control of Time

39

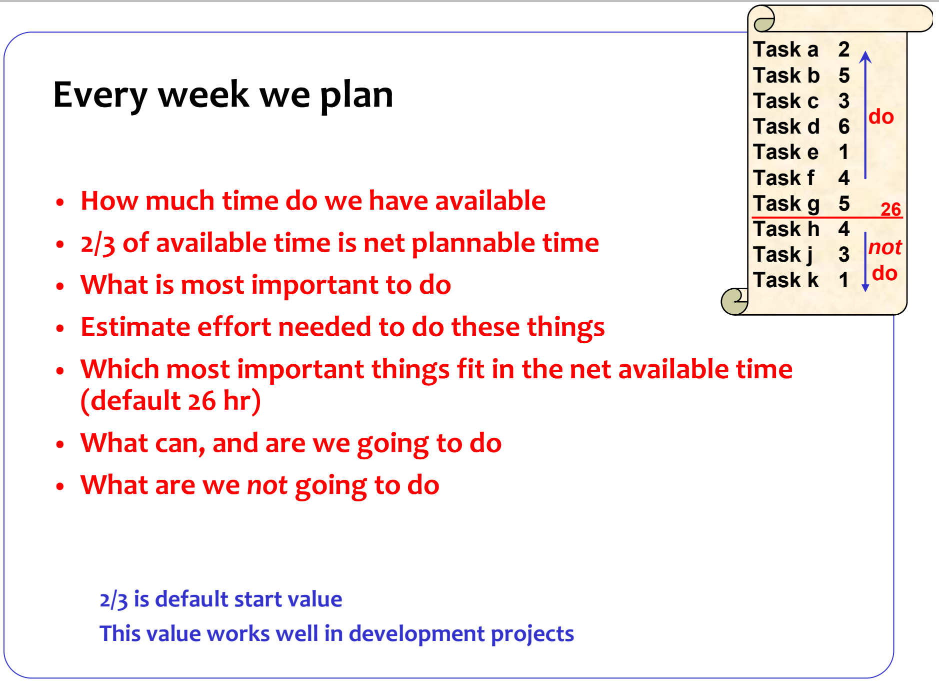


Cycles in Evo: Weekly TaskCycle

task
delivery
project
organization
strategy
roadmap

- Are we *doing* the *right things*, in the *right order*, to the right *level of detail for now*
- Optimizing estimation, planning and tracking abilities to better predict the future
- Select highest priority tasks, never do any lower priority tasks, never do undefined tasks
- There are only about 26 plannable hours in a week (2/3)
- In the remaining time: do whatever else you have to do
- Tasks are always done, 100% done

40



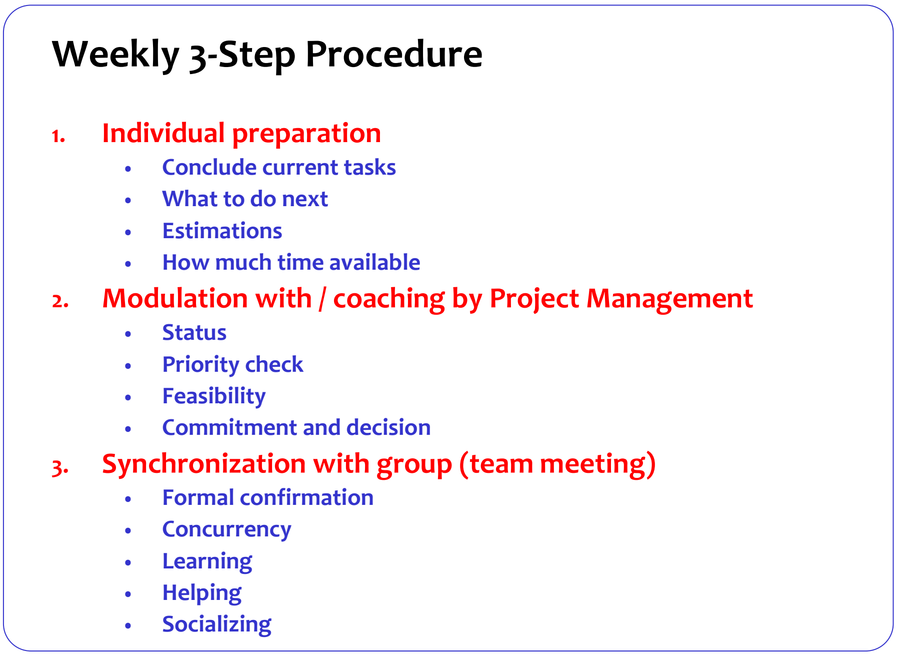
Every week we plan

Task a	2	do
Task b	5	
Task c	3	
Task d	6	
Task e	1	
Task f	4	
Task g	5	26
Task h	4	*not* do
Task j	3	
Task k	1	

- How much time do we have available
- 2/3 of available time is net plannable time
- What is most important to do
- Estimate effort needed to do these things
- Which most important things fit in the net available time (default 26 hr)
- What can, and are we going to do
- What are we *not* going to do

2/3 is default start value
This value works well in development projects

41



Weekly 3-Step Procedure

1. Individual preparation
   - Conclude current tasks
   - What to do next
   - Estimations
   - How much time available
2. Modulation with / coaching by Project Management
   - Status
   - Priority check
   - Feasibility
   - Commitment and decision
3. Synchronization with group (team meeting)
   - Formal confirmation
   - Concurrency
   - Learning
   - Helping
   - Socializing

42

Cycles in Evo: DeliveryCycle

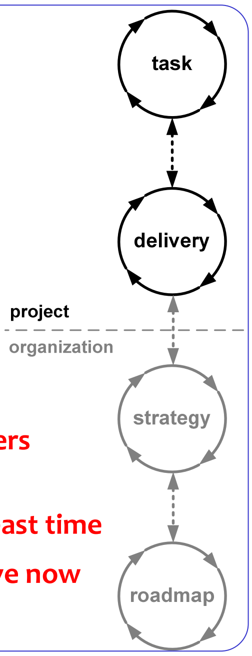
- Are we delivering
  the *right things,*
  in the *right order*
  to the right *level of detail for now*
- Optimizing requirements
  and checking assumptions

a. What will generate the optimum feedback

b. We deliver only to *eagerly waiting* stakeholders

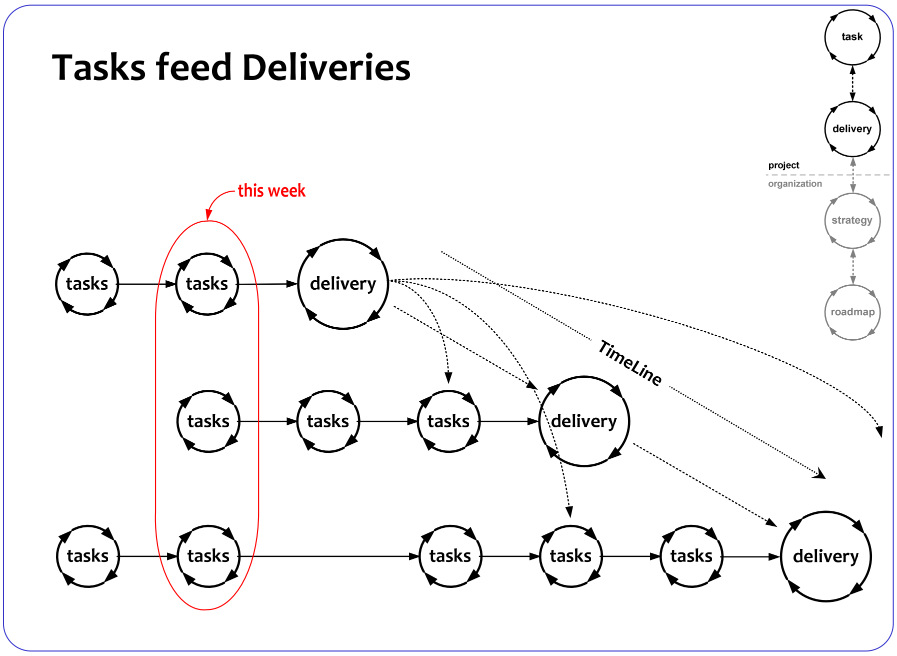
c. Delivering the juiciest, most important
   stakeholder values that can be made in the least time

- What will make Stakeholders more productive now
- Not more than 2 weeks



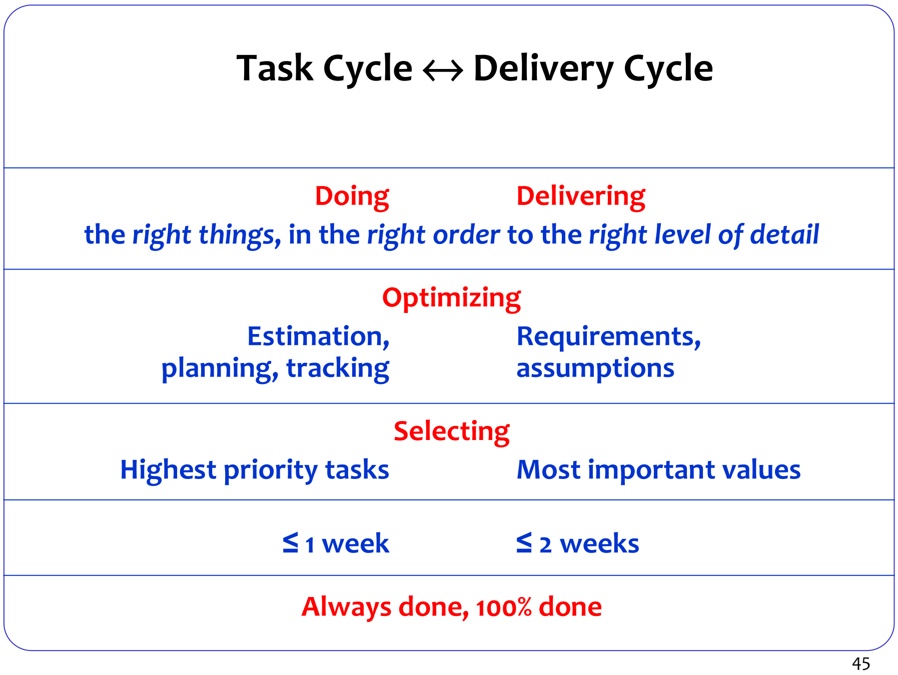
43

Tasks feed Deliveries



44

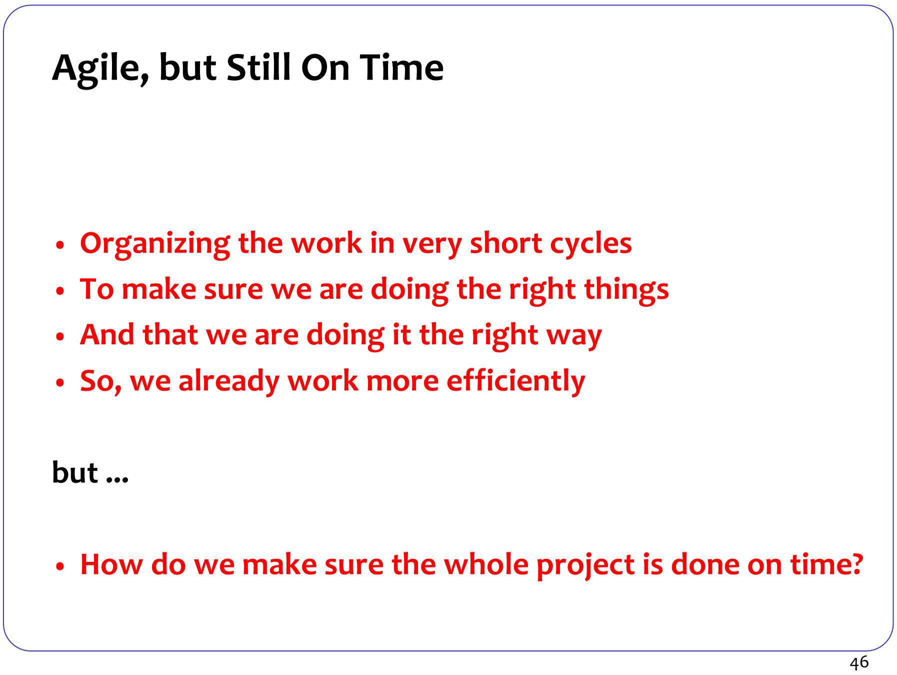
Dag1



Task Cycle ↔ Delivery Cycle

Doing	Delivering
the *right things*, in the *right order* to the *right level of detail*	
Optimizing	
Estimation, planning, tracking	Requirements, assumptions
Selecting	
Highest priority tasks	Most important values
≤ 1 week	≤ 2 weeks
Always done, 100% done	

45



Agile, but Still On Time

- Organizing the work in very short cycles
- To make sure we are doing the right things
- And that we are doing it the right way
- So, we already work more efficiently

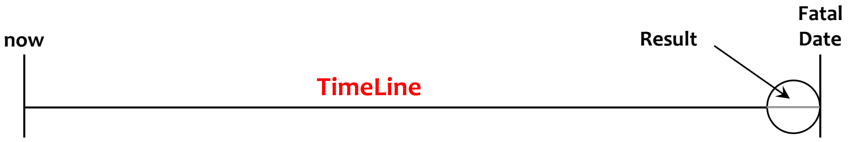
but ...

- How do we make sure the whole project is done on time?

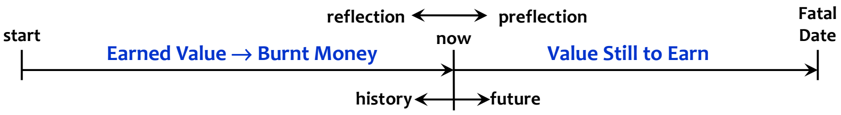
46

TimeLine

- Whatever you do from now,
  at a certain date there should be a Result

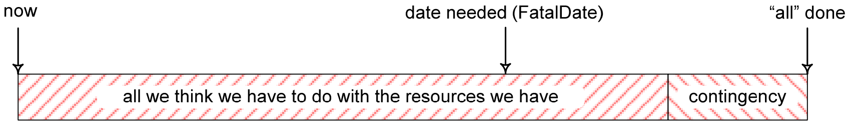


- We use TimeLine to control of longer periods of time
- Earned Value (hindsight) *and* Value Still to Earn (foresight)

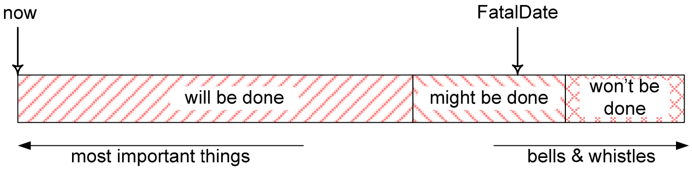


TimeLine

What the customer wants, he cannot afford

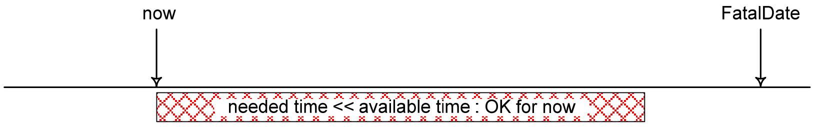


Standard Projects



Evo

If it easily fits ...



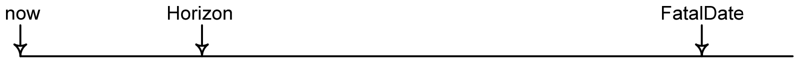
now

FatalDate

needed time << available time : OK for now

49

Result to Tasks and back



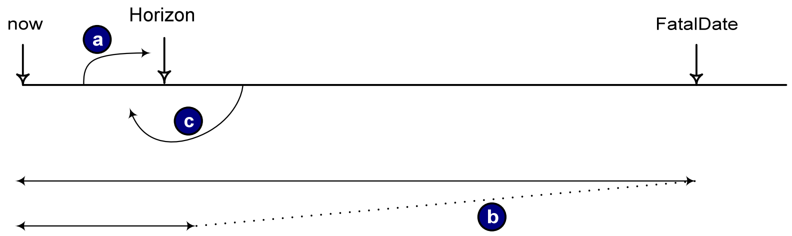
now

Horizon

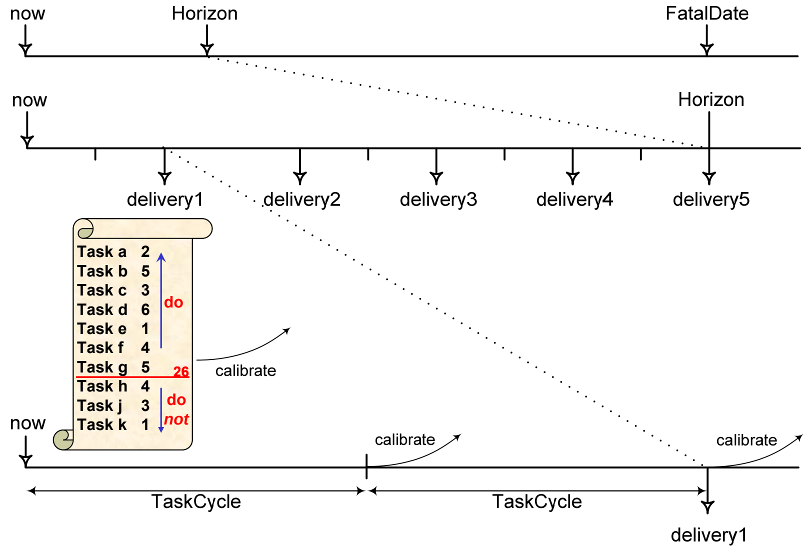
FatalDate

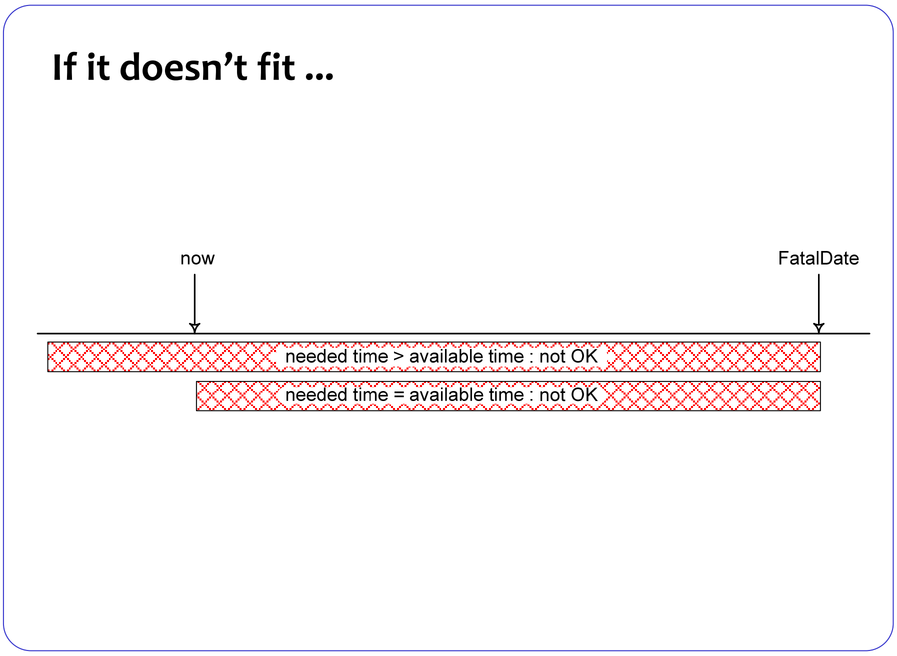
50

Setting a Horizon



Result to Tasks and back





If it doesn't fit ...

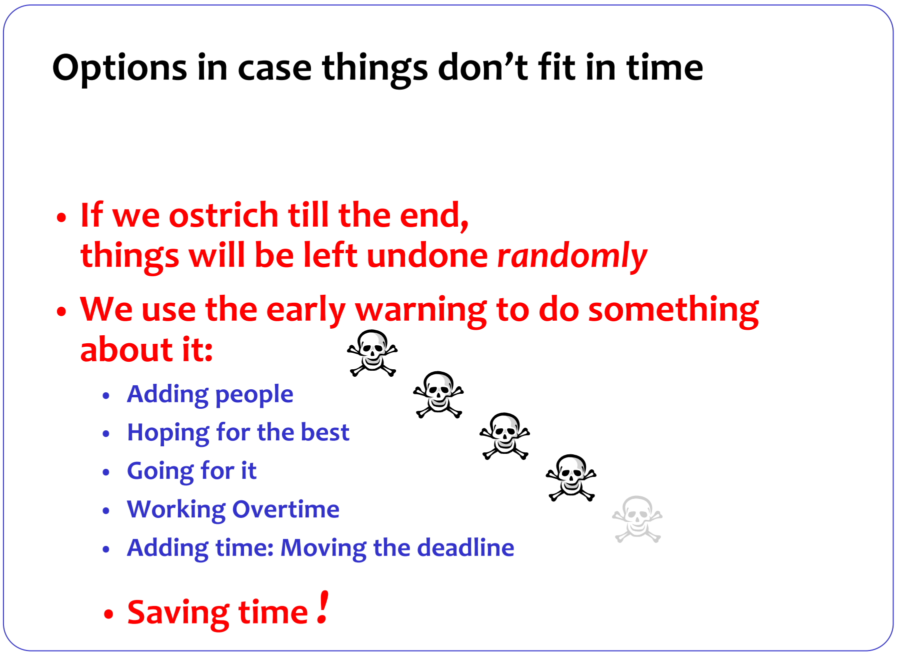
now

FatalDate

needed time > available time : not OK

needed time = available time : not OK

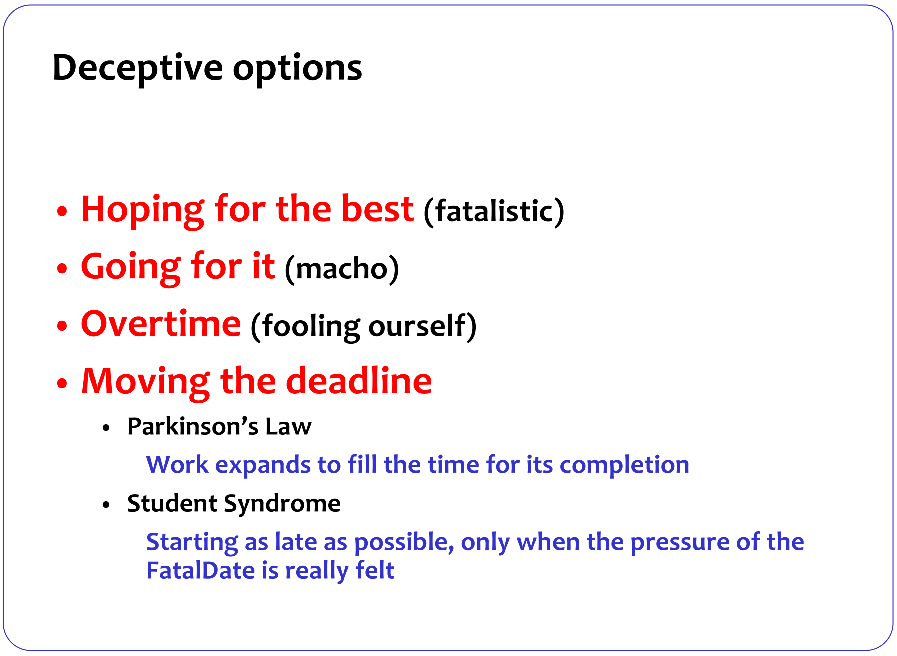
53



Options in case things don't fit in time

- If we ostrich till the end,
  things will be left undone *randomly*
- We use the early warning to do something about it:
  - Adding people
  - Hoping for the best
  - Going for it
  - Working Overtime
  - Adding time: Moving the deadline
  - Saving time **!**

54



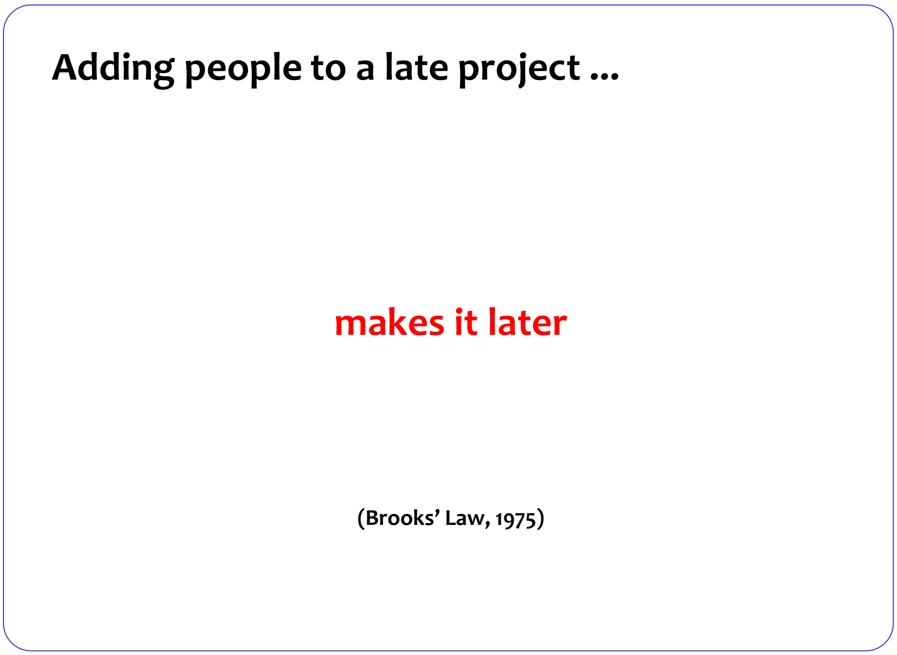
Deceptive options

- Hoping for the best (fatalistic)
- Going for it (macho)
- Overtime (fooling ourself)
- Moving the deadline
  - Parkinson’s Law

    Work expands to fill the time for its completion
  - Student Syndrome

    Starting as late as possible, only when the pressure of the FatalDate is really felt

55



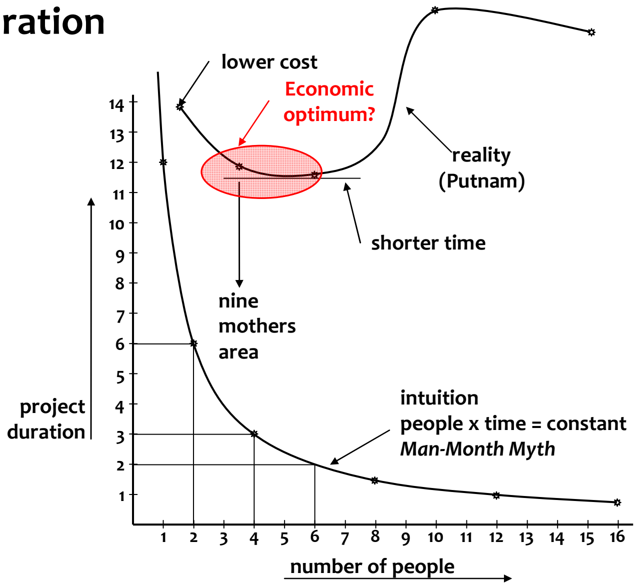
Adding people to a late project ...

makes it later

(Brooks’ Law, 1975)

56

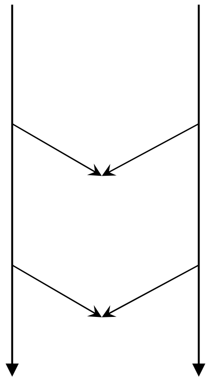
Project-duration



57

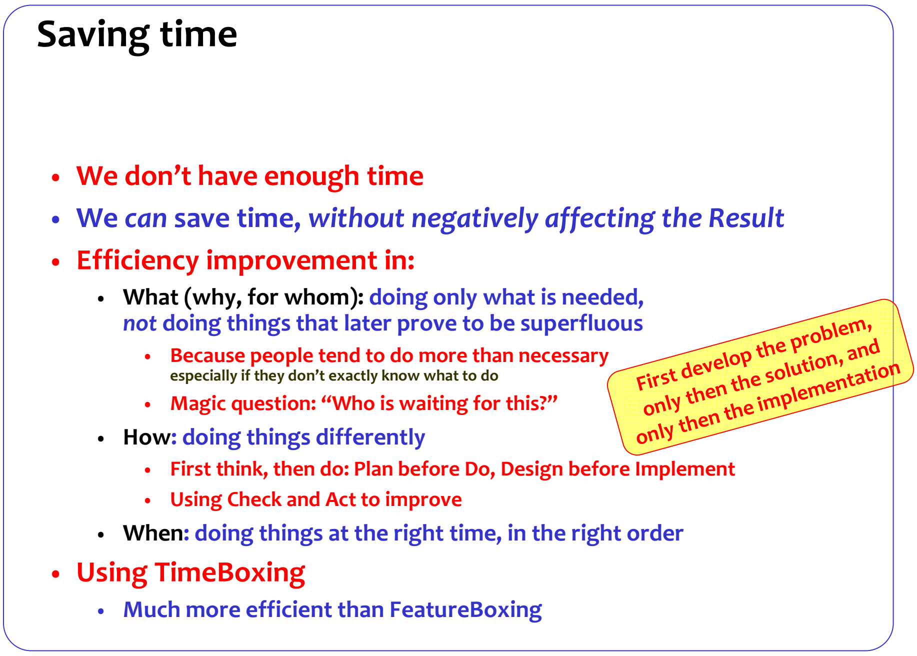
Solution ?

- Small parallel projects
- Regularly synchronizing



58

Boekjes:
www.malotaux.nl/nrm/pdf/MxEvo.pdf
www.malotaux.nl/nrm/pdf/EvoTesting.pdf
www.malotaux.nl/nrm/pdf/TimeLine.pdf
www.malotaux.nl/nrm/pdf/Booklet2.pdf
www.malotaux.nl/nrm/pdf/EvoRisk.pdf

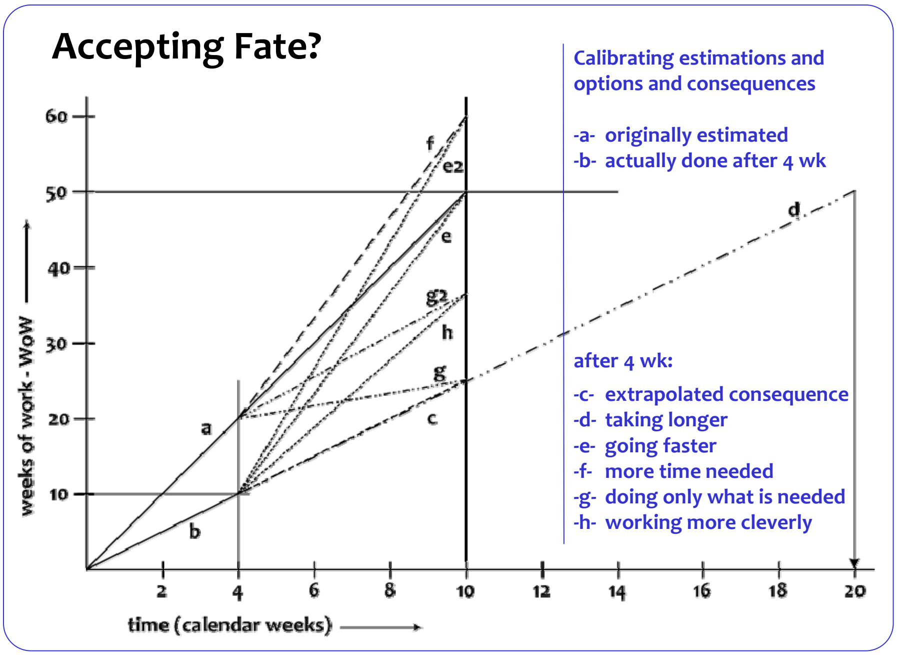


Saving time

- We don't have enough time
- We *can* save time, *without negatively affecting the Result*
- Efficiency improvement in:
  - What (why, for whom): doing only what is needed, *not* doing things that later prove to be superfluous
    - Because people tend to do more than necessary
      especially if they don't exactly know what to do
    - Magic question: "Who is waiting for this?"
  - How: doing things differently
    - First think, then do: Plan before Do, Design before Implement
    - Using Check and Act to improve
  - When: doing things at the right time, in the right order
- Using TimeBoxing
  - Much more efficient than FeatureBoxing

First develop the problem,
only then the solution, and
only then the implementation

59



Accepting Fate?

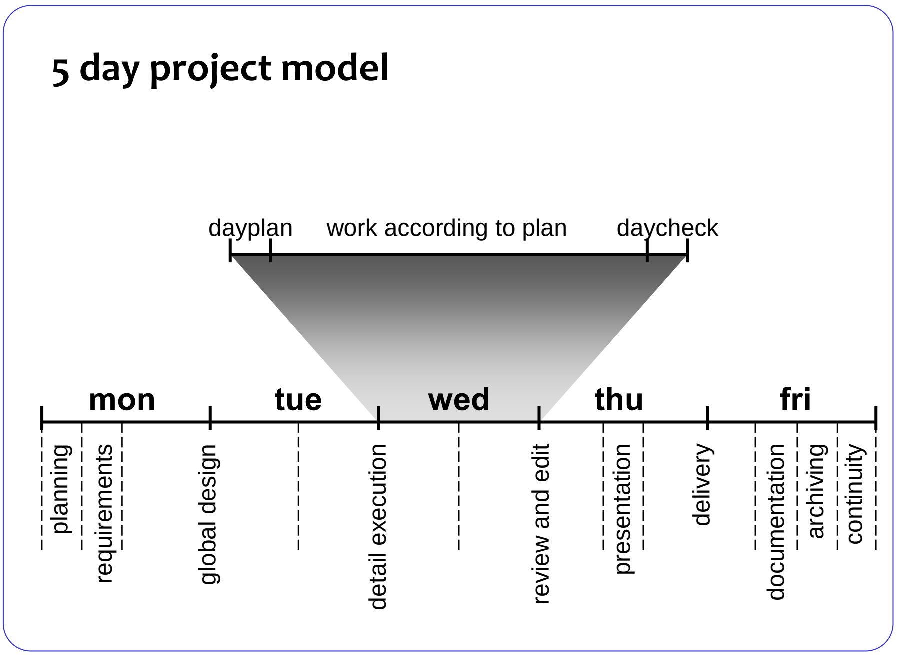
Calibrating estimations and options and consequences

-a- originally estimated
-b- actually done after 4 wk

after 4 wk:

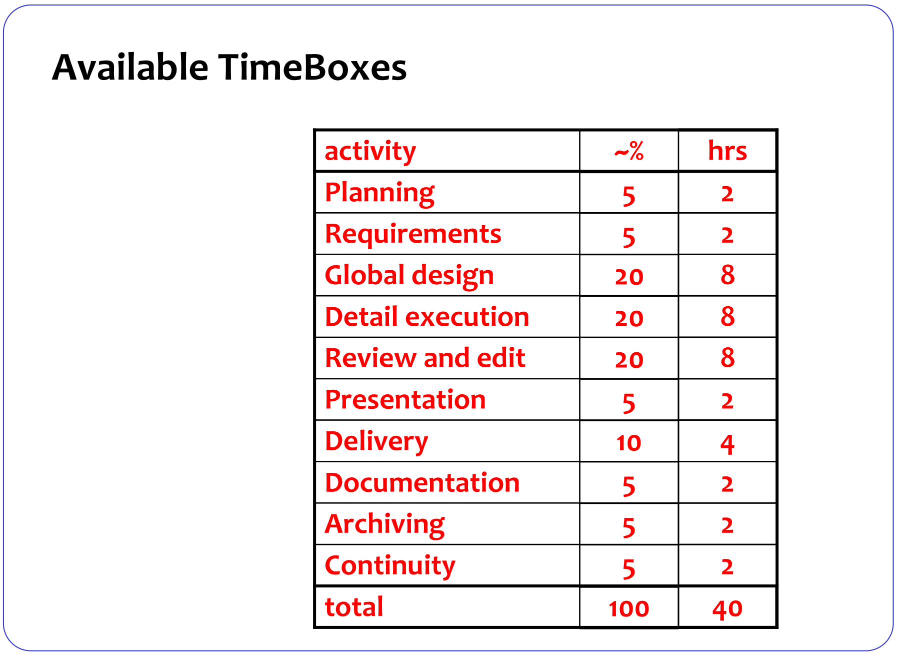
-c- extrapolated consequence
-d- taking longer
-e- going faster
-f- more time needed
-g- doing only what is needed
-h- working more cleverly

60



5 day project model

61

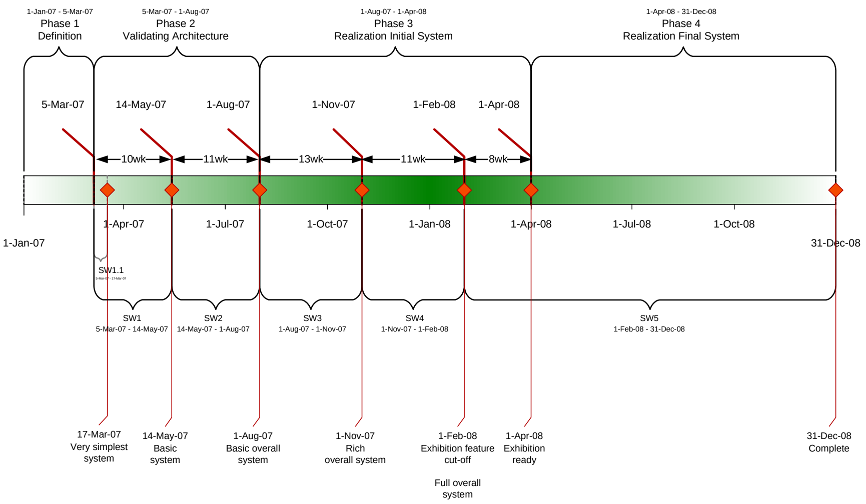


Available TimeBoxes

activity	~%	hrs
Planning	5	2
Requirements	5	2
Global design	20	8
Detail execution	20	8
Review and edit	20	8
Presentation	5	2
Delivery	10	4
Documentation	5	2
Archiving	5	2
Continuity	5	2
total	100	40

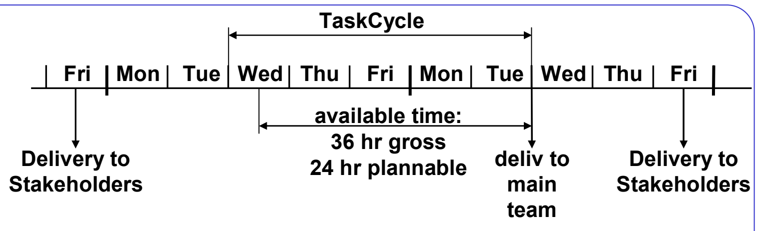
62

TimeLine example



63

Designing a Delivery



Serge (ProjLead)	
MbWA	3
Planning nxt wk	3
Work for deliv	4
-	6
-	2
-	1
-	5
Total	24

Gregory	
Draft design	6
Finish design	6
Work for deliv	3
-	1
-	2
-	2
-	3
-	5
-	6
XMLa	4
XMLb	4
Total	42

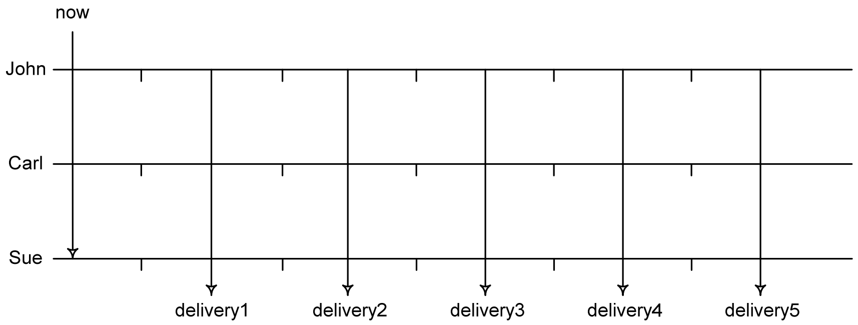
Gregory (later)	
→ Draft design	0
→ Finish design	0
...	
Repair deliv	0
...	

Zero Defects Attitude

Jerome	
→ XMLa	3
→ XMLb	3
...	

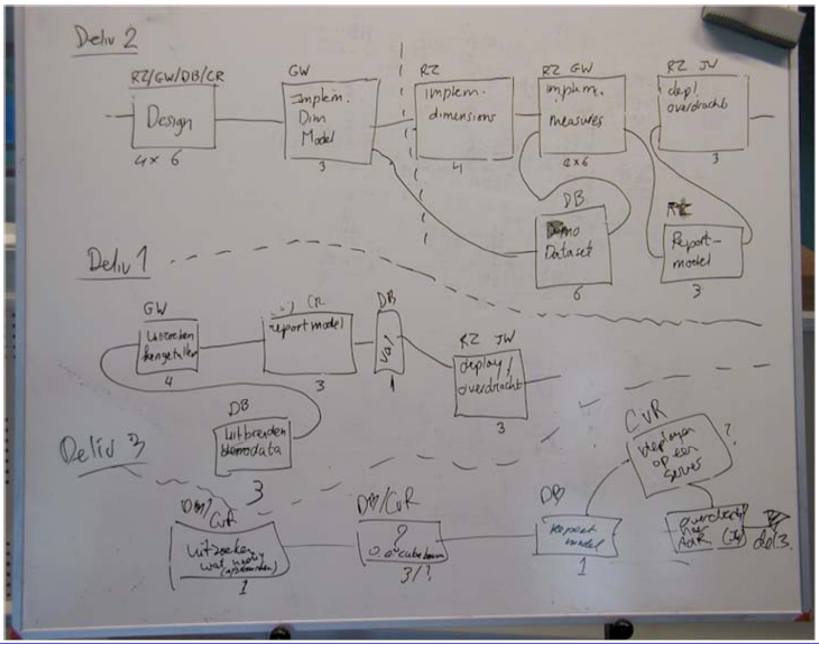
64

Making individual TimeLines



now
John
Carl
Sue
delivery1
delivery2
delivery3
delivery4
delivery5

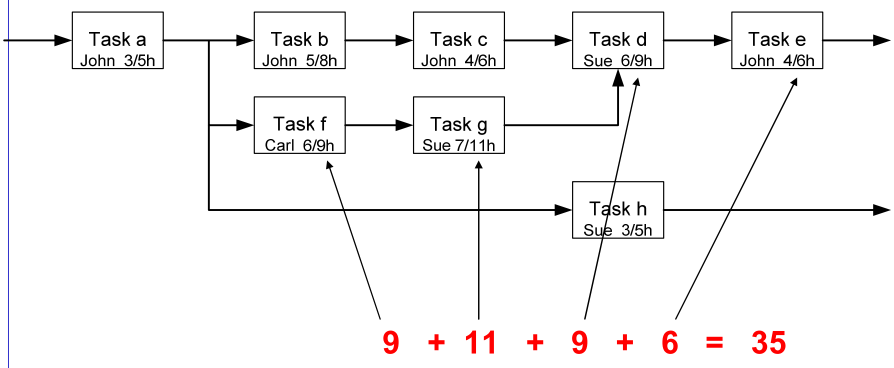
65



66

PERT (Project Evaluation Review Technique)

used for *Designing* a Delivery



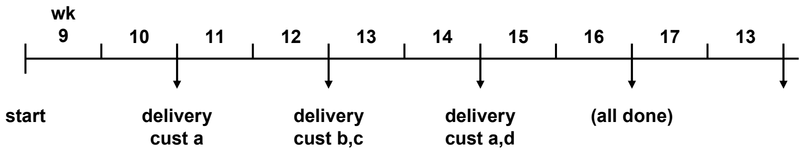
67

We have a QA problem !

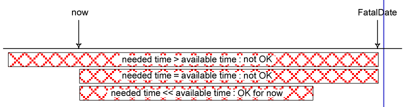
- Large stockpile of modules to be tested
- Estimate: will cost half year of testing
- You shall do Full Regression Tests
- Full Regression Tests take about 15 days each
- QA is bottleneck
- Can we do something about this?

68

TimeLine

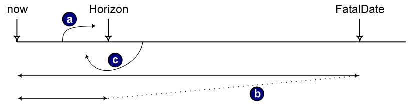


TimeLine exercise for *your* Project, step 1


now
FatalDate
needed time > available time : not OK
needed time = available time : not OK
needed time << available time : OK for now

- What is the FatalDate, how many weeks left (= m)
- What is the expected result (←Business Case / Reqs)
- What do you have to do to achieve that result
- Cut this into chunks and make a list of chunks of work
- Estimate the chunks (in weeks)
- Calculate number of weeks (= n)
- Compensate for estimated incompleteness of the list (new n)
- How many people are available for the work (= R)
  1. n / R > m: more time needed than available
  2. n / R = m: still probably not enough time
  3. n / R << m: probably possible to succeed on time
- Case 1 and 2: work out the consequence at this level
- Case 3: continue to the next level

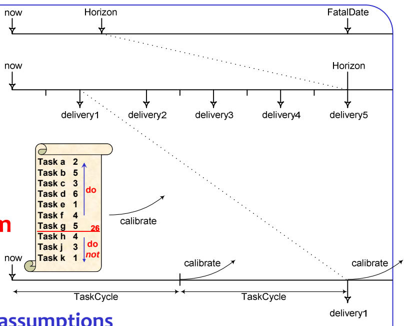
TimeLine exercise for *your* Project, step 2



- Choose Horizon, with clear intermediate Result
- Repeat steps from step 1, with:
  - FatalDate = Horizon
  - Amount of work proportionally to total work
  - Work may be estimated in some more detail now
- Now we have a pile of work to be done in these 10 weeks

71

TimeLine exercise for *your* Project, step 3



- Divide the work for 10 weeks in optimum order, defining Deliveries of 2 weeks
- Deliveries:
  - For feedback, checking requirements and assumptions
  - Therefore, must be delivered to *eagerly waiting* Stakeholders
  - If needed to make them eagerly waiting, give them juicy bits
- Make a rather detailed description of the first one or two Deliveries
- Check the feasibility of completing deliveries in two weeks each
- Determine Tasks for the first week
- Estimate the Tasks
- Calibrate the feasibility of the TimeLine with your first weeks estimations
- Now you have the Tasks for the first week defined

72

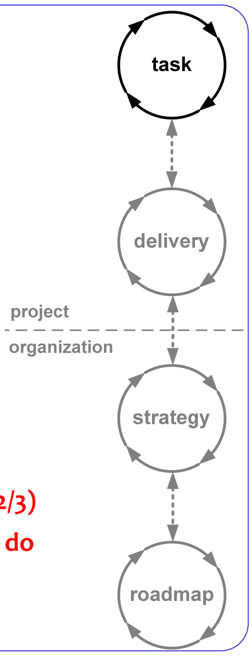
Can you make your own Timeline ?

- If yes, do so
- If no, why not?

Homework:

- Can you make a better TimeLine?

Cycles in Evo: Weekly TaskCycle



- Are we *doing* the *right things*, in the *right order*, to the right *level of detail for now*
- Optimizing estimation, planning and tracking abilities to better predict the future
- Select highest priority tasks, never do any lower priority tasks, never do undefined tasks
- There are only about 26 plannable hours in a week (2/3)
- In the remaining time: do whatever else you have to do
- Tasks are always done, 100% done

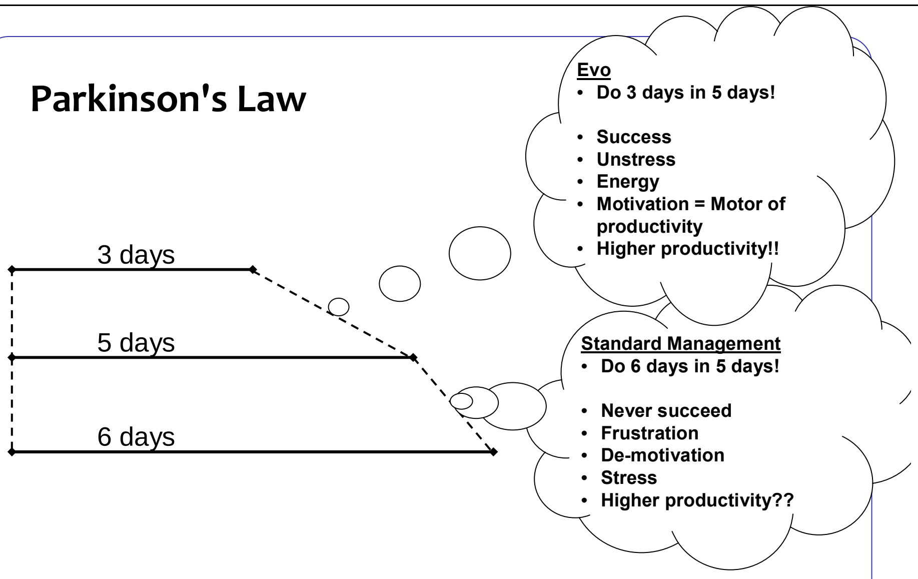
Effort and Lead Time

- Days estimation → lead time (calendar time)
- Hours estimation → effort

- Effort variations and lead time variations have different causes
- So, treat them differently and keep them separate
  - Effort: complexity
  - Lead Time: time-management
    - (effort / lead-time ratio)

75

Parkinson's Law



Evo
- Do 3 days in 5 days!

- Success
- Unstress
- Energy
- Motivation = Motor of productivity
- Higher productivity!!

Standard Management
- Do 6 days in 5 days!

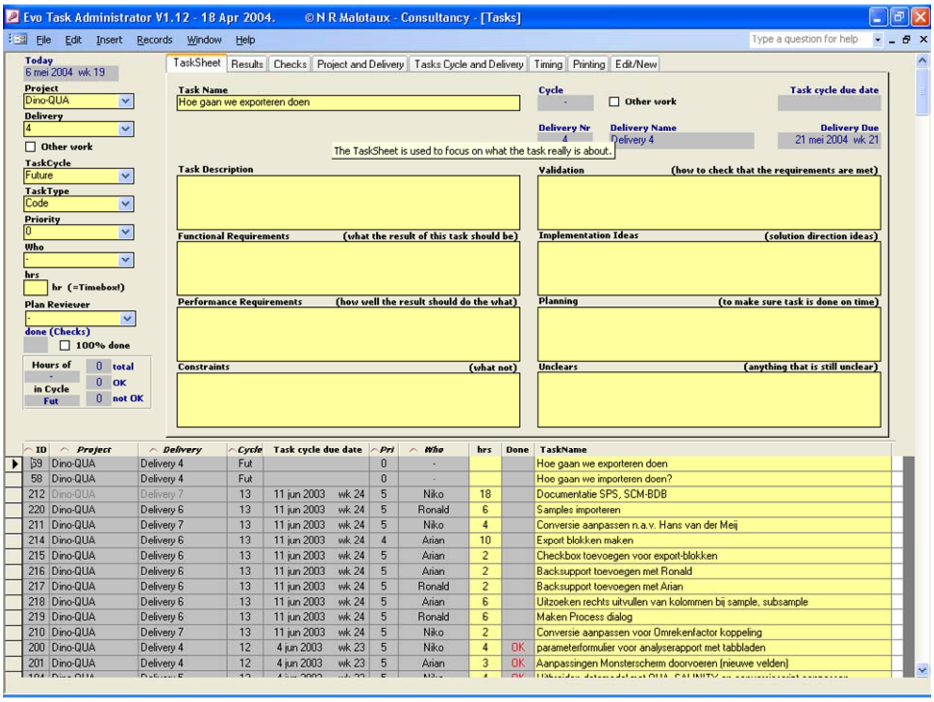
- Never succeed
- Frustration
- De-motivation
- Stress
- Higher productivity??

"Work expands to fill the time available"

76

What to plan and what not to plan

- We plan tasks that don’t get done unless planned
- We do not plan tasks that don’t have to be planned to get done. Such planning costs more than it saves
- Account for these tasks as “unplannable tasks”
- Default we allocate 2/3 for plannable tasks and 1/3 for unplannable tasks
- We may include tasks in the planning to show that the hours for these tasks are not available for other work
- Plan *all* plannable hours



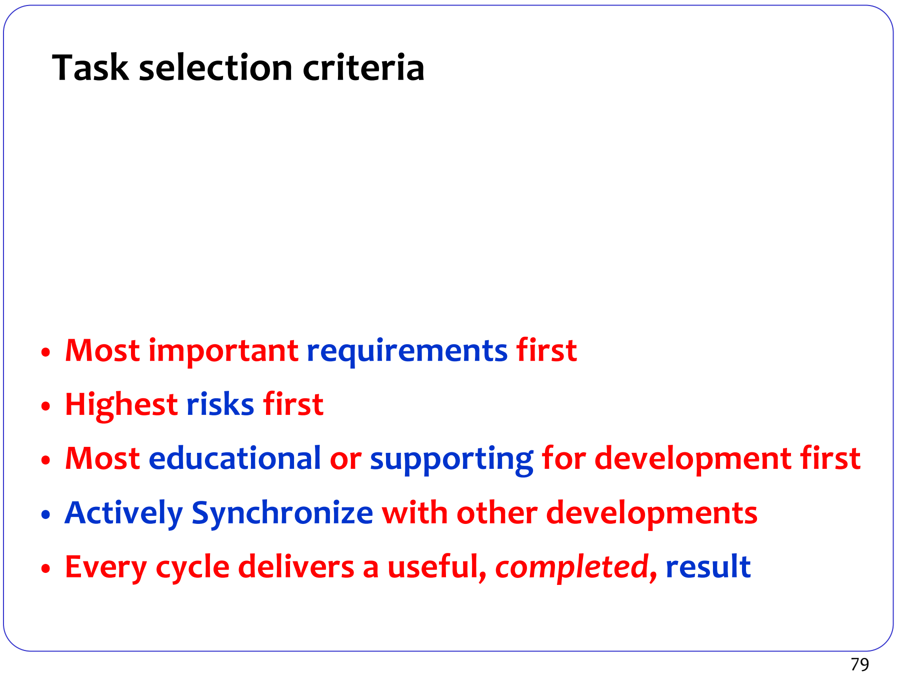
Today: 6 mei 2004 wk 19
Project: Dino-QUA
Delivery: 4
☐ Other work
TaskCycle: Future
TaskType: Code
Priority: 0
Who: -
hrs: hr (=Timebox!)
Plan Reviewer: -
done (Checks)
☐ 100% done
Hours of - in Cycle Fut: 0 total, 0 OK, 0 not OK

Task Name: Hoe gaan we exporteren doen
Cycle: -
☐ Other work
Task cycle due date
Delivery Nr: 4
Delivery Name: Delivery 4
Delivery Due: 21 mei 2004 wk 21

The TaskSheet is used to focus on what the task really is about.

Task Description
Validation (how to check that the requirements are met)
Functional Requirements (what the result of this task should be)
Implementation Ideas (solution direction ideas)
Performance Requirements (how well the result should do the what)
Planning (to make sure task is done on time)
Constraints (what not)
Unclears (anything that is still unclear)

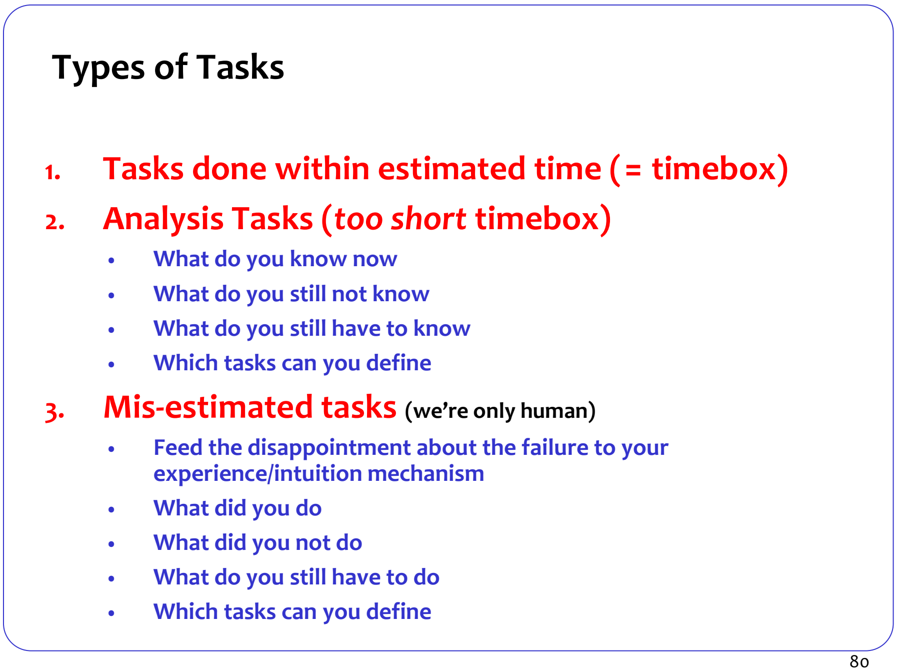
ID	Project	Delivery	Cycle	Task cycle due date	Pri	Who	hrs	Done	TaskName
59	Dino-QUA	Delivery 4	Fut		0	-			Hoe gaan we exporteren doen
58	Dino-QUA	Delivery 4	Fut		0	-			Hoe gaan we importeren doen?
212	Dino-QUA	Delivery 7	13	11 jun 2003 wk 24	5	Niko	18		Documentatie SPS, SCM-BDB
220	Dino-QUA	Delivery 6	13	11 jun 2003 wk 24	5	Ronald	6		Samples importeren
211	Dino-QUA	Delivery 7	13	11 jun 2003 wk 24	5	Niko	4		Conversie aanpassen n.a.v. Hans van der Meij
214	Dino-QUA	Delivery 6	13	11 jun 2003 wk 24	4	Arian	10		Export blokken maken
215	Dino-QUA	Delivery 6	13	11 jun 2003 wk 24	5	Arian	2		Checkbox toevoegen voor export-blokken
216	Dino-QUA	Delivery 6	13	11 jun 2003 wk 24	5	Arian	2		Backsupport toevoegen met Ronald
217	Dino-QUA	Delivery 6	13	11 jun 2003 wk 24	5	Ronald	2		Backsupport toevoegen met Arian
218	Dino-QUA	Delivery 6	13	11 jun 2003 wk 24	5	Arian	6		Uitzoeken rechts uitvullen van kolommen bij sample, subsample
219	Dino-QUA	Delivery 6	13	11 jun 2003 wk 24	5	Ronald	6		Maken Process dialog
210	Dino-QUA	Delivery 7	13	11 jun 2003 wk 24	5	Niko	2		Conversie aanpassen voor Omrekenfactor koppeling
200	Dino-QUA	Delivery 4	12	4 jun 2003 wk 23	5	Niko	4	OK	parameterformulier voor analyserapport met tabbladen
201	Dino-QUA	Delivery 4	12	4 jun 2003 wk 23	5	Arian	3	OK	Aanpassingen Monsterscherm doorvoeren (nieuwe velden)



Task selection criteria

- Most important requirements first
- Highest risks first
- Most educational or supporting for development first
- Actively Synchronize with other developments
- Every cycle delivers a useful, *completed*, result

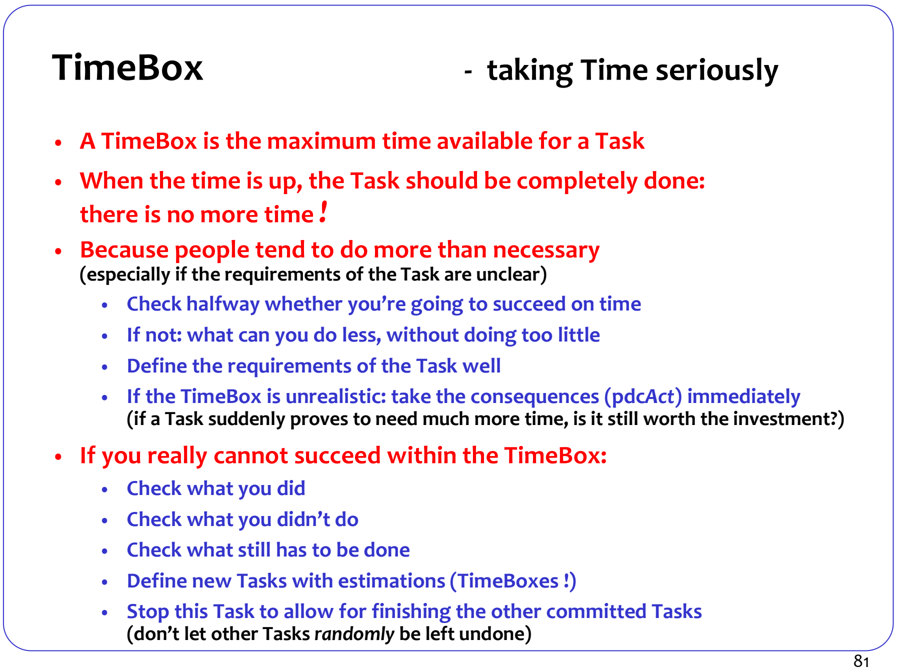
79



Types of Tasks

1. Tasks done within estimated time (= timebox)
2. Analysis Tasks (*too short* timebox)
   - What do you know now
   - What do you still not know
   - What do you still have to know
   - Which tasks can you define
3. Mis-estimated tasks (we're only human)
   - Feed the disappointment about the failure to your experience/intuition mechanism
   - What did you do
   - What did you not do
   - What do you still have to do
   - Which tasks can you define

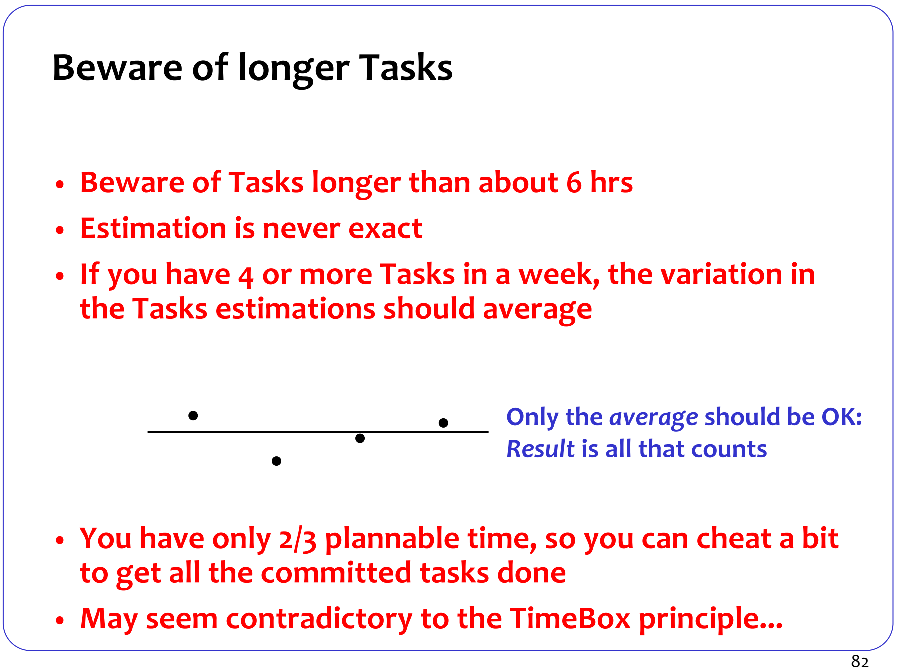
80



TimeBox - taking Time seriously

- A TimeBox is the maximum time available for a Task
- When the time is up, the Task should be completely done: there is no more time!
- Because people tend to do more than necessary (especially if the requirements of the Task are unclear)
  - Check halfway whether you're going to succeed on time
  - If not: what can you do less, without doing too little
  - Define the requirements of the Task well
  - If the TimeBox is unrealistic: take the consequences (pdc*Act*) immediately (if a Task suddenly proves to need much more time, is it still worth the investment?)
- If you really cannot succeed within the TimeBox:
  - Check what you did
  - Check what you didn't do
  - Check what still has to be done
  - Define new Tasks with estimations (TimeBoxes !)
  - Stop this Task to allow for finishing the other committed Tasks (don't let other Tasks *randomly* be left undone)

81



Beware of longer Tasks

- Beware of Tasks longer than about 6 hrs
- Estimation is never exact
- If you have 4 or more Tasks in a week, the variation in the Tasks estimations should average

Only the *average* should be OK:
Result is all that counts

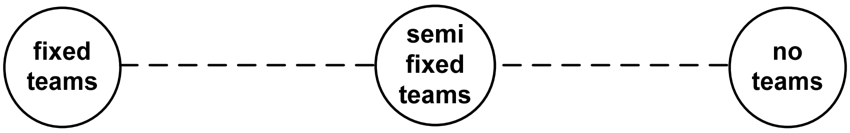
- You have only 2/3 plannable time, so you can cheat a bit to get all the committed tasks done
- May seem contradictory to the TimeBox principle...

82

We work on more projects

- Define how many hours available for this project
- Deliver these hours

- Vision:



fixed teams

semi fixed teams

no teams

83

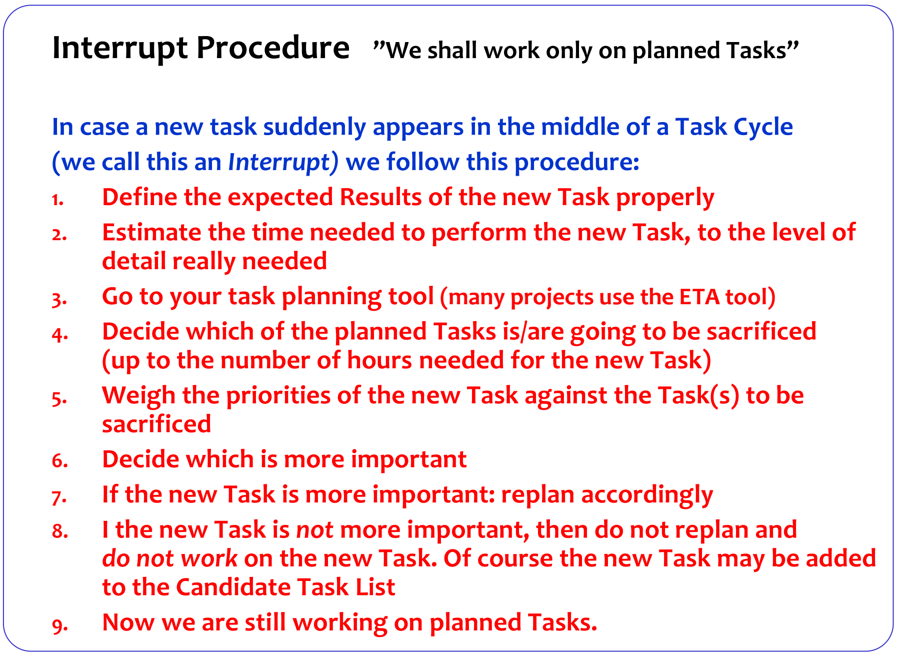
Interrupts

- Boss comes in: “Can you paint my fence?”
- What do you do?

DILBERT, TAKE CARE OF THIS. IT'S OUR TOP PRIORITY.
www.dilbert.com
scottadams@aol.com
SO WHAT IS IT?
I DON'T KNOW... I JUST DIDN'T LIKE IT ON MY DESK.

- In case of interrupt, use interrupt procedure

84

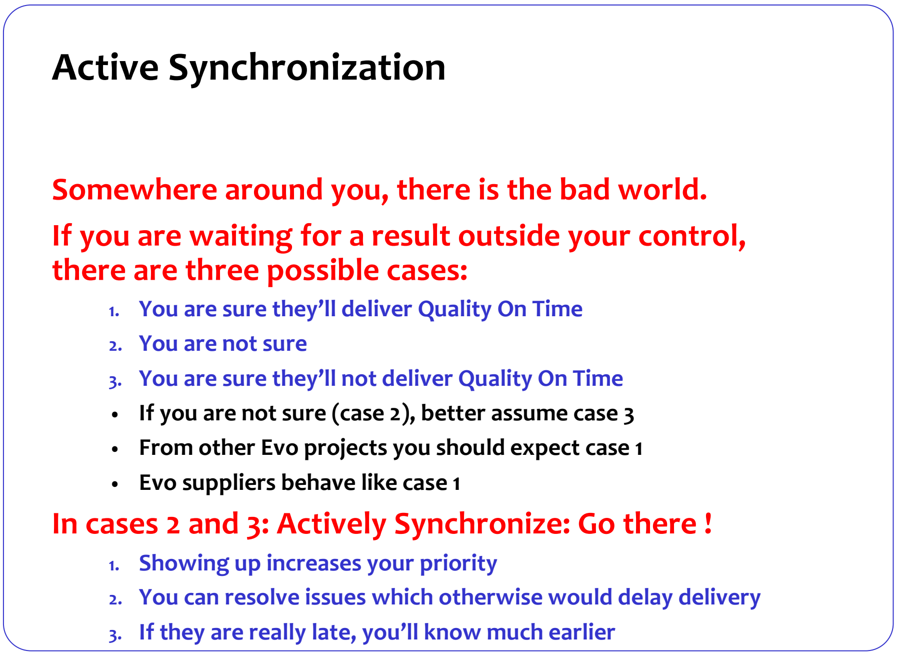


Interrupt Procedure ”We shall work only on planned Tasks”

In case a new task suddenly appears in the middle of a Task Cycle (we call this an *Interrupt)* we follow this procedure:

1. Define the expected Results of the new Task properly
2. Estimate the time needed to perform the new Task, to the level of detail really needed
3. Go to your task planning tool (many projects use the ETA tool)
4. Decide which of the planned Tasks is/are going to be sacrificed (up to the number of hours needed for the new Task)
5. Weigh the priorities of the new Task against the Task(s) to be sacrificed
6. Decide which is more important
7. If the new Task is more important: replan accordingly
8. I the new Task is *not* more important, then do not replan and *do not work* on the new Task. Of course the new Task may be added to the Candidate Task List
9. Now we are still working on planned Tasks.

85



Active Synchronization

Somewhere around you, there is the bad world.

If you are waiting for a result outside your control, there are three possible cases:

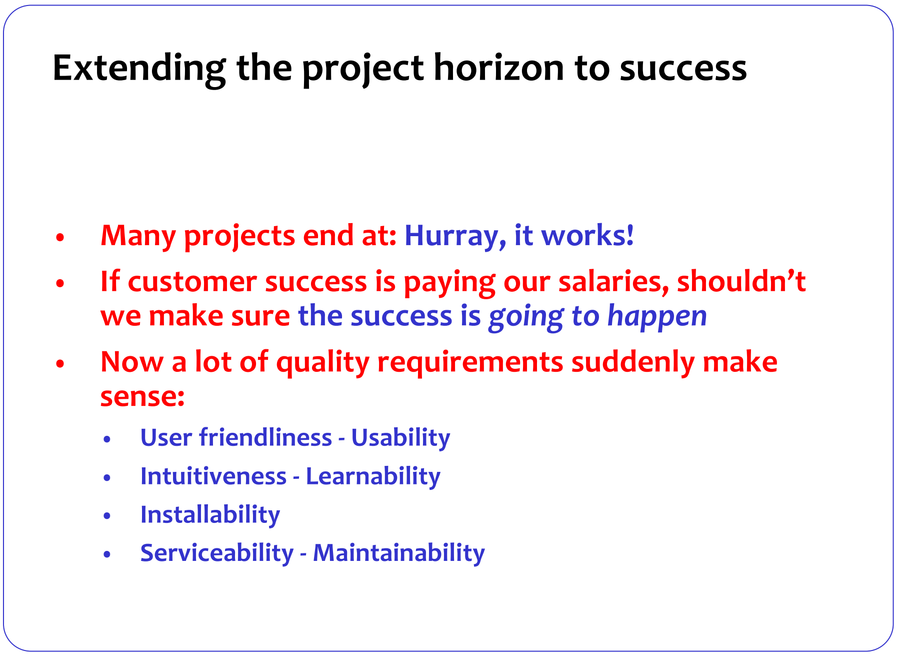
1. You are sure they’ll deliver Quality On Time
2. You are not sure
3. You are sure they’ll not deliver Quality On Time

- If you are not sure (case 2), better assume case 3
- From other Evo projects you should expect case 1
- Evo suppliers behave like case 1

In cases 2 and 3: Actively Synchronize: Go there !

1. Showing up increases your priority
2. You can resolve issues which otherwise would delay delivery
3. If they are really late, you’ll know much earlier

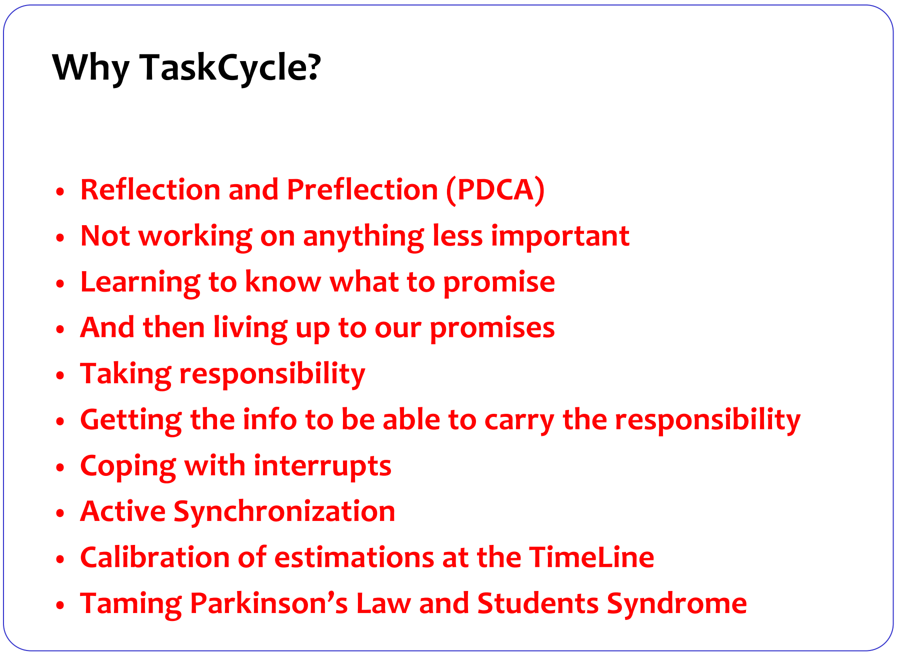
86



Extending the project horizon to success

- Many projects end at: Hurray, it works!
- If customer success is paying our salaries, shouldn't we make sure the success is *going to happen*
- Now a lot of quality requirements suddenly make sense:
    - User friendliness - Usability
    - Intuitiveness - Learnability
    - Installability
    - Serviceability - Maintainability

87



Why TaskCycle?

- Reflection and Preflection (PDCA)
- Not working on anything less important
- Learning to know what to promise
- And then living up to our promises
- Taking responsibility
- Getting the info to be able to carry the responsibility
- Coping with interrupts
- Active Synchronization
- Calibration of estimations at the TimeLine
- Taming Parkinson's Law and Students Syndrome

88

Why would the product need Evo ?

- We don't know the real requirements
- They don't know the real requirements
- Together we have to find out (stop playing macho!)
- What the customer wants he cannot afford
- Is what the customer wants what he needs?
- People tend to do more than necessary
  especially if they don't know exactly what to do

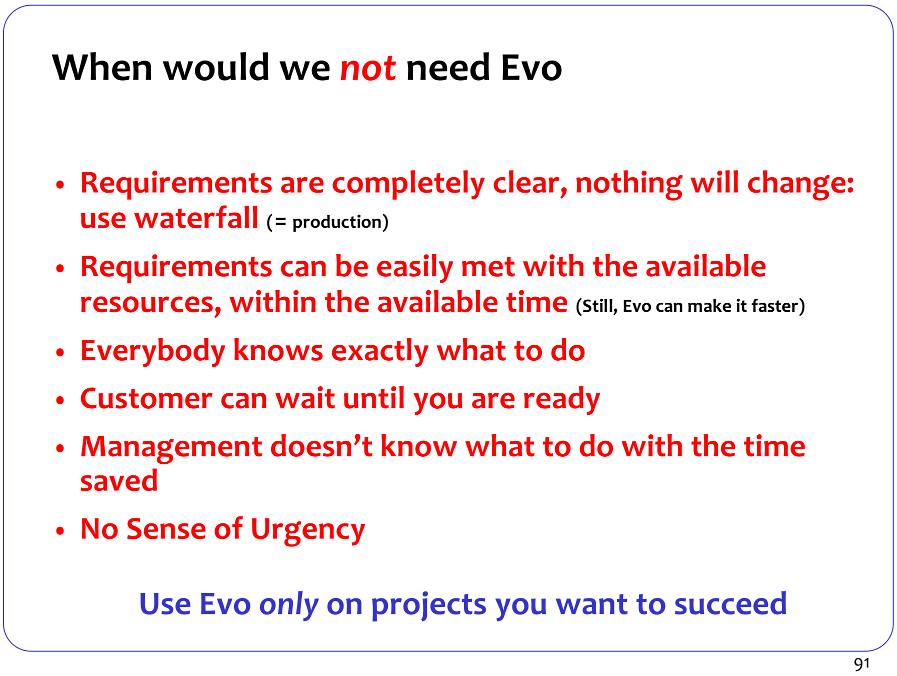
If time, money, resources are limited,
we should not overrun the budgets

89

Why would the project need Evo ?

- Are we effective? (producing Results)
- Are we efficient? (optimally using the available time)
- Are we actively learning from our mistakes? (PDCA)
- How do we estimate, plan and track progress?
- How do we handle interruptions?
- Did we learn from feedback per project (project evaluation)?

90

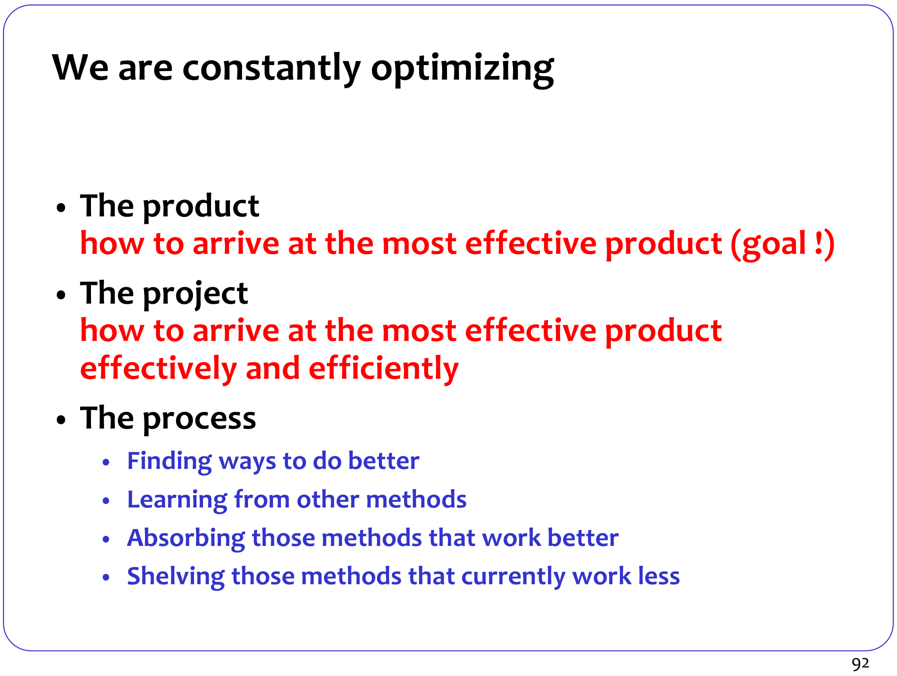


When would we *not* need Evo

- Requirements are completely clear, nothing will change: use waterfall (= production)
- Requirements can be easily met with the available resources, within the available time (Still, Evo can make it faster)
- Everybody knows exactly what to do
- Customer can wait until you are ready
- Management doesn't know what to do with the time saved
- No Sense of Urgency

Use Evo *only* on projects you want to succeed

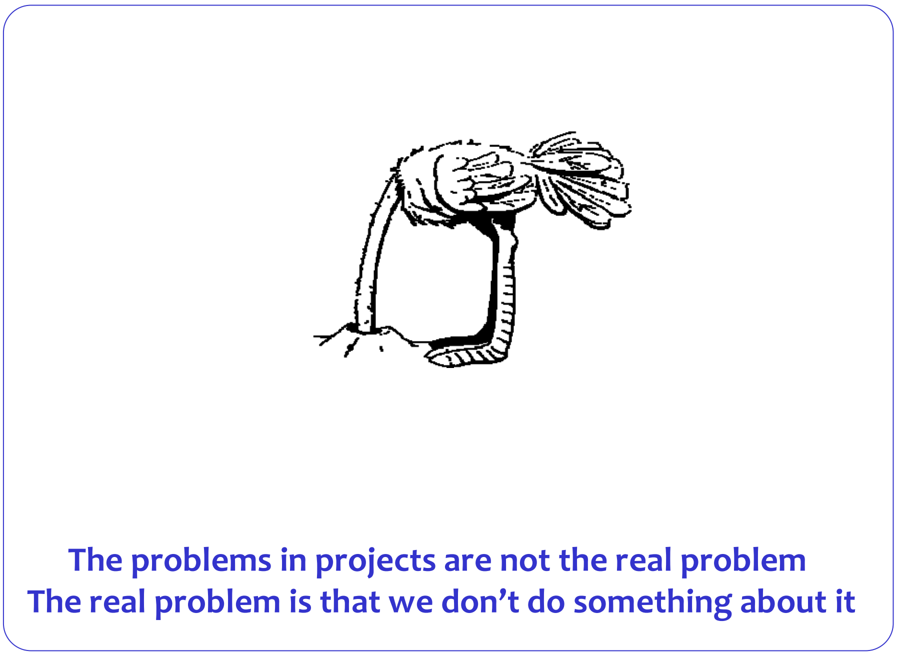
91



We are constantly optimizing

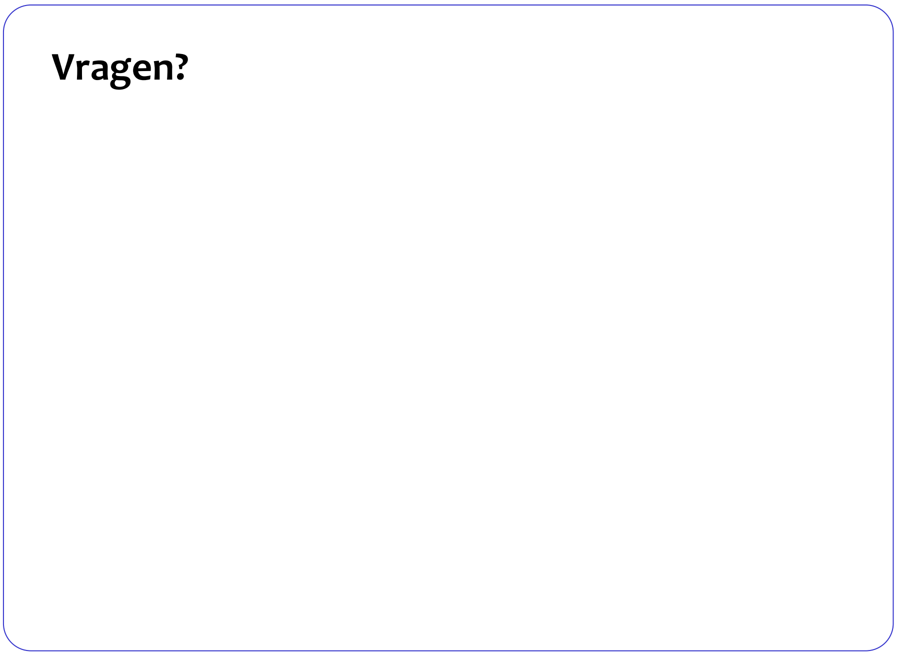
- The product
  how to arrive at the most effective product (goal !)
- The project
  how to arrive at the most effective product effectively and efficiently
- The process
  - Finding ways to do better
  - Learning from other methods
  - Absorbing those methods that work better
  - Shelving those methods that currently work less

92



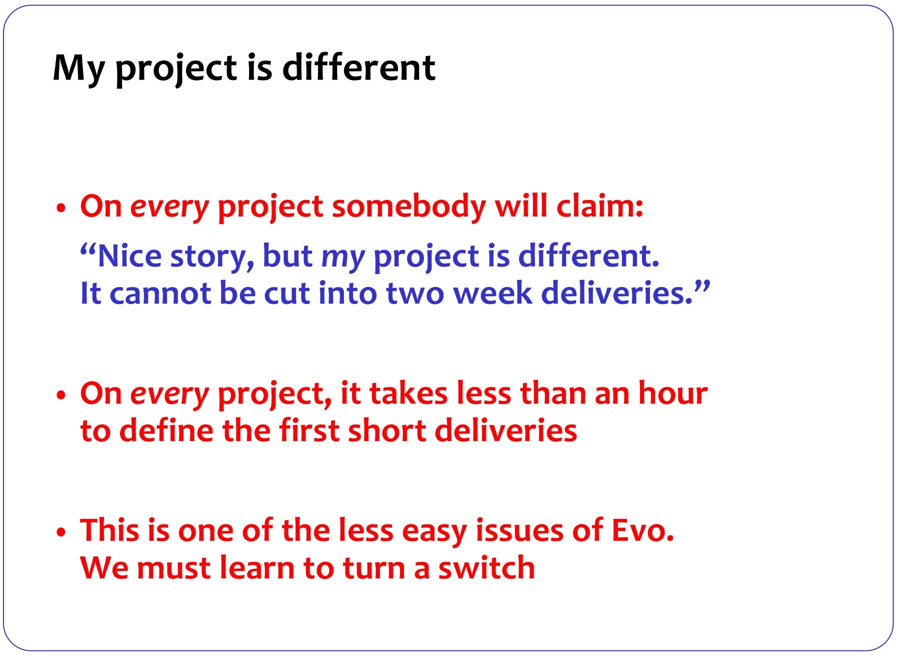
The problems in projects are not the real problem
The real problem is that we don't do something about it

93



Vragen?

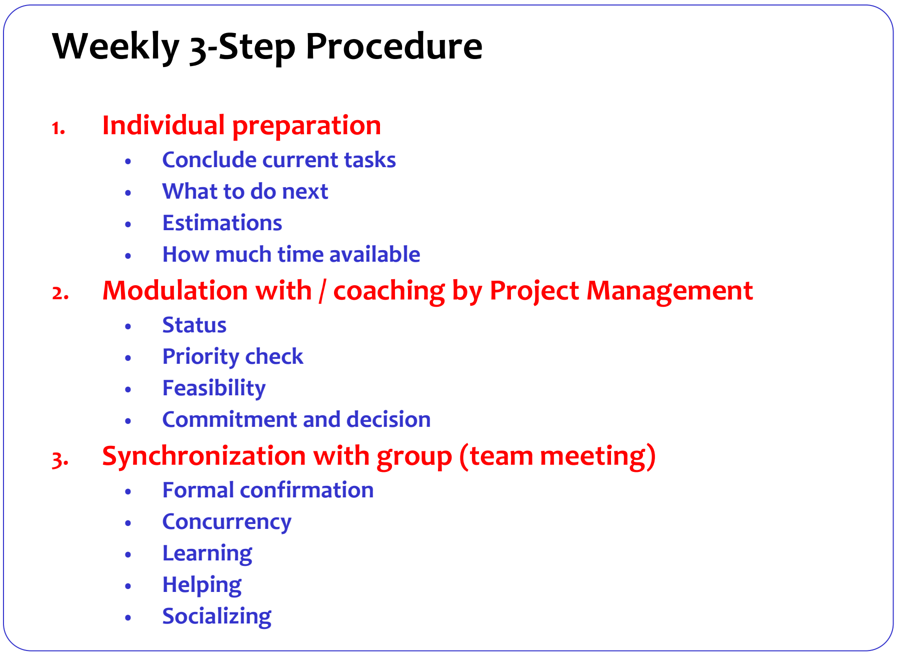
94



My project is different

- On *every* project somebody will claim:
  “Nice story, but *my* project is different.
  It cannot be cut into two week deliveries.”
- On *every* project, it takes less than an hour to define the first short deliveries
- This is one of the less easy issues of Evo.
  We must learn to turn a switch

95



Weekly 3-Step Procedure

1. Individual preparation
   - Conclude current tasks
   - What to do next
   - Estimations
   - How much time available
2. Modulation with / coaching by Project Management
   - Status
   - Priority check
   - Feasibility
   - Commitment and decision
3. Synchronization with group (team meeting)
   - Formal confirmation
   - Concurrency
   - Learning
   - Helping
   - Socializing

96

Huiswerk

- Bepaal de Business Case van je eigen project
- Bepaal de top-3 Requirements (met Stakeholders)
- Beschrijf een van die Requirements met Planguage
- Analyseer de resultaten van je weekplanning (Check)
- Bedenk hoe je nog beter kan werken (Act)
- Bepaal je nieuwe weekplanning op basis van
  - Je TimeLine
  - Wat je geleerd hebt van je vorige weekcyclus resultaten
- Kun je je TimeLine al calibreren met wat er in de week gebeurt?
- Neem een (niet-confidentiëel) document mee voor review
  (volgende keer: o.m. testen, reviews, inspections)
- Iets gemist? → stuur een email (niels@malotaux.nl)

97

Links

- www.gilb.com
  Tom Gilb's website: Evo guru
- www.malotaux.nl
  Niels' activities: Evo evangelist
- www.malotaux.nl/nrm/Evo
  Evo pages
- www.malotaux.nl/nrm/Insp
  Inspection pages
- www.malotaux.nl/nrm/pdf/MxEvo.pdf
  Evolutionary Project Management Methods
  (issues and first - 2001 - experience)
- www.malotaux.nl/nrm/pdf/Booklet2.pdf
  How Quality is Assured by Evolutionary Methods
  (more recent - 2004 - practical implementation experience)
- www.malotaux.nl/nrm/pdf/EvoTesting.pdf
  Optimizing the Contribution of Testing to Project Success (2005)
- www.malotaux.nl/nrm/pdf/EvoRisk.pdf
  Controlling Project Risk *by Design* (2006)
- www.malotaux.nl/nrm/pdf/TimeLine.pdf
  TimeLine: How to get and keep control over longer periods of time (2007)
- www.malotaux.nl/nrm/Evo/ETAF.htm
  Download the Evo Task Administrator (ETA) tool
  (expects MSAccess 2000~2003)

98

Can you afford not to use Evo?

Niels Malotaux

N R Malotaux
Consultancy

030-228 88 68 niels@malotaux.nl www.malotaux.nl/nrm

99

Evolutionary Project Management Methods

Dag 2

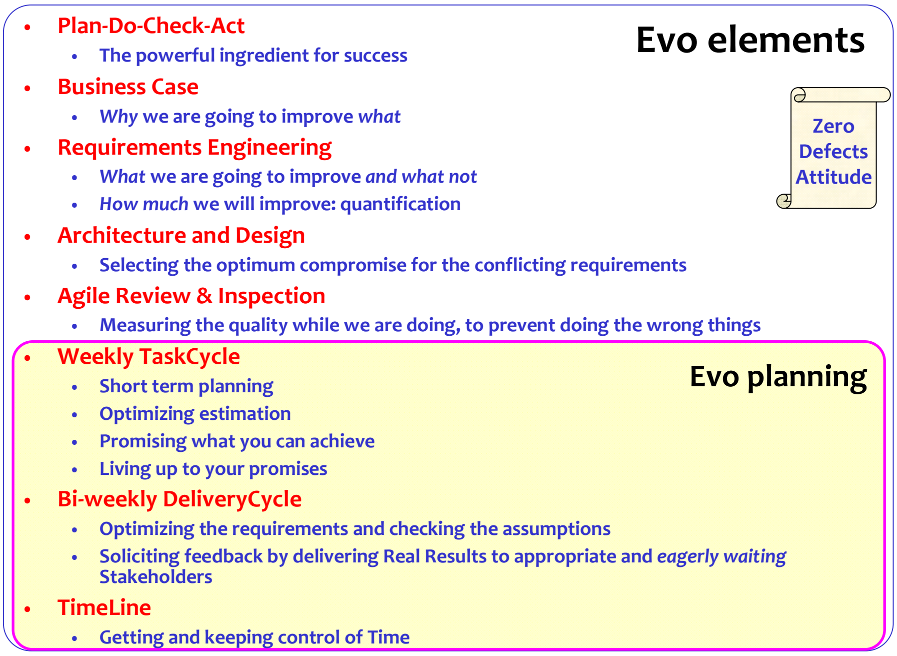
How to get the best results in the shortest time

Niels Malotaux

N R Malotaux
Consultancy

+31-30-228 88 68 niels@malotaux.nl www.malotaux.nl

100



Evo elements

- 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
- Agile Review & Inspection
  - Measuring the quality while we are doing, to prevent doing the wrong things

Zero Defects Attitude

Evo planning

- Weekly TaskCycle
  - Short term planning
  - Optimizing estimation
  - Promising what you can achieve
  - Living up to your promises
- Bi-weekly DeliveryCycle
  - Optimizing the requirements and checking the assumptions
  - Soliciting feedback by delivering Real Results to appropriate and *eagerly waiting* Stakeholders
- TimeLine
  - Getting and keeping control of Time

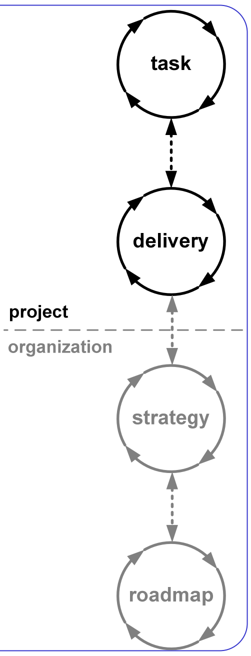
101

Dag2

Boekjes:
www.malotaux.nl/nrm/pdf/MxEvo.pdf
www.malotaux.nl/nrm/pdf/EvoTesting.pdf
www.malotaux.nl/nrm/pdf/TimeLine.pdf
www.malotaux.nl/nrm/pdf/Booklet2.pdf
www.malotaux.nl/nrm/pdf/EvoRisk.pdf

More cycles

- Horizon
- Intermediate Delivery
- Release
- Project
- Program
- Strategy
- Roadmap



102

Huiswerk

- Bepaal de Business Case van je eigen project
- Bepaal de top-3 Requirements (met Stakeholders)
- Beschrijf een van die Requirements met Planguage
- Analyseer de resultaten van je weekplanning (Check)
- Bedenk hoe je nog beter kan werken (Act)
- Bepaal je nieuwe weekplanning op basis van
  - Je TimeLine
  - Wat je geleerd hebt van je vorige weekcyclus resultaten
- Kun je je TimeLine al calibreren met wat er in de week gebeurt?
- Neem een (niet-confidentiëel) document mee voor review
  (volgende keer: o.m. testen, reviews, inspections)
- Iets gemist? → stuur een email (niels@malotaux.nl)

103

Week Planning

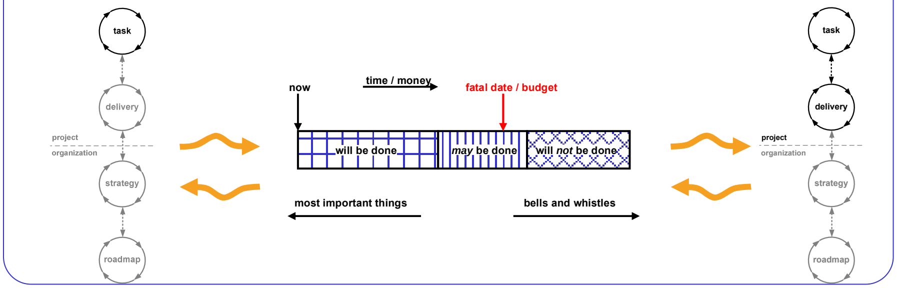
- Could you plan?
- Why not? (for some)
- Could you follow the plan?
- Why not? (for some)
- What can we learn → Check
- What are you going to do differently next time → Act
- What should you do next week ← Requirements
- Planning of next week ← Plan

Evolutionary start pattern

- Evo day
  - Explanation of the Evo approach
  - Organizing the work of the coming week
  - Goal: at the end of the day, people of the team know what they are going to work on and why
- Weekly Evo day
  - Execution of the 3-step procedure (slide 42)

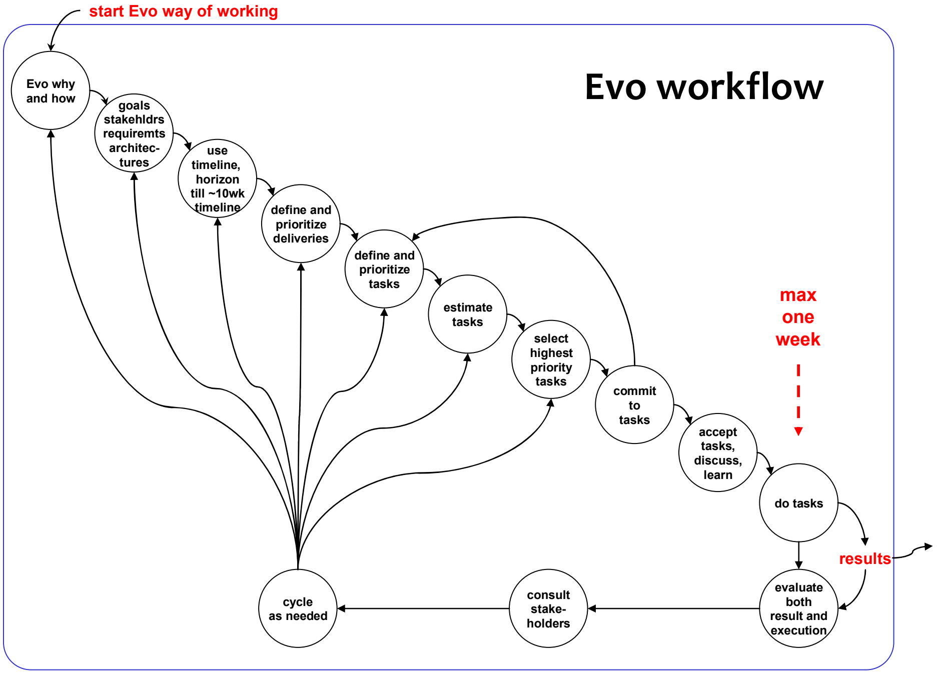
Evolutionary introduction pattern

1. Introducing *Tasks* → Short term view
   How to organize the work
2. Introducing *TimeLine* → Longer term view
   The *design* of the project
3. Introducing *Deliveries* → Connecting long and short
   Focusing on Results



106

Evo workflow


start Evo way of working

Evo why and how

goals stakehldrs requiremts architec-tures

use timeline, horizon till ~10wk timeline

define and prioritize deliveries

define and prioritize tasks

estimate tasks

select highest priority tasks

commit to tasks

accept tasks, discuss, learn

max one week

do tasks

results

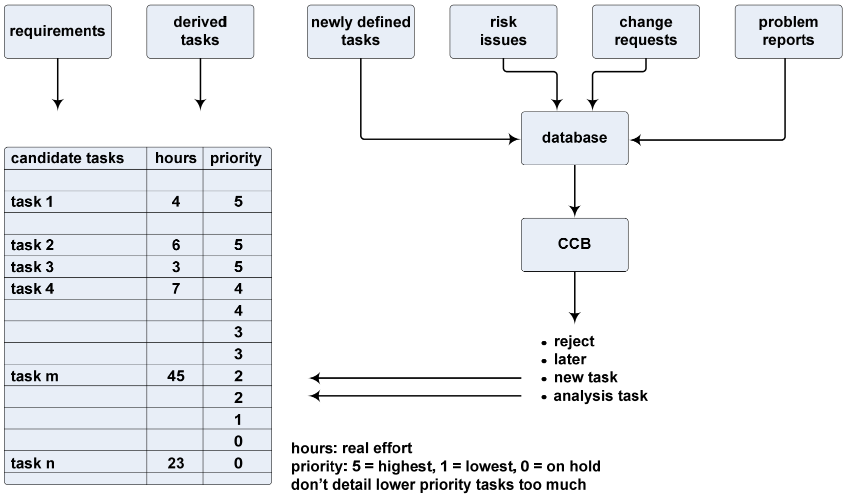
evaluate both result and execution

consult stake-holders

cycle as needed

107

Anything we think must be done goes through the *Candidate Task* Mechanism



requirements

derived tasks

newly defined tasks

risk issues

change requests

problem reports

database

CCB

- reject
- later
- new task
- analysis task

candidate tasks	hours	priority
task 1	4	5
task 2	6	5
task 3	3	5
task 4	7	4
		4
		3
		3
task m	45	2
		2
		1
		0
task n	23	0

hours: real effort
priority: 5 = highest, 1 = lowest, 0 = on hold
don't detail lower priority tasks too much

108

Estimation

- Changing from Optimistic to Realistic
- Only works if we are *Serious about Time*

Sense of Urgency

109

0th- order approximations

- Order of magnitude
- Better than $0 < \text{guess} < \infty$ (Any number is better than no number)
- 0th order is better than *no clue*
- 1st order is often less accurate than 0th order
- Using two different ways of estimation for crosscheck
- Errors may average if we estimate several pieces

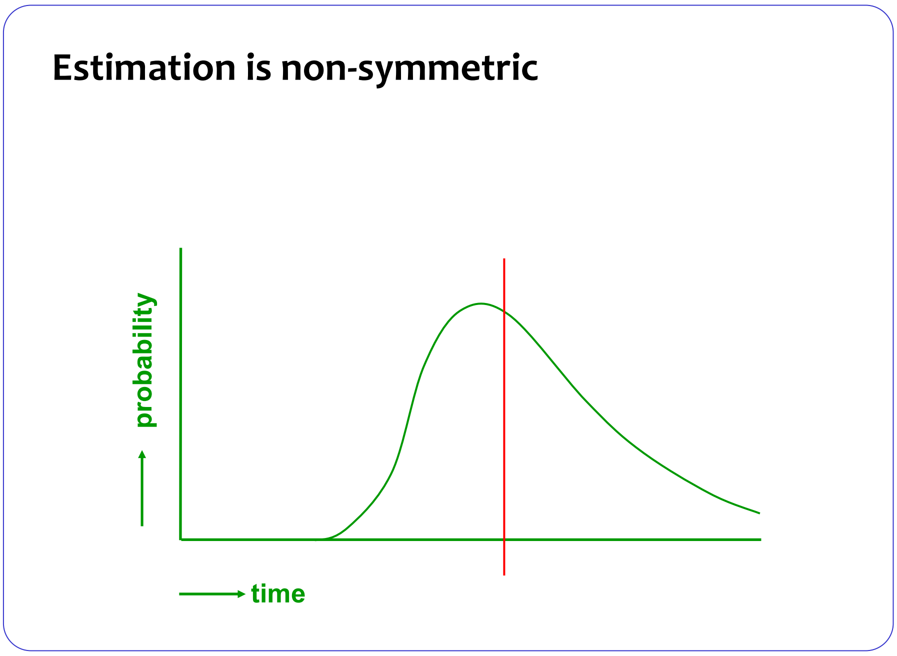
110

Simple Delphi estimation

1. Make a list of things we think we have to do in just enough detail
2. Distribute the list among people who will do the work, or who should be knowledgeable about the work
3. Ask them to add what we apparently forgot, and to estimate how much time the elements of work would cost, “as far as you can judge”
4. In a meeting the estimates are compared
5. If estimates differ significantly between estimators, *do not take the average*, but discuss about the *contents* of the work, *not about the estimate* (some may forget to include things that have to be done, some others may think that more has to be done than necessary)
6. After discussion, people estimate individually again and the estimates are compared again
7. Repeat until sufficient consensus (usually not more than once or twice)
8. Add up all the estimates to end up with an estimate for the whole project

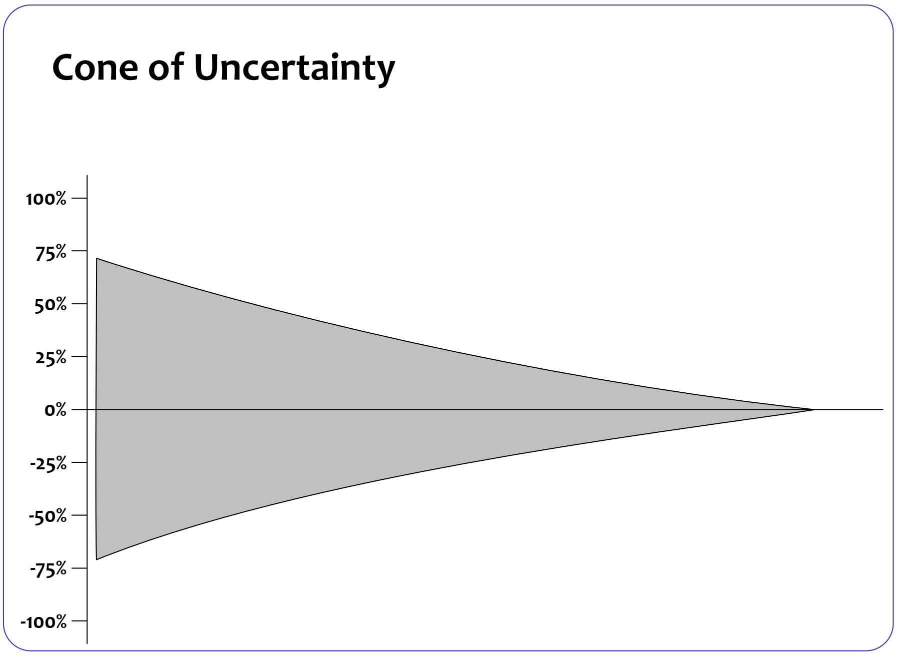
111

Dag2



Estimation is non-symmetric

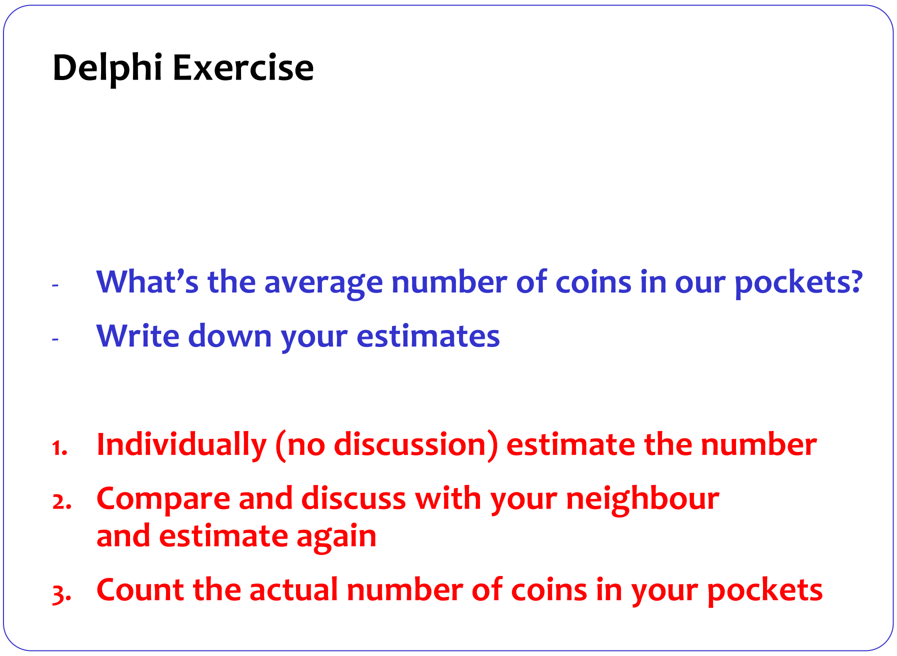
112



Cone of Uncertainty

113

Dag2

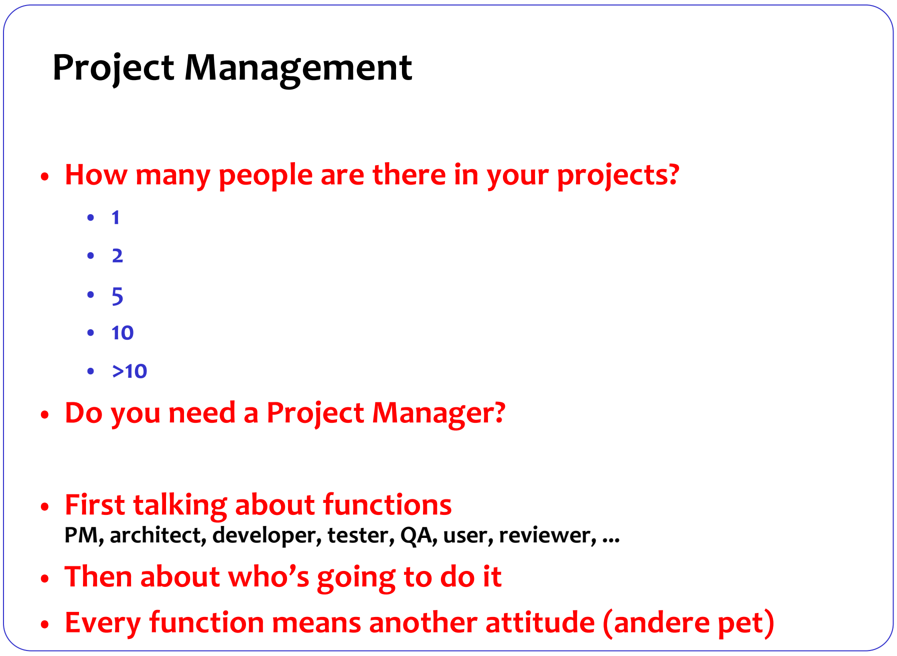


Delphi Exercise

- What's the average number of coins in our pockets?
- Write down your estimates

1. Individually (no discussion) estimate the number
2. Compare and discuss with your neighbour
   and estimate again
3. Count the actual number of coins in your pockets

114

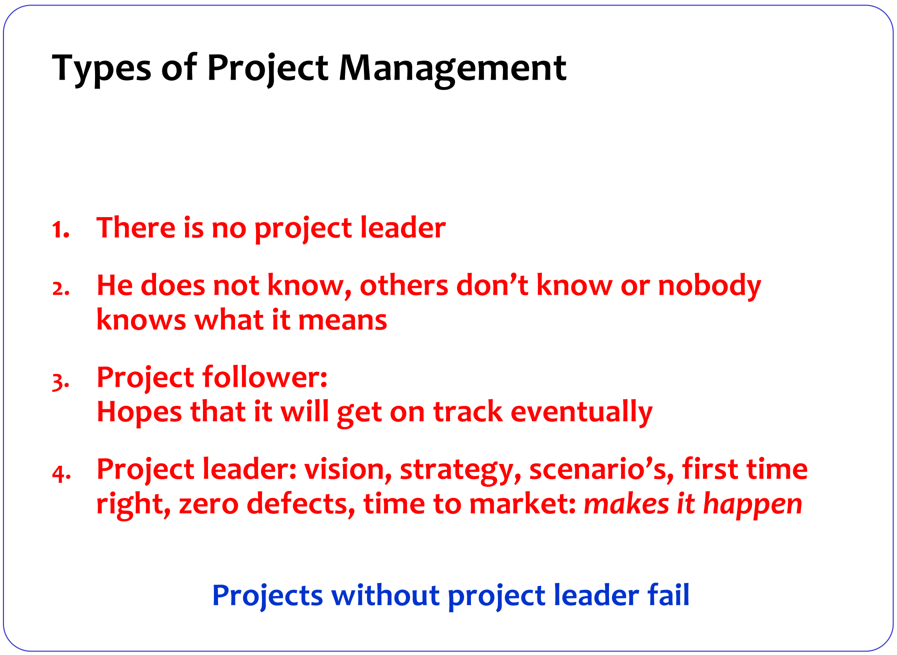


Project Management

- How many people are there in your projects?
  - 1
  - 2
  - 5
  - 10
  - >10
- Do you need a Project Manager?

- First talking about functions
  PM, architect, developer, tester, QA, user, reviewer, …
- Then about who's going to do it
- Every function means another attitude (andere pet)

115

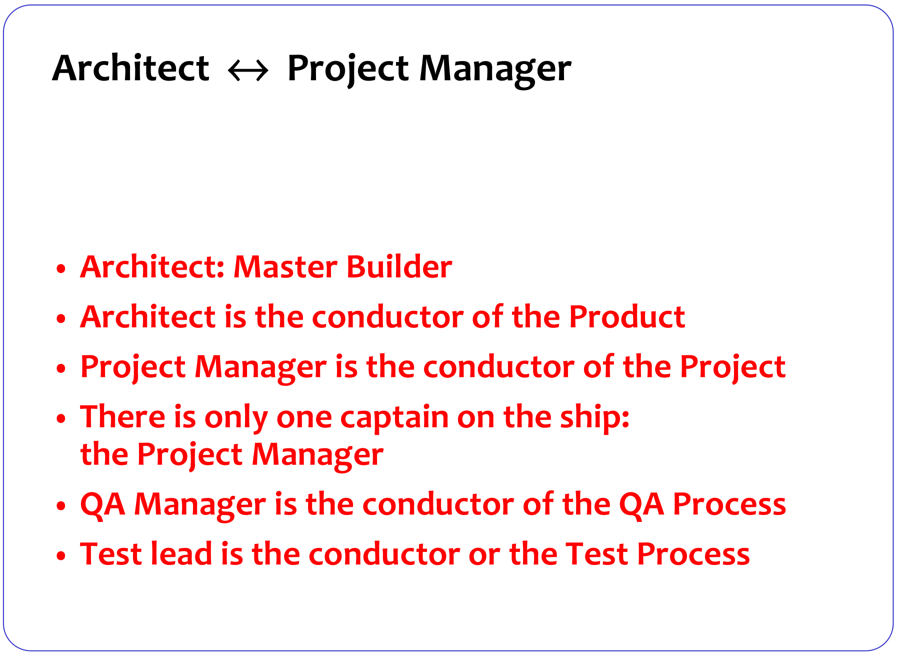


Types of Project Management

1. There is no project leader
2. He does not know, others don't know or nobody knows what it means
3. Project follower:
   Hopes that it will get on track eventually
4. Project leader: vision, strategy, scenario's, first time right, zero defects, time to market: *makes it happen*

Projects without project leader fail

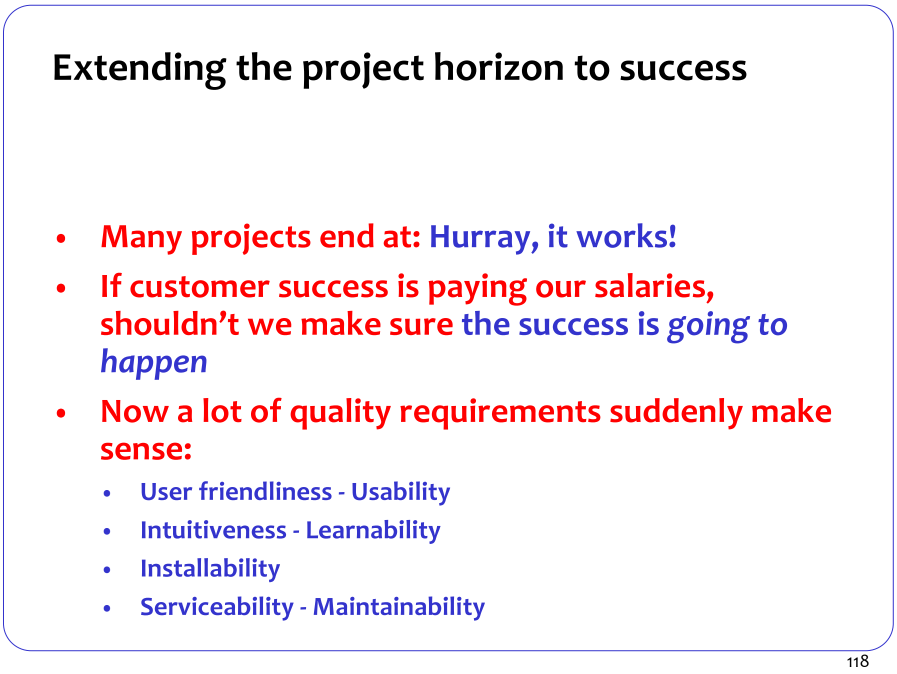
116



Architect ↔ Project Manager

- Architect: Master Builder
- Architect is the conductor of the Product
- Project Manager is the conductor of the Project
- There is only one captain on the ship:
  the Project Manager
- QA Manager is the conductor of the QA Process
- Test lead is the conductor or the Test Process

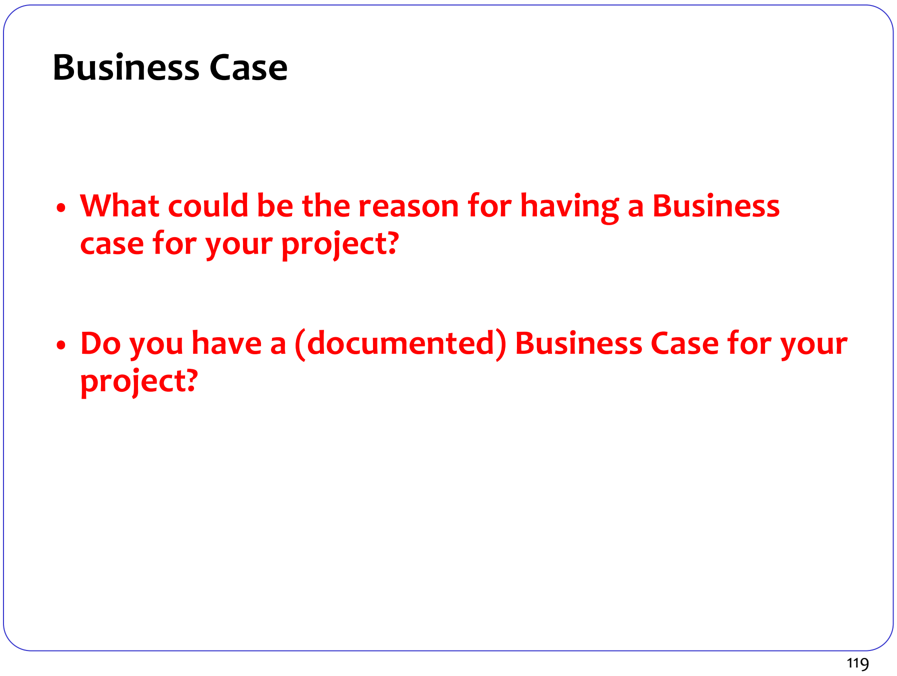
117



Extending the project horizon to success

- Many projects end at: Hurray, it works!
- If customer success is paying our salaries, shouldn't we make sure the success is *going to happen*
- Now a lot of quality requirements suddenly make sense:
  - User friendliness - Usability
  - Intuitiveness - Learnability
  - Installability
  - Serviceability - Maintainability

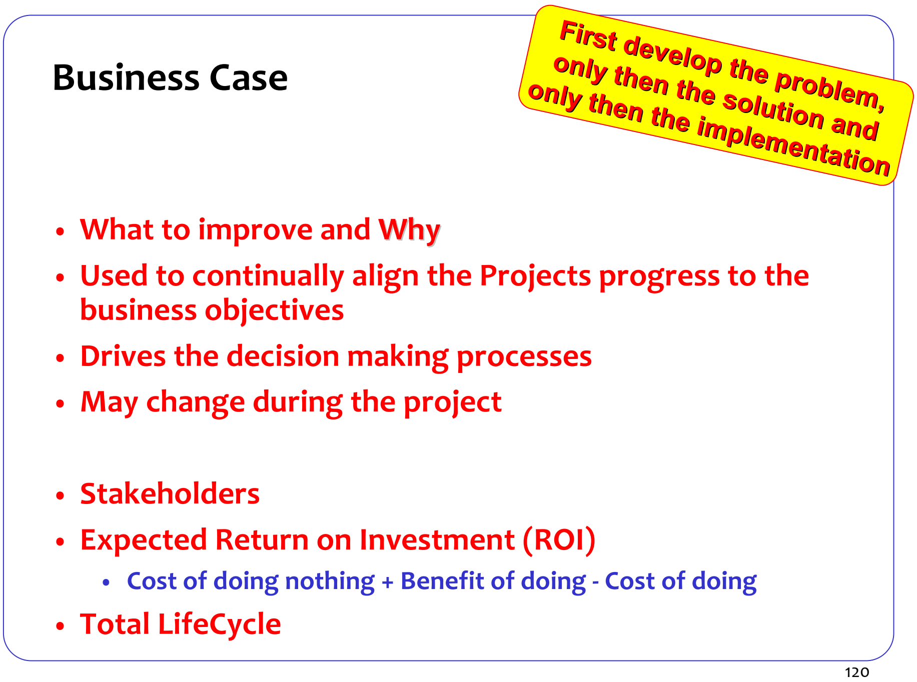
118



Business Case

- What could be the reason for having a Business case for your project?
- Do you have a (documented) Business Case for your project?

119



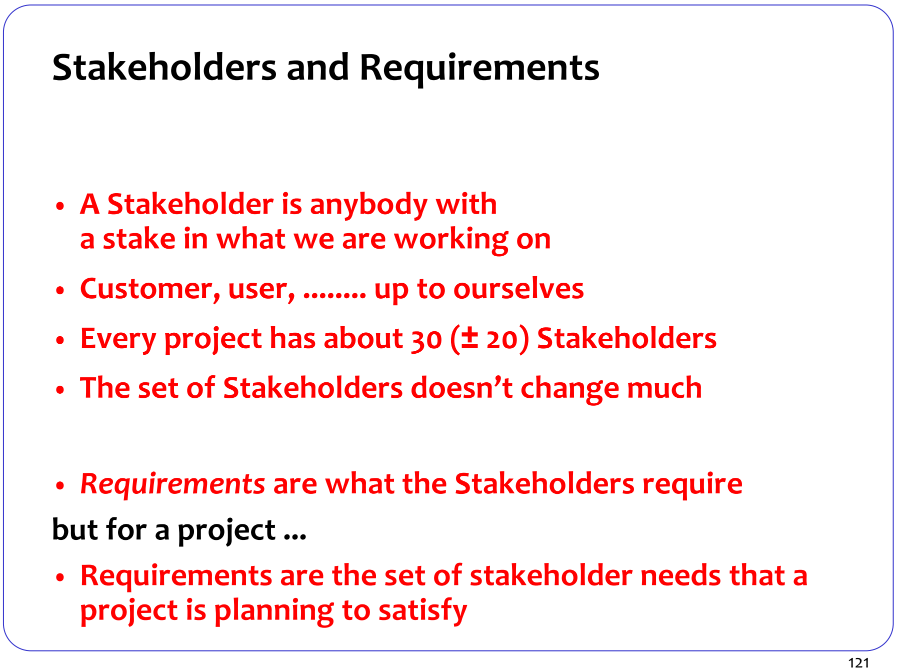
Business Case

First develop the problem, only then the solution and only then the implementation

- What to improve and Why
- Used to continually align the Projects progress to the business objectives
- Drives the decision making processes
- May change during the project

- Stakeholders
- Expected Return on Investment (ROI)
  - Cost of doing nothing + Benefit of doing - Cost of doing
- Total LifeCycle

120



Stakeholders and Requirements

- A Stakeholder is anybody with a stake in what we are working on
- Customer, user, ... up to ourselves
- Every project has about 30 (± 20) Stakeholders
- The set of Stakeholders doesn't change much

- *Requirements* are what the Stakeholders require

but for a project ...

- Requirements are the set of stakeholder needs that a project is planning to satisfy

121

Five times "Why?"

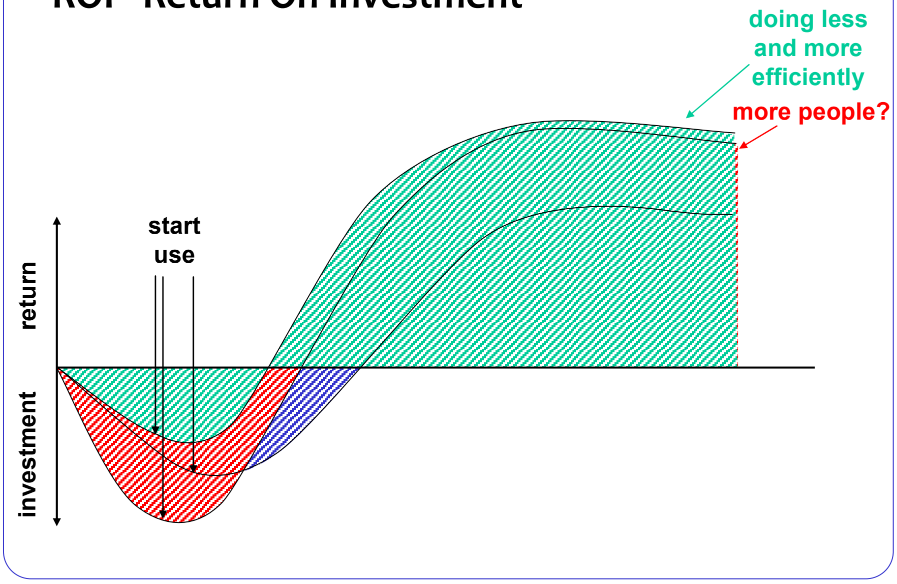
First develop the problem, only then the solution and only then the implementation

- Customer explains what he wants you to do
- You ask: "What's your problem?"
- He says: "I have no problem, just do what I said !"
- You say: "If you have no problem, there is nothing to do" and: "Why do you want me to do what you said; for what purpose?"
- "Well, because ....."
- "And why is that?"
- "Well, because ....."
- etc.

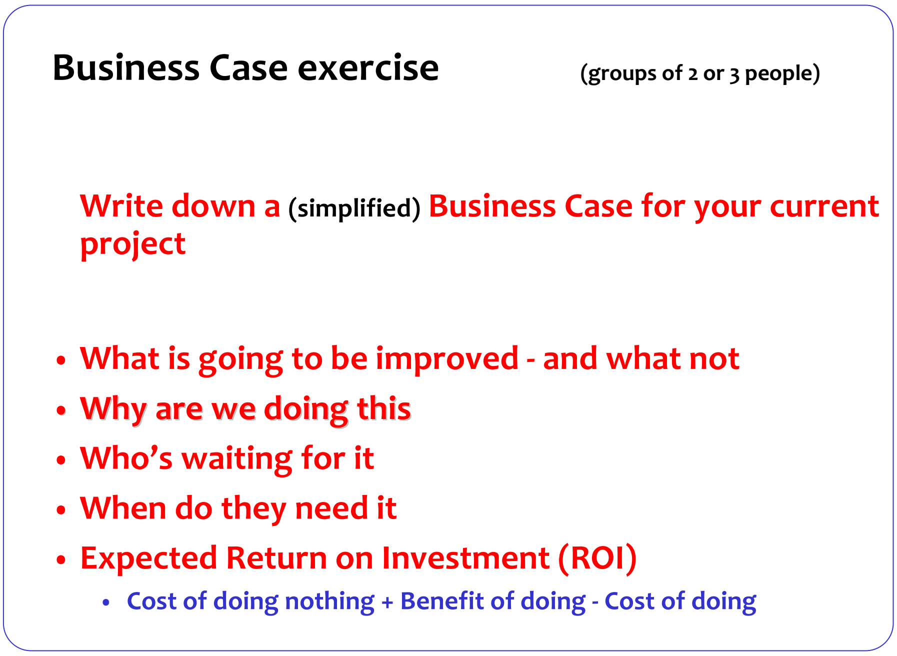
- Go to the bottom and then look for the best solution for the *real* problem
- Within three to five times "Why?" you usually find the real problem

122

ROI - Return On Investment



123



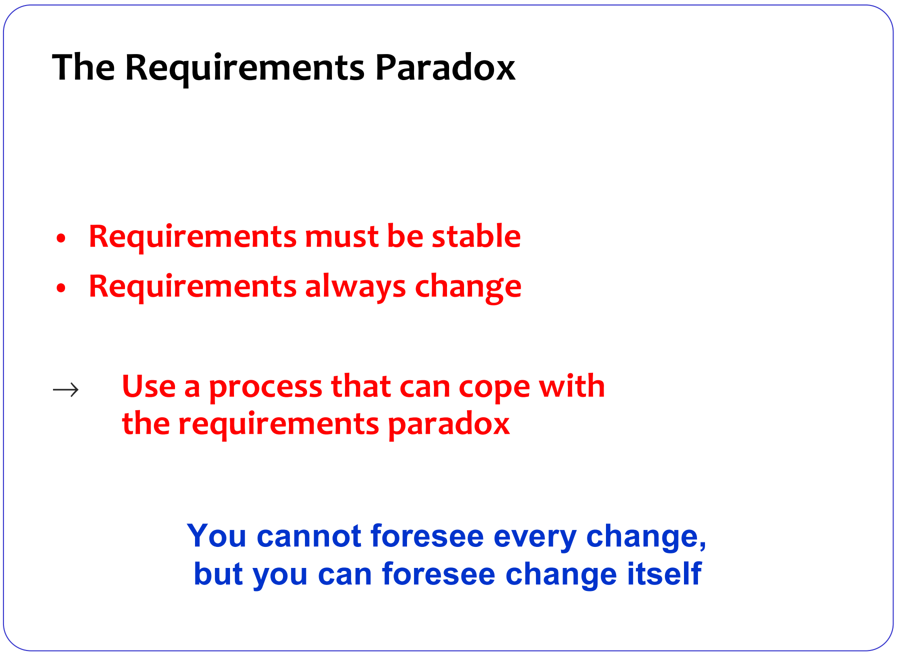
Business Case exercise

(groups of 2 or 3 people)

Write down a (simplified) Business Case for your current project

- What is going to be improved - and what not
- Why are we doing this
- Who's waiting for it
- When do they need it
- Expected Return on Investment (ROI)
  - Cost of doing nothing + Benefit of doing - Cost of doing

124



The Requirements Paradox

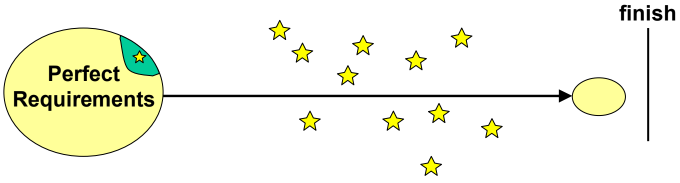
- Requirements must be stable
- Requirements always change

→ Use a process that can cope with the requirements paradox

You cannot foresee every change, but you can foresee change itself

125

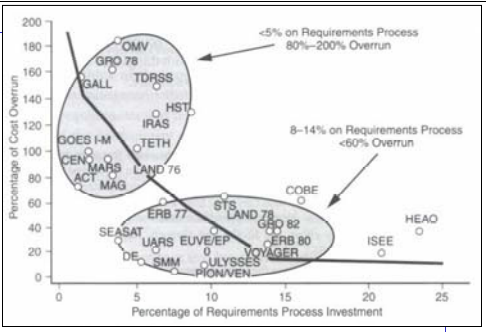
The 2nd requirements paradox



- We don’t want requirements to change, however,
- Because requirements change now is a *known risk:*
  We must *provoke* requirements change
  as early as possible

126

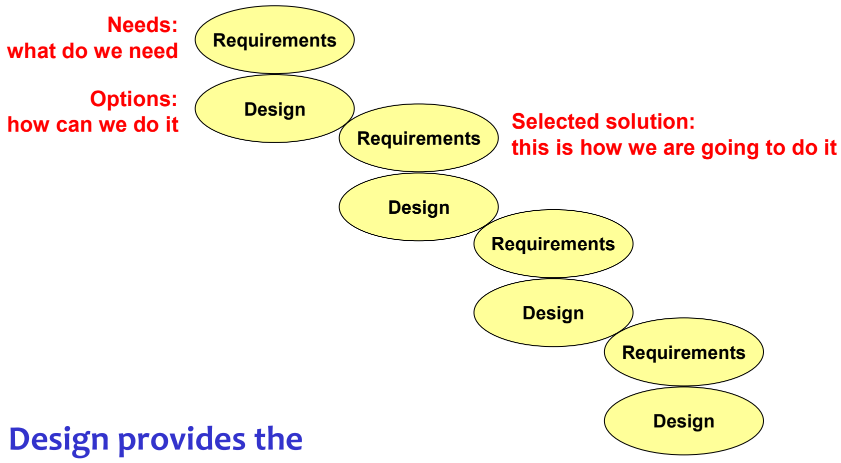
Requirements



- What Stakeholders need
- What the project is planning to satisfy
- You better spend 10 ~ 15% of the project time on Requirements in order to *save* time
- No design (*how* it is to be done)

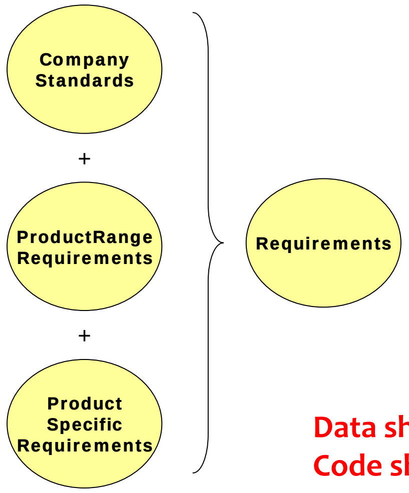
127

No Design in the requirements, but …

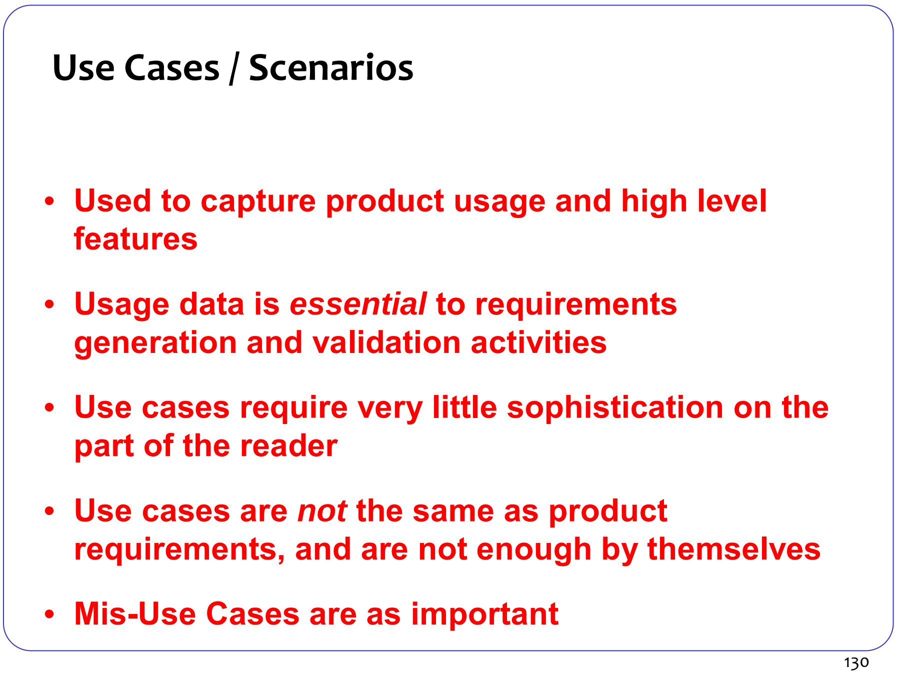


Design provides the
Requirements for the next level

Requirements should be at one place only



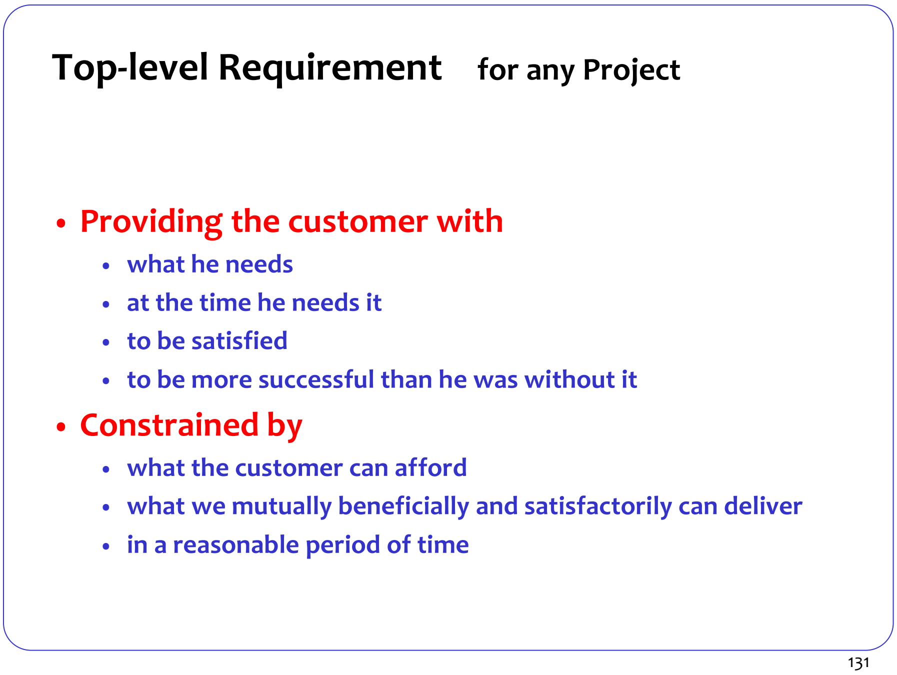
Data should be at one place only
Code should be at one place only



Use Cases / Scenarios

- **Used to capture product usage and high level features**
- **Usage data is *essential* to requirements generation and validation activities**
- **Use cases require very little sophistication on the part of the reader**
- **Use cases are *not* the same as product requirements, and are not enough by themselves**
- **Mis-Use Cases are as important**

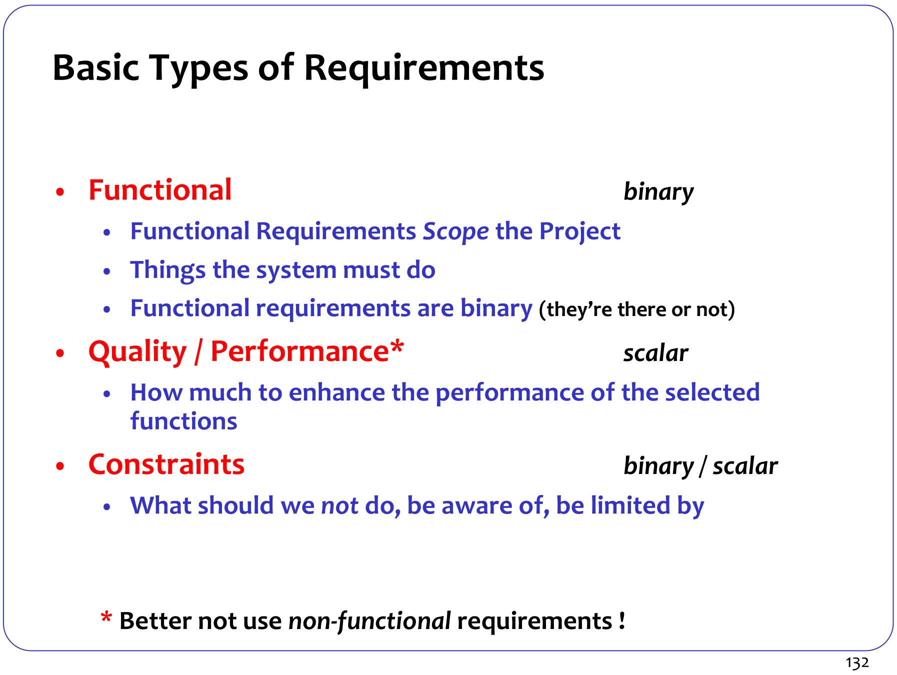
130



Top-level Requirement for any Project

- 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
  - what the customer can afford
  - what we mutually beneficially and satisfactorily can deliver
  - in a reasonable period of time

131

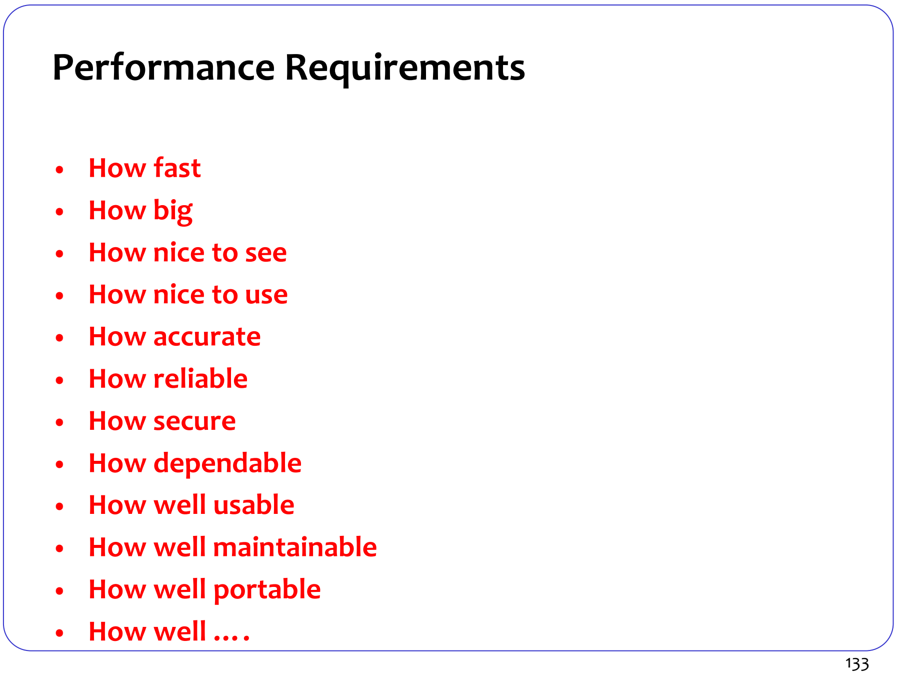


Basic Types of Requirements

- Functional *binary*
  - Functional Requirements *Scope* the Project
  - Things the system must do
  - Functional requirements are binary (they're there or not)
- Quality / Performance* *scalar*
  - How much to enhance the performance of the selected functions
- Constraints *binary / scalar*
  - What should we *not* do, be aware of, be limited by

* Better not use *non-functional* requirements !

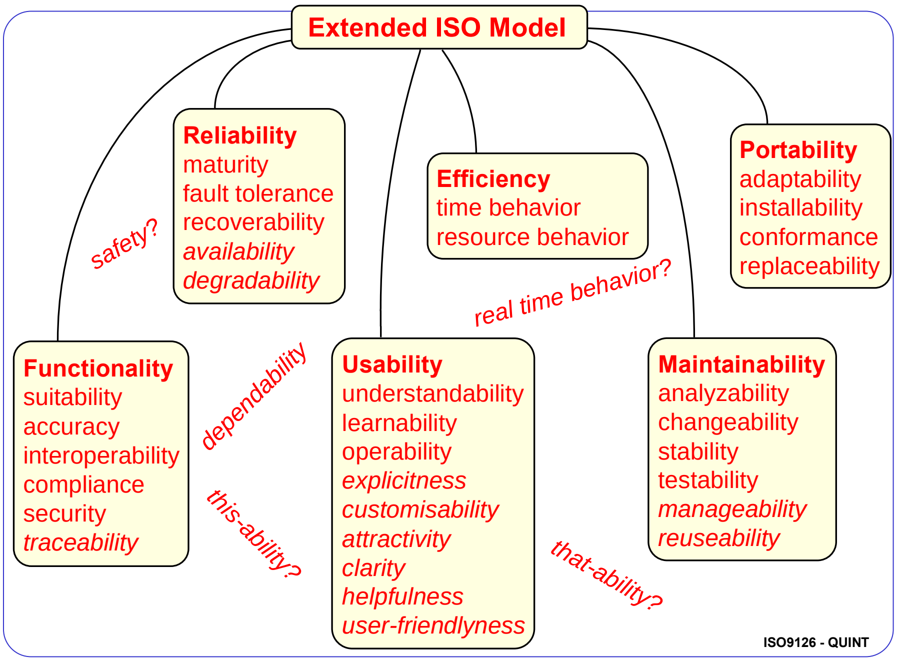
132



Performance Requirements

- How fast
- How big
- How nice to see
- How nice to use
- How accurate
- How reliable
- How secure
- How dependable
- How well usable
- How well maintainable
- How well portable
- How well ….

133



Extended ISO Model

Functionality
suitability
accuracy
interoperability
compliance
security
traceability

Reliability
maturity
fault tolerance
recoverability
availability
degradability

Usability
understandability
learnability
operability
explicitness
customisability
attractivity
clarity
helpfulness
user-friendlyness

Efficiency
time behavior
resource behavior

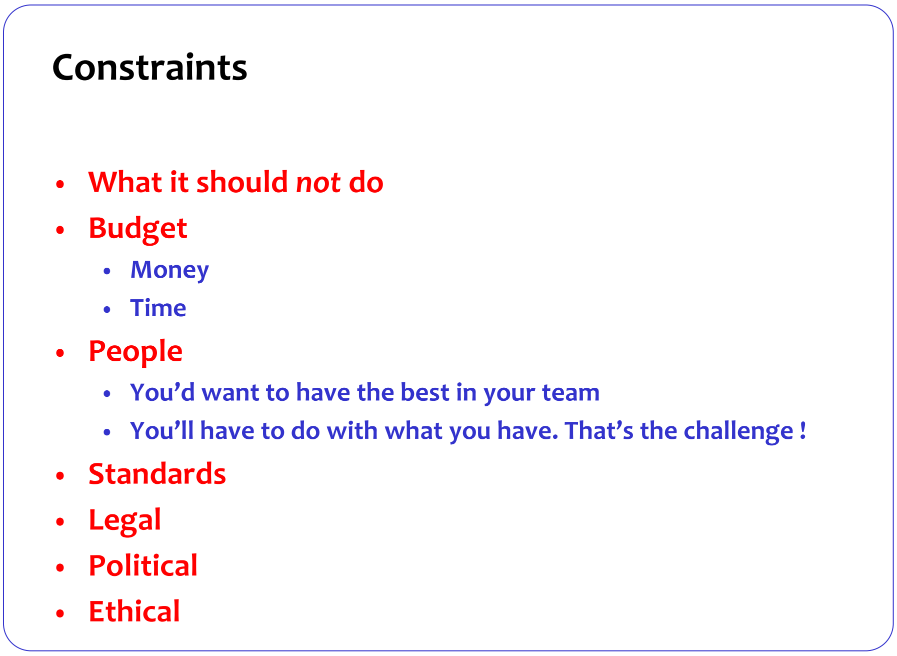
Maintainability
analyzability
changeability
stability
testability
manageability
reuseability

Portability
adaptability
installability
conformance
replaceability

safety?
dependability
this-ability?
real time behavior?
that-ability?

ISO9126 - QUINT

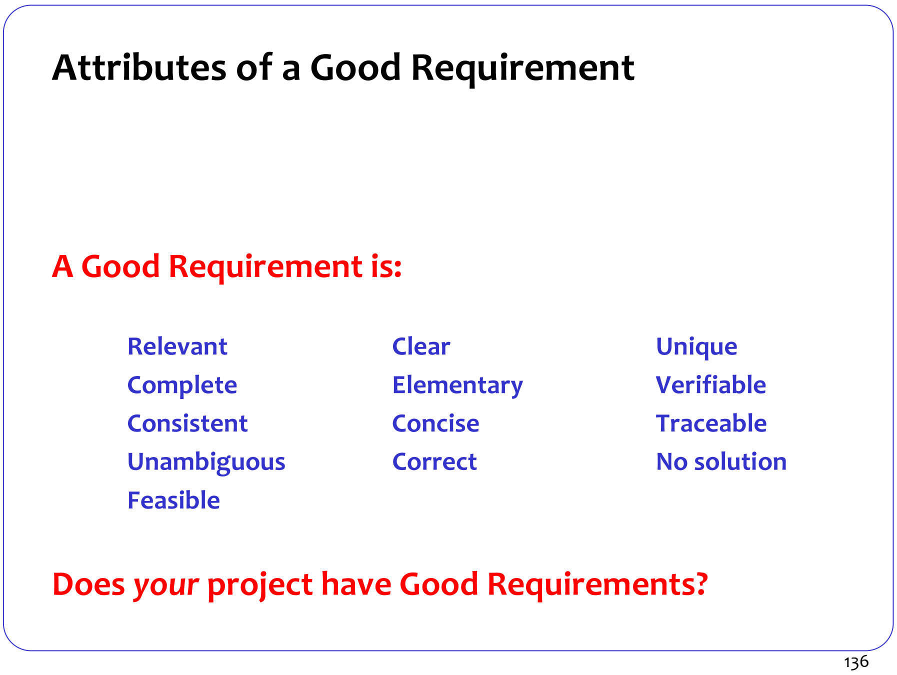
134



Constraints

- What it should *not* do
- Budget
  - Money
  - Time
- People
  - You'd want to have the best in your team
  - You'll have to do with what you have. That's the challenge !
- Standards
- Legal
- Political
- Ethical

135



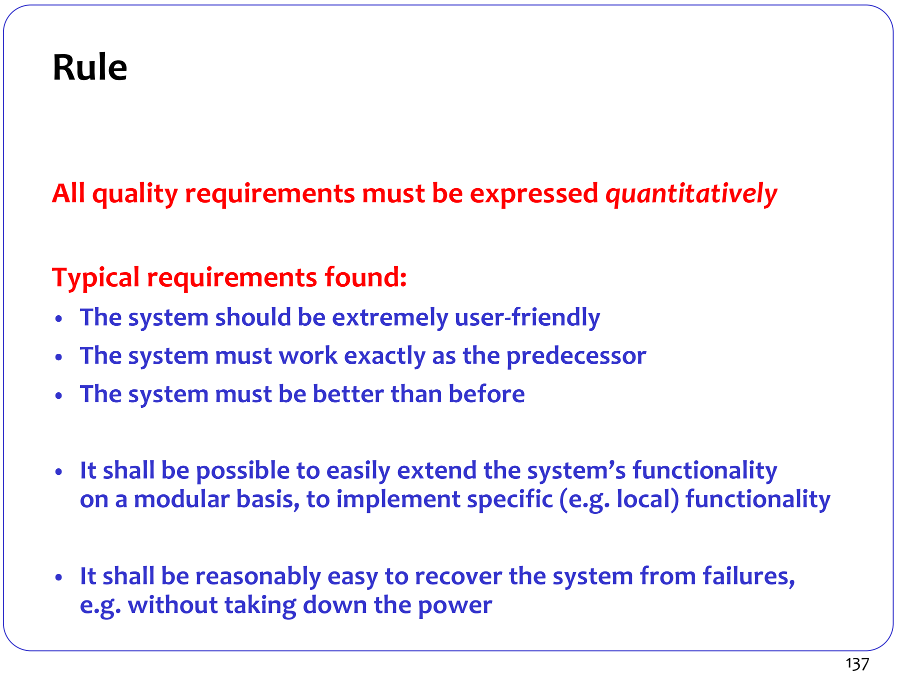
Attributes of a Good Requirement

A Good Requirement is:

- Relevant
- Complete
- Consistent
- Unambiguous
- Feasible
- Clear
- Elementary
- Concise
- Correct
- Unique
- Verifiable
- Traceable
- No solution

Does *your* project have Good Requirements?

136



Rule

All quality requirements must be expressed *quantitatively*

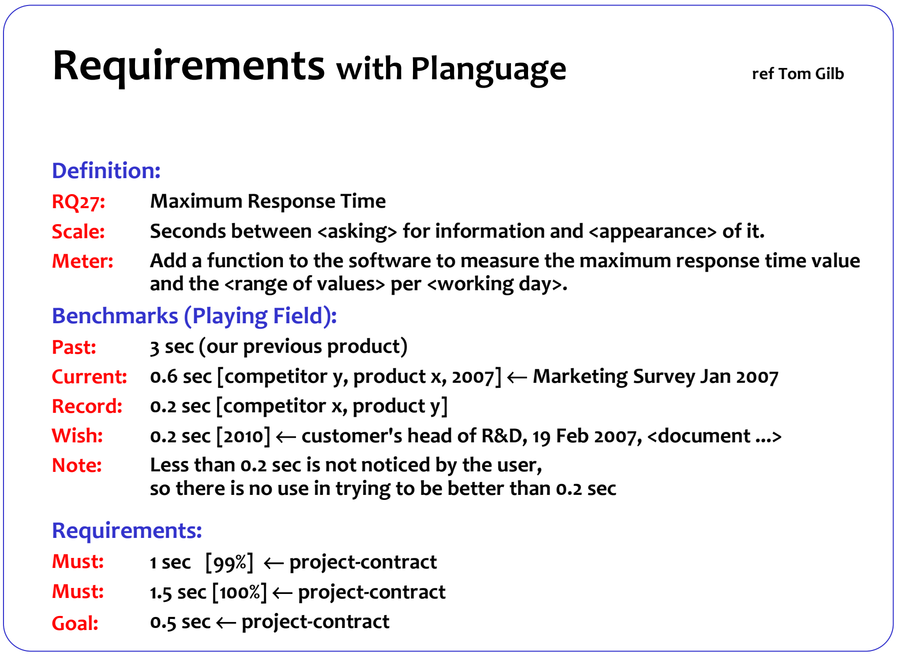
Typical requirements found:

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

- 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

137



Requirements with Planguage

ref Tom Gilb

Definition:

RQ27: Maximum Response Time

Scale: Seconds between <asking> for information and <appearance> of it.

Meter: Add a function to the software to measure the maximum response time value and the <range of values> per <working day>.

Benchmarks (Playing Field):

Past: 3 sec (our previous product)

Current: 0.6 sec [competitor y, product x, 2007] ← Marketing Survey Jan 2007

Record: 0.2 sec [competitor x, product y]

Wish: 0.2 sec [2010] ← customer's head of R&D, 19 Feb 2007, <document ...>

Note: Less than 0.2 sec is not noticed by the user,
so there is no use in trying to be better than 0.2 sec

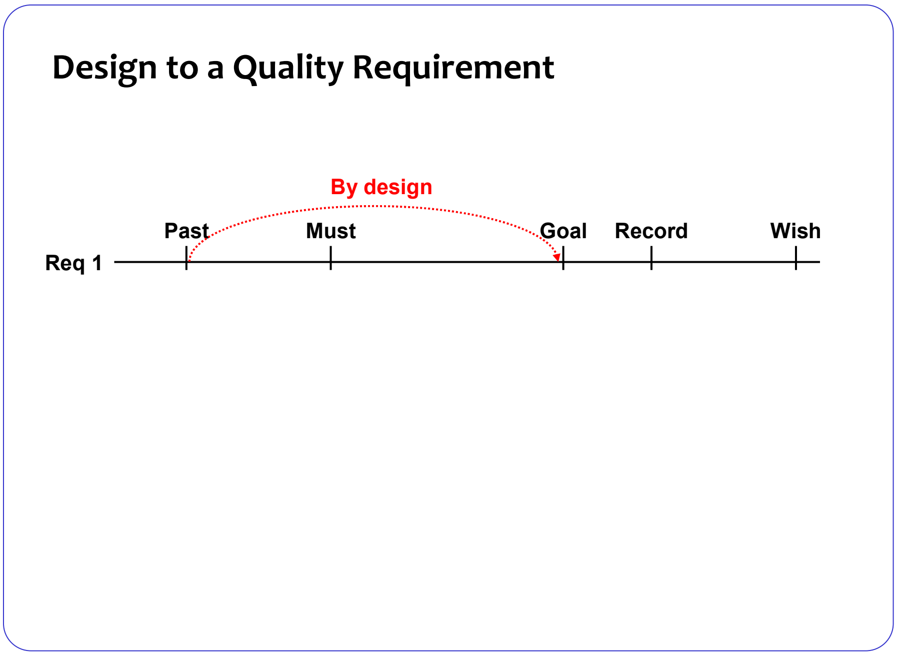
Requirements:

Must: 1 sec [99%] ← project-contract

Must: 1.5 sec [100%] ← project-contract

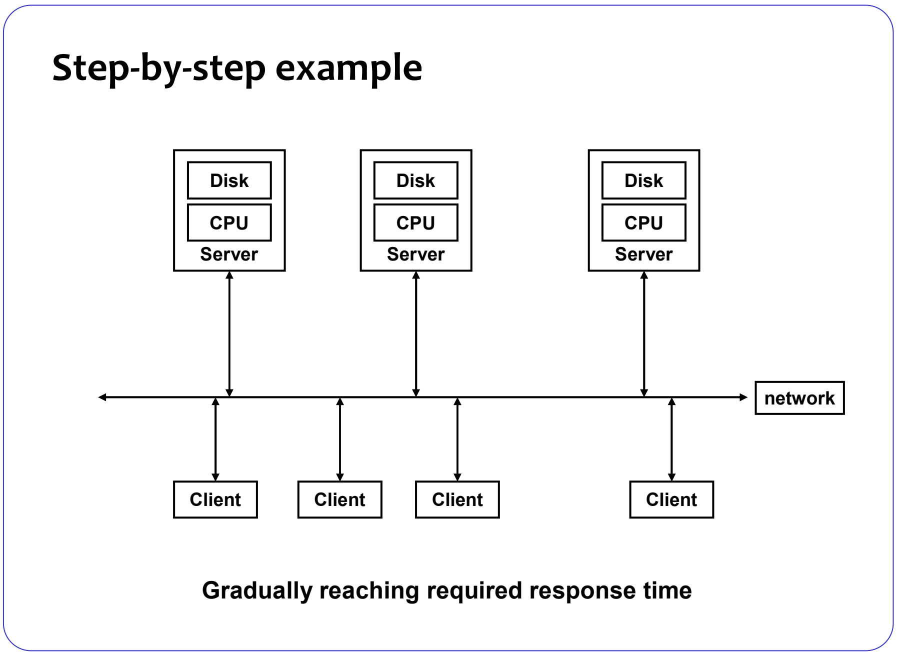
Goal: 0.5 sec ← project-contract

138



Design to a Quality Requirement

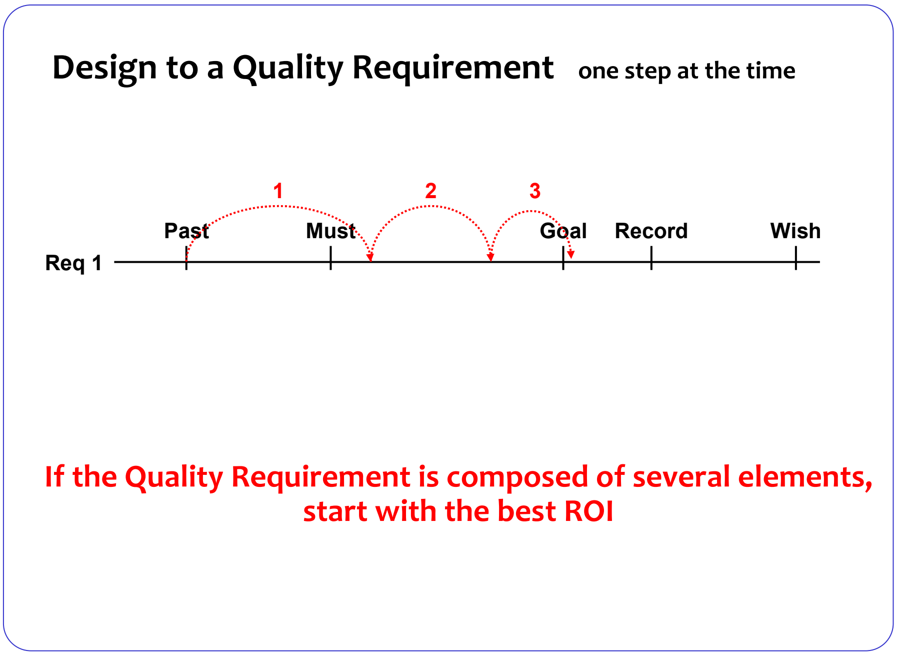
139



Step-by-step example

Gradually reaching required response time

140

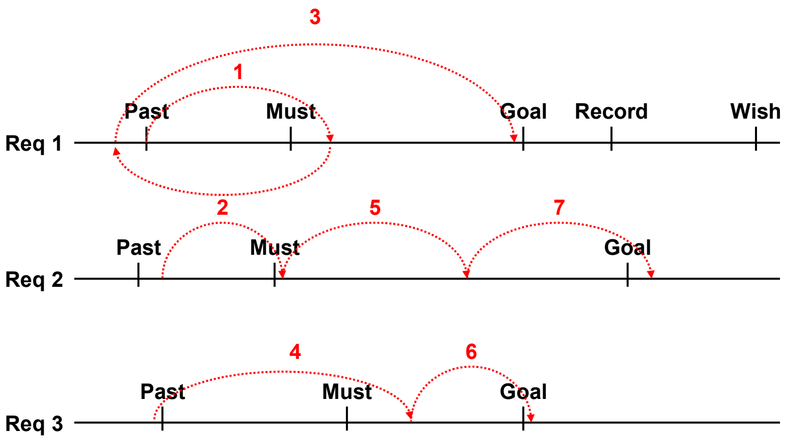


Design to a Quality Requirement one step at the time

If the Quality Requirement is composed of several elements, start with the best ROI

141

Design to Multidimensional Quality Requirements

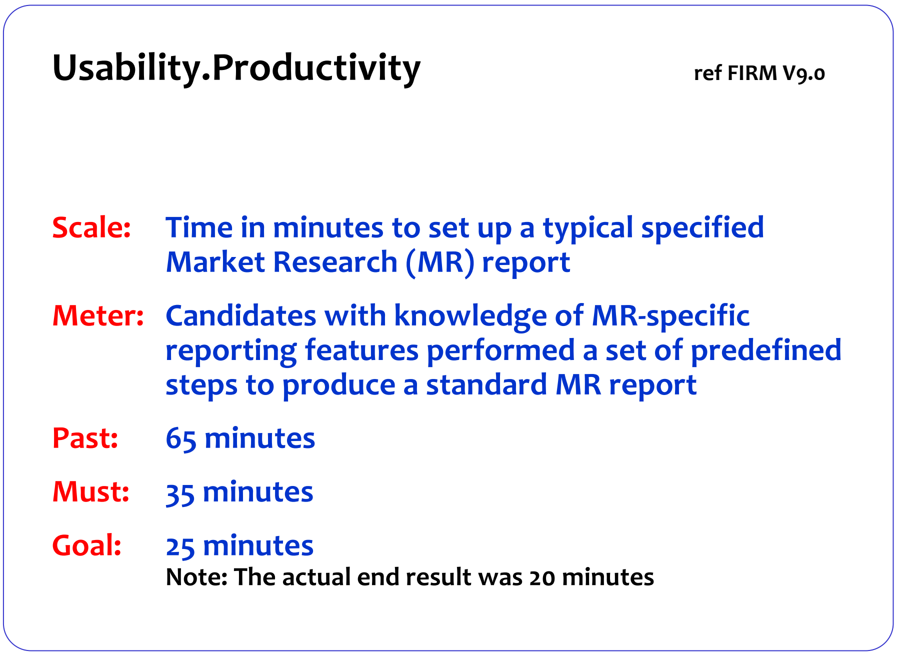


142

Dependability is a Complex Concept

- **Availability**
  Readiness for correct service
  Scale: % per [TimePeriod] a [System] is [Available for its Tasks]
  Example: The ATM will be available to supply cash 99,9% of the year
  8,76 hr down per year…
- **Reliability**
  Continuity of correct service
  Scale: Mean time for a [System] to experience [Failure Type] under [Conditions]
  Example: The ATM will always (100%) deliver the correct amount of cash
- **Safety**
  No danger, harm, risk
  Example: star-system for cars (adult / child, in-car / pedestrian)
- **Security**
  Free from intrusions (theft, alteration)
  Scale: Time required to <break into the system>
  Example: It will take the <best hackers we can find> >8 hrs to <break in>

143



Usability.Productivity

ref FIRM V9.0

Scale: Time in minutes to set up a typical specified Market Research (MR) report

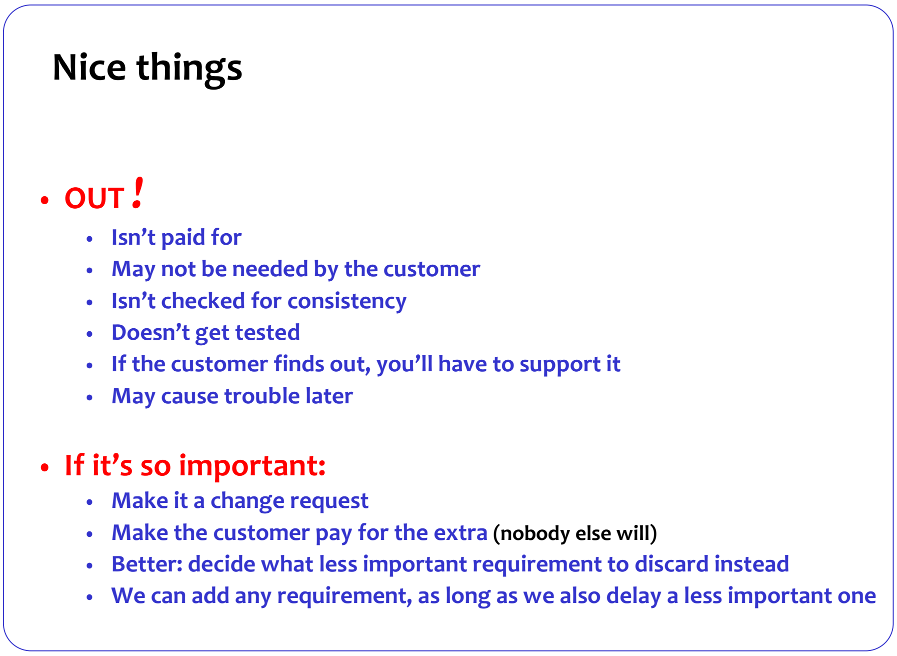
Meter: Candidates with knowledge of MR-specific reporting features performed a set of predefined steps to produce a standard MR report

Past: 65 minutes

Must: 35 minutes

Goal: 25 minutes
Note: The actual end result was 20 minutes

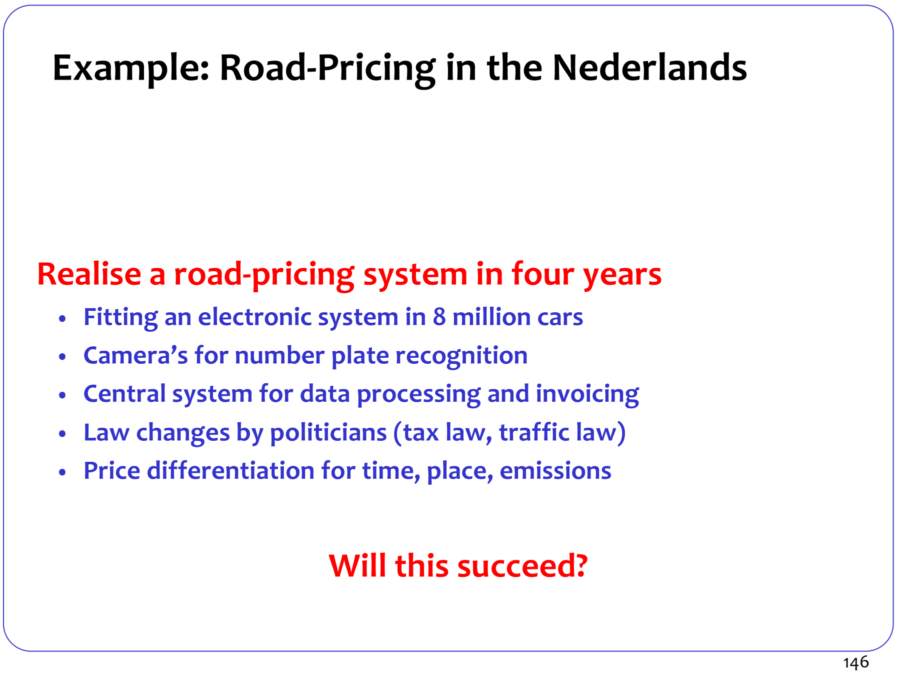
144



Nice things

- OUT !
  - Isn't paid for
  - May not be needed by the customer
  - Isn't checked for consistency
  - Doesn't get tested
  - If the customer finds out, you'll have to support it
  - May cause trouble later
- If it's so important:
  - Make it a change request
  - Make the customer pay for the extra (nobody else will)
  - Better: decide what less important requirement to discard instead
  - We can add any requirement, as long as we also delay a less important one

145



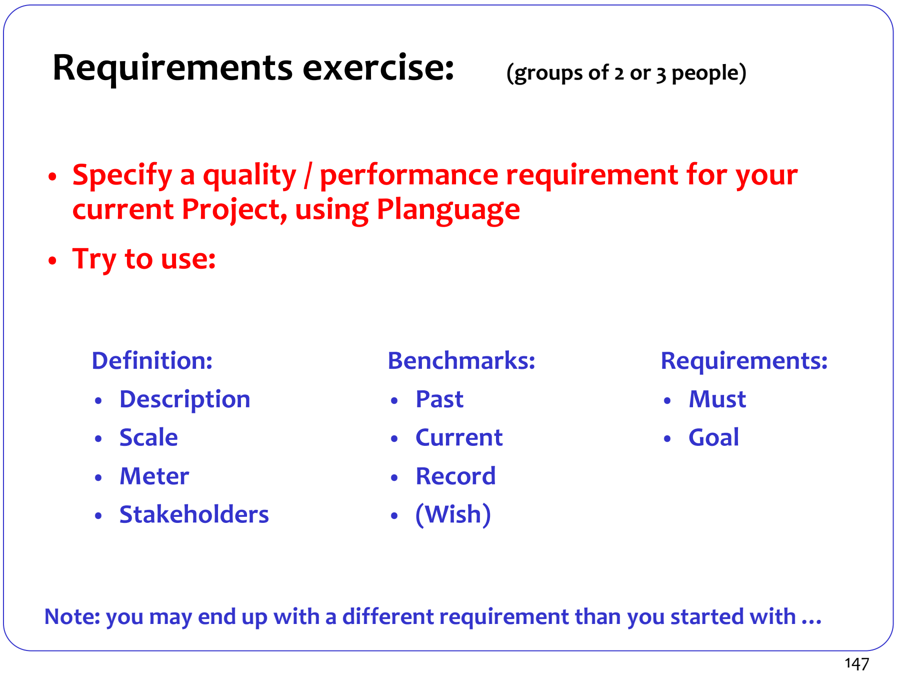
Example: Road-Pricing in the Nederlands

Realise a road-pricing system in four years

- Fitting an electronic system in 8 million cars
- Camera's for number plate recognition
- Central system for data processing and invoicing
- Law changes by politicians (tax law, traffic law)
- Price differentiation for time, place, emissions

Will this succeed?

146



Requirements exercise: (groups of 2 or 3 people)

- Specify a quality / performance requirement for your current Project, using Planguage
- Try to use:

Definition:

- Description
- Scale
- Meter
- Stakeholders

Benchmarks:

- Past
- Current
- Record
- (Wish)

Requirements:

- Must
- Goal

Note: you may end up with a different requirement than you started with …

147

Req	
Scale	
Meter	
Stakehldrs	
Past	
Current	
Record	
Wish	
Must	
Goal	

148

Design is always a compromise

- Design is the process of collecting and selecting options how to implement the requirements
- The Requirements are *always* conflicting

example:

- Performance ↑
- Budget (time, money) ↓

149

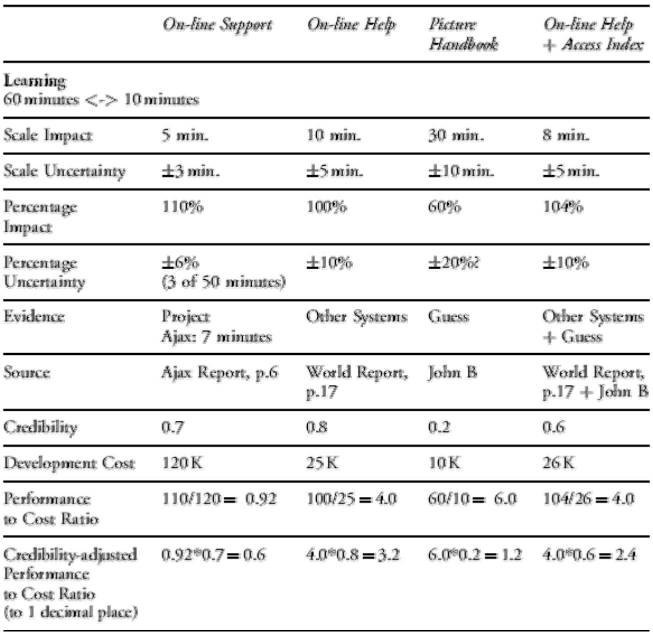
Design Process

- Collect obvious (voor de hand liggend) design(s)
- Search for *one* non-obvious design
- Compare the relative ROI of the designs
- Select the best compromise
- Describe the selected design

- Books:
  - Ralph L. Keeyney: Value Focused Thinking
  - Gerd Gigerenzer: Simple Heuristics That Make Us Smart

150

Impact Estimation



	On-line Support	*On-line Help*	*Picture Handbook*	*On-line Help + Access Index*
Learning 60 minutes <-> 10 minutes				
Scale Impact	5 min.	10 min.	30 min.	8 min.
Scale Uncertainty	±3 min.	±5 min.	±10 min.	±5 min.
Percentage Impact	110%	100%	60%	104%
Percentage Uncertainty	±6% (3 of 50 minutes)	±10%	±20%?	±10%
Evidence	Project Ajax: 7 minutes	Other Systems	Guess	Other Systems + Guess
Source	Ajax Report, p.6	World Report, p.17	John B	World Report, p.17 + John B
Credibility	0.7	0.8	0.2	0.6
Development Cost	120 K	25 K	10 K	26 K
Performance to Cost Ratio	110/120 = 0.92	100/25 = 4.0	60/10 = 6.0	104/26 = 4.0
Credibility-adjusted Performance to Cost Ratio (to 1 decimal place)	0.92*0.7 = 0.6	4.0*0.8 = 3.2	6.0*0.2 = 1.2	4.0*0.6 = 2.4

ref
Tom Gilb
Competitive Engineering

151

Dag2

DesignLog

(project level)

- In computer, not loose notes, not in e-mails, not handwritten
  - Text
  - Drawings!
  - On subject order
  - Initially free-format
  - For all to see
- All concepts contemplated
  - Requirements
  - Assumptions
  - Questions
  - Available techniques
  - Calculations
  - Choices + argumentation:
    - If rejected: why?
    - If chosen: why?
- Rejected choices
- Final (current) choices
- Implementation

<u>Chapter</u>
Requirement → What to achieve
.
Assumptions
Questions + Answers
.
.
.
.
Design options
Decision criteria
Decision → implementation spec

New date: change of idea:
Design options
Decision criteria
Decision → implementation spec

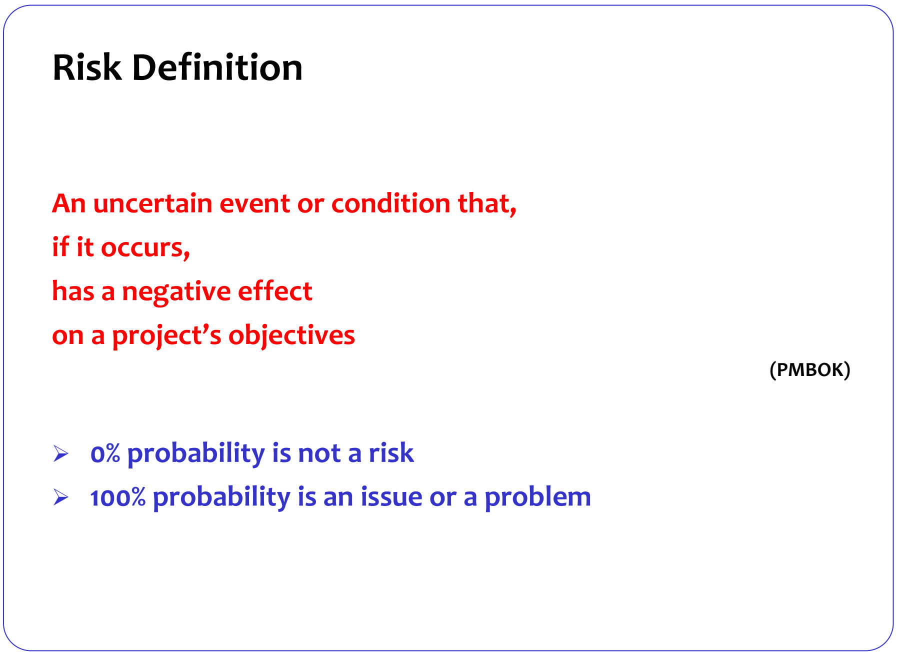
ProcessLog

(department / organization level)

- In computer, not loose notes, not in e-mails, not handwritten
  - Text
  - Graphics (drawings)
  - On subject order
  - Initially free-format
  - For all to see
- All concepts contemplated
  - Related requirement
  - Assumptions
  - Questions
  - Known techniques
  - Choices + argumentation:
    - If rejected: why?
    - If chosen: why?
- Rejected choices
- Final (current) choices

<u>Chapter</u>
Requirement → What to achieve
.
Assumptions
Questions + Answers
.
.
.
.
Design options
Decision criteria
Decision → implementation spec

New date: change of idea:
Design options
Decision criteria
Decision → implementation spec

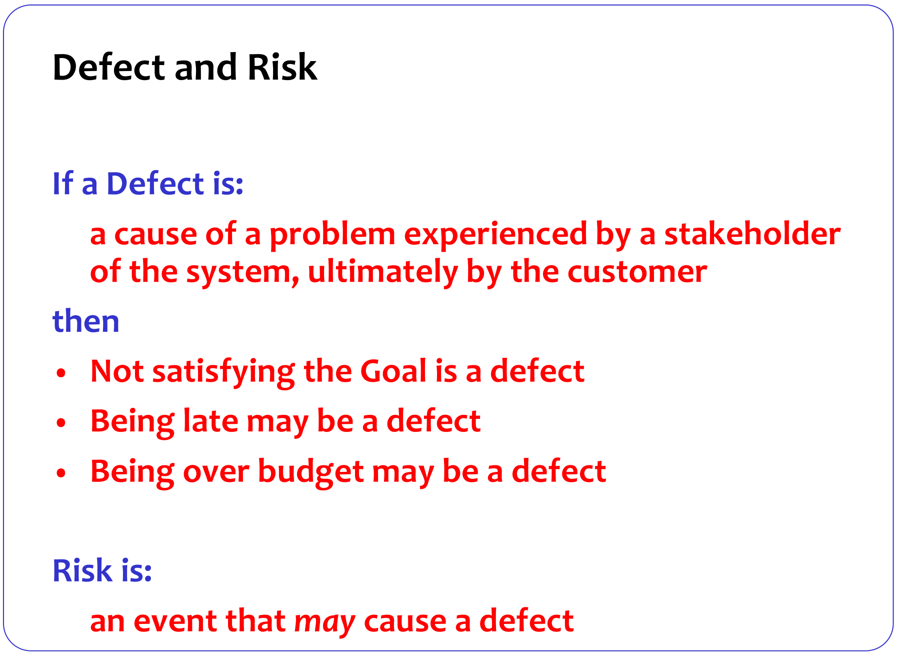


Risk Definition

An uncertain event or condition that,
if it occurs,
has a negative effect
on a project's objectives

(PMBOK)

- ➢ 0% probability is not a risk
- ➢ 100% probability is an issue or a problem



Defect and Risk

If a Defect is:

a cause of a problem experienced by a stakeholder of the system, ultimately by the customer

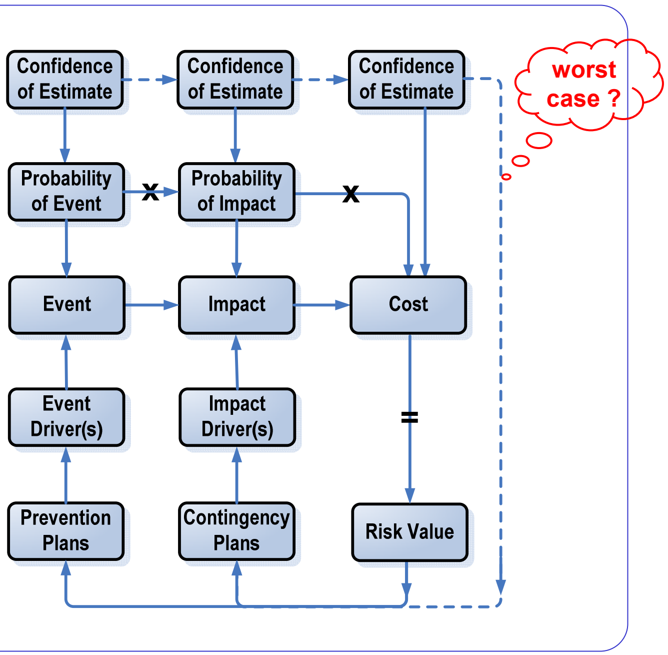
then

- Not satisfying the Goal is a defect
- Being late may be a defect
- Being over budget may be a defect

Risk is:

an event that *may* cause a defect

Risk Model



$$V_R = P_e * P_i * C$$

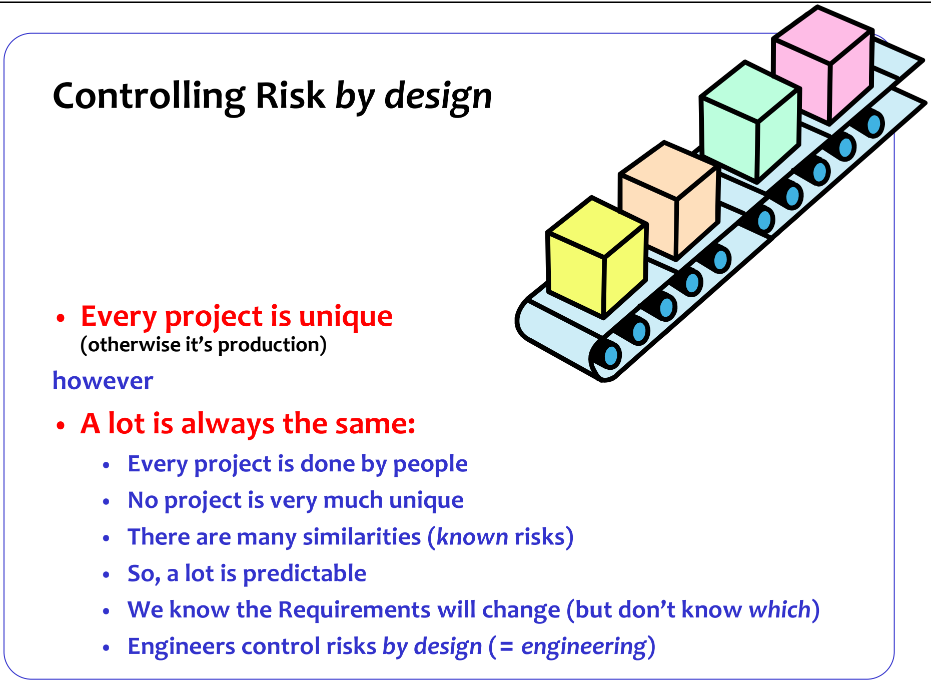
156

What are Risks in your Projects?

- ...
- ...
- ...

- Are these really Risks?
- 0% probablity is not a Risk
- 100% probability is not a Risk

157



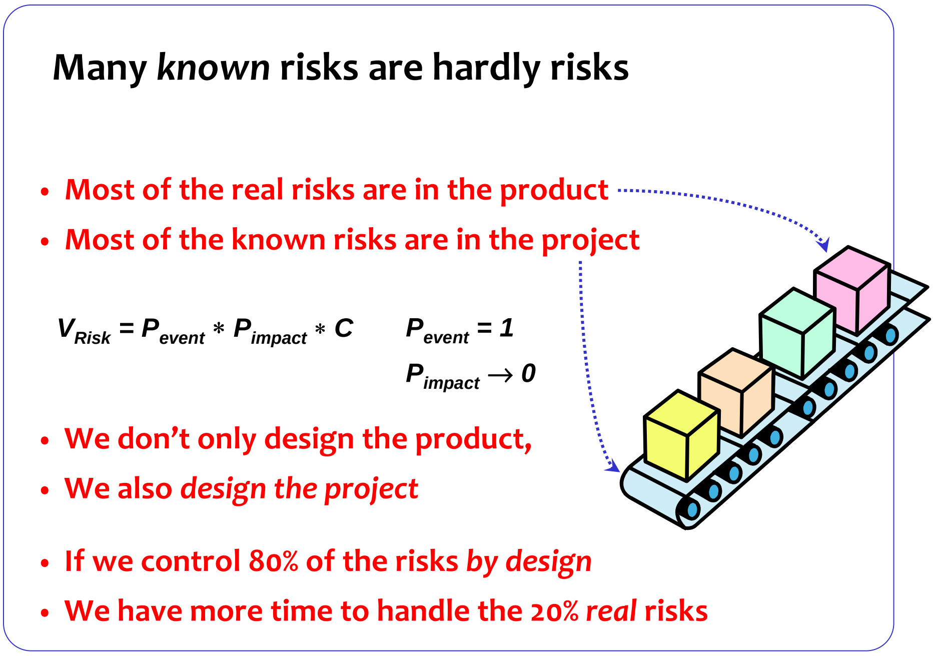
Controlling Risk *by design*

- Every project is unique
  (otherwise it's production)

however

- A lot is always the same:
  - Every project is done by people
  - No project is very much unique
  - There are many similarities (*known* risks)
  - So, a lot is predictable
  - We know the Requirements will change (but don't know *which*)
  - Engineers control risks *by design* (= *engineering*)

158



Many *known* risks are hardly risks

- Most of the real risks are in the product
- Most of the known risks are in the project

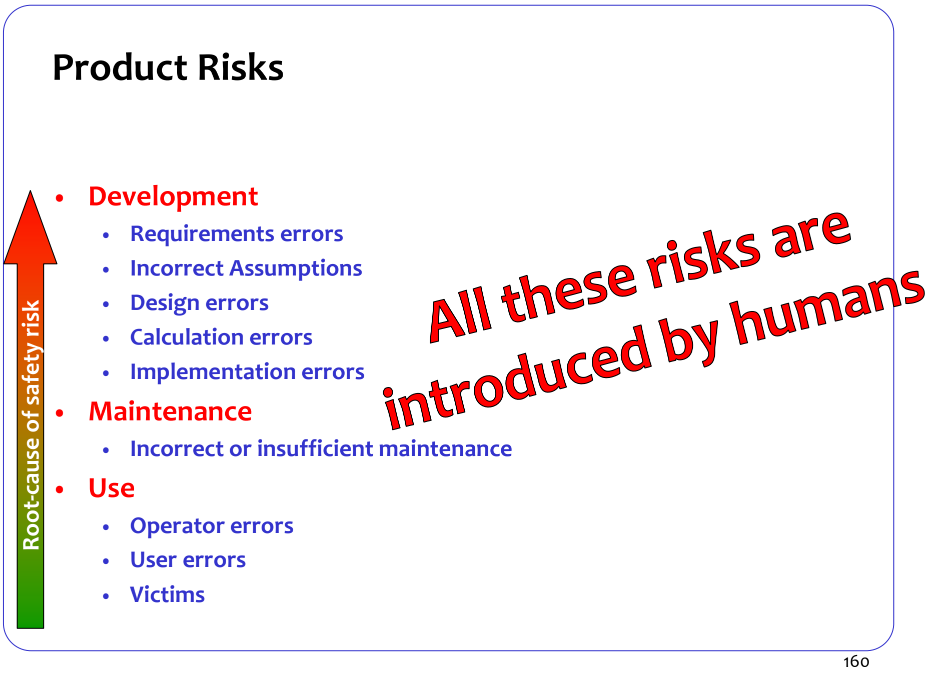
$V_{Risk} = P_{event} * P_{impact} * C$ $\qquad P_{event} = 1$

$P_{impact} \to 0$

- We don't only design the product,
- We also *design the project*

- If we control 80% of the risks *by design*
- We have more time to handle the 20% *real* risks

159



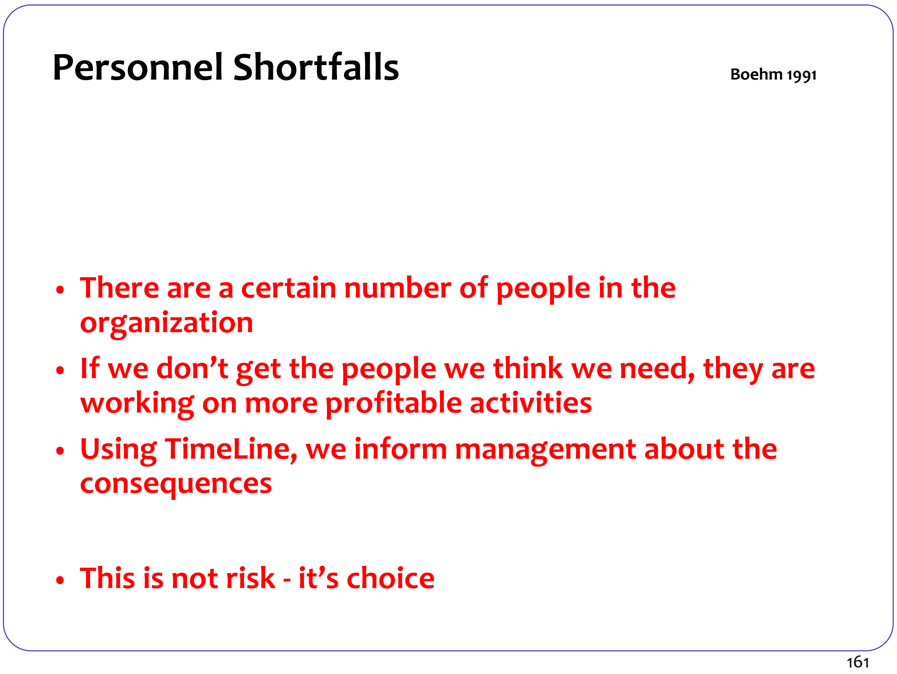
Product Risks

Root-cause of safety risk

- Development
  - Requirements errors
  - Incorrect Assumptions
  - Design errors
  - Calculation errors
  - Implementation errors
- Maintenance
  - Incorrect or insufficient maintenance
- Use
  - Operator errors
  - User errors
  - Victims

All these risks are introduced by humans

160



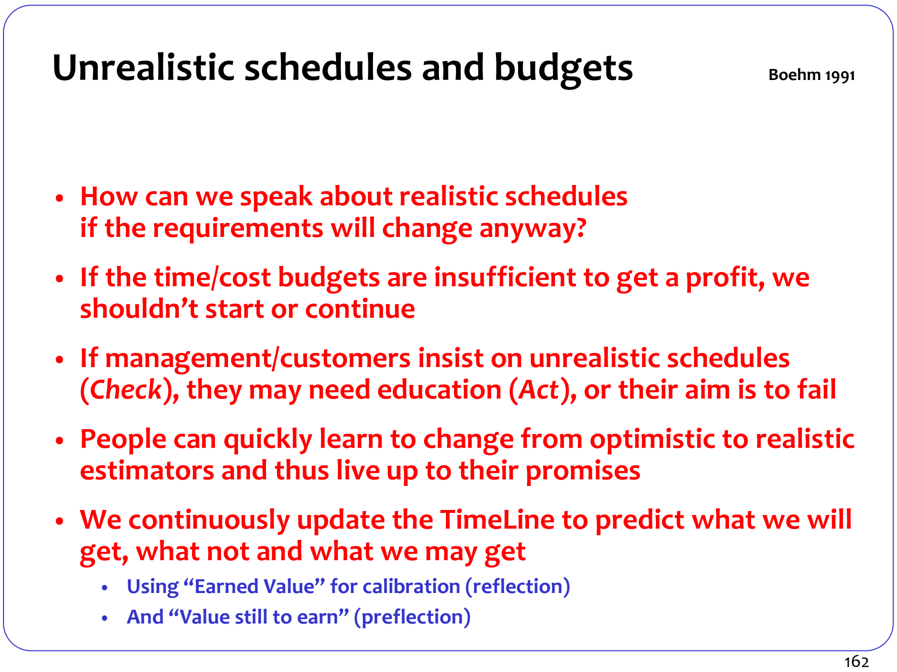
Personnel Shortfalls

Boehm 1991

- There are a certain number of people in the organization
- If we don't get the people we think we need, they are working on more profitable activities
- Using TimeLine, we inform management about the consequences

- This is not risk - it's choice

161

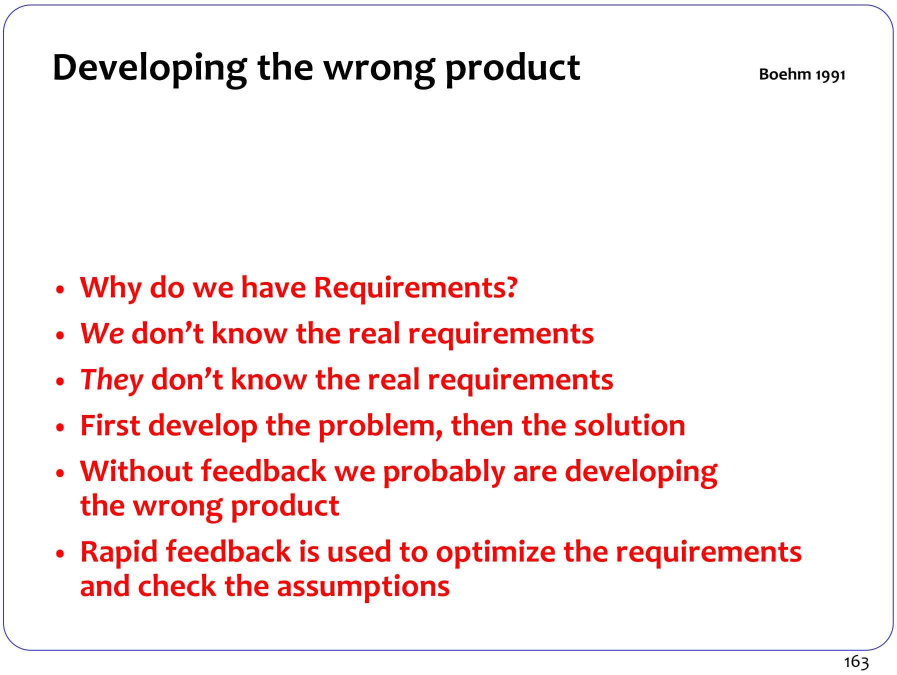


Unrealistic schedules and budgets

Boehm 1991

- How can we speak about realistic schedules if the requirements will change anyway?
- If the time/cost budgets are insufficient to get a profit, we shouldn't start or continue
- If management/customers insist on unrealistic schedules (*Check*), they may need education (*Act*), or their aim is to fail
- People can quickly learn to change from optimistic to realistic estimators and thus live up to their promises
- We continuously update the TimeLine to predict what we will get, what not and what we may get
  - Using "Earned Value" for calibration (reflection)
  - And "Value still to earn" (preflection)

162

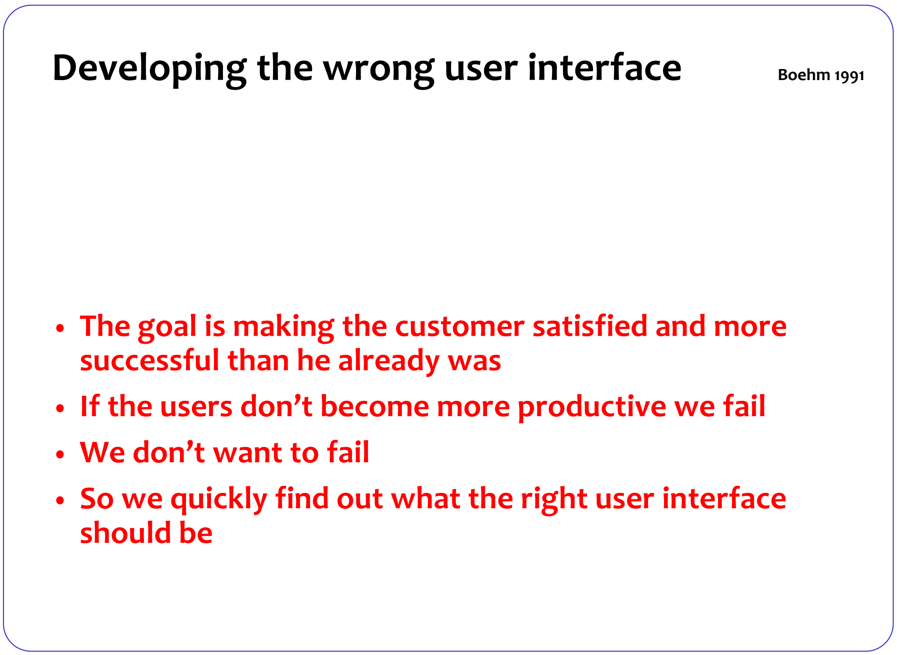


Developing the wrong product

Boehm 1991

- Why do we have Requirements?
- *We* don't know the real requirements
- *They* don't know the real requirements
- First develop the problem, then the solution
- Without feedback we probably are developing the wrong product
- Rapid feedback is used to optimize the requirements and check the assumptions

163

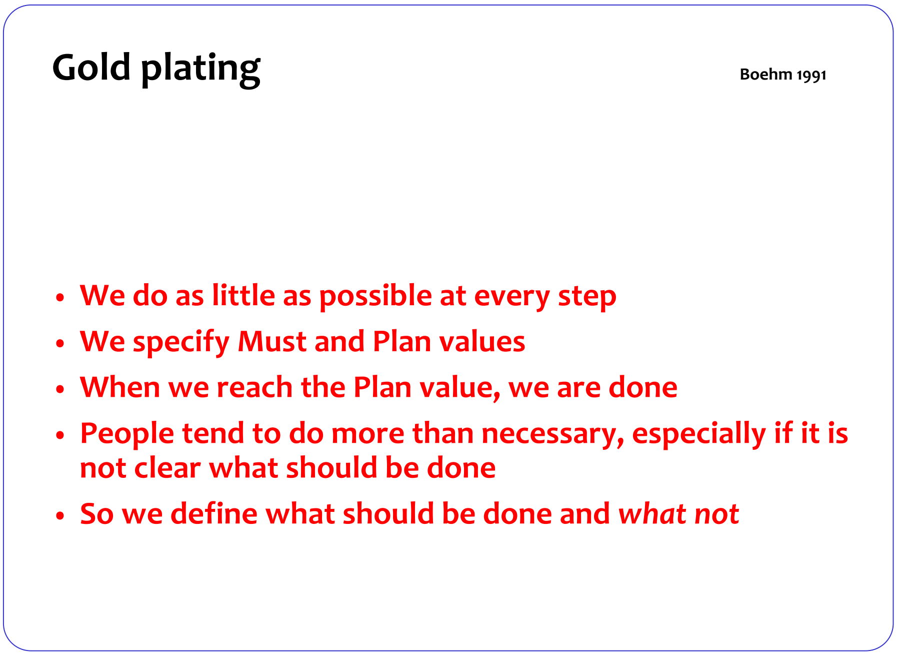


Developing the wrong user interface

Boehm 1991

- The goal is making the customer satisfied and more successful than he already was
- If the users don’t become more productive we fail
- We don’t want to fail
- So we quickly find out what the right user interface should be

164

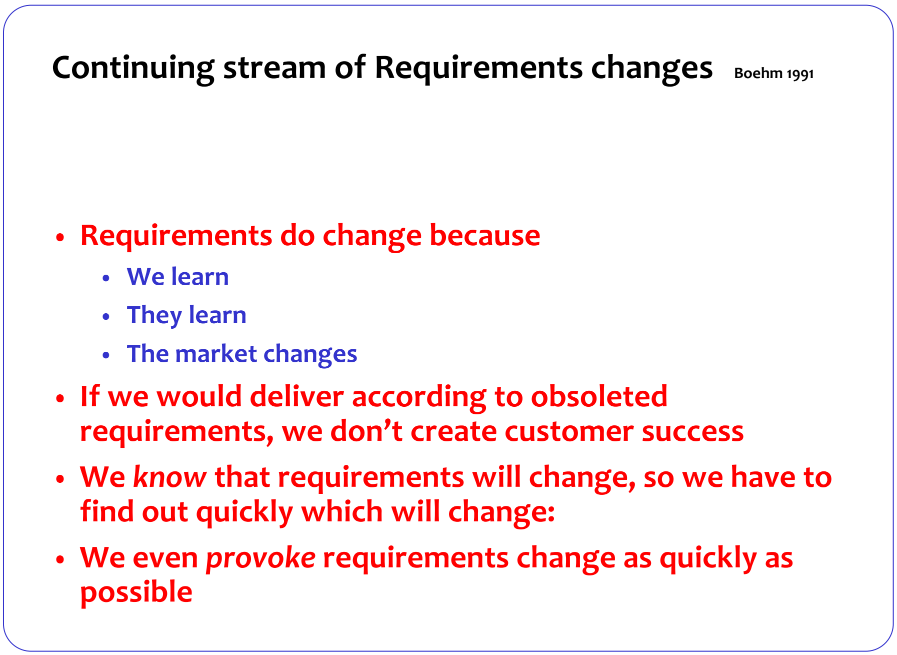


Gold plating

Boehm 1991

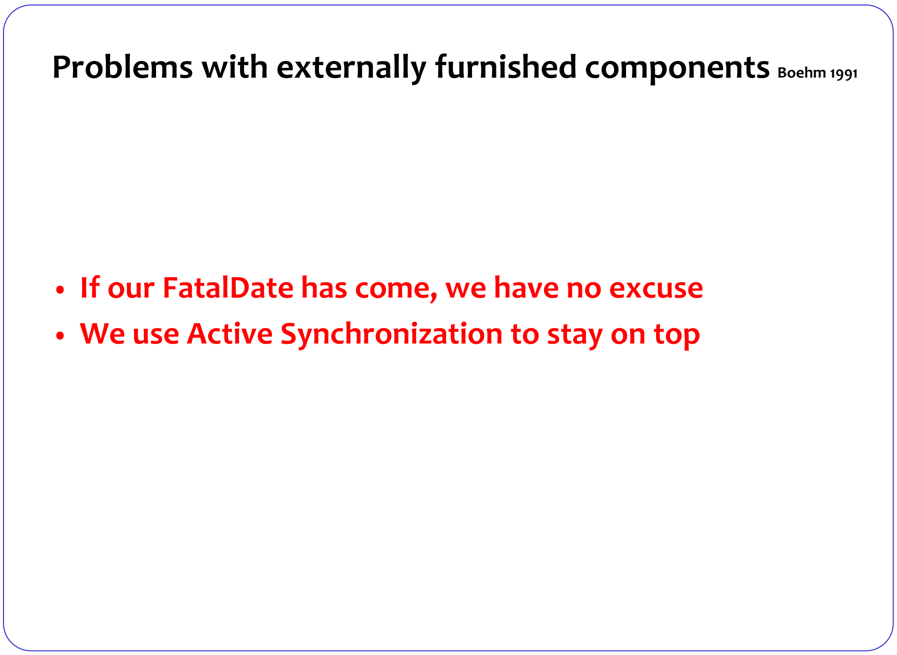
- We do as little as possible at every step
- We specify Must and Plan values
- When we reach the Plan value, we are done
- People tend to do more than necessary, especially if it is not clear what should be done
- So we define what should be done and *what not*

165



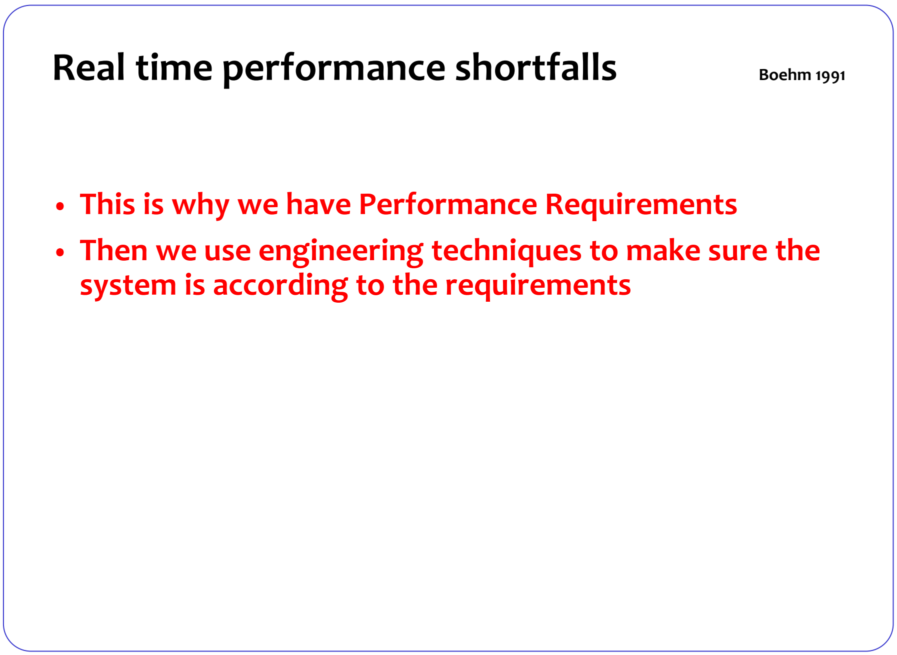
Continuing stream of Requirements changes Boehm 1991

- Requirements do change because
  - We learn
  - They learn
  - The market changes
- If we would deliver according to obsoleted requirements, we don't create customer success
- We *know* that requirements will change, so we have to find out quickly which will change:
- We even *provoke* requirements change as quickly as possible



Problems with externally furnished components Boehm 1991

- If our FatalDate has come, we have no excuse
- We use Active Synchronization to stay on top

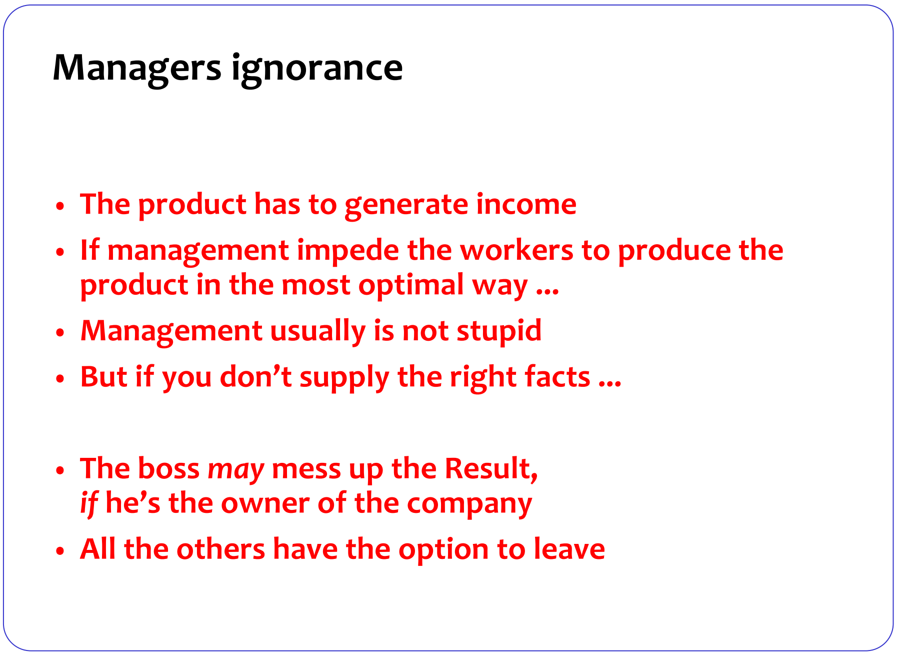


Real time performance shortfalls

Boehm 1991

- This is why we have Performance Requirements
- Then we use engineering techniques to make sure the system is according to the requirements

168



Managers ignorance

- The product has to generate income
- If management impede the workers to produce the product in the most optimal way …
- Management usually is not stupid
- But if you don't supply the right facts …

- The boss *may* mess up the Result,
  if he's the owner of the company
- All the others have the option to leave

169

What is Quality?

I know it when I see it … ?

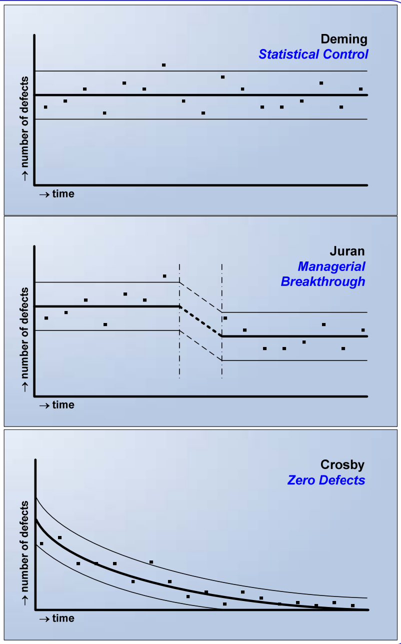
- Should be *measurable*
- Should be *predictable*

But …

ultimately they must like it when they see it

170

Deming - Juran - Crosby

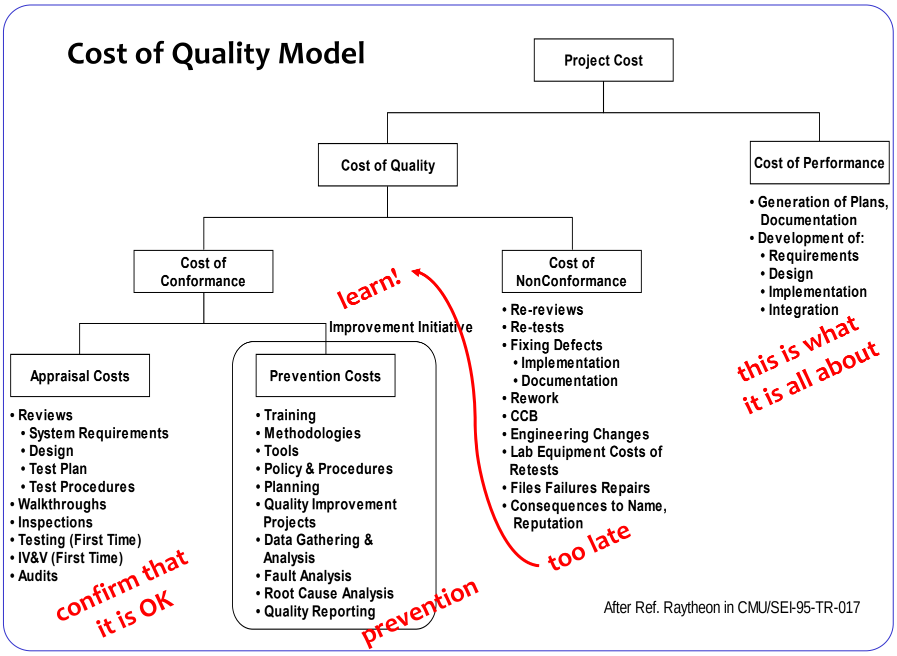


171

Deming

- Quality comes not from inspection (testing), but from improvement of the production process
- Inspection (testing) does not improve quality, nor guarantee quality
- It's too late
- The quality, good or bad, is already in the product
- You cannot inspect (test) quality into a product

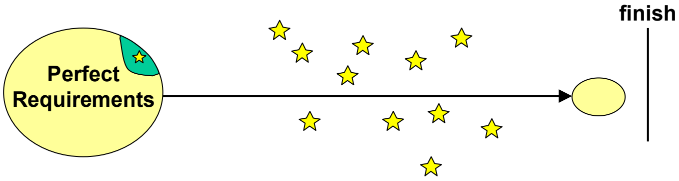
Cost of Quality Model



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

Defects

- A design does not have bugs, it has *defects*
- Defects do not *emerge*
- People make errors and thus cause defects
- Changing a requirement causes a lot of defects



174

Are defects a problem for you?

- Which types of defects?
- How do you know?
- Perhaps there are problems you don’t know?
- What can we do about it?

175

Debugging ? ? ?

176

The process of defect injection and detection

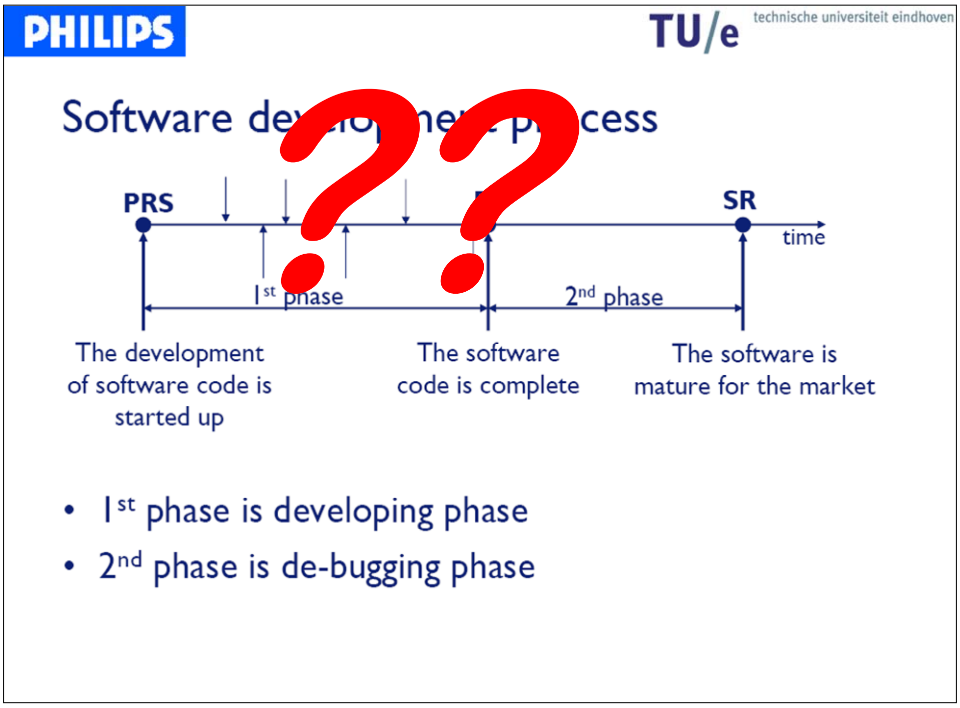
Conventional software development:

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

Can’t we do better, or are we already doing things better?

Real Engineering is
doing (most) things First Time Right

177



Software development process

The development of software code is started up

The software code is complete

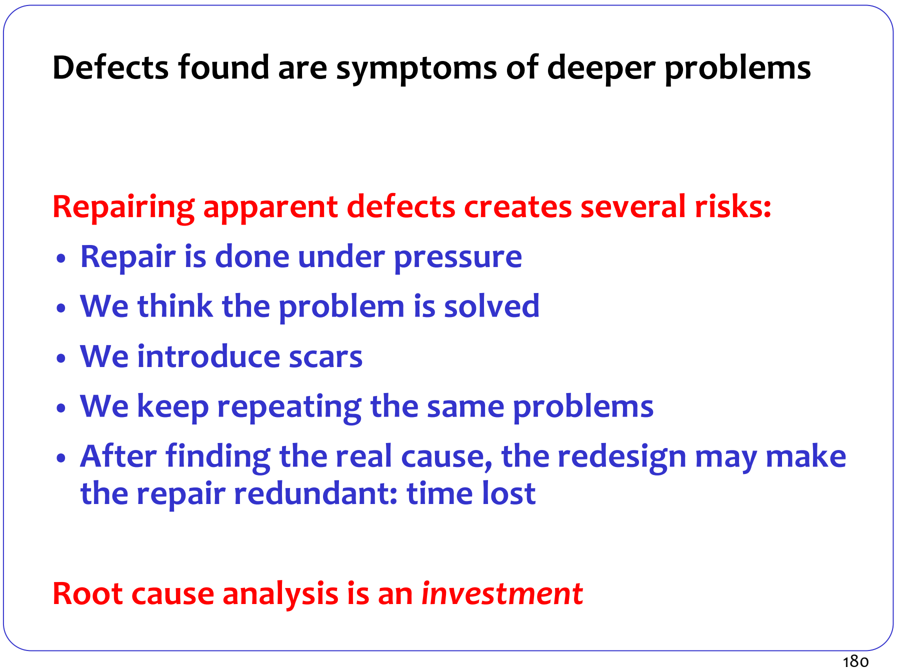
The software is mature for the market

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

Bugs are so important, are they really?

- “Software without bugs is impossible”
- Bugs are counted
- We try to predict the number of bugs we will find
- It is suspect if we don’t find the expected number
- Bugs are normal
- What would we do if there were no bugs any more?

As long as we keep focusing on bugs, there will be bugs



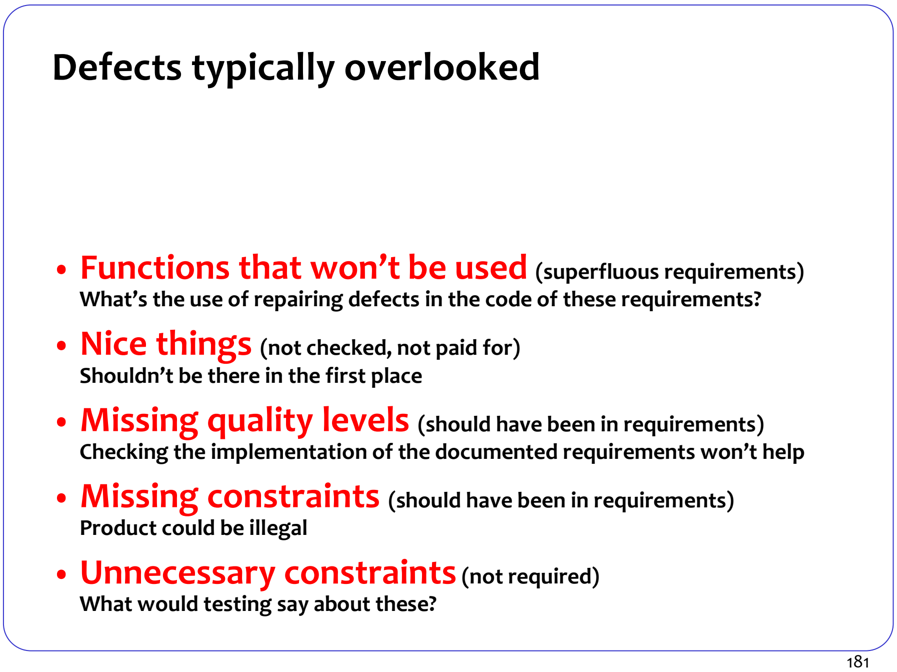
Defects found are symptoms of deeper problems

Repairing apparent defects creates several risks:

- Repair is done under pressure
- We think the problem is solved
- We introduce scars
- We keep repeating the same problems
- After finding the real cause, the redesign may make the repair redundant: time lost

Root cause analysis is an *investment*

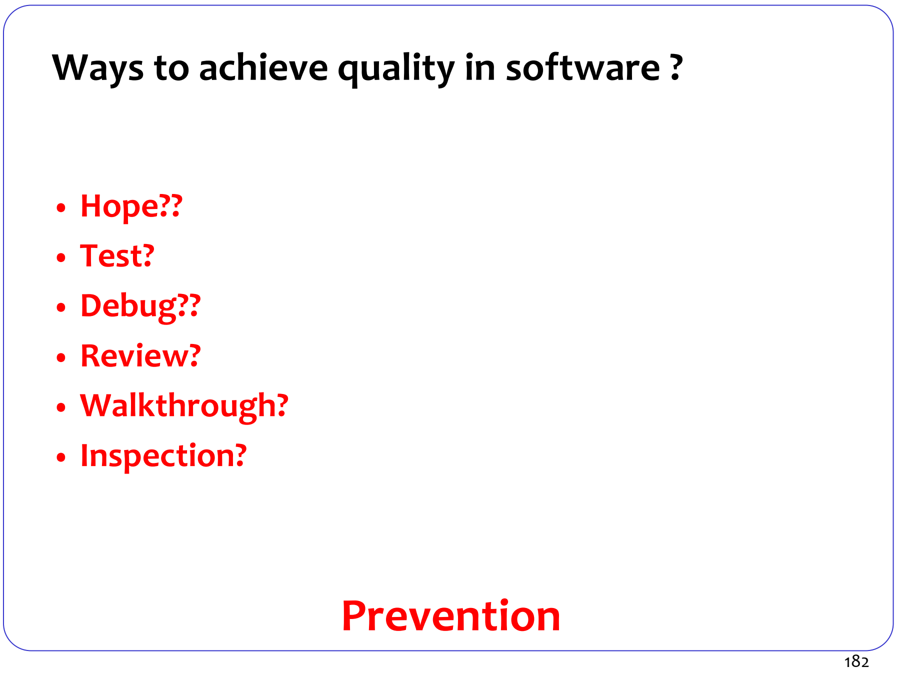
180



Defects typically overlooked

- Functions that won't be used (superfluous requirements)
  What's the use of repairing defects in the code of these requirements?
- 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?

181

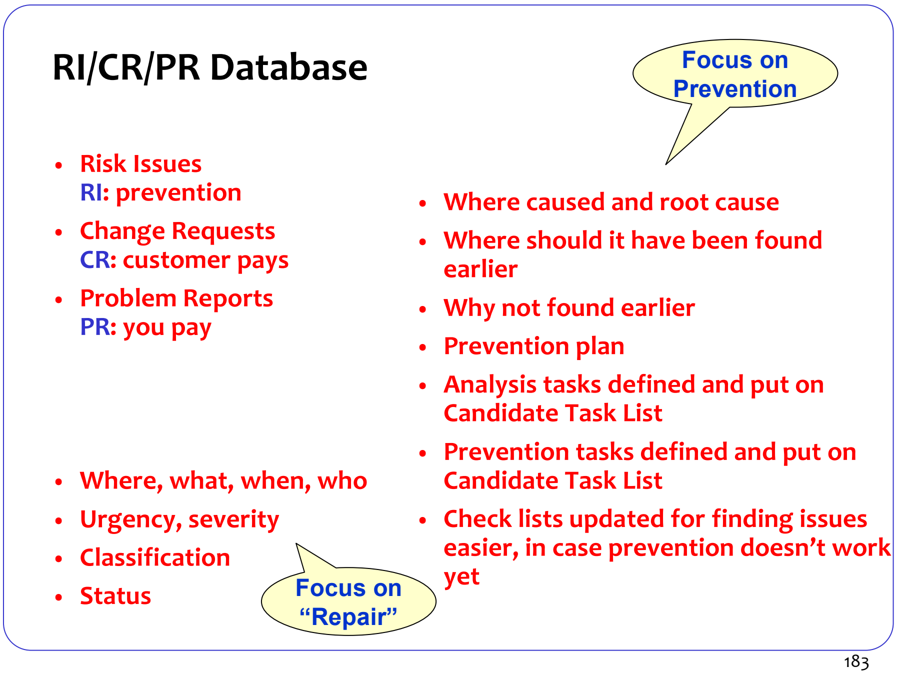


Ways to achieve quality in software ?

- Hope??
- Test?
- Debug??
- Review?
- Walkthrough?
- Inspection?

Prevention

182



RI/CR/PR Database

- Risk Issues
  RI: prevention
- Change Requests
  CR: customer pays
- Problem Reports
  PR: you pay

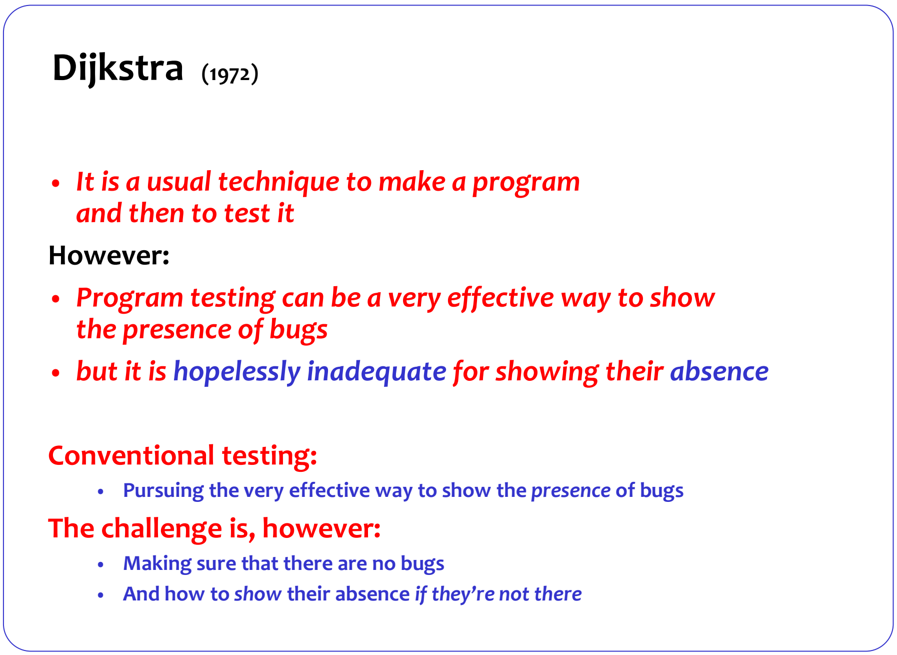
- Where, what, when, who
- Urgency, severity
- Classification
- Status

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

183



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*

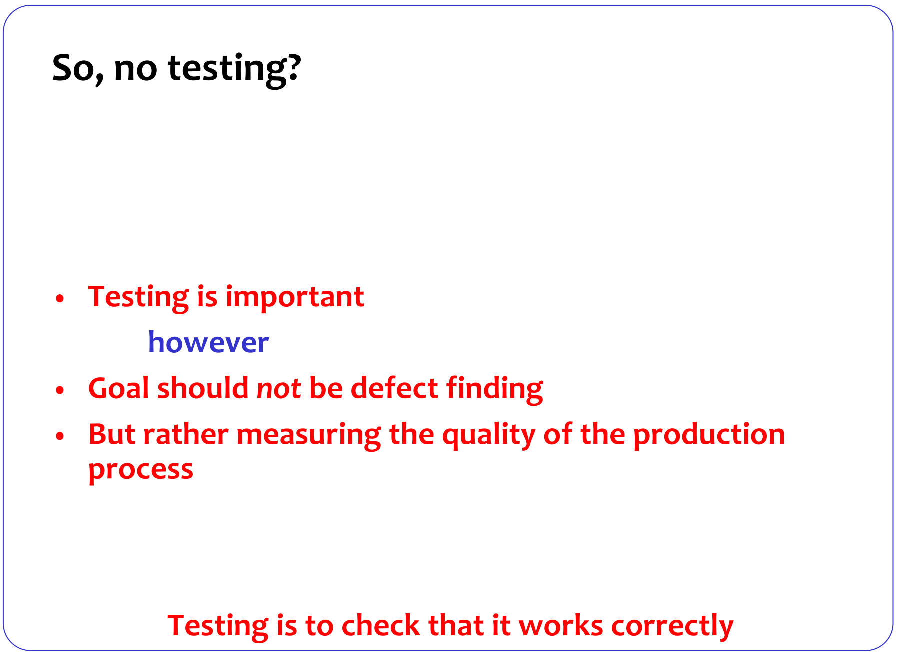
Conventional testing:

- Pursuing the very effective way to show the *presence* of bugs

The challenge is, however:

- Making sure that there are no bugs
- And how to *show* their absence *if they're not there*

184



So, no testing?

- **Testing is important**

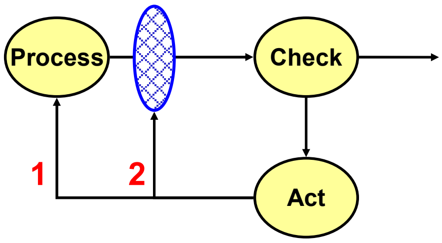
  however

- **Goal should *not* be defect finding**
- **But rather measuring the quality of the production process**

Testing is to check that it works correctly

185

Testing is checking correctness



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

186

Let’s move

Let’s move from

- Fixation to Fix

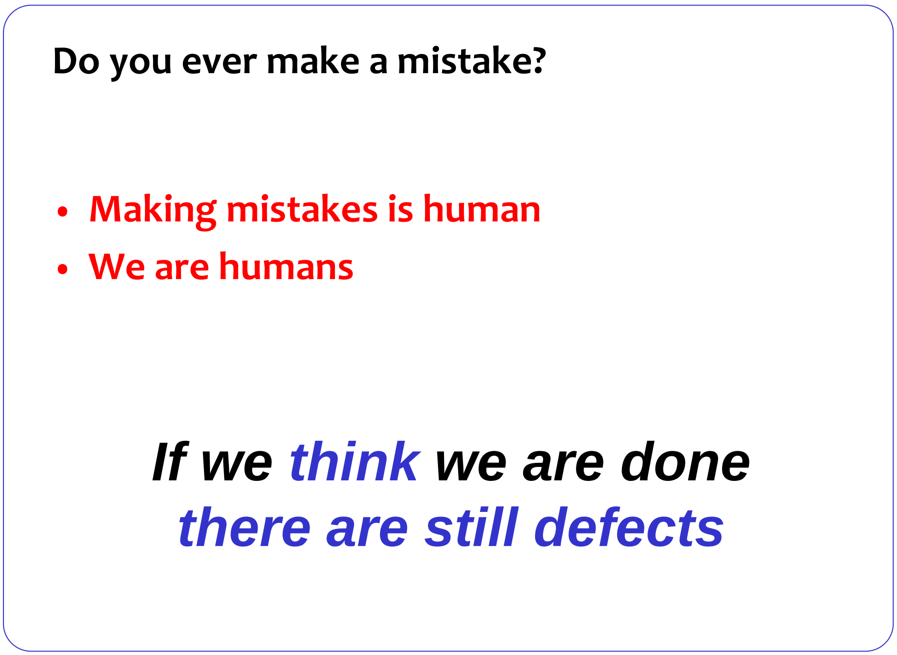
to

- Attention to Prevention

- If we don’t deal with the root, we will keep making the same mistakes over and over

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

187

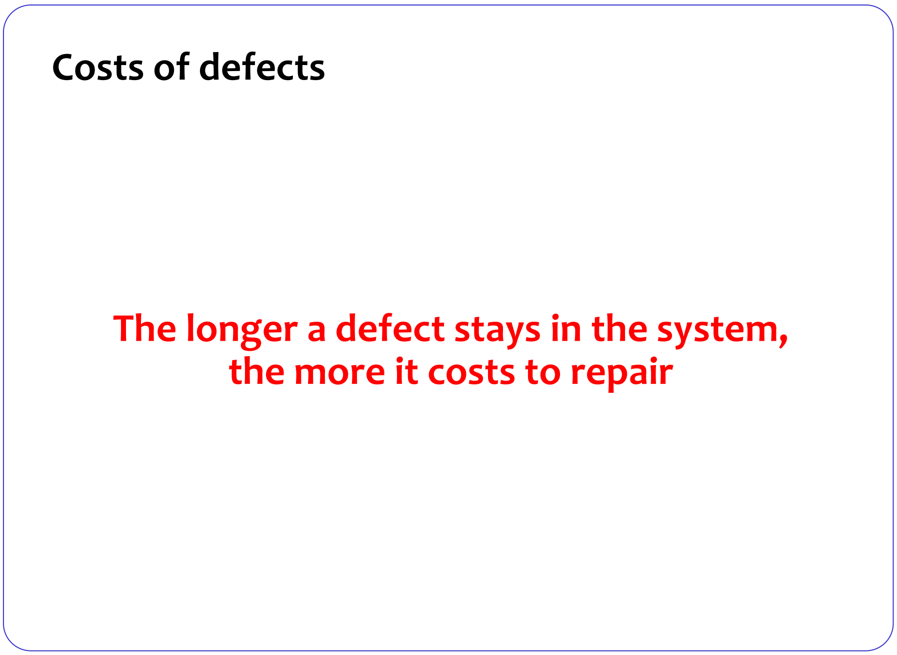


Do you ever make a mistake?

- Making mistakes is human
- We are humans

If we think we are done
there are still defects

188

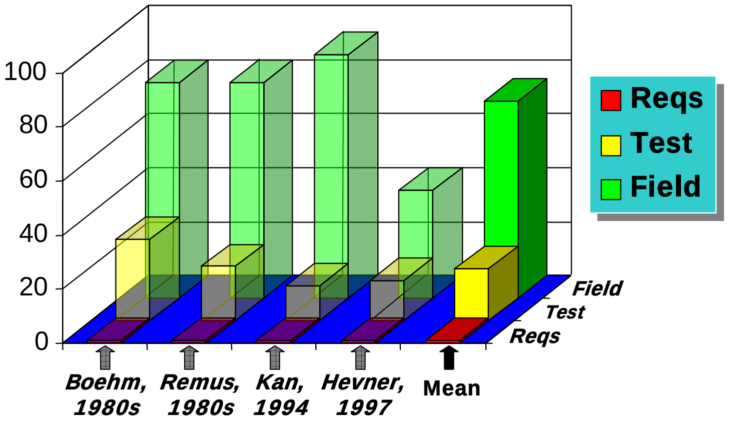


Costs of defects

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

189

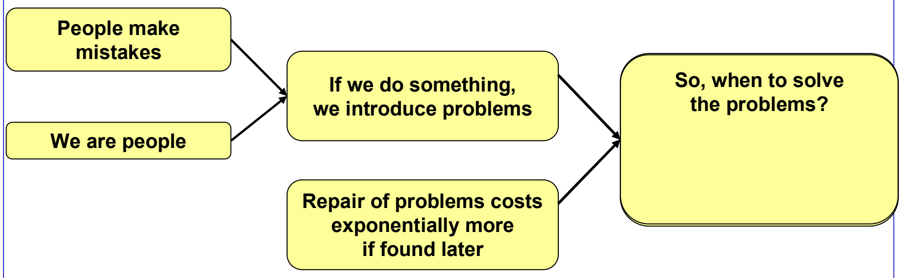
Cost of Requirements Defects



190

DM

Inevitable consequence


People make mistakes

We are people

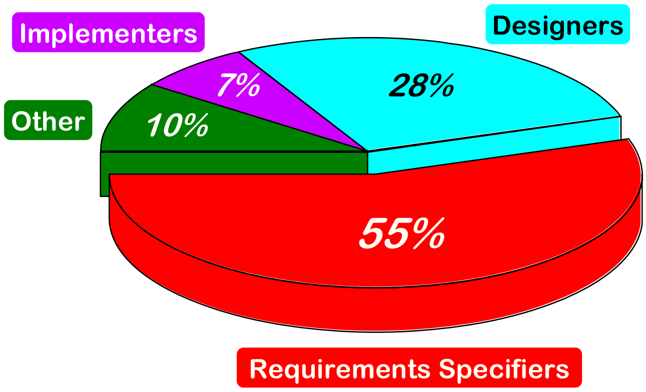
If we do something, we introduce problems

Repair of problems costs exponentially more if found later

So, when to solve the problems?

191

Typical Defect Injectors (*cost* breakdown)



After Bender Associates, *1996*

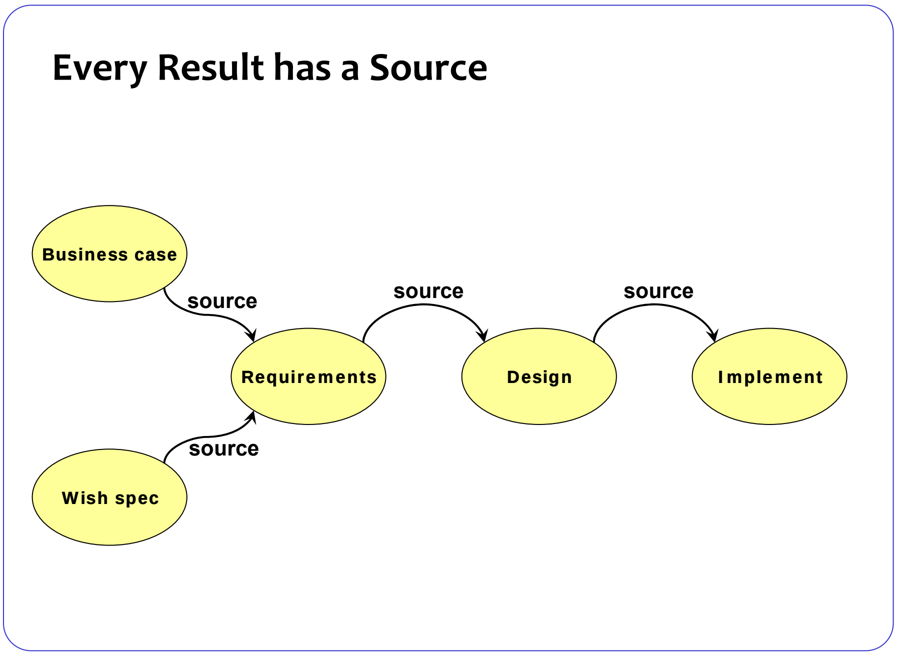
192

DM

Documentation

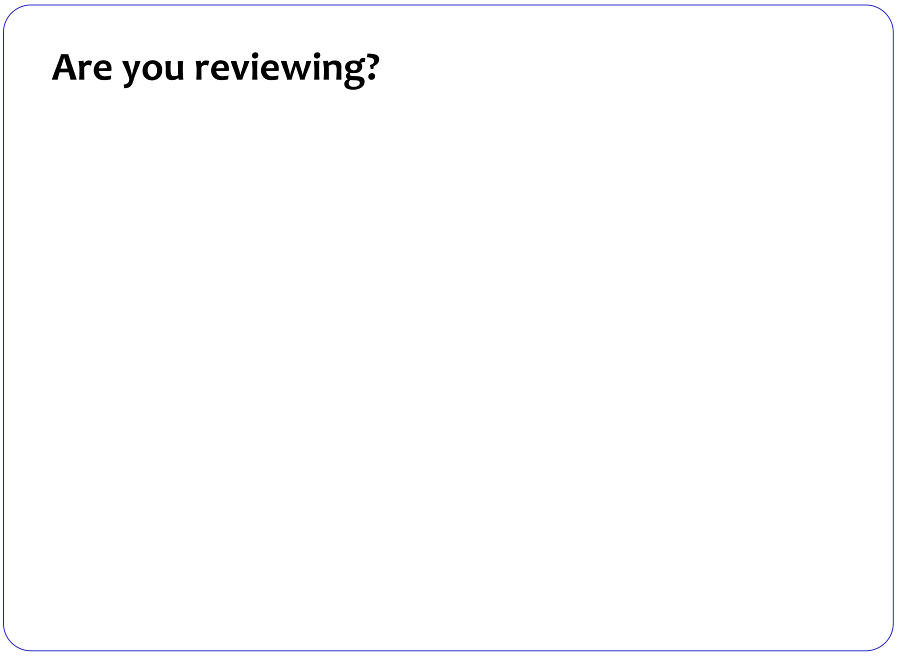
- **Wish specification** Thank you, nice input
- **Business Case** Why we are doing it
- **Requirements** What the project agrees to satisfy
- **DesignLog** Selecting the 'optimum' compromise
- **Specification** This is how we are going to implement it
- **Implementation** Code, schematics, hardware, documentation, training
- **Process Log** Describing how and why you arrived at which current practices

193



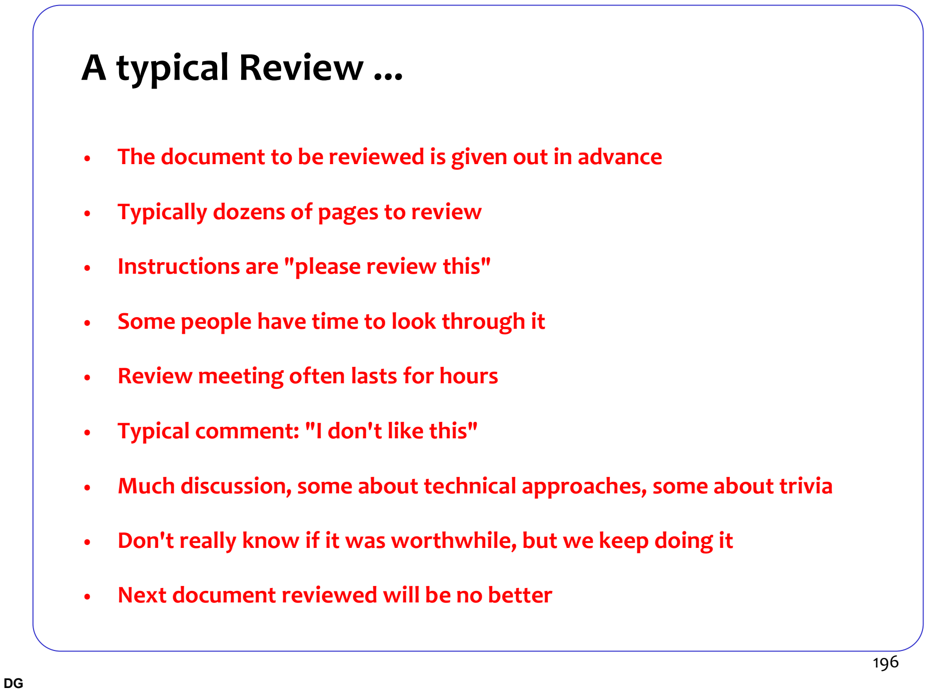
Every Result has a Source

194



Are you reviewing?

195

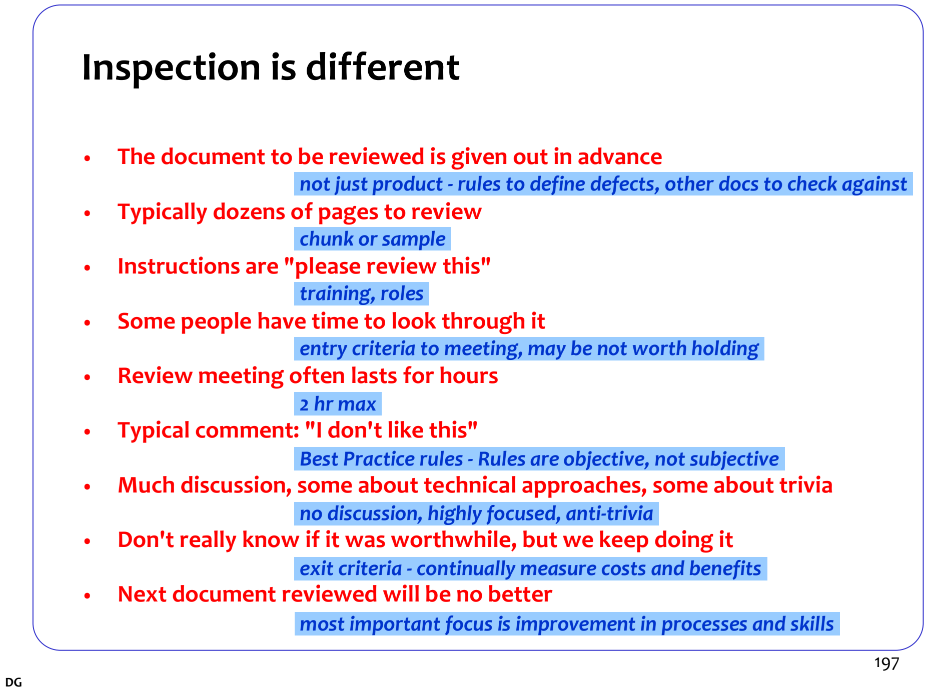


A typical Review ...

- 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

196

DG



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

197

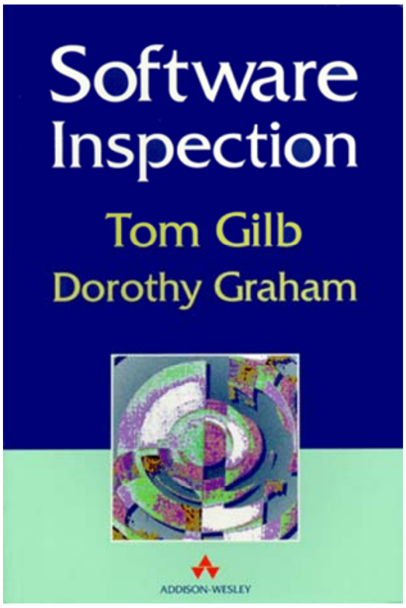
DG

Inspection

- Most rigorous form of review
- Pioneered by Fagan (IBM) (paper 1976)
  - Locating all the defects in a work product
- Introduction of Inspection economics: Gilb/Graham (Software Inspection, 1993)
  - Quantifying the defect density of a work product and preventing poor quality work from moving downstream
- 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 like to base further work or decisions on a document of unknown quality?

198



A ready to use recipe …

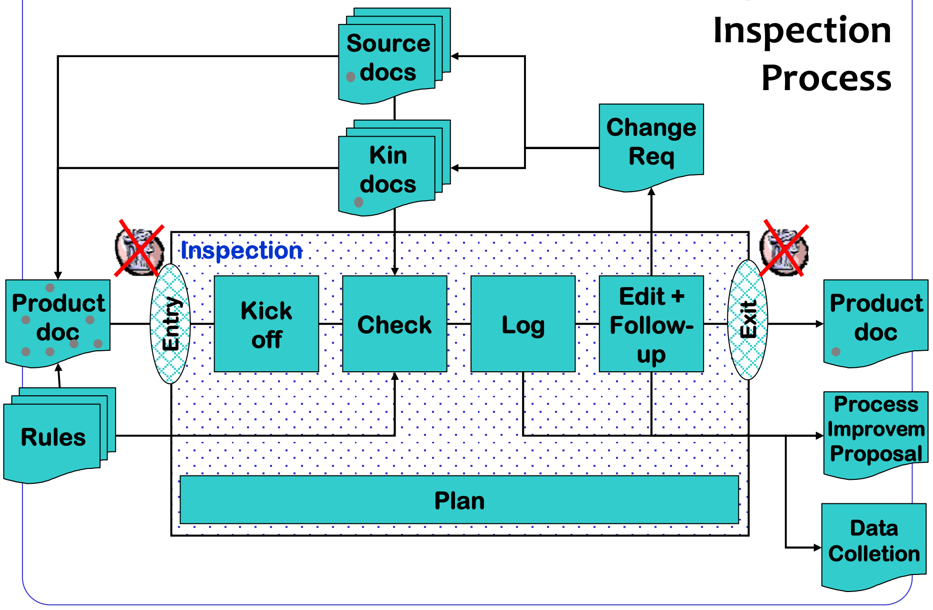
199

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 generate documents in the first place?
- Interesting side-effect:
  People get to know each others documents efficiently

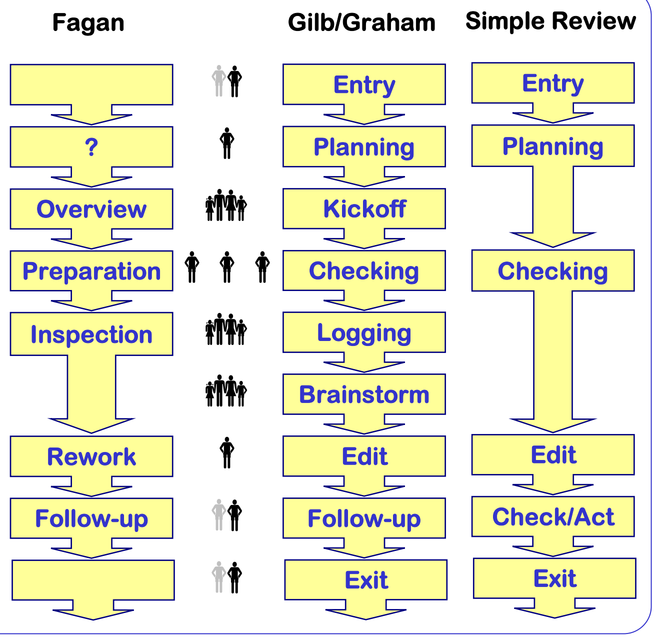
200

Gilb/Graham Inspection Process



201

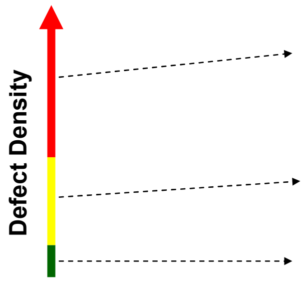
Inspection Process Steps



202

Gilb/Graham Concepts
Entry and Exit Criteria

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



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

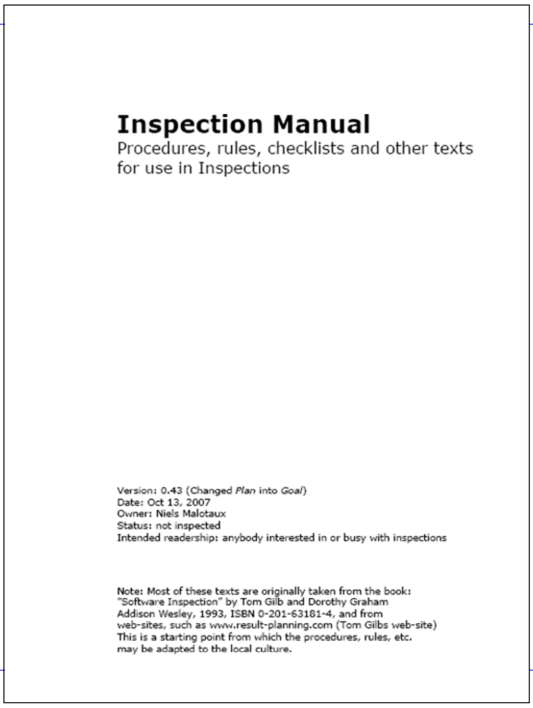
Somewhat above exit criteria: Rework or enlarge inspection sample

Meets exit criteria: Success! Exit

203

ES

16 page Inspection Manual



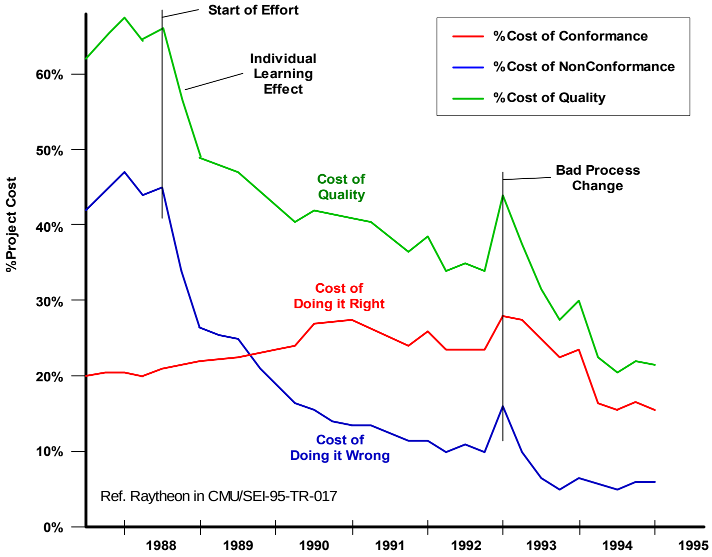
Inspection Manual
Procedures, rules, checklists and other texts
for use in Inspections

Version: 0.43 (Changed *Plan* into *Goal*)
Date: Oct 13, 2007
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.result-planning.com (Tom Gilbs web-site)
This is a starting point from which the procedures, rules, etc.
may be adapted to the local culture.

204

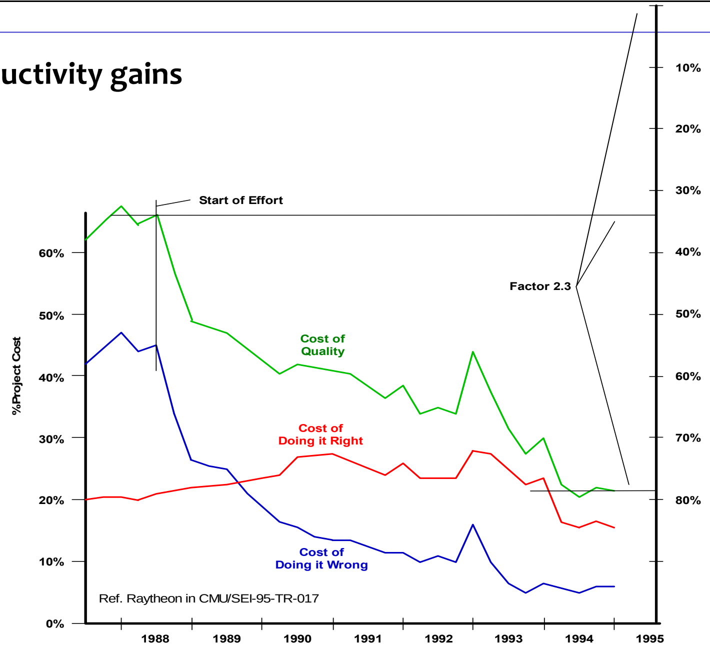
Cost of Quality



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

205

Productivity gains

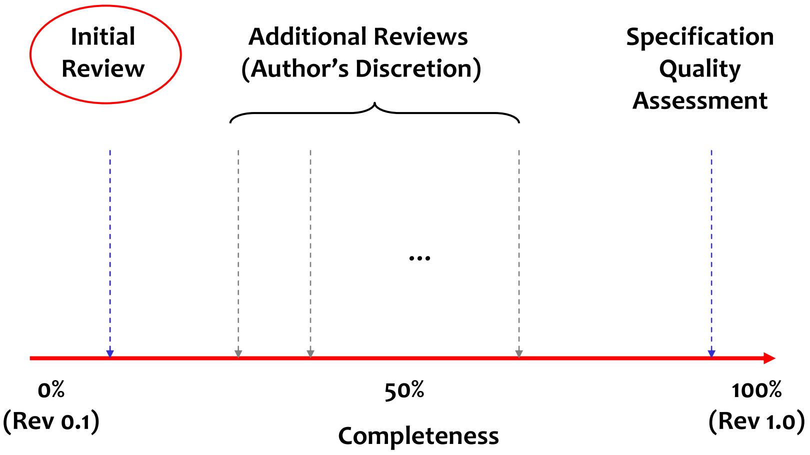


Ref. Raytheon in CMU/SEI-95-TR-017

206

Agile Inspection / Early Inspection

Prevention costs less than Repair



Initial Review

Additional Reviews (Author’s Discretion)

Specification Quality Assessment

…

0% (Rev 0.1)

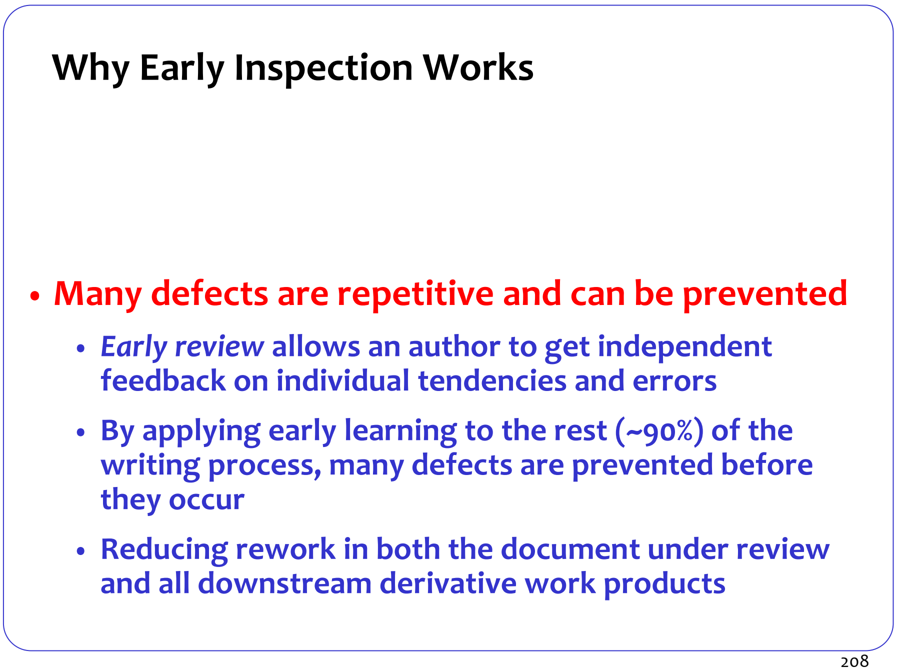
50%

Completeness

100% (Rev 1.0)

207

ES

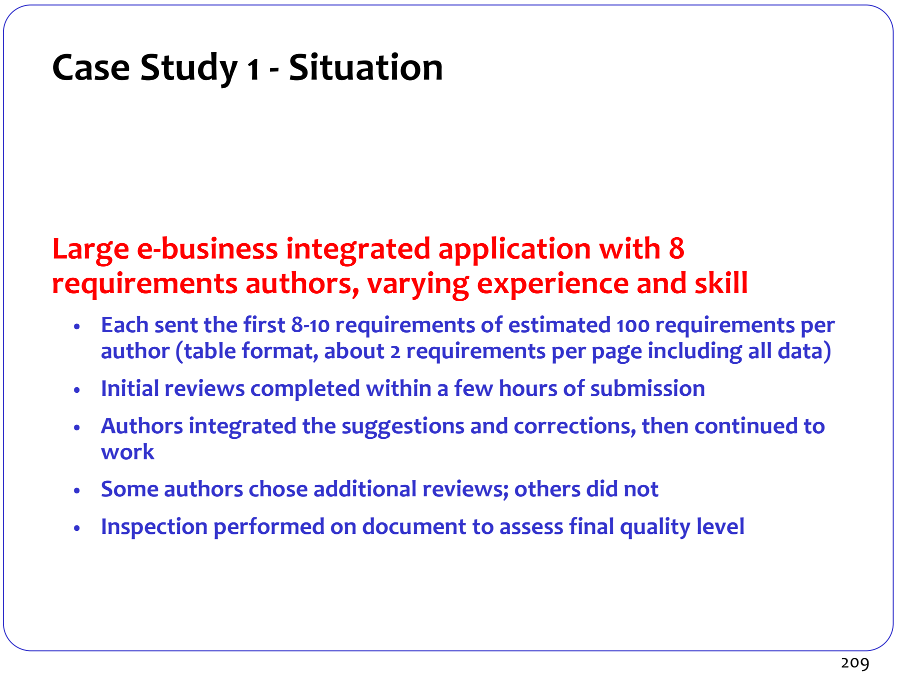


Why Early Inspection Works

- 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 are prevented before they occur
  - Reducing rework in both the document under review and all downstream derivative work products

208

ES



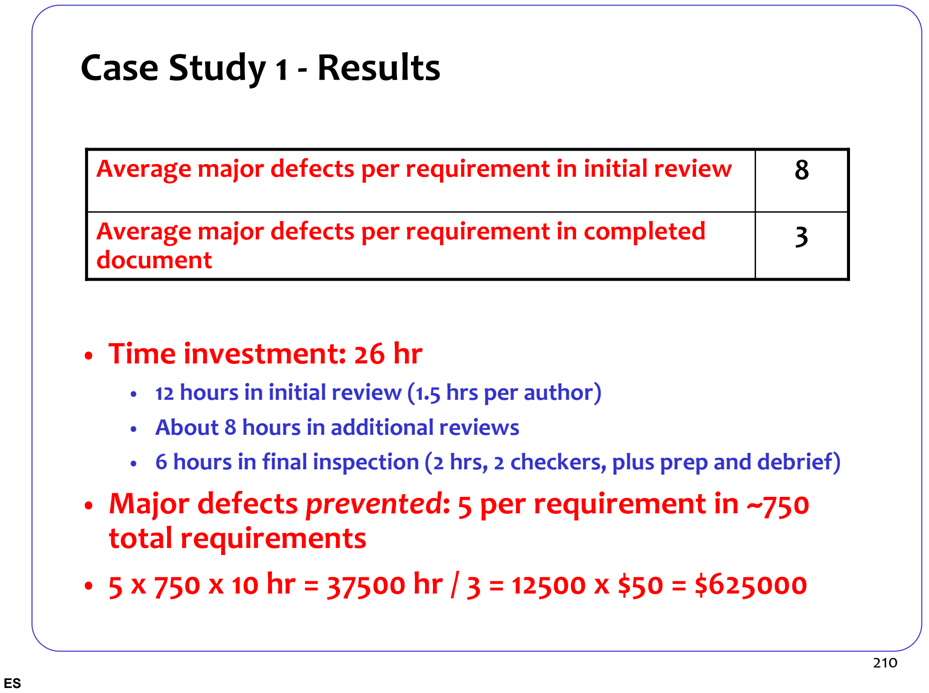
Case Study 1 - Situation

Large e-business integrated application with 8 requirements authors, varying experience and skill

- Each sent the first 8-10 requirements of estimated 100 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

209

ES



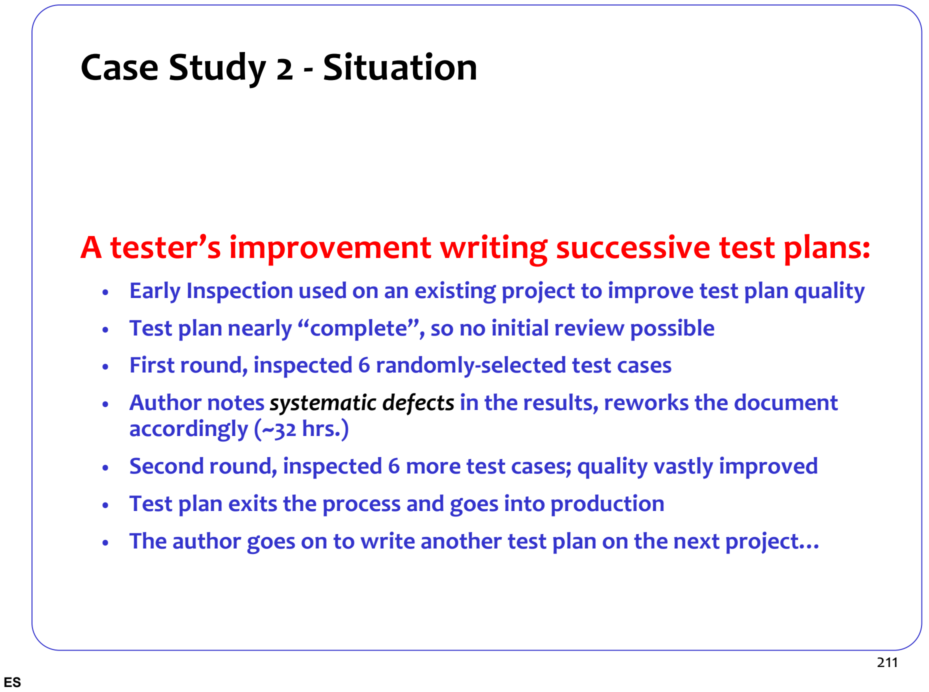
Case Study 1 - Results

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 requirements
- 5 x 750 x 10 hr = 37500 hr / 3 = 12500 x $50 = $625000

210

ES



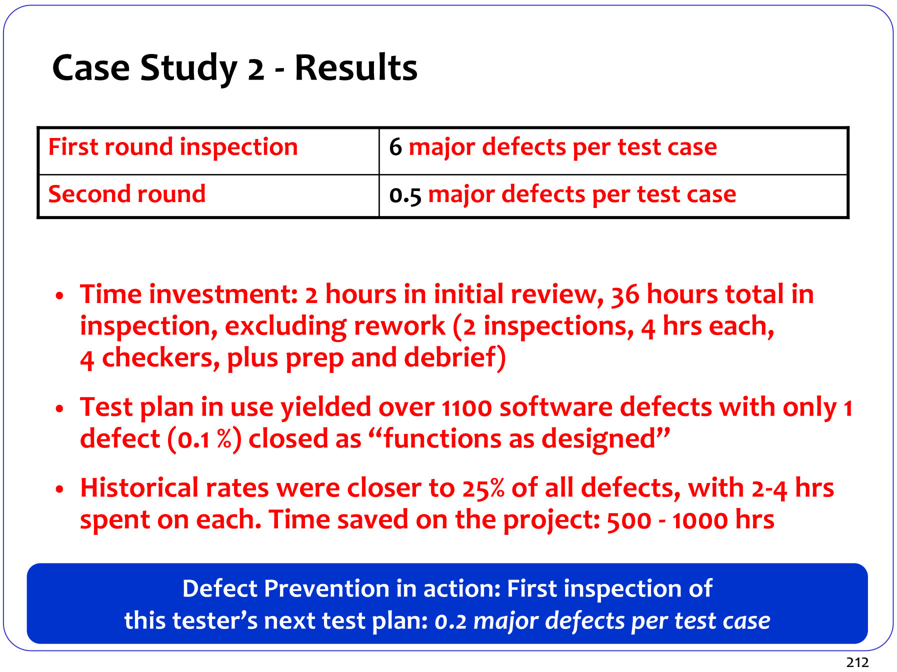
Case Study 2 - Situation

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 no initial review possible
- 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…

211

ES



Case Study 2 - Results

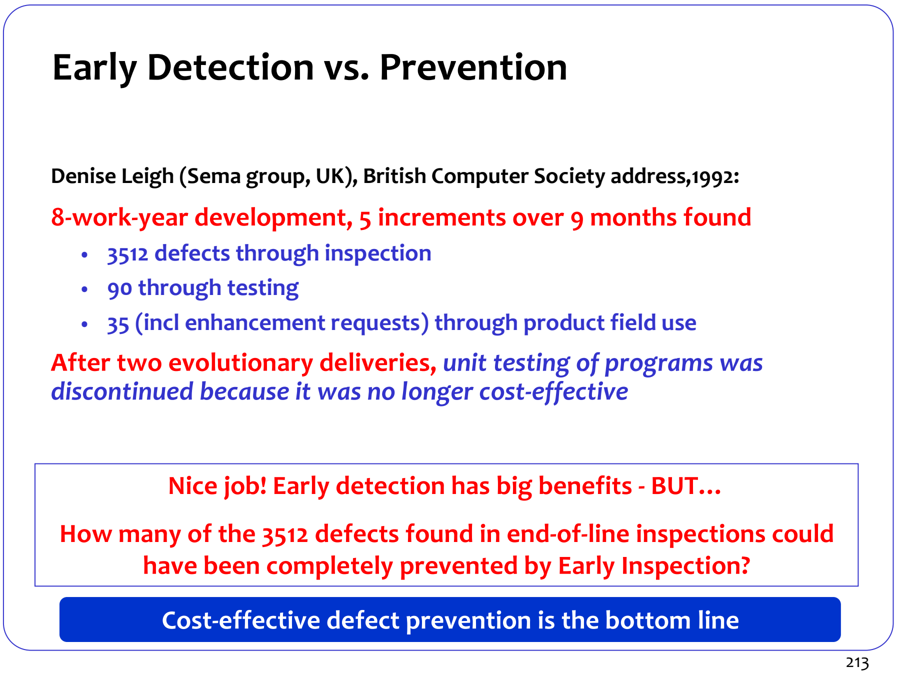
First round inspection	6 major defects per test case
Second round	0.5 major defects per test case

- Time investment: 2 hours in initial review, 36 hours total in inspection, excluding rework (2 inspections, 4 hrs each, 4 checkers, plus prep and debrief)
- Test plan in use yielded over 1100 software defects with only 1 defect (0.1 %) closed as "functions as designed"
- Historical rates were closer to 25% of all defects, with 2-4 hrs spent on each. Time saved on the project: 500 - 1000 hrs

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

212

ES



Early Detection vs. Prevention

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

8-work-year development, 5 increments over 9 months found

- 3512 defects through inspection
- 90 through testing
- 35 (incl 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

213

ES

Optimum Checking Rate

- 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")

214

TG

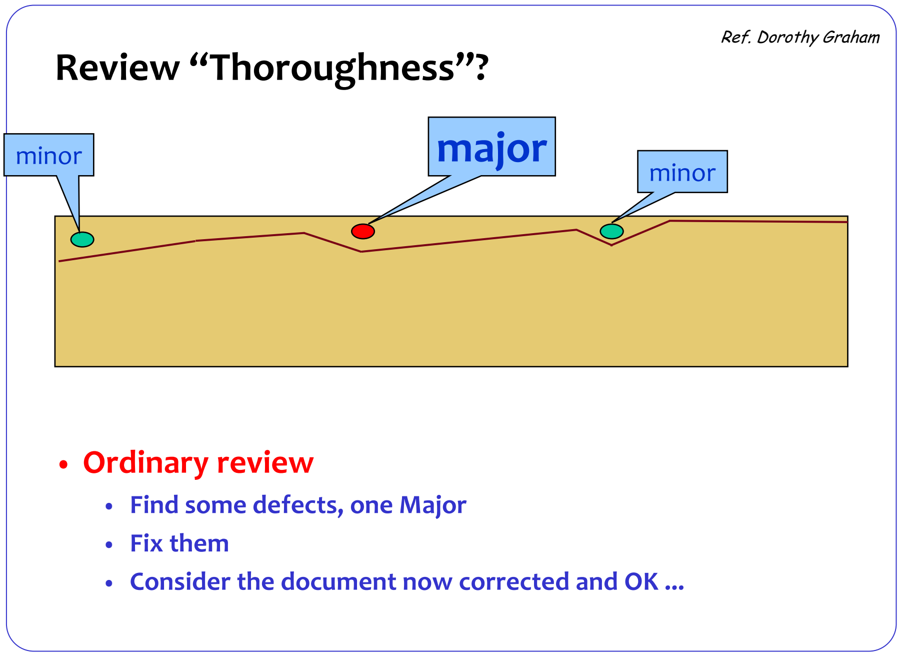
Ref. Dorothy Graham

Optimum checking rate

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

215

DG



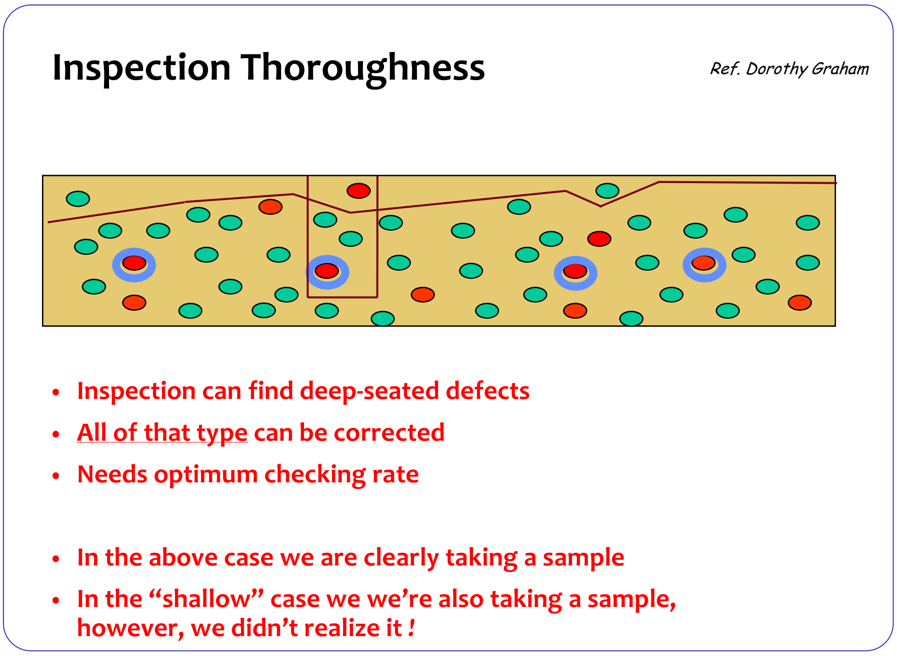
Ref. Dorothy Graham

Review “Thoroughness”?

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

216

DG



Inspection Thoroughness

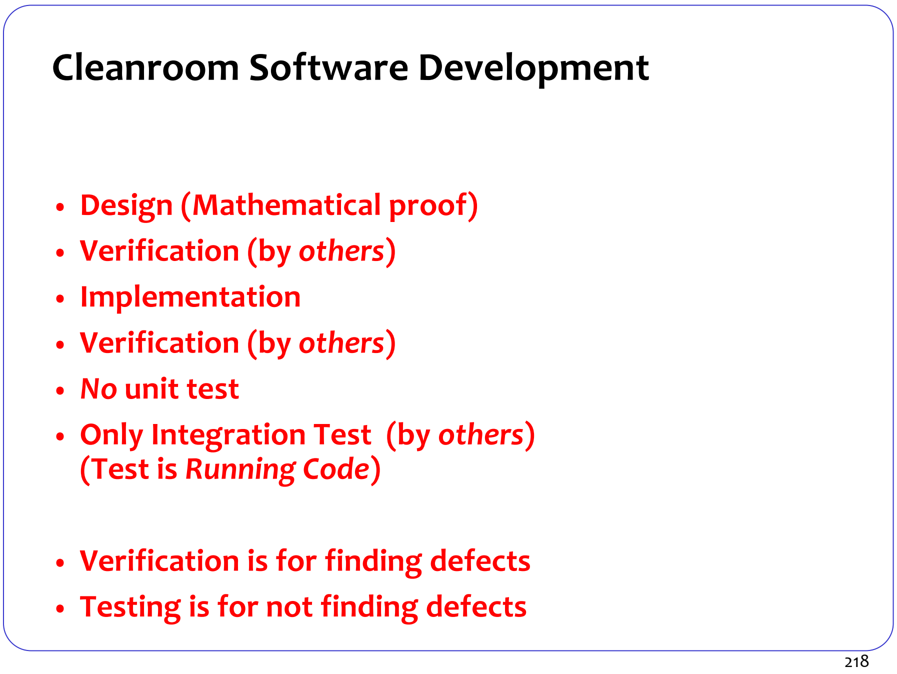
Ref. Dorothy Graham

- 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 we’re also taking a sample, however, we didn’t realize it !

217

DG

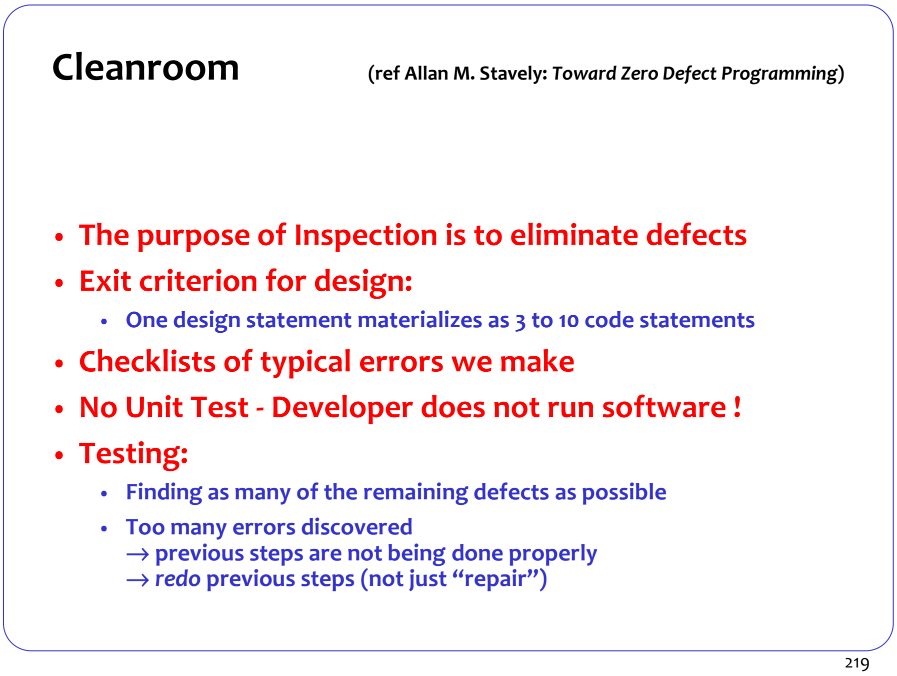


Cleanroom Software Development

- 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

218

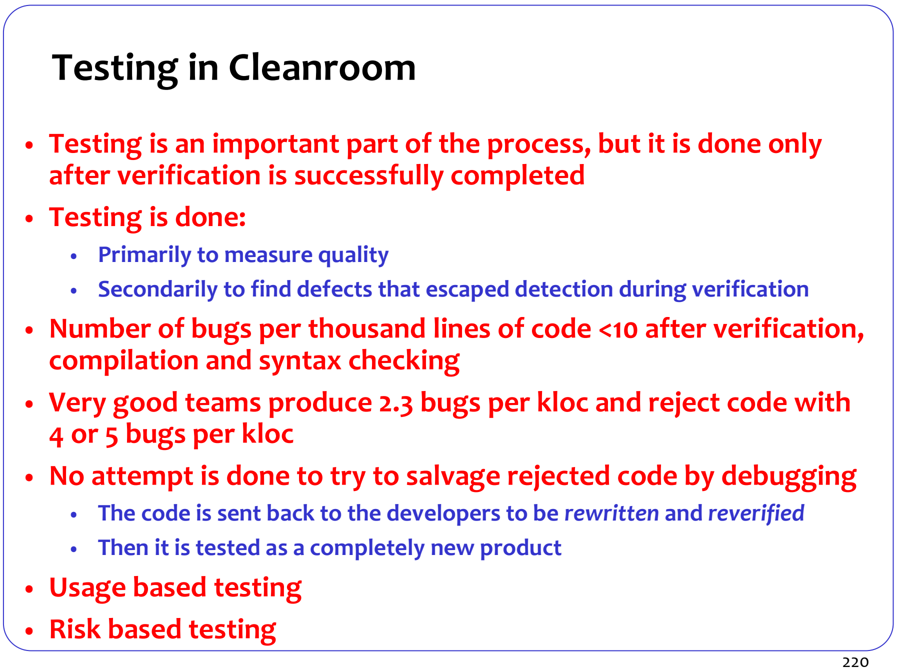


Cleanroom

(ref Allan M. Stavely: *Toward Zero Defect Programming*)

- 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
- No Unit Test - Developer does not run 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 (not just "repair")

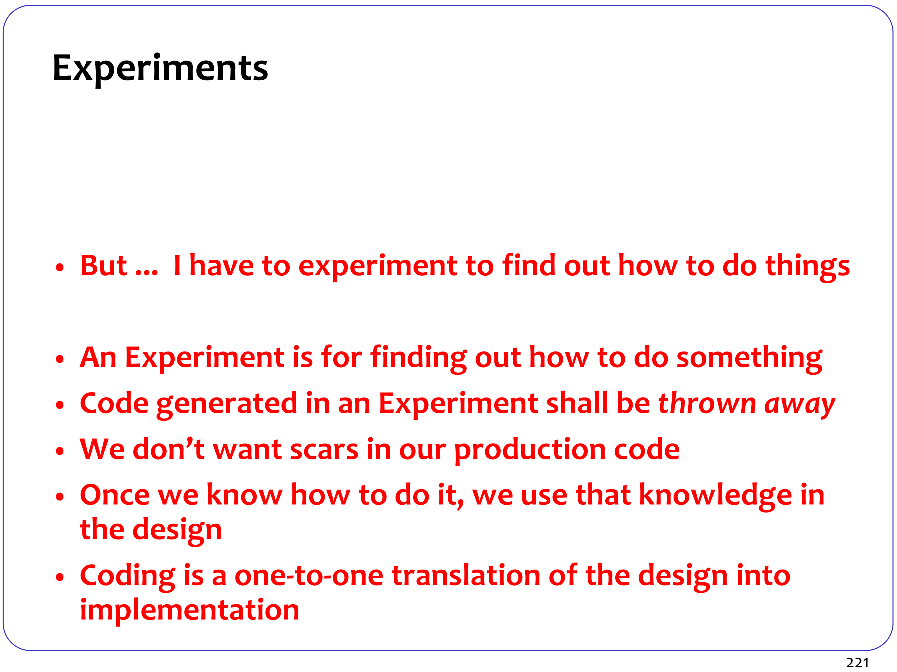
219



Testing in Cleanroom

- Testing is an important part of the process, but it is done only after verification 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 bugs per kloc and reject code with 4 or 5 bugs 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
- Risk based testing

220



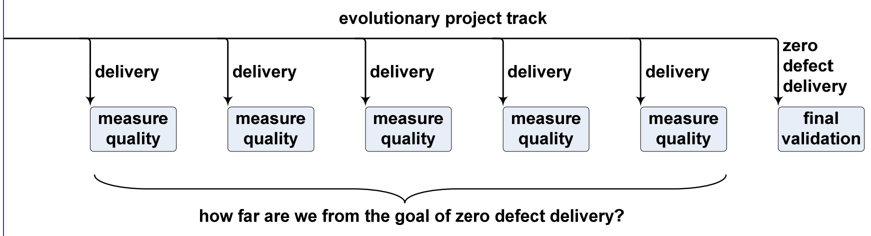
Experiments

- But … I have to experiment to find out how to do things

- An Experiment is for finding out how to do something
- Code generated in an Experiment shall be *thrown away*
- We don't want scars in our production code
- Once we know how to do it, we use that knowledge in the design
- Coding is a one-to-one translation of the design into implementation

221

Evo Testing

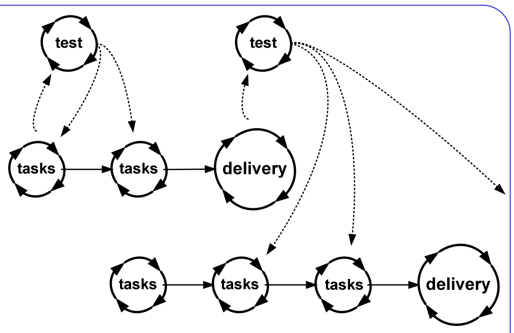


- Final validation shouldn't find any problems
- Earlier verifications mirror quality level to developers: how far from goal and what still to learn
- Evo has *no debugging phase!*

Further Improvement

- Testers focus on a clear goal
- Finding defects is not the goal
- The Goal is Project Success
- Tester's customer is "the developers"
- Testers select and use any method appropriate
- Testers check work in progress *even before* it is finished
- Testers solve the Review and Inspection organizing problem
- Testing is organized the Evo way, entangling intimately with the development process

Evo cycles for Testing



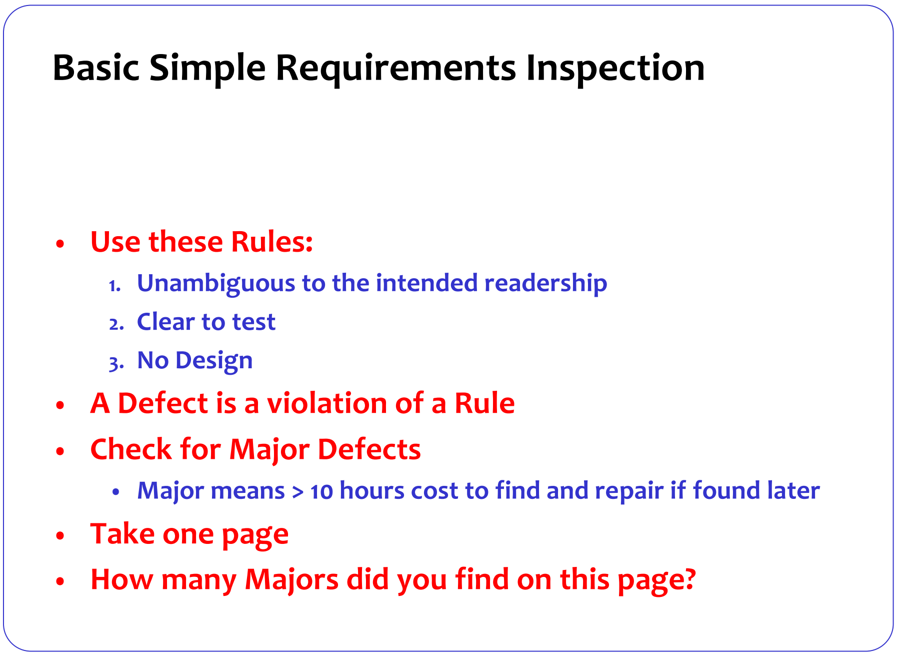
- Testers organize their work in weekly TaskCycles
- DeliveryCycle is the Test-Feedback cycle
- Testers use their own TimeLine, synchronized with the developers TimeLine
- Testers conclude their work in sync with developers
- Testers check work in progress *even before* it is finished

224

Succesvol Plannen van Softwareprojecten

- Business case: waarom doen we het: doen we het juiste
- Requirements: wat doen we daartoe in *dit* project
- Design: wat is de beste oplossing
- Implementatie: uitvoeren van de beste oplossing
- Review & Inspectie: voeden van het preventieproces
- PDCA: continu verbeteren: product, project en proces
- Risico: vaak geen echt risico
- Wekelijkse TaakCyclus: organiseren van het werk
- Tweewekelijkse DeliveryCyclus: zijn we op de juiste weg
- TimeLine: beheersen en optimaliseren van tijdbesteding
- Zero Defects houding
- 5 x Waarom?

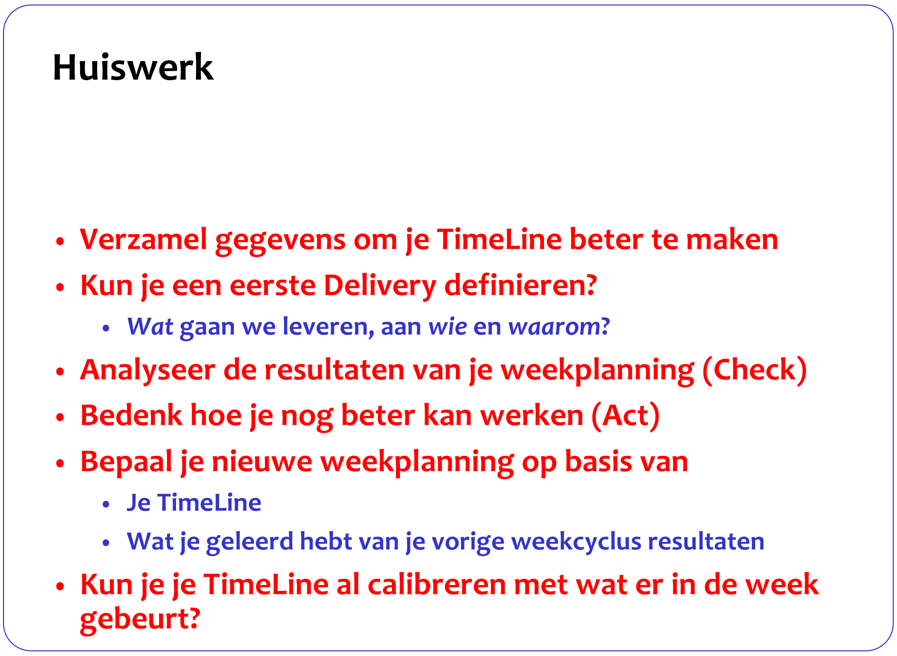
225



Basic Simple Requirements Inspection

- Use these Rules:
  1. Unambiguous to the intended readership
  2. Clear to test
  3. No Design
- A Defect is a violation of a Rule
- Check for Major Defects
  - Major means > 10 hours cost to find and repair if found later
- Take one page
- How many Majors did you find on this page?

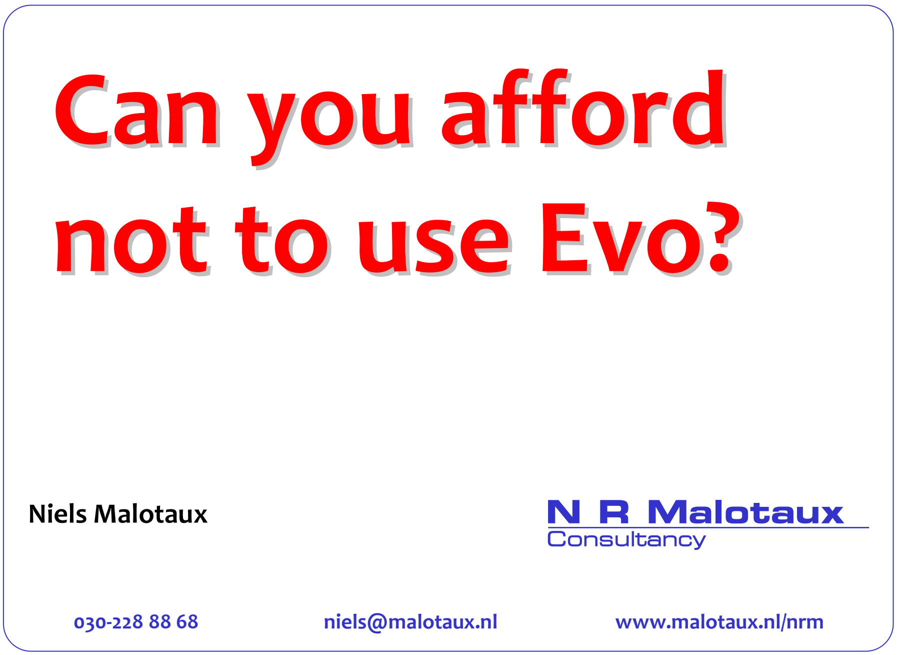
226



Huiswerk

- Verzamel gegevens om je TimeLine beter te maken
- Kun je een eerste Delivery definieren?
  - *Wat* gaan we leveren, aan *wie* en *waarom*?
- Analyseer de resultaten van je weekplanning (Check)
- Bedenk hoe je nog beter kan werken (Act)
- Bepaal je nieuwe weekplanning op basis van
  - Je TimeLine
  - Wat je geleerd hebt van je vorige weekcyclus resultaten
- Kun je je TimeLine al calibreren met wat er in de week gebeurt?

227



Can you afford not to use Evo?

Niels Malotaux

N R Malotaux
Consultancy

030-228 88 68 niels@malotaux.nl www.malotaux.nl/nrm

228

Task Cycle Plan	
Name	
Taskcycle	**Available plannable hrs**

project	delivery due	task due	hrs	tsk sht	done	description

TaskSheet for week	**Assigned to**	**Estimated duration**

Task description

☑ **Requirements for this task to be used as reference for verification**

- **Functions (what should it do?)**

- **Qualities (how well should it do it)** State definitions of e.g. "usability", "user-friendly", "response time", etc. Don't state trivial qualities of your work, like "no bugs", or "no leaks": your work is supposed to be Quality On Time. That is, simply the right things, simply within the time agreed.

- **Constraints**

☑ **Which activities must be done to realize the requirements stated?** What has to be done before I can say "It is completely finished, I don't have to think about it any more". If the task is a modification, state what modifications have to be done.

☑ **Implementation details (how am I going to implement it)**

☑ **Verification approach – test design**
How can I make sure that it does what it should do and that it does not do what it should not do.

☑ **Planning** (in which order am I going to do things to move efficiently towards the final result?)
What to do first, what to do then: evolutionary steps, no big bang

☑ **Is everything really clear?**

☑ **Have this document (and related docs, if any) reviewed**

☑ **Clarify any unclearness until everything is clear and agreed with the reviewer**

☑ **Detail the design**

☑ **Convert the detailed design to code**

☑ **Verify against the written requirements (not less, not more) and against the design according to the defined test. Comments:**

☑ **Checklist for 100% done:**

☑ **The code compiles and links with all files in integration promotion level**

☑ **The code simply does what it should do: no bugs**

☑ **There are no memory leaks**

☑ **Defensive programming measures have been implemented**

☑ **All files are labeled according to the rules agreed**

☑ **File promotion is done**

☑ **I feel confident that the tester will find no problems**

☑ **Project manager is informed about task completion**

Niels Malotaux

IKD training lente 2008
Succesvol Plannen van Softwareprojecten