TimeLine: Getting and Keeping Control over your Project

Niels Malotaux, N R Malotaux - Consultancy, the Netherlands, niels@malotaux.nl

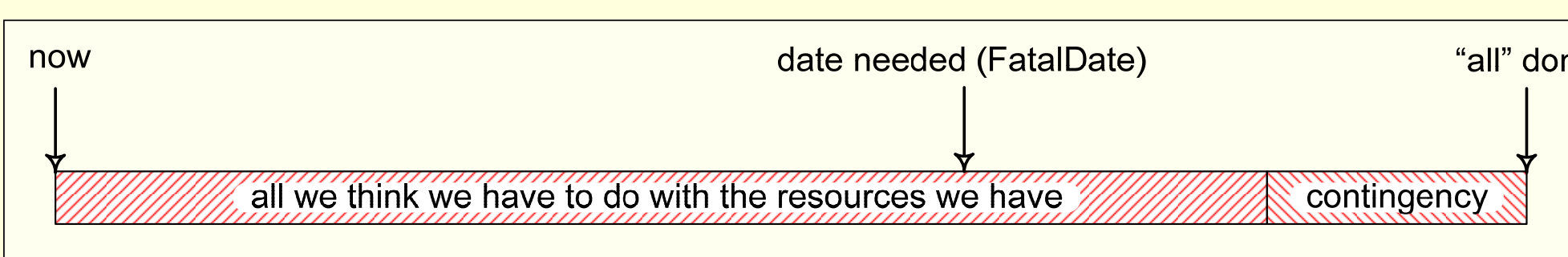
The Goal of any project (Top-Level-Requirement)

Providing the customer with *what* he needs, at the *time* he needs it,
to be *satisfied*, and to be *more successful* than he was without it …
… *constrained* by what the customer can *afford*,
and what we *mutually* beneficially and satisfactorily can deliver in a *reasonable* period of time.

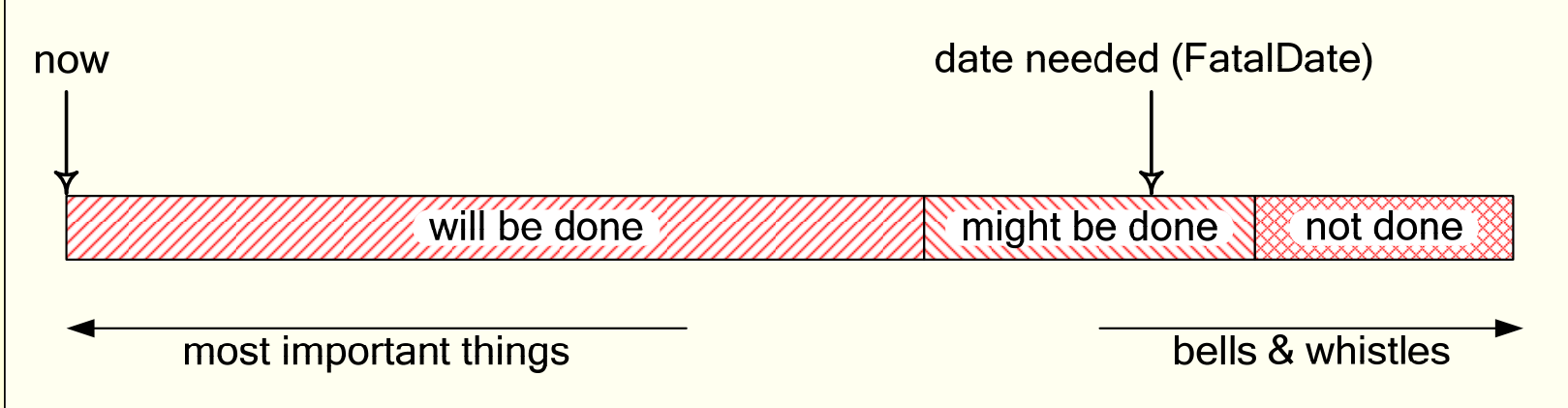
TimeLine is a technique for estimating what can be done in the available time, what *cannot* be done, and what to do when what we think has to be done doesn't fit in the available time. Instead of fatalistically accepting that we need more time, it is usually quite well possible to do *more* in the *available* time. The **TimeLine** technique is part of the *Evolutionary Project Management* (***Evo***) approach.

The basic TimeLine steps are:

1. **Define a deadline or FatalDate**
   Planning beyond the available time/money budget is useless, so we don't waste time once we know that what we need costs more than we have. Counting back from the FatalDate: When should we have started?
2. **Write down whatever you *currently think*** (it probably will change*!*) **you have to accomplish**
3. **List in order of priority**
   Priority is based on value contribution and hence is influenced by many things and even dynamically changing*!*
4. **Write the same down in elements of work** - Don't detail more than necessary
5. **Ask the team to add forgotten elements and add duration estimates**
   Use days or weeks of calendar time, depending on the size of the project
6. **Get consensus on large variations of estimates, using a Delphi process** (see Simple Delphi)
7. **Add up the duration of all elements** (estimation errors will average out)
8. **Divide by the number of available people** (still, see Brooks' Law about the danger of doing this*!*)
9. **This is a first estimate of the duration**



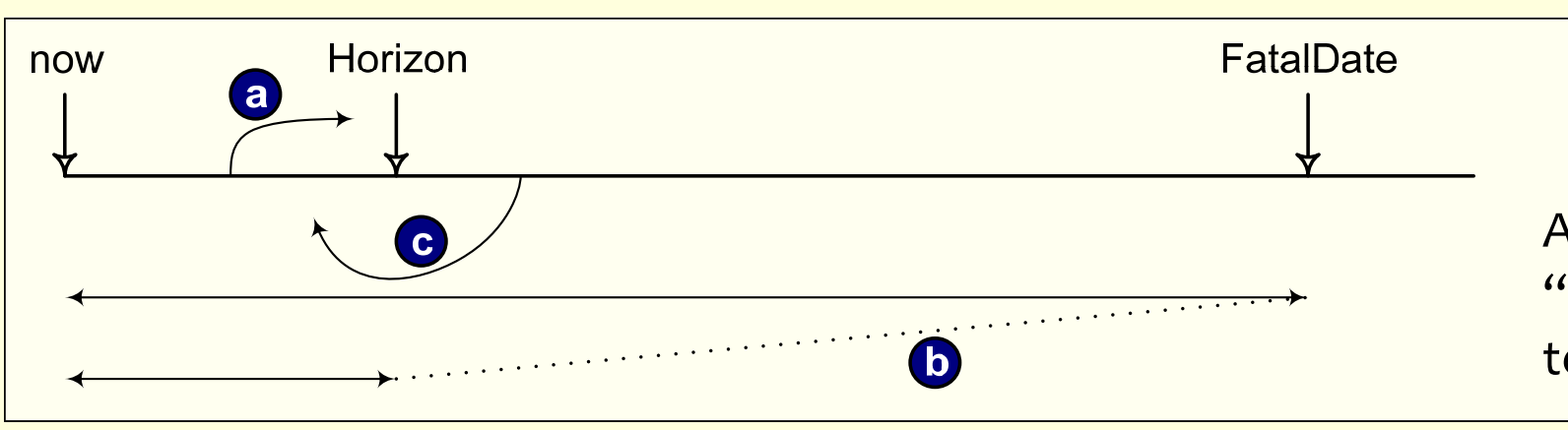
Standard Project Management: Add up all we think we have to do, add some contingency and now we know when "all" will be done. In real practice, the FatalDate is usually earlier and it is not well definable what "all" is.



Evolutionary Project Management: At the FatalDate, some things will surely be done, some will not be done, while some might be done, because estimation is not exact. Better the *most important* 80% 100% done, than 100% 80% done. If what surely can be done in the available time isn't enough to make a profitable ROI, we can kill the project *now* in stead of way beyond the FatalDate. Time is a requirement, just as all other requirements*!*

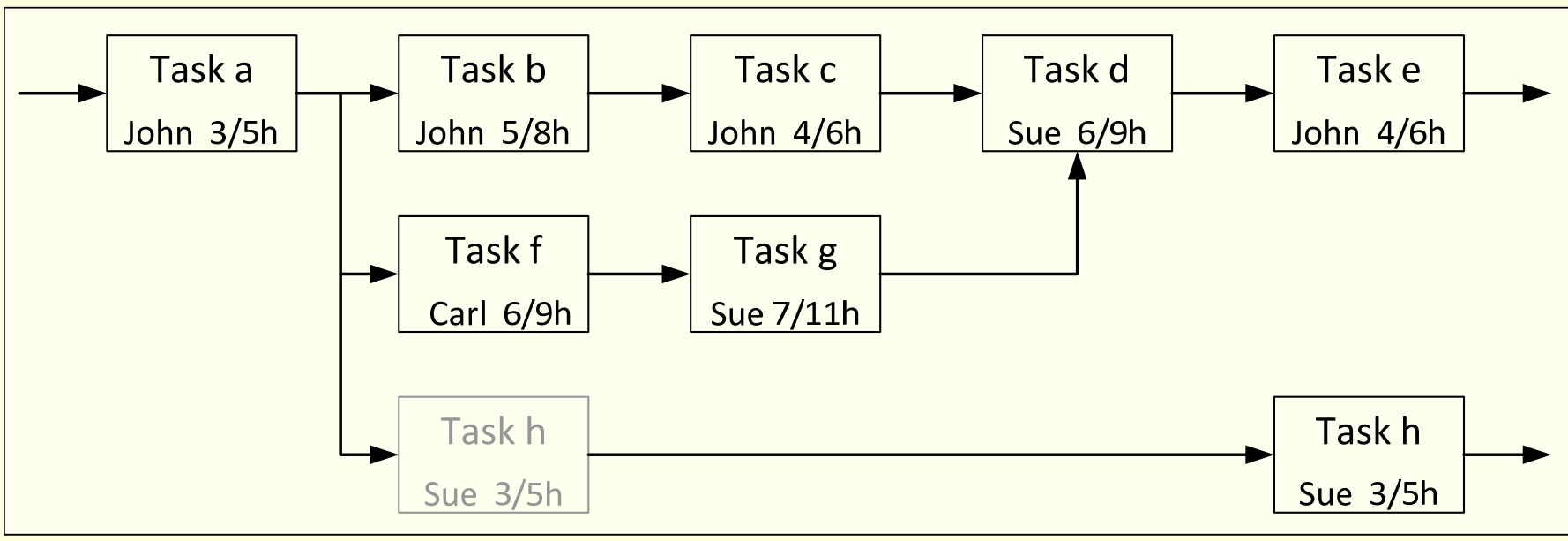
We set a Horizon because people cannot oversee, and hence cannot control longer periods of time well:

10. **Choose a Horizon** (default: 10 weeks from now)
11. **Determine when to look over the Horizon again (a)** (default: halfway)
12. **Determine the amount of work proportionally to the total work (b)**
13. **Pull tasks from beyond the Horizon that need earlier preparation not to get surprised later (c)**



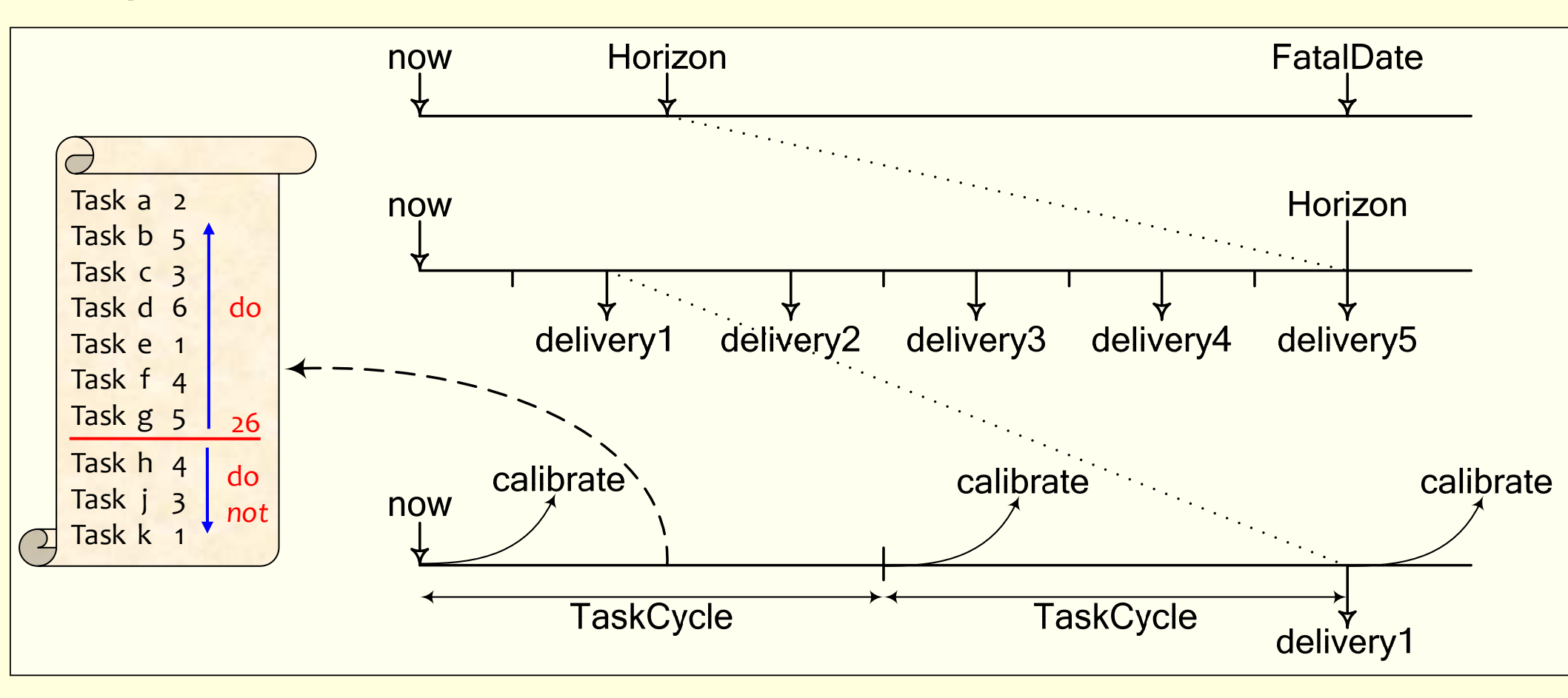
After we've done three things, we can "forget" about what's beyond the Horizon temporarily. See steps 11, 12 and 13 above.

14. **Put the work for the 10 weeks in optimum order, defining Deliveries of 2 weeks**
    Deliveries [8]: *what* are we going to deliver to *whom*, and *why*: deliveries are meant for feedback, checking requirements and assumptions. So we make sure we deliver to appropriate and *eagerly waiting* Stakeholders
15. **Estimate the work in more detail now**
16. **Make a good description of the first one or two Deliveries**
17. **Check the feasibility of completing these Deliveries in two weeks each, with the available resources**
18. **Determine Tasks for the first week**
19. **Estimate the Tasks, now in real effort (net) hours needed to 100% complete each Task**
20. **The people in the team select Tasks to fit the plannable time** (default: 2/3 of available time [8])
21. **Select only the most important Tasks**, never plan nor do less important Tasks, never do undefined Tasks
22. **Now we have the Tasks for the first week defined**
23. **Make sure this is the most important set of Tasks**
24. **Put the Tasks in optimum order, to see how to synchronize individual people's work during the week**



Synchronizing Tasks
(4/6h means 4 hr effort, 6 hr lead-time)
Carl has to start working on Task f 6 + 9 + 11 + 9 = 35 hr before the end of the Cycle , not to fail his team's result. John has to start Task a immediately not to fail. Sue must be careful when to do Task h.

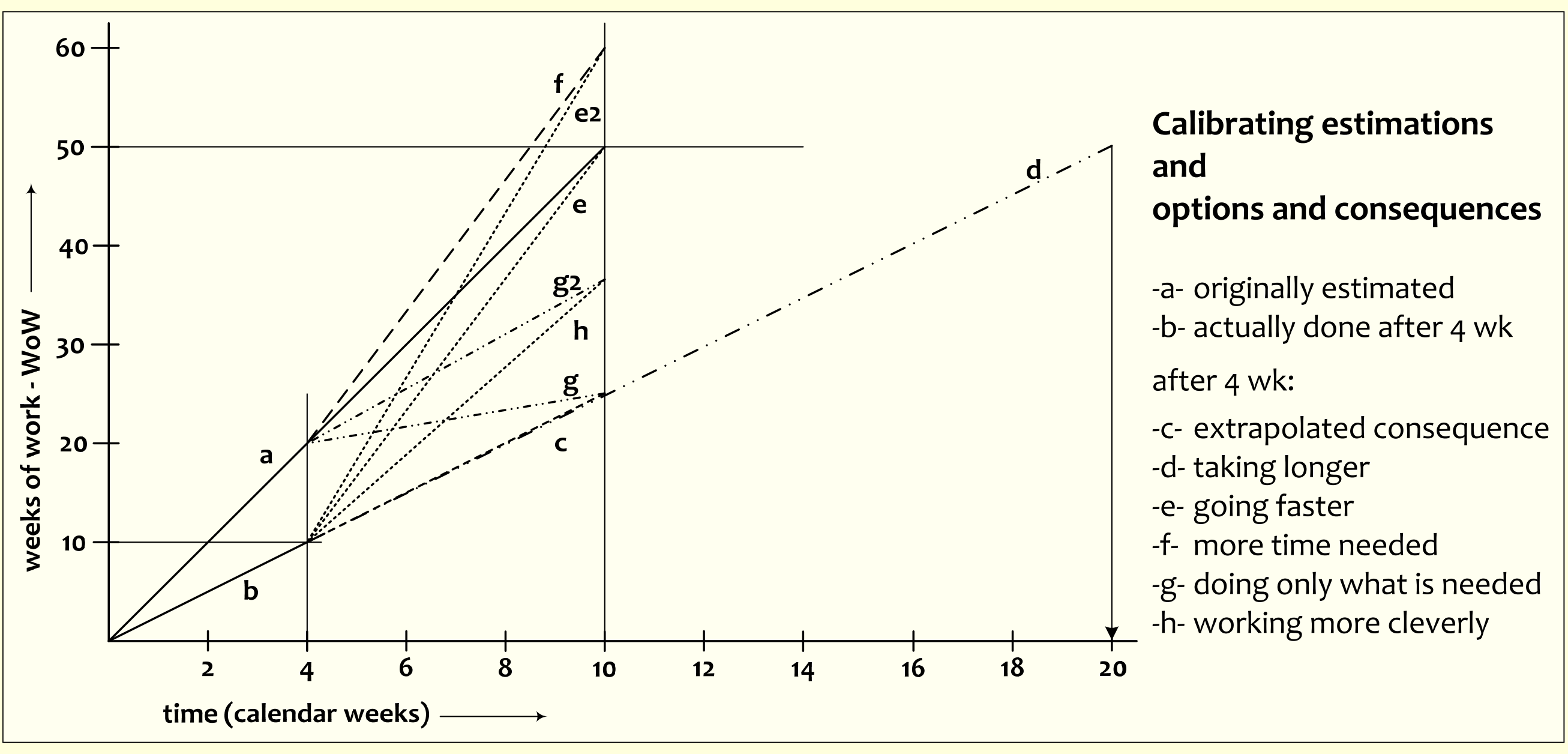
25. **Calibrate back the TimeLine estimations *and take the consequence of what you see***
26. **Repeat every one or two weeks**



TimeLine summary
Setting a FatalDate, a Horizon, Deliveries [8], TaskCycles [8], and then calibrating back

Example of taking the consequence of the TimeLine:

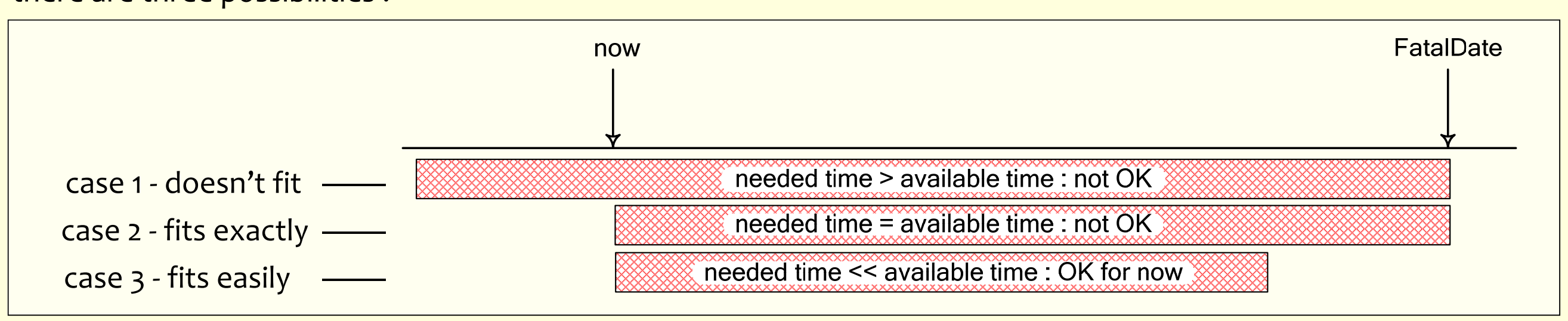
- At the start, we estimate that the work we think we have to do in the coming 10 weeks is about 50 person Weeks of Work (WoW, line a). With a team of 5 people this seems doable.
- After 4 weeks, we find that "only" 10 WoW have been completed (line b), instead of the expected 20 WoW. If we don't change our ways of working, the project will surely be late*!* Note: this is an *optimistic* example, often it's worse *!*



- We now have to assume that at the end of the 10 weeks we'll have completed (10/4) x 10 = 25 WoW (line c). This is 50% less than the original 50 WoW expected. Some managers start panicking at this stage.
- Alternatively, the original 50 WoW will be done in 20 weeks, or 100% more time than originally expected (line d).
- In case the deadline is really hard, the typical reaction of management is to throw more people at the project. How many people? Let's calculate: The velocity (actual accomplished against planned effort; see also Cohn [1]) is 10/20 = 0.5 WoW per week. With a velocity of 0.5, we will need for the remaining 40 WoW 40/0.5 = 80 person weeks in the remaining 6 weeks. Therefore, we think we need 13.3 people instead of the original 5. Management decides to add 9 people (expecting line e).
- There is, however, another issue: based on our progressing understanding of the work we found that we forgot to plan some work that "has" to be done to complete the result we planned for the 10 weeks period: now we think we still have to do 50 WoW in the remaining 6 weeks (line f). This would mean 50/0.5 = 100 person weeks in the remaining 6 weeks, which makes management believe that they actually need 16.7 people. They decide to add 12 people to the project, because they don't want the project to take 100% longer and they think they are prepared to absorb the extra development cost, in order to win Time-to-Market. Beware, however, that this solution won't produce the desired outcome, and even may work out counterproductive, as explained by Brooks' Law [2].
- Much overlooked, but most rewarding and usually quite possible is *actively saving time*, doing only what is necessary (line g), or in practice a combination of not doing what is not necessary (line g2) and doing things more productively (line h). Actively and weekly designing what exactly to do and in which order saves a lot of time.

We cannot change history. We only can influence our future. So we use the concept of ***Earned Value*** to learn from the past, but, more importantly, ***Value Still to Earn***, to fit what we think we still have to do in the remaining time.

What do we do if what we think has to be done doesn't fit the time available?
After we have estimated how much time we need to do everything we think we have to do, there are three possibilities :


now
FatalDate
case 1 - doesn't fit
case 2 - fits exactly
case 3 - fits easily
needed time > available time : not OK
needed time = available time : not OK
needed time << available time : OK for now

In case 1, we don't just continue, but first we have to resolve this problem, because we now already know that the project will fail*!* We don't want the project to fail.
In case 2 we'd better assume it's like case 1, because surely we'll find out that we'll have to do a lot of things more.
In case 3 we could, for the time being, assume that we may succeed: continue with setting a Horizon as described at basic TimeLine steps, see left.

What options do we have if it doesn't fit?
We see several options being used in practice, most of them deceptive, making things worse:

Deceptive options:

- **Hoping for the best** (fatalistic)
- **Going for it** (macho)
- **Working overtime** (getting tired, making more mistakes, thinking you work hard, but not really working smart)
- **Moving the deadline**
  - Parkinson's Law [4]: *Work expands to fill the time for its completion*
    If you extend the deadline, the available time will still be filled with the same amount of work.
  - Student Syndrome [5]: *Starting as late as possible, only when the pressure of the FatalDate is really felt*
    If you extend the deadline, people start working (too) hard later, usually too late.

Then there is the intuitive move we often see if time is running out: "Add people*!*"

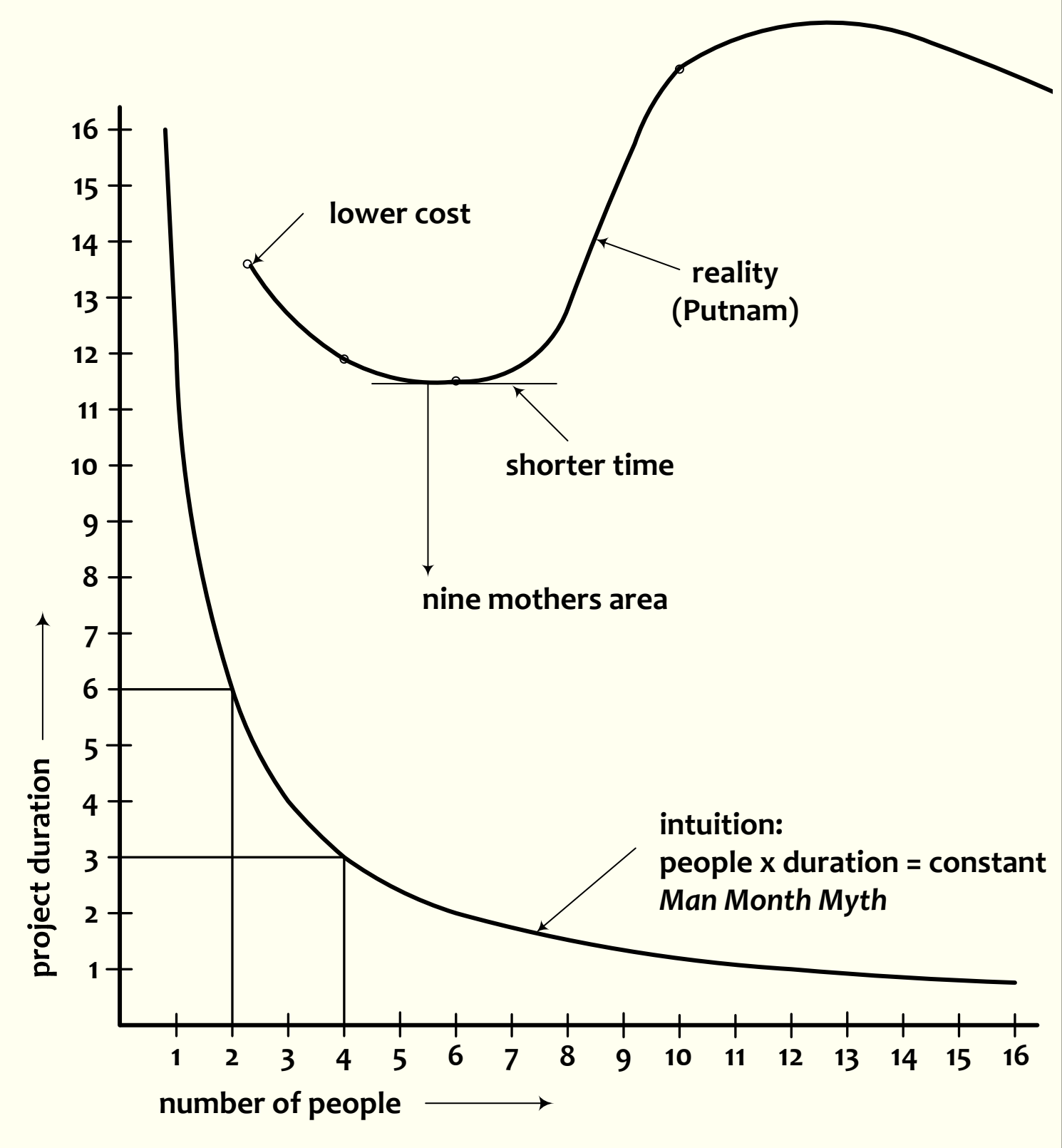
Treacherous option: Adding people - Brooks' Law, 1975: *Adding people to a late project makes it later.*
As Brooks noticed in his essay *The Mythical Man-Month* [2], this doesn't only apply at the end of a project, but it is already true when during the project we are running slower than we expected (see calibration example).
Apparent causes may be:

- The time needed to complete a project is depending on more parameters than just the number of people
- It takes time for the added people to get acquainted with the project
- It takes time of the people in the project to help the new people getting acquainted
- The new people are likely to introduce relatively more bugs during their introduction period, causing the need for extra find-and-fix time
- Having more people on the team seems to increase the capacity linearly, but the lines of communication between these people increase much quicker, every n^{th} person adding (n-1) extra lines of communication
- The project manager as well as the architect become bottlenecks when there are too many people to manage
- The productivity of different people can vary vastly, so people cannot simply be exchanged for time

Intuition makes us think that men and months can be interchanged: "If 2 people can do it in 6 months, then 4 people can do it in 3 months".
Brooks stated in his essay *The Mythical Man-Month*: "… the man-month as a unit for measuring the size of a job is a dangerous and deceptive myth".
Putnam [3] calculated this effect after researching some 500 projects:

- The cheapest software project is done with 2 or 3 people.
- The fastest project is done by 5 to 7 people. The gain in time is, however, only about 20%, so the gain comes at a premium. Still, because Time-to-Market often is an important requirement, this may be an economic optimum.
- The alarming message comes if we run a project with 9 or more people: it becomes exponentially more expensive, because with more people, it takes even more time.

Recently, some 10 years later, Putnam repeated his measurement and arrived at similar conclusions. Improved tools hadn't improved the situation. Apparently, the same human factors are still at work.



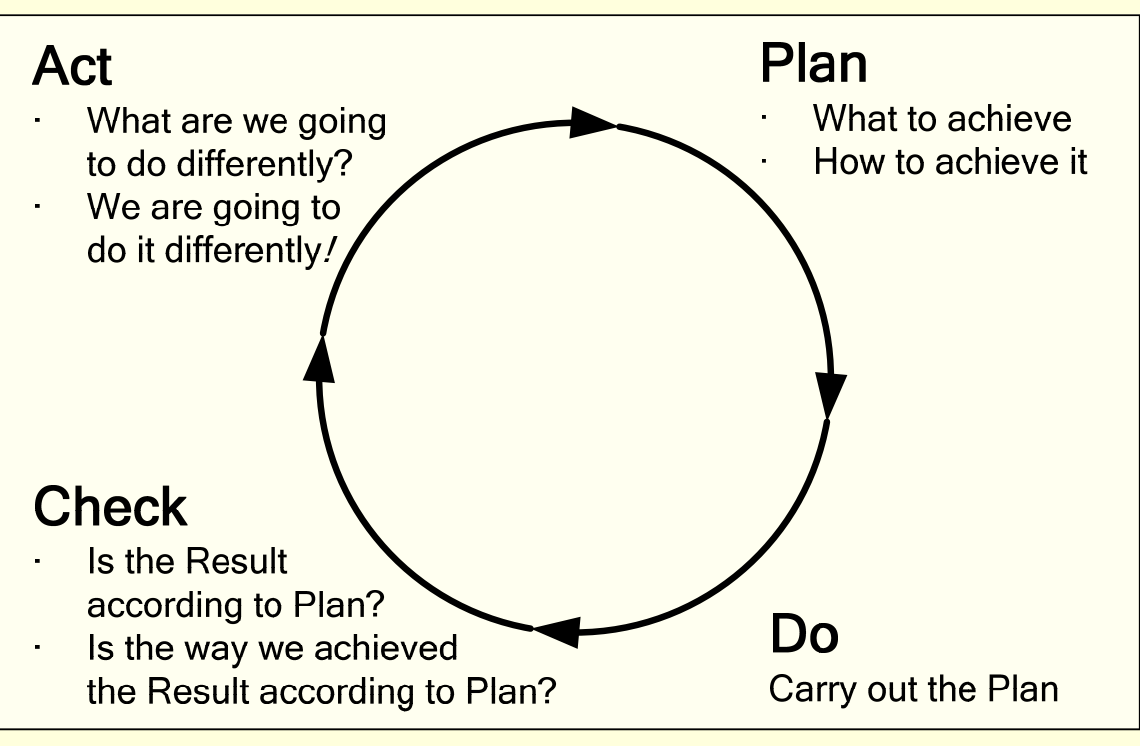
Useful option: Saving time
Mostly overlooked, this is the option that really works in practice: If we don't have enough time, we have to *and we can* save time. In every project we can save a lot of time, *without negatively affecting the result* of the project, even actually *improving the result* of the project by improving:

- **Efficiency in *what* (*why* and for *whom*) to do -** doing only what is needed, not doing things that later *prove to be superfluous*. We use the *Goal of any Project* (see begin) as a focus
  - People tend to do more than necessary especially if they don't exactly know what to do
  - Better 80% 100% done, than 100% 80% done, let it be the most important 80%, the other part isn't used anyway
  - Using a Zero-Defect attitude, optimizing the prevention process, we use less time on avoidable issues
  - Is everything we think we have to do really needed?
  - Magic question: "*Who's waiting for this?*" If nobody is waiting, we shouldn't do it (or find out who should be waiting)
- **Efficiency in *how* to do it -** doing things *differently*
  - 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
  - Don't just use the obvious solution: compare the obvious solution with one non-obvious solution, then decide
  - First think, then do: Plan before Do, Design before Implement
  - Using Check and Act to improve, using the Plan-Do-Check-Act, or Deming-Cycle [6]
- **Efficiency in *when* to do it -** doing things at the right time, in the right order, optimally synchronizing the work
- **Using TimeBoxing,** which is much more efficient than FeatureBoxing. TimeBoxing continuously sets deadlines. FeatureBoxing, waiting until things are done, has no deadline, causing Parkinson's Law [4] and the Student Syndrome [5] to kick in badly.

Preflection, foresight, prevention

- Only if we change our way of working, the result may be different
- Hindsight is easy, but reactive: the time is already spent
- Foresight is less easy, but proactive:
  we can still avoid wasting time
- 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


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

Estimation techniques

Changing from *optimistic* to *realistic* estimation
In the Evo TaskCycle [8] we estimate the effort time for a Task in hours. The estimations are TimeBoxes, within which the Task has to be completely done, because there is not more time. Tasks of more than 6 hours are cut into smaller pieces and we completely fill all plannable time (i.e. 26 hours, 2/3 of the 40hr available time in a work week). The aim in the TaskCycle is to learn what we *can promise* to do and then to *live up to our promises*. If we do that well, we can better predict the future. Experience by the author shows that people can change from optimistic to realistic estimators in just a few weeks, once we get *serious about time*. At the end of every weekly cycle, all planned Tasks are done, 100% done. The person who is going to do the Task is the only person who is entitled to estimate the effort needed for the Task and to define what 100% done means. Only then, if at the end of the week a Task is not 100% done, that person can feel the pain of failure and quickly learn from it to estimate more realistically the next week. If we are not serious about time, we'll never learn, and the whole planning of the project is just quicksand*!*

0^{th} order estimations
0^{th} order estimations, using ballpark figures we can roughly estimate, are often quite sufficient for making decisions. Don't spend more time on estimation than necessary for the decision. It may be a waste of time. We don't have time to waste. Any number is better than no number. If a number seems to be wrong, people will react and come up with reasoning to improve the number. And by using two different approaches to arrive at a number we can improve the credibility of the number.

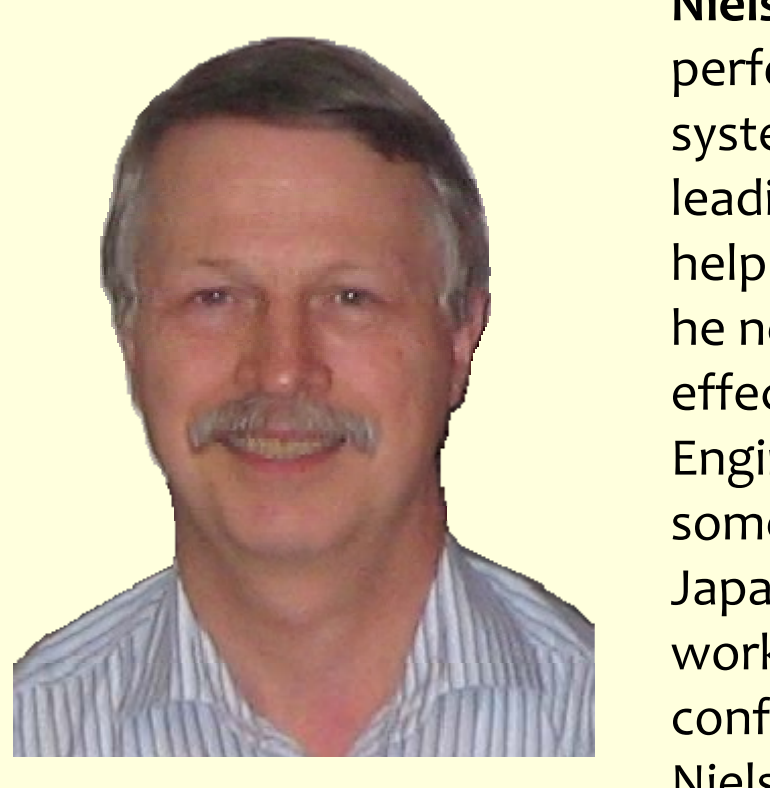
Simple Delphi
If we've done some work of small complexity and some work of more complexity, and measured the time we needed to complete those, we are more capable than we think of estimating similar work, even of different complexity. A precondition is that we become aware of the time it takes us to accomplish things. There are many descriptions of the Delphi estimation process [7], but, as always, we must be careful not to make things more complicated than absolutely necessary. Anything we do that's not absolutely necessary takes time we could save for doing more important things*!*
Our Simple Delphi process goes like this:

1. Make a list of things we think we have to do in just enough detail. Default: 15 to 20 chunks.
2. Distribute this list among people who will do the work, or who are knowledgeable about the work.
3. Ask them to add work that we apparently forgot to list, and to estimate how much time the elements of work on the list would cost, "as far as you can judge".
4. In a meeting the estimates are compared.
5. If there are elements of work where the estimates differ significantly between estimators, *do not take the average*, and do not discuss the *estimates*. Discuss the *contents* of the work, because apparently different people have a different idea about what the work includes. Some may forget to include things that have to be done, some others may think that more has to be done than has to be done.
6. After the discussion, people estimate individually again and then the estimates are compared again.
7. Repeat this process until *sufficient* consensus is reached (usually repeating not more than once or twice).
8. Add up all the estimates to end up with an estimate for the whole project.

Don't be afraid that the estimates aren't exact. They aren't. By adding many estimations, however, the variances tend to average and the end result is usually not far off. Estimates don't have to be exact, as long as the average is OK. Using Parkinson's Law in reverse, we now can fit the work to fill the time available for its completion. We use calibration to measure the real time vs. estimated time ratio, to extrapolate the actual expected time needed.

References

[1] Cohn, Mike: *Agile Estimating and Planning*, 2005, Prentice Hall PTR
[2] Brooks, F.P: *The Mythical Man-Month*, 1975, Addison Wesley. Reprint 1995
[3] Putnam, Doug: *Team Size Can Be the Key to a Successful Project*, www.qsm.com/process_01.html
[4] Parkinson, C. Northcote: *Parkinson's Law, 1955*, alpha1.montclair.edu/~lebelp/ParkinsonsLaw.pdf
[5] Goldratt, E.M.: *Critical Chain*, Gower
[6] Deming, W.E.: *Out of the Crisis*, 1986, MIT
[7] Delphi process: www.iit.edu/~it/delphi.html
[8] Malotaux, Niels: *How Quality is Assured by Evolutionary Methods*, 2004, www.malotaux.nl/nrm/pdf/Booklet2.pdf
Malotaux, Niels: *Controlling Project Risk by Design*, 2006, www.malotaux.nl/nrm/pdf/EvoRisk.pdf
Malotaux, Niels: *TimeLine: Getting and Keeping Control over your Project*, 2007, www.malotaux.nl/nrm/pdf/TimeLine.pdf
Gilb, Tom: *Principles of Software Engineering Management*, 1988, Addison Wesley
Gilb, Tom: *Competitive Engineering*, 2005, Elsevier



Niels Malotaux is an independent Project Coach specializing in optimizing project performance. He has over 30 years experience in designing hardware and software systems, at Delft University, in the Dutch Army, at Philips Electronics and 20 years leading his own systems design company. Since 1998 he devotes his expertise to helping projects to deliver Quality On Time: delivering what the customer needs, when he needs it, to enable customer success. To this effect, Niels developed an approach for effectively teaching Evolutionary Project Management (Evo) Methods, Requirements Engineering, and Review and Inspection techniques. Since 2001, he taught and coached some 80 projects in 20+ organizations in the Netherlands, Belgium, Ireland, India, Japan, Romania and the US, which led to a wealth of experience in which approaches work better and which work less well in practice. He is a frequent speaker at conferences, see www.malotaux.nl/nrm/Conf .
Niels can be reached at niels@malotaux.nl - www.malotaux.nl - +31 - 655 753 604