Release Planning

Akhil Kumar Daida
9408271238
akda15@student.bth.se

MoSCoW method*:* The MoSCoW approach to requirements prioritization originated from the DSDM methodology (Dynamic Software Development Method), which was possibly the first agile methodology – even before we knew iterative development as 'agile'. MoSCoW is a fairly simple way to sort features (or user stories) into priority order – a way to help teams quickly understand the customer's view of what is essential for launch and what is not.

From [1], In MoSCoW method the team divides the user stories into 4 parts. After that task assigning according to priority manner will be a challenging task.

MoSCoW stands for

M- Must have this

S- Should have this if at all possible

C- Could have this if it does not affect anything else

W- Won't have this time but will have it in future

From [2], 'Must Haves' are features that must be included before the product can be launched. It is good to have clarity on this before a project begins, as this is the minimum scope for the product to be useful.
'Should Haves' are features that are not critical to launch, but are considered to be important and of a high value to the user.
'Could Haves' are features that are nice to have and could potentially be included without incurring too much effort or cost. These will be the first features to be removed from scope if the project's timescales are later at risk.
'Won't Haves' are features that have been requested but are explicitly excluded from scope for the planned duration, and may be included in a future phase of development.
It's a good idea to make sure a project has a healthy number of Should Have and Could Have requirements. This helps to provide the project with some flexibility if things start taking longer than expected, effectively providing the project with some contingency.
If a project only has 'Must Haves', the scope cannot be varied. Therefore the cost and timescales should not really be fixed without including a reasonable contingency. 'Could Haves' can be that contingency. They are effectively stretch tasks; features that will be included if possible, but the launch date will not be moved to accommodate them if there is not enough time to complete them.

Pros of MoSCoW prioritization technique:

1. Quality delivery- Project quality may be characterized as functionality, reliability, usability, efficiency, maintainability and portability. Using MoSCoW prioritization method system can be developed according to the targeted system quality.
2. Technical ability- Difficult tasks can be identified and assigned according to expertise of the developer to avoid mistakes due to lack of expertise.
3. Risk minimization- User stories with high risks and high business priority are implemented at early stages of project so that changes in requirement can be caught in early iteration and product can be thoroughly tested in various round of testing.

Cons of MoSCoW prioritization technique:

1. MoSCoW is ambiguous as to timing, particularly when it comes to the "Won't" rating. "Won't" could mean either "not in the next release" or "not ever".
2. MoSCoW doesn't offer any rationale for making the decision about how to rate the priority of a given requirement compared to others.
3. There is a necessity to clear up any distinctions present and to maintain common understanding among all stakeholders on priority order.

Reflections:

We have prioritized 119 issues of the course management system using MoSCoW prioritization technique. These 119 issues were divided into 4 parts they are: must have, should have, could have, won’t have. After dividing into parts whole team have assigned themselves some issues to work on. From my experience, I think MoSCoW prioritization technique is time consuming when dividing the issues into 4 parts which might result progress delay on things which are essential. Besides from MoSCoW prioritization I would like to implement other prioritization techniques considering same 119 issues.

The result of implementation of the prioritization technique can be found below.

References:

[1] R. Popli, N. Chauhan, and H. Sharma, “Prioritising user stories in agile environment,” in 2014 International Conference on Issues and Challenges in Intelligent Computing Techniques (ICICT), 2014, pp. 515–519.
[2] K. Waters, “Prioritization using MoSCoW.” 2009.

Version 0.1

Goal: Early version, get the bare minimums up and running so that teachers can at least keep students informed about a course.

List: 27,62,14,12,36,21,5,40,41,53,18,56,37,54

Dependencies: 14->56->54, 18,27,37,21,62,40,12,36->53,5->41.

S. No	Must	Should	Could	Wont
1	18	56	54	
2	27	62	5	
3	40		41	
4	14			
5	37			
6	12			
7	21			
8	53			

Version 0.5

Goal: The most important features are available, but perhaps not with full functionality.

29 is ignored here.

S. No	Must	Should	Could	Wont
1	16	32	114	33
2	15	64	103	
3	28	55	109	
4	22	50	44	
5	20	30	91	
6	66	58	26	
7	23	43	101	
8	69	45		
9	107	67		
10	77	52		
		59		
		19		
		6		
		94		

Version 0.8

Goal: Full functionality for the most important features. All features at least partially implemented.

S. No	Must	Should	Could	Wont
1	75	33	81	17
2	1	35	79	
3	21	34	78	
4	3	73	2	
5	63	61	97	
6	11	92	86	
7	31	88	84	
8	82	25	85	
9	74	51	100	
10	4	101	110	
11	87	65	83	
12		96	102	
13		38		
14		80		

Version 0.9

Goal: All functionality implemented.

S. No	Must	Should	Could	Wont
1	29	112	106	
2	90	104		
3	111	105		
4	57	113		
5	47	46		
6	48			
7	49			
8				

Version 1.0

Goal: Bugs addressed, most important feedback from earlier releases taken into account.

S. No	**Must**	**Should**	**Could**	**Wont**
1	70			
2	71			
3	72			

s