# Towards an Understanding of Requirements for Model Versioning Support

*Konrad Wieland, Vienna University of Technology, Austria*

*Geraldine Fitzpatrick, Vienna University of Technology, Austria*

*Gerti Kappel, Vienna University of Technology, Austria*

*Martina Seidl, Vienna University of Technology & Johannes Kepler University Linz, Austria*

*Manuel Wimmer, Vienna University of Technology, Austria*

## ABSTRACT

*When software is developed in teams – the standard way software is developed today – versioning systems are the first choice for the management of collaboration. From a technical point of view, versioning systems have to face several challenges. Depending on the applied versioning paradigm, functionalities such as synchronous editing, branching, storing different versions, merging, etc. are required. Since much effort has been spent into realizing these tasks, measurable progress has been achieved over the last decades. Unfortunately, there is a lack of empirical studies to find out the actual requirements arising from practice. Therefore, the authors conducted an online survey and interviewed representative users of versioning systems from academia and industry. Special emphasis is placed on the versioning of software models, which are nowadays becoming more and more important as there is a trend to model-driven software engineering. The results of our empirical studies show that not all requirements of developers are satisfied by current versioning systems. Especially, more emphasis needs to be put on the management of collaborative development, e.g., the division of work and the management of conflicts.*

*Keywords:* *Collaborative Software Development, Empirical Study, Model Versioning, Models, Optimistic Versioning, Software Models*

## 1. INTRODUCTION

Software engineering, as any other engineering discipline, must provide the ability and means to build systems which are so large and complex that they have to be built by teams or even by teams of teams of engineers (Ghezzi et al., 2002). Today, not only code but also other artifacts like models play an important role in software engineering. All of these artifacts entail significant overhead to manage throughout the lifecycle, especially when practical concerns require parallel strands of development.

DOI: 10.4018/ijpop.2011070101

For this purpose, Software Configuration Management (SCM) provides key tools and techniques for making the parallel development of software systems more manageable (Tichy, 1988). Amongst others, SCM offers Version Control Systems (VCS), which allow reusing single-user modeling/programming environments for parallel development. Central repositories, to which developers can commit their changes and from which developers can update their local version to the latest version in the repository, support the management and administration of software artifacts under development such as code and models.

Of course, this holds true not only for traditional, code-centric software engineering, but also for model-driven software engineering (MDSE) (cf. Schmidt, 2006), which has recently gained momentum in academia as well as in practice, changing the way in which modern software systems are built. In MDSE, the task of programming, i.e., writing code in a textual programming language such as Java, is replaced by modeling in a graphical modeling language such as the Unified Modeling Language (OMG, 2010). The powerful abstraction mechanisms of models are not only used for documentation purposes, but also for compiling executable code directly out of models (Bézivin, 2005).

Software artifacts, code and models differ in key ways with implications for versioning and conflict management. In general, standard techniques established for text-based artifacts for handling software evolution like versioning perform poorly if directly used for models (cf. Chawate et al., 1996). From a technical point of view, these incompatibilities might be explained by the graph-based structure of models, which might be taken into account by dedicated algorithms for matching, comparing, and merging models. While there has been considerable work to understand and support conflicting approaches with *code* artifacts, the implications and issues when using *model* artifacts are less well understood.

Model versioning is still a young research area compared to code versioning (cf. Brosch et al., 2011a). In this heterogeneous field, a plethora of research directions exist trying to meet the technical challenges of model versioning systems, mostly concerned with precise conflict detection and supportive conflict resolution (cf. Altmanninger et al., 2009b). However, there is currently a lack of empirical studies trying to derive the "real" needs of software developers in practice concerning the collaborative development of software systems (cf. Mens, 2002). The current need for such studies is great, because several possibilities exist for how software can be developed collaboratively, both on the technical level as well as on the organizational level. The latter aspect is often ignored, especially in the model versioning research field. If we are able to understand real world experiences with model versioning then we would be better able to identify criteria which should determine the selection of versioning technologies as well as collaboration processes from an organizational viewpoint. Furthermore, lessons learned of current best practices in collaborative software development for the various development artifacts may be inferred from such studies. To the best of our knowledge, only a few investigations have been carried out in order to find answers to these questions. For example, several issues arising from practice when merging different versions of a model are identified in Bendix and Emanuelsson (2009). However, these findings are based on informal interviews within one company and do "*not pretend to be general(ly)*" applicable. Furthermore, the premise of Bendix and Emanuelsson (2009) is that "*model-centric development and its problems do not vary much from company to company"* which has not been proven so far. However, models can be used in different ways, namely as a sketch to discuss ideas and design alternatives, as a blueprint for implementation, or for direct code generation (cf. Fowler, 2003) and, thus, the collaborative development of models varies from company to company.

To tackle these mentioned deficiencies, this paper provides a comprehensive empirical study, including on the one hand a survey and on the other hand in-depth qualitative interviews. The overall goal of the empirical study is to gain