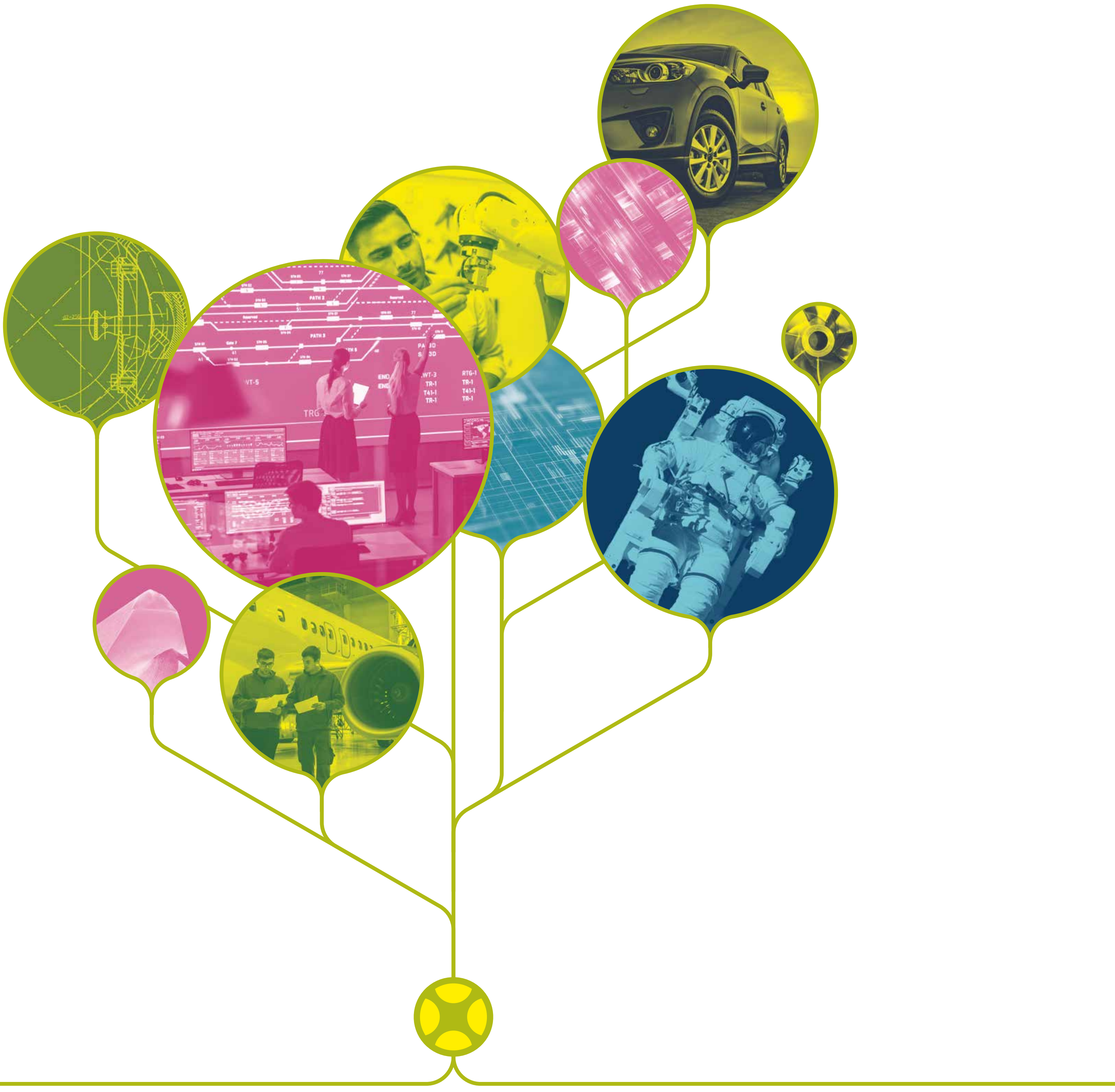


MODELLING COMPLEX CYBER-PHYSICAL SYSTEMS: **IS SUITABLE VERSIONING EVEN POSSIBLE?**

LieberLieber Software

MODELLING COMPLEX CYBER-PHYSICAL SYSTEMS: **IS SUITABLE VERSIONING EVEN POSSIBLE?**

In our everyday practice of model-based systems engineering in various branches of industry, we repeatedly encounter the question of precise and easy-to-handle versioning of models. This question is becoming increasingly important due to legal regulations, stricter safety regulations and the significant increase in product variants. Therefore, we are responding to a frequently expressed wish and answering the question in this white paper as to whether clean versioning is possible at all when modelling complex cyber-physical systems. To do this, we first look at what constitutes a complex system and why clean release is more difficult today when modelling such systems. Then we turn to practical matters by analysing the versioning process in more detail and describing the resulting requirements for tools. Finally, in the summary you will find the recommendations for achieving clean model versioning based on our decades of experience. So follow us on our journey into the land of unlimited variants and learn how to always keep them safely under control with the help of clean versioning.

WHAT IS A **COMPLEX SYSTEM?**

Modern systems in transport, healthcare, etc. are now cyber-physical systems. This means that they increasingly combine classic mechanical components with electronic and software components. Such a system is characterised by its high degree of complexity, which is growing rapidly, especially due to the spread of software. As a recent study (1) shows, companies are relying on the introduction of systems engineering to cope with this omnipresent complexity, which is now even seen as critical to the future viability of companies.

To visualise the dimension of the proliferation of complex cyber-physical systems, the transport sector is very suitable. Aircraft, for example, are large networks of on-board computers and embedded systems in many individual units. Without these embedded systems, not even the turbines could start. And even cars („PC on wheels") and trains are no longer functional today without on-board computers and embedded control systems. Reliability and safety of these systems are therefore of utmost importance. Networked systems, thousands of which work and communicate in a complex overall system, require a whole new quality of reliability and safety.

WHERE DOES THE COMPLEXITY BECOME VISIBLE IN EVERYDAY LIFE?

Of course, the electronic and software systems only become noticeable when they fail - for example, when the new car breaks down. Such failures already account for a high proportion of the warranty costs for new cars.

(1) Study 2021: Systems Engineering in Germany - A Comparison of the German Corporate Landscape - Prozesswerk in Cooperation with GfSE

WHY IS **CLEAN VERSIONING** NECESSARY?

With the proliferation of cyber-physical systems and rapidly increasing digitalisation, companies today are facing major changes. Mechanical and electrical requirements are increasingly taking a back seat, while software is taking over most of the value creation. However, the systems are also becoming more complex and challenges are arising that were previously unknown. A central solution approach for this is model-based systems engineering (MBSE). Here, information about a system to be developed is no longer based on documents, but on models. These models are usually created on the basis of the UML or SysML specification and help to better cope with central challenges such as safety, agile development processes and the work of distributed development teams.

VERSIONING MAKES IT POSSIBLE TO PROVE THAT ALL SAFETY REGULATIONS HAVE BEEN TAKEN INTO ACCOUNT IN THE MODEL

Functional safety is becoming increasingly important, especially in cyber-physical systems, and is mandatory in legal regulations. Put simply, safety refers to the protection of the environment from an object. Taking cars as an example, growing digitalisation with the goal of autonomous driving is currently leading to ever stricter specifications in the area of safety, so that cars do not cause damage to their environment despite software-assisted driving. This is not least a question of liability in the event of an accident or technical failure.

Since many versions and variants of models naturally arise in such an agile and complex environment, clear and secure versioning is particularly important. In order to strengthen trust in model-based systems engineering, it has to be clearly demonstrated that clean versioning is guaranteed and controllable at all times in the development process.

Precise versioning becomes particularly important when using agile methods, since new versions are created at short intervals and you still have to keep track of everything. Agile software development refers to approaches in the software development process that increase transparency and the speed of change and are intended to lead to faster use of the developed system. The design phase is shortened and attention is paid to achieving executable software as early as possible. This is regularly coordinated with the client in order to increase customer satisfaction.

Agile software development is characterised by self-organising teams and a step-by-step approach. In international companies, there is a clear trend towards virtual teams that work together across regional, national and cultural borders as well as time zones.

HOW CAN **CORRECT VERSIONING** BE REALISED?

Due to the challenges described above, it is important to follow a versioning strategy that guarantees control over individual versions and changes at all times.

The central element is the maintenance of a clear history of the development of a system that results from (parallel) changes. It must be possible to precisely identify and reference individual versions. This also makes it possible to select older versions of the system to be developed in order to use them as the basis for a new development or to create a variant of them. Projects are no longer developed from scratch, but often build on existing development efforts. Therefore, it must be possible to go back to any point in the development history to set up the new project from there.

Another important aspect is the personal allocation of development efforts: Which developer made which change and when. In this way, certain changes can be specifically removed from the project in order to be able to quickly undo them, for example, if necessary.

Both the increasing complexity and the ever shorter release cycles of modern software systems make it necessary to develop different model versions in parallel. Added to this is the challenge that these systems are being developed in ever larger and more distributed teams. In the course of a model-based approach, such teams must also be given the opportunity to work efficiently on parallel versions of a model. Last but not least, the integration of established versioning systems such as Git into the modelling process is indispensable.

Agile development processes are now widespread in classical software development and optimistic versioning processes have thus also become established. Based on these processes, methods/processes such as GitFlow, Continuous Integration/Continuous Development or DevOps became established. In this white paper, we want to analyse the GitFlow process in more detail and how it can also be used for the world of modelling.

The central idea here is to develop on the basis of feature branches. Gitflow is a git-branching model that uses long-lived feature branches and multiple primary branches. Developers create a feature branch and delay merging with the main trunk branch until the feature is complete. This way, Git-Flow automatically brings a unified structure to each project, which is especially beneficial when many developers are working on a project at the same time. Due to the parallel branches, several features can be developed, a release prepared and hotfixes carried out at the same time. This makes the model ideal for larger projects.

THE ADVANTAGES OF GITFLOW

The Gitflow workflow offers all the advantages of the feature branch model: pull requests, isolated experiments and more efficient collaboration. The workflow also uses a central repository as a communication hub for all developers.

REQUIREMENTS FOR TOOLS

For the practical implementation of clean versioning in the development of complex systems, suitable tools are needed. In this paragraph, we will show which requirements for these tools result from the circumstances described above.

PRECISE DETECTION AND MERGING OF CHANGES

In the development phase of complex systems, innumerable model versions naturally arise. Therefore, the most important function of a tool is the detailed comparison and easy merging of different versions. This requires a precise and fine-granular recognition of changes that have been made. Conventional approaches in software development use line- and text-based applications for this purpose, but they are not sufficient for graphical models. In addition, knowledge of the semantics of the modelling language is also important. A change report at database level or through XML files is neither user-friendly nor does it allow for a clean merge of changes.

USER-FRIENDLY REPRESENTATION OF CHANGES

In order to version well with a tool, it is not only important to make precise and complete changes at the model level, but also to present the changes in a way that is easy to understand. Model elements can be moved, changed, deleted or added. Moreover, such elements are often represented in several diagrams. In order to maintain an overview, the changes must be easily recognisable. Only in this way the developer is able to get a good picture of what is happening and also understand the changes made by other contributors.

CONFLICTS IN CHANGES AND THEIR INTERDEPENDENCIES

When several people make changes in models, conflicts can arise. Therefore, a tool must calculate these conflicts precisely. After all, a conflict should only be reported if it is not automatically resolved during the merge and human interaction is necessary. Then, with the help of the tool, it must be possible to easily merge these changes and thus eliminate the conflicts. In this process, it is the task of the tool to reliably ensure the validity of the model.

Since proven processes, workflows and tools are already used in practice, models must be able to interact with them. The modelling tool must therefore enable seamless integration into existing versioning systems (SVN, Git, PTC, etc.) and industrial application lifecycle management (ALM) tools.

EDITING PARTIAL MODELS

Today, entire supplier chains are involved in the development process of complex systems, each of which contributes only a part to the overall system. In classic „product line development“, it is therefore inevitable that parts of models can also be processed by different suppliers with one tool and then reintegrated into the overall system. This process is further complicated by countless product variants and the trend towards so-called „platform strategies“ (e.g. one floor assembly for different vehicle types).

This means that it must be ensured that different elements and parts of the overall system can be worked on by different development teams at different times and in different places and ultimately be integrated back into the overall system. Thus, an overall system is actually a system of many subsystems. In this scenario, a tool must ensure that there are no system conflicts or that information is not lost either when model parts are detached or reintegrated.

HERMANN GOLLWITZER, WORKING AS A SYSTEM ARCHITECT FOR INFOTAINMENT SYSTEMS AT VW:

„With LieberLieber's active support, we succeeded in making model-based systems engineering (MBSE) the backbone of our development organisation. The control of all versions and variants of the models created in the development process was for us the most important requirement for the stable anchoring of MBSE in the company. LieberLieber supported us very well with profound practical knowledge as well as with tools such as Enterprise Architect and LemonTree and thus promoted the internal work of persuasion.“

SUMMARY AND **RECOMMENDATIONS**

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, in this white paper we have dealt with the question of whether clean versioning is possible at all in complex systems.

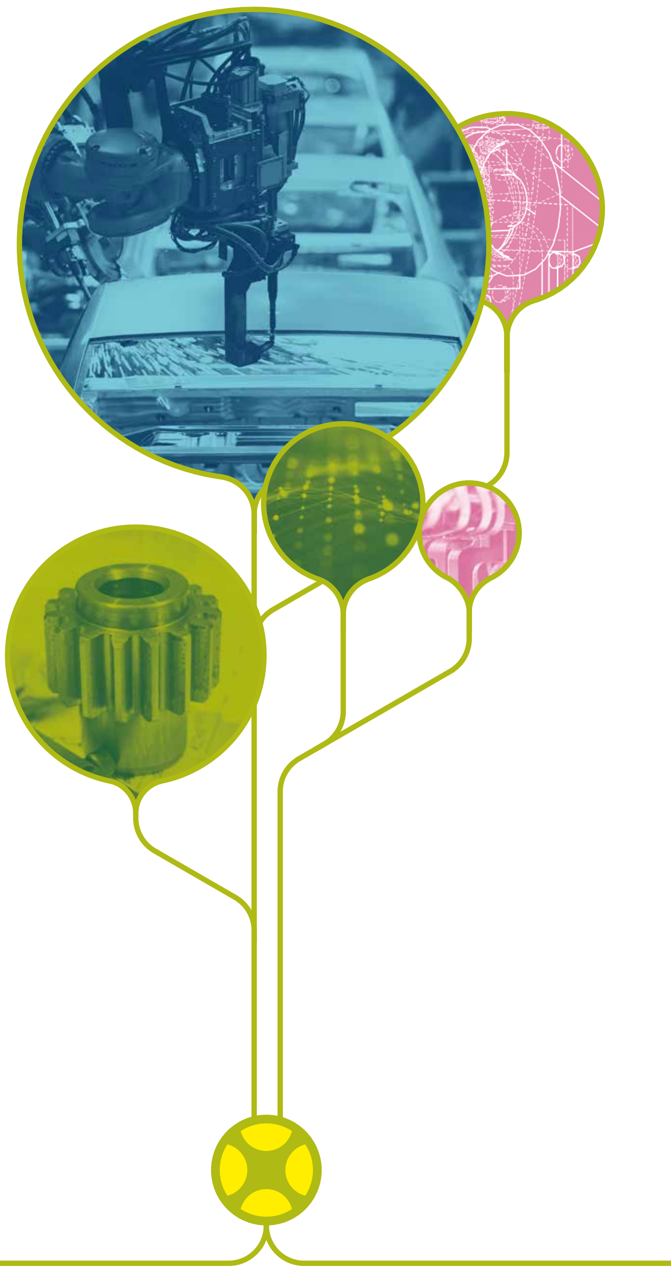
Our answer to this is clearly yes, but it requires corresponding experience and suitable tools. According to our decades of experience, the tool chain we recommend for implementation around Enterprise Architect, LemonTree and Git is currently the best prerequisite for realising clean versioning at the cutting edge of technology. In our opinion, the potential of this tool chain is currently unique on the market and opens up completely new possibilities for our customers.

LEMONTREE: THE RIGHT TOOL FOR CLEAN VERSIONING

LemonTree has been specifically designed to meet the above requirements for tools:

- Accurate change detection and merging
- User-friendly representation of changes
- Precise handling of conflicts in changes and their interdependencies
- Editing of partial models
- Reuse of proven processes (Git, DevOps etc.)

The central task of LemonTree is to support team-based collaboration in today‘s complex projects in the best possible way, and it is constantly being expanded with new releases. For example, a partial model (or the entire model) can still be managed as EAP(x) in versioning systems such as Git, which effectively supports the work of distributed teams. If a component of the model has evolved during parallel editing, it can be easily re-imported thanks to the intelligent merge function. The dependency analysis integrated in LemonTree is used to define model parts more precisely before export. Mutual dependencies can be examined or eliminated if necessary. Simple dependencies and cyclical dependencies between model packages are clearly shown.



Responsible for the content

LieberLieber

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

The Authors

Dipl.-Ing. Rüdiger Maier, M.A.
Head of PR

Dr. Konrad Wieland
CEO

Please contact us under
welcome@lieberlieber.com