TIME BOXING PLANNING: BUFFERED MOSCOW RULES

EDUARDO MIRANDA, INSTITUTE FOR SOFTWARE RESEARCH, CARNEGIE MELLON UNIVERSITY, SEPTEMBER 2011

Keywords: Time boxing planning, prioritization rules, release planning, agile, incremental delivery, requirements prioritization, design to schedule

ABSTRACT

Time boxing is a management technique which prioritizes schedule over deliverables but time boxes which are merely a self, or an outside, imposed target without agreed partial outcomes and justified certainty are at best, an expression of good will on the part of the team. This essay proposes the use of a modified set of Moscow rules which accomplish the objectives of prioritizing deliverables and providing a degree of assurance as a function of the uncertainty of the underlying estimates.

INTRODUCTION

Time boxing is a management technique which prioritizes schedule over deliverables. This means that if during the execution of the task it is anticipated that all requested deliverables will not be ready by a set completion date, the scope of the work will be reduced so that a smaller, yet still useful, output is produced by such date. The two dimensions of the time box are the length of time it is given and the resources available during that time. The time box concept can be applied to individual tasks and single iterations but the focus of this proposal is in larger aggregates, such as a release or a project, culminating in the delivery of an agreed functionality to a customer.

To be effective, time boxing requires that (Miranda, 2002):

1. The features or user requirements[1] that make up the total delivery are grouped into functionally complete subsets;
2. The subsets are prioritized so it is clear which requirements should be implemented first and which ones could be eliminated if there is not enough time to complete all of them; and
3. Reasonable assurance is provided to the customer about the feasibility of a given subset within the imposed frame

Time boxes which are merely a self, or an outside, imposed target without agreed partial outcomes and justified certainty are at best, an expression of good will on the part of the team.

Prioritization is traditionally made by asking the customer to rank his or her preferences into a series of categories such as "Must have", "Should have", "Could have" or "Won't have" where the "Must have" category contains all requirements that must be satisfied in the final delivery for the solution to be considered a success. The "Should have" represents high-priority items that should be included in the solution if possible. The "Could have" corresponds to those requirement which are considered desirable but not necessary. They will be included if there is any time left after developing the previous two. "Won't have" are used to designate requirements that will not be implemented in a given time box, but may be considered for the future. These categories are commonly known by the acronym "Moscow" (Stapleton, 2003). Less used techniques include the pairwise comparisons, cumulative voting, top ten requirements and EVOLVE (Berander & Andrews, 2005).

With the exception of EVOLVE (Greer & Ruhe, 2004) which uses a complex search procedure to maximize value within the constraints imposed by the available resources; all the techniques above suffer from the same problem: they are either unconstrained or arbitrarily constrained. For example in the "top ten" technique the "must have" would be limited to the 10 more important requirements. Why 10? Why not eleven or twelve or nine? This lack of constraints means that in general, as long as the aggregated effort is within the project budget there is no limit to the number of requirements that are assigned to the "must have" category with which the prioritization process ends up not prioritizing anything at all.

In this article we describe a simple requirement prioritization method that: 1) Redefines the MOSCOW categories in terms of the team's ability to complete the requirements assigned to them; and 2) Constrains the number of requirements that the customer can allocate to each category as a function of the uncertainty of the estimates which makes it possible to give the sponsor certain reassurances with regards to their achievability or not. The MOSCOW categories are redefined as follows:

1. Must Have: Those features that the project, short of a calamity, would be able to deliver within the defined time box

[1] This two terms will be used loosely and alternatively to refer to a discrete capability requested by the sponsor of the work

2. Should Have: Those features that have a fair chance of being delivered within the defined time box
3. Could Have: Those features that the project could deliver within the defined time box if everything went extraordinarily well, i.e. if there were no hiccups in the development of requirements assigned to higher priority categories
4. Won't have features, those for which there is not enough budget to develop them

Therefore, the fitting of requirements into these categories is not an a priori decision but rather a consequence of what the development team believes can be accomplished under the specific project context and budget.

In the past I have associated a delivery probability of 90, 45 and 20% with each of the categories, but this quantification it is only possible if one is willing to make assumptions about the independence or covariance of the actual efforts, the number of requirements included in each category and the type of distributions underlying each estimate; or to use a method such as Monte Carlo simulation to expose the distribution of the total effort for each category. If we are not willing to make this, quoting specific numbers is just an analogy, all we can justifiable say is that the likelihood of delivering all requirements in the "must have" category would roughly double the likelihood of those in the "should have" category and quadruple that of those in the "could have" one.

THE IDEA

The process requires that each feature or requirement to be developed has a two points estimate[2]: a *normal completion effort*[3] and a *safe completion* one. The normal completion effort is that, which in the knowledge of the estimator has a fair chance of being enough to develop the estimated feature while the safe estimate is that which will be sufficient to do the work most of the time but in a few really bad cases.

If we wanted to be reassured of being able to deliver all features under most circumstances we would need to plan for the worst case, which means scheduling all deliverables using their safe estimate. This, more likely than not, will exceed the boundaries of the time box[4]. See Figure 1.a.

It is clear that by scheduling features at the safe level, the most work we can accommodate within the time box boundaries is that depicted by the patterned area in Figure 1.b. So for the "must have" category the customer must select, from among all requirements, those which are more important for him until exhausting the number of development hours available while scheduling them at the safe

[2] More sophisticated approaches such as Statistically Planned Incremental Deliveries – SPID (Miranda, 2002) will require three points estimates and the specification of an underlying distribution

[3] As I did in the redefining of the MOSCOW categories in this article I am avoiding the temptation of calling these estimates the 50% and the 90% probability estimates to prevent giving a false sense of mathematical exactness, that will require the making of additional assumptions or an analysis that might not be justified by the practical impact of the added accuracy and precision.

[4] If a single project had to ensure against all possible risks and uncertainty, its price would be prohibitive (Kitchenham & Linkman, 1997)

effort level. This is the constraint missing in other prioritization methods and the key to provide a high level of confidence, in spite of the uncertainty of the estimates and the speed of execution, in the delivery of features in this category.

Once the "must have" requirements have been selected, we will re-schedule them using their normal estimates, see figure 1.c, and reserve the difference between the two effort levels as a buffer to protect their delivery. We will repeat the process for the "should have" and "could have" requirements using the size of the buffer protecting the previous category as the initial budget for the current one, see figure 1.d. The requirements that could not be accommodated in any category at their safe level become the "won't have"s.

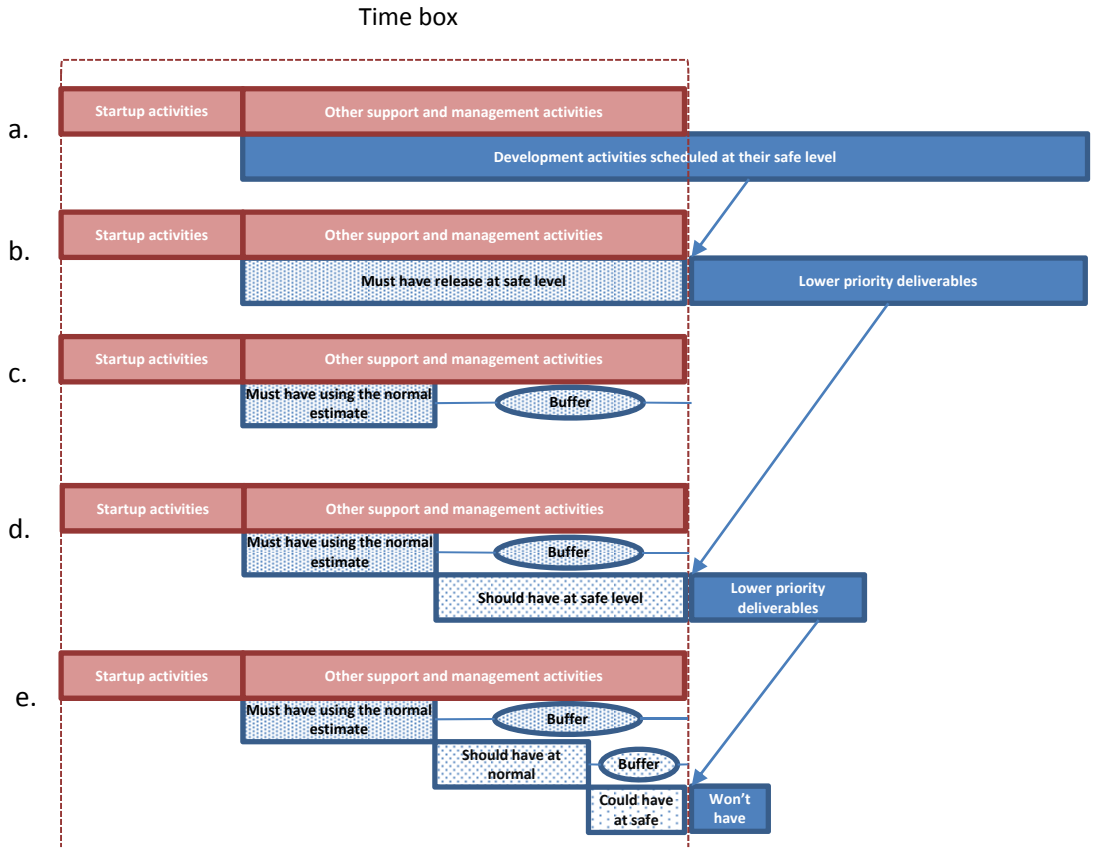

Figure 1 How the method works

We can see now why we said at the beginning of this essay that the "must have" category will have double the likelihood of being completed of the "should have" and quadruple that of the "could have".

We are almost certain that all the requirements in the "must have" category can be completed within the time box because a requirement was only included in it if there was enough room to develop it under a worst case assumption. The "should have" category also have their requirements scheduled at the safe level, but with respect to the overall time box this level of confidence is contingent on the sum

of the actual efforts spent on all the requirements in the "must have" subset being equal or less than the sum of their normal development times. This roughly halves the likelihood of completing all "should have" requirements within the time box. Similarly the likelihood of completing all the "could have" would be half of that of delivering all the "should have" or a quarter of the "must have".

A NUMERICAL EXAMPLE

Table 1 shows the backlog for an imaginary project with a total budget (time box) of 180hrs. Assuming that the startup, and the support and management activities require 60hrs. leave us with a development budget of 120 hrs. The table lists the name of the features, the normal and the safe estimates and the name of other requirements or features in which the current one depends on. For example feature "H" will have a normal estimate of 10 hours, a safe estimate of 20 hours and depends on "J" and "K", meaning that these two features must be present for "H" to provide any business value.

Table 1 Product backlog

Features	Normal Estimate	Safe Estimate	Dependencies
A	20	40	B, C
B	7	9	
C	20	30	
D	5	7	E
E	6	7	
F	5	6	
G	20	40	
H	10	20	J, K
I	15	30	
J	12	15	
K	8	10	
L	10	18	

Let's assume that from a pure business perspective the preferences of the project sponsor are: F, D, A, G, K, E, L, J, H, I, B, C. In a real project this choices will be made during the prioritization meeting.

In our example, the first requirement to be selected for the "must have" category would be requirement "F", applying the process described below we have:

$$AvailableBudget_{i+1} = AvailableBudget_i - SafeEstimate_i = 120hrs - 6hrs = 114hrs$$

Successive requirements are selected as per table 2. Notice that feature "G" cannot be included in the "must have" subset at the safe level because it does not fit into the available budget. At this point the customer must decide whether to resign "G" to another category, if possible, or rearrange the previous selection. For the sake of the example let's assume requirement "G" is passed on, and the customer chooses "K" which follows in his rank of preferences and is schedulable in the available budget.

Table 2 Assigning requirements to the "must have" category

Features	Reason for selection	$AvailableBudget_i$	$SafeEstimate_i$	$AvailableBudget_{i+1}$
F	Customer preference	120	6	114
D, E	Customer preference, Dependency	114	14	100
A, B, C	Customer preference, Dependency	100	79	21
~~G~~	~~Customer preference~~	~~21~~	~~40~~	~~19~~
K	Customer preference	21	10	11

After including "K" there is no other requirement that can be included in the "must have" category, so requirements F, D, E, A, B, C, and K are re-schedule at their normal level:

$$MustHaveBudget = \sum_{i\in\{F,D,E,A,B,C,K\}} NormalEstimate_i = 5 + 5 + 6 + 20 + 7 + 20 + 8 = 71hrs$$

$$MustHaveBuffer = AvailableBudget - MustHaveBudget = 120 - 71 = 49hrs$$

The process is now repeated using the $MustHaveBuffer$ as the available budget for the "should have" CATEGORY, see table 3, and the $ShouldHaveBuffer$ for the "could have". See table 4.

Table 3 Assigning requirements to the "should have" category

Features	Reason for selection	$AvailableBudget_i$	$SafeEstimate_i$	$AvailableBudget_{i+1}$
G	Customer preference	49	40	9

Table 4 Assigning requirements to the "could have" category

Features	Reason for selection	$AvailableBudget_i$	$SafeEstimate_i$	$AvailableBudget_{i+1}$
L	Customer preference	29	18	11

After including "L" nothing more could be included in the available effort at the safe estimate level and in consequence "H", "I" and "J" are declared "won't have".

The final subsets are:

- Must have: F, D, E, A, B, C, K
- Should have: G
- Could have: L
- Won't have: H, I, J

EXECUTION

Figure 2 shows the initial plan resulting from the prioritization process. Now imagine that during the execution of the project feature "A" takes 40hrs, its worst case, instead of the 20 allocated to it in the plan. This will push the development of features "G" and "L" to the right. This would leave us with 29hrs to develop "G", 9 more than the 20hrs estimated at 50%, so one can say there still is a fair chance the customer will get it. If "G" takes 20 hours the budget remaining in the box will be 9 hours, one less than the 10 estimated at 50%, so in this case the chance of the customer getting L would be slim. See Figure 3.

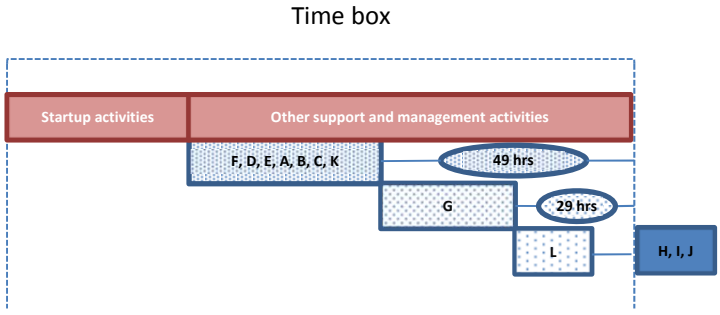

Figure 2 Original plan. Time box = 180 hrs, Startup and other support and management activities 60 hrs, development budget 120 hrs

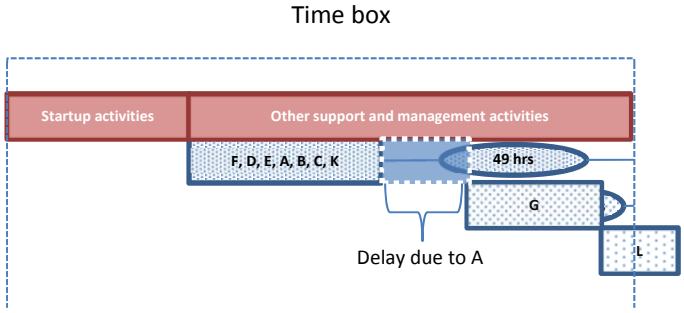

Figure 3 Must have release is delayed because "A" takes longer than the scheduled budget

DEALING WITH CHANGES AND DEFECTS

Changes are natural. When a change occurs it should be ranked against current priorities and if accepted it will be at the expense of an already planned requirement or by changing the time box itself.

With respect to defects a sensible strategy is to fix all critical and major defects within the time allocated at the subset in which they are discovered, postponing minor defects to the end of the project and giving the customer the choice between fixing the problems and developing additional functionality.

BUSINESS IMPLICATIONS

It is obvious that acknowledging from the very start of the project that the customer might not receive everything requested requires a very different communication, and perhaps marketing, strategy from that of a project that promises to do it, even when nobody believes it will do it.

The premise, in which the method is based, is that businesses are better off when they know what could realistically be expected than when they are promised the moon, but no assurances are given with respect as to when they could get it.

To be workable for both parties, the developer and the sponsor, a contract must incorporate the notion that an agreed partial delivery is an acceptable, although not preferred, outcome. A contract that offloads all risk in one of the parties would either be prohibitive or unacceptable to the other. The concept of agreed partial deliveries could adopt many forms. For example the contract could establish a basic price for the “must have” set with increasingly higher premiums for the “should have” and “could have” releases. Conversely the contract could propose a price for all deliverables and include penalties or discounts if the lower priority releases are not delivered. The advantage for the project sponsor is that, whatever happens, he can rest assured that he will get a working product with an agreed subset of the total functionality by the end of the project on which he can base his own plans.

A similar idea could be applied to any reward for the people working in the project. No reward will be associated with delivering the “must have” release since the team members are simply doing their jobs. Subsequent releases will result in increased recognition of the extra effort put into the task. The relative delivery likelihood associated with each release could be used to calculate the reward’s expected value.

SUMMARY

We have presented a simple prioritization procedure that can be applied to the ranking of requirements at the release as well as the project level.

The procedure does not only captures customer preferences, but by constraining the number of features in the “must have” set as a function of the uncertainty of the underlying estimates, is able to offer project sponsors a high degree of reassurance in regards to the delivered of an agreed level of software functionality by the end of the time box.

This simplicity is not free. It comes at the expense of the claims we can make about the likelihood of delivering a given functionality and a conservative buffer. Users seeking to make more definitive statements than "short of a calamity" or optimize the buffer size should consider the use of a more sophisticated approach such like the one described in Planning and Executing Time Bound Projects (Miranda, 2002) which requires considerably more information and an understanding of the problems associated with the elicitation of probabilities.

BIBLIOGRAPHY

Berander, P., & Andrews, A. (2005). Requirements prioritization. In C. W. A. Aurum, *Engineering and Managing Software Requirements.* Berlin: Springer Verlag.

Greer, D., & Ruhe, G. (2004). Software release planning: an evolutionary and iterative approach. *Information and Software Technology, 46*(4).

Kitchenham, B., & Linkman, S. (1997, May). Estimates, Uncertainty and Risk. *IEEE Software*.

Miranda, E. (2002, March). Planning and Executing Time Bound Projects. *IEEE Computer*.

Stapleton, J. (2003). *DSDM Business Focused Development, 2nd ed.* London: Addison Wesley.