Project management with Taskjuggler

THE ART OF JUGGLING

Taskjuggler is a handy project management tool for large or small projects.

We'll show you how to organize a simple remodeling job with Taskjuggler.

BY UWE IRMER

If your memory is less than perfect, you can easily lose track of complex tasks. A written to-do list is an important first step, but the right software tools can really help you plan your projects. This article guides you through a simple remodeling job to demonstrate the use of the Taskjuggler [1] project management tool.

Just imagine you finally decide to get serious about refitting your bathroom – a project you have been postponing for years. As the manager of this project, you will need to know which tasks have to happen when, what kind of help you will need, and most importantly, when you will finally be able to take a bath in your new tub. Our project management software is Taskjuggler [1] by Chris Schläger and Klaas Freitag. Taskjuggler is an open source tool released under the GNU GPL license.

The Plan

Gantt plans (named after their inventor, the US engineer Henry Laurence Gantt – 1861 to 1919) became the de facto standard in project management many years ago. The plan visualizes tasks, with defined starting and finishing points as time bars, and additionally indicates the dependencies for the tasks. A Gantt plan highlights the so-called critical path, that is, the tasks that decide whether your project will be completed on schedule or not.

The Gantt plan gives project managers a graphical overview, allowing them to keep track of the project's progress, its current status, and possible deviations from the schedule. The overview clearly shows the dependencies between individual tasks. This makes it easier to parallelize individual processes, that is, to allow them to take place at the same time (start-start conditions.) In a similar fashion, it is also possible to plan a process to coincide with the completion of a dependent process (end-end condition.)

Parallelization has the advantage of reducing the total time required for the project, assuming that multiple processes can take place at the same time. But it also helps you distribute resources. The resources required for the bathroom refit include human labor, but also machines and possibly space. If you succeed in assigning resources to the individual tasks at the planning stage, it is quite easy to see what kind of help you will need at what stage of the project. The Gantt plan refers to important, intermediate points en route to completing the project as milestones. Milestones mark the points in time when specific tasks have to be completed.

To get started, you need to make a list of the tasks involved in the bathroom

Rough Tasks

Bathroom refitting tasks:

1. Prepare replacement fittings
2. Remove old fittings
3. Install new fittings
4. Commission new fittings and end of project

refit project. The box titled "Rough Tasks" provides a first look at the task list.

You can then go on to specify the details for each task, subdividing them into separate subtasks, as shown in the "Detailed Tasks" box.

At this point, we are consciously avoiding thinking about the order in which the individual steps will need to take place. Instead, we are just concentrating on defining the major tasks. You can take this to any level of detail you like, although generally speaking, you should be fine with the level of detail we have chosen here.

In the last phase of project preparation, you will need to decide who will be performing the individual tasks. In our example, we will be doing the clean up work ourselves. The material will be provided by builders' suppliers, and you have two professionals to help you: a qualified plumber for the installation work and for removing the old tub, and a qualified electrician for the electrical installations. And finally, you have your own family to help you get rid of all the rubble.

We will be using Taskjuggler to assign human resources to the individual tasks and create the schedule.

The Tool

The Taskjuggler project management tool is really a collection of libraries and command line tools with a GUI-based front-end for KDE. Taskjuggler autonomously plans individual tasks and resolves conflicting situations, for example, if there is a dependency between the end of one task and the start of another.

Detailed Tasks

Prepare replacement fittings

1. Check water pipes
2. Check heating
3. Repairs
4. Clean replacement fittings

Remove old fittings

1. Turn off water and heating
2. Remove furniture
3. Remove tub
4. Remove shower
5. Remove tiles
6. Remove ceiling

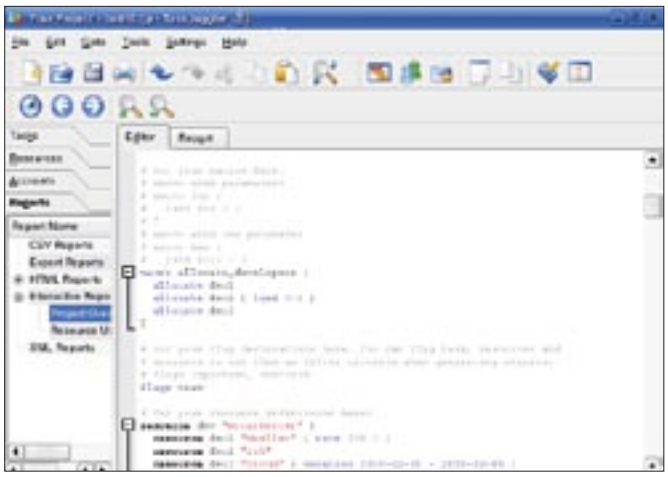

Figure 1: The Taskjuggler editor is used for entering the project data.

Taskjuggler gives you enough leeway to define working hours, leisure hours, and even breaks in the Gantt diagram. Resources can be assigned to groups for ease of management, but this is something we will not need for the bathroom refit project.

Taskjuggler allows you to assign costs to individual resources and to define initial and completion costs for the project. This means you will be able to budget your bathroom refit and manage your cashflow should the project take longer than you envisage.

The program provides useful reports organized by topics such as individual task, progress, or cost. The Gantt diagram visualizes progressive planning and shows you how resources have been assigned to tasks. Taskjuggler gives you a human resources overview, including availability, load, and cost data. The resource calendar shows you which resources are available when and to what extent.

Taskjuggler is a very powerful tool capable of handling professional projects, and it gives you a full set of time, resource, and cost management features. Data input and management are both intuitive, and the tool is rounded off by the reporting functions that visualize the current project status based on the parameters of time, cost, and utilization of resources.

Installation

The Taskjuggler source code is available in the download area of the website at [1]. To build the program, you will need the KDE developer libraries (kdelibs-dev, kdelibs-devel, or similar). The package is bzip2 compressed, so you will need the tar *-j* option to unpack it:

```
tar xfj taskjuggler-2.1.tar.bz2
```

Change to the *taskjuggler-2.1* directory, and *./configure* your Taskjuggler source code, then go on to build and install by entering *make* and *su -c 'make install'*.

Planning

The next step is to enter the individual tasks in Taskjuggler. The program has its own editor for entering tasks (Figure 1).

When you set up a new project, first use the editor to define project data such as the ProjectID (a description of the project that includes information on the time frame for the work), the current date, the time format, and the currency. The entries for our bath refit project are as follows:

```
project bare "Bath Refit" "1.0"
 2005-07-01 2005-08-30 {
now 2005-07-11
timeformat "%Y-%m-%d"
currency "EUR"
```

Listing 1: Subtasks in the Taskjuggler Editor

```
01 task Bare "Bathroom Refit" {
02 task repfit "Replacment
   fittings"
03 task chkwat "Check water
   pipes"
04 task chkhtg "Check heating"
05 task rep "Repairs"
06 task cln "Clean"
07 }
08 task oldfit "Remove old
   fittings"
09 task newfit "Install new
   fittings"
10 task fini "Finished"
11 }
```

```
scenario plan "Plan" {
scenario delayed
"Delayed"
}
}
```

Costs

One of Taskjuggler's biggest benefits is the program's ability to handle costs and cost calculations. This keeps us on top of our budget during the bathroom refit project and gives us the ability to recalculate the project whenever we need to. The next step is to define cost factors by entering *rate 120.0*.

This entry defines the daily wage for the most expensive worker. Taskjuggler has an elegant approach to assigning cost factors based on macros that can be assigned later. This saves typing during data entry. The macro for our project looks like this:

```
macro allocate_workers [
allocate wo1
allocate wo2 { load 0.5 }
allocate wo3
]
```

The workforce *wo1* through *wo3* is covered by a single macro. The *load 0.5* entry shows that the daily wage for the *wo2* label is only half that of the others (factor 0.5). You can then run this macro in the context of a subtask by entering *${allocate_workers}*. We will be defining the project resources in the next step:

```
flags team
resource wo "Workforce" {
resource wo1 "plumber"
resource wo2 "myself"
resource wo3 "electrician"
flags team
}
```

This groups the workforce to create a team, while at the same time assigning people to the *wo1* through *wo* labels. If you need to add more details for the members of your workforce, you can simply add this data to the line for the person involved. For example, our electrician is on vacation between 8.1.05 and 8.10.05.

```
resource wo3 ➥
"electrician" { vacation ➥
 2005-08-01 - 2005-08-10 }
```

Let's assume that the plumber has to rethink the offer he made you and quotes a higher daily wage; you can enter the individual daily wage as follows:

```
resource wo1 "plumber"
{ rate 100.0 }
```

After completing this preparation, you can finally get down to entering and planning the individual tasks:

```
task Bare "Bathroom Refit" {
task repfit ➥
"Replacment fittings"
task oldfit ➥
"Remove old fittings"
task newfit ➥
"Install new fittings"
task fini "Finished"
}
```

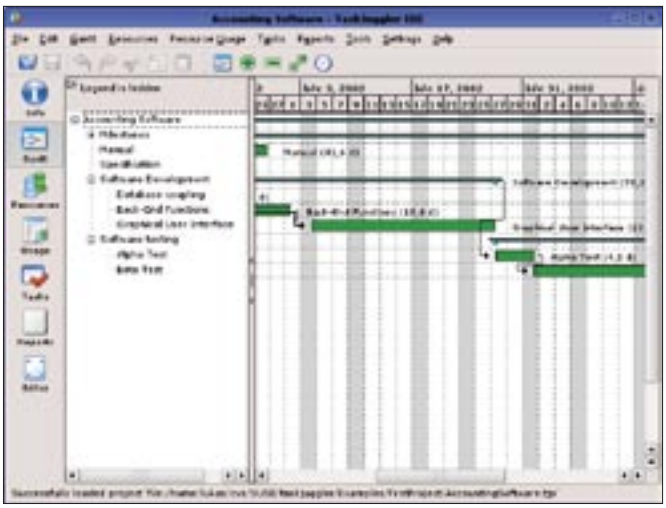

Figure 2: A Gantt plan shows the chronological progression of tasks and subtasks.

In the Taskjuggler editor language, the *task* keyword identifies a task. Each task comprises an identifier, for example *Bare*, and a description such as "Bath Refit." Subtasks for a task are surrounded by braces. Based on our original specs for the refit project, the four subtasks for the main "Bath Refit" task are as follows:

```
"Replacement fittings",
"Remove old fittings",
"Install new fittings" and
"Finished" zu.
```

You can now break down the subtasks. The details in Listing 1 are subtasks for the "Replacement fittings" subtask.

Repeat this for all other subtasks. To complete your schedule, we still need a few details, such as the necessary time for each task and the worker who will perform the task.

Deadlines

Let's start with the first of those items. "How long does each task take?" The Taskjuggler editor has a few special keywords for this, for example, *effort* specifies the number of man-days, followed by the workforce (resource) assignments for the task. The *length* keyword defines the length of the task in working days, and *duration* does the same for calendar days.

Let's assume you have asked the plumber and the electrician to submit an

Listing 2: Subdividing Subtasks

```
01 task repfit "Replacement
   fittings" {
02 task chkwat "Check water
   pipes" {
03 effort 1d
04 allocate wo2
05 }
06 task chkhtg "Check heating" {
07 effort 1d
08 allocate wo2
09 }
10 task rep "Repairs" {
11 effort 2d
12 allocate wo1, wo3
13 }
14 task cln "Cleaning" {
15 effort 1d
16 allocate wo2
17 }
18 }
```

offer for their work, and you can estimate how long your part of the refit will take. The details for the "Replacement fittings" task might then look like the example in Listing 2.

Do you remember the workforce identifiers? *wo1* is the plumber, *wo2* is you, and *wo3* is the electrician. The "Check water pipes" task, which you have assigned to yourself, will take you a day. And the same thing applies to the "Check heating" task. Two days have also been assigned for work to be done by the plumber and the electrician. When they are done, you will be taking over again and finishing up by cleaning the refitted bathroom; again you expect this to take you a day.

The *depends* keyword specifies the chronological order and defines dependencies between multiple tasks. As we want Taskjuggler to calculate the duration, we will be using relative time specifications: "Task 2 can start when Task 1 is completed." Let's look at the "Replacement Fittings" task for a better understanding of the underlying principle. The chronological order is as follows: the "Check water pipes" task is the starting point for the project. This is followed by "Check heating." The repairs can start when the two predecessors "Check water pipes" and "Check heating" have been completed. And cleaning can't start until the repairs have been completed. In Taskjuggler editor speak, this looks like Listing 3.

Bare.start is the starting point for the bathroom refitting project. Its *.start* parameter was automatically created by Taskjuggler. The remaining entries describe relative points in time, which Taskjuggler will evaluate automatically. For example, *!chkwat* is the end point of the "Check water pipes" task. An exclamation mark indicates a relative point in time within a superordinate task (the "Replacement fittings" task in this case.) Two exclamation marks define a relative point in time within the total project, that is, a main task. Based on this, you can enter the chronological dependencies for all subtasks and major tasks.

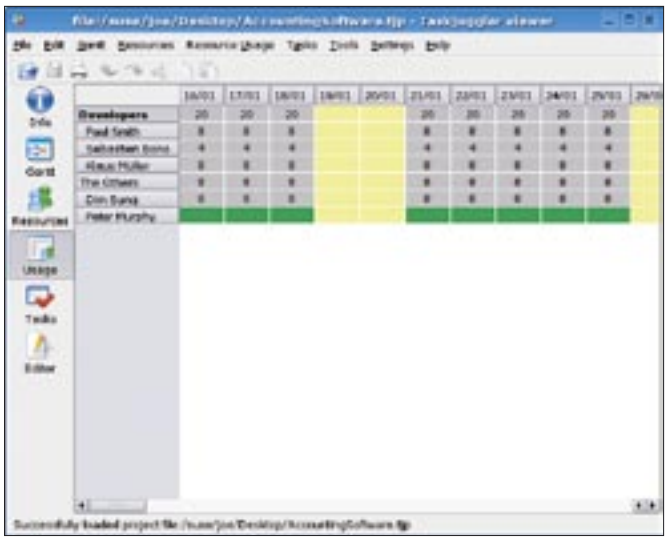

Figure 3: Taskjuggler calculates the cost of a project based on input data such as the hours assigned to each worker.

When you are done, press *F9* to complete your entries. Taskjuggler checks the data for syntactical correctness before going on to calculate the project. You can then see the task interdependencies in the Gantt plan, which shows you when tasks start and finish. An example of a Gantt plan is shown in Figure 2.

Of Mice and Men

Even the best laid plans can go awry, and a project management tool that does not let you make changes to reflect your current situation would be useless. One example of change might be an unforeseen shortage of resources, for instance, if a worker takes ill. On a brighter note, a task might be completed far quicker than you envisaged. For example, when you check the heating and water pipes in the bathroom, you might discover that everything is okay.

You can use the Taskjuggler editor to modify your project schedule to reflect the current status. You need to extend the time parameter to compensate for the worker's sick leave. If repairs turn out to be unnecessary, you can simply delete the task. You just need to enter any changes for the resource, time, and cost parameters using the editor and tell Taskjuggler to rethink the whole project.

As you have consistently used relative times, Taskjuggler has no trouble recalculating the schedule for the project. But if you delete a task that other tasks depend on, you will need to remove these dependencies manually to get things to work.

Conclusions

The intuitive Taskjuggler editor gives users the ability to describe a project and its component tasks. Taskjuggler helps you visualize your entries in the form of an easily readable Gantt chart, and it gives you a useful collection of reports for costing and resource planning. This simple example of refitting a bathroom demonstrates most critical planning elements and shows that Taskjuggler is well-equipped to hander larger-scale projects. ■

Listing 3: Specifying the Chronological Order

```
01 task repfit "Replacement
   fittings" {
02 task chkwat "Check water
   pipes" {
03 effort 1d
04 allocate wo2
05 depends Bare.start
06 }
07 task chkhtg "Check heating" {
08 effort 1d
09 allocate wo2
10 depends !chkwat
11 }
12 task rep "Repairs" {
13 effort 2d
14 allocate wo1, wo3
15 depends !chkwat,  !chkhtg
16 }
17 task cln "Cleaning" {
18 effort 1d
19 allocate wo2
20 depends !rep
21 }
22 }
```

INFO

[1] Taskjuggler: *http://www.taskjuggler.org*