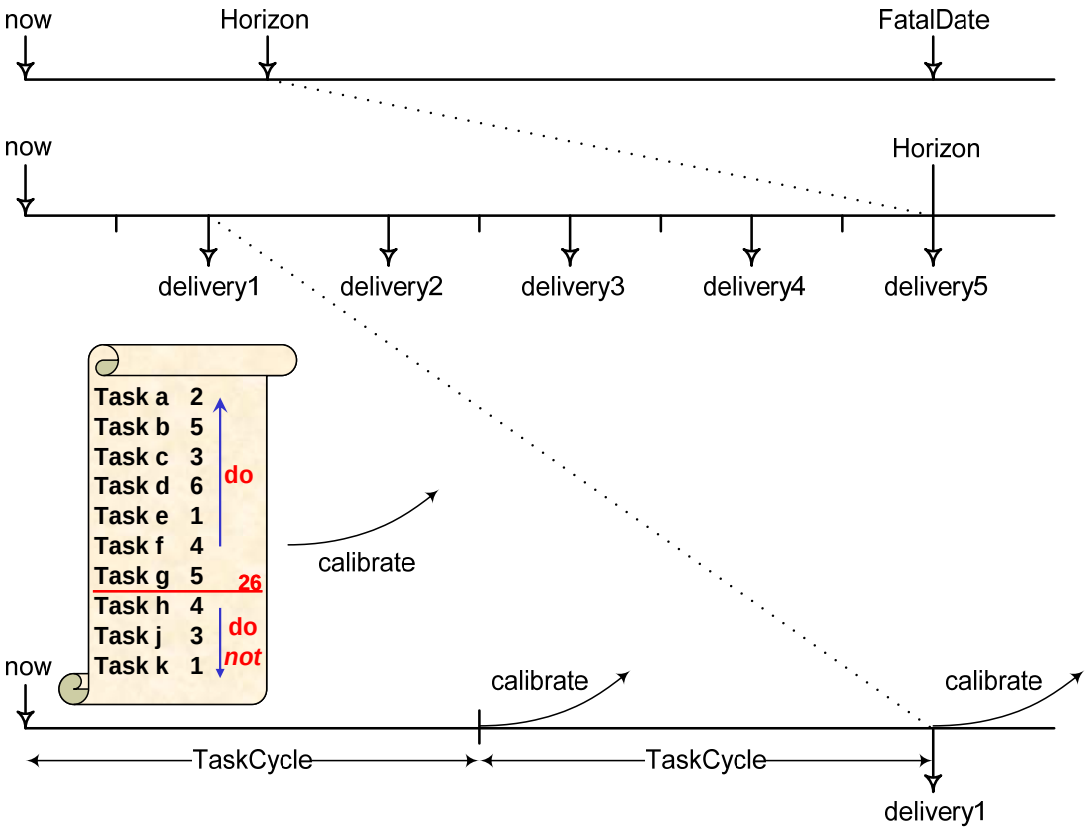
Niels Malotaux

TimeLine

Getting and Keeping Control over your Project



TimeLine

Getting and Keeping Control over your Project

1 How do we get projects on time?

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)

Many projects deliver late. If we don't change our ways, projects will continue to be late. The only way to get projects on time is to change the way we do things. The Evolutionary Project Management (Evo) approach [1] is about continuously introducing small changes (hence *evolutionary*) in the way we do things, constantly improving the performance and the results of what we are doing. Because people can imagine the effect of the change, this evolutionary change can be biased towards improvement, rather than being random.
One of the techniques having emerged out of the Evo way of working is the TimeLine technique, which allows us to get and keep the timing of projects under control while still improving the project results, using just-enough estimation and then calibration to reality. TimeLine doesn't stop at establishing that a project will be late. We actively deal with that knowledge: instead of *accepting* the apparent outcome of a TimeLine exercise, we have ample opportunities of *doing* something about it. One of the most rewarding ways of doing something about it is *saving time*.
An essential prerequisite of getting projects on time is, however, that we *care* about time.

2 The Goal of a project

Many projects treat requirements as sacred: they say: "This is what we have to do!" However, many requirements used in projects are not real Requirements, but rather *wishes* and *assumed* requirements. Let's, as a driver for finding the real Requirements, define the following as an universal Goal for a project:

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 …

What the customer *needs* may be different from what he *asks* for and the time he needs it may be *earlier* or *later* than he asks for. If the customer is not satisfied, he may not *want* to pay for our efforts. If he is not successful, he *cannot* pay. If he is not *more* successful than he already was, why should he pay?

Of course we have to add that what we do in a project is:

… constrained by what the customer can afford, and what we mutually beneficially and satisfactorily can deliver in a reasonable period of time.

What the customer wants, he cannot afford

If we try to satisfy all customer's wishes, we'll probably fail from the beginning. We can do so many nice things, given unlimited time and money. But neither the customer nor we have unlimited time and money. Therefore: The Requirements are what the Stakeholders *require*, but for a project the Requirements are what the project is *planning to satisfy*.

If the Requirements aren't clear *(which they usually aren't)****, any schedule will do***

If the Requirements are unclear or incorrect, we will be spending time on the wrong things, wasting time and money. That's in contradiction to the Goal of the project. And what use is a schedule that plans for doing the wrong things? Using the Goal as a top-level Requirement helps us to focus on what we have to do and *what not*. Understanding better what we *should* and *shouldn't do* is one of the drivers for *doing* less, while *delivering* more. Continuous Requirements Engineering and Management is imperative for optimizing the duration of a project.

In almost all projects the requirements are not really clear. Even if they seem to be, they will change during the project, because *we* learn, *they* learn and the circumstances change. If the requirements aren't really clear and will change anyway, why spend too much time on very detailed estimation of the project based on what we only currently think we have to do? If whatever time needed to do the work cannot be exact, because our understanding of the work is not exact, any ballpark figure will do. "But they want more exact estimations!" Well, if you estimated the work to be between 800 and 1000 days of work, but *they* insist in a more exact number, give them any exact looking figure, like 893 days, or 1093 if you like. If that keeps *them* quiet, you don't have to waste more time on determining a figure that isn't exact anyway. Can I do that? Yes, you can. It saves you time you need for more important things. And we should spend our time only on the most important things, shouldn't we?

3 TimeLine

The standard procedure of defining the time needed to arrive at an expected end date of a project is (figure 1) adding up the time we think we need for all the things we think we have to do and then adding some time for contingency.
Usually, however, the customer needs the result earlier. The date the customer needs a result from the project, we call it the FatalDate, is a Requirement, so it has to be treated *as seriously* as any other Requirement.

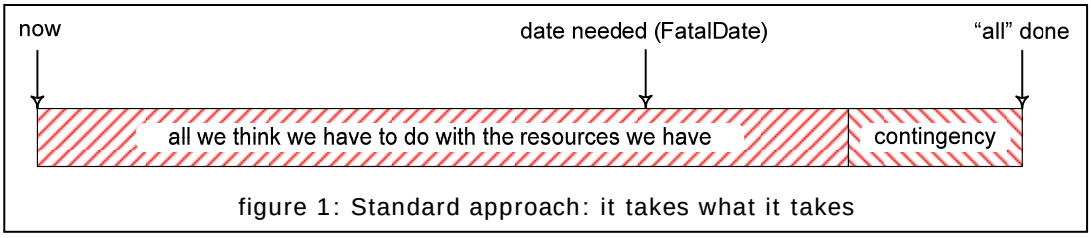


figure 1: Standard approach: it takes what it takes

If we really take the FatalDate seriously, we can define a more refined procedure, called *TimeLine*:

1. Define a deadline or FatalDate. It is better to start with the end: planning beyond the available time/money budget is useless, so we can avoid wasting time if we find out that what we have to do takes more time than we have. Often we count back from the FatalDate to see when we should have started.
2. Write down whatever you *currently think* (it *will* probably change!) you have to accomplish
3. List in order of priority. Note that priority is based on value contribution and hence is influenced by many things!
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 (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 section 6.3)
7. Add up the duration of all elements
8. Divide by the number of available people (still, see sect 4.1 about the danger of doing this!)
9. This is a first estimate of the duration

This technique 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.

If the estimate of the duration is longer than the time available before the FatalDate (figure 2), we will first have to resolve this problem, as it's of no use continuing the work if we know we won't succeed.

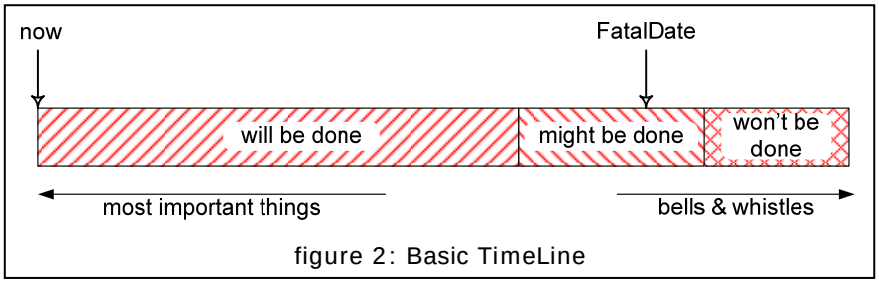


figure 2: Basic TimeLine

We can discuss the TimeLine with our customer and explain:

- What, at the FatalDate, surely will be done
- What surely will not be done
- What might 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 valuable time and money, rather spending it on more rewarding activities. Note that we put and keep putting what we plan in strict order of priority, so that at the FatalDate we'll have done the most important things that could be done in the time available. Customers usually don't really mind about the bells and whistles, as Time to Market is more important. Because priorities may change very dynamically, we have to constantly reconsider the order of what we do and when.

Initially, customers can follow this "will be done - won't be done - might be done" reasoning, but still want "it all". Remember that they don't even exactly know what they really need, so "wanting it all" usually is a fallacy, although we'd better not say that. What we can say is: "OK, we have two options: In a conventional project, at the FatalDay, we would come to you and tell that we didn't make it. In this project, however, we have another option. We already know, so we can tell you *now* that we will not be able to make it and then we can discuss what we are going to do about it. Which option shall we choose?"
If we explain it carefully, the customer will, eventually, choose the latter option. He will grumble a bit the first few weeks. Soon, however, he will forget the whole issue, because what we deliver is what we promised[1]. This enforces trust. Note that many customers ask for more, because they expect to get less. The customer also will become more confident: He is getting Deliveries[2] way before he ever expected it. And he starts recognizing that what he asked was not what he needed, so he's not asking about "all" any more.
At the very first encounter with a new customer we cannot use this method, telling the customer that he will not get "it all". Our competitor will promise to deliver it all (which he won't, assuming that we are not less capable than our competitor), so we lose if we don't tell the same, just as we did before using the Evo approach. There's no risk, because we apparently survived promising "all" in our previous projects. If, after we won the contract, we start working the Evo way, we will soon get the confidence of our customer, and on the next project he will understand and only want to work with us.

If the estimated duration is more than the available time, we first have to resolve this problem, before going into any more detail. If it exactly fits the available time, we'd better assume that we still won't make it, as we probably will have forgotten some elements of work. If the estimated duration easily fits the available time, then there is a good chance that we may succeed (figure 3).

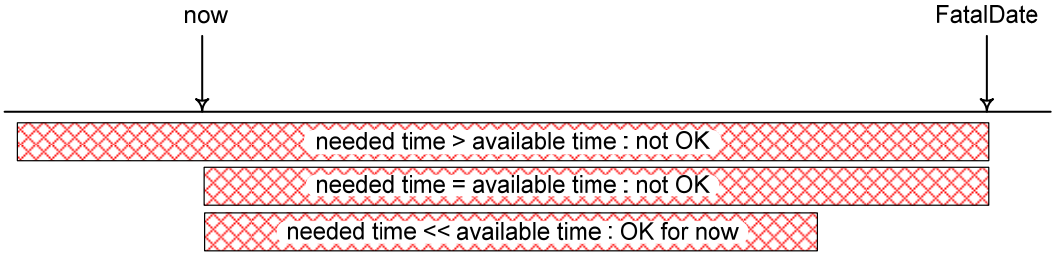


figure 3: Only if the estimated time is well under the available time, we may succeed

3.1 Setting a Horizon

If the total project takes more than 10 weeks, we define a Horizon at about 10 weeks on the TimeLine, because with the limited size of our skull[3] we cannot really oversee longer periods of time. We may set a Horizon once we have done three things to make sure that we'll not be surprised when we start looking over the Horizon again.

[1] We assume that we are using the Evo TaskCycle [14] to organize the work, by which we quickly learn what we can promise and how to live up to our promises.
[2] We also assume that we use Evo DeliveryCycles [14] to check the requirements and assumptions, delivering real results to Stakeholders for feedback.
[3] Thinking of Dijkstra's lecture "The Humble Programmer" [2]: "The competent programmer is fully aware of the strictly limited size of his own skull; therefore he approaches the programming task in full humility, and among other things he avoids clever tricks like the plague." This is just as valid for developers other than programmers.

The TimeLine procedure continues:

10. Choose a Horizon (default: 10 weeks from now)
11. Determine when to look over the Horizon again as shown in figure 4: a (default: halfway)
12. Determine the amount of work proportionally to the total work (figure 4: b)
13. Pull tasks from beyond the Horizon that need earlier preparation not to get surprised later (figure 4: c)

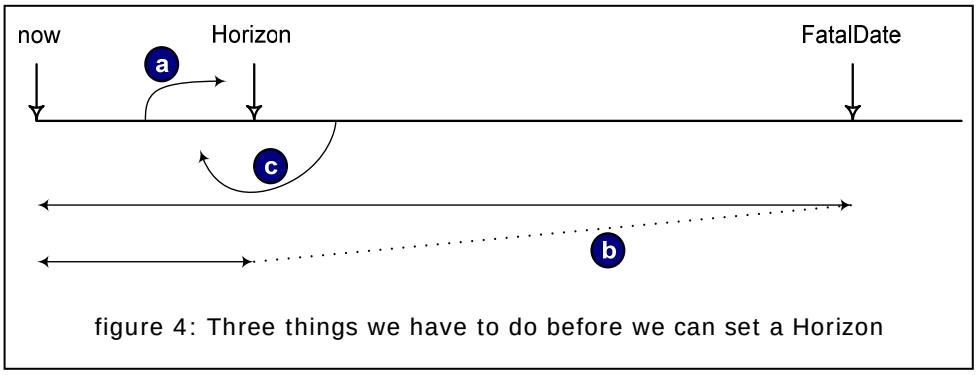


figure 4: Three things we have to do before we can set a Horizon

Now we can, for the time being, "forget" about what's beyond the Horizon and concentrate on a more limited period 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 use a sliding window of 10 weeks, but rather define a 10 weeks period, and try to accomplish our plans within this TimeBox. When we decide the time is right, we move to the next 10 week window.

3.2 DeliveryCycles

Within these 10 weeks, we plan Evo DeliveryCycles [14] (figure 5), each not more than 2 weeks in length, asking: "*What* are we going to deliver to *Whom* and *Why?*" Deliveries are for getting

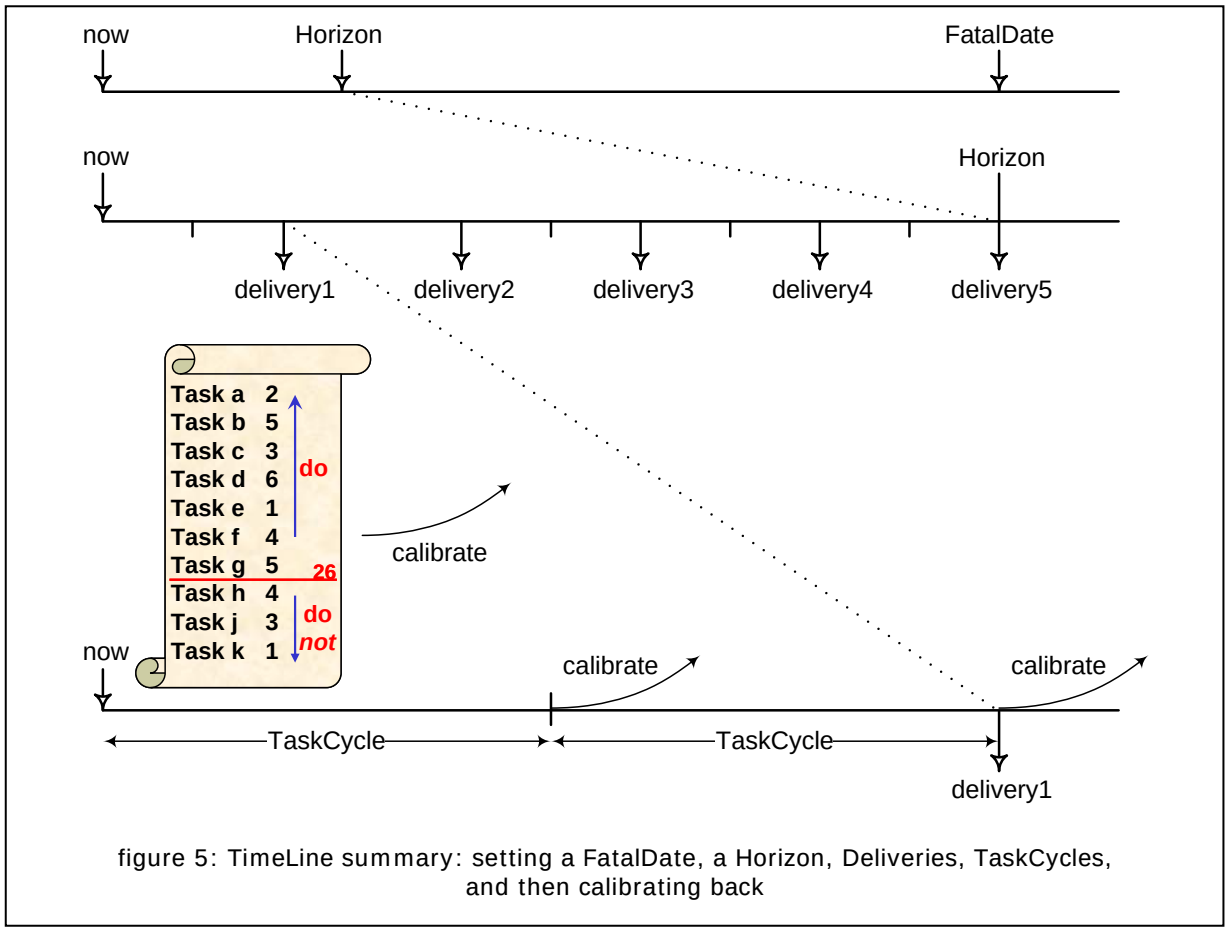


figure 5: TimeLine summary: setting a FatalDate, a Horizon, Deliveries, TaskCycles, and then calibrating back

feedback from appropriate Stakeholders. We are humble enough to admit that our (*and* their) perception of the requirements is probably not perfect and that many of our assumptions may be incorrect. That's why we need communication and feedback and that's why we make many short DeliveryCycles: to find out about the real Requirements, which assumptions are correct, and to waste as little time as possible on incorrect requirements and assumptions, saving precious time. In order to get feedback, we have to deliver to *eagerly waiting* Stakeholders. 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*.

The TimeLine procedure continues:

14. Put the work for 10 weeks in optimum order, defining Deliveries of 2 weeks (or less)
15. Work should be estimated in more detail now
16. Make a rather detailed description of the first one or two Deliveries
17. Check the feasibility of completing Deliveries in two weeks each, with the available resources

We don't only design the product, we are also constantly *(re)designing the project*. Defining Deliveries is about designing the project: in which order should we do things to find out what is really necessary for a successful result and what is *superfluous*. How can we make sure that at any time, looking back, we can say: "We weren't perfect, but we couldn't have done better".

3.3 TaskCycles

Once we have divided the work over Deliveries, which also behave like Horizons, we first 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, every TaskCycle defining a weekly Horizon. In the TaskCycle we define Tasks, estimated in effort-hours (see [14] 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). 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.

If we take the example of figure 6, we can see that if Carl doesn't start Task-f about 6+9+11+9 = 35 hr before the end of the Delivery-Cycle, he's putting the success of the Delivery on the line.

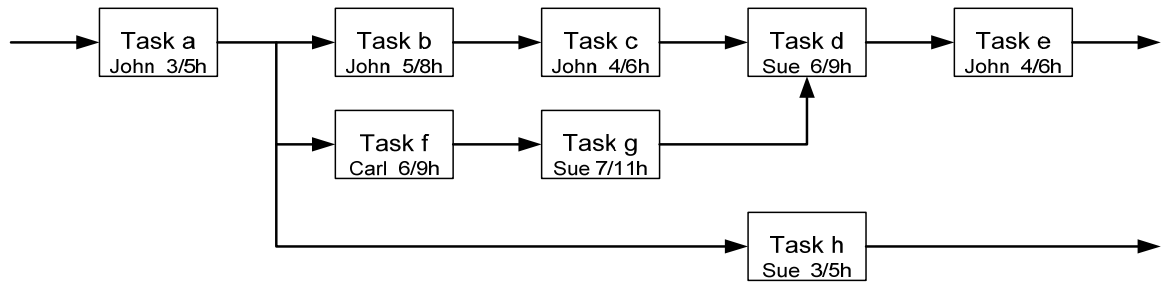


figure 6: Planning serial and parallel Tasks to fit the available time
(based on a diagram emerging, and then being used in an actual project)
Note: "6/9h": 6 is effort hours (planned), 9 is duration (allowing for unplanned activities)

If this Delivery was planned for the coming week, we also see that John should start right at the beginning of the Cycle, otherwise Carl can't start in time. It's easy to imagine that if the work of this Delivery wasn't *designed* this way, the Delivery probably wouldn't be on time. *Designing* the order of work for the Delivery *saves* time.

The TimeLine procedure goes on:

18. Determine Tasks for the first week
19. Estimate the Tasks, now in real effort (net) hours needed to 100% complete each Task
20. Select Tasks to fit the plannable time (default: 2/3 of available time) of the people in the team
21. Select only the most important Tasks, never ever plan nor do less important 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, e.g. as in the example of figure 6.

3.4 Calibration

Having estimated the work that has to be done for the first week, we have captured the first metrics to start *calibrating* the TimeLine. If the Tasks for the first week would deliver only about half of what we need to do in that week, we now can, based on this limited material, extrapolate that our project is going to take twice as long, *if* we don't do something about it. Of course, at the start this seems weak evidence, but it's already an indication that our estimations may be too optimistic. Putting our head in the sand for this evidence is dangerous. One week later, when we have the *real* results of the first week, we have even better numbers to extrapolate and scale how long our project may take. Week after week we will gather more information with which we can calibrate and adjust our notion of what will be done at any FatalDate. 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.

The TimeLine procedure now concludes with two more steps:

25. Calibrate the TimeLine estimations *and take the consequence*
26. Repeat every one or two weeks.

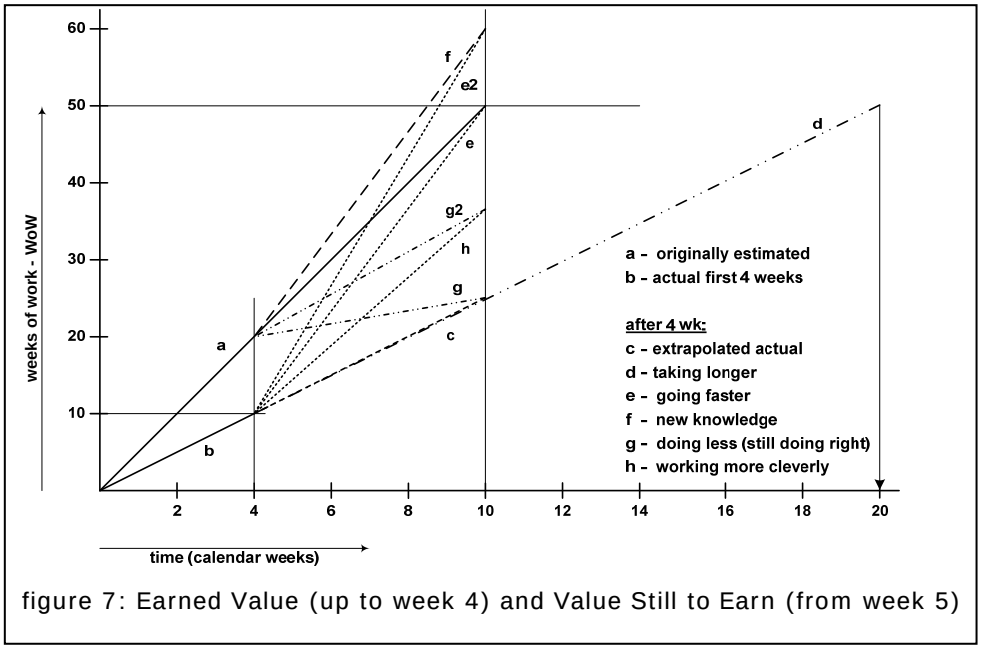


figure 7: Earned Value (up to week 4) and Value Still to Earn (from week 5)

Let's take an 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; figure 7, 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!

We now have to assume that at the end of the 10 weeks we'll have completed $10/4 * 10 = 25$ WoW (line c). This is 50% less than the original 50 WoW expected. 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 [3]) 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).

But there is 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 probably won't produce the desired outcome, and even may work out counterproductive, as explained in section 4.1. 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 only what is not necessary (line g2) and doing things more productively (line h), as explained in section 4.6. Actively and weekly *designing* what exactly to do and in which order saves a lot of time.

4 If things don't fit

If what we think we have to do doesn't fit the available time, or if we want to fit what we think we have to do into a shorter timeframe, there are several options we see being used in practice:

- To be used with utmost care: Adding people
- Deceptive options:
  - Hoping for the best
  - Going for it
  - Working Overtime
  - Adding time: Moving the deadline
- Most interesting to exploit, but mostly overlooked: Saving time
  - Not doing things that later prove to be superfluous
  - Doing things differently
  - Doing things at the right time, in the right order
  - TimeBoxing

4.1 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 8. In his essay *The Mythical Man-Month*, Brooks [4] shows that this is a fallacy, defining Brooks' Law: *Adding people to a late project makes it later*.

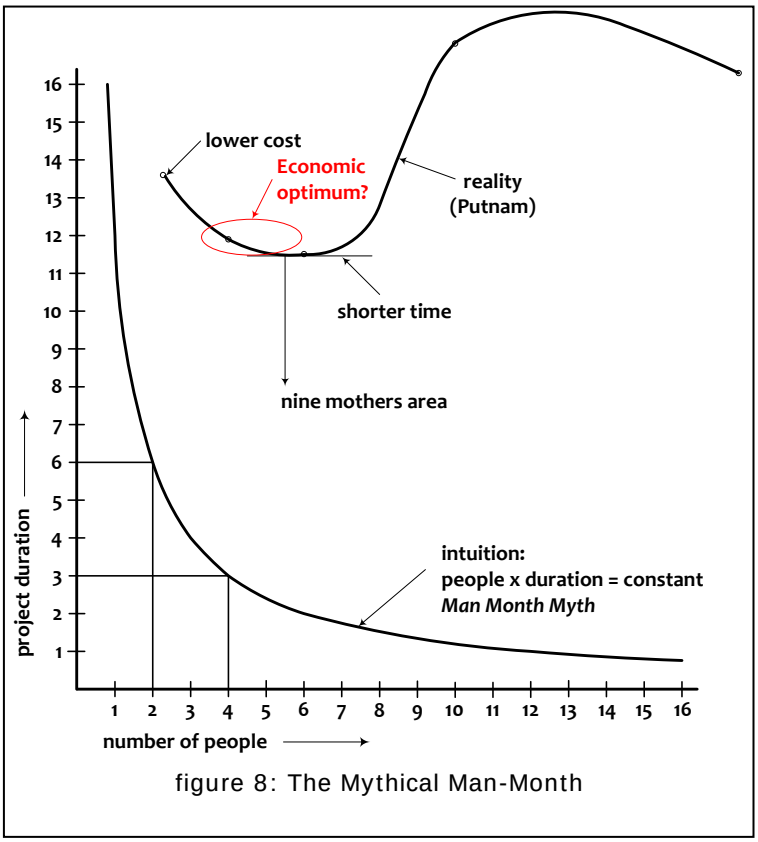


figure 8: The Mythical Man-Month

Putnam [5] confirms this with measurements on some 500 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. 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 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 worse: if in the *first several weeks* of a project we find that the development speed is slower than predicted, 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:

- Apparently, the time needed to complete a development 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 already 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 increases 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 architect who has to prepare and oversee what everybody has to do may become the bottleneck when there are more people to manage
- The productivity of different people can vary vastly, so people cannot simply be exchanged for time

Therefore, adding people is not automatically a solution that works. It can even 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, as found in numerous research, like the Standish reports [6].

The only way to try to circumvent Brooks' Law is to work with many small teams, who can work in parallel, and who only synchronize their results from time to time, for example in bi-weekly DeliveryCycles. And yes, this adds complexity to the design of the project, for which the architect may become a bottleneck.

4.2 Hoping for the best

Most projects take more time than expected. Your past projects 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 a 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 strikes again.

4.3 Going for it

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 is ever the case) and if we work a bit harder (as if we don't already work hard) … Well, forget it.

4.4 Working Overtime

Working overtime is fooling yourself: 40 hours of work per week is already quite exhausting. If you put in more hours, you'll get more tired, make more mistakes, having to spend extra time to find and "fix" these mistakes (half of which you won't), and 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.

4.5 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 may call this Parkinson's Law[4] [7] or the Student Syndrome[5] [8]. At the new deadline we probably hardly have done more, pushing the project result even further. Not a good idea, *unless* we really are in the nine mother's area, 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 is afterwards, in which we can decide what the next best thing there is to do.

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.

4.6 Saving time

Instead of letting things randomly be undone at the FatalDay, it's better to *choose* what we won't have done, preferably those things that weren't needed anyway. We know that we won't have enough time, so let's save time wherever possible!

---

[4] Parkinson's Law: "Work expands so as to fill the time available for its completion".
Observation by Parkinson [7]: "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."

[5] Starting as late as possible, only when the pressure of the FatalDate is really felt. Term attributed to E. Goldratt [8].

There are several ways to save time, *without negatively affecting the Result of the project*:

- **Efficiency in *what* to do**: d*oing only what is needed, not doing things that later prove to be superfluous.*
  This includes efficiency in knowing *why* and *for whom* to do it. Because people tend to do more than necessary, especially if the goals are not clear, there is ample opportunity for *not* doing what is *not* needed.
- **Efficiency in *how* to do it**: *doing things differently*.
  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.
- **Efficiency in *when* to do it**: doing things at the right time, in the right order.
  A lot of time is wasted by synchronization problems, like people waiting for each other, or redoing things because they were done in the wrong order. Actively Synchronizing [15] and *designing* the order of what we do (e.g. as in figure 6), saves a lot of time.

In my experience, these are all huge time savers.
Of course we can also apply these time savers if what we think we have to do easily fits in the available time. We don't have to wait until we're in trouble …

TimeBoxing provides incentives to constantly apply these ways to save time, in order to stay within the TimeBox. 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, and already indicated by Henry Ford in his book "My Life and Work", back in 1922.

5 Preflection, foresight and prevention

Because "hindsight is easy", we can often use it to reflect on what we did, in order to learn: Could we have avoided doing something that now, in hindsight, proves to be superfluous? Could we've done it more efficiently? Reflection, however, doesn't *recover* lost time: the time is already spent and can never be regained. Only with *preflection* we can try to *foresee* and thus *prevent* wasting precious time.

6 Estimation

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

6.1 Changing from *optimistic* to *realistic* estimation

In the Evo TaskCycle [14] 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 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!

6.2 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, it 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.

6.3 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 [10], 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'll never be. 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 (see section 3.4).

6.4 Estimation tools

There are several estimation methods and tools on the market, like e.g. COCOMO [11], QSM-SLIM [12] and Galorath-SEER [13]. 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).
- 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 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.

7 Conclusion

TimeLine doesn't solve our problems. TimeLine is a set of techniques to expose the real status of our project early and repeatedly. Instead of *accepting* the apparent outcome of a TimeLine exercise, we have ample opportunities of *doing* something about it.
We can save a lot of time by not doing the things that later would prove to be superfluous. Because people do a lot of unnecessary things in projects, it's important to identify those things before having started, otherwise the time is already spent, and never can be recovered. By revisiting the TimeLine every one or two weeks, we stay on top of how the project is developing and we can easily report to management the real status of the project.
Doesn't all this TimeLining take a lot of time? The first one or two times it does, because we are not yet acquainted with the various elements of the project and we have to learn how to use the TimeLine techniques. After a few times, however, we dash it off and we're getting into a position that we really can start optimizing the results of the project, producing more than ever before. TimeLine 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 the TimeLine technique for the first time. After having done it once, however, the usual reaction is: "Wow! I got much better oversight over the project and the work than I ever expected", and the hesitation is over.

The TimeLine technique is not mere theory. It is highly pragmatic, and successfully used in many projects coached by the author. The most commonly encountered bottleneck when introducing the TimeLine technique in a project is that no one in the project has an oversight of what exactly the project is really supposed to accomplish. This could even be a reason why Project Managers hesitate to start using the technique. Redefining what the project is to accomplish and henceforth focusing on this goal is the first immediate timesaver of the technique, with many savings to follow.

References

[1] **Gilb**, Tom: *Principles of Software Engineering Management*, 1988, Addison Wesley, ISBN 0201192462.
Gilb, Tom: *Competitive Engineering*, 2005, Elsevier, ISBN 0750665076.
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
Malotaux, Niels: *Controlling Project Risk by Design*, 2006. www.malotaux.nl/nrm/pdf/EvoRisk.pdf

[2] **Dijkstra**, E.W.: *The Humble Programmer*, 1972, ACM Turing Lecture, EWD340. www.cs.utexas.edu/~EWD/ewd03xx/EWD340.PDF

[3] **Cohn**, Mike: *Agile Estimating and Planning*, 2005, Prentice Hall PTR, ISBN 0131479415.

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

[5] **Putnam**, Doug: *Team Size Can Be the Key to a Successful Project*, www.qsm.com/process_01.html

[6] **Standish** Group: *Chaos Report*, 1994, 1996, 1998, 2000, 2002, 2004, 2006. www.standishgroup.com/chaos/intro1.php

[7] **Parkinson**, C. Northcote: *Parkinson's Law*: http://alpha1.montclair.edu/~lebelp/ParkinsonsLaw.pdf

[8] **Goldratt**, E.M.: *Critical Chain*, Gower, ISBN 0566080389

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

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

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

[12] www.qsm.com

[13] www.galorath.com

[14] See [1] (Malotaux, 2004): Chapter 5 and 6

[15] See [1] (Malotaux, 2006): Chapter 10

-o0o-

Niels Malotaux

TimeLine: Getting and Keeping Control over your Project

Many projects deliver late and over budget. The only way to do something about this is changing our way of working, because if we keep doing things as we did, there is no reason to believe that things will magically improve. They won't.
The Evolutionary Project Management (Evo) approach is about continuously introducing small changes (hence evolutionary), constantly improving the performance and the results of what we are doing. Because we can imagine the effect of the change, this evolutionary change can be biased towards improvement, rather than being random.
One of the techniques having emerged out of the Evo way of working is the TimeLine technique, which allows us to get and keep the timing of projects under control while still improving the project results, using just-enough estimation and then calibration to reality. TimeLine doesn't stop at establishing that a project will be late. We actively deal with that knowledge: instead of accepting the apparent outcome of a TimeLine exercise, we have ample opportunities of doing something about it. One of the most rewarding ways of doing something about it is saving time. And if we can save time when a project is late, why not use the same techniques even if the project won't be late, to be done even earlier?

This booklet defines the Goal of a project, which enables us to focus on Result. It then describes the basic TimeLine technique, connecting high-level planning to weekly Task Planning and back. It continues with the options we have for dealing with the outcome, especially when we see that the time available seems insufficient to achieve what we think has to be achieved in the project. Finally, some estimation techniques are explained.

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 .

Find more at: www.malotaux.nl - Evo pages are at: www.malotaux.nl/nrm/Evo
Download booklets:

- Evolutionary Project Management Methods — www.malotaux.nl/nrm/pdf/MxEvo.pdf
- How Quality is Assured by Evolutionary Methods — www.malotaux.nl/nrm/pdf/Booklet2.pdf
- Optimizing the Contribution of Testing to Project Success — www.malotaux.nl/nrm/pdf/EvoTesting.pdf
- Optimizing Quality Assurance for Better Results (same as EvoTesting, but for non-software projects) — www.malotaux.nl/nrm/pdf/EvoQA.pdf
- Controlling Project Risk by Design — www.malotaux.nl/nrm/pdf/EvoRisk.pdf
- TimeLine: Getting and Keeping Control over your Project — www.malotaux.nl/nrm/pdf/TimeLine.pdf

ETA: Evo Task Administration tool — www.malotaux.nl/nrm/Evo/ETAF.htm

N R Malotaux
Consultancy

Niels R. Malotaux
Bongerdlaan 53
3723 VB Bilthoven
The Netherlands
Phone +31-30-228 88 68
Fax +31-30-228 88 69
Mail niels@malotaux.nl
Web www.malotaux.nl
Evoweb www.malotaux.nl/nrm/Evo

Originally prepared for the Annual Pacific Northwest Software Quality Conference, Portland-OR, USA, 2007
Version 1.04 (small edits for new print) - 2 March 2008