

Multuser support in Web Hob v0.3

A few different ways to support team workflows

| | |
|-------------------------------------|----|
| Multiuser requirements | 3 |
| Option 1 - A basic approach | 4 |
| Option 2 - Share option | 5 |
| Option 3 - Diff option | 6 |
| Option 4 - Locks option | 7 |
| Option 5 - git option | 8 |
| Requirements support | 10 |
| Multiuser themes by Tobias & Tobias | 11 |

| Requirement | Score |
|--|-------|
| Support multi-user teams (multi-user accounts, permissions, team tools, etc) | 5 |
| Enable sharing of configuration and setup process with others | 2 |
| Enable duplication of configuration and setup process with others | 2 |
| Teams work separately on local machines then submit changes to server | 2 |
| Allow teams to open and modify files on server quickly | 1 |
| Enable secure sharing of source code within a team | 0 |
| Enable secure sharing of builds within a team | 0 |
| Enable sharing of built toolchains | 0 |
| Enable sharing of intermediate build sets | 0 |
| Enough user account support to allow for different permissions / access | 0 |
| Support source control without surfacing its complexity | -1 |

Multiuser requirements

This is a list of the multiuser requirements that were identified as part of the strategy project led by Tobias & Tobias.

The table also shows the results of the requirements scoring exercise .

It is worth remembering that the scoring exercise did not include any non-Intel parties, except Tobias & Tobias themselves, and therefore might not reflect other Yocto Project partners' priorities.

NOTE: I have split the following requirements:

- "Enable sharing / duplication of configuration and set up process with others"
- "Enable secure sharing of source code and builds within a team"

Some of the design approaches in this document might support one of them (sharing or duplication, source or builds), but not the other.

1. A basic approach

This option aims to provide the minimum functionality required to cover the multiuser use cases.

It implements only 2 user types with very basic permissions, and relies on copy, import and export functions.

This is the option reflected in the design work from Tobias & Tobias, under the premise that it is better to start small than to start with a complex solution and then discover it is the wrong one.

2. Users and permissions

In this option, Web Hob provides 2 types of users:

Project owner: a user creating a project is the project owner of such project. Project owners have full control over the projects they own. They can give other users access to the project and allocate them different permissions (see below).

Project user: any Web Hob user who has access to a project without being the project owner. Project users can have the following permissions:

- Read only
- Download project output
- Run project builds
- Copy project

3. Scenario

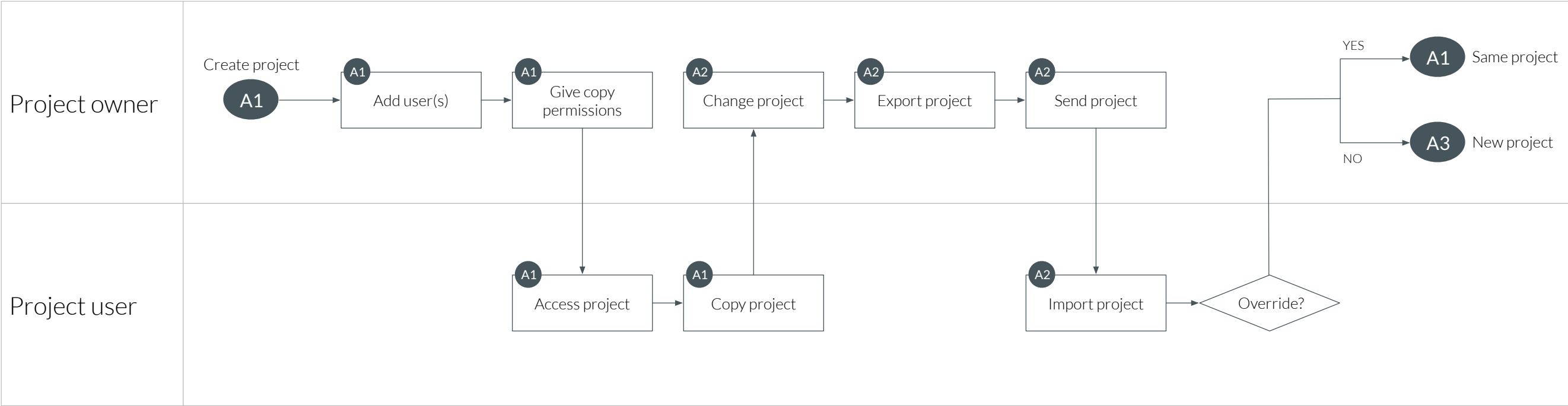
Paul creates a project A1 and he's therefore project A1's owner. He needs to work with Jessica, so he gives her access to project A1 and grants her copy permissions.

Jessica accesses project A1 and copies it, so she is now the owner of project A2, which is identical to A1. She might give Paul access to project A2, maybe on a read-only basis.

Jessica has made some changes to the project A2. In order to share them with Paul, she exports project A2 and sends it to Paul. Paul imports the project A2 and Web Hob gives him 2 options:

Option 1: create a new project A3, which will be identical to A2.

Option 2: override project A1 with A2, so that A1 is now identical to A2. Changes are kept on project A1's change history.



1. No 'Copy' and 'Export' as 'Share'

As Jessica pointed out, the minimalistic approach should also be functional without the 'Copy' functionality (i.e. relying exclusively on export / import options).

As suggested by Richard, the export / import process could be streamlined by replacing it with a 'Share' function that would perform the export + send tasks. This would have the additional benefit of handling the whole project sharing process within Web Hob (i.e. no need for sending exported project files using external tools).

2. Users and permissions

Eliminating 'Copy' means that the 'Copy project' permission is no longer necessary.

It would also be possible to implement a Web Hob with only one user type (the project owner), since not supporting 'Copy' means we could eliminate all access to other users' projects. However, to give other Web Hob users the ability to run builds and download build output from my projects still seems valuable, even if it is not possible to 'Copy' the project.

So, we should still have 2 user types: 'Project owner' and 'Project user', but the latter would have only 3 permissions:

- Read only
- Download project output
- Run project builds

3. Scenario

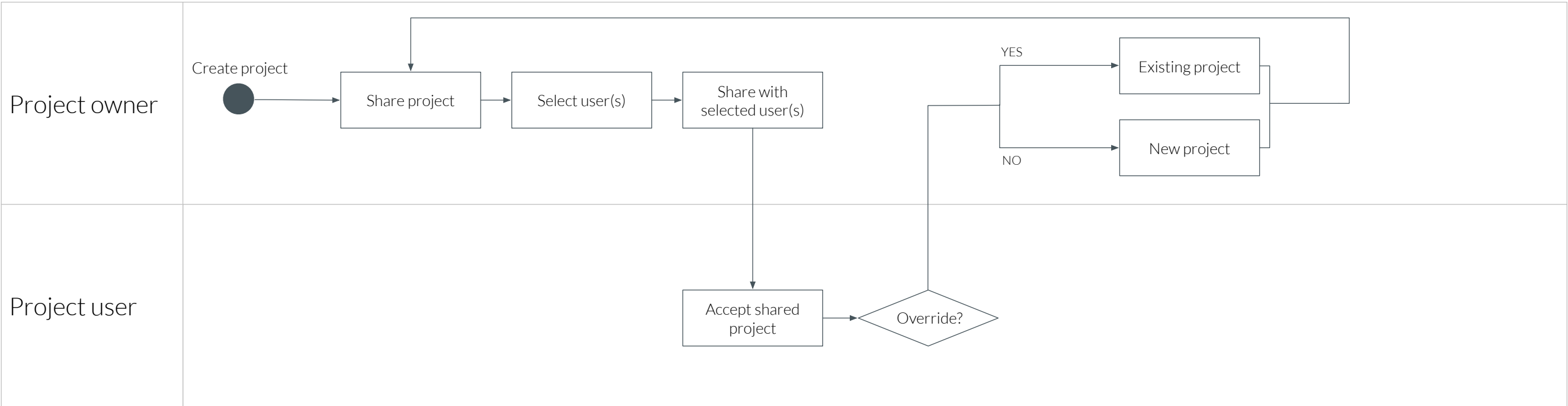
Paul creates a project A1 and he's therefore project A1's owner. He needs to work with Jessica, so he shares project A1 with her.

Jessica accepts the shared project. As part of the acceptance process, she is offered 2 options:

Option 1: create a new project A2, which will be identical to A1.

Option 2: override an existing project, which will become identical to A1. Changes are kept on the overridden project's change history.

To deliver her changes to Paul, Jessica shares her project with him.



1. An enhancement to options 1 and 2

Richard suggested to enhance the override functionality with the ability to see the differences between the project to be imported and the one to be overridden.

Users should then be able to choose which changes they'd like to bring into the project to be overridden.

This enhancement could be applied to both Option 1 (Basic) and Option 2 (Share).

2. Users and permissions

The 'Diff' option has no impact on the users and permissions for Option 1 (Basic) and Option 2 (Share).

3. Scenario

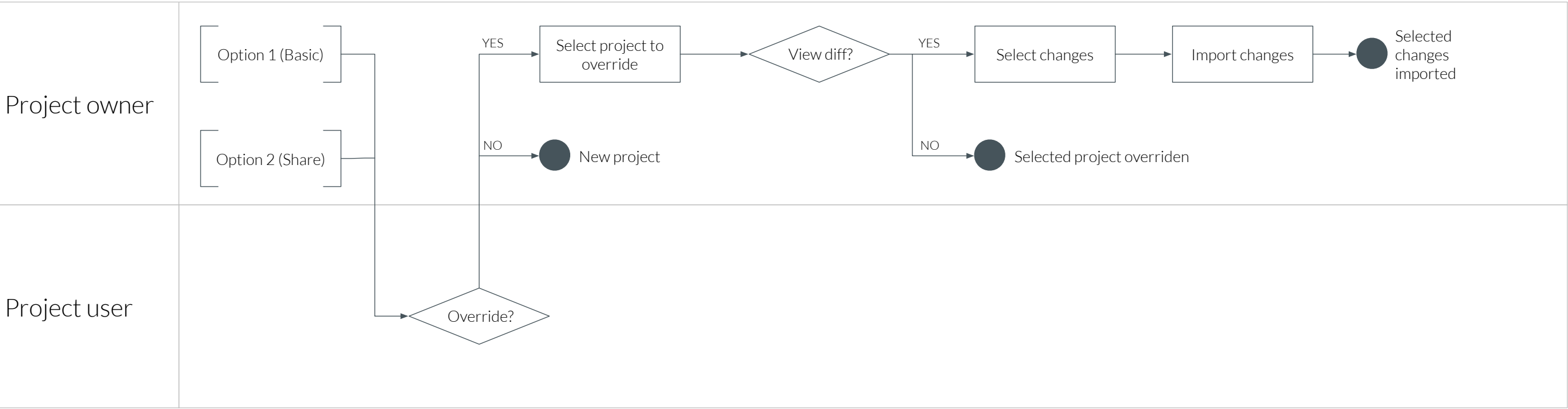
Paul creates a project A1 and he's therefore project A1's owner. He needs to work with Jessica, so he shares project A1 with her.

Jessica accepts the shared project. As part of the acceptance process, she is offered 2 options:

Option 1: create a new project A2, which will be identical to A1.

Option 2: override an existing project, which will become identical to A1. Changes are kept on the overridden project's change history.

To deliver her changes to Paul, Jessica shares her project with him. Paul is only interested in a subset of Jessica's changes, so instead of overriding the whole project he views the list of changes and chooses to 'import' only the ones he is interested in.



1. Based on limited numbers of system developers per project

This model was used in the initial Web Hob concept. It was also brought up by Richard P. and Dave S. It rests on the assumption that even big teams will have only a reduced number of system developers. The bulk of team members have other roles, such as application development and QA.

System developers would edit a single, shared set of project files in the build server. Since not many system developers will be editing project files, it would be possible to lock any files that are being edited so that no conflicting changes can occur. Once editing finishes, the lock is released, and the file is ready to receive additional edits.

2. Users and permissions

There are 2 types of users in this model:

- 1. **System developers:** they have permission to write to project files.
- 2. **Other users:** they don't have permission to write to project files.

3. Scenario

Paul and Jessica are the system developers in project X. They work with application developers and a QA team in project X, but they are the only ones who can edit the project files.

Whenever Paul starts writing to a file, the file is locked: Jessica can read the file, but she cannot write to it. She knows which file Paul is writing to because Web Hob tells her so. She is able to check the nature of Paul's changes via a history feature.

The 'Latest activity' and 'History' features in the initial Web Hob concept

Latest Activity


Yesterday 14:45

Changed setting on **Yocto Block Rollout**

Yesterday 14:43

Selected **busybox-1.19.4** in **Features**

Two days ago 14:35

Invited  **Joe Thomason** **Yocto Block Rollout**

25 May 2012 14:15

Added **busybox-1.19.4** to **Features**

Yocto Web Hob

Builds

Projects

Groups

Search

Queue















Settings

Jim

Yocto Block Rollout

Project

History

| | | |
|---------------------------|--|---|
| 25 May 2012 14:45 (GMT 0) | Changed setting on the Project |  Bob Woodward |
| 25 May 2012 14:43 (GMT 0) | Selected busybox-1.19.4 in Features |  Bob Woodward |
| 25 May 2012 14:35 (GMT 0) | Invited  Joe Thomason to the project |  Bob Woodward |
| 25 May 2012 14:21 (GMT 0) | Add busybox-1.19.4 to Features |  Bob Woodward |
| 25 May 2012 14:15 (GMT 0) | Deselected acl in Recipes |  Bob Woodward |
| 25 May 2012 13:43 (GMT 0) | Modified something in Features |  Mike Tompkins |
| 25 May 2012 13:35 (GMT 0) | Packaged core-image-minimal-atom-pc-11233768765 |  Mike Tompkins |
| 25 May 2012 13:21 (GMT 0) | Created a build core-image-minimal-atom-pc-11233768765 |  Mike Tompkins |
| 25 May 2012 12:40 (GMT 0) | Selected busybox-1.19.4 in Features |  Mike Tompkins |
| 25 May 2012 12:33 (GMT 0) | Modified something in Features |  Mike Tompkins |
| 25 May 2012 11:35 (GMT 0) | Added a the group QA Team |  Bill Watson |
| 25 May 2012 11:01 (GMT 0) | Features |  Mike Tompkins |
| 25 May 2012 10:05 (GMT 0) | Created theYocto Block Rollout Project |  Bill Watson |

History?

Nullam quis risus eget urna mollis ornare vel eu leo. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Nullam id dolor id nibh ultricies vehicula ut id elit.

[More about the build process and how to reduce errors in your builds »](#)

1. Three types of changes

This model was outlined by Paul E. and Jessica Z. separately, and it builds on the familiarity with git of the Yocto Project user base. Users work on their own branches of a master project and submit patches to be merged to this master project.

It distinguishes between 2 types of changes:

1. Changes to project configuration: Web Hob gives read-only access to configuration files in the build server, but provides the ability to edit those configuration files via its GUI. Configuration file changes are then submitted / merged / pulled within Web Hob following the git model.

2. All other changes: users can change any other project files in the build server or using Web Hob. Within Web Hob, such changes can be downloaded as patches, so that they can be submitted / merged / pulled outside Web Hob using a source control management tool (git or any other). Since all layers added through Web Hob are linked to an SCM tool, any updates to the layers via the SCM tool should cascade to Web Hob.

A special case are changes to package source code: changes to package sources can also be downloaded as patches, so that they can be submitted / merged / pulled outside Web Hob using an SCM tool; but Web Hob provides functionality to update the affected recipes within a project branch.

2. Users and permissions

There are 3 types of users in this model:

1. Project maintainer: they review project configuration changes and merge them to the master project.

2. Project contributor: they can

- change configuration in their project branch
- submit configuration changes to the project maintainer
- pull configuration changes from the master project at will
- change any other files and download them as patches.

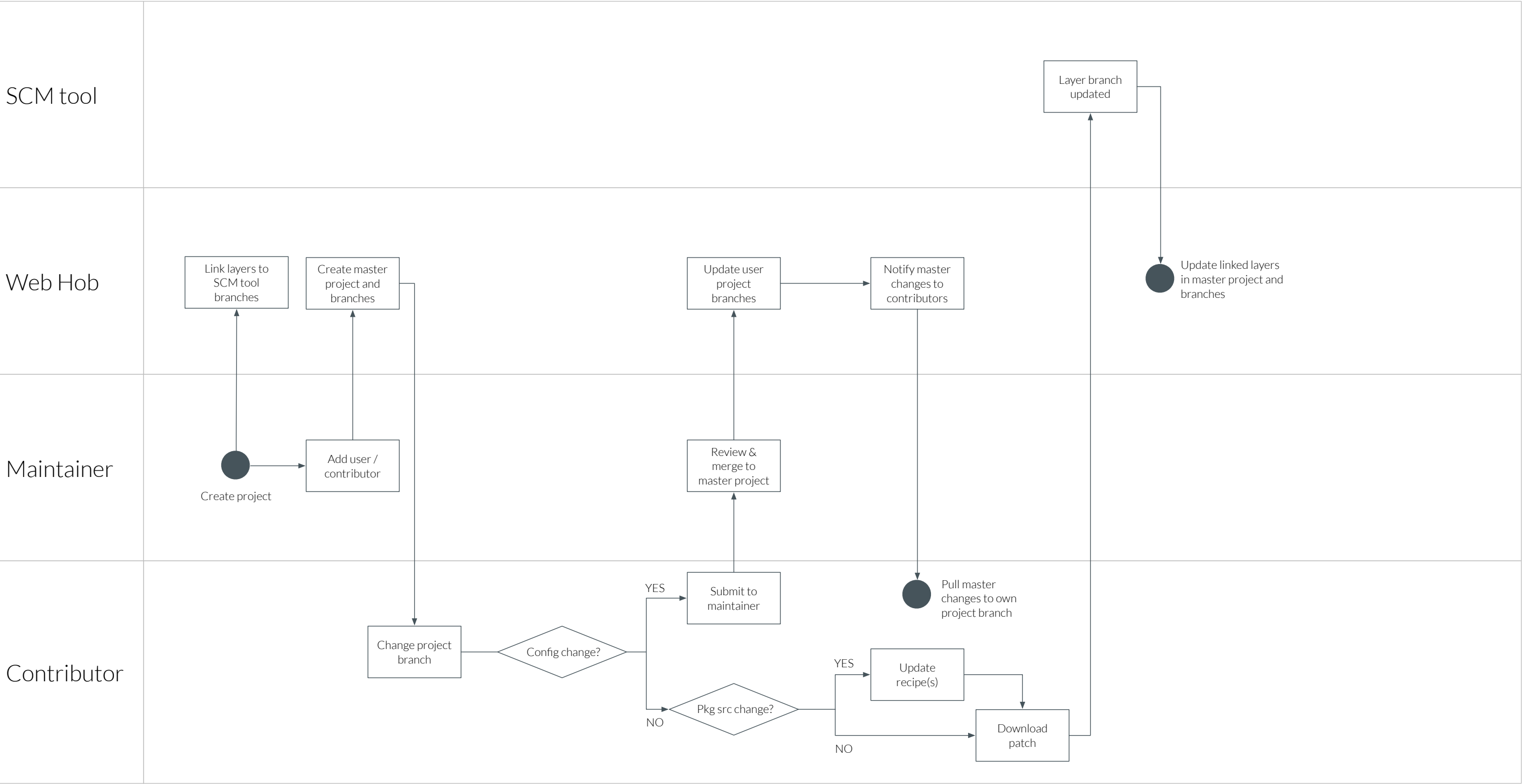
3. Project user: they cannot edit the project files, and any changes to the master project are seamlessly pulled to their project branch.

3. Scenario

Paul creates a project X and he is therefore the project maintainer. He adds Jessica to project X as a contributor, and Alex as a user. When Jessica and Alex log in to Web Hob, they'll find a new project called X on their project lists.

Jessica makes some changes to the project X configuration and submits those changes to Paul. Paul reviews and merges the changes to the master project. Alex's project branch is automatically updated, but Jessica's is not: instead, she receives a notification every time the master project changes. Whenever she is ready, she can pull the changes to her project branch.

Paul, Jessica and Alex have a custom layer they want to use for project X. The custom layer is maintained in a git repository. When creating project X, Paul added the custom layer and specified which git branch project X should use. Whenever that git branch is updated, the changes cascade to project X.



| Requirement | Score | Option 1 (Basic) | Option 2 (Share) | Option 3 (Diff) | Option 4 (Locks) | Option 5 (git) |
|--|-------|---------------------|---------------------|--------------------|---------------------|-------------------|
| Support multi-user teams (multi-user accounts, permissions, team tools, etc) | 5 | basic | basic | basic | basic | basic |
| Enable sharing of configuration and setup process with others | 2 | no | no | yes | yes | yes |
| Enable duplication of configuration and setup process with others | 2 | yes | yes | yes | no | yes |
| Teams work separately on local machines then submit changes to server | 2 | no | no | no | no | yes |
| Allow teams to open and modify files on server quickly | 1 | yes | yes | yes | yes | yes |
| Enable secure sharing of source code within a team | 0 | no | no | yes | yes | yes |
| Enable secure sharing of builds within a team | 0 | yes | yes | yes | yes | yes |
| Enable sharing of built toolchains | 0 | yes | yes | yes | yes | yes |
| Enable sharing of intermediate build sets | 0 | yes | yes | yes | yes | yes |
| Enough user account support to allow for different permissions / access | 0 | basic | basic | basic | basic | basic |
| Support source control without surfacing its complexity | -1 | no | no | no | no | yes |

On February 6th, upon request from Dave, I contacted Tobias & Tobias with the following request:

"Dave suggested we go back to the roadmap work and the validation sessions you guys ran in Barcelona, to understand which level of multiuser support was perceived as useful and we should be supporting. Brendan: Dave specifically asked for your involvement on this, since you were leading that work.

If multiuser flows were discussed during stakeholder and user interviews, could you summarise the outcome for us?"

Below is the answer provided by Tobias & Tobias:

I'll start out with a summary of the multi-user themes & needs that came out of the initial round of interviews. I've taken these from the "Interview Matrix" that Ed & I produced once all those interviews were done and their notes transcribed.

The themes themselves

We identified a number of themes from these interviews. The ones I'll focus on here, as I think they're relevant, are:

- Collaborative environment
- Collaborate more widely
- Share configuration

There are a few other items from those interviews which don't fall under these themes but which might still be relevant. I'll highlight them later on.

List of items

Here are the items from these themes. For some, but not all, I've added some commentary for the purposes of this summary.

* "Take advantage of someone else's work and CPU time" - this implies a flow where user A, when putting together a build, finds that some packages they're going to include have already been compiled by user B on the same server, meaning that they won't need to be recompiled. In other words a cache of compiled packages is maintained by the server which all users have access to.

* "Allow some users to kick off builds but limit some users to only get package or shared state feeds" - implies a flow where someone in an administrator role manages these permissions, and other flows where certain user types have their capabilities restricted when using Web Hob. My personal view is that this aspect of user management should come later on in the roadmap; it becomes useful when there are very large-scale installs of Web Hob whose user populations vary greatly in sophistication.

* "people work on own machine and do local builds and then submit their changes to the team server through git, specifying a tag on source repository which they are going to be building, or a branch" - but maybe Web Hob wouldn't add much value for a user in this situation?

* "Siloed workspaces, work in a private area then share, collaboratively iterate on image" - the implication here is that User A would be working on files stored on the server rather than locally, but the server would maintain these files in a private area devoted to that user. User A might then choose to make some files visible to others, which implies another flow where User B views a set of files edited or made available by User A and chooses to import some of them into their project. Also, in some cases User A might want to do a server-wide overwrite of a certain file, meaning that User B would have no choice about importing.

* "Put tooling in place to enable sharing of intermediate build sets"

* "If we have six people working, why should everyone have to build a toolchain for Xeon when it can be built once?"

* "securely share source code and builds amongst larger teams" - this resonated with some points raised later on, in Barcelona. This security issue is a big advantage of Team Web Hob over Public Web Hob, but when looked at more closely it became a bit more granular. There could be a situation where various developers are working on a single project, but certain files within the project are restricted to only a sub-set of those developers. For example, people in a certain country might not be allowed to view source code that plays a role in DRM or security.

This would create user flows where those permissions are set up on a file-by-file basis, and others where a user might be able to make use of already compiled binaries for a certain package but not be able to view its source code. But this is something I would place later in the roadmap as its usefulness is closely tied to the scale of Web Hob instances. This kind of granular permissioning would probably be overkill in the early phases.

* "Be able to see wide selection of recipes from user community at large" - even though this comes more from Public Web Hob it still has implications for user flows in Team Web Hob. For example User A might create a recipe and choose to make it available for other users of that Web Hob server to view, and User B might see that recipe in a list and bring it into their build. There would then be questions such as whether this kind of "recipe browsing" takes place before or after hardware selection, so the list of recipes shown can be limited to those created for the same hardware, etc; or whether a user who creates a recipe on a Team Web Hob should be able to restrict its visibility to only certain user types, as opposed to a simple "private / public" dichotomy.

* "To see global information" [on build errors] - this was also something that came from the Public Web Hob concept but has implications from Team Web Hob also. When encountering a build error, a TWH server might poll a central database (on a PWH server somewhere) for user-submitted information about that error. But any information it gets from that search might be presented in the UI with less prominence than information in its *own* database: in other words, when User A sees a build error, comments left by User B, her colleague, appear above comments left by the wider Web Hob user community and retrieved from Public Web Hob. The follow-on implications are that there is a user flow for leaving comments for build errors; considering the fact that the user is only really in a position to leave that comment once the error has been resolved, so comments would need to be enabled for historical errors and not just those currently seen on screen; whether User B wants his comment to be visible to all other user of that TWH server or just a sub-set; whether the TWH server should "phone home" and upload error content to the Public Web Hob's central database; how the admin would configure the "phoning home", or whether that should be handled by users on a comment-by-comment basis, etc.

* "Sharing configurations with one another, replicate setup" - so in this situation User A might be able to browse configurations set up by other users and apply them to their own Web Hob area. This can be extended across Web Hob instances as well. For example, there could be a team of developers with a TWH server who want to replicate a configuration from their vendor's server (e.g. TI or Mentor Graphics). However this does raise a question of whether it's feasible for configurations shared via Web Hob to be applicable to a user's local environment, as opposed to their environment on the Web Hob server.

-- EMAIL CONTINUES 15th FEB --

* "Receive package feeds or shared state package feeds, possibly via TWH" - similar to the above mentioned. The aspiration discussed in this interview was for User A to have a working configuration and for this configuration to be packaged and made downloadable by other users within the same network, or possibly outside. It's worth noting that this shared-state topic was discussed most with Chase Maupin at TI, for whom it would make customer support a lot easier. In a way the TI requirement is for a very enterprise-level implementation of multi-user functionality in which one organisation can use TWH to support other organisations, not just as a build server but as a source of replicable configs, config feeds, and package feeds.

* "Set up machines at conferences with GUI tools to give people something interactive to click around in the booth" - this might seem like an odd area in which to identify multi-user flows but bear with me. If there is a TWH server set up at a conference and the UI is being made available to delegates for them to have a play with, we could imagine:

- A machine at the booth logged in to that WH instance with a specific account, where users can safely play around

- The URL of the server to be publicised so delegates can access it from their own laptops, not just the booth machine (this is more in the spirit of WH I guess)

- Delegates accessing from their own machine maybe being able to set up their own accounts on that WH instance. There's a potentially interesting scenario there, I think, that could really help the Web Hob (and by implication the Yocto Project) communicate its essence, purpose and value to conference delegates once multi-user functionality has been implemented.

* "Using the sandbox Web Hob" [to learn how things work] - this implies another multi-user flow where a TWH may treat a new user as effectively "sandboxed" so they can't do any damage while poking around the system. Practically this would mean that if they edit any file it's branched from the main trunk, they can't commit anything back, and they basically have read-only access to files & images on the server. Until an administrator intervenes and changes their permissions, they cannot interact with or affect the work of other users - but they do see what other users are doing, so they have a kind of multi-user experience even though they're behind a glass wall. I imagine a TWH admin may want to set things up this way if they have a lot of people in their organisation wanting to see how Web Hob works but who can't be trusted to mess things up.

* "Modify files on a server... Without having to retrieve files from server, modify, then upload again" - this indicates a whole family of user flows that exist around file modification. Editing source within the Web UI is fairly straightforward but we'd also need to consider situations such as:

- User modifies a file but does not want others on the team to see that modification, e.g. because it's experimental

- User looks at a source file in the central repo but also sees versions/branches of that file created by other users, and might want to import one of those into their own environment

- User modifies some files in a layer and then wants to make that layer available to others, as opposed to the discrete files themselves

- An admin user forcibly replaces all versions / branches of a file on a server with a single version

- An admin user removes right of some or all users to edit certain files, or files within a directory, or layer

- When looking at version history of a file a user may want to see information such as build errors encountered on previous builds that were triggered by that file, along with things like commit notes

- User wants to work on source but outside of the Web Hob UI altogether, so looks for ways to ssh to the relevant directory on the server and access the files directly. There are two hazards when extending out this feature set. One is redundant development, and can be avoided by using existing libraries rather than building from scratch (code editors are an example that comes to mind). The other is redundant functionality, and can be avoided by focusing on tasks that Web Hob specifically can expedite rather than those that people are faster/happier doing with other tools: to put it another way, if we try to replicate Eclipse in the browser we're on the wrong track.

--

So that's my summary of multi-user themes & user flows that were touched on in the initial interviews, with additional comments on their implications. And I apologise for the stream-of-consciousness nature of the email - I hope it's been helpful! Let me know if you'd like to discuss any of these points further or explore in more detail.

Many thanks,

Brendan

Questions, comments, feedback?

Contact Belén

belen.barros.pena@intel.com

or the Web Hob mailing list

<https://lists.yoctoproject.org/listinfo/webhob>