View Infinity: A Zoomable Interface for Feature-Oriented Software Development

Michael Stengel[1]
Janet Feigenspan[1]

Mathias Frisch[1]
Christian Kästner[2]

Sven Apel[3]
Raimund Dachselt[1]

[1]University of Magdeburg, [2]University of Marburg, [3]University of Passau, Germany

[1]{mstengel, feigensp, mfrisch, dachselt}@ovgu.de, [2]kaestner@informatik.uni-marburg.de, [3]apel@uni-passau.de

ABSTRACT

Software product line (SPL) engineering provides efficient means to develop variable software. To support program comprehension of SPLs, we developed *View Infinity*, a tool that provides seamless and semantic zooming of different abstraction layers of an SPL. First results of a qualitative study with experienced SPL developers are promising and indicate that View Infinity is useful and intuitive to use.

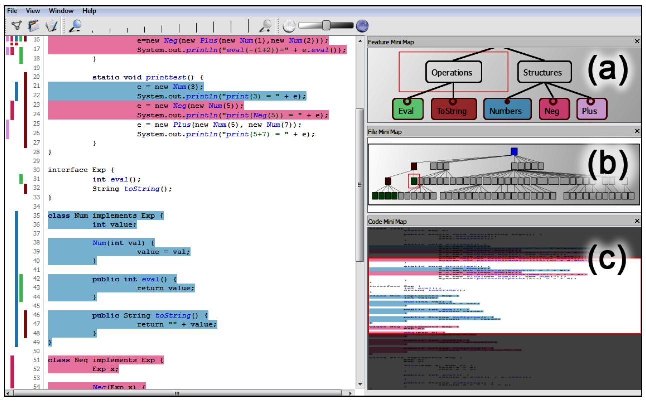
1. INTRODUCTION AND BACKGROUND

To serve an increasing demand for providing variability and customizability in software systems, *software product lines* (SPLs) are often used in practice [6]. In SPLs software is modeled in terms of *features*, which can be defined as user-visible characteristics of a software system and typically implement variable and reusable code fragments. Relationships and dependencies of features are described in *feature models* [4].

Since the source code of SPLs is more complex because of increased variability, it is inherently difficult to understand. Consequently, to better exploit the cost and time benefit of SPLs, we provide tool support for program comprehension.

One way to support program comprehension is to provide different views on a source code base. This can help users to form a mental image of the project, which is one of the goals when exploring source code [1]. A view can be defined as a graphical representation of a data set. Different views on data introduce a discontinuity, when changing from one view to another, because they have to be seperated spatially or temporally. Through that the user has no context information to put the views into relation. To close this gap, we create a linkage between views, while keeping limits of perception and cognitive constraints of the user in mind. This is the point, where zooming is beneficial.

ICSE '11, May 21–28, 2011, Waikiki, Honolulu, HI, USA
ACM 978-1-4503-0445-0/11/05.



```
            e=new Neg(new Plus(new Num(1),new Num(2)));
            System.out.println("eval(-(1+2))=" + e.eval());
        }

        static void printtest() {
            e = new Num(3);
            System.out.println("print(3) = " + e);
            e = new Neg(new Num(5));
            System.out.println("print(Neg(5)) = " + e);
            e = new Plus(new Num(5), new Num(7));
            System.out.println("print(5+7) = " + e);
        }
}

interface Exp {
        int eval();
        String toString();
}

class Num implements Exp {
        int value;

        Num(int val) {
            value = val;
        }

        public int eval() {
            return value;
        }

        public String toString() {
            return "" + value;
        }
}

class Neg implements Exp {
        Exp x;

        Neg(Exp x) {
```

Figure 1: Screenshot of View Infinity. The user interface consists of two parts, left: zoomable main view (currently zoomed to code view), right: mini maps for giving contextual information and showing the visited model layers, feature mini map (a), file mini map (b), code mini map (c). Red rectangles indicate currently focused items.

We contribute the *View Infinity*[1] allowing users to zoom seamlessly between three layers of abstraction of an SPL: feature model, file structure, and source code. For representing the abstraction layers we introduce three different views: *feature model view*, *file view* and *code view* (Fig. 1). In our zooming metaphor, an item of the SPL shows more or less details, depending on its magnification level. By zooming into items, we offer developers an information filter mechanism (*semantic zooming*). To provide a comprehensive (i.e., seamless) transition between all layers, we avoid abrupt transitions and use animation. Furthermore, the zoomable interface is combined with an *Overview + Detail* interface by the use of mini maps [1]. This means that the project data are additionally visualized in miniature to get an overview of the presented data and to provide contextual information. There are other tools that also use continuous zooming and different levels of abstraction, e.g [7]. However, our approach is explicitly tailored to the special needs of SPL development to support programmers and maintainers. The tool aims at intuitive and motivating interaction with feature code and avoids semantic discontinuities between different views.

[1]View Infinity and an extended excerpt of the study are available at `http://fosd.de/vi`

2. VIEW INFINITY

With View Infinity developers can explore an SPL from its feature model down to its implementation. It reads annotations of the source code and the feature model from description files, which can be edited easily in FeatureIDE [5]. View Infinity offers visualization facilities on three layers of abstraction: feature model, file structure, and source code. The central component of the user interface is a zoomable view for immersing important parts of the SPL content. This *main view* is shown in the screenshot of the View Infinity GUI (Fig. 1).

The idea of semantic zooming into project data is to filter the visualized data step by step, while increasing the presented detail of information. The zooming and filtering process is illustrated in Figure 4. A developer starts exploring an SPL at the most abstract level of the feature model in the *feature model view*. In this view, the feature model is visualized as a graph containing connected feature nodes. Furthermore, the user can activate and deactivate features in order to create a specific SPL variant. The user can subsequently zoom into active features and explore the implementation of this features; first at the level of a file structure model (*file view*) and subsequently, after more zooming, at the level of individual code fragments implementing that feature (*code view*). As in the feature view, the file structure is also visualized as a graph containing file nodes and folders (Fig. 3).

View Infinity realizes transitions between abstraction layers with *portals*, which can be feature nodes or file nodes in the respective graphs. When the highest level of detail of one abstraction layer is reached, the next layer is blended smoothly into the node. When the user zooms further, the transition from one view into another is animated.[2] To provide fast navigation between abstraction layers, we offer functions for quick zooming by double clicking nodes. Additionally, there are links to directly change the views.

To scale visualization for larger software projects, we provide different layouts for the graphs of feature model and file structure. Additionally several levels of detail support scalability for larger software projects.

Levels of Detail

To get a better overview of a file without opening it, we provide different levels of detail for the file structure level. When zooming closer to a file node, more information about its source code is visualized (Fig. 2). After a single-color representation at the first level (level 0), indicating that a file contains source code of a feature, we show a feature histogram at the second level (level 1). The feature histogram visualizes a measurement of the amount of code of all features in a file with unique colors for each feature. At the next level (level 2), the approximate positions of source code of an implemented feature in a file are shown. At the highest level of detail (level 3), a thumbnail of the source code is displayed, which shows the source code lines and according features of a file. Additionally, tooltips offer detailed information about features in a file. Further zooming into a file smoothly blends into the source code view (see Fig. 1).

[2]To get a better impression of the zooming in View Infinity, we advise the reader to watch the accompanying video on `http://youtube.com/link` .

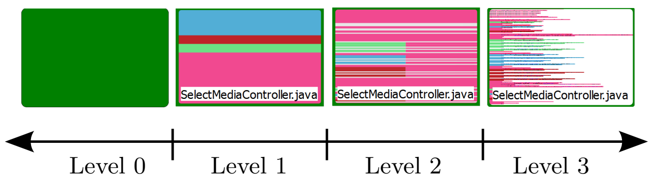

SelectMediaController.java
SelectMediaController.java
SelectMediaController.java
Level 0
Level 1
Level 2
Level 3

Figure 2: Levels of detail for zooming into a file in the file view: simple (0), feature histogram (1), fragments (2), code preview (3)

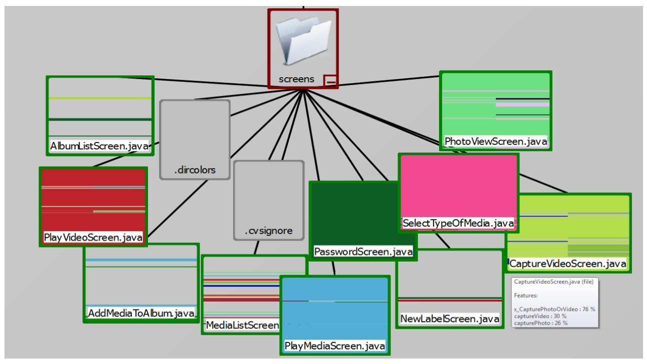


Figure 3: File view with file graph of MobileMedia: feature histograms are visualized for files containing active features.

Mini Maps

To support the user in getting and keeping an overview of an SPL, every abstraction layer is linked with a *mini map*, which is a small representation of the according abstraction layer (right in Fig. 1). Users can browse the mini map without influencing the main view. However, changes in the main view are immediately propagated to the mini map, as motivated in usability tests by Cockburn et al. [1]. In the mini maps of our tool, the visible area of the main view is represented and controlled by a movable rectangle. When zooming, the mini maps appear step by step, showing the visited model layers on top of one another. This provides the benefit that developers know at any time, where they are in the project and what data they currently see.

Feature Colors

Color is one of the most influencing *features of perception* and allows setting the users visual focus preattentatively on relevant information. To improve the mapping of features over multiple layers, the user can assign colors to features of interest. The same colors are used on all layers, to highlight features in the feature model, to highlight the amount of feature implementations per file at file-system level, and to highlight feature implementations at source-code level. As shown before [2], background colors in source code can speed up the comprehension process. When the source code of a file is displayed, code fragments belonging to a certain feature are displayed with the assigned or default background color. Inactive features are greyed out. Vertical bars representing annotated code fragments can be clicked to enable or disable background colors. Furthermore, users can adjust the transparancy of a color with a slider.

3. USER EXPERIENCE

We conducted a qualitative study, to evaluate how experienced developers used View Infinity and whether our semantic zooming concept was comprehensive and considered useful by experienced SPL developers. It was not our intent to compare efficiency of View Infinity with other tools in this study. For the test we used *MobileMedia*, an SPL for the manipulation of multimedia data on mobile devices [3]. The project contains 5,703 lines of code, 51 classes and 11 features.

Participants: We recruited seven participants, who were employed at the University of Magdeburg and who were – according to their own judgment – experienced with SPLs. The programming experience was 10.6 years in average and 3.4 years especially with SPLs. The participants were male with an average age of 28.9 years.

Tasks and Procedure: The study was conducted in two steps: First, participants worked with the IDE, they usually use for programming (Eclipse, Visual Studio). Second, participants worked with View Infinity. For each step, we gave participants two typical, but different tasks (resulting in four tasks for the overall evaluation). In the first task of each step, participants should locate files that belong to a certain feature. In the second task of each step, participants should fix a bug that was located in the code of a certain feature. After completing the second step, we gave participants a questionnaire, in which we asked several questions regarding View Infinity.

Results: We found that the participants could intuitively work with View Infinity and that they liked the idea of seamless zooming. Additionally, most participants said that they would use our tool as part of their preferred IDE. These results indicate that the concepts implemented in View Infinity have high potential and that developers consider them as a useful extension to their IDE. User opinions ('Wow, that's so cool!' or 'That scrolling is annoying.') help us to come to decisions for further design improvements.

4. CONCLUSION

To manage the complexity of SPLs, we implemented View Infinity, which provides seamless zooming from the feature model level to the source code level. Furthermore, mini maps and different levels of detail support the user in navigation and help them to keep an overview in an SPL. Background colors for features displayed in the source code allow quick locating of source code fragments implementing a certain feature. Hence, the use of View Infinity supports the comprehension process and can thus reduce the cost for software development.

In a first evaluation with experienced SPL developers we showed that the zoomable interface concept of View Infinity is considered as useful for feature-based software development. Furthermore, our participants liked the idea of seamless and semantic zooming. However, additional investigation in less experienced users is warranted. In future work, we plan to improve View Infinity and to integrate it into a modern IDE. This would enable the developer to be supported by View Infinity the whole software process. Furthermore, we want to conduct a thorough quantitative evaluation of different aspects of the application, i.e. support in maintenance of feature context, efficiency of the filtering techniques and joy of use by dynamic movements.

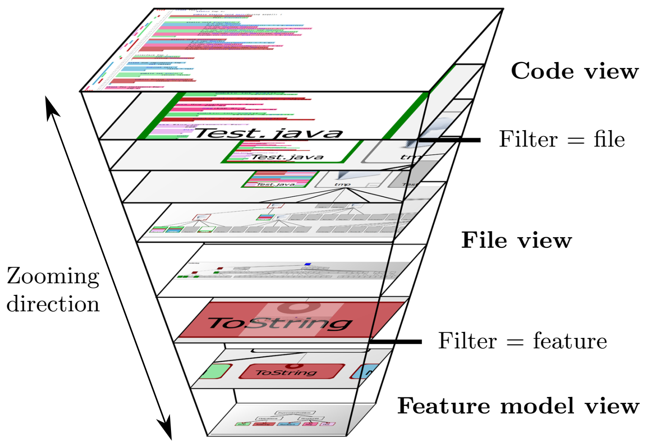


Figure 4: Snapshot of zooming process from feature model layer (bottom) to source code layer (top)

Acknowledgments

We thank Dr. Miriam Goebel–Stengel for support in creating the demonstration video. Janet Feigenspan's work is supported by BMBF project 01IM08003 C. Kästner's work is supported in part by ERC (#203099). Apel's work is supported partly by DFG project #AP 206/2-1.

5. REFERENCES

[1] A. Cockburn, A. Karlson, and B. Bederson. A review of overview+detail, zooming, and focus+context interfaces. *ACM Computing Surveys*, 41(1):1–31, 2008.

[2] J. Feigenspan. Empirical Comparison of FOSD Approaches Regarding Program Comprehension – A Feasibility Study. Master's thesis, University of Magdeburg, 2009.

[3] E. Figueiredo, N. Cacho, M. Monteiro, U. Kulesza, R. Garcia, S. Soares, F. Ferrari, S. Khan, F. Filho, and F. Dantas. Evolving Software Product Lines with Aspects: An Empirical Study on Design Stability. In *Proc. Int'l Conf. Software Engineering (ICSE)*, pages 261–270. ACM Press, 2008.

[4] C. Kästner, S. Apel, and M. Kuhlemann. Granularity in Software Product Lines. In *Proc. Int'l Conf. Software Engineering (ICSE)*, pages 311–320. ACM Press, 2008.

[5] C. Kästner, T. Thüm, G. Saake, J. Feigenspan, T. Leich, F. Wielgorz, and S. Apel. FeatureIDE: Tool Framework for Feature-Oriented Software Development. In *Proc. Int'l Conf. Software Engineering (ICSE)*, pages 611–614. IEEE CS, 2009.

[6] K. Pohl, G. Böckle, and F. van der Linden. *Software Product Line Engineering: Foundations, Principles, and Techniques*. Springer, 2005.

[7] M. Storey, C. Best, and J. Michand. SHriMP Views: An Interactive Environment for Information Visualization and Navigation. In *Proc. 9th Int'l Workshop on Program Comprehension (IWPC)*, pages 111–112. IEEE CS, 2002.