Commit 2.0: Enriching Commit Comments with Visualization

Marco D'Ambros
REVEAL @ Faculty of Informatics - University of Lugano, Switzerland
marco.dambros@usi.ch

ABSTRACT

Software developers use commit comments to document changes and as a mean of communication in their team. However, the support given by IDEs is restricted with this respect, as they limit the users to use only text to document changes.

In this paper we propose and implement an approach to enrich commit comments with software visualization: Commit 2.0 generates visualizations of the performed changes at different granularity levels, and let the user enrich them with annotations.

1. INTRODUCTION

Nowadays the majority of all software development projects employ versioning system with a repository in which software developers commit code changes. Many versioning systems (*e.g.,* Git, CVS, ClearCase) allow the developer to write a comment at commit time and store it together with the changes. The information contained in such comments is extremely useful both for software development and software evolution analysis: In software development commit comments are used to document changes and as a mean of communication in the development team. With respect to software evolution, many approaches in the field of Mining Software Repositories, deal with mining and analyzing this commit related information.

Given the importance of commit comments data, developers should write meaningful comments which exhaustively document the changes. However, developers do not always document all the changes in the commit comment. This happens for a number of reasons which can vary among software projects, development teams and organizations, because of different practices and different development rules. Still, a common cause is that writing exhaustive comments is time-consuming, and -being the last step of a coding session- the necessary time and energy could not always be available. Moreover, for commits with many changes, the developers might not remember all of the modifications.

Another problem of commit comments is the lack of context in which changes occur. For example, if a developer changes 5 classes to add a new feature in a system, in the commit comment he can describe the feature and list classes and methods. However, when reading such a comment, it is not clear where the modified classes (and methods) are in the system, which relationships they have and what the size of the change is, relatively to the size of the system.

We argue that IDEs should provide means to ease the task of documenting changes in commits. We describe an approach, called *Commit 2.0*, to enrich commit comments with software visualizations, which provides a visual context to changes and facilitates their documentation.

ICSE '10, May 2-8 2010, Cape Town, South Africa

2. OUR APPROACH

Commit 2.0 is an IDE enhancement which generates visualizations of the changes at different granularity levels, and let the user enrich them with annotations. The tool starts when the developer wants to commit the code, *i.e.,* when she clicks the commit button. At this point, in addition to the standard dialog where the developer can write the comments, Commit 2.0 shows a coarse grained visualization of the system which highlights the changes. The visualization is automatically generated by comparing the last version of the project in the repository with the local modified version. The developer can interact with the visualization by inspecting entities, moving figures, zooming in and out and, most importantly, adding annotations. Annotations are rendered as floating text boxes and can be placed by the developer next to a modified entity to detail and comment the corresponding change. Moreover, the developer can select one or more entities and spawn a fine grained visualization. As with the coarse grained one, the developer can document the changes in the fine grained view by adding an arbitrary number of annotations. Once the developer has completed the documentation of the changes, with one annotated coarse grained visualization and an arbitrary number of annotated fined grained visualizations, she can complete the commit by submitting the changes, the (traditional) comments if any, and the annotated visualizations.

2.1 Example

Figure 1 shows two combined screenshots of Commit 2.0, displaying an annotated coarse grained view on the left and a fine grained one on the right. The screenshots were taken while documenting the changes of the Spyware software system (www.inf.usi.ch/phd/robbes/spyware.html). The coarse grained visualization shows all the packages in the system as rectangle figures, and within each rectangle all the classes

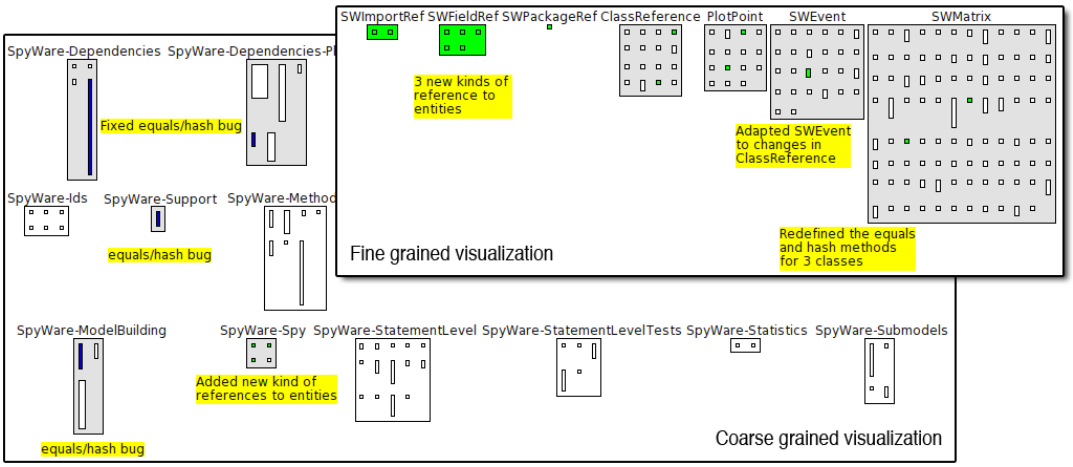

Figure 1: Two combined screenshots of Commit 2.0, showing comments for changes in the large (coarse grained visualization) and detailing them in the small (fine grained visualization).

belonging to the corresponding packages are also depicted as rectangle figures. The width of the rectangles representing classes is proportional to the number of attributes, and the height to the number of methods. The fine grained visualization is a graph where nodes represent classes and edges represent inheritance relationships. Within each node, all the methods belonging to the corresponding class are represented as rectangle figures, where their height is proportional to the number of lines of code. The view is generated from a selection of packages and/or classes in the coarse grained view. In Figure 1 the fine grained view was generated by selecting all the classes that were modified in the coarse grained view. In both views, the following color scheme is applied to highlight the changes: Red represents deletion, *i.e.,* the corresponding entities (packages, classes, methods) have been deleted ; Green represents addition, blue modification and gray represents indirect changes (if an entity is modified the container entity has an indirect change).

2.2 Discussion and Related Work

Commit 2.0 is developed in Smalltalk and is available for the Pharo Smalltalk IDE (http://pharo-project.org). Unfortunately, as other versioning systems (*e.g.,* Cvs, Subversion, Git, ClearCase), the one used in Smalltalk Pharo (called Monticello) does not support the use of images to be attached to commit comments. We solved this problem in the following way: When the developer commits the code and the annotated visualization, the code changes are sent as usually to the versioning system, while the annotated views are automatically published to a blog, having the version number as title. To do this automatically we use Posterous (http://posterous.com), which allows us to post blog entries with images by sending e-mails. As a consequence, every software system versioned using Commit 2.0 will have, in addition to standard historical commit data, a blog with the visual history of changes which can be used by developers as a communication means.

The main benefit of Commit 2.0 is that it provides a visual context to changes, which eases both their documentation and understanding. To help the developer spotting all the changes and to make the approach scalable, the views are interactive, allowing the user to inspect entities, moving them around and zooming in and out. The visualizations are kept simple so that they are easy to learn and understand.

A number of approaches were introduced to visualize versioning system data, as for example the technique proposed by Xie *et al.* [2]. The difference between these approaches and Commit 2.0 is that they visualize the data a posteriori to support retrospective analysis, while Commit 2.0 visualizes changes at commit time to support their documentation. An approach which monitors change data at commit time and checks whether it fulfills architectural constraints were proposed by Knodel *et al.* [1]. Differently from Commit 2.0 this approach does not provide visualization and does not support the documentation of changes.

3. CONCLUSION

In this paper we have proposed a visual approach, and the corresponding implementation, to support the documentation of software changes at commit time. Our technique generates coarse and fine grained visualizations of the changes, and let the developer enrich them with annotations. Since images are not supported as comments in versioning systems, we create an alternative repository by means of a blog, where we store the annotated visualizations. A development team can use such a blog as a communication means.

In the future we plan to develop a version of Commit 2.0 for Eclipse and to conduct a user study to evaluate the effectiveness of documenting changes with our tool.

Acknowledgments. We gratefully acknowledge the financial support of the Swiss National Science foundation for the project “DiCoSA” (SNF Project No. 118063).

4. REFERENCES

[1] J. Knodel, D. Muthig, and D. Rost. Constructive architecture compliance checking – an experiment on support by live feedback. In *Proceedings of ICSM 2008*, pages 287–296, 2008.

[2] X. Xie, D. Poshyvanyk, and A. Marcus. Visualization of cvs repository information. In *Proceedings of WCRE 2006*, pages 231–242. IEEE CS, 2006.