CONTINUOUS INTEGRATION WITH **ENTERPRISE ARCHITECT**

INTERACTION OF LEMONTREE, GIT, PROCLOUDSERVER AND PROLABORATE

LieberLieber Software GmbH

INTRODUCTION

In the course of the increasingly agile orientation in software development, our customers ask us again and again whether modelling and agility go well together. Our opinion on this is quite clear and has also been proven in various studies: Only with properly applied modelling can agile processes be implemented at all in the face of increasing complexity, if you also want to document all regulations and requirements in a comprehensible way. As an example, we want to deal in this text with „Continuous Integration“, which has been state of the art in classical software development for some time. The aim is to continuously compile, test and merge the software product to be developed. In addition, the status of the development is published regularly: internally a stable development status, externally an official release of the software.

But how do you bring „Continuous Integration“ into model-based software development? The tool chain around Enterprise Architect (LemonTree, Git, Pro Cloud Server, Prolaborate) currently offers all the prerequisites to realise this scenario. From our experience, these possibilities are currently unique on the market and they open up completely new possibilities for our customers. Therefore, we want to present to you which procedures of classical software development can be adopted for „continuous modelling“ with this tool chain. It will also become clear that working with a central database (Prolaborate) and using LemonTree/Git are not contradictory, but can be used very well in parallel.

COLLABORATIVE MODELLING **WITH GIT**

With LemonTree by LieberLieber, it is already possible to adopt an established approach from softwaredevelopment to modelling, namely the versioning of model artefacts. For the daily work on the model,we recommend a version control approach with Git. Git enables parallel and distributed work on a model. First and foremost, one advantage of this approach is that each user accesses a local copy of the shared model. Therefore, no network connection to a shared model is necessary and there is no need for waiting times due to network latency. This makes it much easier to make changes quickly or to take a quick look at the model.

When a model is created, the „GitFlow“ workflow is ideal for collaboration. Here, development never takes place directly on the main „develop“ branch (yellow branch), but always in separate, delimited „feature branches“ (red branches).

Only when a feature has been modelled and tested finally, it is completed and brought back into the develop branch. The advantage of this approach is that changes can be made in isolation from other developments without having to constantly integrate new changes. Only when a feature is completed, the interim changes from develop become relevant and a conscious merge of the models can be carried out using LemonTree.

The release process and the final publication of the model are also supported by GitFlow. A „release branch“ is used to declare a status from develop as a release candidate. Here, final changes can be made and any errors can be corrected. Once a release candidate has been released, the status of the model is transferred to the main branch, the so-called „master“. There, the published status is given a name and frozen by means of a „tag“.

Various standards require the use of a corresponding configuration management and this refers to all elements - thus also to models. With the above approach, several features can be modelled in parallel and published states can be saved. It is also possible to compare historical states of the model with e.g. the current release to find out what has changed over time.

Graph

The graphic shows an example from the SmartGit tool. A release candidate „1.1“ was defined here. An unfinished feature for an architecture refactoring as well as a feature „Safety Control“ that has been finished in the meantime did not make it into the release.

When a merge operation is performed on a model, LemonTree attempts to merge the model automatically. This is not the case if a conflict exists on element level. This happens if, for example, the same property of the same element was changed differently in both versions. In order to detect the conflict case ahead of time and not only when integrating the feature, the special Automation Edition of LemonTree can be used.

CONTINUOUS INTEGRATION

When working in a feature branch, the changes that lead to a conflict with the develop branch are of interest. A build server, such as TeamCity or Jenkins, can be used to detect this automatically. Similar to software development, every commit and push in a repository triggers a new build:

Unlike software, however, a model cannot be compiled and checked for any syntax errors. An automated model comparison can be used to determine whether a feature branch conflicts with the develop branch. If this is the case, the build is aborted and an error message is issued:

The two branches (feature/newRequirement and develop) can now be compared with each other in Git:

Graph

It can also be seen that the „newRequirement“ feature has an older origin and that the develop branch has evolved significantly.

LemonTree now shows which changes lead to a conflict:

This allows the modeller to look ahead and resolve the conflict in his model before submitting a request for the completion of his feature (a so-called merge or pull request).

Up to this point, the model is only available in the version control repository. Often, however, it is necessary to make the model available in an automated way so that outsiders can use the model as a knowledge base.

MODEL-REVIEW WITH **PROLABORATE**

In order to operate Continuous Delivery / Deployment in addition to Continuous Integration, the build server structure already mentioned above can be used. However, in order for the model to remain accessible outside of version control, there needs to be an easy way to access models. A web browser that displays the content of a shared model repository is a good way to do this. To host the model outside version control, Sparx Systems‘ Pro Cloud Server is used.

Similar to the Continuous Integration described above, an automatism is also started here with each build. A project transfer transfers the local model from version control to a database repository, which serves as the data source for the Pro Cloud Server and Prolaborate. A distinction is made as to which branch is currently being built. There are separate cloud repositories for the develop and the master branch, which are shared with different users. Those users who regularly conduct reviews have access to the development model, i.e. the develop repository. Outsiders or other users of the model are only authorised to see finalised versions of the model and therefore only have access to the master repository.

To simplify access to the model, the web interfaces of the Pro Cloud Server can be used.

In addition to displaying the model in the browser, Prolaborate also offers dashboards and reviews. This allows intuitive and personalised dashboards to be designed. As soon as a user logs in, he or she lands directly on the dynamic dashboards designed especially for him or her. All the required architectural information is presented on a single page. For conducting reviews, Prolaborate enables an agile review process by assigning the right teams to conduct the detailed reviews at the right time. This allows the results from Enterprise Architect to be reviewed without the need to create additional documents.

SUMMARY

Finally, an overview of the entire „continuous modelling“ workflow is shown.

Responsible for the content

LieberLieber

LieberLieber Software GmbH
Handelskai 340, Top 5
1020 Vienna, Austria
+43 662 90600 2017

Philipp Kalenda
Senior Consultant

Dr. Konrad Wieland
CEO

Contact us at
welcome@lieberlieber.com