Common Approaches to Building Structures

Here are some of the common approaches you can use to visualize your work breakdown structures.

Top-Down Approach

Common use cases:

Such approach works best when you need to visualize the complete hierarchy of issues, starting from the most high-level elements all the way down. For example, a work breakdown structure for a number of projects, where you may have some high-level initiatives on top, then some tasks that implement those initiatives, which are then broken down into smaller tasks. Such overview is required for tracking the progress across a large project, where it's important to roll-up data such as estimates, time spent and progress from low levels all the way to the top. Once built you can then drill down to see the particular details - for example, find the tasks, which are hindering the progress of an initiative.

Performance considerations:

Because of the nature of the use case, such structures can get pretty big. We are visualizing the entire scope of work. It's not necessarily a problem, but usually they are more performance heavy than smaller structures, especially if you have many levels connected by links. It's totally fine to use them, when a full overview is needed, but it's best to avoid the situation, when all users use the same one big structure on a daily basis.

It's best to keep these structures visible to the appropriate users only - for example, project or program managers. It's not recommended to build such structures for individual teams, which are interested only in a sub set of such structure (see the other example below). Also, do not set them as default structures. Default structures are opened by default on the issue page, which means, every time a user opens an issue, this whole structure is loaded and then the part of it related to the issue is shown.

How to build:

The general idea is to start by adding top level items using an Insert generator and then use a number of Extend generators to pull in related items. Here are the common steps:

1. Add the Insert generator that will add the top level items. You can use the JQL Inserter as the most flexible option or add issues from a particular Agile board, for example.
2. Add the Extend generators to pull in the child issues. Depending on your setup, you may have to add several extenders to visualize different types of relationships. For example, you second level items can be Epics linked to your top level items (let's call them Initiatives) with an Issue Link. To pull them in, add the Link Extender for the correct link type and direction. Then to pull in stories under Epics use the Agile Extender. To show sub-tasks, use the corresponding Sub-tasks extender.

It's strongly recommended to set the Levels option for all Extend generators - this will make everything more consistent and will help with performance (especially in larger structures). In the example above, you'd set the Levels to Current level only for the first Link Extender.

If you have several levels linked with a particular link type, you can set the levels as a range. Please note, that the **To** level is the last level that should be scanned for issues to be pulled in, not the level that is pulled in. For example, if you have Theme -> Initiative -> Epic hierarchy in which three levels are connected with the same link type, you want to set the levels to: *1 to 2*. This means the generator will scan the top level (themes) and will pull in the second level (initiatives). Then it will scan the second level and will pull in the third level (epics) and will stop. There is no need to scan the third level because there is nothing linked to it with this link type.

Following the example above, to pull in stories under epics you will set the levels option: *2 to 2*, since the Epics are on the second level and we don't need to check if there are any stories under stories. Same idea should be applied to the sub-tasks extender.

Limitations:

Since such a structure is built top-down, we will only see issues that have an incoming link. This may be not convenient, when we want to visualize the entire set of issues in the form of a hierarchy. For example, we may want to visualize issues from a number of Scrum Boards. In this case, stories, which do not have an epic defined, will not be shown in such a structure.

There are several ways to resolve this. One approach is to build a structure slightly differently - please see the **Visualizing Entire Scope** section.

Another approach is to use the secondary panel to identify the missing issues. Open the secondary panel, select JQL option and specify the JQL which describes the scope you want to see in a structure. Click the Diff button in the toolbar ( ) to hide issues which are already present in your structure. This will give you all the orphans, which you can now drag-and-drop into correct position in your structure.

Focus on a Specific Level

Common use cases:

Often you have a multi-level hierarchy, but are primarily interested in issues on a particular level. For example, you are managing a team that works together with a number of other teams and you want to see how the scope you are responsible for fits into the full hierarchy. So you would like to see a set of issues you are interested it and then show the child issues below them and the parent issues above. For example, in a hierarchy where you have Initiative -> Epic -> Story -> Sub-task, you want to see the Stories from a certain sprint and all the related issues - both parents (epics and initiatives these stories belong to) and sub-tasks.

Performance considerations:

One of the options would be to build a full hierarchy as described in the section above and then filter it to show only the part relevant to the user. This approach has one significant drawback - it means you would need to build a really large structure first, and then hide a part of it. This means that even though the issue count in the resulting structure maybe low, it will still have a significant impact on the performance.

A better approach in this situation would be to start by pulling in the scope you want to see and then use group Generators to show the levels above. This will produce a much smaller structure straight away, plus the groupers are more efficient in terms of CPU than extenders.

How to build:

The idea is to use the Inserter to add the initial scope - the issues you are focused on, and then use groupers to show the parents and extenders to show the children. Here is how to build a structure using the example above (Initiative -> Epic -> Story -> Sub-task hierarchy, but only with stories from a particular sprint):

1. Use an inserter to add the initial scope. For example, you can use the JQL inserter to pull in issues from a certain sprint.
2. It's very simple to show the children - in this example sub-tasks. Simply add an extend generator for subtasks (don't forget to set the correct level settings - in this case, Current level only).
3. The next step is to show the direct parent - in this example, Epic. To do it add a *Group by Epic* generator. The stories will be distributed between the epics they belong to. The ones without an epic will end up in the No Epic folder. If the parents are connected to children with issue links instead of epic links, use *Group by Issue Links*, select the link type and direction and you'll get parent items shown as parents.
4. If you need to show one more level above the one you've just created (like in the example, initiatives above the epics), you'll need to add another grouper. There are two important things you need to do here:
    a. As you add this second grouper, make sure you select the "Consider other groups" option. Without this checkbox, the grouper will try to group not the level we've just created, but the issues that were added by the inserter originally (as when you are grouping by two fields, for example).
    b. Once the generator is added, reverse the order of these two groupers in the generators list, so that the one you group by first is higher.
5. If there are more levels above - keep adding more groupers, until you get the last level you want.

Visualizing the Entire Scope

Common use cases:

In some situations you need to see a certain scope of issues arranged into hierarchy. For example, a number issues from a certain project, which are related to each other like in the example we used above. Initiatives, epics, stories and subtasks, but only from a particular project (or component, version, team, etc).

Performance considerations:

The approach to building this kind of structure is very similar to the top-down approach we discussed above. If you make sure to use the correct level settings, the performance impact will be comparable, so the implications are the same.

How to build:

The main difference from the simple top-down approach is that we start by adding not only the top-level items, but the entire scope of issues and then want to organize them into the hierarchy. Here is how to build it:

1. Use the insert generator to pull in the issues you want to see. For example, the JQL inserter.
2. Add the Extend generators to pull in the child issues like we've done before. The important difference in this case is that as you add extenders, you will be getting duplicate issues - one instance of the child issue is added to the top level by the inserter and then another is added by the extender. To hide the duplicates, proceed to the next step.

    > It's very important to set level parameters for extenders in this configuration - even more so than in top-down approach. Failing to do so, can seriously affect the performance.
    >
    > If you do not restrict the scope of extenders by setting correct levels, you will end up not just with 2 instances of all child issues, but with an instance per level, which can be a lot of issues, if you have 4-5 levels.
    >
    > To illustrate, let's use the same example (Initiatives, Epics, Story, Sub-Task). If you do not set the levels for extenders, here is what you will get: On the top level you will see the results of the inserter. On the second level, you will see results of each of the extenders - epics under initiatives, stories under epics, sub-tasks under stories - all duplicates. On the third level you will see stories under epics and sub-tasks under stories - the third time we see them. And on the forth levels we'll see the sub-tasks for the fourth time under stories.
    >
    > Even though all the duplicating items can be hidden (see the next step), we still had to build a really large structure first, which will affect performance negatively. If you do set the levels, we'll only get one set of duplicates after adding extenders.

3. Add Remove Inserter/Extender Duplicates filter generator - it will hide the issues added to the top level by the inserter if they were also added later by the extenders. The ones that do not have any parent, will stay on the top level as a result.
4. The extenders we have added could pull in some issues, which are outside of the original scope we defined. For example, epics in our project can have some stories from other projects too and the agile extender would pull them in. So if we only want to see issues from our original scope we'll need to add a JQL filter generator on top with the same query we used originally.