

Timeboxing

A project has the triple constraints of time, scope and cost (there are other parameters of course). If one parameter changes, something else must change to keep the three as a constant. Timeboxing is normally used to complete a defined amount of work in a fixed period.

Timeboxing makes time a fixed component and adjust the other two constraints (scope and cost) to fit. This can lead to a number of interpretations:

- We will complete as much as we can in the period available.
- We will apply as many resources as we need to complete the work in the period available.
- Or something in between.

Timeboxing is a very simplistic approach to management but quite common in software development. Some of the problems include:

- The lack of any correlation between resource numbers and productivity. Increasing resources will definitely cost more but there is unlikely to be a commensurate increase in productivity¹. This problem was elegantly defined by Fredrick P. Brooks in *The Mythical Man-Month*², summarised by the saying adding manpower to a late software project makes it later!
- De-scoping normally reduces value far quicker than any other element³. Timeboxing can quickly lead to a software release that has cost as much as planned and taken the time planned but delivered almost no value.

Just because time is easy to measure, it does not mean time is the most important component of a project. To be effective, time boxing requires that:

1. The features or user requirements⁴ that make up the total delivery are grouped into functionally complete subsets;
2. The subsets are prioritised 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 estimations of what's practical to achieve are of little value or use.

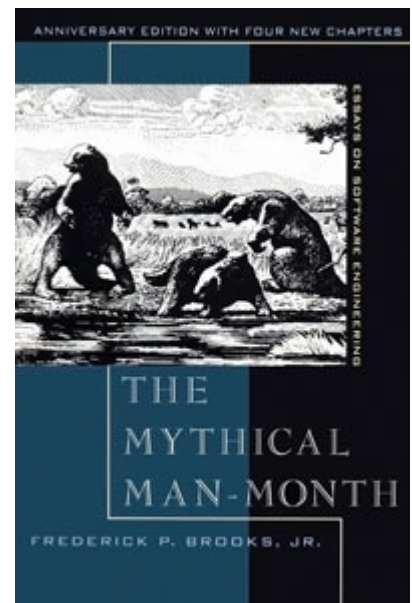
Moscow is one method of prioritisation which requires the customer to rank his or her preferences into a series of categories such as 'Must have', 'Should have', 'Could have' or 'Won't have'. These categories are commonly known by the acronym 'Moscow'.

¹ For more on estimating task durations see: http://www.mosaicprojects.com.au/WhitePapers/WP1052_Time_Estimating.pdf

² First published 1975 - *The Mythical Man-Month: Essays on Software Engineering*, Frederick P. Brooks (ISBN 0-201-00650-2) (Cover from the 20th Anniversary edition – 1995)

³ For more on benefits and value see: http://www.mosaicprojects.com.au/WhitePapers/WP1023_Benefits_and_Value.pdf

⁴ For more on requirements see: http://www.mosaicprojects.com.au/WhitePapers/WP1071_Requirements.pdf



The definition of each category, aligned to sensible framing of a ‘time box’, are:

- **Must have** contains all requirements that must be satisfied in the final delivery for the solution to be considered a success. Short of a disaster, these features should be able to be delivered within the defined time box. Based on ‘safe’ estimates of the time and effort involved.
- **Should have** represents high-priority items that should be included in the solution if possible. These features should have a fair chance of being delivered within the defined time box if normal circumstances prevail.
- **Could have** are those requirements which are considered desirable but not necessary. They will be included if there is any time left after developing the previous two categories (ie, the work goes reasonably well).
- **Won’t have** is used to designate requirements that will not be implemented in a given time box, but may be considered for the future.

Fitting of requirements into these categories is a consequence of what the development team believes can be accomplished under the specific project context and budget⁵. As a consequence, the client can be almost certain that all the requirements in the “must have” category will 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. In all probability, a good proportion of the ‘should haves’ will be delivered as well with a possibility of a few ‘could haves’.

For more information on scheduling and planning, visit Mosaic’s planning and scheduling home page at:
<http://www.mosaicprojects.com.au/Planning.html>

⁵ For more sophisticated ranking options see:
http://www.mosaicprojects.com.au/WhitePapers/WP1062_Ranking-Requirements.pdf