The “Could have” corresponds to those requirements 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” [2]. Less used techniques include the pairwise comparisons, cumulative voting, top ten requirements and EVOLVE [3].

With the exception of EVOLVE [4] 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
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 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.
2. **THE IDEA**

The process requires that each feature or requirement to be developed has a two points estimate: a *normal completion effort* 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. 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 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”.

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”.

---

2 More sophisticated approaches such as Statistically Planned Incremental Deliveries – SPID [1] 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, which 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 [5]
3. 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>
<thead>
<tr>
<th>Features</th>
<th>Normal Estimate</th>
<th>Safe Estimate</th>
<th>Dependencies</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>20</td>
<td>40</td>
<td>B, C</td>
</tr>
<tr>
<td>B</td>
<td>7</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>20</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>5</td>
<td>7</td>
<td>E</td>
</tr>
<tr>
<td>E</td>
<td>6</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>5</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>G</td>
<td>20</td>
<td>40</td>
<td></td>
</tr>
<tr>
<td>H</td>
<td>10</td>
<td>20</td>
<td>J, K</td>
</tr>
<tr>
<td>I</td>
<td>15</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td>J</td>
<td>12</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>K</td>
<td>8</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>L</td>
<td>10</td>
<td>18</td>
<td></td>
</tr>
</tbody>
</table>

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:

\[ \text{AvailableBudget}_{i+1} = \text{AvailableBudget}_i - \text{SafeEstimate}_i \]
\[ = 120 \text{hrs} - 6 \text{hrs} = 114 \text{hrs} \]

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.

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:

\[ \text{MustHaveBudget} = \sum_{i \in \{F, D, E, A, B, C, K\}} \text{NormalEstimate}_i \]
\[ = 5 + 5 + 6 + 20 + 7 + 20 + 8 = 71 \text{hrs} \]

\[ \text{MustHaveBuffer} = \text{AvailableBudget} - \text{MustHaveBudget} \]
\[ = 120 - 71 = 49 \text{hrs} \]

The process is now repeated using the \text{MustHaveBuffer} as the available budget for the “should have” CATEGORY, see table 3, and the \text{ShouldHaveBuffer} for the “could have”. See table 4.

### Table 3 Assigning requirements to the “should have” category

<table>
<thead>
<tr>
<th>Features</th>
<th>Reason for selection</th>
<th>Available Budget$_i$</th>
<th>Safe Estimate$_i$</th>
<th>Available Budget$_{i+1}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>G</td>
<td>Customer preference</td>
<td>49</td>
<td>40</td>
<td>9</td>
</tr>
</tbody>
</table>

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

4. 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.

5. 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.
6. 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.

7. 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 [1]
which requires considerably more information and an understanding of the problems associated with the elicitation of probabilities.

8. REFERENCES


