Is Our Merging Right? Towards More Reliable Merging Decision

Ahmed Shah Mashiyat
Department of Computer Science
University of Toronto, Canada
mashiyat@cs.toronto.edu

ABSTRACT

In the distributed and collaborative development environment, where a optimistic version controlling is used, software developer inevitably have to manually resolve the conflict while merging. There are a number of tools (Git, SVN, CVS, AMOR, IBM Jazz, etc.) that support automatic merging for two non-conflicting segment of two versions of a program or a model. However, these tools have very little to offer for resolving the conflict of the programs or the models, and the conflict resolution is largely dependent on the manual intervention. Often, a developer may not have sufficient knowledge about the conflicting blocks that he is merging, and then he makes guesses based on his experience and some knowledge on the problem domain. These illiterate guesses, if wrong, may cause unnecessary reworks: identify the merging problem, consult with team-mates to resolve the conflict correctly, redeploy, re-regression, testing, etc. The consequence is more catastrophic if the merging is wrong and that is not identified before deployment. In this project, we explore the state of the art conflict resolution tools for model versioning and identify a problem, outline a model of provenance information semantic, and propose a framework to potentially help the developer to take better merging decisions.

Categories and Subject Descriptors

D.2.7 [**Software Engineering**]: [Distribution, Maintenance, and Enhancement—Version Control]; K.6.3 [**Management of Computing and Information Systems**]: [Software Management—software selection]

General Terms

Management, Design

Keywords

Version Control, Model Management, Conflict Resolusion, Interface Design

1. INTRODUCTION

Software development is a collaborative effort. Software models are the most important citizen of model driven software engineering. Albeit the fact that same models can be developed concurrently among several developers (represent modelers as well), tool support for version controlling of software models is limited. There are some promising tools for model versioning such as AMOR, IBM Jazz, EMF store. However, when it comes to the problem of conflict resolution between two merging blocks, the tools depend on the developers knowledge of the problem domain and that particular code block. Figure 1 shows a typical conflict resolution activity while merging software models.

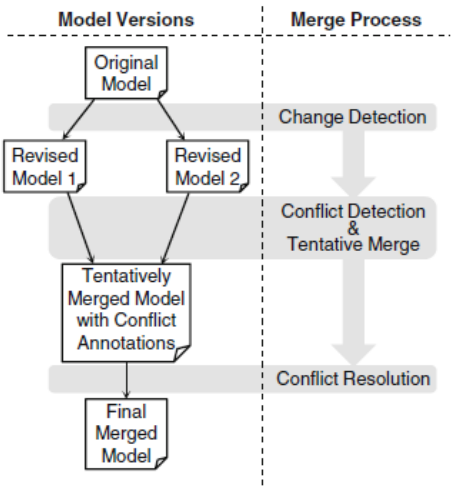


Figure 1: A sample merging process [15]

The model versioning tools, mentioned above, provides different merging options when there is a potential conflict. They work well when the conflicted block has minor changes, but ultimately the developer has to take the final decision. However, in practice the conflicting blocks contain a series of modifications and the tool support in these cases are very limited. Again, it comes down to the developers to merge the models and take the final decision. To this end, we want to empower the developers with better information so that they can take better decision while merging conflicting blocks. We proposed to leverage the software development provenance information as a guide to merge during conflict resolution.

In large software development life cycle, a developer who

is performing a merging tasks often asks: who made these changes?, why and when this changes has been done?, or how the code/model evolved to this point? These are among the most frequently reported question asked about software history [9]. Code versioning tools like CVS' "annotate" (or "blame") or Eclipse's "show annotations" answer the questions related to who and when. Tools such as Hipikat and Deep Intellisense tries to answer the questions related to why. The rationale is often found using the artifacts (e.g., bugs, e-mails, or documents) that appear relevant to a piece of code [1]. The model versioning tools (AMOR, IBM Jazz, and EMF Store) did not attempt to answer the questions related to why and how. To answer such questions, software model provenance should be tracked at each point of its life cycle. Based on the project life span, team size, project size, etc. the volume of the accumulated data is much larger then the original model or code itself and may be considered a small "Big Data" Problem. To make the data useful to the developer, its very important that the analysis or question result is conceivable by the developers. We want leverage some visualization techniques to facilitate this cognitive exercise and measure their effectiveness. Again, extracting the provenance information from an existing system is very difficult problem [14]. To this end, we outlined a framework for model versioning so that we can harvest the provenance information during the model life cycle, and can be easily extracted when needed. To summarize, our contributions through this project are as follows:

- Investigate and analyse the state of the art model versioning tools and identify a need which is not fulfilled by the current model/code versioning tools.
- Outline a model of provenance semantic for Model versioning.
- Propose a framework which can potentially fulfil the need.

The report is structured as follows. In Section 2, we discuss a motivating example, that we will use in this report. In Section 3, we discuss three state of the art model versioning tools. In Section 4, we discuss an underlying semantics of model versioning provenance. The architectural aspects of the proposed framework, implementation and evaluation are described in Section 5. In Section 6, we report on some related work. Finally, in Section 7, we discuss the current status of the project, directions for the future, and conclusion.

2. MOTIVATING EXAMPLE

Optimistic version control systems allow developers to work in parallel with an increased possibility of conflict during code merging. Three types of conflicts are identified in [10]: syntactic, structural, and semantic. To better explain the problem and our approach we have used a simple UML ER diagram as an example with syntactic and structural conflict. Lets assume developer Bob was asked to update their company's transport system model. He started with adding drivers, vehicle and checked-in the model. Molly was asked to refine the classes that are available in the model. While Molly was refining the model, the company decided to outsource its transportation system and asked Rob to reflect that in the model. Rob made the changes quickly and checked-in the model. Molly did not know about this requirement and when she wanted to check-in there is merge conflict notification and she was not sure what to do. Figure 2 depicts the scenario.

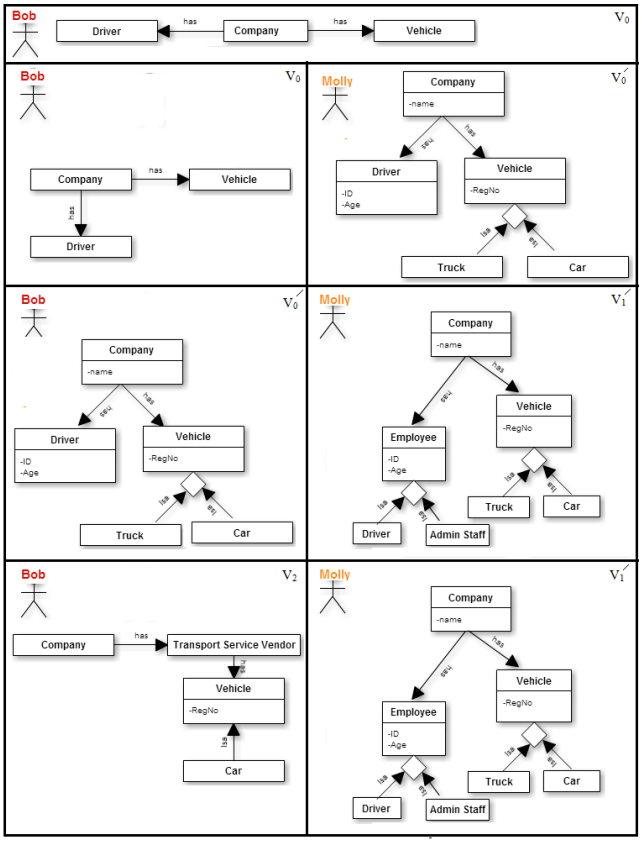


Figure 2: A motivating example (imaginary)

Molly will not be able to check-in the code unless she resolved the conflict between V_2 and V_1'. The state of the art model version control systems can detect the conflicts but their suggestion does not encompass the resolution. To resolve the conflict its important to know why and how the model came to this point in their life cycle.

3. MODEL VERSIONING TOOLS AND MERGE SUPPORT

In [2], Brosch did an comprehensive study on various model version control systems and categorize them. She described the overall version control systems in four categories (shown in Figure 3). One dimension of the categorization is based on *text* or *graph* based artifact representation. The other dimension is based on how differences are identified and merged in order to create a consolidated version. A versioned element might be a line in a flat text file, a node in a graph, or whatsoever constitutes the representation used for merging [2]. We looked at the following tools more closely.

AMOR [4] [2] is an adaptable model versioning system. AMOR [1] can detect conflicts in Ecore based models, resulting from both atomic and composite changes. It has

[1]http://www.modelversioning.org/

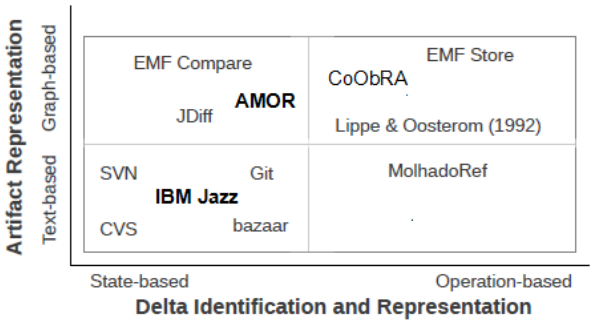


Figure 3: Version control systems categorization based on [2]

a recommender system for immediate conflict resolution by providing a resolution pattern, which can be automatically executed. It also has a mechanism that can defer a conflict resolution and store a modeler's annotation about the conflict in the model. Overall, AMOR project has three research goals: precise conflict detection, intelligent conflict resolution, and adaptable versioning framework.

EMFStore is an eclipse based model repository for EMF model instances. EMF Store [2] allows to checkout a copy of a model instance from the repository. Then it tracks change on the model instances on the clients and provides an API to send the changes to the repository. The API allows updating the model instances according to changes of other clients via the repository. With this operation-based approach, it is possible to efficiently and precisely detect composite changes. However, composite operations like refactorings are only detectable if they are explicitly available within the modeling editor [2]. EMF Store supports interactive model merging to resolve conflicts if two clients changes the same data in a model instance. It supplies interactive user interfaces to support model merging. EMF store has extension points to incorporate new conflict detection and merging algorithms.

IBM Rational Software Architect (RSA), which is based on IBM Jazz [3], is a UML modeling environment built upon the Eclipse Modeling Framework. It provides two-way and three-way merge functionality for UML models. RSA considers syntax and semantics of both EMF and UML model while merging. The differences between two conflicting models are shown either in a tree-editor, or directly in the diagram. Conflict resolution is manually done by the modeler, by either rejecting or accepting changes. Furthermore, the RSA offers a model validation facility which checks the conformance of the merged version to the UML metamodel [2].

There are a whole array for model versioning tool with different functionalities. A more rigorous categorization and analysis can be found in [3].

4. PROVENANCE SEMANTICS FOR MODEL VERSIONING

[2]http://www.eclipse.org/proposals/emf-store/
[3]https://jazz.net/

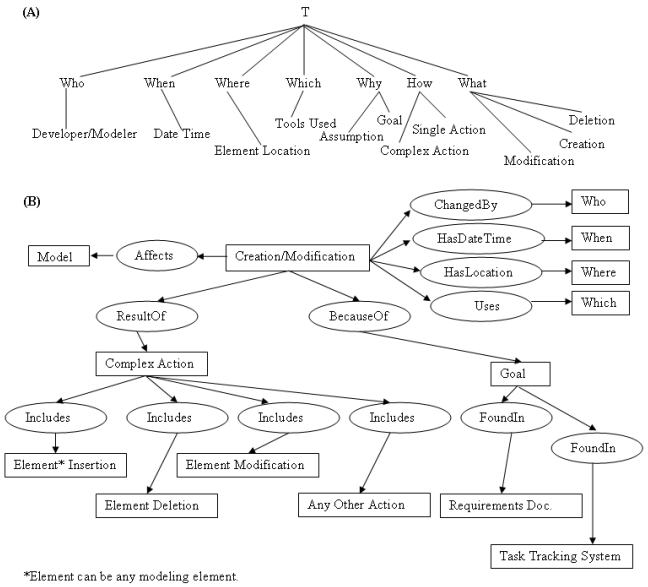


Figure 4: Provenance semantics for model versioning

Provenance information describes the origins and the history of data in its life cycle. Such information (also called lineage) is important to many data management tasks [8]. As Model Driven Engineering (MDE) approach is not yet used by mass population, the need for an industrial strength model version control system is yet to visible. However, its also pertinent to say that the wide acceptance of MDE is also depended on these tool support to hit the mass adoption. To this end, this is a high time to develop a version controlling system with a good underlying foundation, so that future needs can be fulfilled. With that in mind, we urge to store model provenance information in a fine granular level, because at this point, we are not sure exactly when and in what future purpose this information can be used. Since provenance information grow over time, an under lying semantic for model version provenance would help to track and analyzes the information better in future. We adopted Bunge's ontology [5] as the same way Ram et al. adopted in [14]. Bunge's theory sees the history as a sequence of events that occurred to a thing in its lifespan. Based on this theory, we define model provenance as consisting of various events (what) that happen to that model over its lifespan (from creation to destruction) and then include how (the actions performed for the event), who (persons responsible for the event), when (the date and time for that event), where (location of the event), which (means of the event) and why (rationale behind the event) associated with each event. Figure 4 shows semantics of provenance for a model versioning domain. Figure 4(A) shows the providence information that we are interested in and want to harvest for future. Figure 4(B) shows an event graph for model life cycle from creation, modification, and destruction. A model evolve as a result of a series of actions performed upon it. These provenance information can be stored in any physically data store such as a relational or noSQL database, XML, or RDF.

5. PROPOSED FRAMEWORK

In a large scale software development environment, having multiple teams working in different stream (branching) is very common. Often, these teams modify some shared code to fulfil their needs. However, in the integration phase (merging) these changes frequently result in merge conflict requiring to understood the code. In a recent study [12] it has been reported that fixing these merge conflicts are very tedious and poses a crucial bottleneck for developers productivity. The merge conflict has to be done manually and there is not much tool support for exploring the rationale for different code snippet. Exploring appropriate information while conflict resolution is an obvious need [18]. Although, there are some work done for exploring code history [1], we have not found any framework that support the information needs while merging two model developed in parallel. Historically, MDE approach is used for large scale software development and parallel model development may not be very uncommon. (At this point, I don't have any proof for this statement, but it is a general guess). In this section, we will outline a framework that will make use of the provenance semantics described in previous section and will be able to answer developers question while conflict resolution.

5.1 Architecture

Figure 5 shows the architecture of proposed model version control system with the capability of exploring model provenance information. We assume that the system can be plug-able to any EMF based model management or model designing tools such as MMTF [16]. In the proposed framework, we have used a decentralized version control system (DVCS) as opposed to a centralized version control system (CVCS). Although CVCSs such SVN, CSV, and Subversion are the most commonly-used version control system, DVCS such as git, mercurial, bzr, and bitkeeper have emerged addressing some of the limitations of current CVCS to better support decentralized workflows. For this reason many open source and closed source projects are proposing to move, or have already moved their source code repositories to a DVCS [6]. In our proposed approach, based on the principal of DVCS, each developer have their own local repository and they can add/commit to this repository any time they want. There is a central repository which is maintained by a designated integrator. When a developer pushes his changes to the main repository, the integrator get a copy of the change and merge into the main branch. Is this the time when conflicts are visible and need extensive human intervention. In the proposed framework, the integrator is provided the provenance information when he is resolving any conflicts. The following section describe the summarization and visualization of the provenance information for conflicting model elements.

5.2 Prototype Implementation

To demonstrate the possible effectiveness, we have implemented an ad-hoc system to summarize and visualize the provenance information. Although our proposed framework outlined a version control system that will harvest information over the life cycle of software development, in our ad-hoc implementation we have used EMF store as an version control system on top of MMTF. We have used MEAD summarizer to get a small spinet of related information from a large document. We are trying to use Mylan as a source of documented tasks. After a developer made a change in the model and commit to the repository, we query, based on the commit message, the project documents and tasks to get some relevant information. Often, developer mention the task IDs in their commit, in that case we can directly query to those tasks and get some related information. Developer have a tendency of not providing enough information during their commits. For this reason, we are querying the tasks or requirement documents to provide more rationale for the changes made. We have employed similar technique for visualization as found in [1] with some additional information (only mock up done).

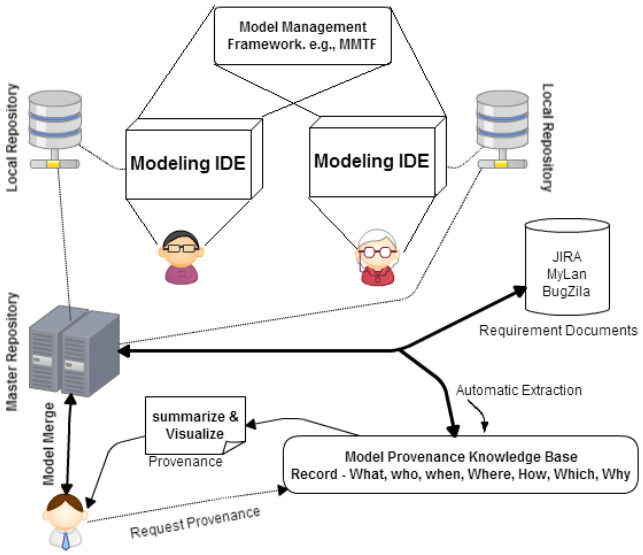


Figure 5: Architecture of the proposed version control system

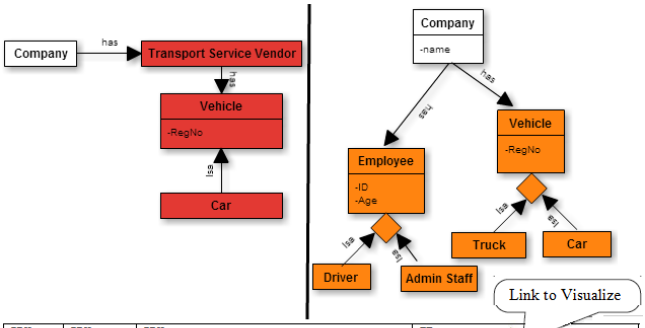
5.2.1 *Summarization*

For a merge job, it's highly unproductive to try to find out the rationale behind some developed part of a model. In current manual system, the integrator have to browse through all the commit messages, opens any related requirement documents and reads/skims through it to find a rationale behind some part of a model development. To make this process faster and efficient we have used a summarizer to provide a quick overview of a task or requirement. There are a number of text summarization tools available, such as Open Text Summarizer (multiple contributor), MEAD (University of Michigan), SweSum (kth, sweden), FociSum (CS, Columbia), SUMMARIST(ISI, Southern California), etc. Among those we choose the MEAD [4] because of its wide range of support. It has multiple summarization algorithms implemented such as position-based, centroid-based, largest common subsequence, and keywords. We are interested on the query-based summaries, where we will pass some commit message keywords and based on that the summarizer will provide us a concise task summary. Here we have not actually implemented any traceability technique, our summarization and task linking is very ad-hoc. Tracking back to the requirement document from a model element could potentially be a good project as there is a lot of work done in traceability [7, 11].

[4]http://www.summarization.com/mead/

5.2.2 *Visualization*

Our approach to resolve conflict is to educate the integrator with more related information, therefore, we are aiming to potentially explore a large amount of historical information. The representation visuals have a deep impact (both on productivity and correctness) on how the integrator conceive provenance information. In Figure 6, we have outlined a visual representation for a conflict resolution scenario given in section 2.



Who	When	Why	How
Bob	10 Dec, 2012	Msg: Change requirement in Task 107. **Summary**: Due to underutilization of resources, we need to outsource our transformation service.	Deleted element (MID: 12, name='driver'), Deleted element (MID: 12, name='has'), Added element (MID: 35, name=' Transport Service Vendor'), etc.

Figure 6: Mock-up user interface for the system

5.3 Evaluation

We have not conducted any scientific evaluation of our proposed method yet. However, from our experience on merging, we think that this approach could be very useful for the integrator performing conflict resolution. However, we have some outline how we will evaluate the approach. There are three key metric that we want to measure: How fast the merging is done, Precision, and Recall. We plan to conduct an on-line survey with the following possible experiments.

- provide a set of conflicting models with some possible options (Generated from AMOR) along with option of "none of them" and the correct one (if correct one is not present in AMOR's suggestions).
- provide the same set of models with the same set of options along with some provenance information.
- Measure the time required, Precision, and Recall.

This will provide us a comparative study of Precision, and Recall with AMOR. However, it is very subjective to measure the productivity improvement because that will require some developer actually try to merge two models, try to find out the rationale behind their choices manually, and measure the time. Again, the same merging task can be given to another set of developers (how do we know they have same efficiency for merging? same experience? same education?) with provenance information and measure the time required.

6. RELATED WORK

we have not found any work that approach the conflict resolution by exploring historical information of model repository as a visual aiding tool. However, exploration of software provenance and traceability information between different phase of software development life cycle has a long research background [7]. The closet work that is done similar to this work is by Bradley et al. [1] for code history information. Their purpose of history exploration is more broader and can be used in conflict resolution to some extent. The intention of their work is to find who, when, and why some piece of code was developed. They want to answer the question like "Why a piece of code is implemented in this way.", our focus is also on "how the implementation has evolve to this point." They focus on mainly visualization, but did not explore the semantic of history of code. Brosch et al. [2, 4] worked extensively on the conflict resolution of model versioning and included those techniques in AMOR. They provided conflict resolution pattern options to the integrator to facilitate the merging process. They did not attempt to explore provenance information, their conflict detection and resolution techniques are automatic and state of the art. EMF store uses EMF compare to detect model conflict and provide a suggestion for resolution, it works well for small conflicts but does not scale well for a series of conflicts. Sillito et al. [12, 18] argues for the provenance information need in branching and merging and how they effect the developers. Although they don't provide any conflict resolution techniques, their surveys are a strong evidence of provenance information need while merging.

7. FUTURE WORK

Our implementations are in bits and pieces and our first tasks would be integrate them together to get a holistic view where we stands. We have outlined a evaluation plan, now we have to conduct an extensive evaluation on the approach and get some comprehensive metric how useful is it. The work done in [17] is very interesting and we would like to explore any possibility of incorporating this work into our.

8. CONCLUSION

Recent works done by many groups show that provenance information is a useful part of code version merging. We think model merging is no exception of that. By having a well defined infrastructure for harvesting provenance information during the life cycle of any model development, we can potentially answer different questions about the model. This can be an useful tool in the software life cycle especially in maintenance (aka. evolution) phase which possesses about 60 percent of total cost. Model versioning is getting more and more popularity in industry; having tools support will defiantly make MDE approaches attractive to its adopters. Although, we have not managed to finished the tools, we have made some progress on defining and finding the requirement for a useful tool for model merging.

9. ACKNOWLEDGMENTS

This work is financially supported by Ontario Graduate Scholarship and NECSIS.

10. REFERENCES

[1] A. W. Bradley and G. C. Murphy. Supporting software history exploration. In *Proceedings of the 8th Working Conference on Mining Software Repositories*, MSR '11, pages 193–202, New York, NY, USA, 2011. ACM.

[2] P. Brosch. *Conflict Resolution in Model Versioning.* PhD thesis, Vienna University of Technology, 2012.

[3] P. Brosch, G. Kappel, P. Langer, M. Seidl, K. Wieland, and M. Wimmer. An introduction to model versioning. In *Proceedings of the 12th international conference on Formal Methods for the Design of Computer, Communication, and Software Systems: formal methods for model-driven engineering*, SFM'12, pages 336–398, Berlin, Heidelberg, 2012. Springer-Verlag.

[4] P. Broschy. Improving conflict resolution in model versioning systems. In *Proceedings of the 31st International Conference on Software Engineering, ICSE 2009, Canada*, pages 355–358. IEEE, 2009.

[5] M. Bunge. *Treatise on Basic Philosophy: Ontology I: The Furniture of the World*, volume 3. Reidel Boston, 1977.

[6] B. de Alwis and J. Sillito. Why are software projects moving from centralized to decentralized version control systems? In *Proceedings of the 2009 ICSE Workshop on Cooperative and Human Aspects on Software Engineering*, pages 36–39. IEEE Computer Society, 2009.

[7] O. Gotel, J. Cleland-Huang, J. H. Hayes, A. Zisman, A. Egyed, P. GrÃijnbacher, and G. Antoniol. The quest for ubiquity: A roadmap for software and systems traceability research. In *Proceedings of the 20th IEEE International Requirements Engineering Conference*, Chicago, Illinois, USA, 2012.

[8] L. C. James Cheney and W.-C. Tan. Provenance in databases: Why, how, and where. *Foundations and Trends in Databases*, 1:379–474, 2007.

[9] T. D. LaToza and B. A. Myers. Hard-to-answer questions about code. In *Evaluation and Usability of Programming Languages and Tools*, PLATEAU '10, pages 8:1–8:6, New York, NY, USA, 2010. ACM.

[10] T. Mens. A state-of-the-art survey on software merging. *IEEE Trans. Softw. Eng.*, 28(5):449–462, May 2002.

[11] P. Mukherjee, K. Saller, A. Kovacevic, K. Graffi, A. Schürr, and R. Steinmetz. Traceability link evolution with version control. In R. Reussner, A. Pretschner, and S. Jähnichen, editors, *Software Engineering (Workshops)*, volume 184 of *LNI*, pages 151–161. GI, 2011.

[12] S. Phillips, J. Sillito, and R. Walker. Branching and merging: an investigation into current version control practices. In *Proceedings of the 4th International Workshop on Cooperative and Human Aspects of Software Engineering*, CHASE '11, pages 9–15, New York, NY, USA, 2011. ACM.

[13] S. Rastkar, G. C. Murphy, and A. W. Bradley. Generating natural language summaries for crosscutting source code concerns. *Software Maintenance, IEEE International Conference on*, 0:103–112, 2011.

[14] J. L. Sudha Ram. A semantic foundation for provenance management. *Journal on Data Semantics*, 1:11–17, 2012.

[15] G. Taentzer, C. Ermel, P. Langer, and M. Wimmer. A fundamental approach to model versioning based on graph modifications: from theory to implementation. *Software & Systems Modeling*, pages 1–34, 2012.

[16] R. Salay, M. Chechik, S. Easterbrook, Z. Diskin, P. McCormick, S. Nejati, M. Sabetzadeh, and P. Viriyakattiyaporn. An Eclipse-based tool framework for software model management. In *Proceedings of the 2007 OOPSLA workshop on eclipse technology eXchange*, Eclipse '07, pages 55–59, New York, NY, USA, 2007. ACM.

[17] Y. Brun, R. Holmes, D. Ernst, and D. Notkin. Proactive detection of collaboration conflicts. In *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*, ESEC/FSE '11, pages 168–178, New York, NY, USA, 2011. ACM.

[18] S .Phillips, G .Ruhe, and J .Sillito. Information needs for integration decisions in the release process of large-scale parallel development. In *Proceedings of the ACM 2012 conference on Computer Supported Cooperative Work*, CSCW '12, pages 1371–1380, New York, NY, USA, 2012. ACM.