MASTERING VERSIONS AND VARIANTS

A UNIFIED APPROACH FOR COMPLEX SYSTEMS ENGINEERING

Versioning and variant handling are indispensable in the engineering of complex systems in order to ensure quality, efficiency and traceability and to effectively overcome the challenges of complexity and diversity. Unfortunately, in practice it is usually the case that either a good versioning concept is introduced or attempts are made to manage variants. LieberLieber has now succeeded in developing an effective solution for version and variant management.

LieberLieber Software GmbH

Berger et al. stated in the 2019 Dagstuhl Reports that variant handling is particularly important for software configuration, while software product line engineering requires support for versioning. However, it has not yet been possible to establish an overarching version and variant management system that has proven itself in practice. To solve this problem, LieberLieber proposes a combination of model versioning and variant management using the tools Enterprise Architect, LemonTree, Git and pure::variants from PTC. The workflow illustrated shows a solution using these components:

- Git Feature Branches
- Enterprise Architect model
- pure::variants for the transformation of variants

This description provides an initial insight into the joint handling of versions and variants of a model artefact. The combination of pure::variants and LemonTree also offers further approaches (integrated LemonTree Merge in pure::variants, LemonTree.Components as variants, etc.), which we will discuss in more detail in an upcoming white paper.

COMBINING VERSION AND VARIANT MANAGEMENT

Overall, versioning and variant handling are indispensable in the engineering of complex systems in order to ensure quality, efficiency and traceability and to effectively overcome the challenges of complexity and diversity.
While version control is essential for the long-term maintenance and further development of a system, variant management enables the further development or updating of existing versions of a system without jeopardising the integrity of other variants.
If, for example, a product line is developed on a common platform, it must be possible to document the different versions of the system in parallel and independently of each other. However, if a general problem is rectified on the platform, it is difficult to roll out this correction consistently to the variants that have already been further developed.

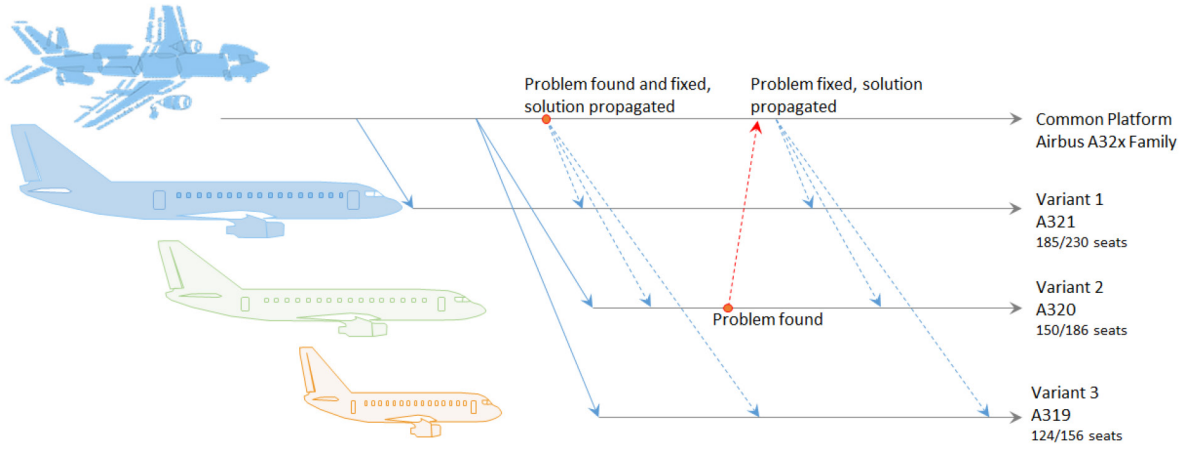

Figure 1: Variant management using the example of Airbus (Source: LieberLieber)

In practice, it has been shown that one either introduces a good versioning concept or tries to manage variants. Berger et.al. stated in the 2019 Dagstuhl Reports that the software configuration community is regularly confronted with the need to support variants, while software product line engineering needs support for versioning. They believe that neither community has succeeded in developing standardised techniques for version and variant management that are effective in practice.

FEATURE BRANCH APPROACH FOR MBSE

Inspired by the approach used in software development, LieberLieber has been promoting a 'feature branch' based approach for MBSE artefacts for several years.

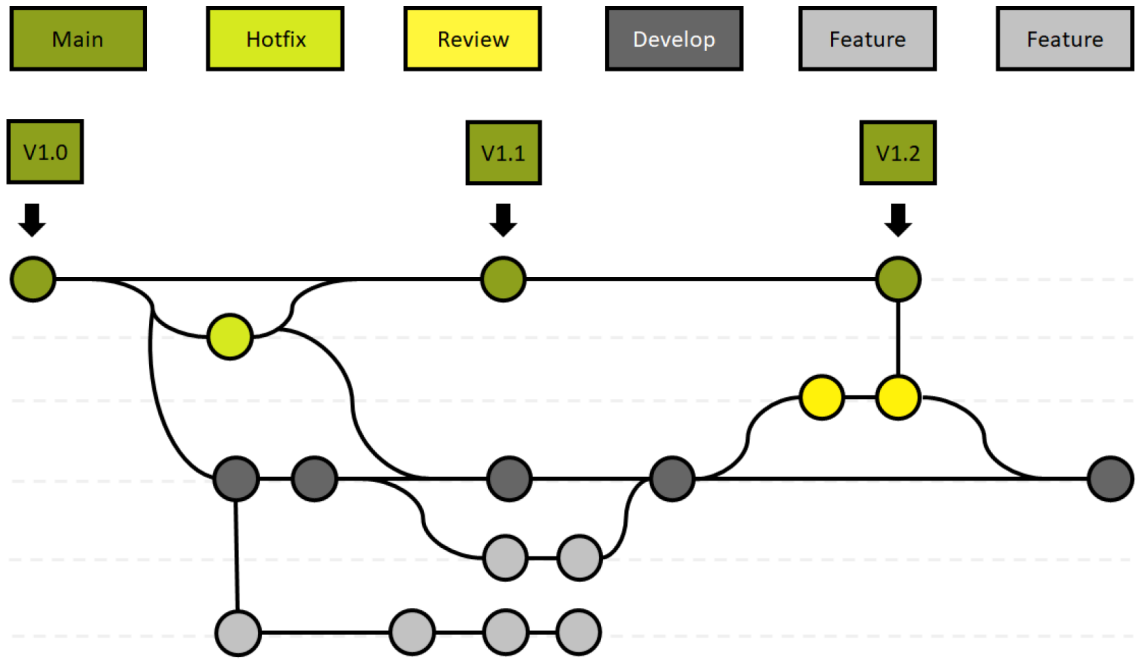

Figure 2: LieberLieber relies on a 'feature branch' based approach for MBSE artefacts (Source: https://atyantik.com/coding-smart-with-git-and-gitflow-a-tutorial-for-better-code-management/)

A model repository is treated similarly to source code and placed under version control, e.g. with Git. With the help of LemonTree, it is possible to compare and merge versions of the model, recognise conflicts and manage Git branches (Note 2).
To manage the variants, we recommend pure::variants from PTC, which specialises in the definition of feature models and the derivation (transformation) of variants. The aim is to combine feature modelling and feature-branch-based MBSE. In this way, variant management can be applied to an Enterprise Architect model. pure::variants transforms the complete system model into variant models, which are subsequently managed, compared and merged in Git with the help of LemonTree. The next section explains how a system model with the 150% approach is described with pure::variants and versioned via LemonTree and Git. The changes between variant models can be maintained with LemonTree and feature branches.

THE INTERACTION OF PURE::VARIANTS, LEMONTREE AND GIT

The central component of the workflow is a Git repository; the different variants can be mapped with branches. The initial model (150%) contains all variants in combination and is versioned in the main branch (main). With pure::variants, variant constraints can be added to Enterprise Architect elements, diagrams, connectors, etc.. These constraints are linked to the so-called feature model in pure::variants and define which artefacts from the Enterprise Architect belong to which variant. A pure::variants transformation creates a variant-specific (100%) Enterprise Architect model. The result of the transformation is added to a new branch and marked there with a Git tag. This tag serves as a reference for the time at which the variant was derived. In this way, it is possible to further develop all derived variant models (100%) and the base model (150%) independently of each other and in parallel. Managing the models in separate Git branches allows independent development. However, a so-called merge must be carried out for updates between the models. In the 3-way merge with LemonTree, three models are taken into account for the merge:

- Git tag of the last transformation
- Result of the current transformation
- Status of the current 100% model

The result of this merge is marked again as a Git tag. A selective merge with LemonTree can also be used to transfer changes between different 100% models or changes from a 100% model to the base (150%).

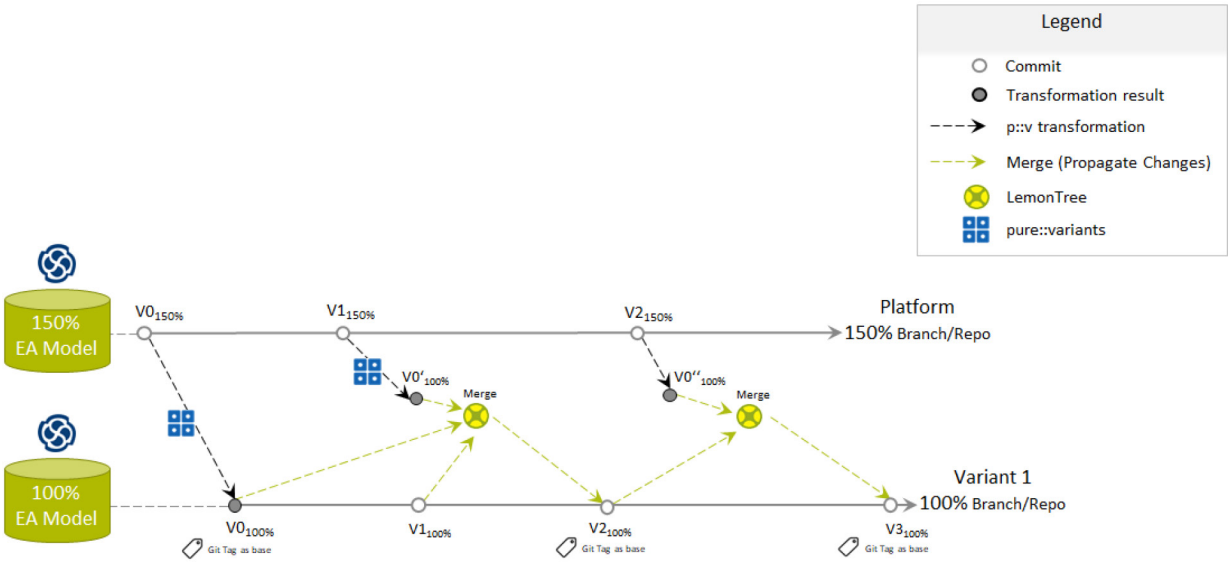

Figure 3: The interaction of pure::variant, LemonTree and Git (Source: LieberLieber & PTC)

SUMMARY

By managing models through branches, Git offers unlimited possibilities for merging changes between the different versions or variants of a model. The perfect interaction of the three tools presented here is crucial for success:

- LemonTree supports the merging of model elements in terms of content
- Git manages the branches and versions the model
- pure::variants provides an overview of the feature model with variant constraints and performs model transformations

The workflow shown above describes a working method proposed by LieberLieber that combines existing tools and practices. pure::variants and LemonTree can also be integrated even more seamlessly. An example of this is the direct merging of the Enterprise Architect model during the transformation or the use of LemonTree.Automation, LemonTree.Components and pure::variants as an instructor for the composition of submodels.

Notes:

1) Thorsten Berger, Marsha Chechik, Timo Kehrer, and Manuel Wimmer. Software Evolution in Time and Space: Unifying Version and Variability Management (Dagstuhl Seminar 19191). In Dagstuhl Reports, Volume 9, Issue 5, pp. 1-30, Schloss Dagstuhl – Leibniz-Zentrum für Informatik (2019)
2) https://customers.lieberlieber.com/downloads/WP_final_DE_Saubere-Versionierung.pdf

Responsible for the content

LieberLieber

LieberLieber Software GmbH
Gumpendorfer Straße 19
1060 Vienna, Austria
+43 662 90600 2017

Author
Philipp Kalenda
Head of Consultant

Editing
Rüdiger Maier
Public Relations

Contact us at
welcome@lieberlieber.com