Basics:

What is Lean ?
What is Agile ?

Niels Malotaux

N R Malotaux
Consultancy

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

Niels Malotaux

Result Management

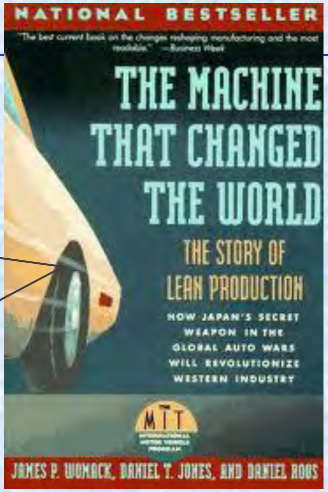
- Project Coach
  - Evolutionary Project Management (Evo)
  - Requirements Engineering
  - Reviews and Inspections
  - Dependable Embedded Systems

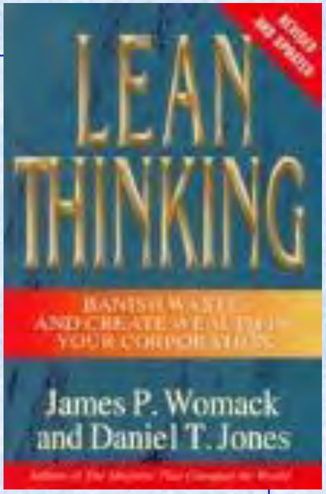
Helping projects and organizations to become Lean quickly

Lean

A lot of the cost of vehicles is based on:

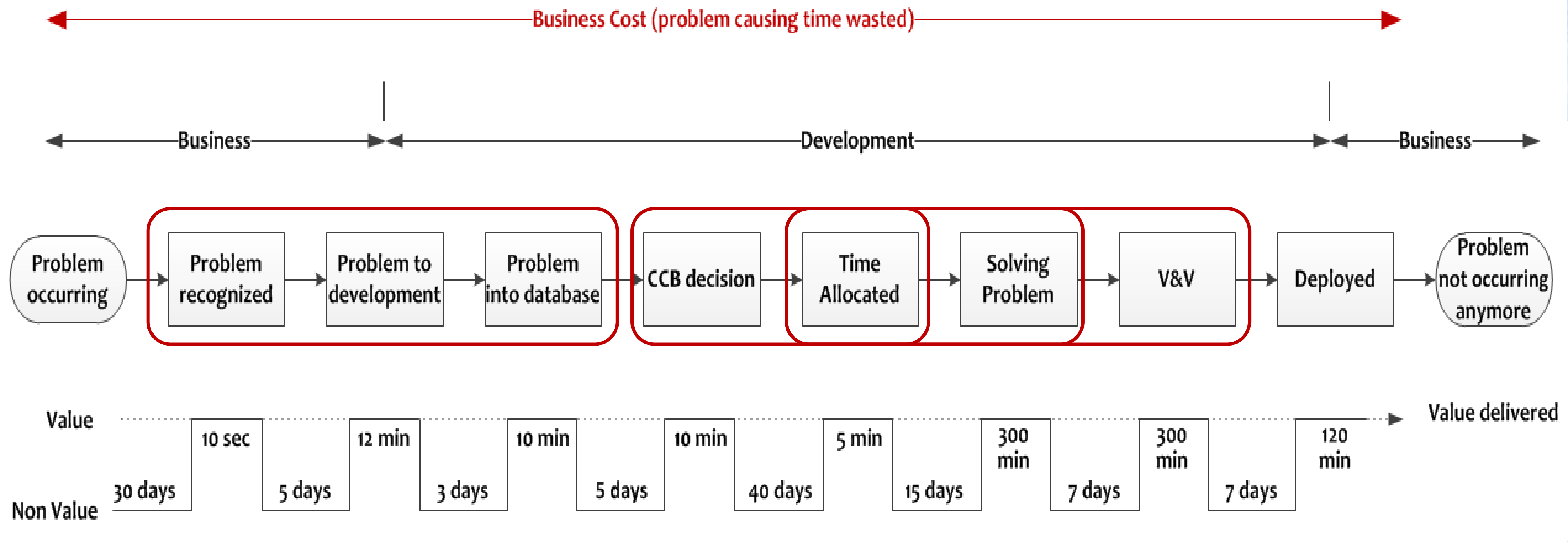
- bad design
- poor management
- an attitude that problems, no matter how small, can be overlooked


NATIONAL BESTSELLER
THE MACHINE THAT CHANGED THE WORLD
THE STORY OF LEAN PRODUCTION
HOW JAPAN'S SECRET WEAPON IN THE GLOBAL AUTO WARS WILL REVOLUTIONIZE WESTERN INDUSTRY
MIT
JAMES P. WOMACK, DANIEL T. JONES, AND DANIEL ROOS

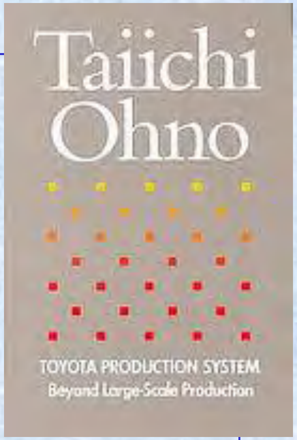

LEAN THINKING
BANISH WASTE AND CREATE WEALTH IN YOUR CORPORATION
James P. Womack and Daniel T. Jones

- The goal is reduction of waste
- To achieve this, a company must look at what creates value and eliminate all other activities
  - Understand and specify the value desired by the customer
  - Identify the value stream for each product providing that value
  - Challenge all of the wasted steps (generally nine out of ten) currently necessary to provide it
  - Make the product flow continuously through the remaining value-added steps
  - Introduce pull between all steps where continuous flow is possible
  - Manage toward perfection so that the number of steps and the amount of time and information needed to serve the customer continually falls

Value stream example



- **Total Business Cost 114 days, Cost of Non Value: 112 days**
- **Occurrence: 2 x per day, delay per occurrence: 10 min**
- **Number of business people affected: 100**
- **Business Cost of Non Value:** 2 x 100 people x 10 min x 112 days x 400€/day = 187 k€
- **Net Cost of Value:** 1.6 days: ~3 people x 1.6 days x 1000€/day = 5 k€


Taiichi Ohno

TOYOTA PRODUCTION SYSTEM
Beyond Large-Scale Production

Toyota Production System (TPS)

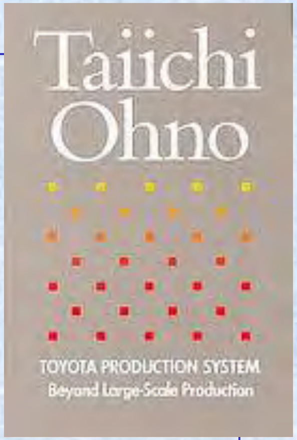
1950

- Toyota almost collapsed
- Laying off 1/3 of workforce

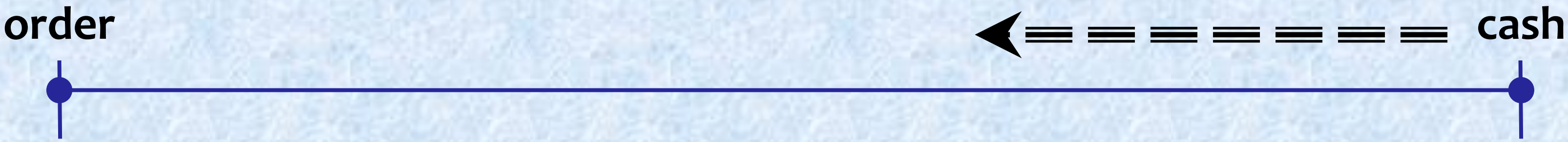
Four specific aims:

- Deliver the highest possible quality and service to the customer
- Develop employee’s potential based upon mutual respect and cooperation
- Reduce cost through eliminating waste in any given process
- Build a flexible production site that can respond to changes in the market

Taiichi Ohno - The Toyota Production System



- All we do is looking at the TimeLine from Order to Cash (p.ix)


order

cash

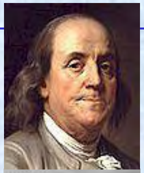
Reducing the time by removing non-value-added wastes

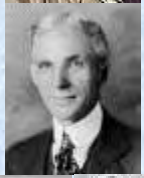
- The Toyota Production System began when I challenged the old system (p11)
- Necessity is the mother of invention:
  improvements are made on clear purposes and need (p13)
- The TPS has been built on the practice of asking "Why?" 5 times (p17)
- The time that provides me with the most vital information about management is the time I spent in the plant, not in the office (p20)
- Toyota's top management watched the situation quietly and I admire the attitude they took (p31)

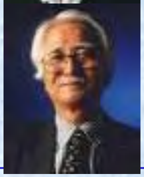
Building on quite some history

- Benjamin Franklin (1706-1790)
  - Waste nothing, cut off all unnecessary activities,
    plan before doing, be proactive, assess results and learn continuously to improve
- Henry Ford (1863-1947)
  - My Life and Work (1922)
    - We have eliminated a great number of wastes
  - Today and Tomorrow (1926)
    - Learning from waste, keeping things clean and safe, better treated people produce more
- Toyoda's (Sakichi, Kiichiro, Eiji) (1867-1930, 1894-1952, 1913-)
  - Jidoka: Zero-Defects, stop the production line (1926)
  - Just-in-time – flow – pull
- W. Edwards Deming (1900-1993)
  - Shewart cycle: Design-Produce-Sell-Study-Redesign (Japan – 1950)
  - Becoming totally focused on quality improvement (Japan – 1950)
    Management to take personal responsibility for quality of the product
  - Out of the Crisis (1986) – Quality reduces waste
- Joseph M. Juran (1904-2008)
  - Quality Control Handbook (1951, Japan – 1954)
  - Total Quality Management – TQM
  - Pareto Principe
- Philip Crosby (1926-2001)
  - Quality is Free (1980)
  - Zero-defects (1961)
- Taiichi Ohno (1912-1990)
  - (Implemented the) Toyota Production System (Beyond Lange-Scale Production) (1978, 1988)
  - Absolute elimination of waste - Optimizing the TimeLine from order to cash
- Masaaki Imai (1930-)
  - Kaizen: The Key to Japan's Competitive Success (1986)
  - Gemba Kaizen: A Commonsense, Low-Cost Approach to Management (1997)

Eliminating Waste
Not doing what
doesn't yield value







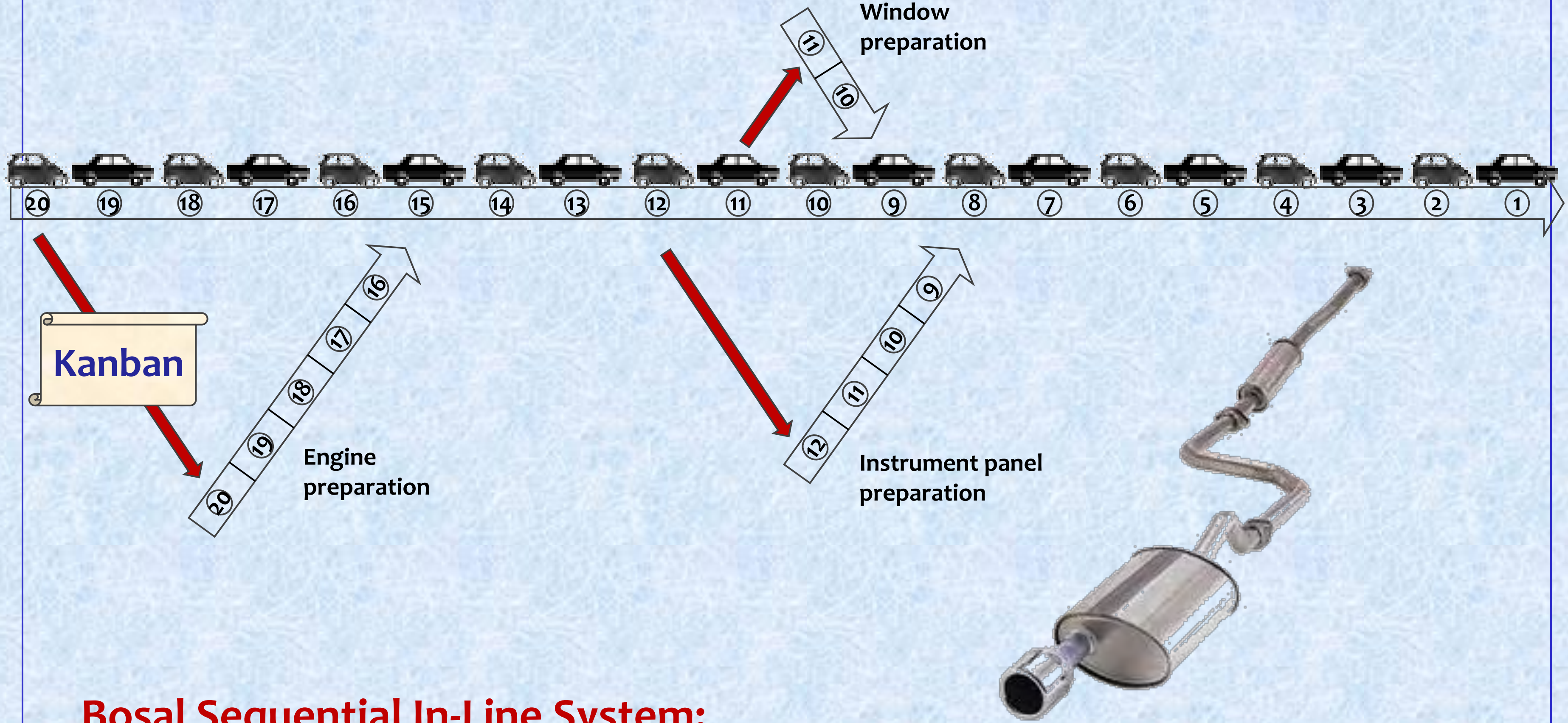
Pillars of the TPS

- **Just in Time**
  - No inventory
  - Doing the right things at the right time
- **Perfection**
  - Perfection is a condition for JIT to work
  - If a defect is found, stop the line, find cause, fix immediately
  - Continuous improvement of product, project and process
- **Autonomation**
  - The loom runs unattended until signalling it needs help

  For development:
  - The development team runs unattended until signalling they need help (caused by an issue beyond their control)
  - Management **observes** the team and **facilitates** them to become ever more efficient, to *prevent* issues delaying them beyond the teams control – *Education*, *Empowerment* and *Responsibility* of people
  - If an issue does occur, management helps to **remove obstacles** quickly, making sure it doesn't happen again

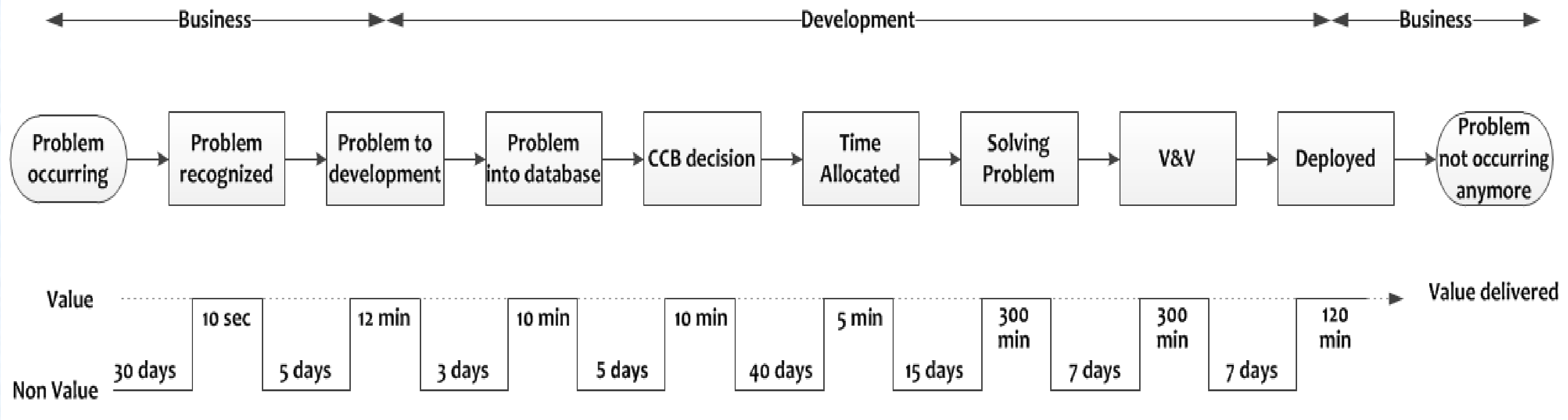
Just In Time delivery – *no inventory*

(after Ohno)



Bosal Sequential In-Line System:
We pioneered just-in-time delivery of exhaust systems - supplying systems to the assembly line within 80 minutes of receiving the order

Capacity = Work + Waste



Work Capacity

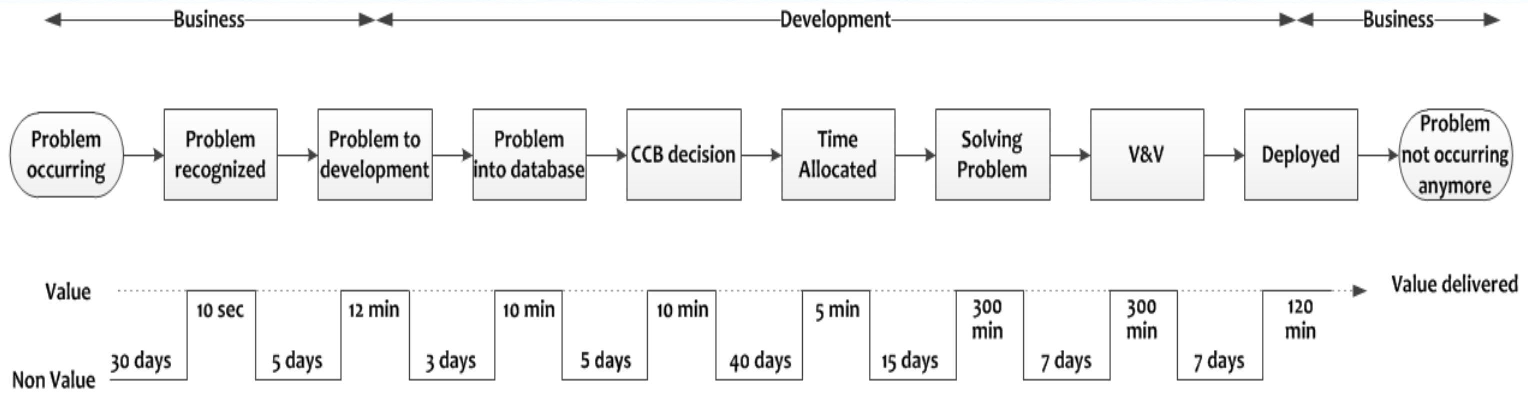
- Net Work, creating value
- Non-value adding, but necessary work
- Waste

Because it costs nothing, eliminating waste is one of the easiest ways for an organization to improve it's operations

Identifying waste

Manufacturing	Development	*Possible* Remedies
Overproduction	Extra features Unused documents	Prioritizing, Real Requirements, Deciding what not to do
Inventory	Partially done work	Synchronization, Just In Time
Transport	Handoffs	Keeping in one hand/mind: - Responsibility (what to do) - Knowledge (how to do it) - Action (doing it) - Feedback (learning from Result)
Processing	Design inefficiency Wishful thinking	Knowledge, experience, reviews Preflection
Waiting	Delays	Process/Organization redesign
Movement	Task Switching	Max 2 tasks in parallel
Defects	Defects	Prevention
Ignoring ingenuity of people	Ignoring ingenuity of people	Real management, Empowerment Bottom-up responsibility

5-S



- **Seiri - Remove unnecessary things** → waste
- **Seiton - Arrange remaining things orderly** → flow
- **Seiso - Keep things clean** → uncovers hidden problems
- **Seiketsu - Keep doing it, standardize** → know what to improve
- **Shitsuke - Keep training it** → fighting entropy

The 3 Mu's to remove

- Muda - Waste → minimize waste
- Mura - Irregularities → optimize flow
- Muri - Stress → sustainable pace

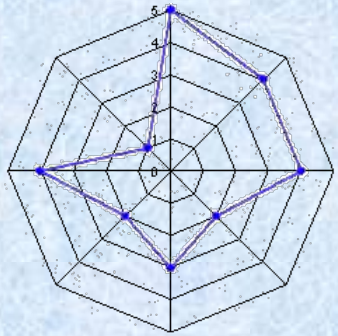
Determine value

- Heathrow Terminal 5: “Great success !”
  - Normal people aren’t interested in the technical details of a terminal
  - They only want to check-in their luggage as *easily* as possible
    and
  - Get their luggage back as *quickly* as possible in *acceptable condition at their destination*
  - They didn’t
- One of the problems is to determine
  what the project (or our work in general) really is about
- A project only can deliver the conditions for value

What is Agile ?

- Actually a philosophy (Agile Manifesto)

The Agile Manifesto (2001)



We are uncovering better ways of developing software by doing it and helping others do it

Through this work we have come to value:

- Individuals and interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan

That is, while there is *value in the items on the right*, we value the items on the left more

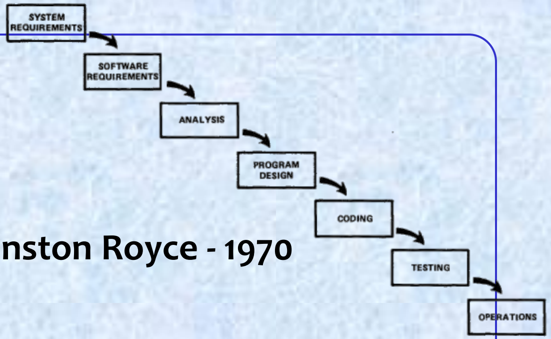
Principles behind the Agile Manifesto - 1

- Our highest priority is to satisfy the customer through early and continuous delivery of valuable software
- We welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage
- We deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale
- Business people and developers must work together daily throughout the project
- We build projects around motivated individuals. We give them the environment and support they need, and trust them to get the job done
- The most efficient and effective method of conveying information to and within a development team is face-to-face conversation

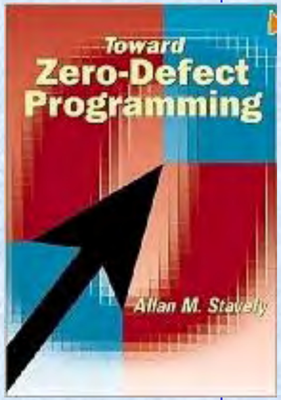
Principles behind the Agile Manifesto - 2

- Working software is the primary measure of progress
- Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely
- Continuous attention to technical excellence and good design enhances agility
- Simplicity - the art of maximizing the amount of work not done - is essential
- The best architectures, requirements, and designs emerge from self-organizing teams
- At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behaviour accordingly

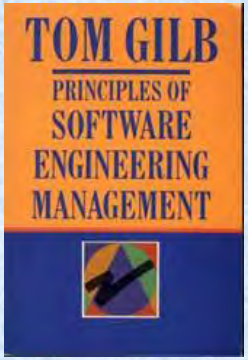
It has old roots

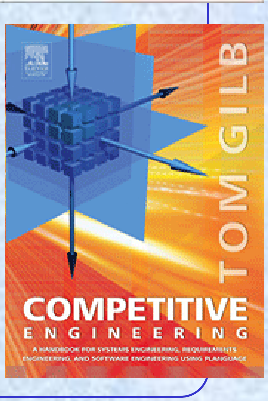


- **Managing the development of large software systems** - Winston Royce - 1970
  - Famous ‘Waterfall document’: figure 2 showed a ‘waterfall’
  - Text and other figures showed that Waterfall doesn’t work
  - Anyone promoting Waterfall doesn’t know or didn’t learn from history
- **Cleanroom software engineering** - Harlan Mills - 1970’s
  - Incremental Development - Short Iterations
  - Defect *prevention* rather than defect removal
  - Inspections to feed prevention
  - No unit tests needed
  - Statistical testing
  - If final tests fail: no repair - start over again
  - 10-times less defects at lower cost
  - Quality is *cheaper*



- **Evolutionary Delivery - Evo** - Tom Gilb - 1974, 1976, 1988, 2005
  - Incremental + Iterative + *Learning and consequent adaptation*
  - Fast and Frequent Plan-Do-Check-Act
  - Quantifying Requirements - Real Requirements
  - Defect *prevention* rather than defect removal




TOM GILB
COMPETITIVE ENGINEERING
A HANDBOOK FOR SYSTEMS ENGINEERING, REQUIREMENTS ENGINEERING, AND SOFTWARE ENGINEERING USING PLANGUAGE

What is Agile ?

- Actually a philosophy (Agile Manifesto)
- Agile = ability to move quick, easy and adaptable
- Short iterations – not Waterfall
- Delivering value (not much notion how to define value)
- Retrospectives (no retrospectives on retrospectives)
- Not a standard: You can make of it what you want
- XP - focus on software development techniques
- Scrum – very basic short term organization of development
- Are you Agile if you religiously focus on a ‘method’ ?
- Is Agile Lean ?

XP – eXtreme Programming

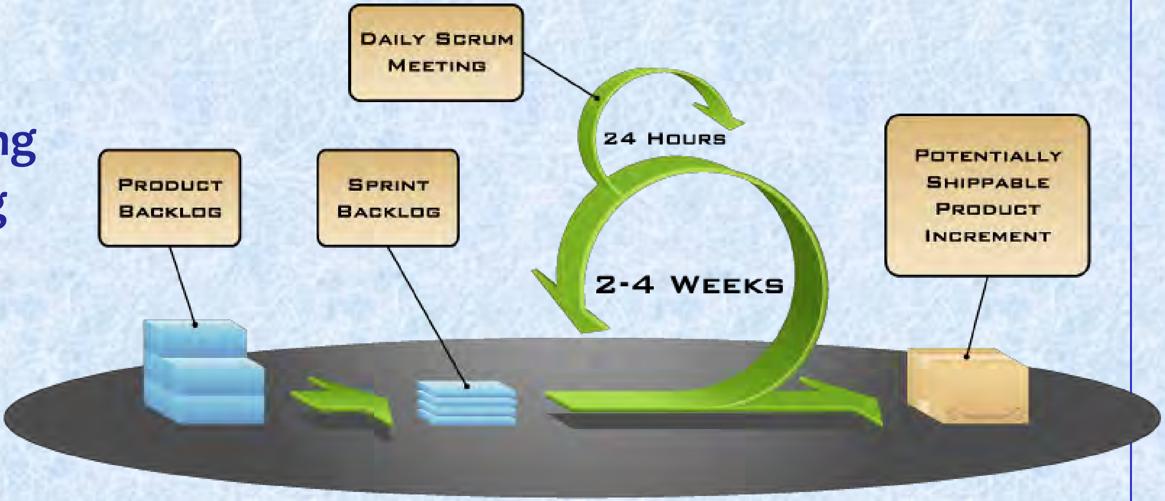
- Planning Game
- Metaphor
- Simple Design
- Testing (TDD)
- Refactoring
- Coding standards

- Small releases
- Pair programming
- Collective Ownership
- Continuous integration
- 40-hour week
- On-site customer

Original project was not successful
as soon as the writer of the book left the project

Scrum

- **Sprint**
  - 1 – 4 weeks
  - Sprint Planning meeting
  - Sprint Review meeting
  - Sprint Retrospective
- **Artifacts**
  - Product backlog
  - Sprint backlog
  - Sprint burn down chart



- **Roles**
  - Scrum Master (facilitates, coaches on rules)
  - Team – multifunctional (design, develop, test, etc)
  - Product Owner – voice of customer
- **Daily Scrum - Stand-up meeting**
  a. What have you done since yesterday
  b. What are you planning today
  c. Impediments limiting achieving your goals ?

80% of Scrum projects are ScrumBut

What's missing in Agile ?

Ref Tom Gilb

Stakeholder Focus

- Real projects have dozens of stakeholders
  - Not just a customer in the next room, not just a user with a use case or story

Results Focus

- It is not about *writing code*, it is about delivering *value to stakeholders*
- It is not about *programming*, it is about making *systems* work, for *real* people

Systems Focus

- It is not about coding, but rather:
  reuse, data, hardware, training, motivation, sub-contracting, outsourcing, help lines, user documentation, user interfaces, security, etc.
- So, a *systems engineering* scope is necessary to deliver results
- Systems Engineering needs *quantified performance and quality objectives*

Planning

Ref Niels Malotaux

- Retrospectives within the Sprint
- Retrospectives of retrospectives
- Planning what *not* to do → *preflection*

Summary

- Lean:
  - Delivering quality, by
    - Determining value
    - Eliminating waste
    - Minimizing / optimizing non-value-added work
    - JIT, Autonomation, Kanban, PDCA, Value Stream Mapping, Flow
    - Perfection
    - Involving the whole organization
  - Website INCOSE Lean-SE Working group (192 Lean Enablers)

    http://cse.lmu.edu/about/graduateeducation/systemsengineering/INCOSE.htm
- Agile:
    - Philosophy
    - Ability to move quick, easy and adaptable
    - Delivering (however, not much notion how to define value)
    - Self-organization (assumes everybody in the team is entrepreneur)
    - Removing impediments (usually ad-hoc)
    - Non-waterfall: Adapting to changing requirements
    - TimeBoxed, short iterations
    - Retrospectives (?)
    - XP ? Scrum ? ← some formalized processes

Basics:

What is Lean ?
What is Agile ?

Niels Malotaux

N R Malotaux
Consultancy

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

How is it possible that most organizations still survive while their competitors are applying Lean?

Niels Malotaux

N R Malotaux
Consultancy

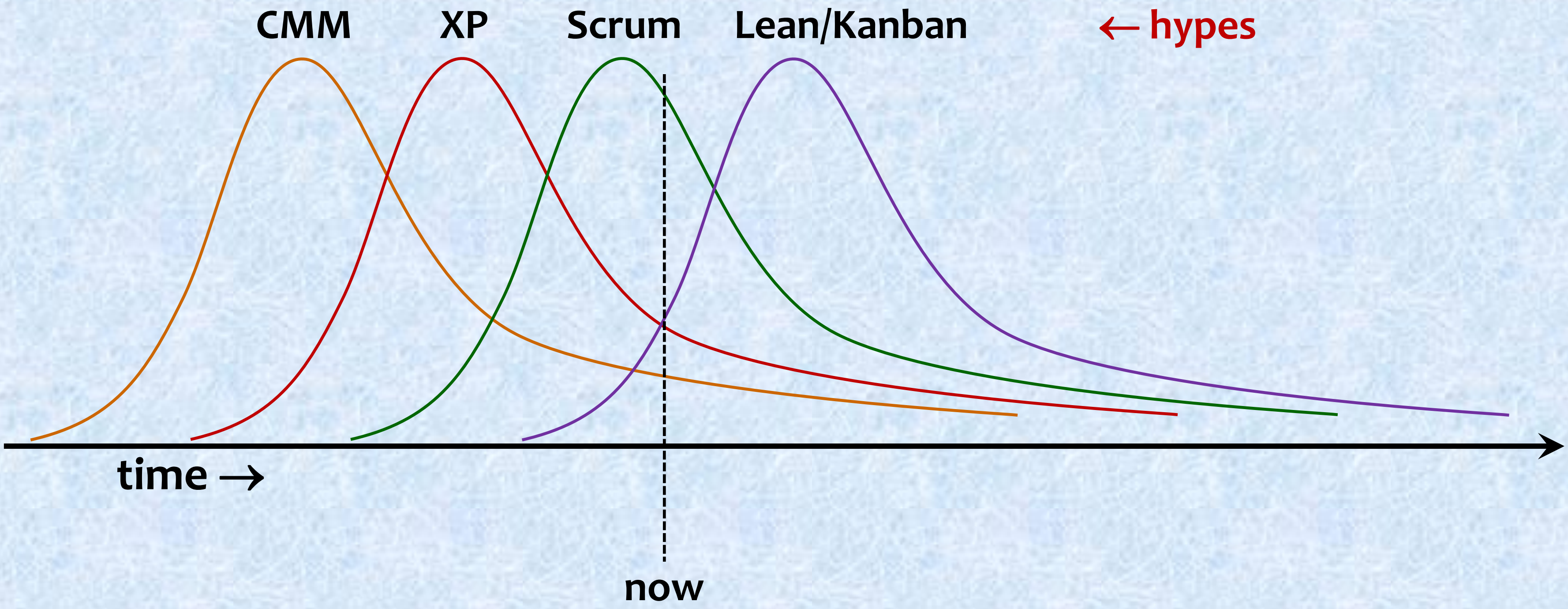
+31-30-228 88 68 niels@malotaux.nl www.malotaux.nl

How is it possible ?

- In manufacturing and development, Lean can save 50-70% of cost and time and increase quality
- How is it possible that most organizations still survive while their competitors are applying Lean?
- They don’t
- Why is it so difficult to apply Lean or Agile principles ?
- It’s not tools, it’s people
- What we should do often is contra-intuitive
- Otherwise it would have happened a long time ago

It’s not the method

(which method do *you* use ?)



If the previous method didn’t work, the next won’t work either

Causes of Delay

- Some typical causes of delay are:
  - Developing the wrong things
  - Unclear requirements
  - The fallacy of 'all' requirements
  - Misunderstandings
  - No feedback from stakeholders
  - No adequate planning
  - No adequate communication
  - Doing unnecessary things
  - Waiting (before and during the project)
  - Changing requirements
  - Doing things less cleverly
  - Doing things over
  - Indecisiveness
  - Suppliers
  - Quality of suppliers results
  - Hobbying
  - Political ploys
  - Boss is always right (culture)
- Excuses, excuses: it's always "*them*". How about "*us*" ?
- A lot of delay is avoidable and therefore unjustifiable

Causes of causes (use 5 times ‘Why ?’)

- Management
- No Sense of Urgency
- Uncertainty
- Perceived weakness
- Fear of Failure
- Ignorance
- Incompetence
- Politics

- Indifference
- Discipline
- Intuition
- Perception
- Lack of time
- Not a Zero Defects attitude
- No techniques offered
- No empowerment

- So called Scientific Management Techniques
- No dissemination through the whole organization

What has this to do with Systems Engineering ?

- The Project Manager is *responsible* for ***delivering*** the right result at the right time
- The Project Worker’s (including SE’s) work and decisions ***determine*** the ***result*** and the ***time*** it is delivered
- This makes everybody in the project implicitly ***as responsible as*** Project Management

Human Behaviour

- Most project process approaches (PMI, INCOSE, CMMi, developers)
  - ignore human behaviour,
  - incorrectly assume behaviour,
  - or decide how people should behave *(ha ha)*

- 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
- To succeed in projects, we must study and adapt to real behaviour rather than assumed behaviour
- *Even if we don't agree with that behaviour*

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
  - The good that I want to do, I do not ...

→ Helping each other (watching over the shoulder)

→ Rapid success (do it 3 weeks for me…)

→ Making mistakes (provides short window of opportunity)

→ Openness (management must learn how to cope)

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

We failed because of politics

- Good politics:
  - People decide differently on different values
- Bad politics: hidden agenda's ← Not Lean !
  - Say this, mean that - often even unintentionally
  - Politics thrive by vagueness
  - Facts can make bad politics loose ground
  - Lean and Agile systematically expose and get rid of the waste

- If you accepted the responsibility for the project, failure because of "politics" is just an excuse
- What did you really do about it ?

Culture

- It failed because of the existing culture ← Not Lean !

- Culture is the result of how people work together
- Culture can’t be changed
- Culture *can* change
- By doing things differently

Lean things

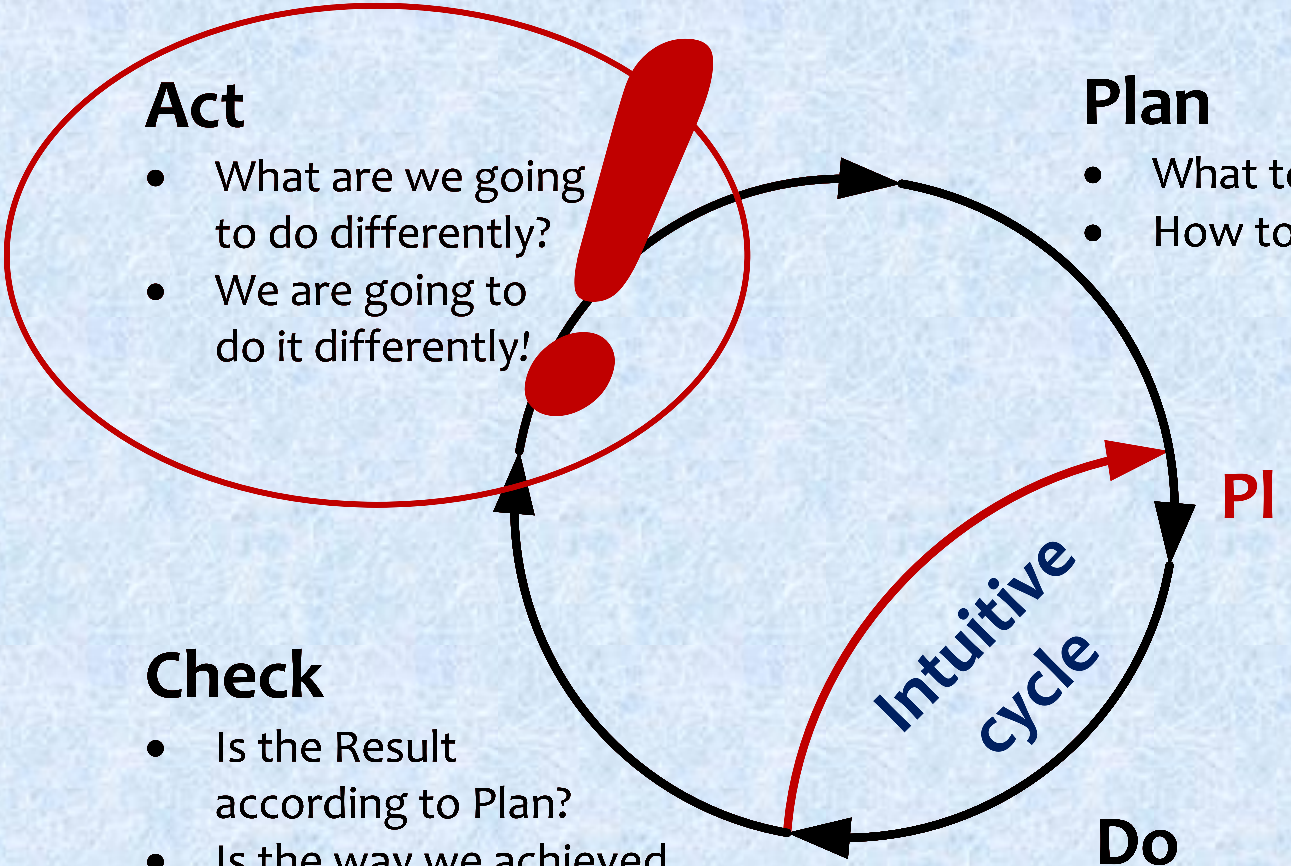
- Most managers think their greatest contribution to the business is doing work-arounds on broken processes, rather than doing the hard work to get the process right so that it never breaks down (Womack)
- 90 per cent of all corporate problems can be solved using common sense and improving quality while reducing cost through the elimination of waste
  Imai: *Gemba Kaizen* - A Commonsense Low-Cost Approach to Management
- Plan-Do-Check-Act cycle was by far the most important thing we did in hindsight (Tom Harada)
- Root-Cause-Analysis on every defect found ?
  We don’t have time for that ! (project manager)

Help! We have a QA problem !

- Large stockpile of modules to test
  (hardware, firmware, software)
- You shall do Full Regression Tests
- Full Regression Tests take about 15 days each
- Too few testers ("Should we hire more testers ?")
- Senior Tester paralyzed
- Can we do something about this?

The essential ingredient: the PDCA Cycle

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



Act

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

Plan

- What to achieve
- How to achieve it

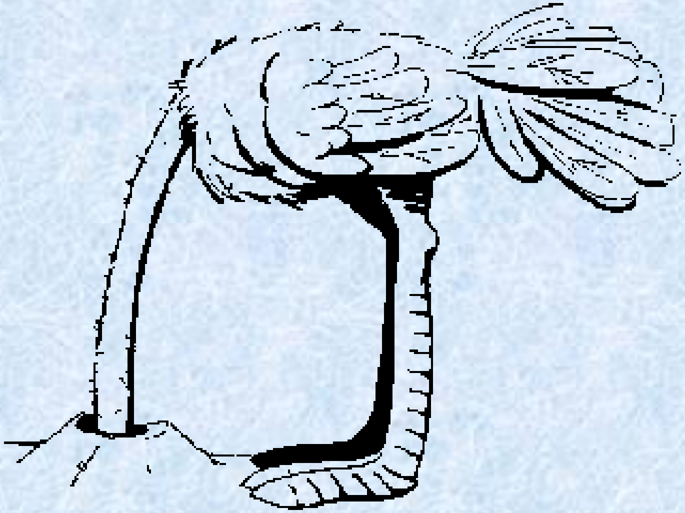
Check

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

Do

Do something

Carry out the Plan



Instead of complaining about a problem …

(Stuck in the Check-phase)

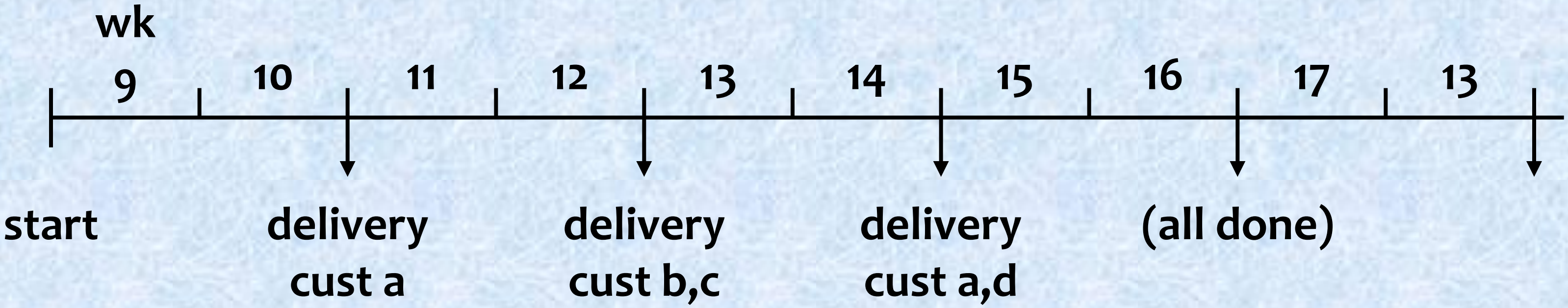
Let's do something about it !

(Moving to the Act-phase)

Objectifying and quantifying the problem is a first step to the solution

Line	Activity	Estim	Alter native	Junior tester	Devel opers	Customer	Will be done (now=22Feb)
1	Package 1	17	2	17	4	HT	
2	Package 2	8	5		10	Chrt	
3	Package 3	14	7	5	4	BMC	
4	Package 4 (wait for feedback)	11				McC?	
5	Package 5	9	3		5	Ast	
6	Package 6	17	3	10	10	?	
7	Package 7	4	1		3	Cli	
8	Package 8.1	1	1			Sev	
9	Package 8.2	1	1			?	
10	Package 8.3	1	1			Chrt	24 Feb
11	Package 8.4	1	1			Chrt	
12	Package 8.5	1.1	1.1			Yet	28 Feb
13	Package 8.6	3	3			Yet	24 Mar
14	Package 8.7	0.1	0.1			Cli	After 8.5 OK
15	Package 8.8	18	18			Ast	
	totals	106	47	32	36		

TimeLine



Selecting the priority order of customers to be served

- “We’ll have a solution at that date … Will you be ready for it ?”
  Another customer could be more eagerly waiting
- Most promising customers

Result

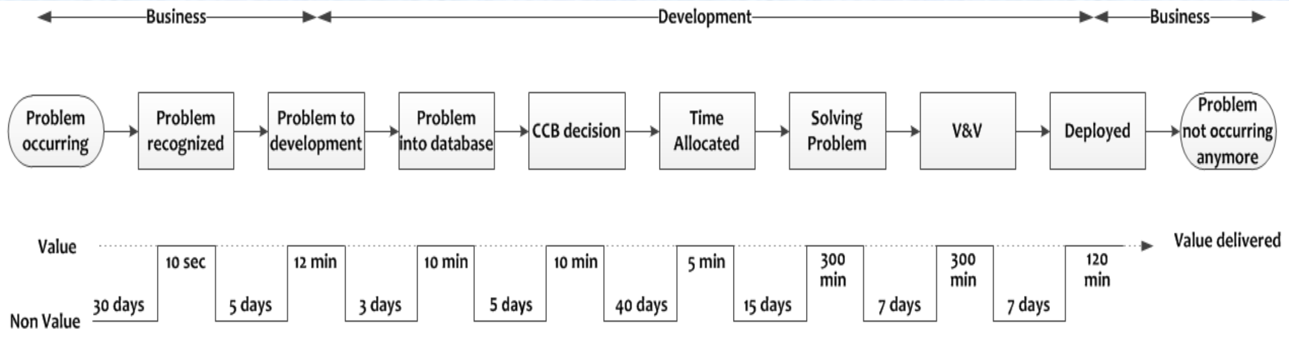
- Tester empowered
- Done in 9 weeks
- So called “Full Regression Testing” was redesigned
- Customers systematically happy and amazed
- Kept up with development ever since
- Increased revenue

Recently:

- Tester promoted to product manager
- Still coaching successors how to plan

Value Stream Optimization in Development

- Projects take longer than necessary
- We can only save time by *actively preventing* waste
  - Optimizing Value Stream in repetitive processes, like production, is relatively easy



- But how to optimize non-repetitive processes, like explorative development ?
  - First discard the 'obvious' (repetitive) waste
  - Then prevent the non-repetitive waste by *preflection*

My Practical Approaches for Becoming Lean and Really Agile

Predictable Projects

Niels Malotaux

N R Malotaux
Consultancy

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

Ultimate Goal of a Project

Quality on Time

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

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

Lean is about continuous improvement

Insanity is doing the same things over and over again and hoping the outcome to be different *(let alone better)*

Albert Einstein 1879-1955, Benjamin Franklin 1706-1790, it seems Franklin was first

Only if we change our way of working,
the result may be different

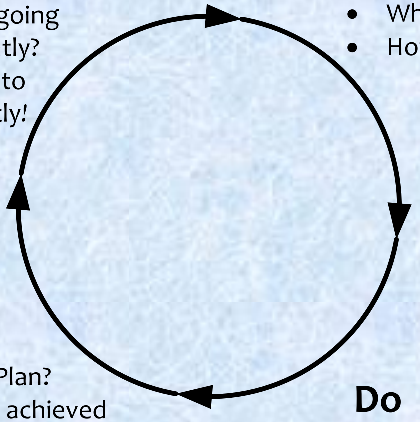
- Hindsight is easy, but reactive
- Foresight is less easy, but proactive
- Reflection is for hindsight and learning
- Preflection is for foresight and prevention

Only with *prevention* we can save precious time

This is used in the Deming or Plan-Do-Check-Act cycle

The essential ingredient: the PDCA Cycle

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



Act

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

Plan

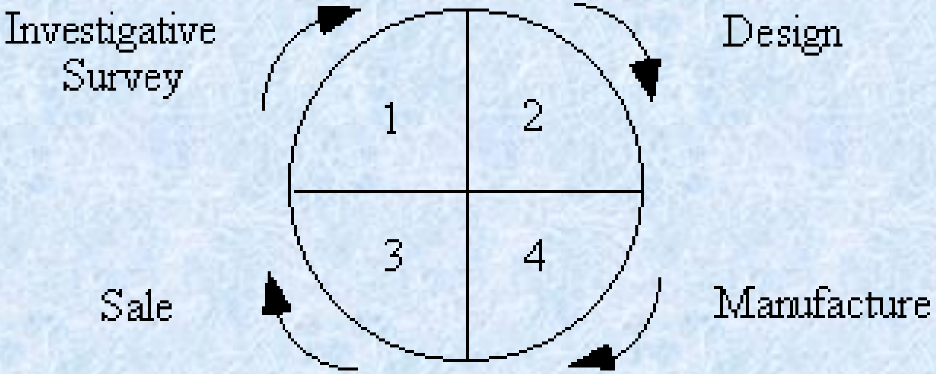
- What to achieve
- How to achieve it

Check

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

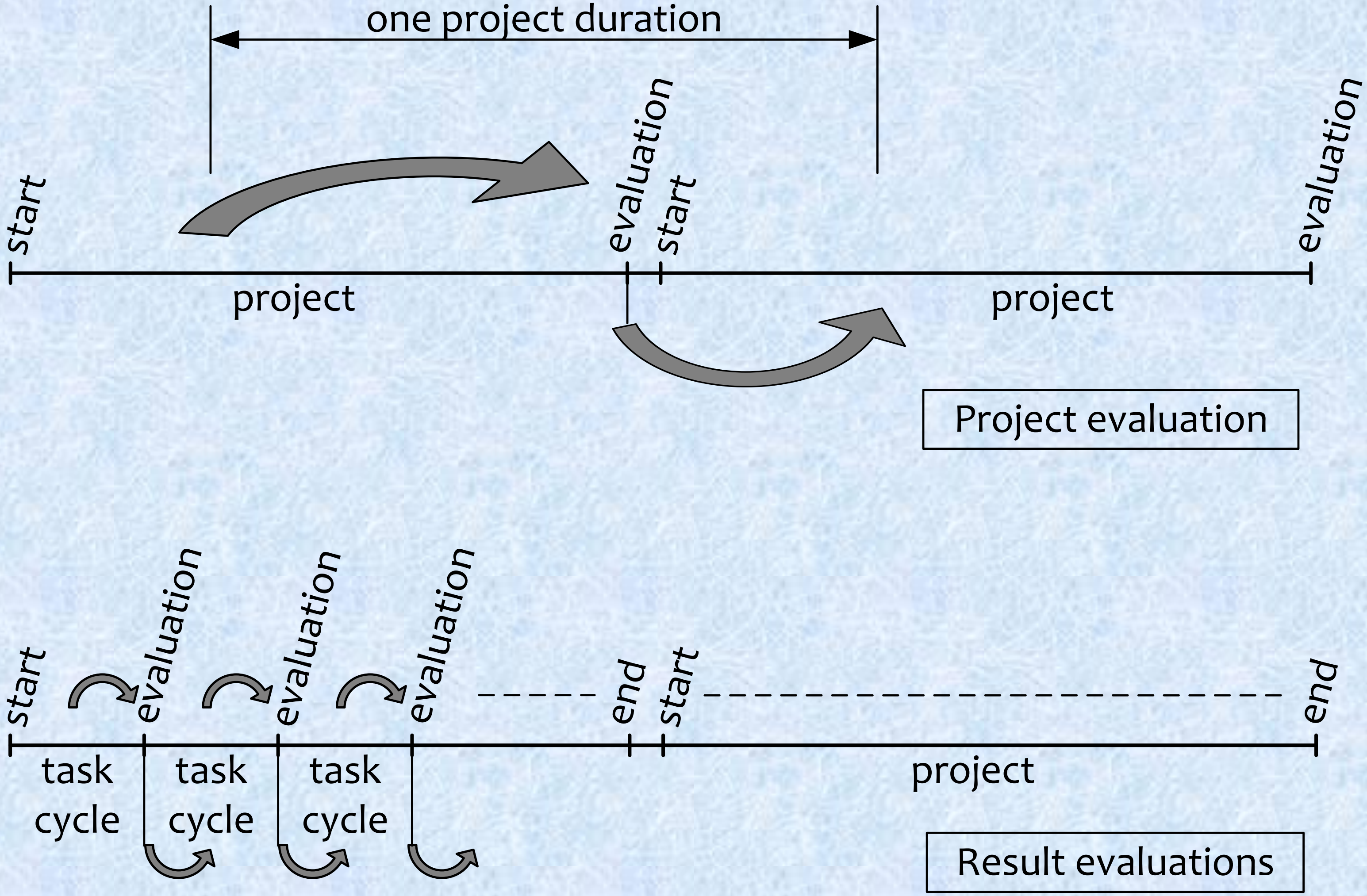
Do

Carry out the Plan

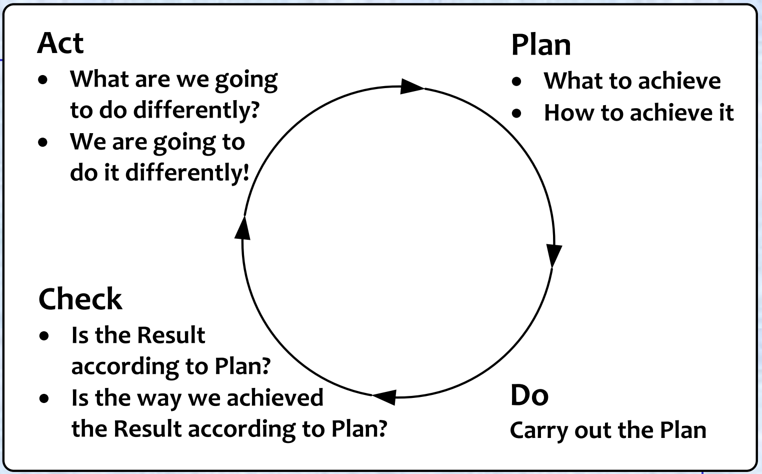


What Deming showed in Japan in 1950

Project evaluations



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

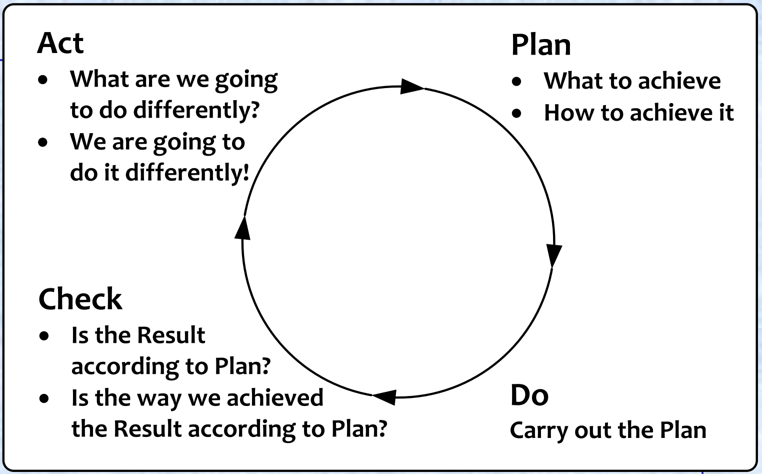
- Most quickly learn what the real requirements are
- Learn how to most effectively and efficiently realize these requirements

and we can

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

doing the right things right

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 shouldn't be affected
- Evo is about delivering Real Stuff to Real Stakeholders doing Real Things *"Nothing beats the Real Thing"*
- Projects seriously applying Evo, routinely conclude successfully on time, or earlier

Evolutionary Project Management (Evo)

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

Zero Defects Attitude

Right product

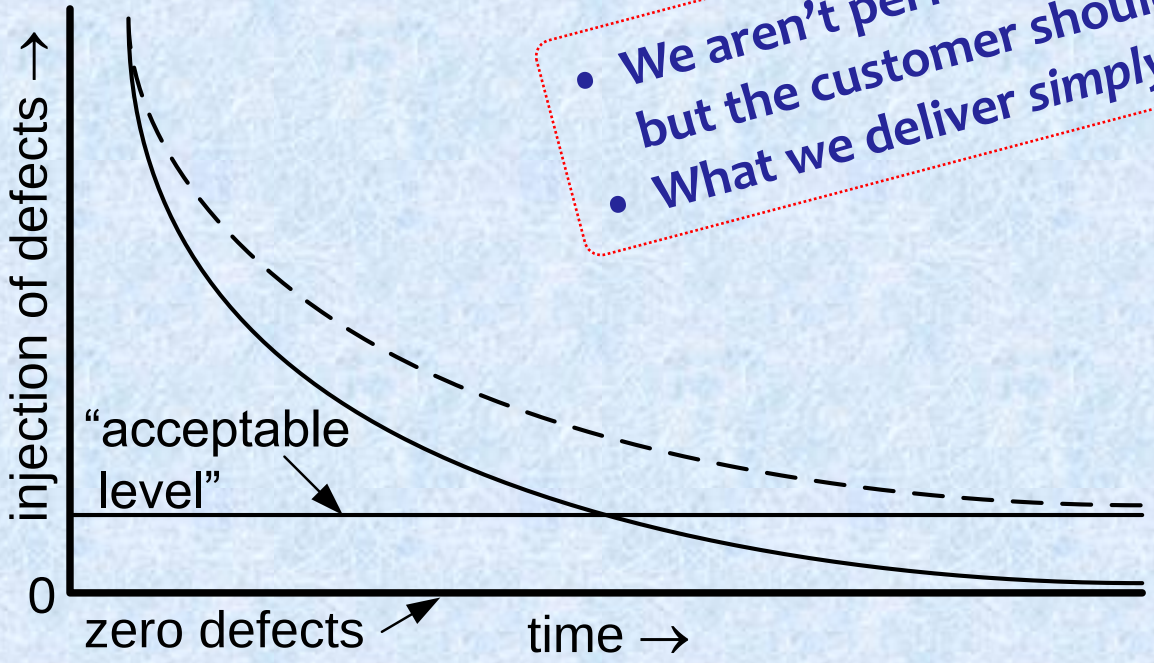
Evo Project Planning

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

Right time

Is Zero Defects possible?

- Zero Defects is an asymptote

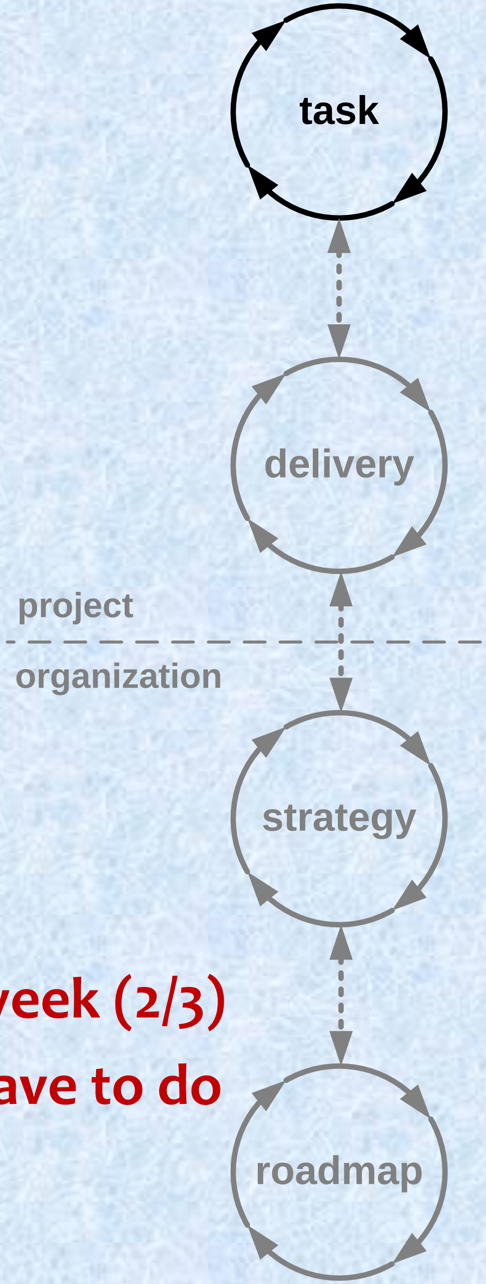


- We aren't perfect, but the customer shouldn't find out
- What we deliver simply works

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

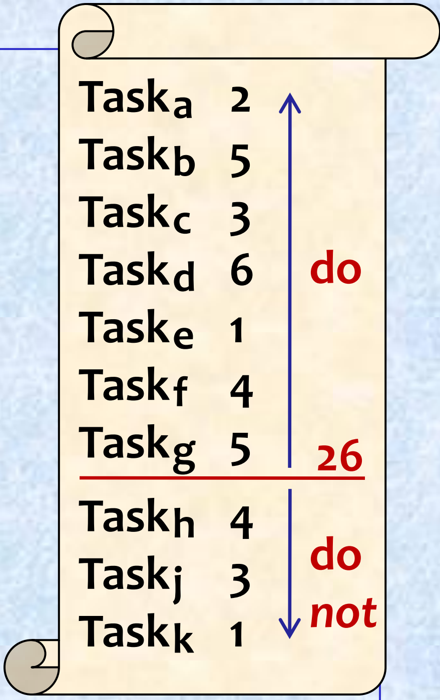
Evo Planning: Weekly TaskCycle

- 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



Every week we plan

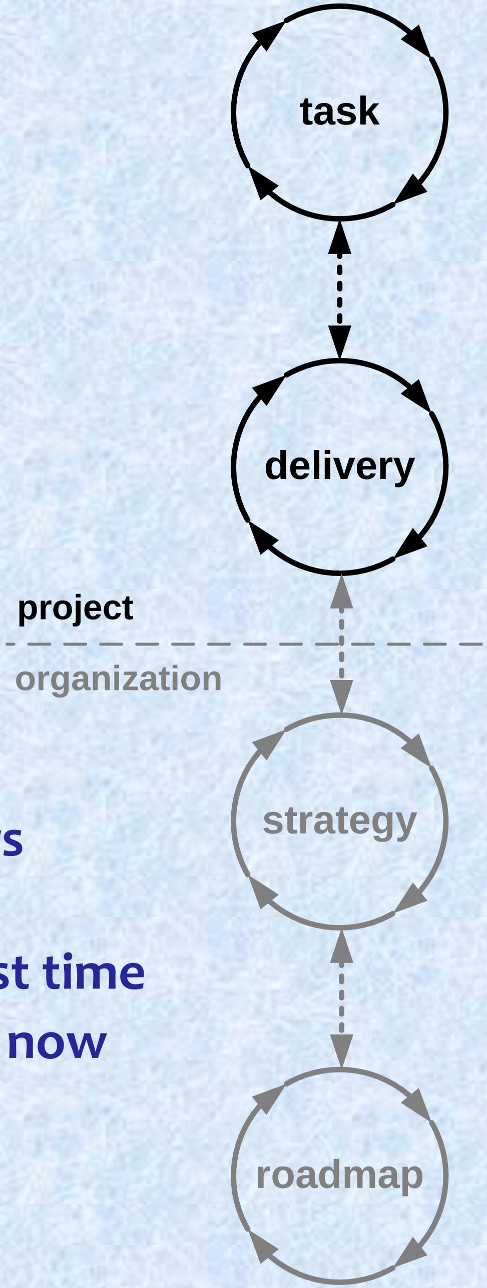
- How much time do we have available
- 2/3 of available time is net plannable time
- What is most important to do
- Estimate effort needed to do these things
- Which most important things fit in the net available time (default 26 hr per week)
- What can, and are we going to do
- What are we *not* going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



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	do *not*
Task$_j$	3	
Task$_k$	1	

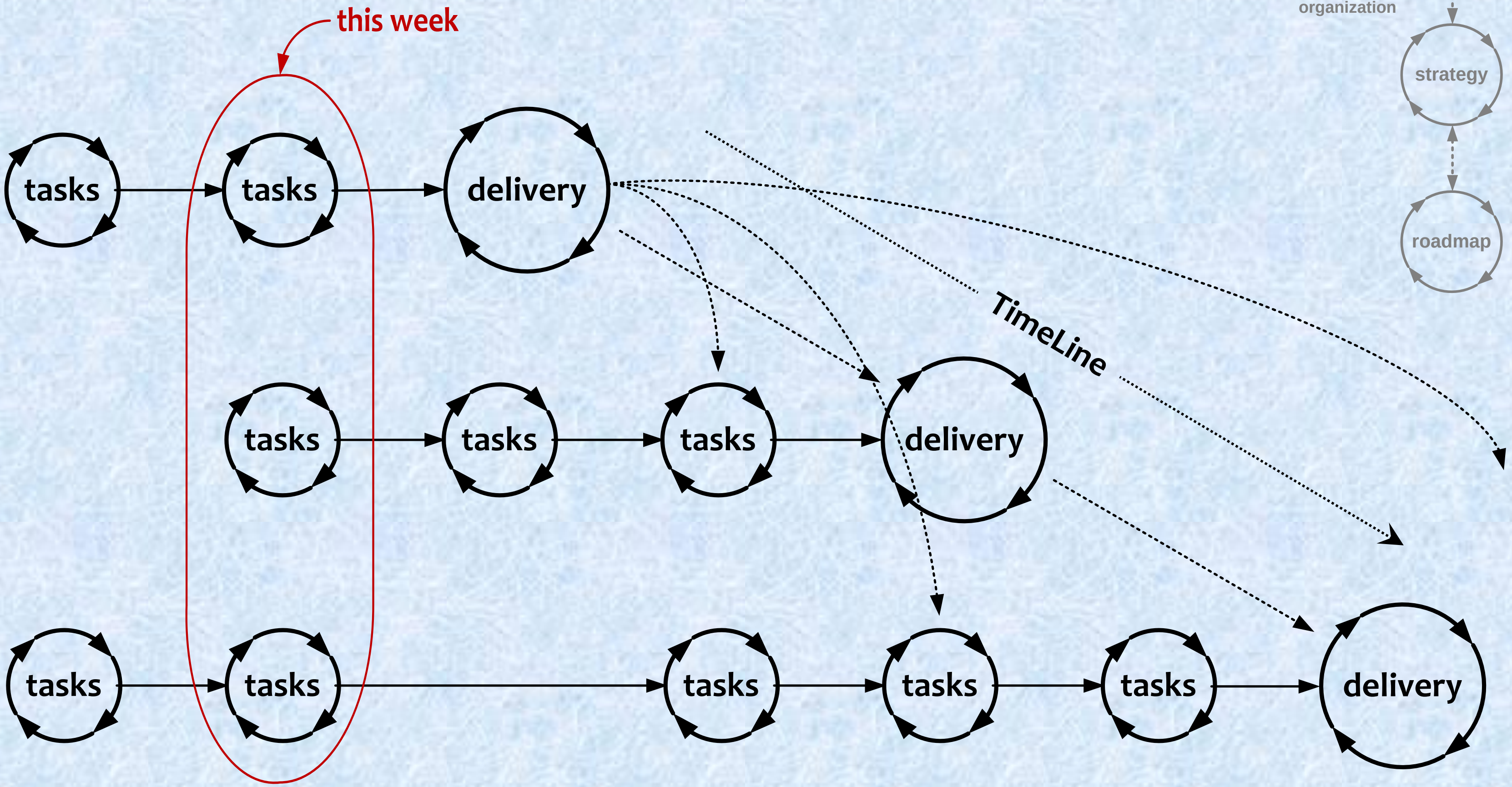
2/3 is default start value
this value works well in development projects

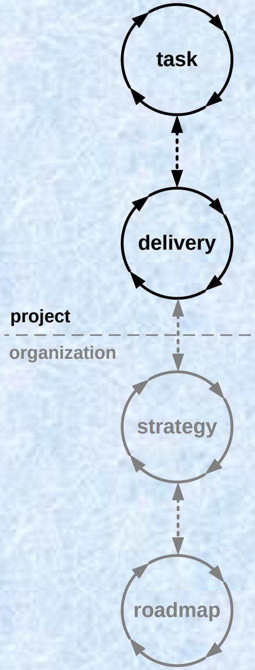
DeliveryCycle



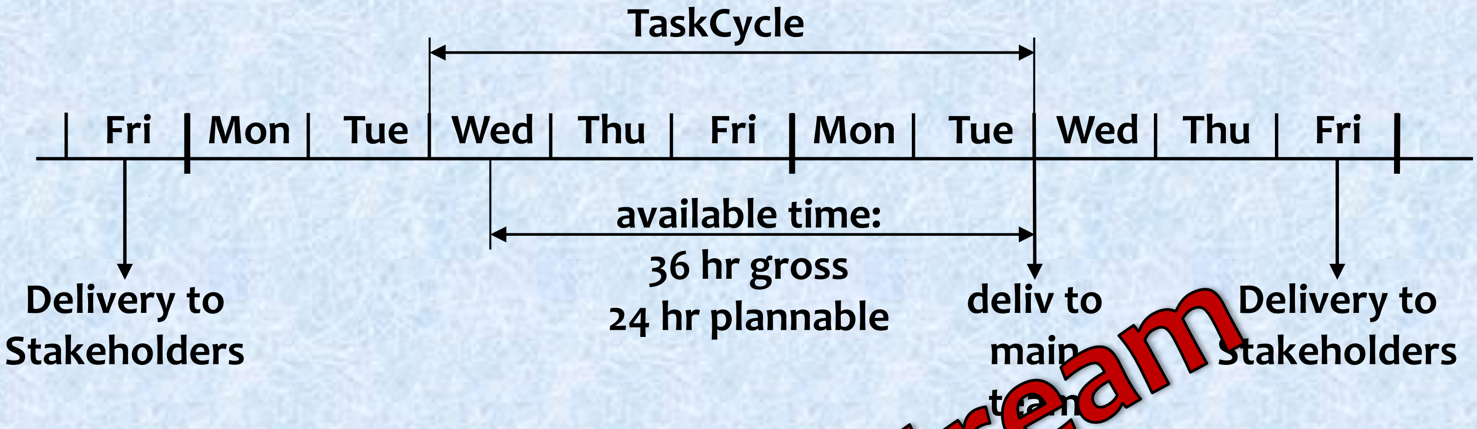
- Are we *delivering* the right things,
  in the right order
  to the right level of detail for now
- Optimizing requirements
  and checking assumptions
  1. What will generate the optimum feedback
  2. We deliver only to eagerly waiting stakeholders
  3. Delivering the juiciest, most important
     stakeholder values that can be made in the least time
  - What will make Stakeholders more productive now
- Making sure we understand what value is
- Not more than 2 weeks

Tasks feed Deliveries





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
-	[illegible]
-	2
-	2
-	3
-	5
-	6
XMLa	4
XMLb	4
Total	40

Gregory (later)	
Draft design	0
Finish design	0
...	

Jerome	
XMLa	3
XMLb	3
...	

Optimizing Value Stream by Prevention

Agile, but will we be on time ?

- Organizing the work in very short cycles
- Making sure we are doing the right things
- Doing the right things right
- Continuously optimizing (what not to do)
- So, we already work more efficiently

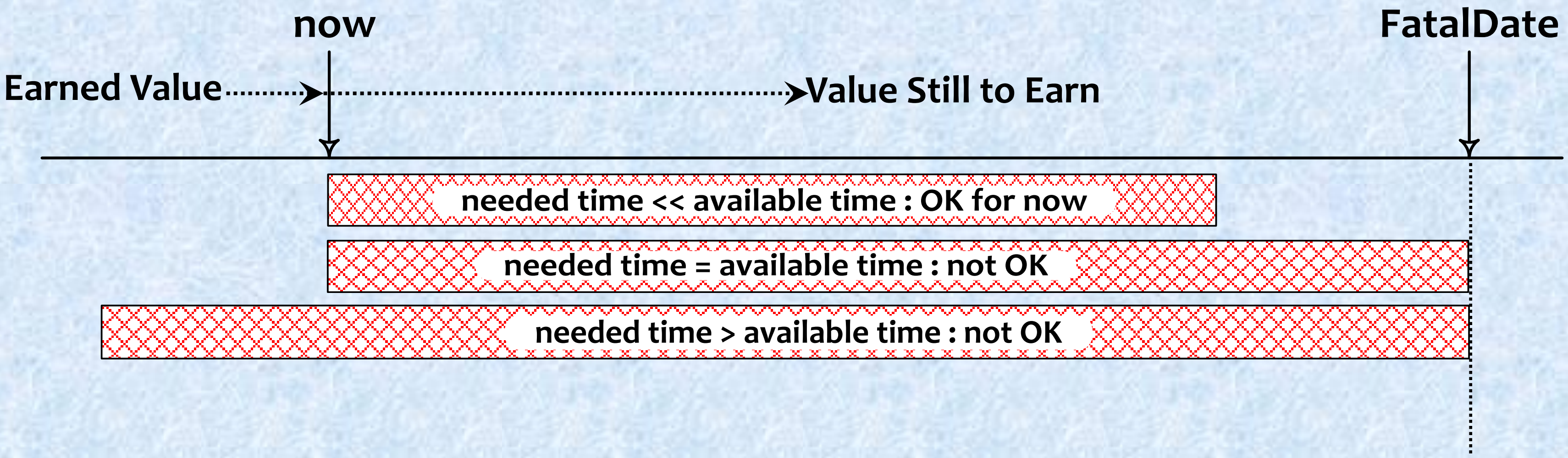
but …

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

Predicting *what* will be done *when*

Line	Activity	Estim	Spent	Still to	Ratio	Calibr	Calibr	Date
				spend	real/es	factor	still to	done
1	Activity 1	2	2	0	1.0			
2	Activity 2	5	5	1	1.2	1.0	1	30 Mar 2009
3	Activity 3	1	3	0	3.0			
4	Activity 4	2	3	2	2.5	1.0	2	1 Apr 2009
5	Activity 5	5	4	1	1.0	1.0	1	2 Apr 2009
6	Activity 6	3				1.4	4.2	9 Apr 2009
7	Activity 7	1				1.4	1.4	10 Apr 2009
8	Activity 8	3				1.4	4.2	16 Apr 2009
↓	↓							
16	Activity 16	4				1.4	5.6	2 Jun 2009
17	Activity 17	5				1.4	7.0	11 Jun 2009
18	Activity 18	7				1.4	9.8	25 Jun 2009

What do we do if we see we won’t make it on time ?


now
FatalDate
Earned Value
Value Still to Earn
needed time << available time : OK for now
needed time = available time : not OK
needed time > available time : not OK

- **If it doesn’t fit … count backwards**
- **When the match is over, you can’t score a goal any more**

Deceptive options

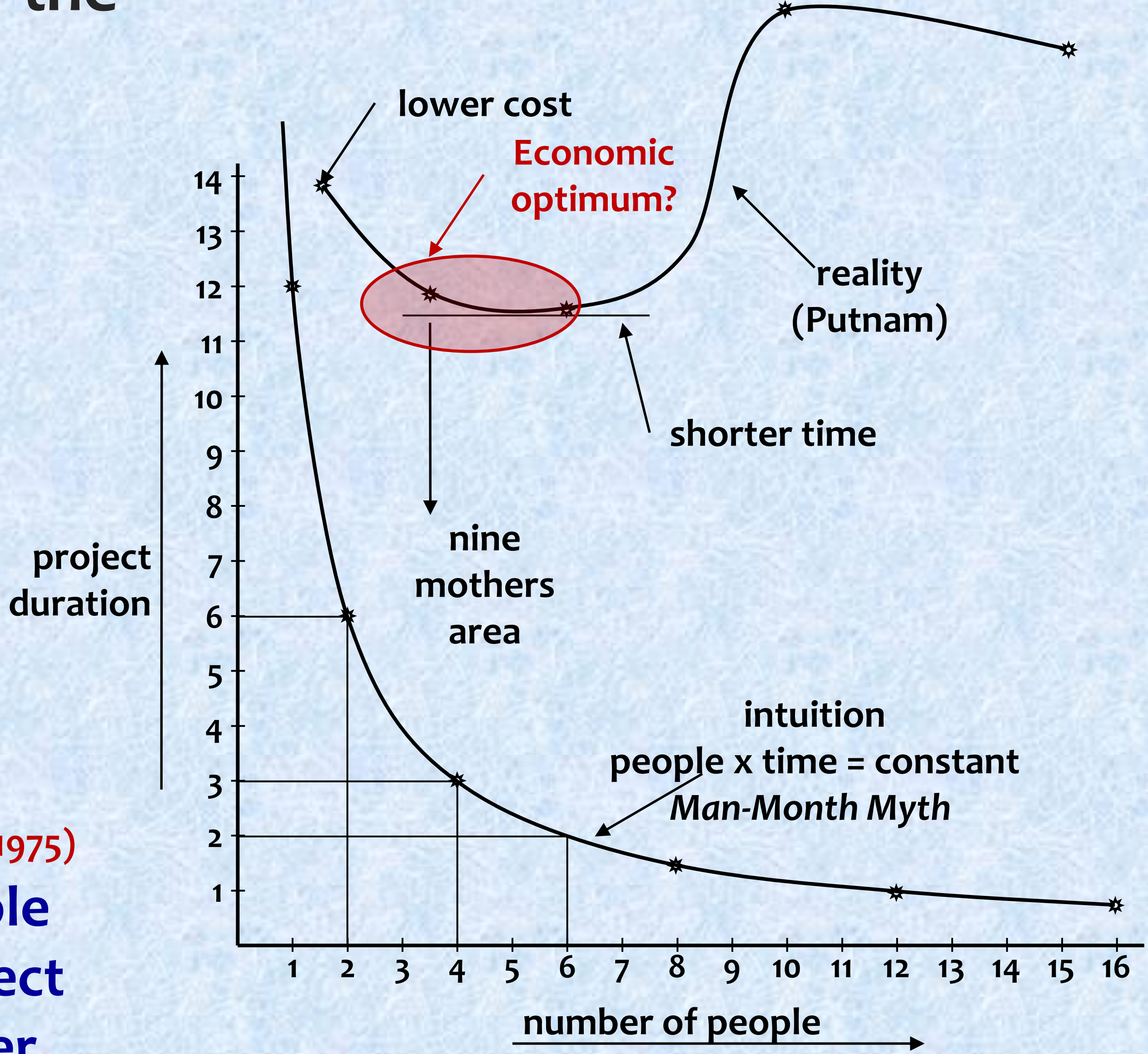
- **Hoping for the best** (fatalistic)
- **Going for it** (macho)
- **Working Overtime** (fooling ourselves)
- **Moving the deadline**
  - Parkinson's Law
    - Work expands to fill the time for its completion
  - Student Syndrome
    - Starting as late as possible,
      only when the pressure of the FatalDate is really felt

Adding people to a late project ...

makes it later

(Brooks' Law, 1975)

The Myth of the Man-Month



Brooks' Law (1975)

Adding people to a late project makes it later

Saving time

Continuous elimination of waste

We don’t have enough time, but we can save time ***without negatively affecting the Result !***

- **Efficiency in *what* (*why*, for *whom*) we do** - doing the right things
  - *Not* doing what later proves to be superfluous
- **Efficiency in *how* we do it** - doing things differently
  - The product
    - Using proper and most efficient solution, instead of the solution we always used
  - The project
    - Doing the same in less time, instead of immediately doing it the way we always did
  - Continuous improvement and prevention processes
    - Constantly learning doing things better and overcoming bad tendencies
- **Efficiency in *when* we do it** - right time, in the right order
- **TimeBoxing** - much more efficient than FeatureBoxing

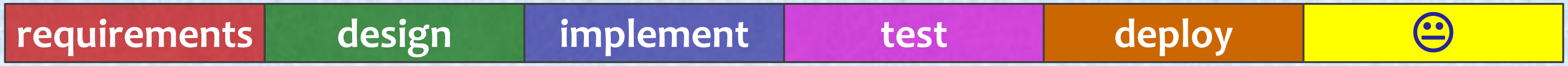
Delivery ?

Customer Relations Management project

- CRM system, original plan: 6 months and € 1M
- Spent 1.5 years and € 5M
- Business hasn't seen any result whatsoever
- Systems Integrator still "working hard"

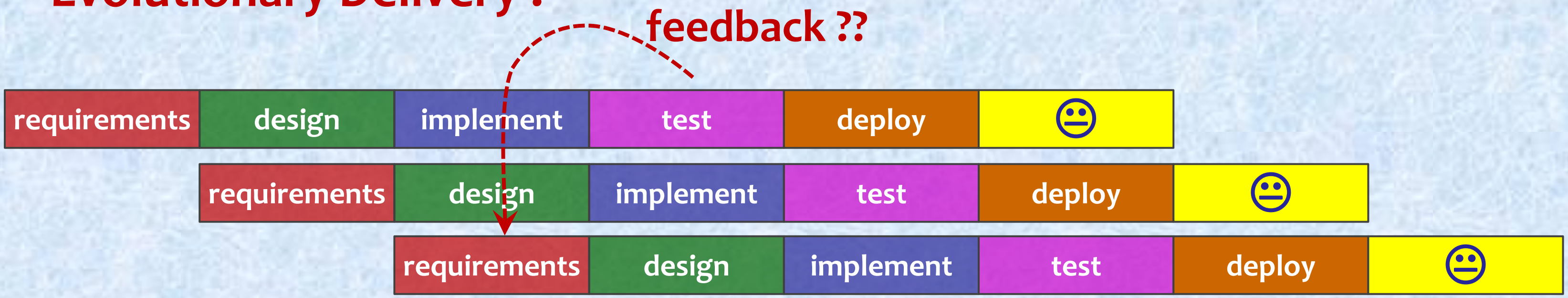
- New Project Manager, new System Integrator
- Started working in exactly the same fashion …

Delivery Requirements



requirements | design | implement | test | deploy | 😐

- Evolutionary Delivery ?



feedback ??

requirements | design | implement | test | deploy | 😐

requirements | design | implement | test | deploy | 😐

requirements | design | implement | test | deploy | 😐

- Suggested Requirements:
  1. Within one week of any delivery, the business is not *less* efficient than before
  2. The business decides whether they are satisfied
- "Unacceptable" means supplier is saying:
  1. Within one week of a delivery, the business will be less efficient than before
  2. The business will not be satisfied

Improving the world, start with yourself

- Define what your work is all about
- Don’t do things that will not be needed
- Challenge everything, however, in small steps
- Use Plan-Do-Check-Act
- Seek perfection

More

www.malotaux.nl/Booklets

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

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

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

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

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

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

6 Human Behavior in Projects (APCOSE 2008)
Human Behavioral aspects of Projects

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

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

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

www.malotaux.nl/nrm/Insp

Inspection pages

My Practical Approaches for Becoming Lean and Really Agile

Predictable Projects

Niels Malotaux

N R Malotaux
Consultancy

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

Random unsorted slides

Not used because of time constraints

Niels Malotaux

N R Malotaux
Consultancy

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

Overproduction

- Every part of the total process produces just enough to keep their standard inventory
- Overproduction is producing more than the standard inventory
  - Detailing requirements too early, may make them obsolete before use
  - In Evo we determine what we will have done before doing the work, so the previous and the next process know when we may become idle and when we will be ready
  - Required production is determined by the market

Inspection

- **Inspection to *find* defects is waste** (non-value added waste)
- **Inspection to *prevent* defects is required** (value added waste)

Testing

- Final tests shouldn’t find any defects
- If you routinely find defects at final testing, then you are testing too late

Poppendieck Implementing Lean Software Development, p 88

Value stream mapping exposes sources of waste

- Churn
  - Requirements churn is a symptom of writing requirements too early
  - or Requirements being detailed too early
  - or Having a too long process
  - Test and fix churn indicates that tests are developed and run too late
- Delays
  - Long queues, by too much work being dumped into the organization
  - Hand-off to organization not ready for it (eagerly waiting principle)
  - Arduous approval process
- Extra work
  - Failure to synchronize
  - Lack of involvement of operations and support

Efficiency bad for personnel ?

- Efficiency is merely doing the work in the best way rather than in the worst way
- Would you like the people to do the work in a bad way ?

What is the most important Requirement ?

- Delivery Time is a Requirement,
  like all other Requirements
- How come most projects are late ???
- Apparently all other Requirements
  are more important than Delivery Time
- Are the really?

Fallacy of '*all*' requirements

- "We're done when *all* requirements are implemented"
- Isn't delivery time a requirement ?
- Requirements are always *contradictory*
- Perception of the requirements
- Who's requirements are we talking about ?
- Do we really know the *real* requirements ?
- Are customers able to define requirements ?
  - Customers specify things they do not need
  - And forget things they do need
  - They're even less trained in defining requirements than we are
- What we think we have to do should fit the available time
- Use the Business Case

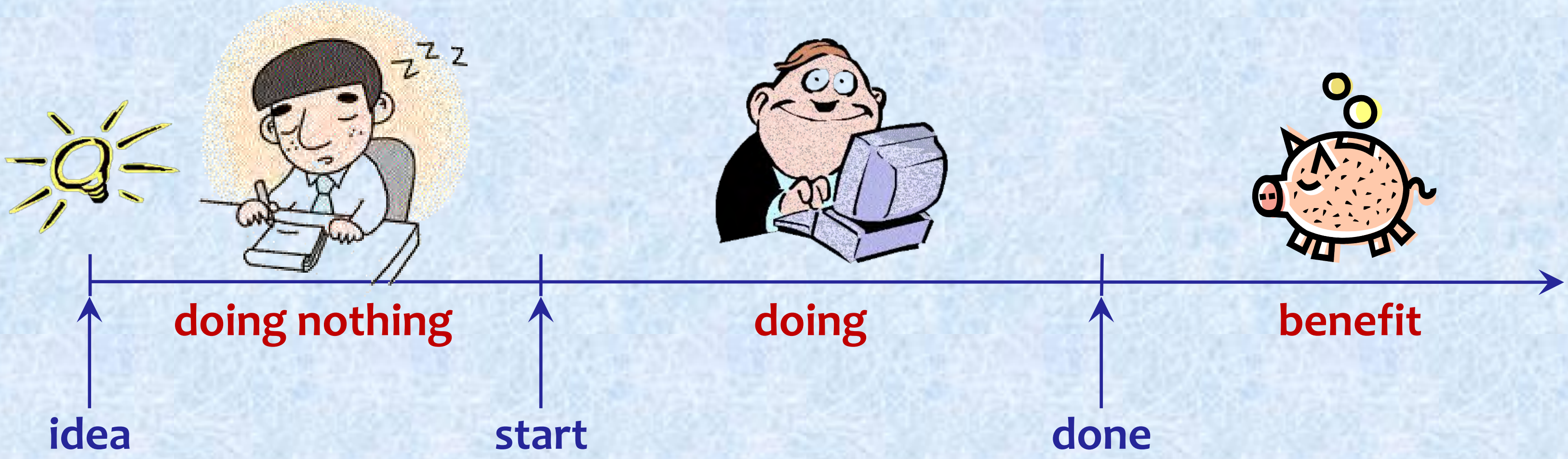
**If our previous project was late,
our current project will also be late**

unless we do things *differently* and *better*

**If we don't learn from history,
we are doomed to repeat it**

**Projects don't have to be late
They deserve better**

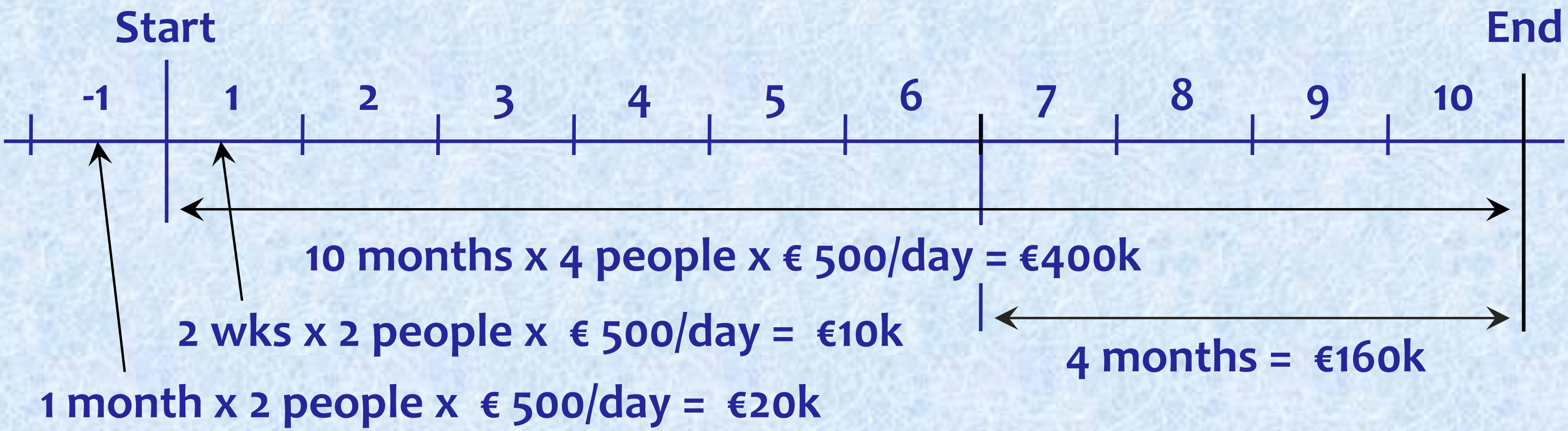
Project ROI



Return on Investment (ROI)

+ **Benefit of doing** - huge (otherwise other projects would be more rewarding)
– **Cost of doing** - project cost, usually minor compared with other costs
– **Cost of doing nothing** - every day we start later, we finish later
– **Cost of being late** - lost benefit

The Cost of Time



10 months x 4 people x € 500/day = €400k

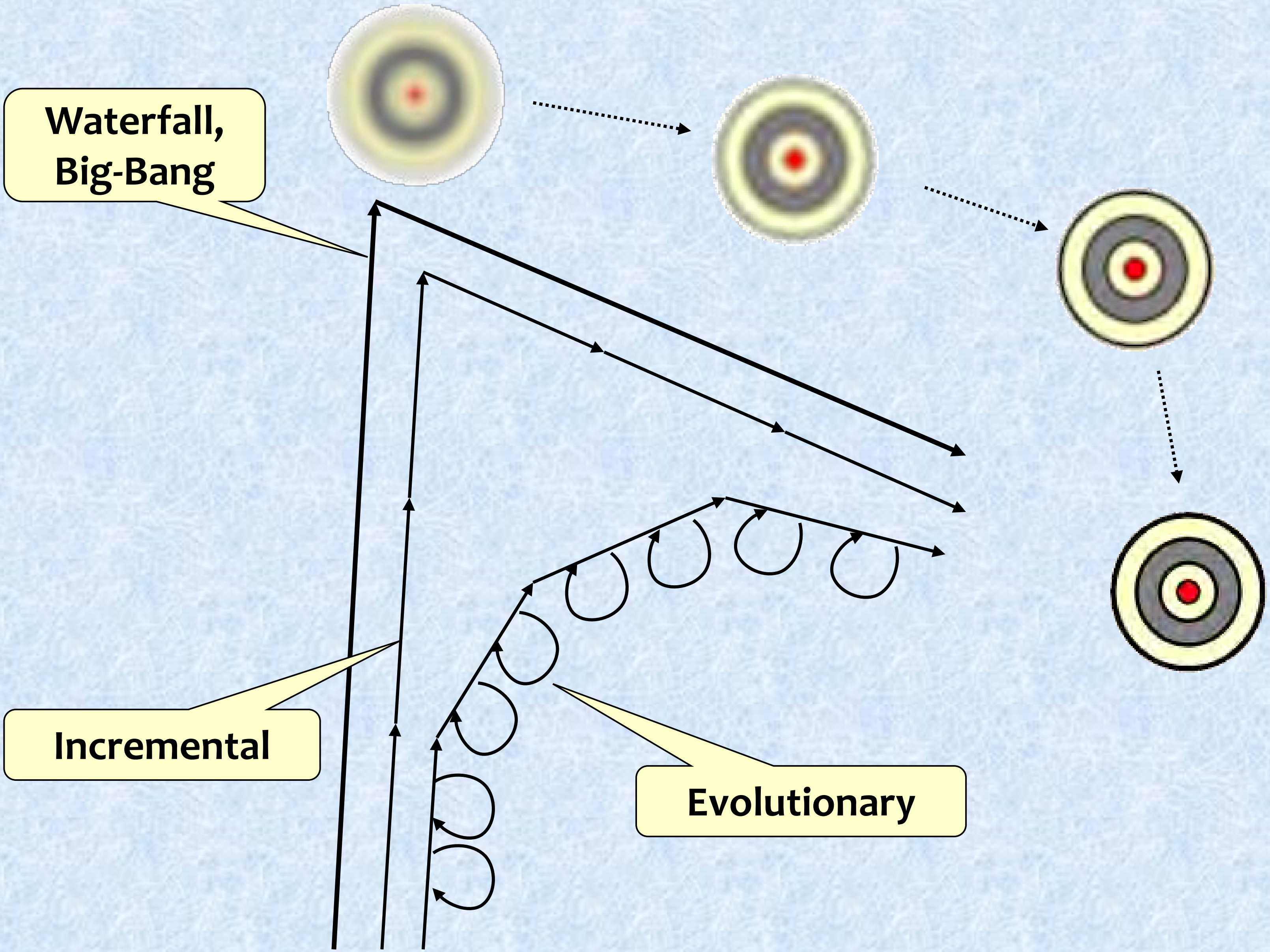
2 wks x 2 people x € 500/day = €10k

4 months = €160k

1 month x 2 people x € 500/day = €20k

- We can save 4 months by investing €200k → "That's too much !"
- It's a *nicer* solution - Let's do 2 weeks more research on the benefits
- What are the expected revenues when all is done? → €16M/yr (1.3M/mnd)
- So 2 weeks extra doesn't cost €10k, but rather €16M/24 = €670k
- And saving 4 months brings €16M/3 = €5M extra

➔ Invest that €200k *NOW* and *don't waste time* !

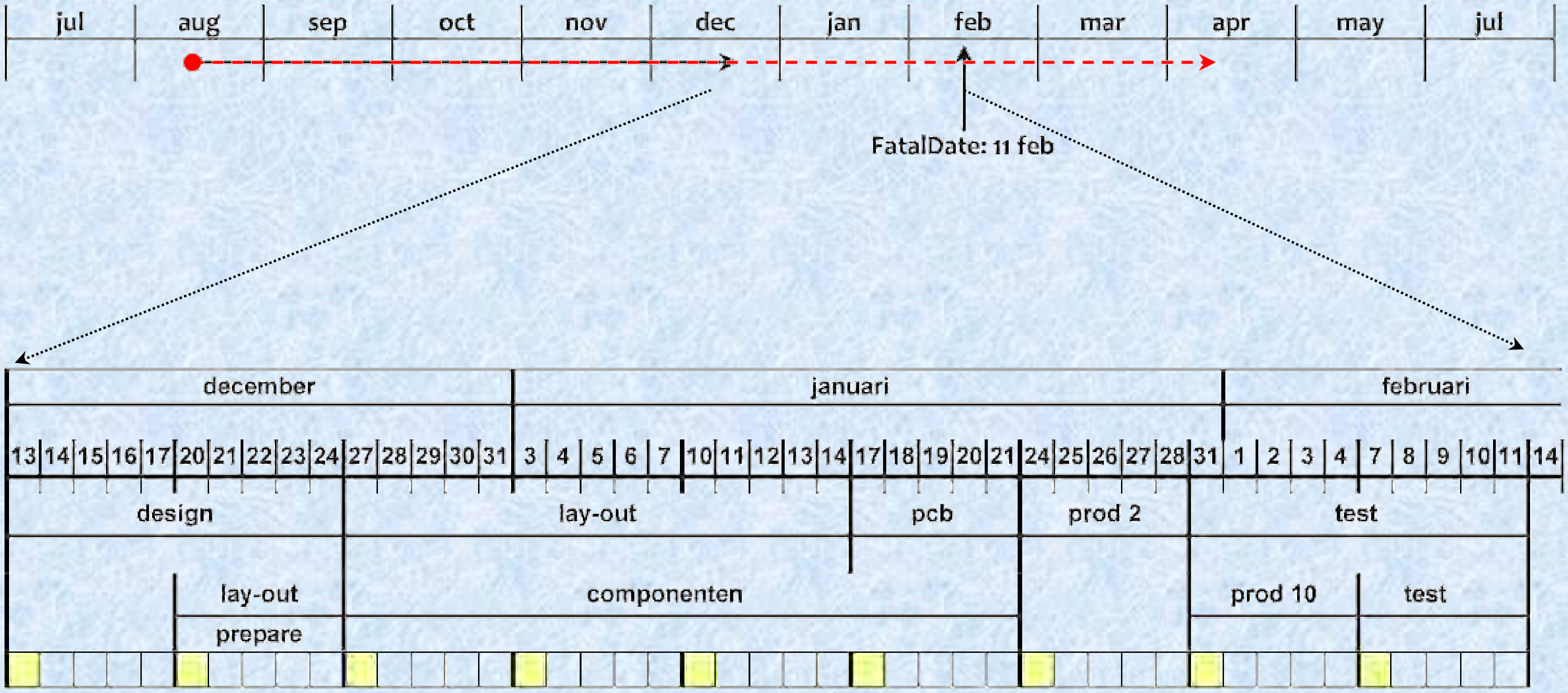


Waterfall, Big-Bang

Incremental

Evolutionary

TimeLine planning



jul aug sep oct nov dec jan feb mar apr may jul

FatalDate: 11 feb

december januari februari

13 14 15 16 17 20 21 22 23 24 27 28 29 30 31 3 4 5 6 7 10 11 12 13 14 17 18 19 20 21 24 25 26 27 28 31 1 2 3 4 7 8 9 10 11 14

design lay-out pcb prod 2 test

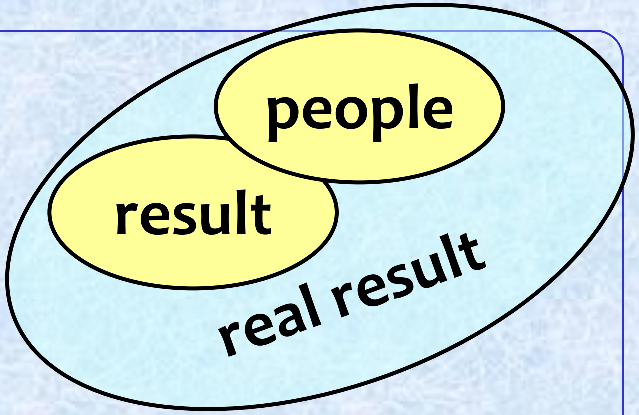
lay-out prepare componenten prod 10 test

Higher Productivity

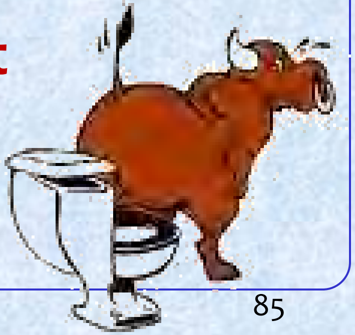
- All functionality we produce *does already exist*
- The real reason for running our projects is creating *better performance*

- Types of improvement:
  - Less loss
  - More profit
  - Doing the same in shorter time
  - Doing more in the same time
  - Being happier than before
- In short: *Adding Value*

Stakeholders are people



- Every project has some 30±20 Stakeholders
- Stakeholders have a stake in the project
- The concerns of Stakeholders are often contradictory
  - Apart from the Customer *they don't pay*
  - So they *have no reason to compromise !*
  - In many cases, finally, *we all pay*
- Some Stakeholders are victims of the project
  - They have no reason for the project to succeed, on the contrary
- Project risks, happening in almost every project
- No excuse to fail !



What are the Requirements for a Project ?

- Requirements are what the Stakeholders require

but for a project ...

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

- The set of Stakeholders doesn’t change much
- Do you have a checklist of possible Stakeholders ?

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?

Requirements carved in stone ?

- *We* don't know the real requirements
- *They* don't know the real requirements
- Together we'll 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

5 times "Why?" technique

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

- **Freud and Jung:**
  - Problems are in our sub-consciousness
  - Solutions pop up
  - Solutions are how people tell their problems
- **What's your problem ?**
  - If there's no problem, we don't have to do something
- **Within 5 times "Why?"**
  we usually come down to the real problem to solve
  - Otherwise we will be perfectly solving the wrong problem

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) ↓

Design and requirements

- Design:
  Finding the best compromise between the conflicting requirements

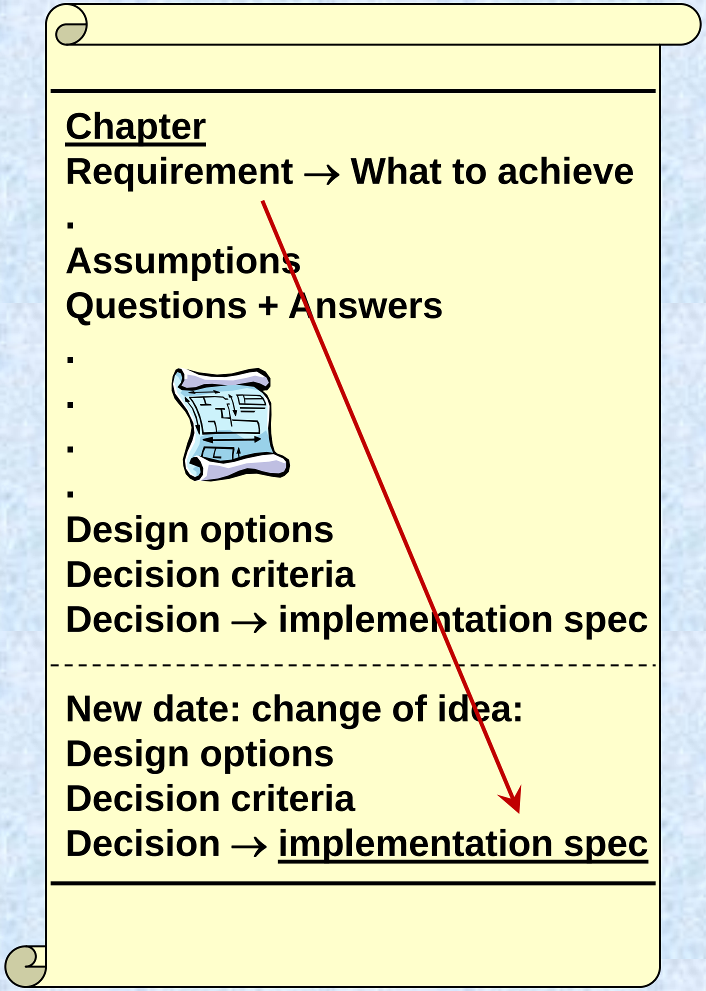
- All requirements are equal,
  but some are more equal than the others
- Some aren’t really requirements
- Some elements will never be used
- Some requirements are incorrect
- A lot of real requirements are unexplored

~~MoSCoW~~ ?

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
  - Requirement
  - Assumptions
  - Questions
  - Available techniques
  - Calculations
  - Choices + reasoning:
    - If rejected: why?
    - If chosen: why?
- Rejected choices
- Final (current) choices
- Implementation



Chapter

Requirement → What to achieve

.

Assumptions

Questions + Answers

.

.

.

.

Design options

Decision criteria

Decision → implementation spec

New date: change of idea:

Design options

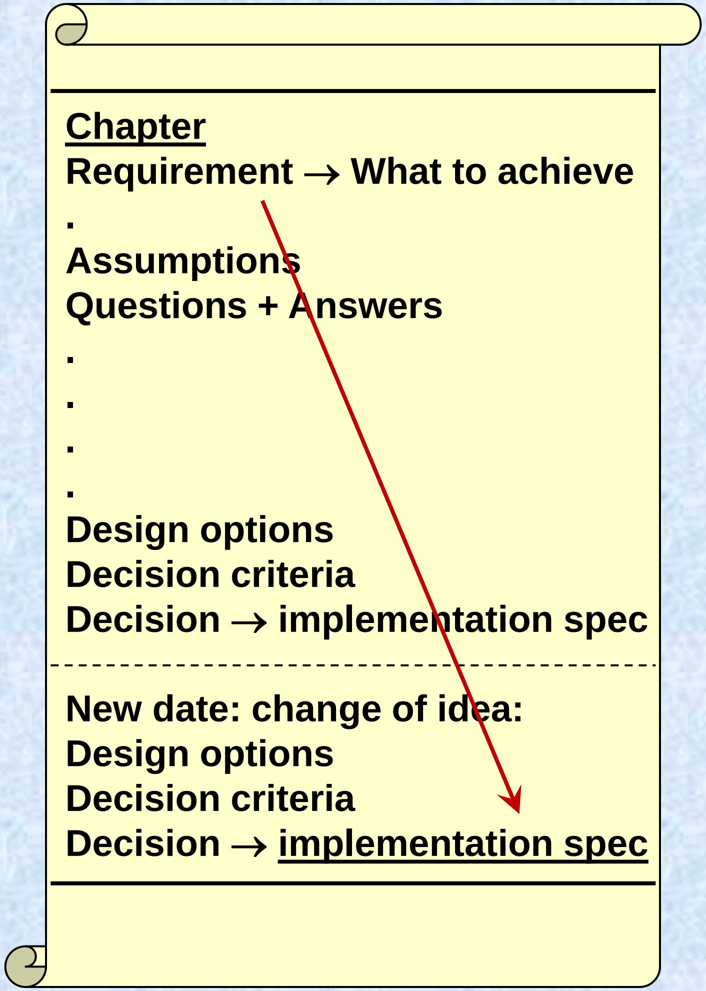
Decision criteria

Decision → implementation spec

ProcessLog

(department / organization level)

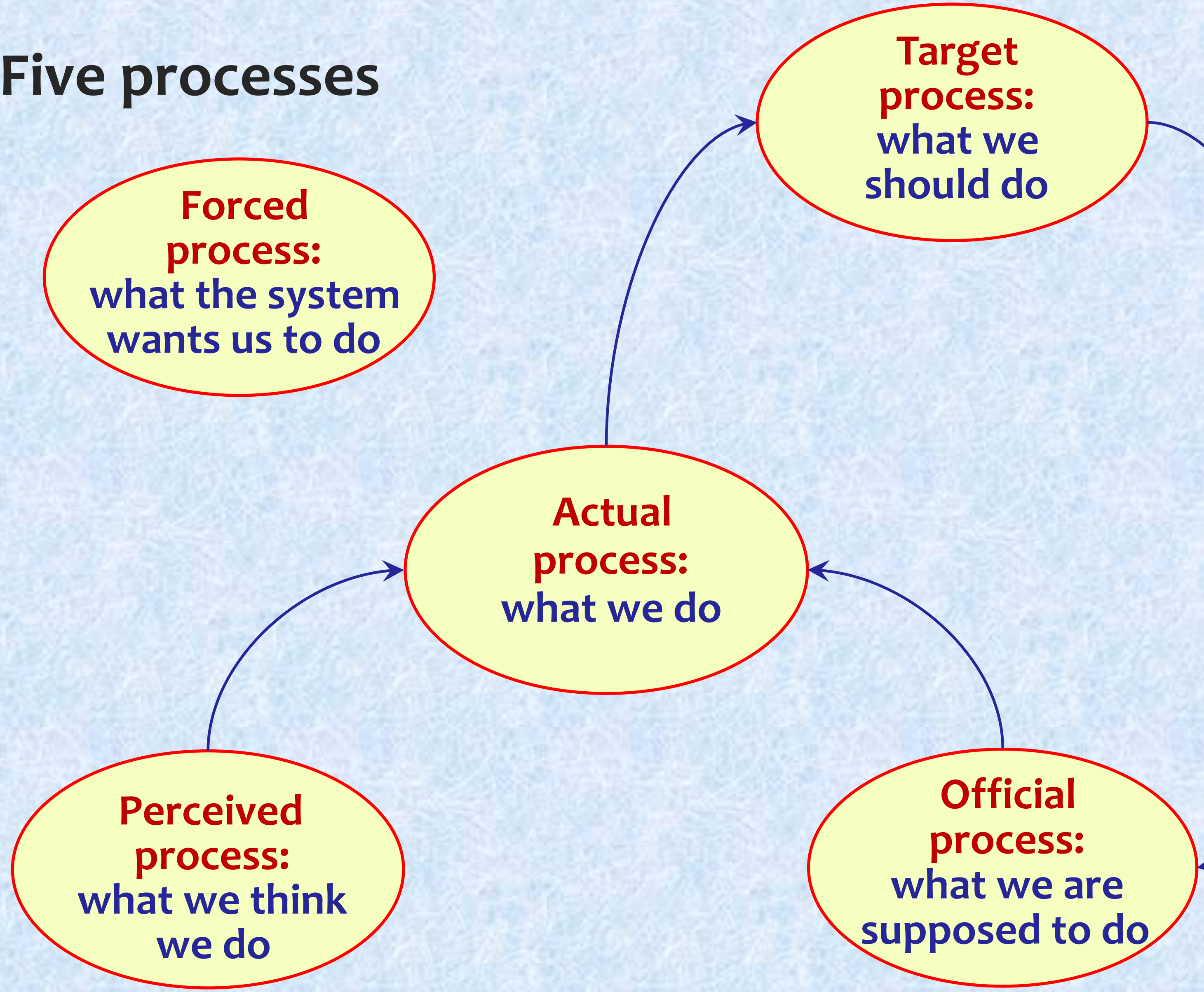
- In computer, not loose notes, not in e-mails, not handwritten
  - Text
  - Graphics (drawings)
  - On subject order
  - Initially free-format
  - For all to see
- All concepts contemplated
  - Requirement
  - Assumptions
  - Questions
  - Known techniques
  - Choices + reasoning :
    - If rejected: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



Chapter
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

Five processes


Forced process: what the system wants us to do
Target process: what we should do
Actual process: what we do
Perceived process: what we think we do
Official process: what we are supposed to do

Poppendieck p118

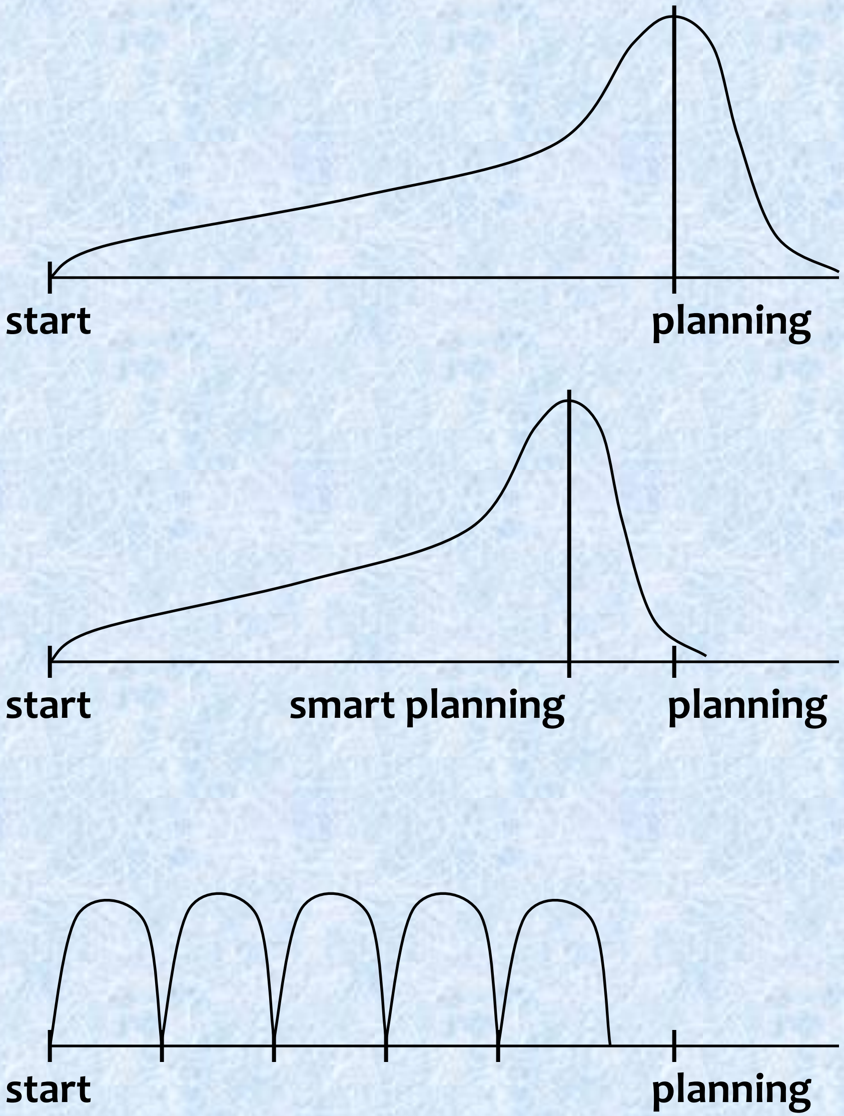
- Delays were caused because:
- People were not working together
- People were not looking for problems early enough
- People were not paying attention when a problem arose
- People were not communicating with each other promptly and effectively

- No secrets: share early and often
- Test early - fail fast

Popp p134

- **If you expect teams to meet aggerssive deadlines, you must limit work to capacity**

Development cycles



It’s just common sense

- Bicycle factory – cyclic production volumes
- Originally put new machines at space available
- Put machinery in flow order
- Seemed very quiet while producing at full capacity
- That was ~1960

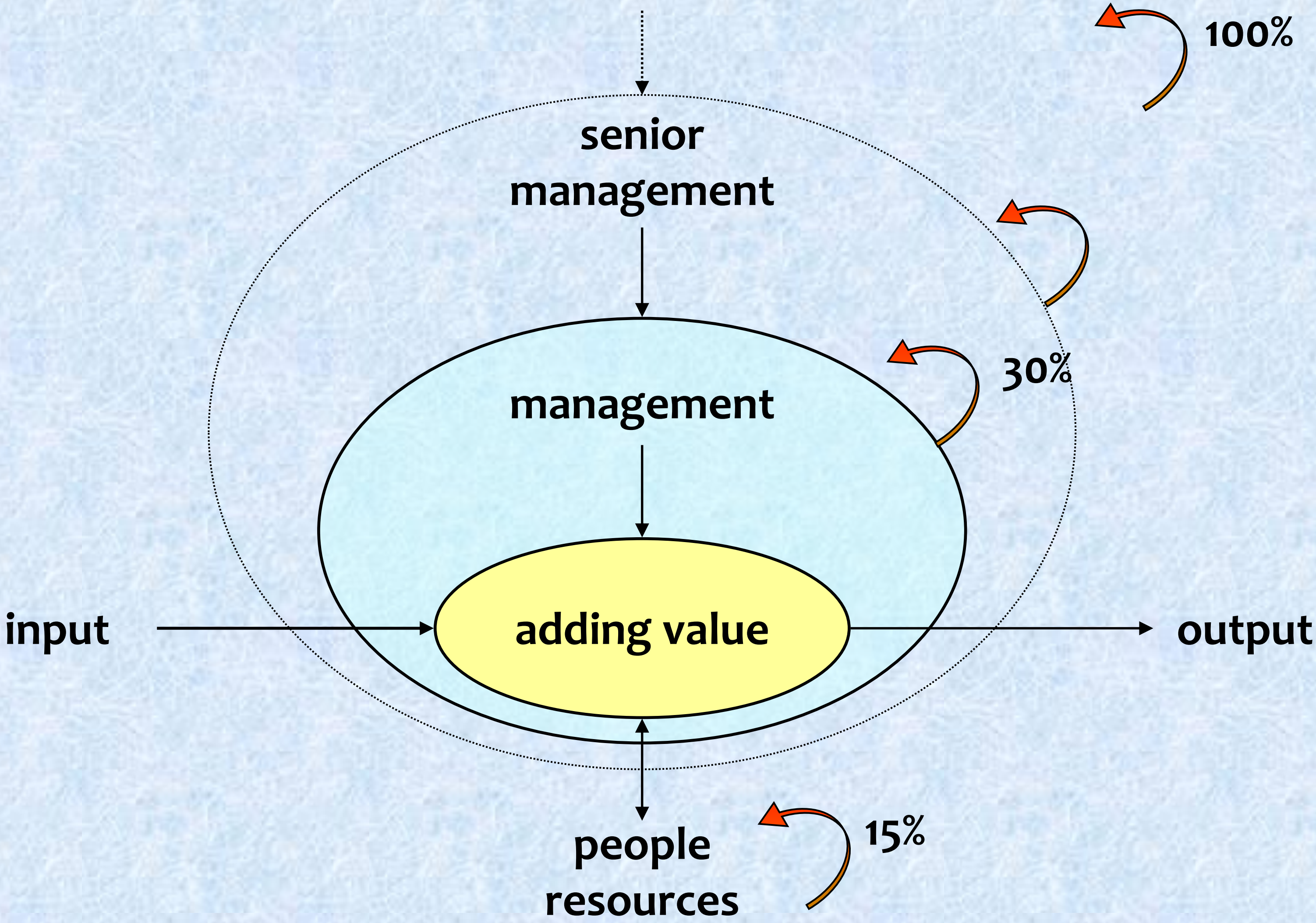
- Common sense seems to be not so common

Managers have to learn

- Managers *facilitate* their people to be successful
- Managers should be coaches
- Not police
- Managers and workers have to understand the *essence*

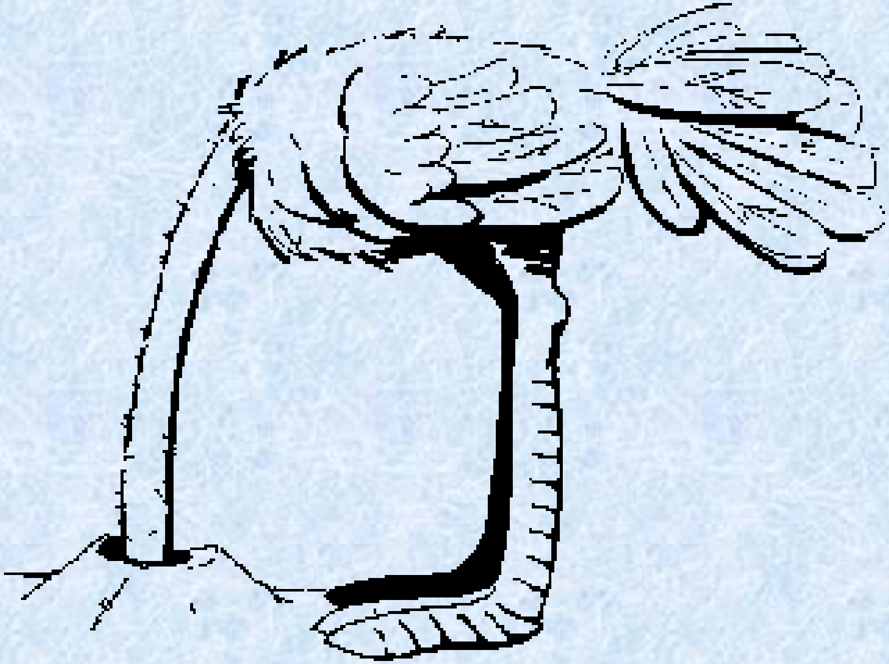
- The most important and difficult issue of Lean is *to involve everyone*
- And to keep everyone involved

The managers task



People responsible for success

- *During* the project
  - Can still influence the performance of the project
  - First responsibility of the Project Manager
  - Actually responsibility of the whole development organization
- *After* the project, once the system is out there
  - No influence on the performance of the system any more
  - System must perform autonomously and Lean for the business
  - So the Lean performance must be there *by design*
  - Including appropriate interface with humans
  - Responsibility and required skill of Systems Engineering



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

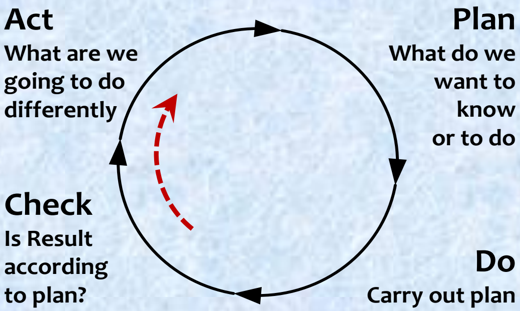
It can’t be done - it must be done

- It can’t be done
  - Management doesn’t allow it
  - “They” won’t do it
  - It’s impossible
- It must be done
  - How are we going to do it

→ Stuck in the *Check Phase*

→ Moving to the *Act phase*

- We don’t let others make us fail


Act
What are we going to do differently
Plan
What do we want to know or to do
Check
Is Result according to plan?
Do
Carry out plan

Effort and Lead Time

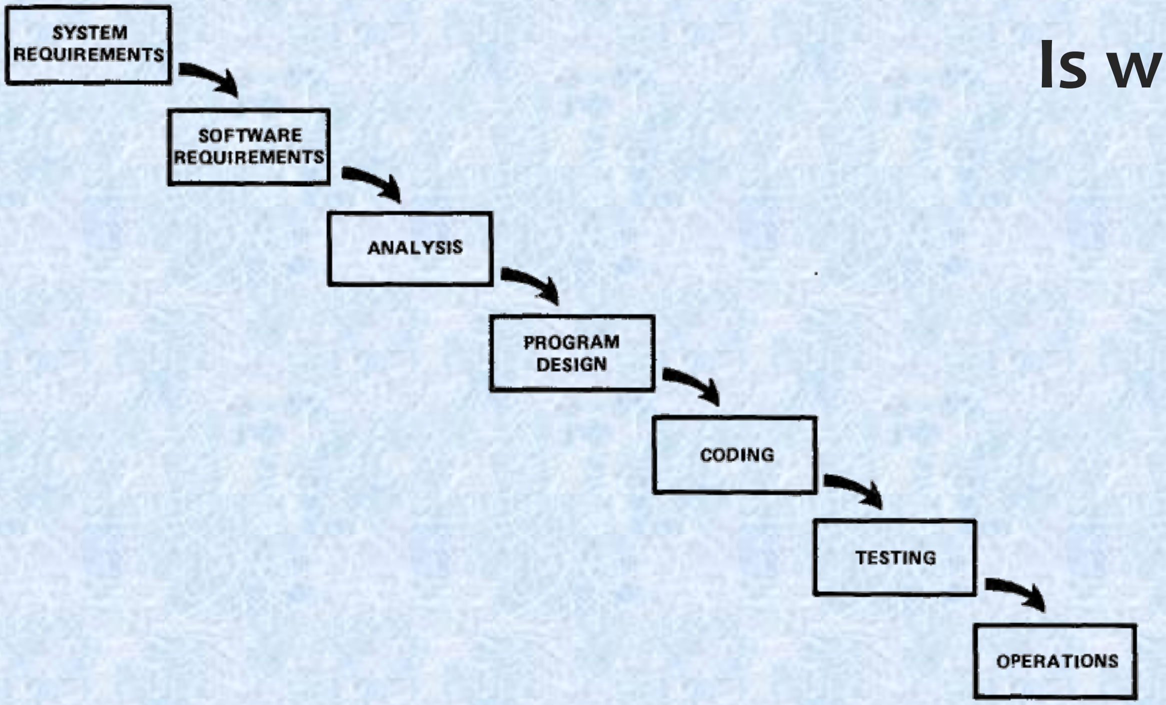
- Days estimation → lead time (calendar time)
- Hours estimation → effort

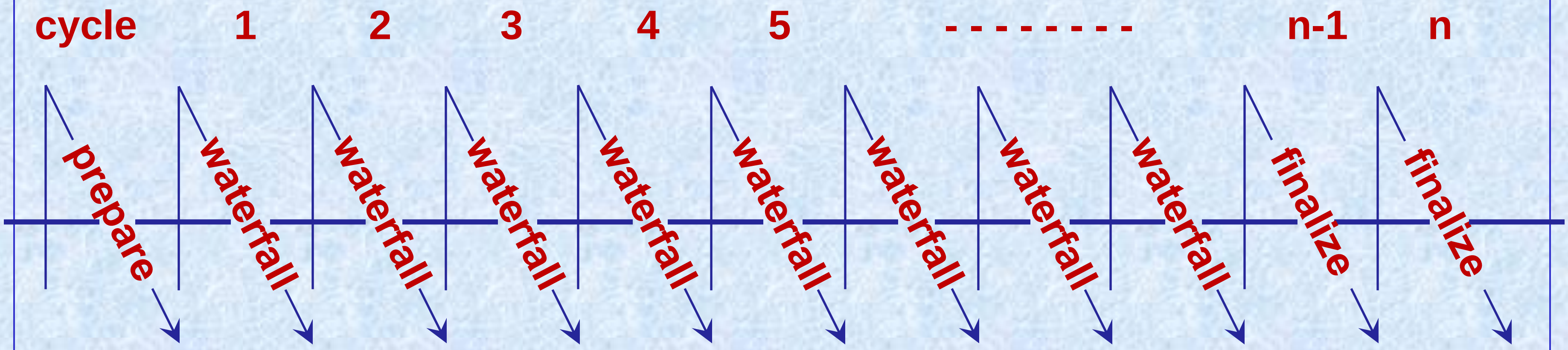
- Effort variations and lead time variations have different causes
- Treat them differently and keep them separate
  - Effort: complexity
  - Lead Time: time-management
    - (effort / lead-time ratio)

Weekly 3-Step Procedure

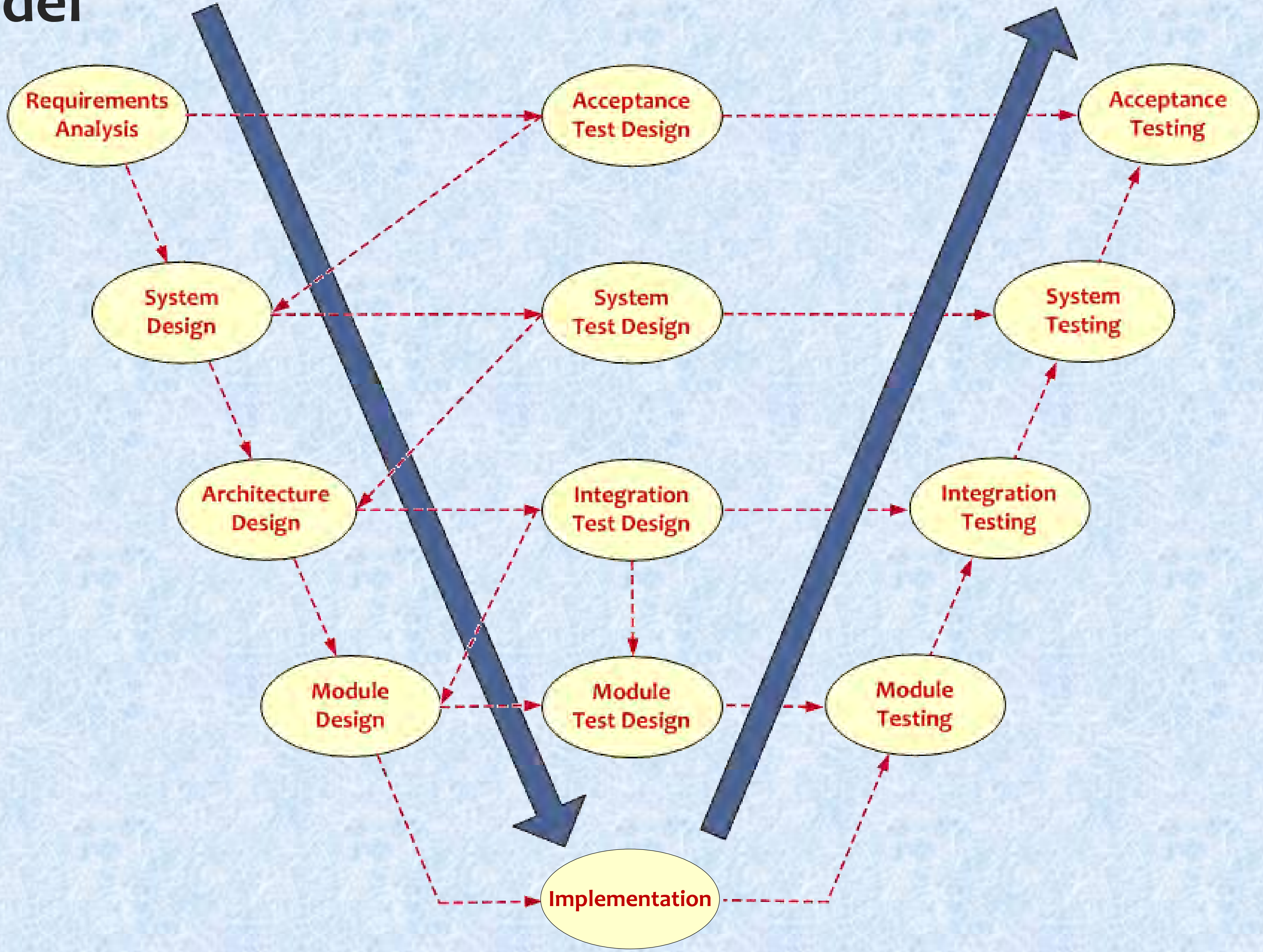
- Individual preparation
  - Conclude current tasks
  - What to do next
  - Estimations
  - How much time available
- Modulation with / coaching by Project Management
  - Status
  - Priority check
  - Feasibility
  - Commitment and decision
- Synchronization with group (team meeting)
  - Formal confirmation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

Is waterfall wrong ?

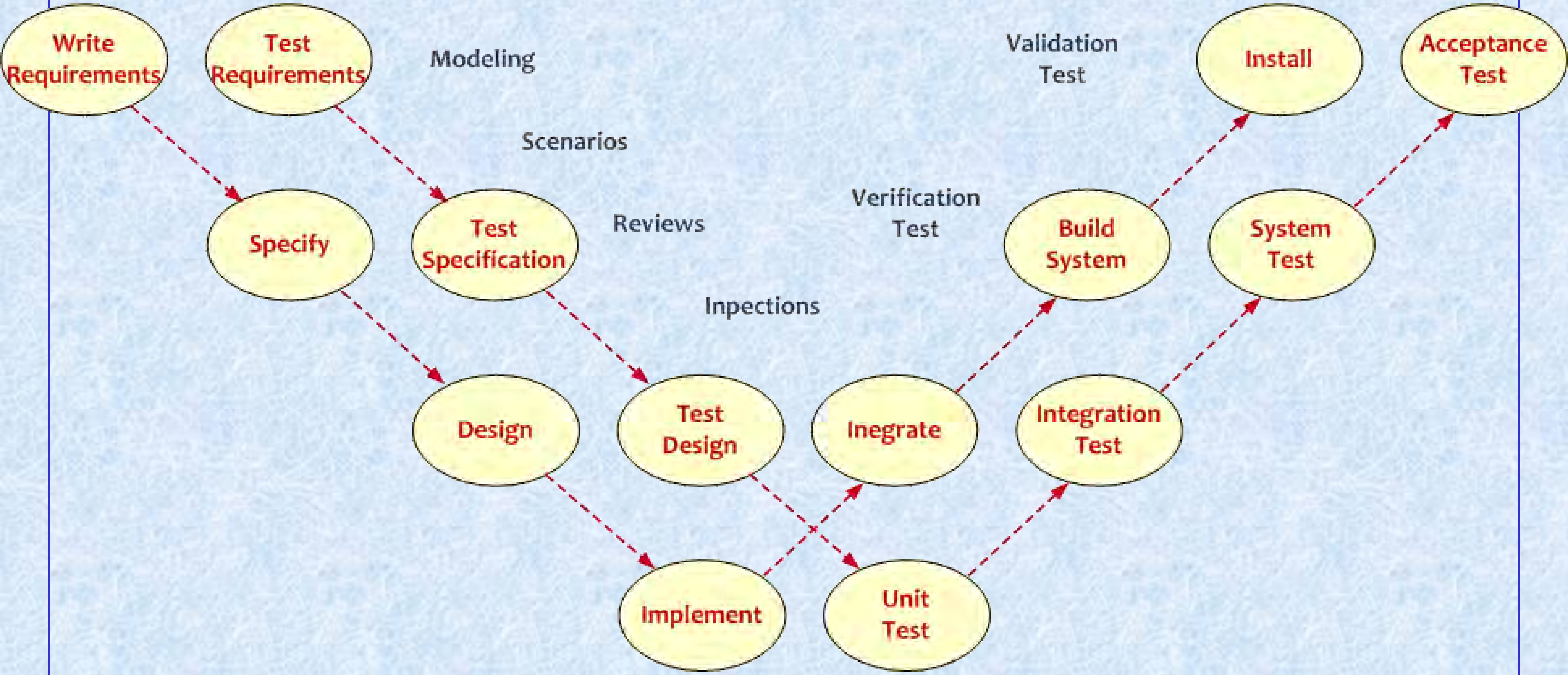




V-Model

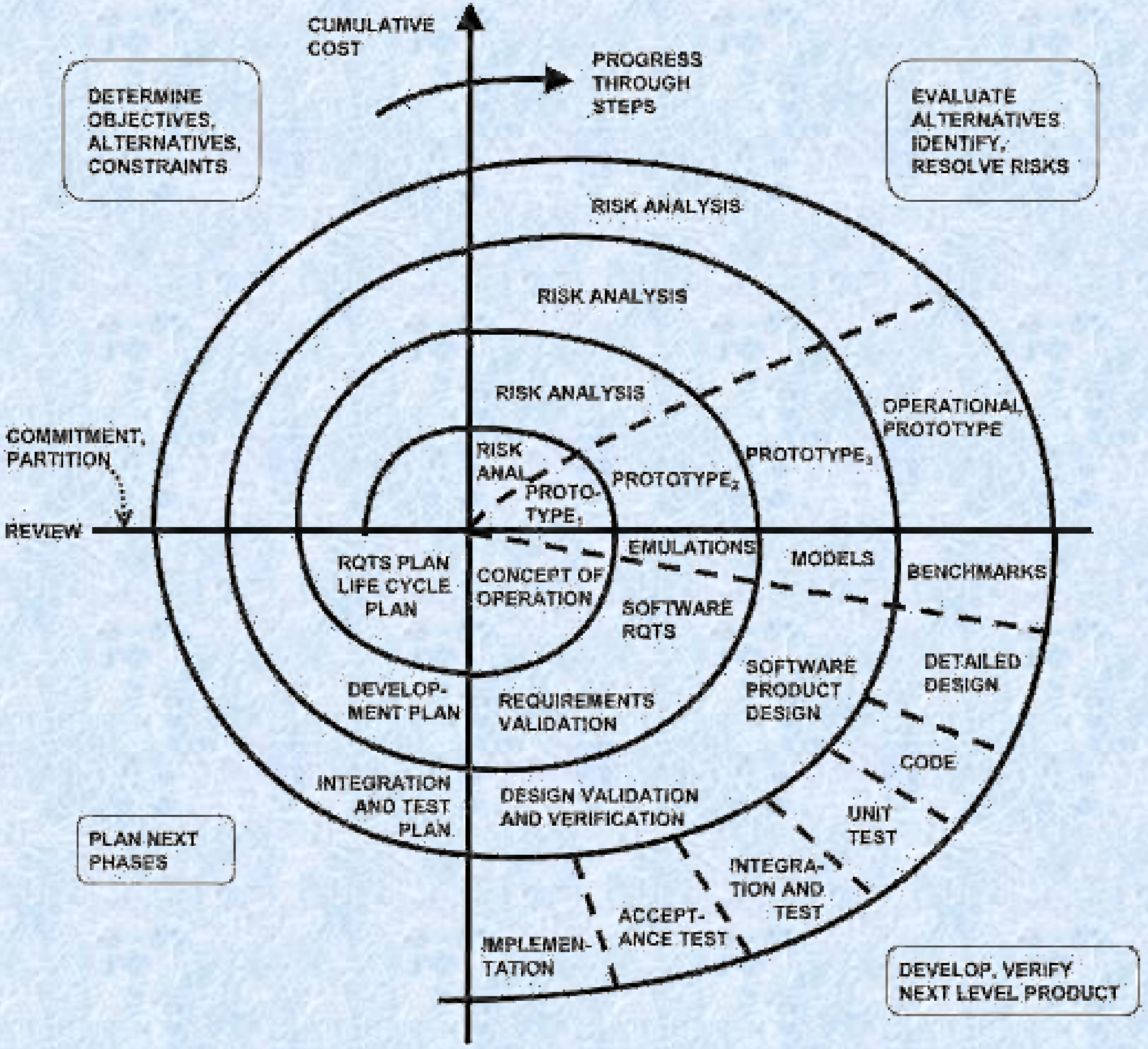

Requirements Analysis
Acceptance Test Design
Acceptance Testing
System Design
System Test Design
System Testing
Architecture Design
Integration Test Design
Integration Testing
Module Design
Module Test Design
Module Testing
Implementation

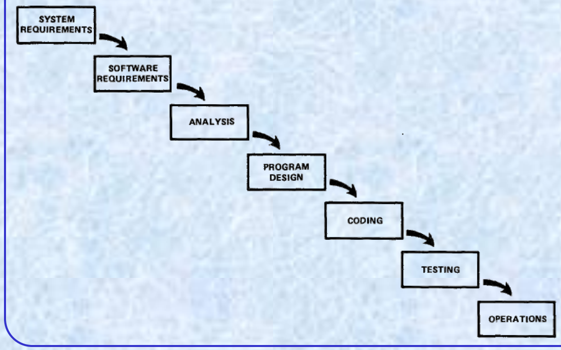
W-model


Write Requirements
Test Requirements
Modeling
Validation Test
Install
Acceptance Test
Scenarios
Specify
Test Specification
Reviews
Verification Test
Build System
System Test
Inpections
Design
Test Design
Inegrate
Integration Test
Implement
Unit Test

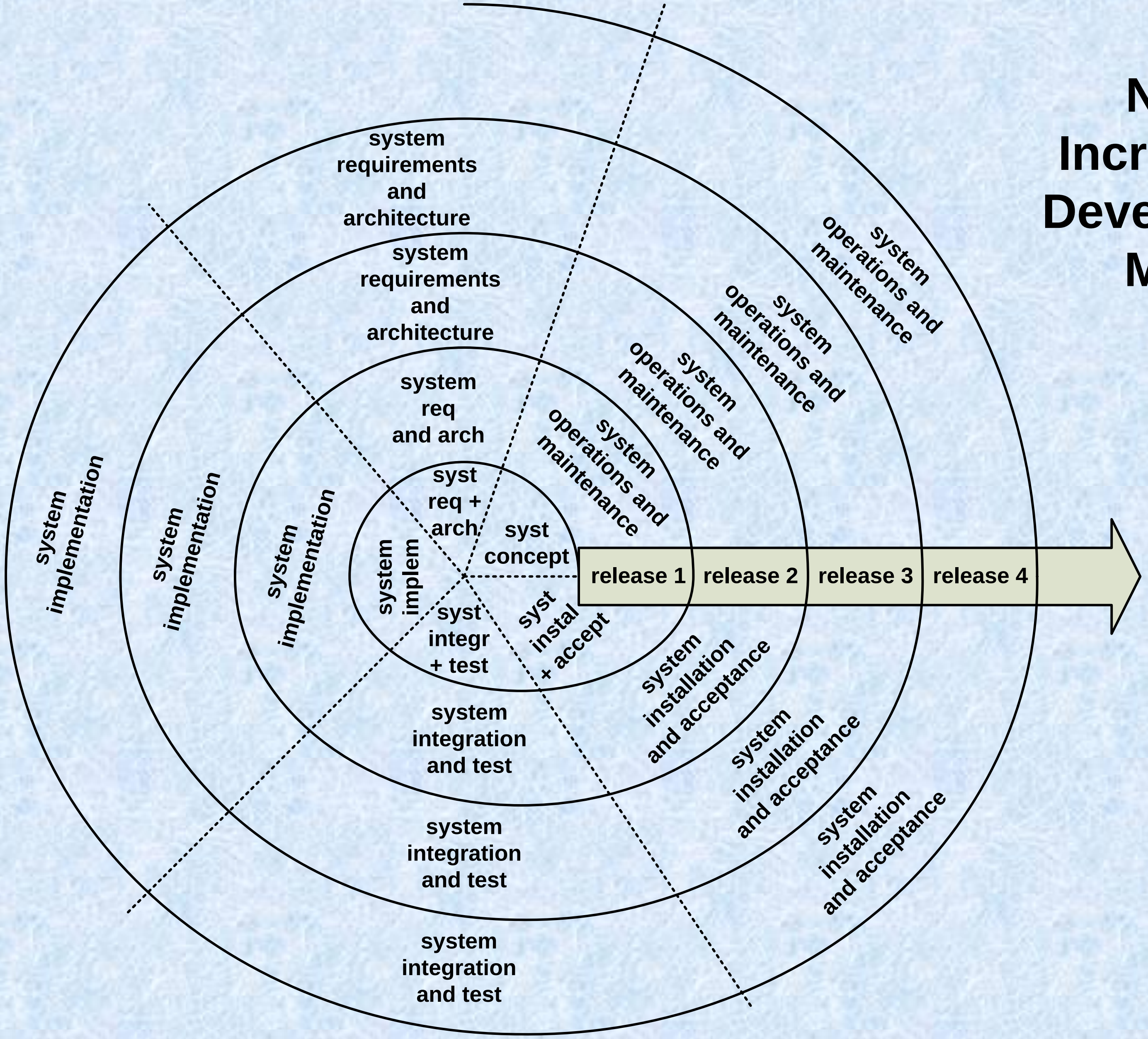
Spiral Process model

(Boehm 88)





NASA Incremental Development Model

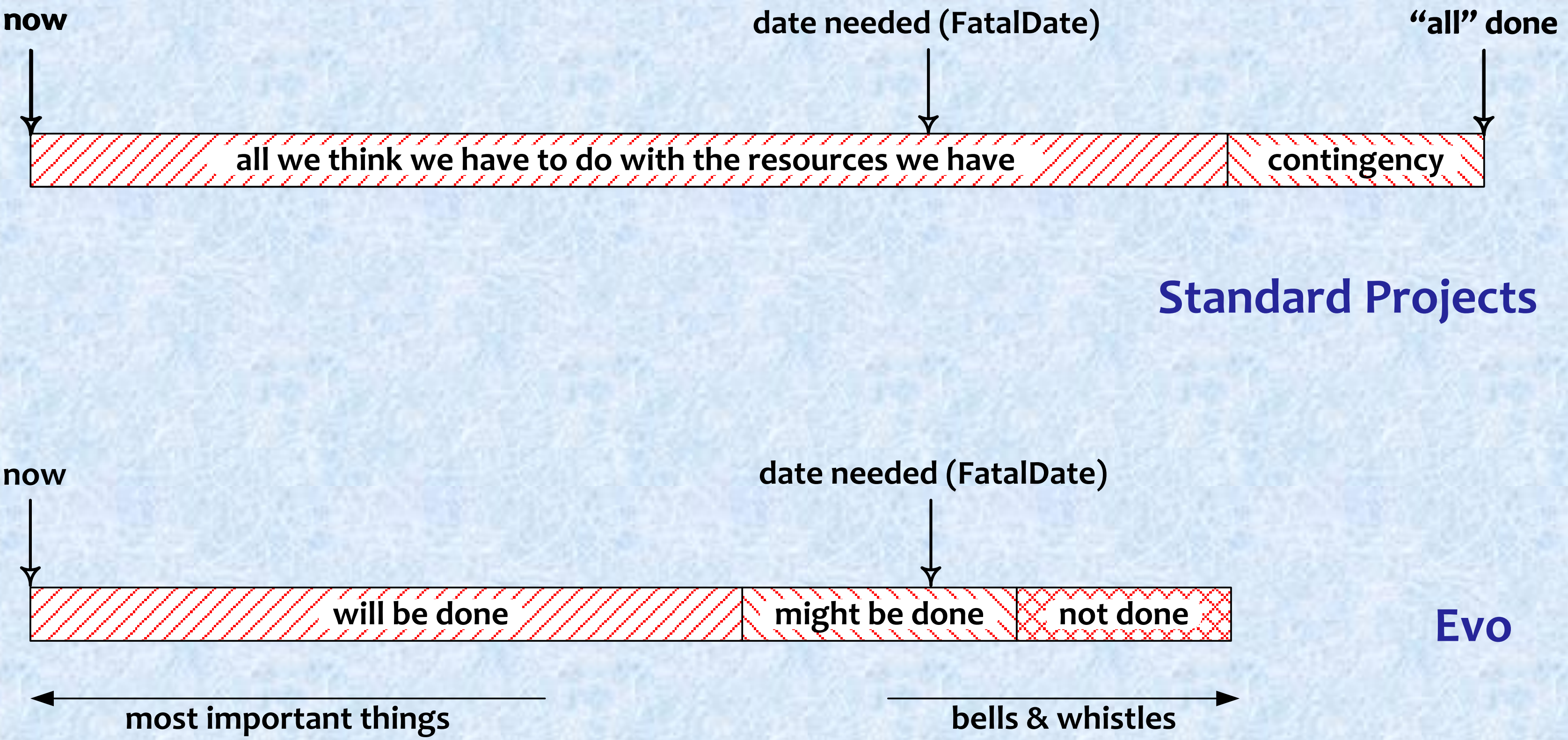

system requirements and architecture
system requirements and architecture
system req and arch
syst req + arch
syst concept
system operations and maintenance
system operations and maintenance
system operations and maintenance
system operations and maintenance
system implementation
system implementation
system implementation
system implem
release 1
release 2
release 3
release 4
syst integr + test
syst instal + accept
system installation and acceptance
system installation and acceptance
system installation and acceptance
system integration and test
system integration and test
system integration and test

All Models are wrong

Some are useful

TimeLine

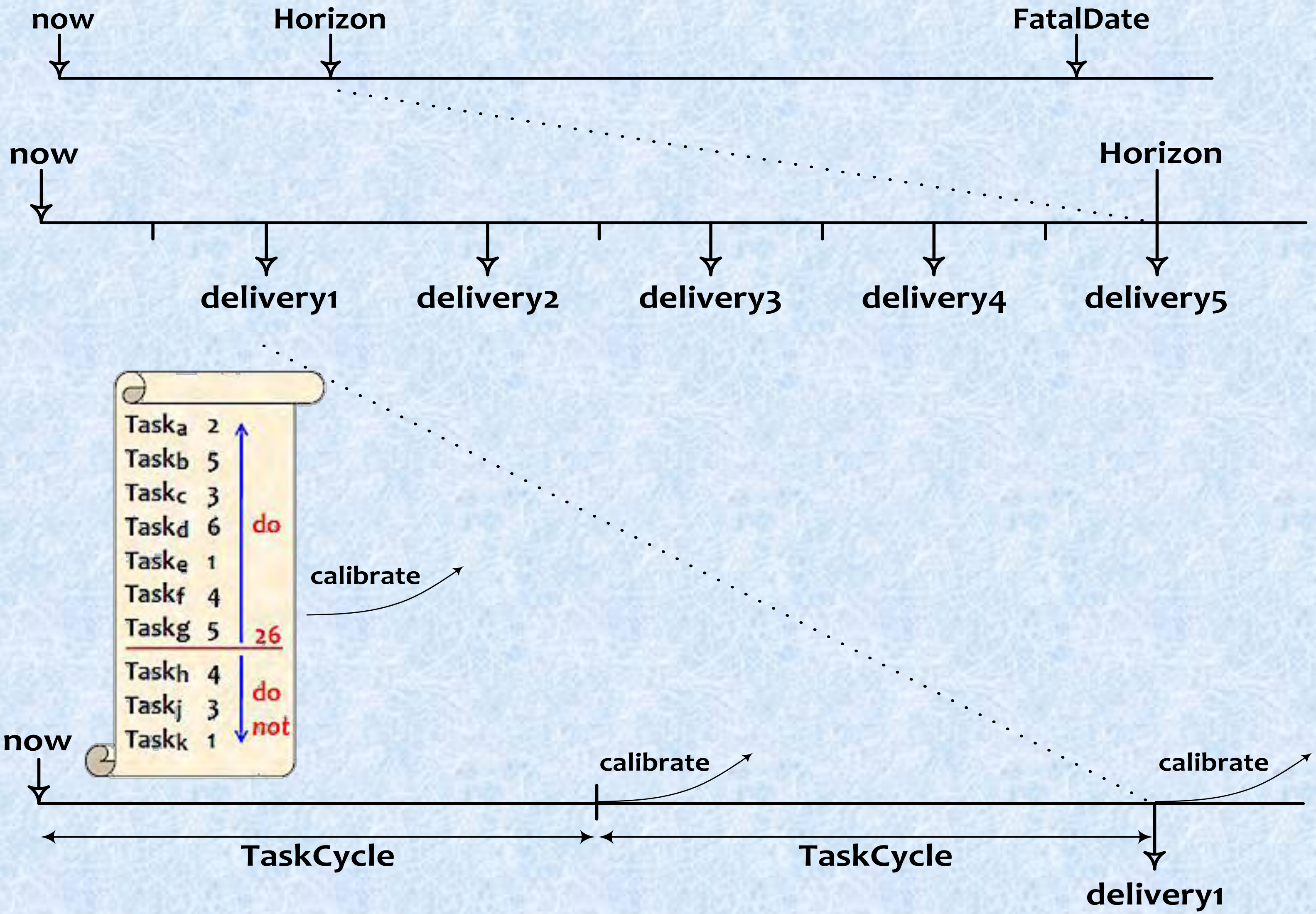
What the customer wants, he cannot afford



Standard Projects

Evo

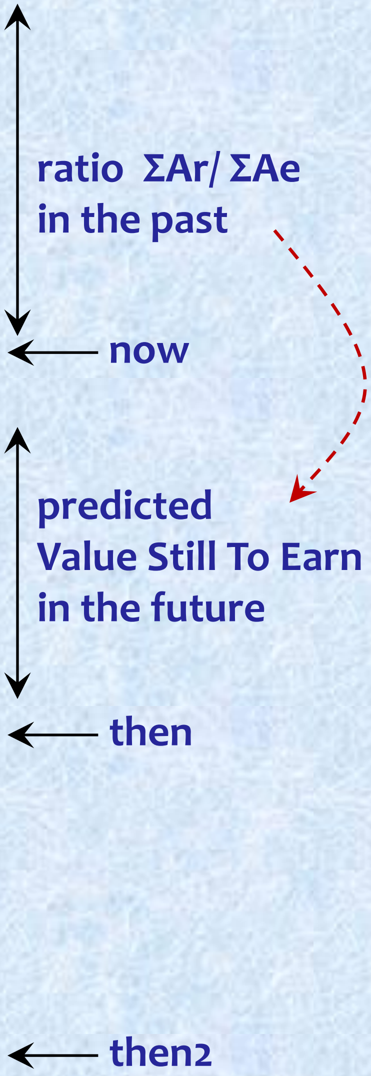
Result to Tasks and back



Task$_a$	2	
Task$_b$	5	
Task$_c$	3	
Task$_d$	6	do
Task$_e$	1	
Task$_f$	4	
Task$_g$	5	26
Task$_h$	4	
Task$_j$	3	do
Task$_k$	1	not

Calibration

Activity	Estimate	Real
Act1	Ae1	Ar1
Act2	Ae2	Ar2
Act3	Ae3	Ar3
Act4	Ae4	Ar4
Act5	Ae5	Ar5
Act6	Ae6	Ar6
Act7	Ae7	Ar7
Act8	Ae8	Ar8
Act9	Ae9	Ar9
Act10	Ae10	Ar10
Act11	Ae11	
Act12	Ae12	
Act13	Ae13	
Act14	Ae14	
Act15	Ae15	
Act16	Ae16	
Act17	Ae17	
Act18	Ae18	
Act19	Ae19	
Act20	Ae20	
Act21	Ae21	
⋮	⋮	
⋮	⋮	
⋮	⋮	
Act…	Ae…	



ratio ΣAr/ ΣAe in the past

now

predicted Value Still To Earn in the future

then

then2

Calibration Factor

$$\frac{\sum_{now-1}^{now-n} Ar}{\sum_{now-1}^{now-n} Ae}$$

Value Still To Earn

$$\textbf{Calibration Factor} * \sum_{now}^{then} Ae$$

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

Perception

- Quick, acute, and intuitive cognition (www.M-W.com)
- 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!

Systems Engineering

- Other Engineering (?)
  - Silo thinking
  - Sub-optimizing
  - Gold plating (hobbies)
  - Little attention to interfaces
  - Projects are always *multidisciplinary*

- Systems Engineering
  - Multi-dimensional thinking
  - *Optimizing* design decisions over *all* dimensions
  - Whole life-cycle (cradle to cradle)
  - Balancing requirements
  - Including delivery time
  - All disciplines → *interdisciplinary*
  - First *developing the problem*