# Chapter 15
# The Past, Present, and Future of Model Versioning

**Petra Brosch**
*Vienna University of Technology, Austria*

**Philip Langer**
*Johannes Kepler University Linz, Austria*

**Martina Seidl**
*Johannes Kepler University Linz, Austria*

**Konrad Wieland**
*Vienna University of Technology, Austria*

**Manuel Wimmer**
*Vienna University of Technology, Austria*

**Gerti Kappel**
*Vienna University of Technology, Austria*

## ABSTRACT

*The evolution of software models induces a plethora of challenging research issues. Only when these problems are solved, the techniques of model-driven engineering (MDE) are able to fully exploit their potential in practice. Otherwise the advantages of MDE are relativized by time-consuming and cumbersome management tasks which are already well supported for traditional development based on textual code. One of these challenges is model versioning.*

*Version Control Systems (VCS) are an essential part of the software development infrastructure which (i) store the history of evolution of software artifacts, (ii) support multiple developers working in parallel, and (iii) manage different development branches. For all of these tasks, changes performed on the artifacts under version control have to be tracked. For the second and third task it is additionally necessary to detect conflicts between concurrently evolved versions of one artifact and to resolve such conflicts in order to obtain a consolidated version.*

DOI: 10.4018/978-1-61350-438-3.ch015

*Compared to code versioning, which works well in practice, model versioning is still in its infancy as the established approaches for code versioning may be hardly reused. However, several dedicated approaches for model versioning have been proposed. In this chapter, we review the active research field of model versioning, establish a common terminology, introduce the various techniques and technologies applied in state-of-the-art versioning systems, and conclude with open issues and challenges which have to be overcome for putting model versioning into practice.*

## INTRODUCTION

During the software development lifecycle, the various software artifacts under construction are subject to successive changes. Consequently, tool support for managing the evolution of these artifacts is indispensable (Estublier et al., 2005; and Mens, 2008). To this end, the discipline of Software Configuration Management (SCM) provides tools and techniques for making evolution manageable (Tichy, 1988). Amongst others, these tools include Version Control Systems (VCS) whose origins may be dated back to the early 70s. Since then, the discipline of versioning is an active research topic generating a variety of different concepts, formalisms, and technologies.

The aims of versioning approaches are threefold. First, versioning systems maintain a historical archive of the different versions an artifact adopts during its development. With this archive, it is possible to undo harmful modifications by restoring previous development states. Second, versioning systems support handling different development branches, e.g., for building different software variants. Third, versioning approaches manage the parallel evolution of software artifacts performed by a (distributed) team of developers. In this book chapter we focus on the latter aim.

In general, two different versioning strategies exist to cope with the concurrent evolution of one artifact. When *pessimistic versioning* is applied, an artifact is locked while it is changed by one developer. Since other developers cannot perform any changes while the artifact is locked, conflicts are completely avoided with this strategy. However, the drawbacks are possible idle times for developers waiting for the release of a locked artifact. To avoid such idle times, *optimistic versioning* allows the developers to change the same artifact in parallel and independently of each other. The typical workflow of optimistic versioning is depicted in Figure 1. Two users of the optimistic versioning system, Harry and Sally, check out the same artifact at time *t0*. Both modify the checked out Version 0 independently of each other. After Sally has finished, she checks in her modified version (Version 1) at *t1*. When Harry also tries to check in his modified version, he first has to *merge* his version (Version 2) with the latest version in the repository (Version 1). Merging, often a time-consuming and tedious task, is the price to pay, when concurrent modifications by several users are allowed. In general, the merge process may be divided into four steps: (i) identifying the differences between two concurrently modified versions, (ii) detecting conflicts between these two modifications, (iii) resolving these conflicts either automatically or manually, and finally (iv) creating a new consolidated version which, in the best case, combines all intentions behind all concurrently performed modifications.

From a technical point of view, a wealth of works have been published which contribute in solving various versioning challenges (cf. Conradi & Westfechtel, 1998 for a survey), but how versioning really works in practice when huge software applications are developed is hardly discussed. To fill this gap, in 2010 we have conducted an online survey on best practices in versioning, which has been answered by approximately 100 software engineers, software architects, and IT managers. We wanted to learn about their habits