Modeling Tough Scheduling Problems with Project Management Software

Alex S. Brown, PMP
Strategic Planning Manager
Mitsui Sumitomo Insurance Group, USA

Handout – Scenarios and Exercises

Sample scenarios help to illustrate, teach, and discuss abstract scheduling concepts. This appendix contains concrete examples to help managers learn and apply these principles. By experimenting on a small schedule, a manager can learn the pros and cons of different techniques.

In these examples, "duration" is the minimum possible duration for a task in days. "Work" is the number of work hours estimated to complete the task. Milestones have the abbreviation "(MS)" in their task name. Milestones have zero work and zero duration.

Basic Scenario

A manager has six tasks, each of which can be done simultaneously; there are no hard dependencies. The manager has three people available to perform the work. The task list appears as follows:

Task	Duration (days)	Work (hours)	Resource	Predecessor
1. Project Start (MS)	0	0		
2. Construct #1	5	20	A	1
3. Construct #2	5	20	A	1
4. Construct #3	2	16	B	1
5. Construct #4	2	16	B	1
6. Construct #5	5	16	C	1
7. Construct #6	2	16	C	1
8. Project End (MS)	0	0		2, 3, 4, 5, 6, 7

To start, assume all three resources are available 40 hours per week, and a week is five workdays.

Resource Leveling

Most software will show dates that are too optimistic after entering the raw data above. The use of project resources (A, B, and C) must be leveled, so they are used no more than 40 hours per week. Scheduling options include:

- Keep the dependency relationships as defined above
- Manually "fix" start dates for each task to make the resources level over time
- Allow all tasks to start on day 1, but change the allocation of each resource to create a level use of resources – in other words, adjust the time-per-day that each resource spends on each task until the end dates all match
- Adjust the time-phased work assignments for each task until the resource usage for each person is level (procedure will vary with each software package)
- Introduce "soft" dependencies
- Keep the work for "A" as is, since the two five-day tasks add up to 40 hours already
- Make "Construct #3" a predecessor for "Construct #4", so that "B" has a level set of work – 32 hours over 4 days
- Review the work for "C" and find a way to reschedule it, possibilities include
- Split "Construct #5" into two tasks – one with 8 hours work for one day, and a second with 8 hours of work for one day; put a mandatory delay of 4 days between them to keep the overall duration of five days, and schedule "Construct #6" between these two, new, one-day tasks
- Spread the work for "Construct #6" over four or five days, so it becomes a level set of work that can be done simultaneously with "Construct #5"

Enter this scenario into your scheduling software a few different ways:

- Which method gets an accurate, level schedule the most quickly?
- Which method lets the manager set the start and end dates most flexibly?
- Which way do you typically use?
- Have you discovered any new techniques that might work better than your usual approach?

Representing Hard and Soft Dependencies

In the scenario above, use dependencies to force the schedule to be resource-leveled. The task list becomes:

Task	Duration (days)	Work (hours)	Resource	Predecessor
1. Project Start (MS)	0	0		
2. Construct #1	5	20	A	1
3. Construct #2	5	20	A	1
4. Construct #3	2	16	B	1
5. Construct #4	2	16	B	4
6. Construct #5a	1	8	C	1
7. Construct #6	2	16	C	6
8. Construct #5b	1	8	C	6+3days, 7
9. Project End (MS)	0	0		2, 3, 4, 5, 7,8

In your scheduling software, consider the following questions:

- The hard dependencies for the schedule are no longer documented in the schedule clearly. Where will you document them now?
- According to your scheduling software, what is the critical path? Do you agree with the software?
- Assume that "Construct #4" becomes a four-day task with 32 hours of work. The critical path will then run through "Construct #3" and "Construct #4". Many people would say that these soft dependencies could be broken, so the critical path should not change. Do you agree?

Only Full-Time Work

Advocates of critical chain* methods call for assigning resources to one and only one, full-time job assignment wherever possible. Some project managers report better control over assignments, better worker efficiency, and more accurate reporting when resources work on one and only one deliverable at a time. Make the original task list compliant with these scheduling standards. The task list becomes:

Task	Duration (days)	Work (hours)	Resource	Predecessor
1. Project Start (MS)	0	0		
2. Construct #1a	1	8	A	1
3. Construct #1b	1.5	12	A	2+2.5 days
4. Construct #2a	2	16	A	1
5. Construct #2b	0.5	4	A	4+2.5 days
6. Construct #3	2	16	B	1
7. Construct #4	2	16	B	1
8. Construct #5a	1	8	C	1
9. Construct #6	2	16	C	1
10. Construct #5b	1	8	C	8+3 days
11. Project End (MS)	0	0		3,5,6,7,9,10

Transforming "Construct #1" and "Construct #2" to full-time tasks adds more information to the schedule.

- "A" now needs at least 6 days to complete his or her assignments. Why?
- Is this representation of the schedule more or less "accurate" than the original?
- This schedule requires three more tasks than the original, adding complexity. Is it worth the extra complexity? Which approach is better for your typical project?
- What methods would you prefer to use to level the work for "A" in this scenario? Are the different than your answers to the exercises above?
- Is it easier or more difficult to level, when all work is full-time work?

** Note that true critical-chain scheduling requires buffers and other techniques that are outside the scope of this exercise.*

Managing Quickly-Changing Work Assignments

In the exercises above, you created several versions of the same schedule, each using different techniques and features of your scheduling software. Apply the following changes to each schedule:

- Resource "A" is only available for 20-hours per week, and you cannot change which tasks are assigned to "A". Forecast the new end-date.
- You now discover that you can substitute any resource for any other resource, but it will add an extra, eight hours to the work for any task that you switch. Optimize the schedule, with "A" at a 20-hour workweek.
- Assume that "B" volunteers to work 60 hours per week to get this project done. Optimize the schedule, following the two rules above.

After applying the changes to several schedules, answer some questions:

- Which schedule was easiest to change? Was it the easiest one to build initially or one of the others?
- While applying these changes, did your scheduling software unexpectedly shift dates? When? Why?

Making Schedules Easy-To-Maintain During Project Execution

Week one of the project is now complete. Team status reports show:

Task	Resource	Status	Actual Work	Remaining Duration (d)	Remaining Work (h)
1. Project Start (MS)					
2. Construct #1	A	Completed	16	0	0
3. Construct #2	A	Started	8	2	16
4. Construct #3	B	Not Started	0	2	16
5. Construct #4	B	Started	8	1	8
6. Construct #5	C	Started	8	3	8
7. Construct #6	C	Not Started	0	2	16
8. Project End (MS)					

Apply these actuals to different versions of the schedules, as created in the earlier exercises. Review the new end-dates and the resource utilization for each resource for the remaining work. Answer some questions:

- Was this a new experience, applying actuals and creating an updated forecast? If so, what new techniques did you learn?
- After applying actuals to different versions of the same schedule, how many different end-dates resulted? Which ones were the most accurate? Which ones were the easiest to change and to make realistic?
- While applying changes, did your scheduling software unexpectedly shift some dates around? When? Why?
- Resource "B" started "Construct #4" before "Construct #3". How did your scheduling software respond to that?
- How does your software reschedule work that is started but not completed? Not started?

Summary

After running these exercises, review

- Which techniques did you usually use before? What are their strengths and weaknesses?
- Which of these tasks do you usually perform weekly? Monthly? Daily?
- Which scheduling techniques make schedule entry easiest? Which make maintenance easiest? Which strike a balance?
- Which techniques do you now plan to use in the future?
- What did you learn about your scheduling software that you did not understand before?

Future Development

These examples illustrate a schedule with parallel opportunities. Develop a sample schedule with serial dependencies, and examine the behavior of scheduling software when:

- Tasks are completed in the expected order
- Tasks are completed out-of-order
- Tasks with many dependencies end late
- Tasks with many dependencies end early
- Tasks use dependency relationships other than "finish-to-start", including "start-to-start", "finish-to-finish", and "start-to-finish" (not all software supports all these relationships, and some software has flaws in their support for unusual relationships)

It would be useful for teaching purposes or for software evaluation to have a common set of scheduling exercises, shared across the project management community. Reviewers, researchers, and practitioners could draw upon this dictionary of exercises when teaching scheduling or when evaluating scheduling software. After developing a critical mass of examples, the group could develop a standard set of solutions to each problem. This standard could then help scheduling software developers test their own software against a standard benchmark of results.

Part of a presentation prepared for the 2005 PMI New Jersey Symposium and the Second PMI College of Scheduling Conference, both in May 2005. For more information, contact Alex S. Brown through his web site: http://www.alexsbrown.com/