How Systems Engineers learnt to meet all deadlines

Niels Malotaux

N R Malotaux
Consultancy

+31 655 753 604 niels@malotaux.nl www.malotaux.nl

Niels Malotaux

- Independent Project and Organizational Coach
- Expert in helping optimizing performance

- Helping projects and organizations very quickly to become
  - More effective – doing the right things better
  - More efficient – doing the right things better in less time
  - Predictable – delivering as predicted
- Getting projects on track
- (Embedded systems architect)

Result Management

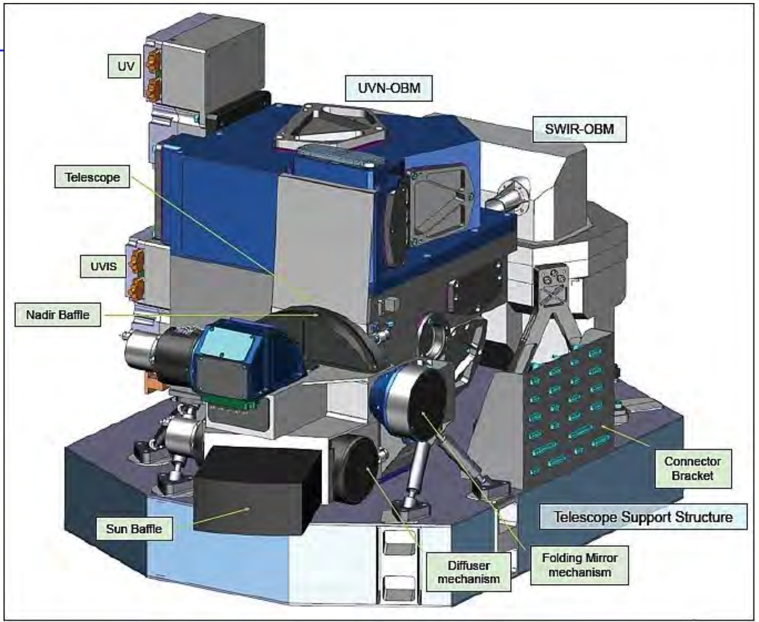
Happy customers

- From one happy customer to another one

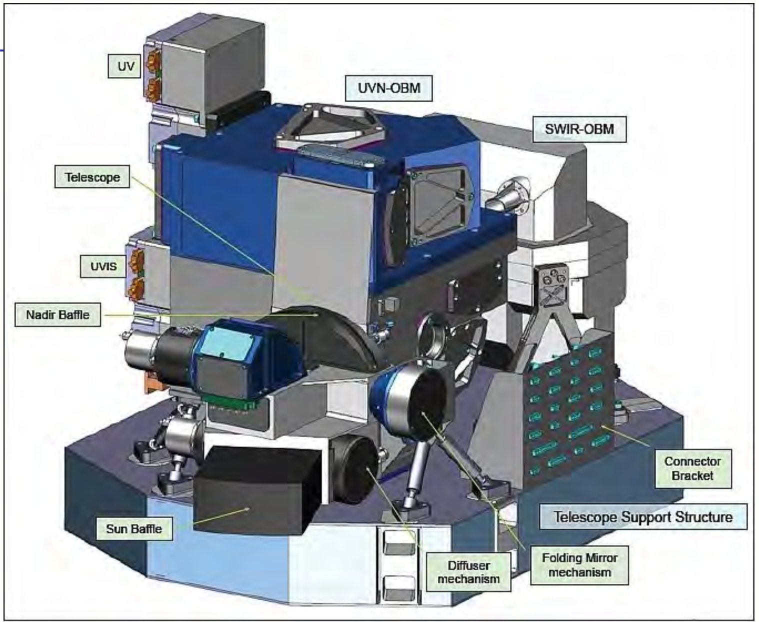
- We will be late and we don’t want to be late
- We cannot afford to be late
- When the money is used up, there is no more

Earth observation satellite

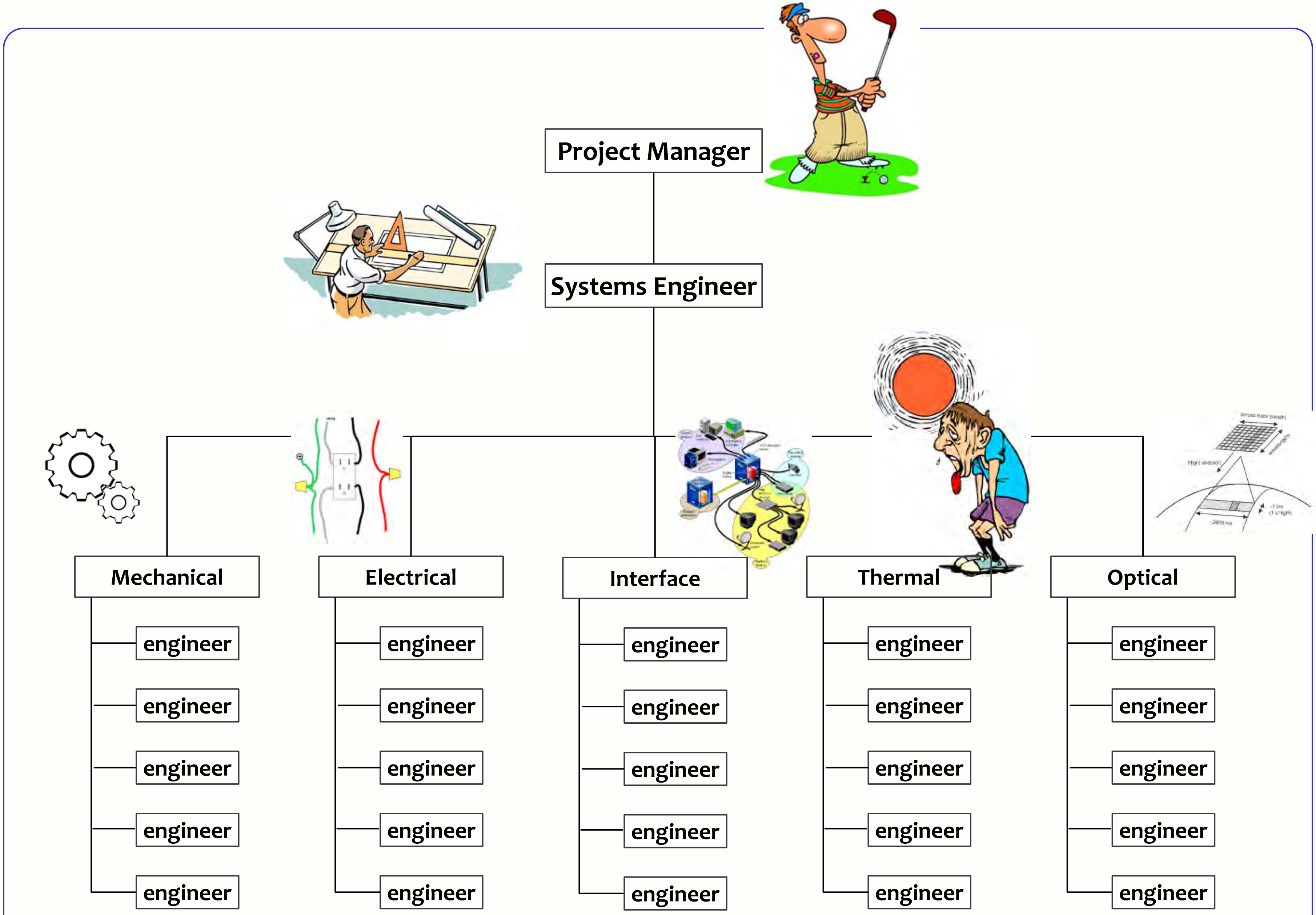
Earth Observation Instrument



In short



- Very experienced Systems Engineers
- Using quantified requirements routinely
- 8 year pure waterfall project (imposed by ESA process)
- Don’t know exactly where they’ll end up
- One problem: They missed all deadlines (can you help us)
- 9 weeks later: They haven’t missed any deadline since
- Recently: delivered 1 day early (instead of expected 1 year late)
- Savings: at least 40 man-year (about €6M)
- How did they do that ?



Project Manager

Systems Engineer

Mechanical

engineer

engineer

engineer

engineer

engineer

Electrical

engineer

engineer

engineer

engineer

engineer

Interface

engineer

engineer

engineer

engineer

engineer

Thermal

engineer

engineer

engineer

engineer

engineer

Optical

engineer

engineer

engineer

engineer

engineer

Issues

- Many interdependent Deadlines
- Many unforeseen issues, resulting in significant changes
- Delay declared unacceptable by customer – launch date fixed
- Team overstressed, no clear focus on tasks at hand
- Everything 80% complete, nothing 100%

Ref Project Systems Engineer

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

Why

What
How much
Are we done

How

Zero
Defects
Attitude

Check as early
as possible

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

Efficiency
of what we do

Effectiveness
of what we do

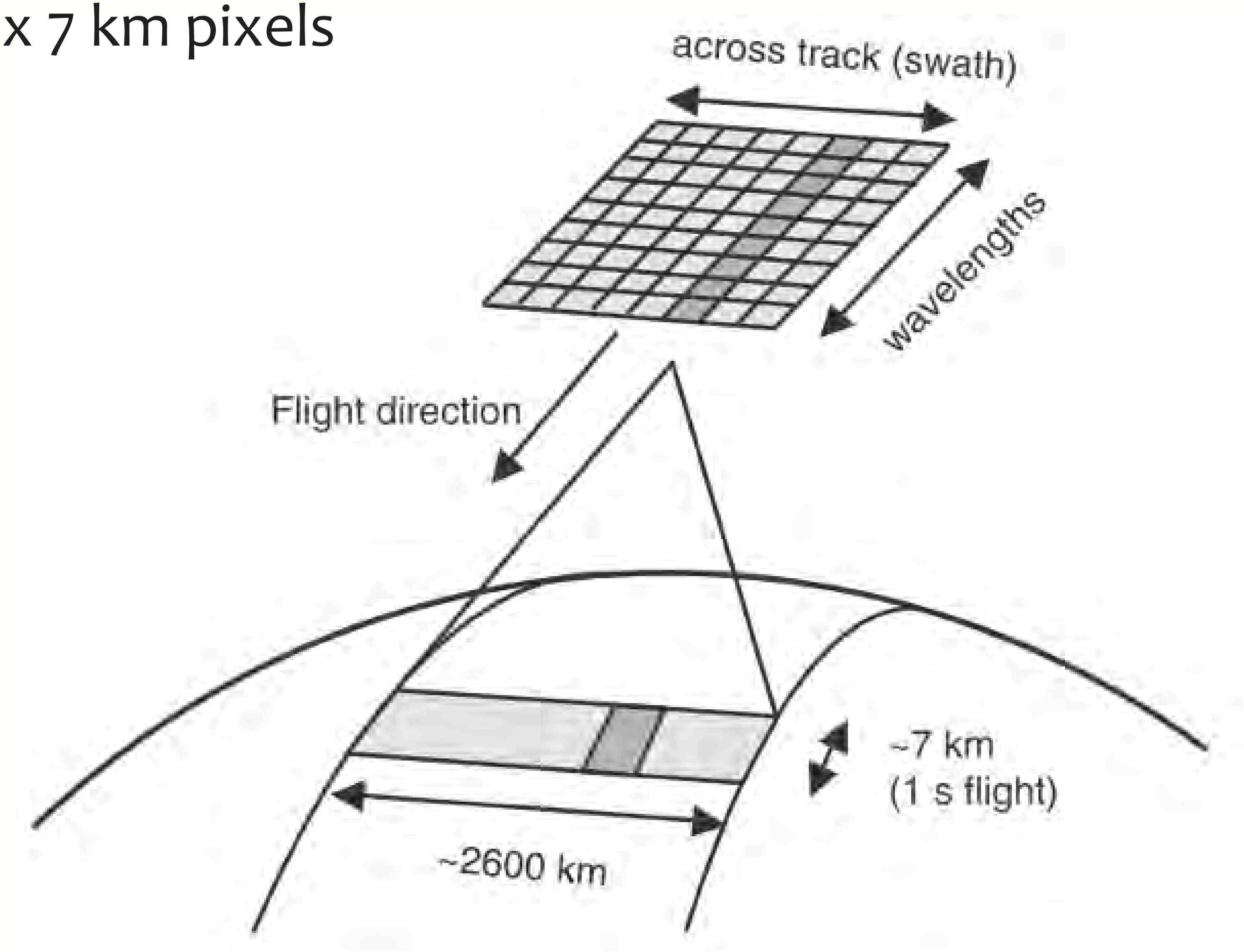
What will happen
and what will we
do about it ?

No ‘point’ requirements

(more by Łukasz Szóstek, this afternoon)

- Requirements for tropospheric O3
  - Ground-pixel size : 20 × 20 km2 (*threshold*); 5 × 5 km2 (*target*)
  - Uncertainty in column : altitude-dependent
  - Coverage : global
  - Frequency of observation :
    daily (*threshold*); multiple observations per day (*target*)
- Requirements for stratospheric O3
  - Ground-pixel size : 40 × 40 km2 (*threshold*); 20 × 20 km2 (*target*)
  - Uncertainty in column : altitude-dependent
  - Coverage : global
  - Frequency of observation :
    daily (*threshold*); multiple observations per day (*target*)
- Requirements for total O3
  - Ground-pixel size : 10 × 10 km2 (*threshold*); 5 × 5 km2 (*target*)
  - Uncertainty in column : 2%
  - Coverage : global
  - Frequency of observation :
    daily (*threshold*); multiple observations per day (*target*)

7 x 7 km pixels



source: www.tropomi.eu

Evolutionary Project Management (Evo)

- Plan-Do-Check-Act
  - The powerful ingredient for success
- Business Case
  - *Why* we are going to improve *what*

Why

- Requirements Engineering
  - *What* we are going to improve *and what not*
  - *How much* we will improve: quantification

What
How much
Are we done

Zero
Defects
Attitude

- Architecture and Design
  - Selecting the optimum compromise for the conflicting requirements

How

- Early Review & Inspection
  - Measuring quality while doing, learning to prevent doing the wrong things

Check as early
as possible

Evo Project Planning

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

Efficiency
of what we do

- Bi-weekly DeliveryCycle
  - Optimizing the requirements and checking the assumptions
  - Soliciting feedback by delivering Real Results to *eagerly waiting* Stakeholders

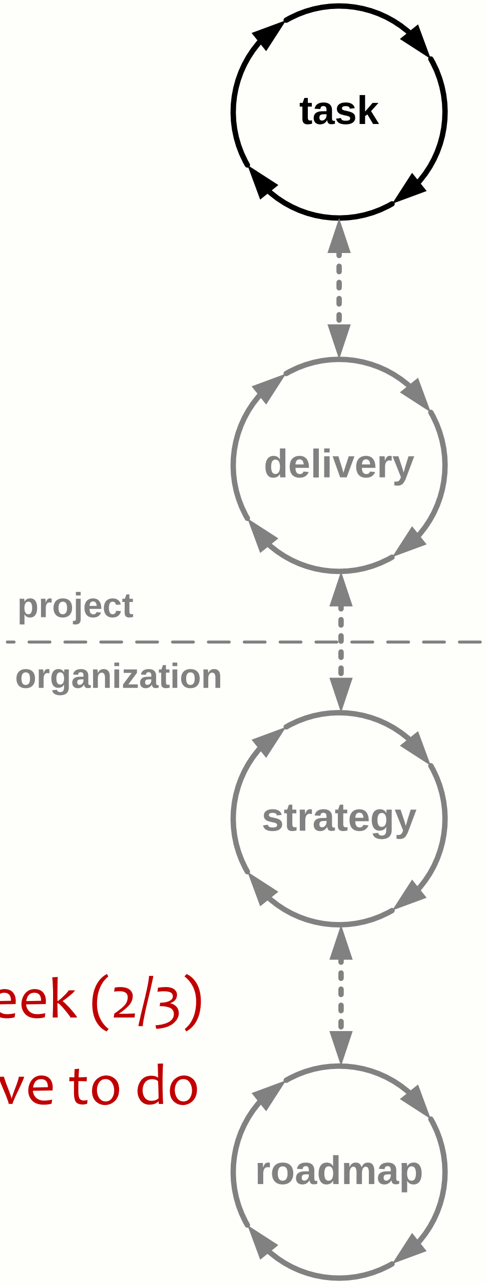
Effectiveness
of what we do

- TimeLine
  - Getting and keeping control of Time: Predicting the future
  - Feeding program/portfolio/resource management

What will happen
and what will we
do about it ?

Weekly TaskCycle

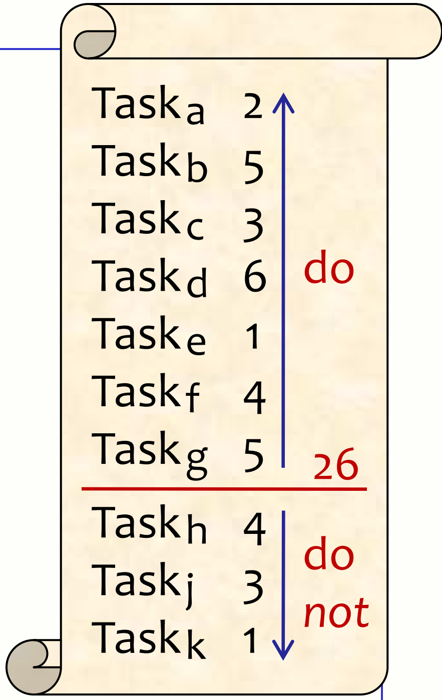
- Are we *doing* the right things,
  in the right order,
  to the right level of detail for now
- Optimizing estimation, planning and tracking abilities to better predict the future
- Select highest priority tasks, never do any lower priority tasks, never do undefined tasks
- There are only about 26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



Weekly planning

- Individual preparation
  - Conclude current tasks
  - What to do next
  - Estimations
  - How much time available
- Modulation with / coaching by Project Management (1-on-1)
  - Status (all tasks done, completely done, not to think about it any more ?)
  - Priority check (are these really the most important things ?)
  - Feasibility (will it be done by the end of the week ?)
  - Commitment and decision
- Synchronization with group (team meeting)
  - Formal confirmation (this is what we plan to do)
  - Concurrency (do we have to synchronize ?)
  - Learning
  - Helping
  - Socializing

Weekly 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	do *not*
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 per week)
- 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

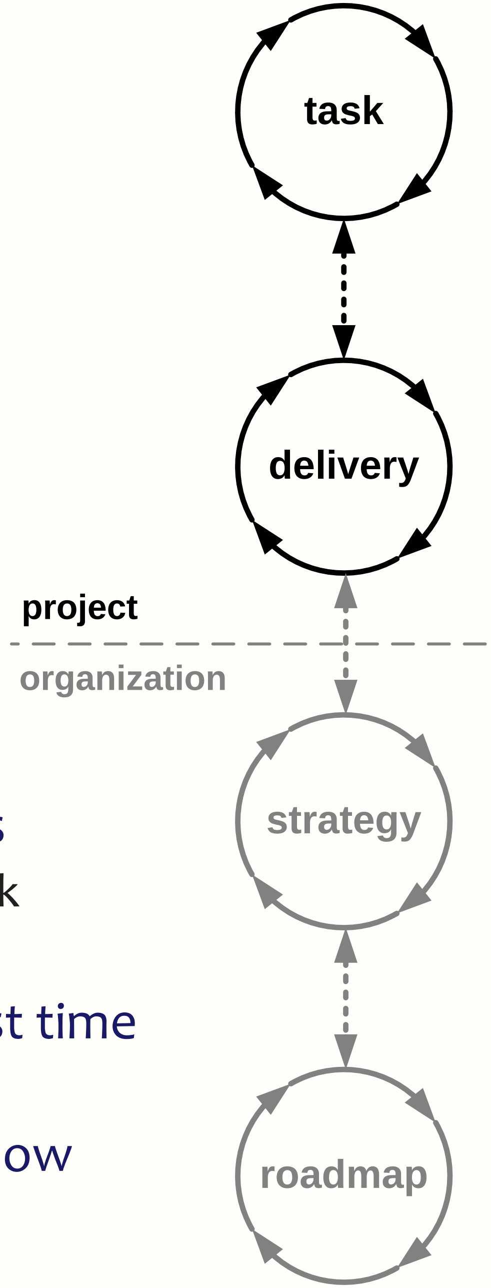
Why is this important?

- Half (±30%) of what people do in projects later proves not having been necessary → use ~~Retrospectives~~ Prespectives
- During the TaskCycle planning we can very efficiently see
  - What our colleagues think they're going to do
  - Make sure they're going to work on the most important things
  - Not on unnecessary things
  - In line with the architecture and design
  - Leading most efficiently to the goal of the delivery
- Everyone in the project-team knows what the others will do
- We see any issues before they become a problem

DeliveryCycle

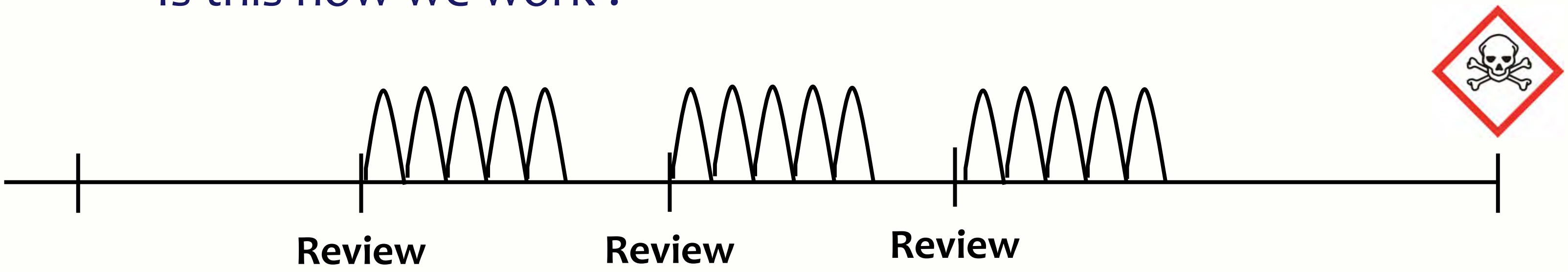
getting

- Are we *delivering* the right things,
  in the right order
  to the right level of detail for now
- Optimizing requirements *and solutions*
  and checking assumptions
  1. What will generate the optimum feedback
     How can we organize optimum feedback
  2. We deliver only to eagerly waiting stakeholders
     We give feedback immediately or max within a week
  3. Delivering the juiciest, most important
     stakeholder values that can be made in the least time
     We check that we get results in the right order
  - What will make Stakeholders more productive now
- Not more than 2 weeks

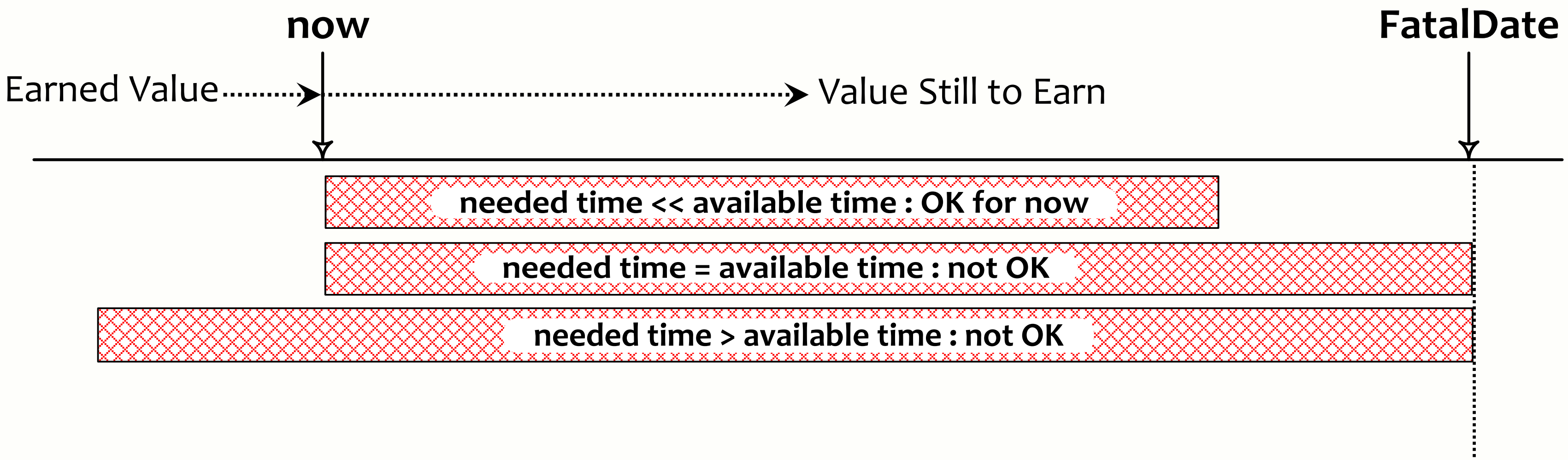


Formal Reviews with suppliers

- Formal Reviews at the end of a project phase
  - ~~What is OK~~
  - ~~What is not yet OK~~
  - Everything is OK
  - We’re not perfect: OK within 5 days
    (we: they and us together)
  - Is this how we work ?



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

- Value Still to Earn

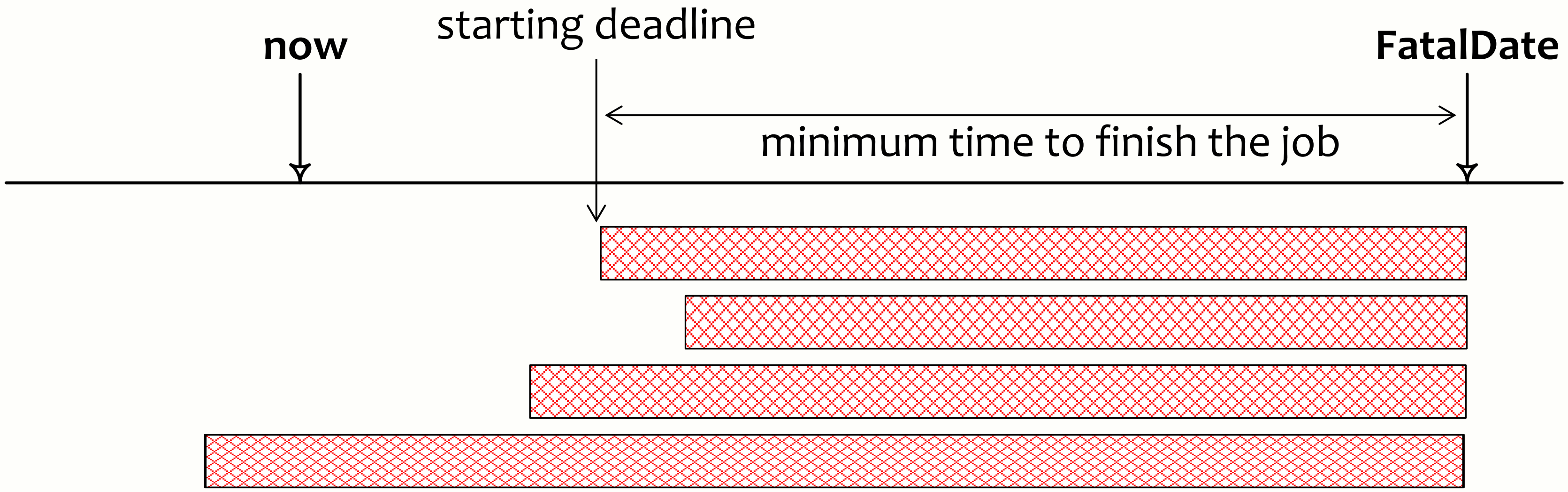
versus

- Time Still Available

If the match is over, we cannot score a goal

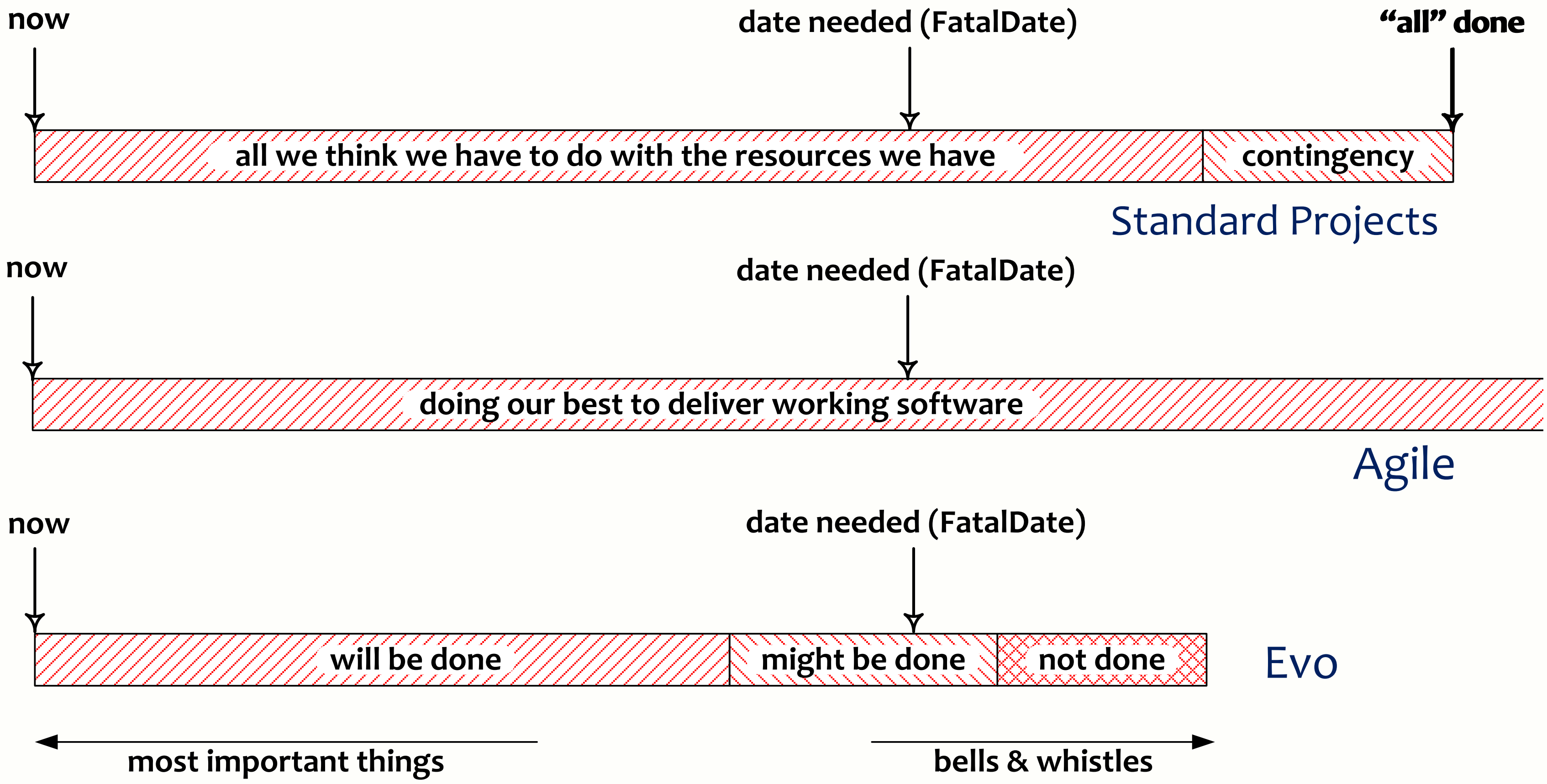
Even more important: *Starting Deadlines*

- Starting deadline
  - Last day to start to make the finish deadline
  - Every day we start later, we will end later



TimeLine

How do we know that we do, and get *what* is needed, *when* it's needed ?



now

date needed (FatalDate)

"all" done

all we think we have to do with the resources we have

contingency

Standard Projects

now

date needed (FatalDate)

doing our best to deliver working software

Agile

now

date needed (FatalDate)

will be done

might be done

not done

Evo

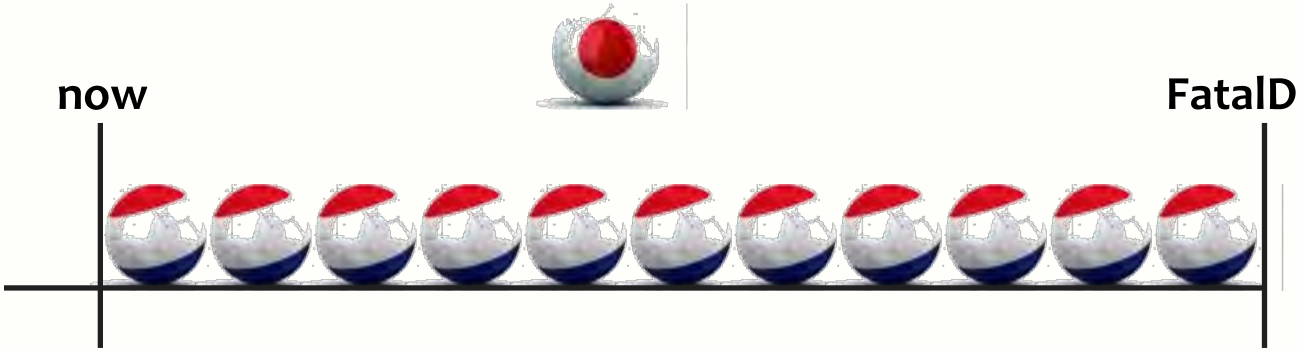
most important things

bells & whistles

- Better 80% 100% done, than 100% 80% done
- Let it be the most important 80%

If we add something …

If we add something, something else will not be done



Rather than letting it happen randomly

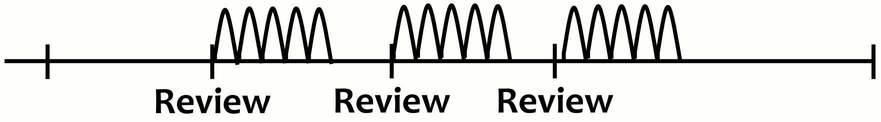
We better decide what will happen

First needed to convince the Project Manager

- We’ve been doing this kind of projects for 27 years
- We’re very good at it
- What do you think you can contribute to that ?

- Do you have to deliver anything by the end of the week ?
- A status report
- How much time do you need ?
- How much time do you have ?
- Does it fit ?
- What are we going to do about it ?

Awful schedule pressure !



- Meeting with sub-contractors in three weeks
- Many documents to review
- Impossible deadline

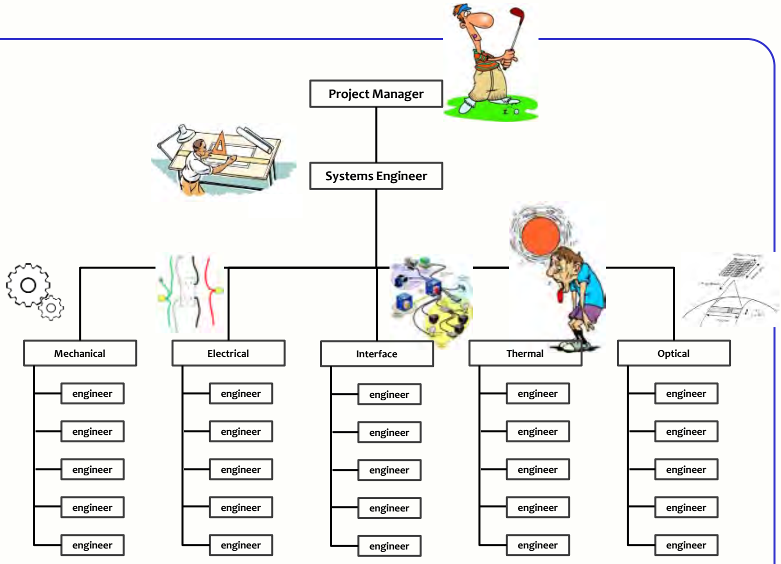
- How many documents to review ?
- How much time per document ?

	per doc	hr
4 heavy	15	60
3 easy	2	6
	total	66
other work		33
	total	99

available	2 x 26	52

- Some suggestions …
- Result: well reviewed, great meeting, everyone satisfied

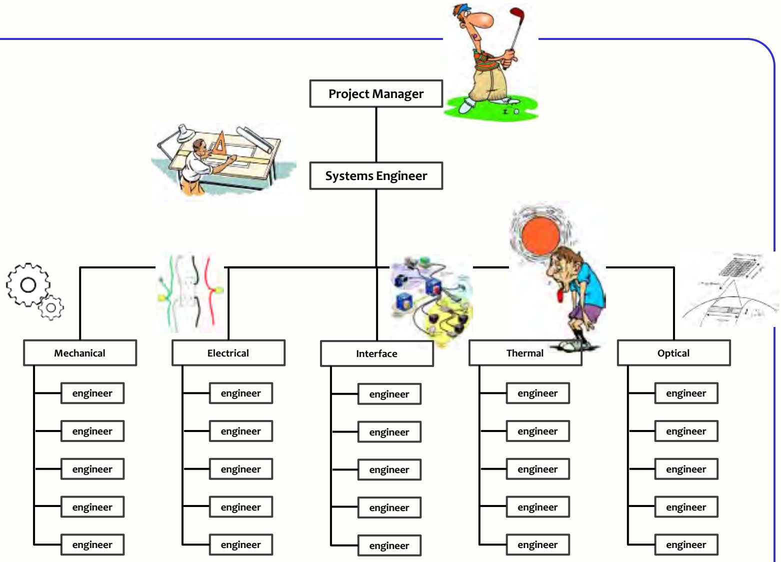
Evo Process



- Detailed activities planned with one week horizon and granularity ≤ 5 hr
  - Based on a personal timeline of 10 weeks
  - Based on the list of major project milestones
- Dependencies (resources, input) identified and secured at the start of each week
- Weekly 15 min individual check with Systems Engineer
- Plenary 15 min check during Engineering Meeting

Ref Project Systems Engineer

Boundary Conditions



Project Manager

Systems Engineer

Mechanical
engineer
engineer
engineer
engineer
engineer

Electrical
engineer
engineer
engineer
engineer
engineer

Interface
engineer
engineer
engineer
engineer
engineer

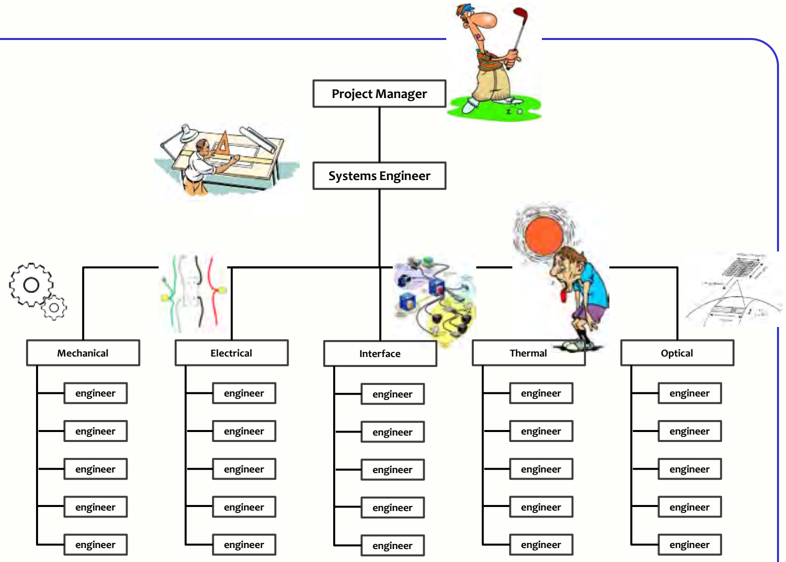
Thermal
engineer
engineer
engineer
engineer
engineer

Optical
engineer
engineer
engineer
engineer
engineer

- Planned work is 2/3 of working time
- Only do the most important - *you* decide what that is
- Better to finish one activity on time than 5 “almost”
- Interrupt Procedure for unforeseen activities
- Only in case of unresolvable conflict, escalate

Ref Project Systems Engineer

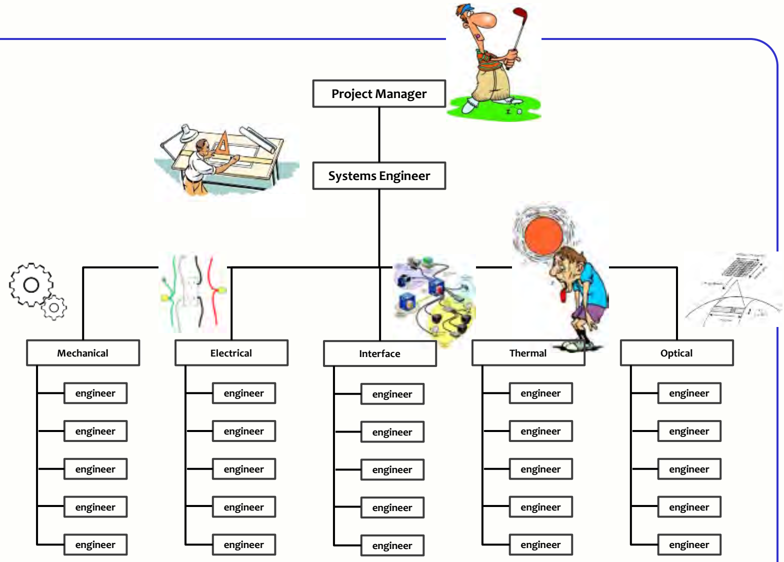
Checklist Evo: Previous week



- All planned Tasks OK ?
- If yes, did you really work on the most urgent things ?
- If no
  - Any unavoidable causes (e.g. illness)
  - Correct use of Interrupt Process
    - Balanced consequences of the interrupt ?
    - Clearly marked the Tasks you had to sacrifice ?
  - Is your plannable time right ?
  - Did you underestimate any Tasks? What did you do about it ?
  - Did you need results from others, who didn’t deliver on time ?
  - Did you use Active Synchronization ?
  - Was there any reason to escalate and did you escalate ?

Ref Project Systems Engineer

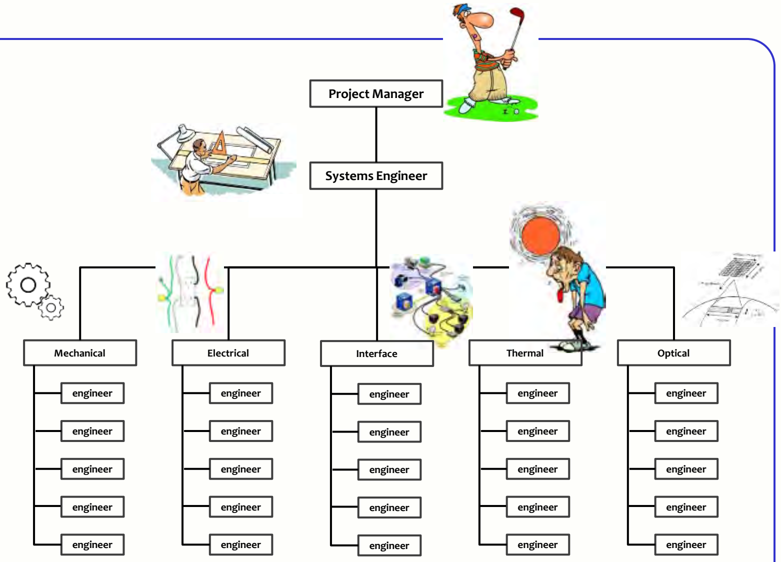
Checklist Evo: Coming week



- Used your personal,
  or team TimeLine ?
- Haw many plannable hours ?
- Any issues that should be escalated ?
- All Tasks urgent ?
- How did you deal with unfinished Tasks of previous week ?
- Dependencies managed ?
- Are Tasks concrete enough ?
- Should some Tasks better be split into smaller Tasks ?
- Can you estimate better if split into more detail ?
- Will you succeed ?

Ref Project Systems Engineer

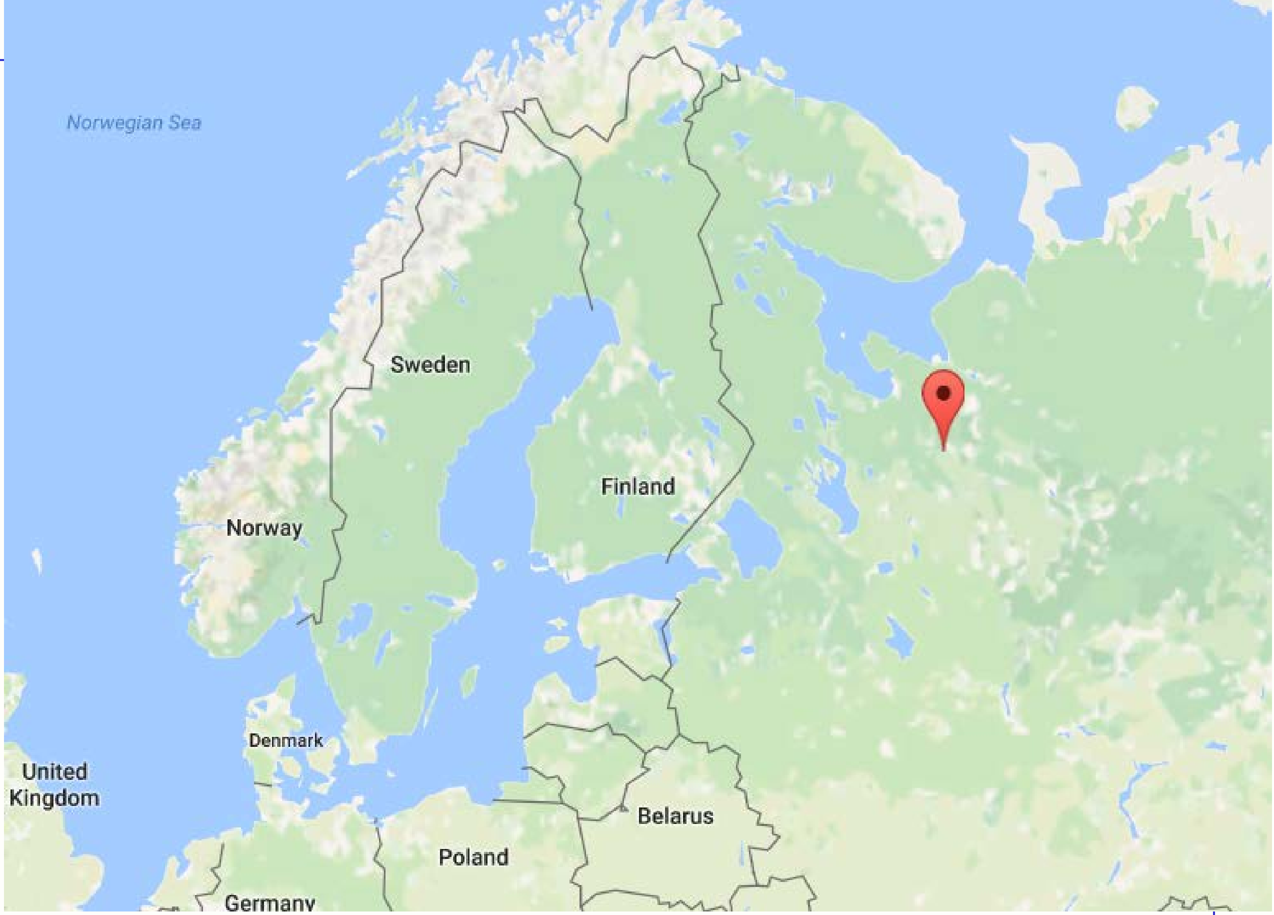
Did it work for this project ?



- 2 months needed
  to get the process in full swing
- All Engineering docs in PDR and CDR data packages on time
- Stress level in team greatly reduced
- More supervisory work for Systems Engineer - can effectively handle up to 8 people
- People not in the Evo swing tend to lag behind
- So, we need everyone to follow
- Good enough to become company standard ? I say YES

Ref Project Systems Engineer

Launch ?



- Current status
  The satellite is in storage, awaiting shipment to the launch site at Plesetsk
- The launch is planned for 2017

Why is the satellite not launched yet ?

- The launch is delayed caused by issues you cannot predict even with Evo:
  - The launch SW from the Ukraine, bought by ESA 5 years ago, is to be used in Russia. Incomprehensibly that's a bit more difficult than it was 5 years ago
  - By now the problems seem to have been solved and the launch is planned for March/April (current deadline: August)
- Coincidentally I just today introduced our Evo way of working to a new team member of our current project (mapping the large-scale structure of the Universe over a cosmic time covering the last 10 billion years)
- I'm curious to find out how quickly she'll really get the idea

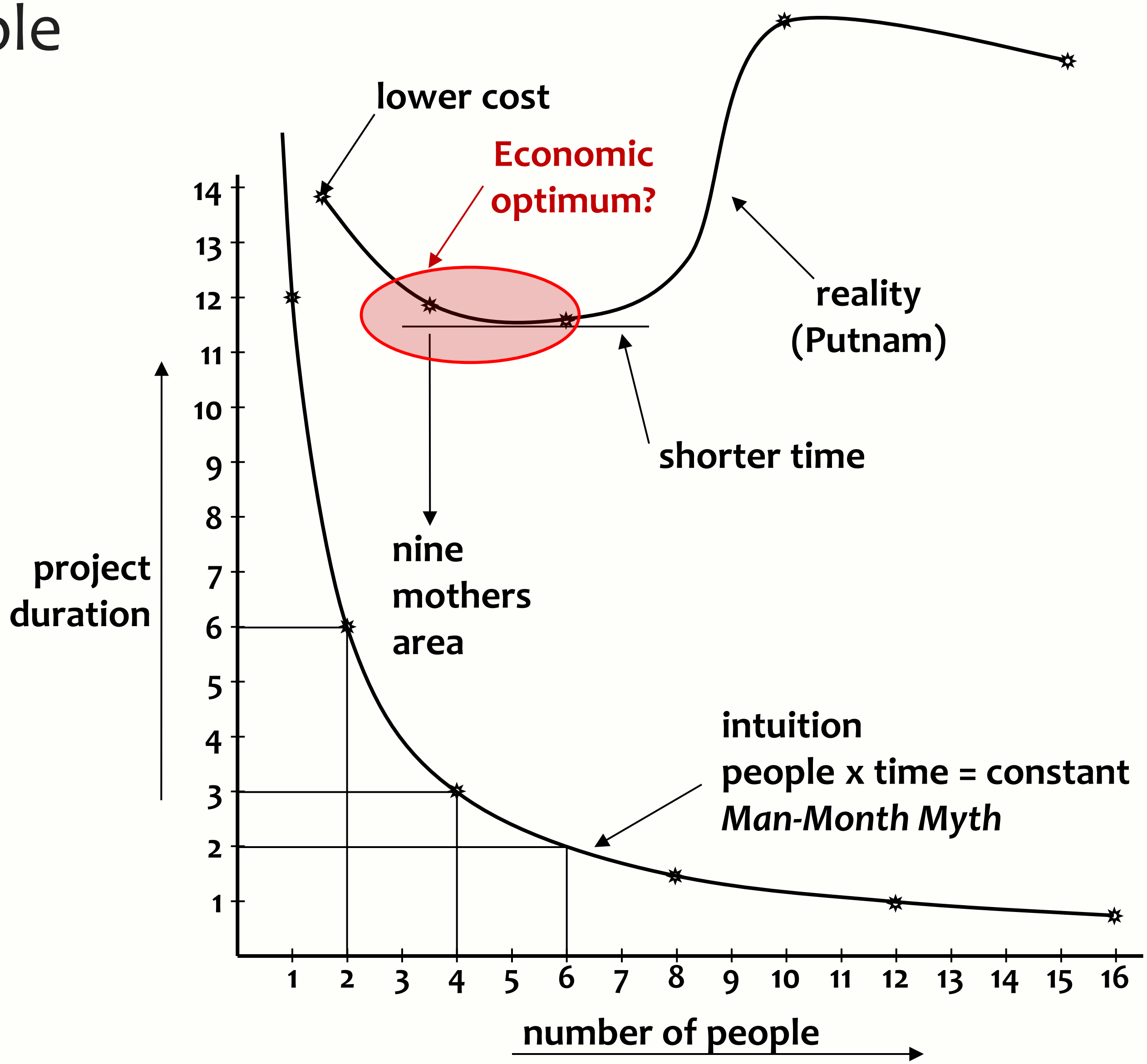
Ref Project Systems Engineer

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

Intuition often guides us in the wrong direction

Adding people



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

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/Inspections

Inspection pages

How Systems Engineers learnt to meet all deadlines

Niels Malotaux

N R Malotaux
Consultancy

+31 655 753 604

niels@malotaux.nl

www.malotaux.nl