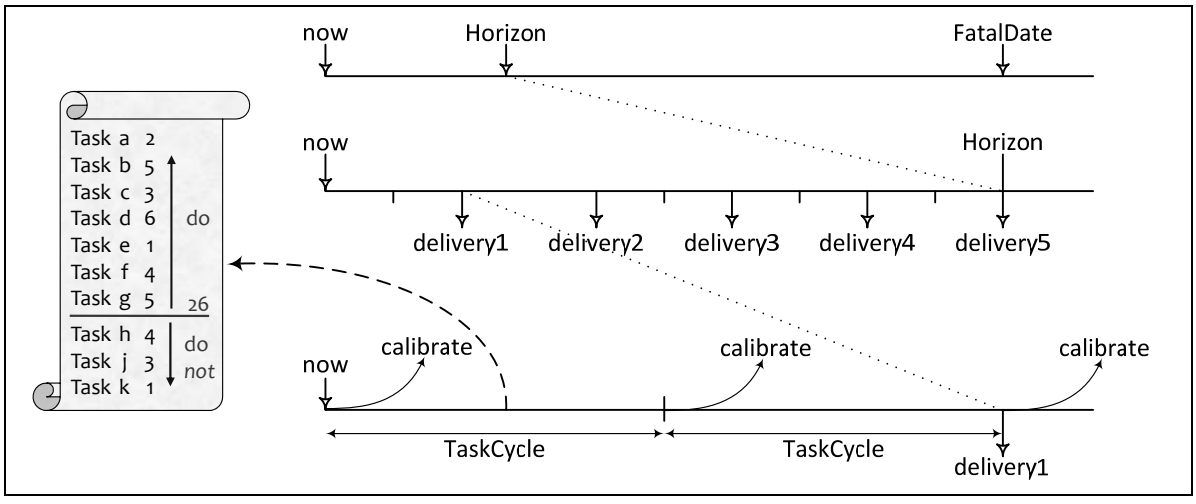
Niels Malotaux

Evolutionary Planning

or

How to Achieve the Most Important Requirement



Niels Malotaux

Evolutionary Planning
or
How to Achieve the Most Important Requirement

1 The Most Important Requirement

The most important requirement for most projects is time - *time for completion*. Projects are supposed to generate a considerable Return on Investment. Therefore the cost of one day delay is not only the cost of running the project one day longer, but also the cost of not being ready one more day (cost of people or equipment waiting, missed revenue, etc), which is usually a lot more than the cost of the project itself. Project delay is costly.

Still, most projects are late. Isn't it weird that projects apparently judge all other requirements to be more important than the requirement of time while time is one of the most important requirements? Both Project Management (responsible for the project) and Systems Engineering (responsible for the product) are responsible for the consequences of ignoring this important requirement.

2 Are Systems Engineers Interested in Time?

Many people in projects, including Systems Engineers, think that the delivery time of the project result is not their responsibility, but rather the responsibility of project management. They also seem to think that the system is ready only if "all" requirements have been met. The existence of this thinking is the very reason of producing this booklet.

All people working in a project spend time and should spend their time wisely, taking into account the impact of their decisions on the success of the project. Where "other" engineers still may be accused of silo-thinking, the very reason of Systems Engineering is to avoid silo-thinking, taking responsibility for a multi-dimensional variety of issues: whole lifetime (*cradle* to *cradle*), over *all* disciplines (including e.g. human behaviour, see [12]), balancing all systems requirements, including performances, and optimizing the design decisions over *all* requirements, *including* delivery time.

The engineers who designed and built the baggage handling system of London Heathrow Airport Terminal 5 claimed that their system was a huge technical success and that the failure to get tens of thousands of bags on board of the proper aircraft was caused by "human error". After all, the terminal was delivered on time and on budget, which admittedly was quite an achievement. However, a passenger is not interested in the technical detail of baggage handling at an airport. The passenger checking in his baggage expects to receive it back in correct condition as quickly as possible after arriving at his destination. That's what performance is about. How this is achieved is irrelevant to the passenger. The "system", as seen by an essential group of users (the passengers - without passengers there wouldn't even be an issue), was not delivered properly on time and the delays caused a lot of inconvenience and extra costs.

3 Why are Projects Late?

If we ask people of a project why they are late, they have a lot of excuses, usually external factors being the cause of delays. If we ask them what we could have done about it, they easily have suggestions. We usually know why we are late and we know ways to do something about it. The problem is that we don't *do* something about it. One of the problems is that customers fatalistically think that this is the way it is and keep paying. If the customers would insist on the delivery date or else wouldn't pay, the problem would have been solved a long time ago.

Some typical causes of delay are:

- Unclear Requirements
- Changing requirements (they *do* change anyway)
- No Stakeholder feedback
- No adequate planning
- No adequate communication
- Misunderstanding
- Waiting (before and during the project)
- Indecisiveness
- No Sense of Urgency
- Doing things wrong
- Doing unnecessary things
- Doing things over again
- Doing things less cleverly than we could
- Suppliers being late
- Suppliers delivering inadequate quality
- Hobbies
- Political ploys
- Boss is always right (cultural issues)

The only justifiable cost is the cost of developing the right things at the right time. This looks like perfection and we know that people are not perfect. That is, however, not a license to fatally accept all these delays. A lot of delay is avoidable and therefore unjustifiable.

4 Why is Time so Important?

We run a project to design and realize a new system, because the new system improves upon previous performance. If it doesn't, there is no reason for the system to be realized. The improvement (e.g. less loss, more profit, faster achieving the same, doing more in shorter time, being happier than before) should have a value way more that the cost of the project.

Initially, every day of the project adds value, but towards the end of the project we are in the area of diminishing returns and every extra day may add less than the return we would gain by the use of the result of the project. This calls for a constant attention to the business case, and a requirements, architecture and design process that optimizes the opportunities and challenges of the business case. This puts the attention to *delivery time* right in the centre of the Systems Engineering activities. In some cases some extra time can significantly increase the performance of the system, however, in other cases spending less time can also increase the revenues from the system: the longer the development takes, the longer the users have to wait for the enhanced performance that the project will provide. Time is money and we don't have the right to waste it, unless it's our own.

If the system we are realizing is a part for a larger system, the system integrator (our customer if we are a sub-contractor) prepares other systems, people and equipment to do the integration into his system at a certain time. He also alerts the potential users of the system that they can start reaping the benefits of the new system at a certain time. If he doesn't get our sub-system on time, he's losing money, the other systems, people and equipment staying idle, while the potential users of the system also have to change their plans. The cost of one day of delay to our customer and the deprived benefit to the ultimate users is a lot more than we realize.

Even doing nothing is a cost factor. Managers often think that there is no cost involved when people are not (yet) working on a project. This is a misconception. Once the idea of the project is born, the timer starts ticking. Every day we start a project later, it will be finished a day later, depriving us from the revenues which by definition are higher that the cost of the project, otherwise we shouldn't even start the project. The only good reason why we delay this project is that we are spending our limited resources on *more profitable* projects.

5 The Fallacy of "All Requirements"

In many projects people say: "*All* requirements have to be done, and it simply takes as much time as it takes; we cannot stop before *all* is done". What *all* is, usually isn't really clear and should be defined by the requirements, which have to be in tune with the business case, which in most projects isn't clear to the project either. These people for some strange reason forget that delivery time is as much a requirement as "all" other requirements.

Systems Engineers are supposed to know how to define real requirements, and they also know that defining the right requirements is not easy. For most customers, defining requirements is not a part of their normal work, so for customers this is even more difficult. How can we expect that customers can properly provide us with the right requirements?

Customers specify things they do not really need and forget to specify things they do need. It's the challenge for the Systems (or Requirements) Engineer to find the real relevant requirements, together with all of the relevant Stakeholders. Furthermore, the Requirements are what the Stakeholders require, however, for a project, the Requirements are what the project is *planning to satisfy*. After all, we can make great systems, but if the customer cannot afford the cost, or has to wait a long time, we both lose.
Because there are always *conflicting* requirements (e.g. more performance can be at odds with acceptable development time or cost), the design process is there to balance and come to an optimum compromise between the conflicting requirements. The notion of "all" requirements pretends that "all" requirements can be met concurrently. If this were the case, projects would be a lot easier. We know better.

6 How to Meet the Most Important Requirement

There are many things we can do to save time in order to get the result of our project on time. As soon as we see that it's impossible to be on time, we can tell our customer and discuss what we do with this knowledge. If we tell the customer only at the end of the project, he really has a problem. If we tell it as soon as we *could have* known, which is much, much earlier in the project, the customer may not like it, but he has more time to cope with the consequences.
In the remainder of this booklet we'll first discuss the options we (seem to) have to get our project result earlier. Then we'll discuss the techniques that are available to really actively make sure that we always will be on time.

7 Which Options Do We (seem to) Have to be On Time?

What can we do if what we think[1] we have to do doesn't fit the available time, or if we want to do things faster? There are several ways we see people use to try to finish a project earlier, most of which are intuitively right, but don't work. This contradiction causes people to think that we have to accept late projects as a fact of life. After all, they did their best, even took measures (correct measures according to their intuition), and it didn't work out. There are, of course, also measures that do work.

Deceptive measures
Let's first do away with the deceptive measures. Deceptive measures are measures we often see applied, but which don't work. It's surprising that people don't learn and keep using them.

7.1 **Hoping for the best** (fatalistic type)

Most projects take more time than expected. Your past project took longer than expected. What makes you think that this time it will be different? If you don't change something in the way you run the project, the outcome won't be different, let alone better. Just hoping that your project will be on time this time won't help. We call this ostriching: putting your head into the sand waiting until Murphy[2] strikes again.

7.2 **Going for it** (macho type)

We know that the available time is insufficient, but it *has* to be done: "Let's go for it!" If nothing goes wrong (as if that ever is the case) and if we work a bit harder (as if we don't already work hard) … Well, forget it.

7.3 **Working Overtime** (fooling yourself)

Working overtime is fooling yourself: 40 hours of work per week is already quite hard. If you put in more hours, you'll get more tired, make more mistakes, having to spend extra time to find and "fix" the mistakes, half of which you won't. You think you are working hard, but you aren't working *smart*. It won't work. This is also ostriching. As a rule, never work overtime, so that you have the energy to do it once or twice a year, when it's really necessary.

---

[1] We keep saying "what we *think* we have to do", because however good the requirements are, they will change, because *we* learn, *they* learn and the circumstances change. The longer the project, the more the requirements have a chance to change. And they will change! However, what we do not yet know, we cannot yet plan for.

[2] *Whatever can go wrong, will go wrong* is the popular version of Murphy's Law. The real version is: *What can go wrong, will go wrong, so we have to predict all possible ways it can go wrong, and make sure that these cannot happen.* Spark [1].

7.4 Adding time: moving the deadline

Moving the deadline further away is also not a good idea: the further the deadline, the more danger of relaxing the pace of the project. We call this Parkinson's Law[3] or the Student Syndrome[4]. At the new deadline we probably hardly have done more, getting the project result even later. Not a good idea, unless we really are in the nine mother's area (see next), where nobody, even with all the optimization techniques available, could do it. Even then, just because of the Student Syndrome, it's better to optimize what we can do in the available time before the deadline. The earlier the deadline, the longer our future afterwards, in which we can decide what the next best thing there is to do. So the only way a deadline may move is *towards* us. We better optimize the time spent right from the beginning, because we'll probably need that time anyway at the end. Optimizing only at the end won't bring back the time we lost at the beginning. Optimizing only towards the end also means that there is much less we still can optimize.

7.5 A riskful measure: adding people …

A typical move is to add people to a project, in order to get things done in less time. Intuitively, we feel that we can trade time with people and finish a 12 person-month project in 6 months with 2 people or in 3 months with 4 people, as shown in Figure 1. In his essay *The Mythical Man-Month*, Brooks [2] shows that this is a fallacy, defining Brooks' Law: *Adding people to a late project makes it later*.

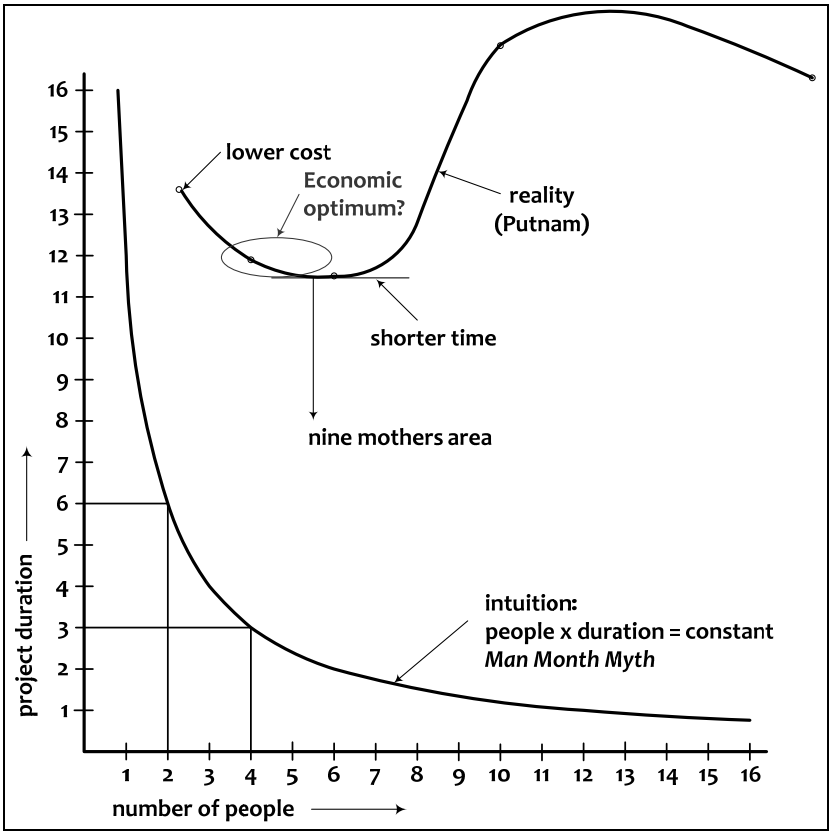


Figure 1: The Myth of the Man-Month: reality is completely different

Putnam [3] confirms Brook's Law with measurements on some 500 (software) projects. He found that if the project is done by 2 or 3 people, the project-cost is minimized, while 5 to 7 people achieve the shortest project duration at premium cost, because the project is only 20% shorter with double the amount of people. Adding even more people makes the project take longer at *excessive cost*. Apparently, the project duration cannot arbitrarily be shortened, because there is a critical path of things that cannot be

[3] Parkinson's Law: "*Work expands so as to fill the time available for its completion*" (People use the time given). Parkinson [4] observed: "Granted that work (and especially paperwork) is elastic in its demands on time, it is manifest that there need be little or no relationship between the work to be done and the size of the staff to which it may be assigned."

[4] Starting as late as possible, only when the pressure of the Fatal Date is really felt. Term attributed to E. Goldratt [5].

parallelized. We call the time in which nobody can finish the project the *nine mothers area*, which is the area where nine mothers produce a baby in one month. When I first heard about Brooks' Law, I assumed that he meant that we shouldn't add people at the *end* of a project, when time is running out. After all, many projects seem to find out that they are late only by the end of the project. The effect is, however, much trickier: if in the *first several weeks* of a project we find that the speed is slower than expected, and thus have to assume that the project will be late, even then adding people can make the project later. The reason is a combination of effects. More people means more lines of communication and more people to manage, while the project manager and the architect or the Systems Engineer can oversee only a limited number of people before becoming a bottleneck themselves. Therefore, adding people is not automatically a solution that works. It can be very risky.

How can those mega-projects, where 100's of people work together, be successful? Well, in many cases they are not. They deliver less and later than the customer expects and many projects simply fail. The only way to try to circumvent Brooks' Law is to work with many small teams, who can work in parallel, and who synchronize their results only from time to time. If you think Brooks' Law won't bite you, you better beware: it will!

In a recent project that went too slow, the number of people was increased from 5 people to 20 people. The measured productivity increased by 50%. It took project management quite some time to decide to decrease (against their intuition!) the number of people from 20 to 10. Once they did, the net productivity of the 10 people was the same as with 20 people.

7.6 The Measure That Always Works: Saving Time

Fortunately, there are ways to save time, *without negatively affecting the Result of the project* (*on the contrary!*). These techniques are collected and routinely used in the Evolutionary Project Management (Evo) approach in order to achieve the best solution in the shortest possible time.

The Evo approach uses and constantly evolutionarily optimizes the elements of saving time: Plan-Do-Check-Act cycles (or Deming cycles - Deming [6]), Zero-Defects attitude (Crosby [7]), Business Case techniques, specific Requirements Management techniques [8,11], Design techniques, Early Reviews, and Evolutionary Planning techniques like TaskCycles, DeliveryCycles and TimeLine. Background of the Evo approach can be found in Gilb [8] and Malotaux [9,10,11,12]. Projects starting to use the Evo approach start saving 30% time within a few weeks, while delivering better results.

The elements of saving time are:

Improving the efficiency in *what* (*why*, for *whom*) we do: doing only what is needed, not doing things that later prove to be not needed, preventing mistakes and preventing working on superfluous things. Because people tend to do more than necessary, especially if the goals aren't clear, there is ample opportunity for *not* doing what is *not* needed. We use the *Business Case* and *continuous Requirements Management* to control this process. We use the *TaskCycle*, to weekly decide what we are going to do and what we are *not going* to do, *before* we do it. This saves time. Afterwards we only can identify what we unnecessarily did, but the time is already spent and cannot be regained.

Improving the efficiency in *how* we do it: doing things differently.

This works in several dimensions:

The product

Choosing the proper and most efficient solution. The solution chosen determines both the performance and cost of the product, as well as the time and cost of the project. Because performance and project time are usually in competition, the solution should be an optimum compromise and not just the first solution that comes to mind. We use *Architecture* and *Design* processes to optimize the result. We use *DeliveryCycles* to check the requirements and assumptions with the appropriate Stakeholders.

The project

We can probably do the same in less time if we don't immediately do it the way we always did, but first think of an alternative and more efficient way. We do not only design the product, we also continuously *redesign the project*. We use *Evolutionary Planning* to control this process.

Continuous improvement and prevention processes

Actively and constantly learning how to do things better and how to overcome bad tendencies. We use

rapid and frequent Plan-Do-Check-Act (PDCA or Deming) cycles to actively improve the product, the project *and* the processes. We use *Early Reviews* to recognize and tackle tendencies before they pollute our work products any further and we use a *Zero-Defect attitude* because that is the only way ever to approach Zero Defects.

Improving the efficiency of *when* we do it: doing things at the right time, in the right order. A lot of time is wasted by synchronization problems like waiting for each other, or redoing things that were done in the wrong order. *Actively Synchronizing* and *designing* the order of what we do saves time. We use *Evolutionary Planning* with constant, active prioritization to control this process, with *TaskCycles* and *DeliveryCycles* to make sure we do the right things in the right order, and TimeLine to get and keep the whole project under control. Elements of these are *Just Enough Estimation*, *Dynamic Prioritizing* and *Calibration* techniques.

All of these elements are huge time savers. Of course we don't have to wait for a project getting into trouble. We also can apply these time savers if what we think we have to do easily fits in the available time, to produce results even faster. We may even need the time saved to cope with an unexpected drawback, in order still to be on time and not needing any excuse.

TimeBoxing provides the incentive to constantly apply these ways to save time, in order to stay within the TimeBox. For TimeBoxing to work properly, it is important to change from optimistic or pessimistic, to realistic estimation. If the TimeBox is too short, we cause stress with adverse effects. If the TimeBox is too long, we're wasting time. In the experience of the author, people in projects can easily change into realistic estimators in a few weeks time, if and only if we are *serious about time*. TimeBoxing is much more efficient than FeatureBoxing (= waiting until we're ready), because with FeatureBoxing we lack a deadline, causing Parkinson's Law and the Student Syndrome to kick in badly.

Note that this concept of saving time is similar to "eliminating waste" in Lean thinking, as already indicated by Henry Ford in his book "My Life and Work", back in 1922 [13]: "*We eliminated a great number of wastes*". Dr. Deming also mentioned "Not so much waste" on page 1 of his book *Out of the Crisis* [6].

Because the time saving actions don't come easy (otherwise this would be practiced already everywhere), it's advisable for Systems Engineering to work together and synchronize adequately with Project Management, to constantly seek for ways to improve on this. We suggest studying the Evolutionary approach and using it to the advantage of the project success.

8 Evolutionary Planning

The Evolutionary Planning process uses three main elements, see Malotaux [9,10]:

- **The weekly TaskCycle** to organize the work, to make sure we are at any time working only on the most important things and don't work on less important things. We quickly learn to promise what we can do and then to live up to our promises. This removes a lot of quick-sand from under the project.
- **The bi-weekly DeliveryCycle** to check the requirements and challenge our assumptions and perceptions.
- **TimeLine** to get and keep control over longer periods of time and to provide reliable status information to the project, as well as to Portfolio/Program/Resource management.

We'll show now how these three elements fit together to get and keep the project under control.

TimeLine. In many projects all the work we think we have to do is cut into pieces, the pieces are estimated, and the estimates are added up to arrive at an estimate of the effort to do the work. Then this is divided over the available people (however, beware of Brooks' Law!), to arrive at an estimate of the duration of the work, which, after adding some contingency, is presented as the duration of the project (Figure 2).

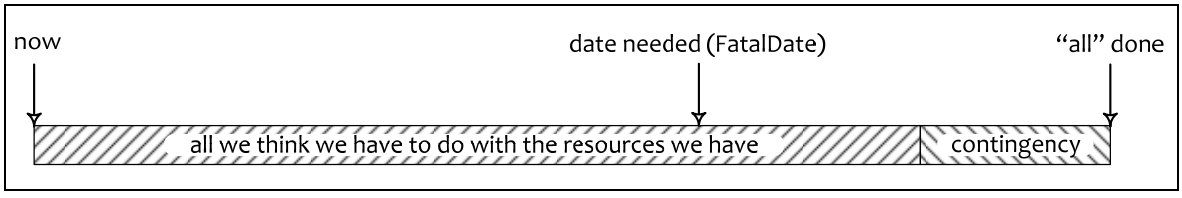


Figure 2: Standard approach: it takes what it takes, but often that's too late

A problem is that in many cases the required delivery date is earlier. The tension between estimated and expected delivery causes extra time spent in discussions, while the required delivery date doesn't change, leaving even less time for the project.
Because the delivery date is a requirement just as all the other requirements, it has to be treated as seriously as all the other requirements. With TimeLine, we treat delivery dates seriously and we meet these dates, or we very quickly explain, based on facts, why the delivery date cannot be met.
We don't wait until the FatalDate to tell that we didn't make it, because if it's really impossible, we know it much earlier. If it is possible, we deliver, using all the time-saving techniques to optimize what we can deliver when.
TimeLine can be used on any scale: on a program, a project, a sub-project, on deliveries, and even on tasks. The technique is always the same. We estimate what we think we have to do, see that we need more time than we have and then discuss the TimeLine with our customer or other appropriate Stakeholders and explain (Figure 3):

- What, at the FatalDate, surely will be done
- What surely will not be done
- What may be done (after all, estimation is not an exact science)

If what surely will be done is not sufficient for success, we better stop now to avoid wasting time and money. Note that we put what we plan in strict order of priority, so that at the FatalDate at least we'll have done the most important things. Customers don't mind about the bells and whistles if Time to Market is important. Because priorities may change very dynamically, we have to constantly reconsider the order of what we do when.

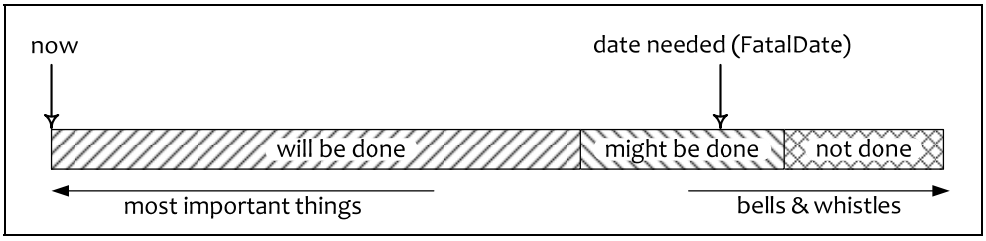


Figure 3: Basic TimeLine: what will surely be done, what will not be done, and what may be done

Setting a Horizon. If the total project takes more than 10 weeks, we define a Horizon at about 10 weeks on the TimeLine, because we cannot really oversee longer periods of time. A period of 10 weeks proves to be a good compromise between what we can oversee, while still being long enough to allow for optimizing the order in which we deliver results. We don't forget what's beyond the horizon, but for now, we concentrate on the coming 10 weeks.

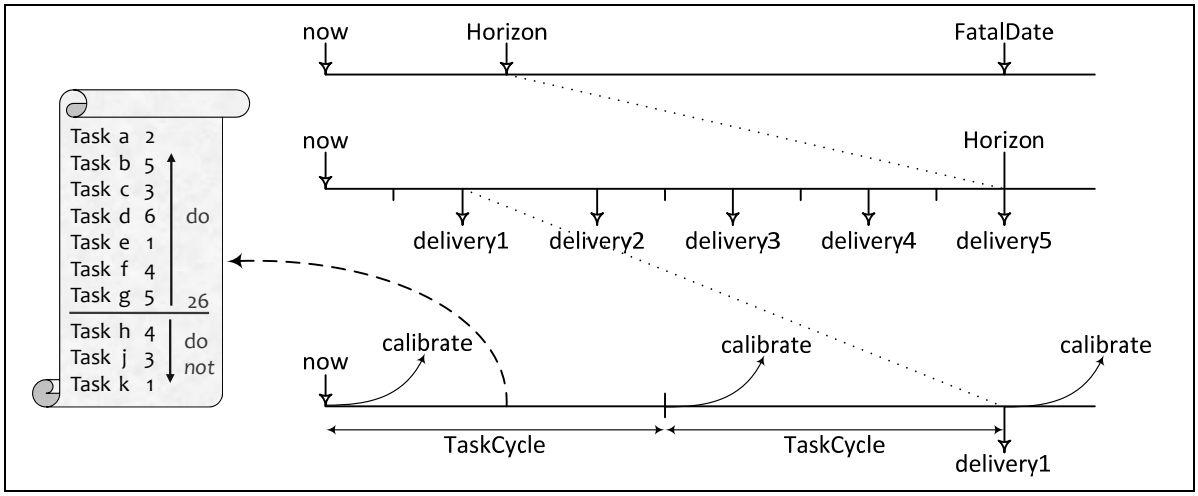


Figure 4: TimeLine summary: setting a FatalDate, a Horizon, Deliveries, TaskCycles, and then calibrating back

DeliveryCycles. Within these 10 weeks, we plan DeliveryCycles (Figure 4) of maximum 2 weeks, asking: "*What* are we going to deliver to *whom* and *why*?" Deliveries are for getting feedback from Stakeholders. We are humble enough to admit that our (and their) perception of the requirements is not perfect and that many of our assumptions may be incorrect. Therefore we need communication and feedback. We deliver to *eagerly waiting* Stakeholders, otherwise we don't get feedback. If the appropriate Stakeholders aren't eagerly waiting, either they're not interested and we may better work for other Stakeholders, or they *have to* be made eagerly waiting by delivering what we call *Juicy Bits*. How can juicy bits have a high priority? If we don't get appropriate feedback, we will probably be working based on incorrect assumptions, causing us to doing things wrong, which will cause delay later. Therefore, if we need to deliver juicy bits to Stakeholders to make them eagerly waiting in order to get the feedback that we awfully need, this has a high priority.

TaskCycles. Once we have divided the work over Deliveries, which are Horizons as well, we now concentrate on the first few Deliveries and define the actual work that has to be done to produce these Deliveries. We organize this work in TaskCycles of one week. In a TaskCycle we define Tasks, estimated in net effort-hours (see [9], section 6.1, for a more detailed explanation). We plan the work in plannable effort time, which defaults to 2/3 of the available time (26 hrs in case of a 40 hr week), confining all unplannable project activities like email, phone-calls, planning, small interrupts, etc, to the remainder of the time. We put this work in optimum order, divide it over the people in the project, have these people estimate the time they would need to do the work, see that they don't get overloaded and that they synchronize their work to optimize the duration.

9 Just Enough Estimation

There are several methods of estimation. There are also ways to quickly change from optimistic to realistic estimation. An important prerequisite is that we start treating time seriously, creating a Sense of Urgency and that we *care* about time. It is also important to learn how to spend *just enough* time on estimation. Not more and not less.

9.1 Changing from *optimistic* to *realistic* estimation

In the Evo TaskCycle we estimate the effort time for a Task in hours. The estimates 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 only 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!

9.2 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.

Example: How can we estimate the cost of one month delay of the introduction of our new product?

How about this reasoning: The sales of our current most important product, with a turnover of about $20M per year, is declining 60% per year, because the competition introduced a much better product. Every month delay costs about 5% of $20M, being $1M. Knowing that we are losing about $1M a month, give or take $0.5M, could well be enough to decide that we shouldn't add more bells and whistles to the new product, but rather finalize the release. Did we need a lot of research to collect the numbers for this decision …?

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.

9.3 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 [14], 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* (estimates are not negotiable!). 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. Making more clear what has to be done and what has not to be done, usually saves time.
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 won't be anyway. By adding many individual estimates, 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 (see chapter 11).

In a recent case, to save even more time on the estimation process, we used "Simpler Delphi": in stead of steps 6 and 7 of the process shown, we took the minimums and maximums of the individual estimates, and then decided by quick consensus which time (within the min-max range) to use. This short-cut worked quite well.

9.4 Estimation tools

There are several estimation methods and tools on the market, like e.g. COCOMO [15], QSM-SLIM [16] and Galorath-SEER [17]. These tools rely on historical data of lots of projects as a reference. The methods and tools provide estimates for the optimum duration and the optimum number of people for the project, but have to be tuned to the local environment. With the tuning, however, a wide range of results can be generated, so how would we know whether our tuning provides better estimates than our trained gut-feel? The use of tools poses some risks:

- For tuning we need local reference projects. If we don't have enough similar (similar people, techniques, environments, etc …) reference projects, we won't be able to tune. So the tools may work better in large organizations with a lot of similar projects.
- We may start working for the tool, instead of having the tool work for us. Tools don't pay salaries, so don't work for them. *Only* use a tool if it provides good Return on Investment (RoI) for you.
- A tool may obscure the data we put in, as well as obscure what it does with the data, making it difficult to interpret what the output of the tool really means, and *what we can do to improve*. We may lose the connection with our gut-feel, which eventually will have to make the decision.

Use a tool only when the simple Delphi and 0^{th} order approaches, combined with realistic estimation rather than optimistic estimation, really prove to be insufficient and if you have sufficient reasons to believe that the tool will provide good ROI.

10 Calibration

Having estimated the work that has to be done in the first week, we have captured the first metrics for calibrating our estimates on the TimeLine. If the Tasks for the first week would deliver about half of what we need to do in that week, we now can extrapolate that our project is going to take twice as long, *if* we keep working the way we did, that is: *if we don't do something about it*. Initially the data of the first week's estimate may seem weak evidence, but it's already an indication that our estimates may be too optimistic. Putting our head in the sand for this evidence is dangerous: I've heard all the excuses about "one-time causes". Later there were always other "one-time causes".

One week later, when we have the *actual* results of the first week, we have slightly better numbers to extrapolate and scale how long our project really may take. Week after week we will gather more information with which we calibrate and adjust our notion of what will be done at the FatalDate or what will be done at any earlier date. This way, the TimeLine process provides us with very early warnings about the risks of being late. The earlier we get these warnings, the more time we have to *do something about it*.

Let's take an example of taking the consequence of the TimeLine (Figure 5): 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). We start with 5 people. After 4 weeks, we find that 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 take twice as long (line d) or produce only half (line c). If the deadline is really hard, a typical reaction of management is to throw more people at the project (line e). However, based on our progressing understanding of the work, we found that we forgot to plan some work that also "has" to be done: we now think we still have to do 50 WoW in the remaining 6 weeks (line f).

Management decides to add even more people to the project, because they don't want the project to take longer. You can even calculate how many people management will have to add, based on the numbers in the example of figure 5. This solution, however, won't produce the desired outcome, and even work out counterproductive, because of Brooks' Law.

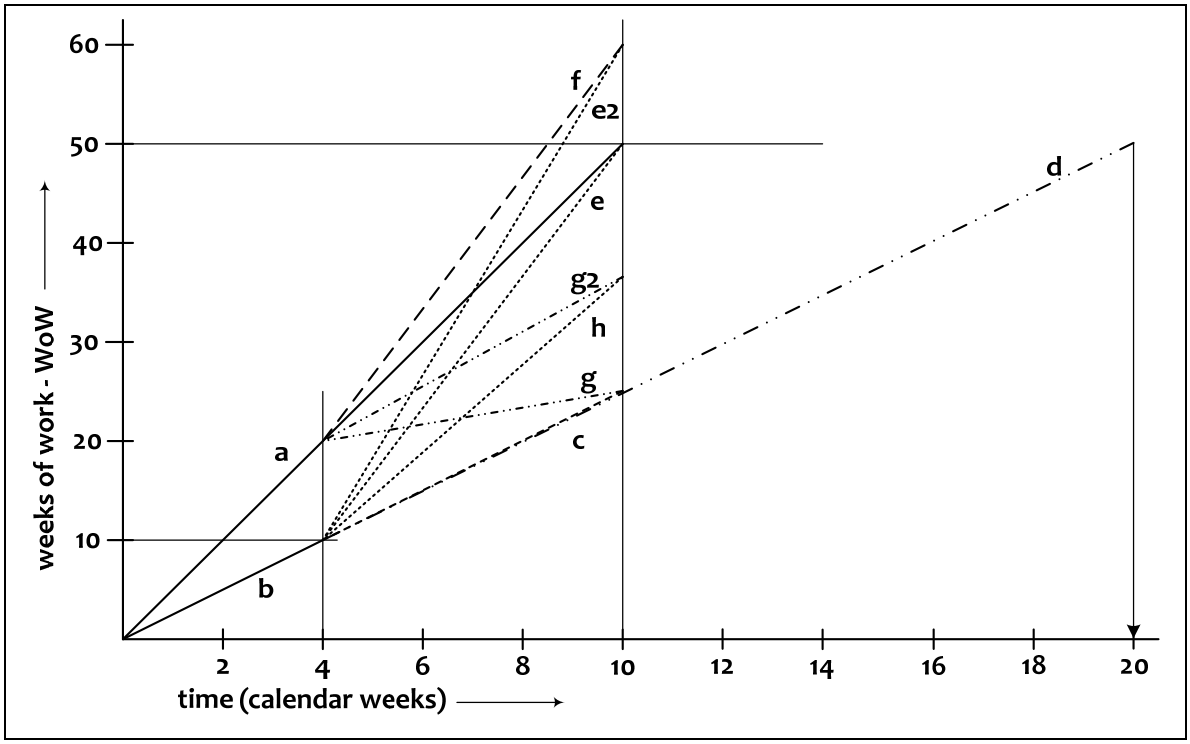


Figure 5: Earned Value (up to week 4) and Value Still to Earn (from week 5)

We can counter this dilemma by *actively saving time*, doing only what is really necessary (line g), or a combination of *not* doing what is *not* necessary (line g2) and doing things more productively (line h), as explained in section 7.6. Actively *designing* what exactly to do and in which order, saves a lot of time.

11 Predicting the future.

Using calibration, we can quite well predict what will be done when (figure 6). The estimates don't have to be exact, as long as the relative values are consistent: if Activity1 is estimated to take 2 (units of estimation) and Activity2 to take 1, then we assume that Activity1 will take twice as long as Activity2. We see that people are reluctant to accept that rather imprecise estimates yield rather good overall predictions. In practice, the positive and negative inaccuracies average out, providing a quite good accuracy of the summed total. Some people are even reluctant to estimate at all, being afraid to fail their estimates. However, if you don't have an estimate to fail on, you cannot learn, while the experience of failure makes us learn quickly, as long as we *want* to learn.

Once we have done several Activities, we know how long these activities took and now we can calibrate the remainder of the estimates to reality. We average the calibration factor over several recent activities until now:

$$\text{Calibration Factor} = \frac{\sum_{now-1}^{now-n} Ar}{\sum_{now-1}^{now-n} Ae}$$

(Ar is real time,
Ae is estimated time of an Activity)

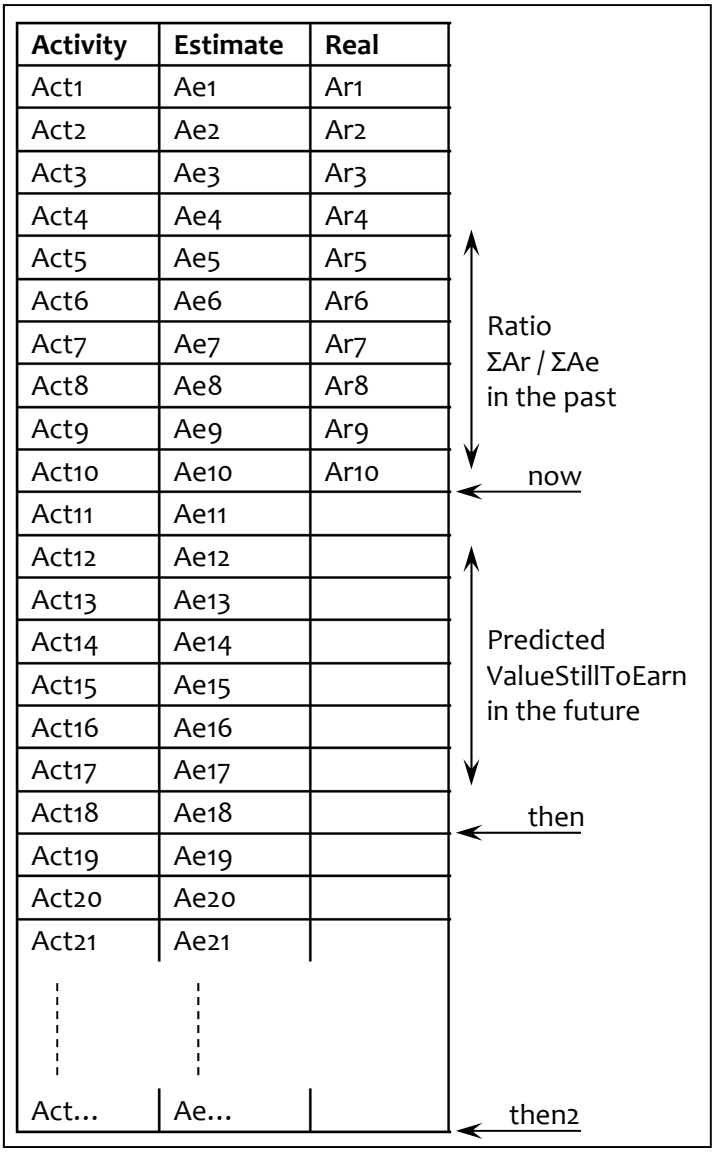


Figure 6: Using the list of activities to predict *what* will be done *when*

Now we can use this calibration factor to predict how much time we need for future activities:

$$\text{Time needed for Value Still to Earn}_{\text{(by then)}} = \text{Calibration Factor} * \sum_{now}^{then} Ae$$

This way we can predict when we will have done what, or when "all" is done.

This list of activities still to do (*Value Still to Earn*) will constantly be updated:

- Activities will be added when we recognize that we forgot some things we have to do
- Activities will be updated when we realize that we have to better define them
- Activities will be deleted, or moved to the bottom once we see that they don't add enough value
- The order of activities will be changed, once we find out that the priorities have changed
- Estimates will be updated to reflect better insight, although the estimates should be made based on the same assumptions as the original estimates, to keep the calibration working

Note that we shouldn't use these numbers mechanistically. We still have to judge the credibility of what the mathematics tell us and adjust our understanding accordingly.

In conventional projects this manual interpretation may still lead to over-optimistic predictions, especially if what the numbers tell us is "undesirable". In Evo projects, however, we want to succeed in the available time or earlier, so we are realistic and rather see any warning we can use to constantly improve, or to discuss the consequences with the customer as soon as possible.

In practice I've seen calibration factors of 2 at the start of the project and then growing and stabilizing at 4 when the project is running at full strength. In some hardware development projects I've seen calibration factors between 1 to 1.5. In other projects we may see yet other factors. Note that the calibration factors of different projects are not good or bad and *cannot be compared*: they are simply the ratio of how much time *this* project needs to accomplish its activities, and the estimates as produced by the project's estimation standard. Different projects have different people and estimate in different ways. They merely calibrate the assumptions used at the original estimate, where they may not have taken into account V&V, SE, project management, education, and many other things that have to be done in the project as well. Once the calibration factor has stabilized, we can use the *slope* of the factor to warn for deterioration and to see the effect of process improvements.

12 Summarizing the TimeLine technique

Summarizing the TimeLine technique:

- Cutting what we think we have to do into up to some 20 chunks (packages, activities) and estimating these chunks. Adding up the estimates usually provides sufficient evidence that we need more time than we have available. At this point, most projects decide that they simply need more time, or complain that management is imposing impossible deadlines.
- With Evolutionary Planning, however, we don't stop here, but think of alternative strategies of doing things, doing different things or doing things differently. We estimate the impact on the result and choose the optimum strategy. Now we have well-founded arguments to explain management why things will take as much as they still will do.
- Now the chosen strategy is planned focused on the optimum order of implementing the optimum solution, still being aware that "optimum" gradually may change by advancing understanding. It's of no use continuing an initial plan once we see that it should be changed. That's why we have to continuously keep using the Plan-Do-Check-Act technique, with the Business Case as a reference.
- Now we can start predicting *what* will be done *when*, based on the estimations and subsequent calibration to reality. This provides the business with quite reliable predictions, allowing them to provide reliable predictions to their customers.

Line	Activity	Estim	Spent	Still to spend	Ratio real/es	Calibr factor	Calibr still to	Date 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

Figure 7: Simplified TimeLine sheet, indicating what will be done when based on estimations and a calibrated future. It also shows what will *not* be done at a certain date, giving us early warnings: on 5 June, Activities 17 and 18 won't be done. The earlier we get a warning, the more time we have to do something about it. Some notes: In this table we *don't* calibrate *Still to Spend* (by using calibration factor 1.0), because of assumed improved insight. Activities not yet started are calibrated by the ratio of *Spent* plus *Still to Spend* and the original estimates. Apparently, this is a snapshot of 29 March.

Figure 7 shows a simplified example of a TimeLine table, stating the Activity-description, the estimate, the time already spent and the time still to spend, the ratio of real and estimated time, the calibration factor (ratio of total real time and estimated time during a past period), the resulting calibrated ("real") time still to spend and the resulting dates. If in this example the project has to be concluded on 5 June, we now can say that Activities 17 and 18 won't be done at that deadline, unless we do something differently. This way, we can very early in a project predict what will be done when and take the consequence of the prediction, rather than sticking our head in the sand until reality hits us somewhere. The biggest hurdle is that most project managers have trouble finding out which activities have to be done, causing starting up this technique to take some time. Once this hurdle is taken, however, it hardly takes time to keep the TimeLine up to date, giving real control over the further prediction and the progression of the project.
Traditionally, Program/Portfolio/Resource Management (PPRM) is based on hope. After all, if most projects are late, planning based on assumed deadlines which are not kept is more a game than management. Once the projects have learnt to sufficiently reliably predict what can be done when, PPRM can finally start really managing. This is what we see happening once this process is in place.
The Ratio real/estimated proved also to be an interesting indicator: an organization outsourcing refactoring and design of software, found that the supplier was quite good in refactoring and debugging, with a ratio slightly less than 1, meaning always done within the estimated time. When they saw ratios of 4 and 6, they checked the type of activities and found that these were all *design* activities. Apparently this supplier wasn't good at design (yet?), or at least at *estimating* design time.

13 Time as a Requirement for the System After the Project

There are many more Time Requirements than only the duration of the project. *During* the project, we can still influence and optimize the time spent on what we think we have to do to realize the system. *After* the project, however, the system is left to its own devices and must perform autonomously. We have no influence on the timings of the system any more and the performances delivered by the new system have to be there *by design*. This is typically a responsibility and required skill of Systems Engineering. Very important for the success of the project, and of essential concern of Evo, however, this is not further elaborated in this booklet.

14 Conclusion

Evolutionary Planning doesn't *solve* our problems. It rather is a set of techniques to plan and early expose the real status of our project. Then, instead of *accepting* an undesired outcome, we have ample opportunity of *doing something about it*. People do a lot of unnecessary things in projects, so it's important to identify those things *before* spending time on them. If we later find out that we did unnecessary things, the time is already spent, and never can be regained. By revisiting the TimeLine every week, we stay on top of how the project is developing and we can easily report to management the real status of the project and also show the consequences of management decisions affecting the project.
Doesn't all this planning take a lot of time? The first few times it does, because we have to learn how to use the techniques. After a few weeks, however, we dash it off and we can start optimizing the results of the project, producing more than ever before. Evolutionary Planning allows us to take our head out of the sand, stay in control of the project and deliver Results successfully, on time. Still, many Project Managers hesitate to start using these techniques. However, after having done it once, the usual reaction is: "Wow! I got much better oversight over the project than I ever expected", and the hesitation is over. Another reaction: "We never did this before. Now we're finally in control!"
These techniques are not mere theory. They're highly pragmatic, and successfully used in many projects coached by the author. The most commonly encountered bottleneck is that no one in the project has an oversight of what exactly the project is supposed to accomplish. This may be why Project Managers hesitate to start using these techniques. If you don't know well what to do, planning isn't easy. Redefining what the project is to accomplish and henceforth focusing on this goal is the first immediate timesaver, with many more savings to follow.
I hear many Systems Engineers say that they know all these things and that they are doing these things already. Be honest. We do know most of the techniques mentioned in this booklet, but do we really use and continuously improve on them? If we really would, we wouldn't need excuses for late deliveries any more.

References

[1] **Spark, Nick T**: *A History of Murphy's Law*, 2006, Periscope Film, ISBN 0978638891

[2] **Brooks, Fred P**: *The Mythical Man-Month*, 1975, Addison Wesley, ISBN 0201006502
Reprint 1995, ISBN 0201835959

[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, http://alpha1.montclair.edu/~lebelp/ParkinsonsLaw.pdf, and www.adstockweb.com/business-lore/Parkinson's_Law.htm

[5] **Goldratt, Eli M:** *Critical Chain*, 1997, Gower, ISBN 0566080389

[6] **Deming, W.E**: *Out of the Crisis*, 1986. MIT, ISBN 0911379010
Walton, M: *Deming Management at Work*, 1990. The Berkley Publishing Group, ISBN 0399516859

[7] **Crosby Philip B**: *Quality is still free*, 1996, McGraw-Hill, ISBN 0070145326

[8] **Gilb, Tom**: *Principles of Software Engineering Management*, 1988, Addison Wesley, ISBN 0201192462
Gilb, Tom: *Competitive Engineering*, 2005, Elsevier, ISBN 0750665076

[9] **Malotaux, Niels**: *How Quality is Assured by Evolutionary Methods*, 2004. Pragmatic details how to implement Evo, based on experience in some 25 projects in 9 organizations
www.malotaux.nl/nrm/pdf/Booklet2.pdf

[10] **Malotaux, Niels**: *TimeLine: Getting and Keeping Control over your Project*, 2007,
www.malotaux.nl/nrm/pdf/TimeLine.pdf

[11] **Malotaux, Niels**: *Controlling Project Risk by Design*, 2006, www.malotaux.nl/nrm/pdf/EvoRisk.pdf

[12] **Malotaux, Niels**: *Recognizing and Understanding Human Behaviour to Improve Systems Engineering Results*, 2008, APCOSE2008, www.malotaux.nl/nrm/pdf/HumanBehavior.pdf

[13] **Ford, Henry**: *My Life and Work*, 1922, www.gutenberg.org/dirs/etext05/hnfrd10.txt

[14] www.iit.edu/~it/delphi.html

[15] **Boehm, Barry**: *Software Engineering Economics*, Prentice-Hall, 1981, ISBN 0138221227

[16] www.qsm.com

[17] www.galorath.com

Niels Malotaux

Evolutionary Planning
or
How to Achieve the Most Important Requirement

The most important requirement for most projects is time - *time for completion*. Still, most projects are late. Isn't it weird that projects apparently judge all other requirements to be more important than the requirement of time while time is one of the most important requirements? Both Project Management (responsible for the project) and Systems Engineering (responsible for the product) are responsible for the consequences of ignoring this important requirement. This booklet describes why it is important to be on time, what measures we can take to make sure we are on time, which often applied intuitive measures *don't work*, and how we can use Evolutionary Planning techniques to make sure that we will be on time, or, if that is simply impossible, to take the consequence. These techniques allow us in the early stages of our project to predict and to optimize what will be ready at a certain time.

Note - This is not a scientific study but rather based on empirical evidence collected by the author while coaching over 100 projects in the past 10 years.

Niels Malotaux is an independent Project Coach and expert in optimizing project performance. He has well over 35 years experience in designing electronic and software systems, at Delft University, the Dutch Army, Philips, and 20 years leading a systems design company. He devotes his expertise to helping projects to deliver Quality On Time: being successful and predictable. To this effect, Niels developed an approach for effectively teaching successful Project Management, Requirements, and Review and Inspections. Since 2001, he taught and coached over 200 projects in 30+ organizations in the Netherlands, Belgium, China, Germany, India, Ireland, Israel, Japan, Romania, South Africa, the UK and the US, which led to a wealth of experience in which approaches work better and which work less in the practice of real projects. He is a frequent speaker at conferences, see www.malotaux.nl/conferences

Find more booklets at: www.malotaux.nl/booklets

1. Evolutionary Project Management Methods
2. How Quality is Assured by Evolutionary Methods
3. Optimizing the Contribution of Testing to Project Success
3_a. Optimizing Quality Assurance for Better Results (same as 3, but now for non-software projects)
4. Controlling Project Risk by Design
5. TimeLine: Getting and Keeping Control over your Project
6. Recognizing and Understanding Human Behaviour
7. Evolutionary Planning (this booklet) (similar to TimeLine, but other order and added predictability)
8. Help! We have a QA problem!

ETA: Evo Task Administration tool - www.malotaux.nl/?id=downloads#ETA

N R Malotaux
Consultancy

Niels R. Malotaux
phone +31-6 5575 3604
mail niels@malotaux.nl
web www.malotaux.nl

Originally prepared for the INCOSE International Symposium, Singapore 2009
Version 2.2a (address) - 26 Jan 2014