

DevOps, Software Evolution and Software Maintenance

IT University of Copenhagen

Thomas Tyge Andersen
thta@itu.dk

Ask Harup Sejsbo
asse@itu.dk

Joakim Hinnerkov
jhhi@itu.dk

Petya Buchkova
pebu@itu.dk

Kasper Olsen
kols@itu.dk

May 4, 2021

Contents

1	System's Perspective	3
1.1	Design of your ITU-MiniTwit systems	3
1.2	Architecture of your ITU-MiniTwit systems	3
1.3	Important interactions of subsystems	3
2	Process' perspective	4
2.1	How do you interact as developers?	4
2.1.1	How is the team organized?	4
2.2	Stages and tools	4
2.3	Organization of repositories	4
2.4	Applied branching strategy	4
3	Lessons learned perspective	5
3.1	Evolution and refactoring	5
3.2	Operation	5
3.3	Maintenance	5

1 System's Perspective

Automatic CI build test

1.1 Design of your ITU-MiniTwit systems

1.2 Architecture of your ITU-MiniTwit systems

1.3 Important interactions of subsystems

2 Process' perspective

2.1 How do you interact as developers?

2.1.1 How is the team organized?

2.2 Stages and tools

2.3 Organization of repositories

2.4 Applied branching strategy

In week 2 we discussed what branching strategy we should use. We quickly narrowed the field of possible strategies to two - Git Flow and GitHub Flow, since these were the two strategies most people had experience with within the group.

Git Flow is a strategy where the main branch is used for released code e.g. stable releases. A development branch is merged out from the main branch, which is used to branch out feature branches where actual new work is done. New features are then merged back into the development branch. Once the team is ready to make a release, the development branch is merged into a release branch, then into main and released.

GitHub Flow is more simple than Git Flow since it does not have release and development branches. New features are instead merged out from the main branch, and back into the main branch once the feature is done. This will result in that the main branch containing code that is not yet released if deployment is not automatically set up.

We initially decided to go with Git Flow, since we would work on features throughout the week, and then create a single release on Sundays. We quickly realized that releasing once a week was not satisfactory for our needs, since we often wanted features to be pushed immediately once ready, and merging back and forth between the main branch and the development branch was cumbersome.

By week 7, we decided to change our branching strategy to GitHub Flow due to the addition of static code analysis and the rewrite of API simulation tests from Python to Golang. The motivation for the change in strategy is a move towards Continuous Deployment. Our pipelines now supported a full deployment process from our main branch to our production servers with automatic releases. This means that a Git Flow strategy will only hinder our lead time from user story to production.

3 Lessons learned perspective

3.1 Evolution and refactoring

3.2 Operation

3.3 Maintenance