Git workflow

In this project we make use of the "git flow" branching model. This means we branch in the same way for each of the projects.

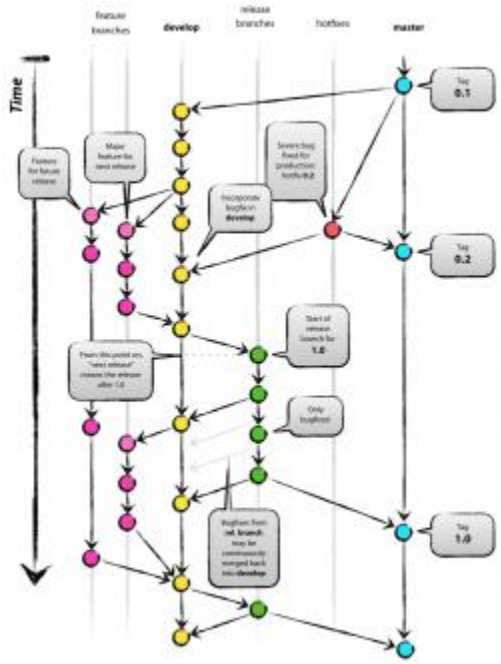

Figure 1 *Example of Git flow*

Master branches

In every project we have at least two branches. The "master" branch and the "development" branch.

Master branch

The branch "master" represents the current release of the module. The commits in the master always contain working, building, tested, inspected and validated code. It's only allowed to push to this branch after the client has approved the tests and inspections. The only way to push to this branch is by doing a merge from a release branch. The code must be put through a code inspection by then and all of the failures found must be corrected. Every commit to the master branch gets a version tag.

Development branch

The development branch represents the current state of development for the module. All the code in this branch is tested and inspected. This doesn't have to be a full code inspection but the code have to be inspected by two people. This means that the development always contains working and building code.

Support branches

Feature branch

To implement a new feature in the module we first create a feature branch. Also in this branch all the code is working and building. The code doesn't have to be inspected by other people. After the feature is done it could be merged with development. The requirements of the development must be met by then.

Release branches

Then there are release branches. Before a version in the development branch could be released it first has to branch of to the release branch. Then, on the first version of the release branch, a code inspection is performed. The failures come to light by the code inspection will be fixed in the same release branch. If all the failures are solved and the client gives his approval then the release branch is merged to the master.

Hotfix branches

If a major bug arises in the master branch we should repair that before the next release. To do this a hotfix branch is created. This branch branches from the master branch. The bug is then solved in this branch. Then the client inspects if the bug is fixed correctly and according to the regulations. If he gives his approval the branch may be merged back to the master.

--no-ff

With every merge the --no-ff parameter must be supplied:

```
git merge –no-ff $branch
```

This makes sure that no fast-forward is performed. This way the merge information is being preserved.

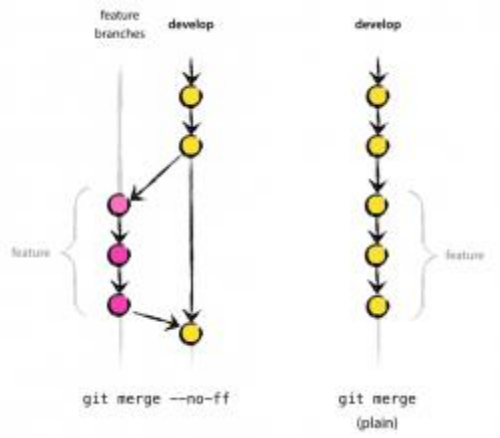

Figure 2 -no-ff vs fast forward

Versions

The version number consists of 3 different digits. The first one is the major version. This digit goes up by one if there is an major release of the module. If there is a partial delivery the second digit goes up. The third digit is for hotfixes. If the first digit change the other two are reset to zero. If the second digit changes, the third digit is set to zero.

So if we have the version 2.4.6. Now there has been 2 major versions. 4 partial deliveries and 6 hotfixes. If we release a new major version we get version 3.0.0 .

In short

Branches

Branch	Representatie	Branched off	Merged to	Naming convention	
Master	Release version	-	-	master	
Development	Development version	-	-	development	
Functionality	New funtionalities	Development	Development	funct-*	
Release	Pre-release	Development	Master, development	release-<<version>>	
Hotfix	Solving major bugs	Master	Master, development	hotfix-*	