Detecting Bad Smells in Software Systems with Linked Multivariate Visualizations

Haris Mumtaz
VISUS, University of Stuttgart
Germany
Haris.Mumtaz@visus.uni-stuttgart.de

Fabian Beck
paluno, University of Duisburg-Essen
Germany
Fabian.Beck@paluno.uni-due.de

Daniel Weiskopf
VISUS, University of Stuttgart
Germany
Daniel.Weiskopf@visus.uni-stuttgart.de

***Abstract*—Parallel coordinates plots and RadViz are two visualization techniques that deal with multivariate data. They complement each other in identifying data patterns, clusters, and outliers. In this paper, we analyze multivariate software metrics linking the two approaches for detecting outliers, which could be the indicators for bad smells in software systems. Parallel coordinates plots provide an overview, whereas the RadViz representation allows for comparing a smaller subset of metrics in detail. We develop an interactive visual analytics system supporting automatic detection of bad smell patterns. In addition, we investigate the distinctive properties of outliers that are not considered harmful, but noteworthy for other reasons. We demonstrate our approach with open source Java systems and describe detected bad smells and other outlier patterns.**

I. Introduction

Sometimes inappropriate design or implementation decisions degrade software quality (indicated by quality attributes, such as maintainability and modularity) [1]. These decisions introduce bad smells (code smells) [1], which are often reflected numerically in software metrics. This motivates researchers to investigate software metrics for the detection of bad smells. Mostly, multiple metrics are required to measure a software quality attribute, hence, the detection of a bad smell cannot be accomplished through a single metric. The basic idea of this paper is to visually analyze the class-level code quality as reflected in object-oriented software metrics and observe their relations with bad smells.

Since we need to investigate several metrics at the same time, we require the visualization to display multiple variables. Multivariate visualization techniques, such as parallel coordinates and RadViz, apply different strategies to represent the data. Parallel coordinates plots depict multivariate data by displaying the variables in the form of parallel vertical axes [2], whereas RadViz applies non-linear projection of multivariate data onto a 2D space [3][4]. Both techniques assist in identifying relationships among different variables [5][6], discovering clusters, and detecting outliers [7]–[15]. In this paper, we link these visualizations because of their complementing characteristics. Parallel coordinates display data distributions per variable, whereas RadViz is helpful in understanding the interplay of multiple variables. Moreover, parallel coordinates only provide direct correlation information of neighboring variables, whereas RadViz allows for observing relations between more variables.

To summarize, we analyze multivariate object-oriented software metrics using parallel coordinates plots and RadViz to study outliers and their connection to bad smells and software quality attributes. The main contribution of our paper is an interactive visual analytics system to study class-level metrics of Java systems (Section III). The interface of our visual analytics system is shown in Figure 1. The objective is achieved by observing the outliers in the visualizations and analyzing their potential relationships to bad smells. We evaluate our approach using open source Java systems (Section IV). We find that our approach is able to detect bad smells plotted as outliers. In addition, we explore noteworthy outliers that are not detected as bad smells.

II. Related Work

The following subsections provide an overview of related work regarding parallel coordinates, RadViz, integration of these methods with other visualization approaches, and bad smell detection.

A. Parallel Coordinates Plots

Some researchers have used parallel coordinates plots to discern clusters, for instance, Artero et al. [16] propose an approach to identify data clusters by reducing the visual clutter in parallel coordinates. Similarly, Fua et al. [17] develop a multi-resolutional display using clustering that conveys information about the clusters. Johansson et al. [18] also deal with the limitation of parallel coordinates with visualizing large datasets by constructing clusters with high-precision textures. Zhou et al. [19] improve visual clustering by exploiting curved edges and their arrangements. However, only few papers deal with identifying outliers. Novotny and Hauser [20] present an approach that does pre-processing in terms of detecting some features that are used to identify outliers. Zhou et al. [21] propose a technique for pattern identification in parallel coordinates.

B. RadViz

Many data visualization techniques have been proposed that map multivariate data into a low-dimensional space (typically two-dimensional for visualization). In this paper, we adopt RadViz, which applies a non-linear projection, where variables act as anchor points [3][4]. In the context of outlier detection

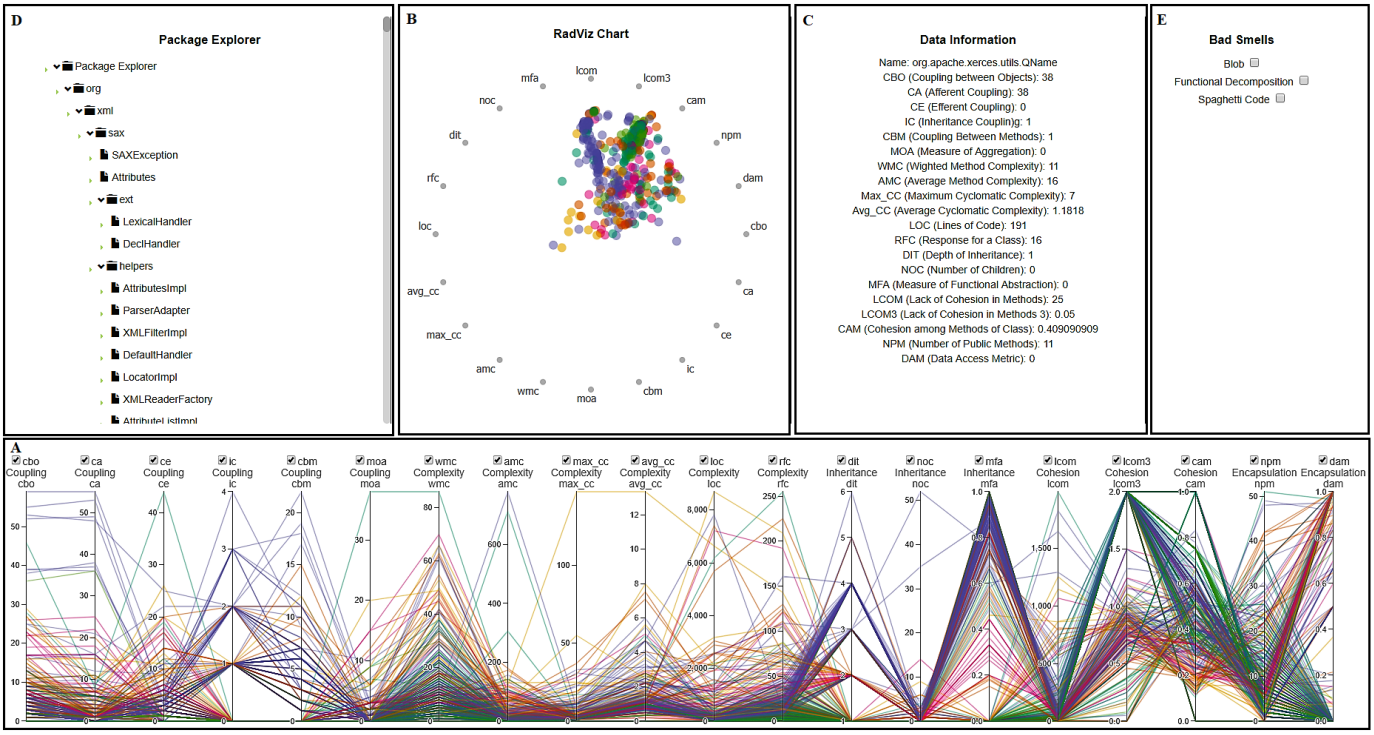


Fig. 1. Interface of our visual analytics system. A: Parallel coordinates view. B: RadViz view to explore noteworthy outlier patterns in detail with respect to a focused set of metrics. C: Software metrics details for a selected class. D: Package explorer for selecting packages and classes. E: Options for automatic detection of basic bad smells.

and cluster identification, the main focus of research is the automatic layout for the anchor points because plotting of points in RadViz is linked with ordering of dimensions [10]–[15]. Artero et al. [10] separate clusters through a dimension ordering and dimension reduction approach. Caro et al. [11] also try to improve visual clusters through dimension ordering. McCarthy et al. [13] and Albuquerque et al. [15] explore high-dimensional visualizations in the context of data analysis techniques for machine learning and quality measurement.

C. Integration of Parallel Coordinates and RadViz

Multiple views are useful in discovering correlations and understanding data from multiple perspectives. However, multiple views should be employed systematically to maintain consistency and correlations between views in order to support meaningful inferences from the visualizations. Bertini et al. [22] build *SpringView*, which simultaneously shows parallel coordinates and RadViz. The integrated view allows observing similarities and clustering on the parallel coordinates plots through interactions with the RadViz plot. The interactions (e.g., brushing) in RadViz allow them to reduce visual clutter in parallel coordinates to easily understand clusters. Geng et al. [23] present angular histograms and attribute curves to deal with high level of cluttering in parallel coordinates. The combination of parallel coordinates plots with angular histograms and attribute curves allows the user to discover clusters, identify correlations, and detect outliers without cluttering and over-plotting. Yuan et al. [24] present an integrated view of parallel coordinates plots and scatterplots. Pillat and Freitas [25] develop a tool that provides multiple views (scatterplots, parallel coordinates, and RadViz), and conduct user studies to evaluate the usability of their tool [26]. They find that users are able to identify few issues while interacting with these visualizations, such as edge crossings in parallel coordinates plots and missing quantitative information in RadViz.

D. Bad Smells Detection Approaches

The majority of the studies used multiple visualizations to detect bad smells. For instance, Carneiro et al. [27] present four views with concern properties: package-class-method structure, inheritance structure, dependencies graph, and dependencies-weighted graph. The visualizations provide support to detect *blob* and *divergent change* bad smells. Steinbeck [28] also propose a visualization approach that combines treemaps, heatmaps, edge bundling, and bar charts arranged in a circular shape. Similarly, Demeyer et al. [29] develop a hybrid approach to comprehend the program structure and detect design anomalies in software systems. They adopt class level, methods level, and attribute level metrics to build hybrid visualizations (tree, histogram, correlation, etc.).

Some researchers conduct user studies with domain experts in detecting bad smells through visualizations. Murphy and Black [30] develop a smell detector tool that provides programmers with an overview of the bad smells and helps understand their origins. Similarly, Santos and Mendonca [31]

involve software developers in detecting bad smell through visualizations. There are few other detection approaches that involve a single visualization. Simon et al. [32] explore bad smells using distance-based cohesion. The components of a class are plotted on a canvas and their positions are computed using their cohesiveness with other classes. They identify the following refactoring opportunities: move methods, move attribute, extract class, and inline class. Dhambri et al. [33] propose a 3D visualization approach to detect design anomalies. Their technique use metrics and structural data to build 3D representations of object-oriented systems. They detect *blob*, *functional decomposition*, and *divergent change*. Parnin et al. [34] present a catalog of bad smells visualizations. They categorize bad smells into class, methods, statement, and collaboration. At class level, they discussed *data class*, *blob*, and *refused bequest*.

It can be observed from this discussion of related work that parallel coordinates and RadViz have been extensively used to identify clusters and outliers. However, they have not been studied in combination with the objective to detect bad smells. In addition, the integrated approaches focus mainly on introducing new interaction mechanisms, dealing with clusters, and evaluating the usability of the visualizations. The detection approaches identify bad smells at a very generic level, for instance, using inheritance hierarchies and package dependencies.

III. Visual Analytics Approach

We have built a visual analytics system linking parallel coordinates and RadViz to detect bad smells and other important outliers. Parallel coordinates provide an overview of the plotted data along with correlation information among neighboring axes. The purpose of RadViz is to view interesting data features (discovered in parallel coordinates) with respect to a focused set of metrics relating to a specific bad smell or outlier pattern. There are several reasons why we have selected these multivariate visualizations. In a glyph-based approach, it is difficult to encode many variables, i.e., it is not feasible to represent each data element as a glyph. Scatterplots do not provide the flexibility to present more than two variables at a time, which restricts understanding of overall picture of data. In case of other projection methods (e.g. t-SNE), we argue that they are less intuitive to interpret because they do not show anchor points, which RadViz does.

In this paper, we consider object-oriented software metrics belonging mainly to the CK metrics suite [35] and QMOOD metrics [36]. The rationale behind selecting these metrics is their ability to measure the object-oriented properties of software systems at class level. We also map the metrics to corresponding quality categories to ease understanding. For instance, *coupling between objects* is mapped to the *coupling* quality category. Similarly, other metrics are mapped to their corresponding quality categories. The mapping is accomplished using the guidelines of Jetter [37]. Table I lists the object-oriented metrics and their relation to quality characteristics.

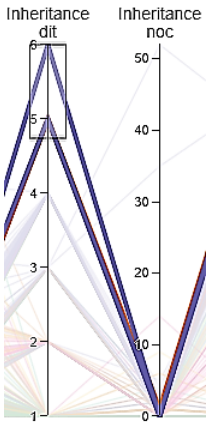


Fig. 2. Highlighting interesting outliers through brushing, here, a negative correlation between *depth of inheritance (dit)* and *number of children (noc)*.

A. Parallel Coordinates View

Figure 1(A) shows the parallel coordinates view. The neighboring axes help discover interesting data features, for instance, positive or negative correlations between software metrics. Functionality to invert axes is also provided to understand the correlations more conveniently. In addition, parallel coordinates plots assist in exploring patterns through brushing. For instance, Figure 2 highlights the outliers with respect to *depth of inheritance (dit)* and *number of children (noc)* through brushing. As mentioned earlier, the order of axes in parallel coordinates is static to enable the users to maintain their mental map and semantical grouping of metrics.

Parallel coordinates are linked with RadViz to support the identification of bad smells and outliers. The brushing at parallel coordinates allows linking with RadViz to display only the selected data elements. In this manner, the interesting patterns visible in parallel coordinates can also be observed in RadViz. The other interactions are annotated in Figure 3 and also presented in supplemental material.

B. RadViz View

The purpose of the RadViz view is to investigate in more detail interesting data patterns observed in the parallel coordinates plot. Figure 1(B) provides the RadViz view plotted with all variables. However, is is possible to restrict the number of variables in RadViz by selecting specific variables from the parallel coordinates plot. This dynamic functionality allows the user to project interesting patterns observed in parallel coordinates onto RadViz.

It is imperative to cross-check identified suspicious outliers with the source code because metrics can easily be misinterpreted. In this regard, a direct link to the source code of a class selected in RadViz is provided. In this manner, the source code can also be inspected for the legitimacy of the bad smells and outliers. We also provide a details-on-demand panel for RadViz to show software metrics values. Only the information of those metrics is presented that are plotted in RadViz. The interactions are presented in Figure 5 and in the supplemental material.

TABLE I
OBJECT-ORIENTED SOFTWARE METRICS

Coupling	Complexity	Cohesion	Inheritance	Encapsulation
• Coupling Between Objects (CBO) • Afferent Coupling (CA) • Efferent Coupling (CE) • Inheritance Coupling (IC) • Coupling Between Methods (CBM) • Measure of Aggregation (MOA)	• Weighted Method per Class (WMC) • Average Method Complexity (AMC) • Maximum Cyclomatic Complexity (MAXCC) • Average Cyclomatic Complexity (AVGCC) • Lines of Code (LOC) • Response For a Class (RFC)	• Lack of Cohesion Metric (LCOM) • Lack of Cohesion Metric (LOCM3) • Cohesion Among Methods of Class (CAM)	• Depth of Inheritance (DIT) • Number of Children (NOC) • Measure of Functional Abstraction (MFA)	• Data Access Metric (DAM) • Number of Public Methods (NPM)

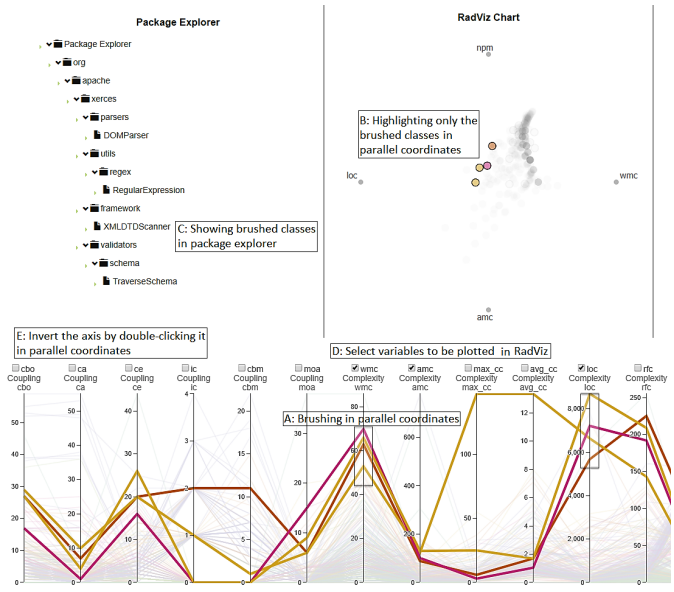


Fig. 3. Interactions through the parallel coordinates view. A: Brushing in parallel coordinates with *wmc (weighted method per class)* and *loc (lines of code)*. B: RadViz view showing the classes that are brushed in parallel coordinates. C: Package explorer showing only the brushed classes. D: RadViz is plotted with the variables selected in parallel coordinates. E: Axes in parallel coordinates can be inverted by double-clicking the axis header (metric name).

C. Package Explorer

This component acts like file explorers, where the folders and files are shown as a tree. The rationale is to provide a view (Figure 1(D)) for exploring packages and classes of the plotted software system. The package explorer is also linked with parallel coordinates through brushing. For instance, brushing in the parallel coordinates plot updates the package explorer with the brushed classes (Figure 4). It is helpful in analyzing the association of bad smells or interesting outliers to packages. For example, data classes with high coupling could belong to a single package. The color mapping used for parallel coordinates and RadViz is based on packages, meaning the classes that belong to a package are plotted with the same color. This makes it easier to identify the package a class belongs to. The color mapping is also beneficial when a certain type of bad smell is found in the classes of the same package.

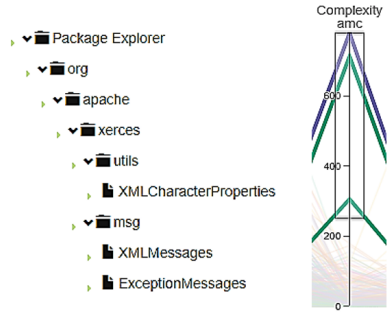


Fig. 4. Brushing of three classes (*XMLCharacterProperties* ■, *XMLMessages* ■, and *ExceptionMessages* ■) in parallel coordinates shows these classes and their package structure on the package explorer.

D. Automatic Detection of Bad Smells

The objective to add this component is to provide support for automatic detection of basic bad smells. In this paper, we have considered the following bad smells:

- *Blob*: A large class that handles most of the systems processing [38].
- *Functional decomposition*: A class with many private data members and few methods [38].
- *Spaghetti code*: A class that declares long methods without parameters [38].

The detection of these bad smells is accomplished by executing rules presented by Ouni et al. [39] and Kessentini et al. [40]:

- R1: IF(*lines of code (loc)* $\geq$ 1500 AND *average method complexity (amc)* $\geq$ 129) OR *weighted method per class (wmc)* $\geq$100) THEN *blob*.
- R2: IF(*number of public methods (npm)* $\leq$ 8 AND *weighted method per class (wmc)* $\geq$ 16) THEN *functional decomposition*.
- R3: IF(*average method complexity (amc)* $\geq$ 151) THEN *spaghetti code*.

Here, *lines of code (loc)* counts the lines of code in a class; *average method complexity (amc)* computes the average method size of each class; *weighted method per class (wmc)* sums the complexities of all the methods in a class (if complexities are unity then wmc equals number of methods

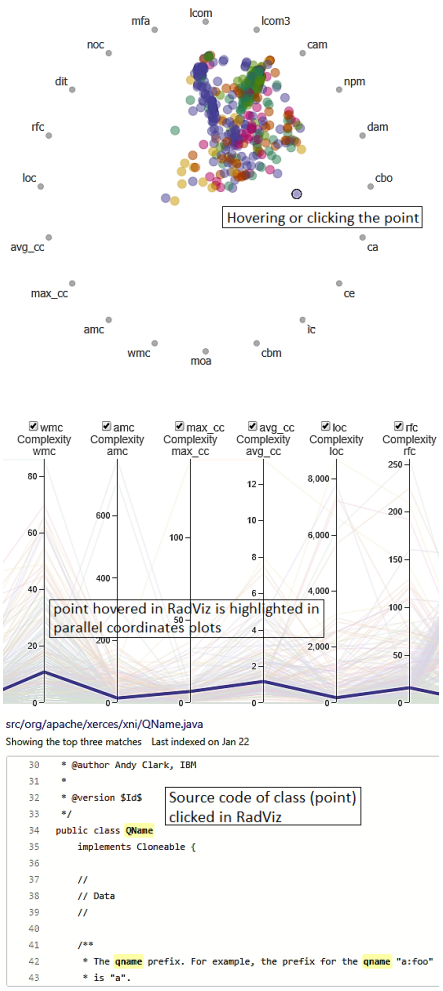


Fig. 5. Interactions through RadViz. The point (class) on RadViz can be hovered or clicked. If hovered, the corresponding edge (class) in the parallel coordinates plot is highlighted. The source code is shown in a separate window if the point (class) is clicked.

in a class); *number of public methods (npm)* counts all the public methods. Once we have detected the bad smells through our approach, we observe whether the bad smells correspond to their distinctive nature as outliers. The selection of a bad smell from the interface (Figure 1(E), also presented in the supplemental material) would detect the particular bad smell by highlighting the points and edges in RadViz and parallel coordinates, respectively.

E. Implementation

The system is built using D3js because of the flexibility it provides for implementing the visualizations. It reads software metrics from a *CSV* file to plot the visualizations.

IV. Results and Analysis

To reiterate, the main purpose of the approach is to identify outliers, which could be the indicators of bad smells. We demonstrate the approach with three open source Java projects: *xerces 1.2*[1], *xalan 2.4*[2], and *poi 3.0*[3]. The selected projects have been extensively investigated with respect to bad smells detection in other studies, primarily in empirical studies [39] [40]. In this paper, we analyze these projects using our visual analytics approach for detection of bad smells and other important outliers. We present our results in detail for one Java project (*xerces 1.2*), to thoroughly explore and understand bad smell detection and other important outliers. The *xerces*[4] is a processor for parsing XML (Extensible Markup Language) files. The detection results from other projects are summarized in Table II and presented in more detail as part of the supplemental material.

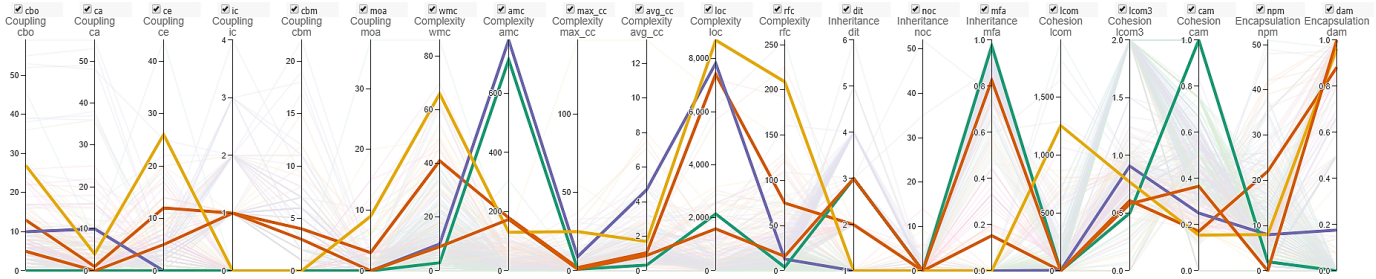
A. Bad Smells Analysis

As a starting point, we find the connection between outliers and bad smells. In other words, we observe whether detected bad smells are depicted as outliers in our linked visualizations. Through our automatic detection functionality, we are able to see the plotting of *blob*, *functional decomposition*, and *spaghetti code* as outliers in RadViz. Figure 6 shows the automatic detection of these bad smells in both visualizations in *xerces 1.2*. It can be observed that the detected bad smells are visually depicted as outliers in RadViz (Figure 6(d), 6(e), and 6(f)). This can also be seen in parallel coordinates (Figure 6(a), 6(b), and 6(c)). For instance, Figure 6(a) shows high values of *lines of code (loc)* and *average method complexity (amc)* metrics associated with *blob* bad smell that also make *blob* to be perceived as outlier in parallel coordinates.

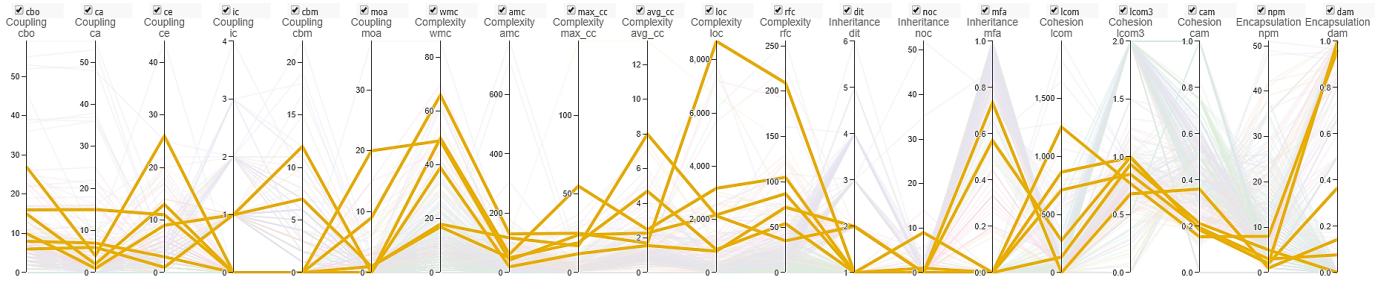
To further support this argument, we focused the variables in RadViz to four software metrics (*weighted method per class (wmc)*, *lines of code (loc)*, *average method complexity (amc)*, and *number of public methods (npm)*), which are all used in detection rules. We observed the connection between bad smells and outliers in abridged RadViz as well. Data elements possessing bad smells as well can be clearly seen as outliers in Figure 7. We noticed a similar behavior of bad smells in the other projects.

We also explored the associations of bad smells in terms of packages of the project. For instance, a bad smell type can belong to a single package of a project. In this case, the color mapping described in Section III is helpful because it is easier to see the affiliations of detected bad smells and their respective packages. We encountered an instance in *xerces 1.2* where five classes out of six, that have *functional decomposition*, belong to a single package. This can be seen in parallel coordinates and RadViz in Figure 6(b) and 6(e), respectively. By skimming through the source codes of these classes, we found that they basically define different tokens for parsing. The functionality here is tokenization, which is decomposed into multiple classes. That is why it is classified as *functional decomposition* bad smell.

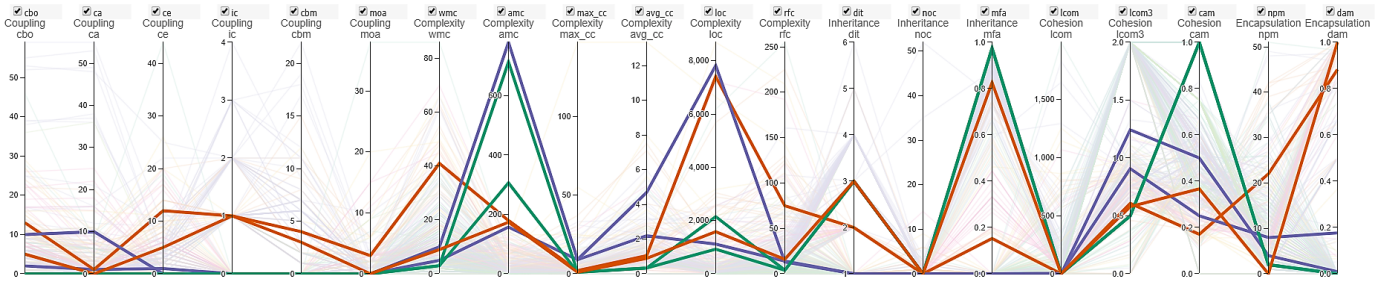
[1]https://github.com/apache/xerces2-j/releases/tag/jaxp-ri-1_2_0-fcs-04
[2]https://github.com/apache/xalan-j/releases/tag/xalan-j_2_4_0
[3]https://github.com/apache/poi/releases/tag/REL_3_0
[4]http://xerces.apache.org/



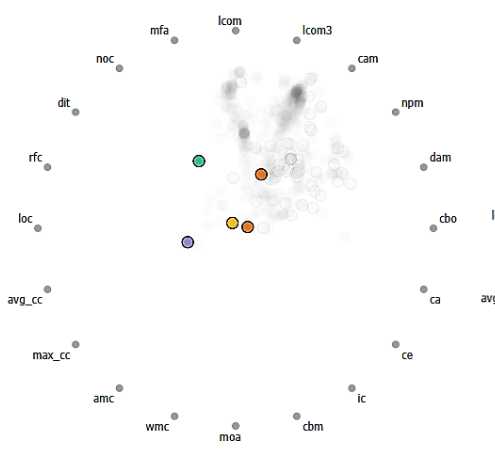
(a) *Blob* detection in parallel coordinates



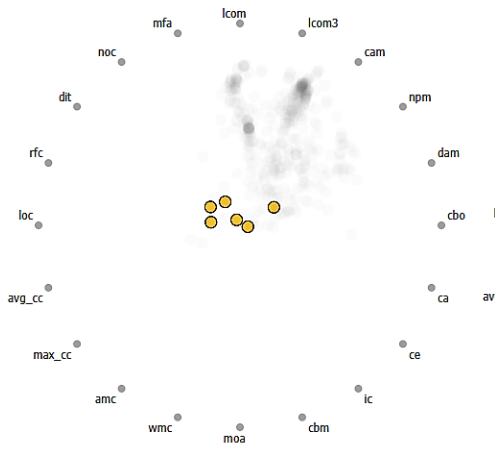
(b) *Functional decomposition* detection in parallel coordinates



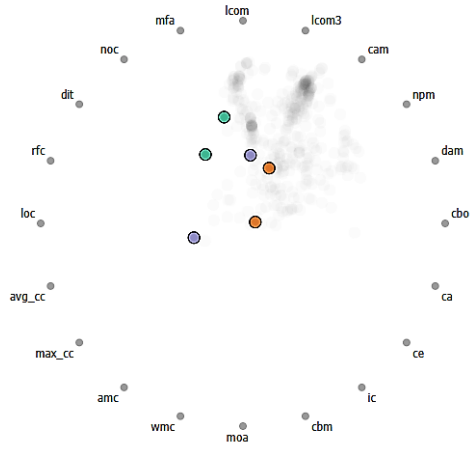
(c) *Spaghetti code* detection in parallel coordinates



(d) *Blob* detection in RadViz



(e) *Functional decomposition* detection in RadViz



(f) *Spaghetti code* detection in RadViz

Fig. 6. Automatic detection of bad smells in *xerces 1.2*.

(a) RadViz view with bad smells related metrics

(b) *Blob* detection

(c) *Functional decomposition* detection

(d) *Spaghetti code* detection

Fig. 7. Automatic detection of bad smells in *xerces 1.2* visualizing four metrics in RadViz.

We also discovered classes that carry multiple bad smells. For instance, we found four classes (*UTF8Reader, UTF8CharReader, XMLCharacterProperties,* and *XMLMessages*) in *xerces 1.2* that have both *blob* and *spaghetti code*. We examined *UTF8Reader* to verify the existence of both these bad smells. We observed that although the *lines of code (loc)* metric is not very high, complexities of methods (*average method complexity (amc)*) were significant, which makes these classes carry *blob* and *spaghetti code*. The reason why *UTF8Reader* has a significant *average method complexity (amc)* is a large number of *if-else* and *switch* statements, which are the key contributors of *spaghetti code*.

B. Outliers Analysis

We also explored outliers that are not detected as any of our selected bad smells. This investigation assists in addressing questions like why a particular outlier is not detected by our detection rules and what metrics properties have classified it as an outlier but not one of the investigated bad smells.

We observed an instance in *xerces 1.2* where two classes (*XMLCharacterProperties* ■ and *RegularExpression* ■) are plotted as outliers in parallel coordinates and RadViz, but only *XMLCharacterProperties* ■ is detected as a bad smell. In RadViz, they are adjacent to each other, which reflects similarities in their metrics values. This behavior can be visually observed in Figure 8 and Figure 9. *XMLCharacterProperties* ■ is detected with *blob* and *functional decomposition*, but *RegularExpression* ■ does not possess any of these bad smells. The reason behind is high values of *lines of code (loc)* and *average method complexity (amc)* in *XMLCharacterProperties* ■, but low values of *weighted method per class (wmc)* and *average method complexity (amc)* in *RegularExpression* ■. However, it is seen as an outlier because it has high *cyclomatic complexity*.

Here, the question arises why method complexities are low, but *cyclomatic complexity (cc)* is high. To answer this concern we examined the source code of *RegularExpression* ■. We found that the implementation has a large number of *while* and *for* loops with *if-else* statements. Since *cyclomatic complexity (cc)* is computed using *control flow graph*, the large number of loops makes it complex. This is the reason why *RegularExpression* ■ has high *cyclomatic complexity (cc)* and low method complexities, which depicts it as an outlier but not a *blob* or *spaghetti code*.

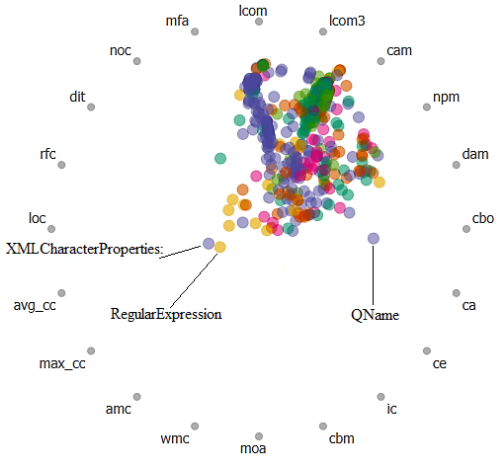


Fig. 8. *XMLCharacterProperties* ■ and *RegularExpression* ■ are adjacent to each other in RadViz, but only *XMLCharacterProperties* ■ is detected as a bad smell. *QName* ■ is a distant outlier but not identified as any of our investigated bad smells.

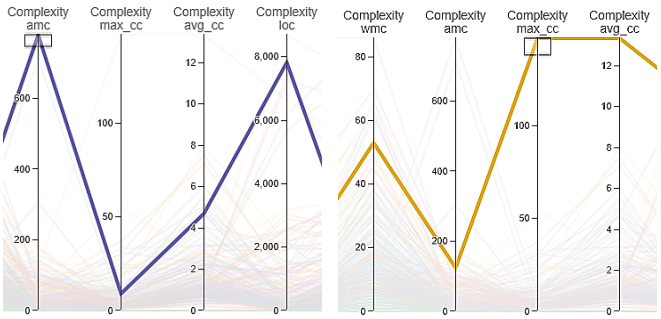


Fig. 9. High *average method complexity (amc)* and *lines of code (loc)* make *XMLCharacterProperties* ■ an outlier and a *blob*. Low *weighted method per class (wmc)* and *average method complexity (amc)* but high *cyclomatic complexity (cc)* make *RegularExpression* ■ an outlier but not a *blob*.

We also identified a distant outlier (*QName* ■) that is not classified with any of our bad smells. This is also annotated in Figure 8. The reason that *QName* ■ is an outlier is that the

majority of the metrics have low values except for coupling-related metrics. Since coupling metrics do not measure *blob*, *functional decomposition*, and *spaghetti code*, *QName* ■ is not classified as a bad smell.

V. Discussion and Limitations

Our results show the ability and usefulness of our approach to detect bad smells and other important outliers. We argue that the majority of the existing approaches achieve bad smells detection at a generic level, for instance, using only *lines of code* to determine the size of classes and then displaying them as boxes. Similarly, assessing the size of project by visualizing inheritance tree using *depth of inheritance* and *number of children*. Our visual analytics approach provides support for multiple metrics and visual exploration, which we argue is beneficial to understand bad smells and build relationships with other data elements. This helps not only detect bad smells but also assists in building the connection between detected bad smells and other important data elements. Our system is practical in a way that it has shown working with software metrics of real open source projects. In addition, we are able to reach interesting results that motivate to study more software projects. Our system could be of interest to software practitioners who want to understand the interplay of software metrics and detect bad smells.

There are some limitations of our approach. The automatic detection of bad smells is dependent on the published detection rules. In this case, the threshold values of software metrics are of critical importance. However, this only partly affects our results and findings because we provide the option to explore the data for further bad smells. In other words, the visual depiction of data complements the detection rules. Another limitation is related to the order of metrics in RadViz. The option to dynamically order the metrics in RadViz would provide more exploration flexibility. In parallel coordinates, the mental map could be disturbed if axes are inverted. We argue that it is beneficial to include the inversion of axes property in parallel coordinates because it helps in understanding correlations more conveniently. Our results are limited to the detection and exploration of bad smells investigated in this paper. The inclusion of other bad smells would further improve the applicability of our approach. In terms of evaluation, we tested our approach only on medium-scale Java projects. Finally, we cannot make statements about the usability of the approach because it is not yet tested by users other than the authors.

VI. Conclusion and Future Work

Parallel coordinates and RadViz are visualization techniques that deal with multivariate data. These visualizations have been used for identifying data patterns, clusters, and outliers. In this paper, we demonstrated our approach, which links parallel coordinates and RadViz, to analyze multivariate object-oriented software metrics to detect outliers, which could be connected to bad smells in the context of software quality. To demonstrate our approach we considered three bad smells (*blob*, *functional decomposition*, and *spaghetti code*) to assess their detection in open source Java systems. As a result, we found that our approach helps visually identify the data elements as bad smells, which are also perceived as outliers in the linked visualizations. In addition, we analyzed software metrics and examined source codes of the detected bad smells and other noteworthy outliers. In future work, we aim to investigate more types of bad smells to generalize our approach. In addition, we plan to evaluate the usability of our visual analytics approach to improve its interface design.

Acknowledgment

Fabian Beck is indebted to the Baden-Württemberg Stiftung for the financial support of this research project within the Postdoctoral Fellowship for Leading Early Career Researchers.

References

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

[2] A. Inselberg, "The plane with parallel coordinates," *The Visual Computer*, vol. 1, no. 2, pp. 69–91, 1985.

[3] K. Daniels, G. Grinstein, A. Russell, and M. Glidden, "Properties of normalized radial visualizations," *Information Visualization*, vol. 11, no. 4, pp. 273–300, 2012.

[4] G. Grinstein, C. B. Jessee, P. Hoffman, P. ONeil, and A. Gee, "High-dimensional visualization support for data mining gene expression data," *DNA Arrays: Technologies and Experimental Strategies, CRC Press*, pp. 86–131, 2001.

[5] J. Heinrich and D. Weiskopf, "State of the art of parallel coordinates." in *Eurographics (STARs)*, 2013, pp. 95–116.

[6] H. Siirtola and K.-J. Räihä, "Interacting with parallel coordinates," *Interacting with Computers*, vol. 18, no. 6, pp. 1278–1309, 2006.

[7] H. Hauser, F. Ledermann, and H. Doleisch, "Angular brushing of extended parallel coordinates," in *IEEE Symposium on Information Visualization*. IEEE, 2002, pp. 127–130.

[8] M. Ankerst, S. Berchtold, and D. A. Keim, "Similarity clustering of dimensions for an enhanced visualization of multidimensional data," in *IEEE Symposium on Information Visualization*. IEEE, 1998, pp. 52–60.

[9] W. Peng, M. O. Ward, and E. A. Rundensteiner, "Clutter reduction in multi-dimensional data visualization using dimension reordering," in *IEEE Symposium on Information Visualization*. IEEE, 2004, pp. 89–96.

[10] A. O. Artero, M. C. F. de Oliveira, and H. Levkowitz, "Enhanced high dimensional data visualization through dimension reduction and attribute arrangement," in *Tenth International Conference on Information Visualization*. IEEE, 2006, pp. 707–712.

[11] L. Di Caro, V. Frias-Martinez, and E. Frias-Martinez, "Analyzing the role of dimension arrangement for data visualization in RadViz," in *Pacific-Asia Conference on Knowledge Discovery and Data Mining*. Springer, 2010, pp. 125–132.

[12] G. Leban, B. Zupan, G. Vidmar, and I. Bratko, "Vizrank: Data visualization guided by machine learning," *Data Mining and Knowledge Discovery*, vol. 13, no. 2, pp. 119–136, 2006.

[13] J. F. McCarthy, K. A. Marx, P. E. Hoffman, A. G. Gee, P. O'Neil, M. L. Ujwal, and J. Hotchkiss, "Applications of machine learning and high-dimensional visualization in cancer detection, diagnosis, and management," *Annals of the New York Academy of Sciences*, vol. 1020, no. 1, pp. 239–262, 2004.

[14] A. Russell, R. Marceau, F. Kamayou, K. Daniels, and G. Grinstein, "Clustered data separation via barycentric radial visualization," in *Proceedings of the International Conference on Modeling, Simulation and Visualization Methods (MSV)*, 2014, pp. 101–107.

[15] G. Albuquerque, M. Eisemann, D. J. Lehmann, H. Theisel, and M. Magnor, "Improving the visual analysis of high-dimensional datasets using quality measures," in *IEEE Symposium on Visual Analytics Science and Technology (VAST)*. IEEE, 2010, pp. 19–26.

[16] A. O. Artero, M. C. F. de Oliveira, and H. Levkowitz, "Uncovering clusters in crowded parallel coordinates visualizations," in *IEEE Symposium on Information Visualization*. IEEE, 2004, pp. 81–88.

[17] Y.-H. Fua, M. O. Ward, and E. A. Rundensteiner, "Hierarchical parallel coordinates for exploration of large datasets," in *Proceedings of the Conference on Visualization*. IEEE Computer Society Press, 1999, pp. 43–50.

[18] J. Johansson, P. Ljung, M. Jern, and M. Cooper, "Revealing structure within clustered parallel coordinates displays," in *IEEE Symposium on Information Visualization*. IEEE, 2005, pp. 125–132.

[19] H. Zhou, X. Yuan, H. Qu, W. Cui, and B. Chen, "Visual clustering in parallel coordinates," in *Computer Graphics Forum*, vol. 27, no. 3. Wiley Online Library, 2008, pp. 1047–1054.

[20] M. Novotny and H. Hauser, "Outlier-preserving focus + context visualization in parallel coordinates," *IEEE Transactions on Visualization and Computer Graphics*, vol. 12, no. 5, pp. 893–900, 2006.

[21] H. Zhou, W. Cui, H. Qu, Y. Wu, X. Yuan, and W. Zhuo, "Splatting the lines in parallel coordinates," in *Computer Graphics Forum*, vol. 28, no. 3, 2009, pp. 759–766.

[22] E. Bertini, L. Dell'Aquila, and G. Santucci, "Springview: Cooperation of radviz and parallel coordinates for view optimization and clutter reduction," in *Third International Conference on Coordinated and Multiple Views in Exploratory Visualization*. IEEE, 2005, pp. 22–29.

[23] Z. Geng, Z. Peng, R. S. Laramee, J. C. Roberts, and R. Walker, "Angular histograms: Frequency-based visualizations for large, high dimensional data," *IEEE Transactions on Visualization and Computer Graphics*, vol. 17, no. 12, pp. 2572–2580, 2011.

[24] X. Yuan, P. Guo, H. Xiao, H. Zhou, and H. Qu, "Scattering points in parallel coordinates," *IEEE Transactions on Visualization and Computer Graphics*, vol. 15, no. 6, pp. 1001–1008, 2009.

[25] R. M. Pillat and C. M. Freitas, "Coordinating views in the InfoVis toolkit," in *Proceedings of the Working Conference on Advanced Visual Interfaces*. ACM, 2006, pp. 496–499.

[26] R. M. Pillat, E. R. Valiati, and C. M. Freitas, "Experimental study on evaluation of multidimensional information visualization techniques," in *Proceedings of the Latin American Conference on Human-Computer Interaction*. ACM, 2005, pp. 20–30.

[27] G. d. F. Carneiro, M. Silva, L. Mara, E. Figueiredo, C. Sant'Anna, A. Garcia, and M. Mendonca, "Identifying code smells with multiple concern views," in *Brazilian Symposium on Software Engineering (SBES)*. IEEE, 2010, pp. 128–137.

[28] M. Steinbeck, "An arc-based approach for visualization of code smells," in *24th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 2017, pp. 397–401.

[29] S. Demeyer, S. Ducasse, and M. Lanza, "A hybrid reverse engineering approach combining metrics and program visualisation," in *Sixth Working Conference on Reverse Engineering*. IEEE, 1999, pp. 175–186.

[30] E. Murphy-Hill and A. P. Black, "An interactive ambient visualization for code smells," in *Proceedings of the 5th International Symposium on Software Visualization*. ACM, 2010, pp. 5–14.

[31] J. A. M. Santos and M. G. de Mendonça, "Exploring decision drivers on god class detection in three controlled experiments," in *Proceedings of the 30th Annual ACM Symposium on Applied Computing*. ACM, 2015, pp. 1472–1479.

[32] F. Simon, F. Steinbruckner, and C. Lewerentz, "Metrics based refactoring," in *Fifth European Conference on Software Maintenance and Reengineering*. IEEE, 2001, pp. 30–38.

[33] K. Dhambri, H. Sahraoui, and P. Poulin, "Visual detection of design anomalies," in *12th European Conference on Software Maintenance and Reengineering*. IEEE, 2008, pp. 279–283.

[34] C. Parnin, C. Görg, and O. Nnadi, "A catalogue of lightweight visualizations to support code smell inspection," in *Proceedings of the 4th ACM Symposium on Software Visualization*. ACM, 2008, pp. 77–86.

[35] S. R. Chidamber and C. F. Kemerer, "A metrics suite for object oriented design," *IEEE Transactions on Software Engineering*, vol. 20, no. 6, pp. 476–493, 1994.

[36] J. Bansiya and C. G. Davis, "A hierarchical model for object-oriented design quality assessment," *IEEE Transactions on Software Engineering*, vol. 28, no. 1, pp. 4–17, 2002.

[37] A. Jetter, "Assessing software quality attributes with source code metrics," Diploma thesis, University of Zurich, Department of Informatics, Zurich, 2006.

[38] W. H. Brown, R. C. Malveau, H. W. McCormick, and T. J. Mowbray, *AntiPatterns: Refactoring Software,Aarchitectures, and Projects in Crisis*. John Wiley & Sons, Inc., 1998.

[39] A. Ouni, M. Kessentini, H. Sahraoui, and M. Boukadoum, "Maintainability defects detection and correction: a multi-objective approach," *Automated Software Engineering*, vol. 20, no. 1, pp. 47–79, 2013.

[40] M. Kessentini, W. Kessentini, H. Sahraoui, M. Boukadoum, and A. Ouni, "Design defects detection and correction by example," in *19th International Conference on Program Comprehension (ICPC)*. IEEE, 2011, pp. 81–90.