Introduction
This document describes the complete git setup KDE could implement. We are fully aware that parts of this subject have been discussed to death numerous times on this list, but we feel that when we put it into the wider perspective of the complete setup, we could arrive at a different solution than what has been made previously. We hope you can find the energy to look at our proposal objectively.

This document has been written from a sysadmin point of view, but we are fully aware of our position within the decision about the problems outlined below. We can merely give advice on how we would approach the problems. The decision about the solutions is entirely yours and we will respect that completely. Consider this document as well meant advice.

We would appreciate that this thread will not end up in an gigantic discussion. Of course, it is fine to reply to ask for clarification but please try to keep the focus on this document instead of bikeshedding all over again about the pros and cons of a certain solution. You all can read them in the archives of this list and probably in this document.

The following is a summary of how KDE's Git repositories can be laid out. There are two options available: Monolithic repositories, and Split application repositories.

Description of the problem: monolithic vs split
First, monolithic repositories. With that we mean that we choose to match the current top-level layout as we have in subversion. For example, that means that there will be one repository called 'kdepim' and within that repository all applications are available.

With split we mean that each application will have its own repository. There will be no 'kdepim' repository but there will be different repositories for kmail, kalarm, korganizer, etc.

There will be some exceptions to this rule. We are pretty certain that the kdebindings developers will choose to split. We also think kdelibs should not be split. The split that has been made on the CMake level for this module already accommodates some of the advantages of splitting.

Executive summary
From a sysadmin point of view we would like to suggest the following setup:
- Split repositories. One repository for each app.
- A flat structure of repositories on git.kde.org.
- The repository of the app is attached to the project page of that same app within Redmine.
- The existing subversion hierarchy is matched within Redmine.
- Use the same globally unique name in Redmine, Reviewboard and as git repository name.

Let's now zoom in to these points.

Advantages to the monolithic approach
Generally speaking the scm-interest mailinglist is leaning towards the monolithic approach. In that case there is only a single repository that has to be cloned to be able to contribute to one of KDE's modules. If you for example want to focus on educational applications, you can get everything you need to have with a single clone command.

This also allows items to be freely moved around within the module, without involving movements between several repositories. It also allows for a single module tag to be easily established by the release-team.

Disadvantages to the monolithic approach
The biggest downside is that it will be very difficult to move applications out of modules while still keeping history intact. It will also be impossible, without completely rewriting history, to expunge the application from the git repository. Rewriting history (by that we mean invalidating all SHA1-sums) of a published git repository goes against all established work-flows for git, and requires that all contributors effectively re-clone.

That means we should arrive at the conclusion that with a monolithic approach we can no longer move applications across repository boundaries. This would seriously break the current work-flow within KDE.

Typically an application's life cycle is: start in playground, move to kdereview, and move to a main kde module or extragear. That means that the application has to move 2 times out of a repository and into a new one. This work-flow would no longer be possible.

If we keep monolithic repositories, we need to reconsider this work-flow. For example, allowing new applications to be 'born' in the final destination to prevent moves. In that case we would stop playground and allow apps to be created in kdepim or the kdeedu repositories, for example.

This will have the disadvantage of increasing the size of clones of monolithic repositories. This makes cloning prohibitive over time, especially for those in low-bandwidth or high data cost areas and also in terms of disk space consumed on developer hard drives. This will create a higher barrier to entry for contributing to KDE applications. Something that's opposite of some people's motivation to move KDE to git.

Effects of the decision on infrastructure
Surrounding the move to git, the sysadmin team will provide some new services to accommodate the move. Let's see how the choice reflects on those systems. We are aware that technology should not dictate the work-flows but it would be narrow-minded not to look at the consequences.

Reviewboard
Reviewboard will be used for patch review. Unfortunately, Reviewboard only operates on a flat structure. There is no possibility to use a hierarchy in the projects. We would like to keep a 1-to-1 mapping between repositories and Reviewboard projects. This will allow us to automate the process between a new project and automatic creation of the reviewboard setup.

In the monolithic setup we would create a 'kdepim' reviewboard entry. This entry holds one email address where the reviews will all go to. The monolithic setup will therefore prohibit individual groups from easily being able to review code. For example the Marble group would probably like their review requests to go to a different mailinglist rather than the kde-edu mailinglist. This will not be possible.

When an application moves to a different module we will need to move the review requests to the new module. You can imagine that this won't be much fun to figure out. It will involve going through all review requests and see if they were filed against that application and move them over. The other option is to simply discard the review requests from the module the application is leaving.

We think for reviewboard it would be far easier if we chose the split-approach. Within the split approach, every project can have their own reviewboard entry and set the recipient for it.

But there is a catch within this split approach. We need to have unique names. This is called the namespace problem within the sysadmin team. Basically, we can not have 2 projects with the same name in two different modules. We'll get back to that later.

Redmine
If you have not read the document which the sysadmin team has published before, we suggest you do so now to get an idea what Redmine is.

Redmine can use hierarchies. That means we can create projects which are a sub-project of another project. This means the split vs monolithic approach will have less effect in this case.

In both cases we will create a 'kdepim' parent project and create child projects for each application. KMail, KOrganizer, KAlarm, etc. will all be children of the 'kdepim' project. All children are 'complete' projects in Redmine terms. There is no difference between a parent project and a child project in terms of functionality.

Within the monolithic approach we only have one repository. But we will have multiple projects for that one repository. For example, the kdepim parent project and all of its child projects - which are the applications. We need to decide to which project (the parent or the children) we attach that single repository.

By attaching we mean connecting a repository to a project so you can browse the sources from that project. We suggest to attach the repository to the parent project if we choose the monolithic approach. The child projects then have no repository attached, and should point to the parent project for browsing.

In the split approach we have a single repository and a single project. That way we can attach the repository of KMail directly to the KMail project page. That means new contributors to an application that want to browse through the code can do that with one click. In the monolithic approach, there is a referral to the parent project page, and when you arrive there, you should click through the KMail subdirectory. We find this a lot less user friendly for both the existing contributors of that project and for any other potential contributors.

In Redmine, we will also be faced with the namespacing problem. Each project needs to have a project name. The problem is that each project name has to be globally unique. For example, if there is a separate website project below Konversation called 'website', we can not have a project below Amarok also called 'website'. We would need to prepend it with the parents projects name. This name will also used in reviewboard, as that has no hierarchy either, so it should remain user readable.

Gitolite

Gitolite is the service where all the repositories are located. On the repository level the difference between the two approaches are the following. Within the monolithic approach there will be a kdepim.git repository in the root. For the split scenario there will be dozens more, one for each application.

For the split approach we considered rebuilding the hierarchy as we have it now, so a kdepim folder with a repository for each kdepim applications in there. This is a bad approach. It would mean that when an application moves from playground to kdereview, the git repository moves to a different folder. This causes all people to adjust their push paths and is something we should not do in our opinion.

We already see some of the confusion of moving repositories within SVN. When an app moves to kdereview and then in to the final destination, contributors need to checkout from new locations even though you know it will only last for two or three weeks. That's a pain that we can solve in the new setup.

We would like to propose keeping the repositories in a completely flat structure. This has the consequence that the project names again need to be globally unique. But it also means that we would clean up unused repositories over time. As all playground, extragear and the main modules are located in the same folder, we need to make sure that unmaintained repositories are removed, so they won't hinder the naming of new projects.

A possible solution to this is that we keep a list of maintainers for each repository. Every 3-6 months we ping each of them, asking if the repository should remain if no activity occurs. No answer or 'no' as an answer will move the repository away. Details of this should be further discussed and is a bit out of scope for this document at this point in time.

Another advantage is that this approach is the path of the least surprise. If you want to work on RSIBreak, you can simply clone git.kde.org/rsibreak.git. You do not have to find out in which folder it is located. Is it in extragear? Then in which folder of extragear is it? Is it in the main modules or in review? No, you just know where it is.

Splitting repositories also lowers the barrier to entry substantially for external contributors, as individual application repositories will be significantly smaller, consuming less disk space and bandwidth.

However, it would require that the applications are able to be built independently of their current top-level modules. We think we are at a point where we need to decide if we will let individual applications be compiled individually. We already see this happening however, see Marble and Kate. Playground and Extragear applications have always done this. This does have the benefit of clarifying dependencies however.

Whilst the transition takes place we recommend that the top level shell projects contain the needed build system files to build the module. In practice, this should work like this: you checkout the top level module, then checkout the applications you need into it. We hope that tools like kdesrc-build will abstract this further.

Final Advice

Based simply on the information above, we advise that a split repository layout be adopted. Each application will be located in it's own separate Git repository. The current structure on applications being in modules will be reflected in Redmine, and not in the architecture of Git repositories. All 3 applications (Redmine, Reviewboard, Gitolite) will use the same names to represent a repository.