niques, for example from [4], for prioritizing recommendations.

- The recommendations are based on single user changes. Future work should take multiple user changes into consideration.
- Our evaluation only included meta models. We plan to also evaluate our technique for types of instance model for which SiLift can generate differences.
- Our technique needs to be integrated into an intuitive user interface and evaluated by users, because batch evaluations alone are not sufficient [16].
- An extension of our technique that exploits model constraints could filter out recommendations that violate model constraints. It could also be possible to emphasize recommendations that fix constraint violations, which would be similar to Eclipse's Quick Fix recommendations. Muşlu et al. [14] have already done similar work for Eclipse Quick Fixes.
- We want to extend our technique to multiple meta and instance models that are evolving simultaneously. For this, we suspect that it is possible to find relations between changes that are made simultaneously to different models. For example, if two elements are added simultaneously to two different models, we could deduce a relation between them and then try to apply our technique.

6. ACKNOWLEDGEMENTS

This work was funded by the German Research Foundation (DFG) as part of the DFG Priority Programme 1593 (SPP1593).

7. REFERENCES

[1] T. Arendt, E. Biermann, S. Jurack, C. Krause, and G. Taentzer. Henshin: advanced concepts and tools for in-place emf model transformations. In *International Conference on Model Driven Engineering Languages and Systems*, pages 121–135. Springer, 2010.

[2] A. Breckel. Error mining: bug detection through comparison with large code databases. In *Proceedings of the 9th IEEE Working Conference on Mining Software Repositories*, pages 175–178. IEEE Press, 2012.

[3] P. Brosch, M. Seidl, and G. Kappel. A recommender for conflict resolution support in optimistic model versioning. In *Companion to the 25th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, SPLASH/OOPSLA*, pages 43–50, 2010.

[4] M. Bruch, M. Monperrus, and M. Mezini. Learning from examples to improve code completion systems. In *Proceedings of the the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*, pages 213–222. ACM, 2009.

[5] A. Cicchetti, D. Di Ruscio, R. Eramo, and A. Pierantonio. Automating co-evolution in model-driven engineering. In *Enterprise Distributed Object Computing Conference, 2008. EDOC'08. 12th International IEEE*, pages 222–231. IEEE, 2008.

[6] M. Eysholdt and H. Behrens. Xtext: implement your language faster than the quick and dirty way. In *Proceedings of the ACM international conference companion on Object oriented programming systems languages and applications companion*, pages 307–309. ACM, 2010.

[7] S. Getir, M. Rindt, and T. Kehrer. A generic framework for analyzing model co-evolution. In *Model Evolution, International Conference on Model Driven Engineering Languages and Systems*, 2014.

[8] M. Herrmannsdoerfer, D. Ratiu, and G. Wachsmuth. Language evolution in practice: The history of gmf. In *International Conference on Software Language Engineering*, pages 3–22. Springer, 2009.

[9] W. Janjic and C. Atkinson. Utilizing software reuse experience for automated test recommendation. In *8th Int. Workshop on Automation of Software Test*, pages 100–106, 2013.

[10] T. Kehrer, U. Kelter, M. Ohrndorf, and T. Sollbach. Understanding model evolution through semantically lifting model differences with SiLift. In *Software Maintenance (ICSM), 2012 28th IEEE International Conference on*, pages 638–641. IEEE, 2012.

[11] T. Kehrer, U. Kelter, and G. Taentzer. Consistency-preserving edit scripts in model versioning. In *Automated Software Engineering (ASE), 2013 IEEE/ACM 28th International Conference on*, pages 191–201. IEEE, 2013.

[12] P. Langer, M. Wimmer, P. Brosch, M. Herrmannsdörfer, M. Seidl, K. Wieland, and G. Kappel. A posteriori operation detection in evolving software models. *Journal of Systems and Software*, 86(2):551–566, 2013.

[13] N. Meng, M. Kim, and K. S. McKinley. Lase: locating and applying systematic edits by learning from examples. In *Proceedings of the 2013 International Conference on Software Engineering*, pages 502–511. IEEE Press, 2013.

[14] K. Muşlu, Y. Brun, R. Holmes, M. D. Ernst, and D. Notkin. Speculative analysis of integrated development environment recommendations. *ACM SIGPLAN Notices*, 47(10):669–682, 2012.

[15] M. P. Robillard, W. Maalej, R. J. Walker, and T. Zimmermann, editors. *Recommendation Systems in Software Engineering*. Springer, 2014.

[16] A. H. Turpin and W. Hersh. Why batch and user evaluations do not give the same results. In *Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 225–231. ACM, 2001.