A New Approach of Visualizing Code Smells

Marcel Steinbeck
University of Bremen
marcel@informatik.uni-bremen.de

Abstract

Code smells are indicators of design flaws that may have negative effects on the comprehensibility and maintainability of a software system. Previous studies on software clones have shown that visualization of findings are useful to identify clones suitable for refactorings. However, techniques to visualize code smells in general are rare and, thus, an interesting field of research to bridge the gap between code smell detection and code smell refactoring. This paper presents an approach that is supposed to help in assessing the extent of code smells and in ranking the relevance of refactoring opportunities.

1 Introduction

In the last couple of years several design principles were introduced to enhance the quality of software at code and design level. Code smells, on the other hand, are practices indicating violations of such principles and may have negative effects on software comprehensibility and maintainability. Fowler et al. presented definitions of 22 code smells and suggested to remove findings whenever possible [2]. Based on this suggestion, different tools have been developed to detect and remove code smells by applying appropriate refactorings. However, the visualization of code smells has been widely neglected yet. Though, the visualization of software clones—the code smell claimed most harmful by Fowler—was shown to be a very useful way of assessing individual clones as well as the degree of cloning in software entities such as files, packages, and subsystems.

This paper presents a new approach to emphasize the smell rate of packages and source code files along with the extent of potential refactorings. With that goal in mind, we propose to combine Voronoi treemaps arranged on a circle (similar to the radial layout) with heatmaps and hierarchical edge bundles in order to a) visualize the structure of a particular package consisting of source code files, b) highlight refactoring opportunities and, c) depict the relations of files that contain code smell to each other.

2 Related Work

Balzer et al. [1] presented Voronoi Treemaps as an hierarchy-based approach to visualize arbitrary metrics of a software system. Instead of using traditional rectangle-based layout algorithms, the layout of a Voronoi Treemap is computed by an iterative relaxation of Voronoi tessellations. The advantage of this approach is the enhanced aspect ratio between the width and the height of particular objects represented by Voronoi cells.

Holten [4] introduced hierarchical edge bundles as a flexible approach to reduce the visual clutter of techniques visualizing compound (di)graphs, particularly the adjacency relations between nodes. Moreover, Holten demonstrated how hierarchical edge bundles can be used in conjunction with existing tree visualization techniques, for instance, treemaps and radial trees. In order to customize the presentation of edges and, thus, to provide a continuous trade-off between a low-level and a high-level view of the adjacency relations, a parameter used to adjust the bundling strength has been added.

Hauptmann et al. [3] suggested the usage of edge bundle views to interrelate the structure of a software system with clone detection results. The developed prototype has been implemented on top of the open source quality assessment toolkit ConQAT and provided beneficial indications to uncover errors in the setup of the clone detection tools.

3 Visualizing the Software

Visualizing code smells assist developers to gain insights into a software system and is utile to point out information used to enhance its quality, for instance, by applying appropriate refactorings. Our approach consists of three components:

1. Voronoi treemaps arranged on a circle visualizing the size of packages and source code files in terms of LOC (lines of code).

2. Heatmaps superimposed onto the treemaps emphasizing the smell rate of particular source code files.

3. Hierarchical edge bundles depicting relations between source code files containing code smells.

An example of the resulting visualization is given in Figure 1. In the following sections we will describe our approach in more detail.

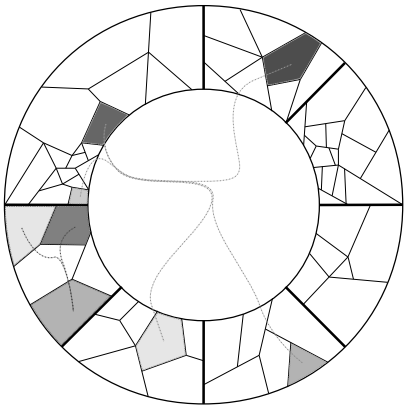

Figure 1: Seven Voronoi treemaps arranged on a circle representing seven packages and their source code files. A heatmap is superimposed onto the treemaps that emphasizes the smell rate of each file. Furthermore, edge bundles are used to depict local and global relations between files that contain code smells.

Voronoi Treemaps and Treemap Arrangement Treemaps are widely used to visualize the structure of a software system as they are well suited to subdivide a surface into smaller polygons based on an arbitrary metric and to depict recursive entities, such as directories that contain further directories and files. Voronoi treemaps have been introduced to enhance the aspect ratio between the width and the height of polygons and, thus, to enhance their comparability. The approach presented in this paper uses Voronoi treemaps to subdivide the surfaces of arcs which, taken together, form a circle in order to a) subdivide a non-rectangular object and b) make usage of the enhanced comparability of polygons. In particular, each package p of a software system is represented by an individual arc a. The length of a corresponds to the size of p in terms of LOC. Thus, the longer a is, the more lines of code p contains compared to other packages of the system. Afterwards, the surface of p is subdivided into smaller polygons by using Voronoi treemaps that are based on LOC as well. The resulting visualization is similar to already known techniques except of the extra space around the center of the circle that is used by edge bundles presented in the remainder of this paper.

Superimposing A Heatmap Layer In addition to Voronoi treemaps, heatmaps are used to visualize the smell rate (SR) of individual source code files. The smell rate of a file f puts LOC of f into relation to the number of lines of f that are part of a code smell. LOC in conjunction with SR is a good indicator for the effect of a source code file on the overall system in terms of maintainability. That is, the larger a file and its smell rate is, the larger the overall amount of source code that is classified as bad practice. Resolving such large instances is most probably more efficient than resolving small ones. However, other metrics may be sufficient, too. The example given in Figure 1 uses different intensities of gray, turning from white in case of files without any code smells at all to dark gray with increasing smell rates. Relating the size of a polygon with its color intensity allows to easily catch files that show high relevance according to both metrics.

Using Edges to Point Out Relations Since every file is independently represented by a polygon, connecting related files with edges is suitable to visualize existing relationships. Examples of such relationships include the usage of functions located in one file by functions located in others. In case of maintenance tasks, pointing out relations between files may give further hints towards a relevance ranking. In order to reduce visual clutter resulting from numerous edges connecting lots of polygons, we suggest to a) draw an individual edge e only if one of the files connected by e contains code smells and b) use hierarchical edge bundles in conjunction with an adjustable bundling strength to provide a continuous trade-off between a low-level and a high-level view of relationships. As one can see in Figure 1, relations between files located in different packages are depicted by edges going through the center of the circle while relations between files located at the same package are depicted by edges staying in the corresponding treemap. This technique may assist developers to further assess the extent of code smells. That is, applying refactorings to files with lots of global relations might be more time-consuming than applying refactorings to files with local relations only.

4 Conclusion

We presented a new approach that is supposed to assist developers in assessing the extent of code smells and in ranking the relevance of refactoring opportunities by combining Voronoi treemaps arranged on a circle with heatmaps and edge bundles. We suggested to visualize packages and source code files with their metrics LOC and SR. However, the proposed approach is neither limited to the visualized elements nor to the selected metrics.

References

[1] M. Balzer, O. Deussen, and C. Lewerentz. Voronoi treemaps for the visualization of software metrics. In *Proceedings of the 2005 ACM symposium on Software visualization*, pages 165–172. ACM, 2005.

[2] M. Fowler, K. Beck, J. Brant, W. Opdyke, and D. Roberts. *Refactoring: Improving the Design of Existing Code*. Addison-Wesley, 1999.

[3] B. Hauptmann, V. Bauer, and M. Junker. Using edge bundle views for clone visualization. In *Software Clones (IWSC), 2012 6th International Workshop on*, pages 86–87, June 2012.

[4] D. Holten. Hierarchical edge bundles: Visualization of adjacency relations in hierarchical data. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):741–748, Sept 2006.