MetricAttitude++: Enhancing Polymetric Views with Information Retrieval

Rita Francese
Department of Computer Science
University of Salerno
Email: francese@unisa.it

Michele Risi
Department of Computer Science
University of Salerno
Email: mrisi@unisa.it

Genoveffa Tortora
Department of Computer Science
University of Salerno
Email: tortora@unisa.it

***Abstract*—MetricAttitude is a visualization tool based on static analysis that provides a mental picture by viewing an object-oriented software by means of polymetric views. In this tool demonstration paper, we integrate an information retrieval engine in MetricAttitude and name this new version as MetricAttitude++. This new tool allows the software engineer to formulate free-form textual queries and shows results on the polymetric views. In particular, MetricAttitude++ shows on the visual representation of a subject software the elements that are more similar to that query. The navigation among elements of interest can be then driven by the polymetric views of the depicted elements and/or reformulating the query and applying customizable filters on the software view. Due to its peculiarities, MetricAttitude++ can be applicable to many kinds of software maintenance and evolution tasks (e.g., concept location and program comprehension).**

***Keywords*-Program Comprehension; Reverse Engineering; Software Visualization; Information Retrieval**

I. Introduction

One of the main difficulties of software maintainence is software comprehension, since maintainers spend 50% of their time simply trying to comprehend the source code to be changed [1]. They can be supported in this activity by software visualization, enabling the visualization of information related to a software system by means of visual representations of its structure, execution, behavior, or evolution [2].

There are a number of software visualization approaches based on graph representations and polymetric views (i.e., lightweight visualizations enriched with software metrics) [3], [4], [5], [6]. Recently, Risi and Scanniello [6], [7] proposed MetricAttitude, an Eclipse Rich Client Platform application for software visualization based on polymeric view, which shows traditional software metrics, object-oriented (OO) metrics and infers instance level relationships among software entities. This approach works at class level and uses only static information from source code. Successively, Francese *et al.* [8] have conducted an empirical investigation using questionnaire-based surveys with students and software professional developers to assess the validity of the approach implemented in the MetricAttitude prototype. Although the validity of proposed approaches in supporting the software comprehension, they do not provide an adequate support to interact with software visualizations to mine the information needed to get a deep understanding of the whole software or part of it.

In this tool demonstration paper, we present MetricAttitude++,[1] an extension of MetricAttitude. The main characteristics of MetricAttitude++ can be summarized as follows:

- OO and traditional metrics are visualized;
- A novel static analysis based approach to recover relationships among classes;
- Relationships among classes and software metrics are recovered and inferred from source code to simulate late binding;
- Customized filters in terms of expressions on software metrics;
- A metric based on textual similarity among classes (each of them represents a document in the corpus) and with respect to free-form textual queries.

The latter two characteristics are new in MetricAttitude++. Among them the feature that allows the computation and visualization of classes similar to a given textual query is the most interesting if considered in the software visualization field. Indeed, Xie *et al.* [9] proposed a visualization approach showing the similarities of a query to a software system (although the visualization is in 3D) by merging sv3D [10] and IRiSS [11].

The rationale behind the new feature in MetricAttitude++ was to fill a gap between software visualization and information retrieval (IR), in order to ease the browsing and the understanding of a software. For example, imagine executing a query (i.e., a change request of a faulty feature) on the documents in the corpus (i.e., Java files), an IR engine could be used to retrieve documents based on their similarity to that query and to arrange these documents in a ranked list. The software engineer investigates results in the order they are retrieved. If a software engineer finds the Java file implementing the faulty feature, then the search succeeds; otherwise, he/she has to formulate a new query, taking into account new knowledge obtained from the investigated Java files. While searching a faulty feature, the software engineer has a limited view of the entire software since he/she only analyzes a ranked list of the Java files and possibly inspects these files. On the other hand, the combination of software visualization and of an IR engine allows the software engineer to get a wider understanding of the software (e.g., relationships among Java

[1] http://www.sesa.unisa.it/MetricAttitude++/

TABLE I
METRICS AND VISUAL MAPPING IN METRICATTITUDE++

OO Metrics	
WMC (Weighted Methods per Class)	Size of ellipse
DIT (Depth of Inheritance Tree)	Flatten the ellipse
NOC (Number Of Children)	Color of ellipse border
CBO (Coupling Between Object classes)	Thickness of ellipse border
RFC (Response For a Class)	Color of ellipse
Traditional Code-size Metrics	
FI (Flow Info)	Color of rectangle
NMS (Number of Message Sends)	Thickness of rectangle border
NM (Number of Methods)	Color of rectangle border
LOC (Lines Of Code)	Height of rectangle
NC (Number of Comments)	Width of rectangle
Relationships	
Hierarchy	Hollow triangle shape
Delegation	Blue arrowed line
Field Access	Cyan arrowed line
Abstract Delegation	Orange arrowed line
Virtual Delegation	Green arrowed line
Method Call Loop	Blue line
Field Access Loop	Cyan line

files are clearly represented and their polymetric views are available), thus better supporting the software engineer to either localize the faulty feature or formulate a new query.

II. METRICATTIDUDE++

MetricAttitude++ adopts software visualization techniques based on static analysis, providing a mental picture of an OO software system enriched with OO metrics and traditional code-size metrics. In particular, MetricAttitude++ exploits the most popular OO metrics presented by Chidamber and Kemerer [12] and well known traditional code-size metrics, as shown in Table I. On the bottom of this table, we show the mapping between the class relationships and the used graphical representations. Software entities (i.e., abstract classes, interfaces, concrete classes, and inner classes) are graphically represented with an ellipse and a rectangle placed just below (see Figure 1). The properties of each ellipse summarize OO metrics, while those of the rectangle traditional code-size metrics. For example, we used the flattening of an ellipse to represent DIT (Depth of Inheritance Tree). The more the flattening, the higher the metric value. The height of a rectangle represents the number of lines of code (LOC). The interested reader can find further details on the mapping between graphical elements and software metrics in [6].

MetricAttitude++ builds a graph, where each node is a representation of a software entity and directed edges are invocations from a method in a class to another method in the same class or in a different class. The relationships among classes are graphically represented as edges. The edges can assume different colors (i.e., the kind of relationship as shown on the bottom of Table I) and thickness (i.e., the larger the number of links between two classes, the larger the thickness).

Since a graph may have a huge number of nodes, MetricAttitude++ simplifies the search of classes with specific characteristics by allowing the software engineer to define and apply filters to a software view on the basis of user-defined metric expressions and by integrating IR-based search features. In addition, MetricAttitude++ allows filling a gap between fine- and coarse-grained comprehension. For example, the software engineer might perform a comprehension task focusing on a small part of a subject software (e.g., a class). His/her software comprehension could incrementally increase following relationships among classes, namely the edges of the graph. This might be particularly useful in concept location, where features are special concepts (or a subset of concepts) that are associated with the developed visible functionality of the software [13]. The goal is to identify the source code units (such as, methods, function, classes, etc.) that implement (part of) a concept of interest from the problem or solution domain of the software. The identification of source code unit may follow an incremental process based on software visualization.

A. Main Components and Used Technologies

MetricAttitude++ is composed of four main software components aimed at: *(i)* selecting source files of a subject software and interacting with the software engineer; *(ii)* extracting both the software metrics for each class and their relationships; *(iii)* building an intermediate representation of the software; and *(iv)* searching software entities with an IR engine. To this end, we used the following technologies:

Eclipse RCP.[2] This technology based on Java and a plug-in architecture that allows developers to use the Eclipse platform to create flexible and extensible desktop applications. We opted for Eclipse RCP also because it made MetricAttitude++ platform independent.

JDT (Java Development Tools) Core.[3] It is included in the Eclipse SDK and offers a built-in incremental Java compiler to run and debug code, which still contains unresolved errors. JDT Core also provides a model for Java source code. The model offers API for navigating elements like package fragments, compilation units, binary classes, types, methods, fields, and so on. Although JDT Core implements a number of features, MetricAttitude++ exploits the incremental Java compiler to make independent a subject software from the libraries it uses. The prototype also uses the Java source

[2]http://www.eclipse.org/rcp/
[3]http://www.eclipse.org/jdt/core/

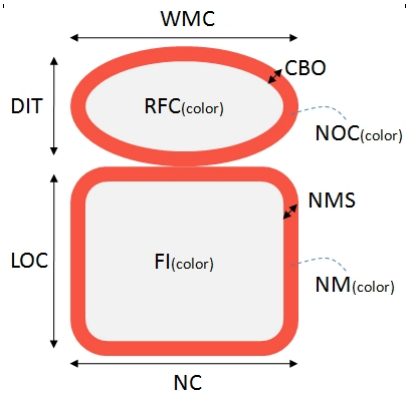

Fig. 1. Visual properties of nodes in MetricAttitude++.

TABLE II
USAGE STEPS FOR A SAMPLE APPLICATION TO CONCEPT LOCATION

STEP	OUTPUT	DESCRIPTION
1. Selection of the software to study.	A new Eclipse project is built.	It creates a new project that contains the source code of a subject software.
2. Analyzing the project.	MetricAttitude++ graphically shows the software.	It allows a software engineer to visualize the graphical representation of a subject software.
3. Formulate the query and set the similarity threshold.	The filtered graph which highlights the class more similar to the query.	Choose the query (e.g., the original bug description) and set the similarity threshold for SIM so that very different classes (from the lexical point of view) are not visualized. The graphical representation is modified accordingly.
4. Navigation of the results.	Graphical objects are displayed accordingly the chosen layout algorithm.	The software engineer starts from the class that MetricAttitude++ found more similar to the formulated query and follows a path from this class to any class whose methods are in the original change set.

code model to identify class instance level relationships and structure of hierarchies. MetricAttitude++ integrates JDT Core to get all the information to compute the metrics to be used in the metaphor. Relationships and metric values are then used to build an intermediate representation of the software.

Graphical Editing Framework (GEF).[4] It is a Java library, which adopts the MVC architecture for creating advanced interactive diagrams. MetricAttitude++ uses that technology to visualize a software on the basis of its intermediate representation. The diagram layout is drawn by exploiting the features offered by Zest, also included in GEF, containing a graph layout package offering algorithms to dispose the nodes in a two-dimensional space (e.g., hierarchical, organic, and tree). Draw2d, another tool of GEF, has been adopted for displaying graphical components on an SWT Canvas.

Lucene.[5] It is a full-featured text search engine library written in Java. Our prototype uses Lucene as the IR engine. In particular, Lucene provides the parser to extract text from Java files and to reduce it to a normal form by adopting a stemmer algorithm and removing the stop-words. Then, Lucene uses an indexer to compare textual queries to the corpus, and through the cosine similarity measure Lucene creates the ranked list.

Although there are other IR engines, we opted for Lucene because it is well known and widely adopted.

III. CONCEPT LOCATION IN METRICATTITUDE++

During concept location a software engineer starts with a change request and finds a place in the code where a change should be made. To verify that this location is correct, the complete change should be implemented and tested. In Table II, we show the four steps the software engineer has to carry out to use MetricAttitude++ for concept location. The expected output for each step is also mentioned together with its short description. Although we use concept location as an example task, MetricAttitude++ could also be used in other maintenance tasks (e.g., impact analysis)

Once a subject software has been selected and graphically shown (Steps 1 and 2, respectively, MetricAttitude++ allows the specification of textual queries (Step 3). The similarity between the query and each class and between pairs of classes (SIM) is determined by using the vector space model of Lucene and cosine similarity. SIM determines the similarity of classes (each of them represents a document in the corpus) with respect to a textual query. SIM belongs to $[0, 1]$, where *0* denotes that the textual content is completely different, while *1* denotes equality. MetricAttitude++ also provides a histogram to show the distribution of the SIM values (see Figure 2). This simplifies the choice of thresholds to filter-out irrelevant classes from the software visualization on the basis of SIM values. For all the metrics on which MetricAttitude++ visualization is based, histogram representations are also available for similar purposes.

As for concept location, the main novelty of MetricAttitude++ is Step 4. In traditional concept location, results are organized in a ranked list of classes arranged according to their similarity with the query (in descending order). The software engineer navigates such a list to find concepts in the source code. Moreover, the software engineer can open a text editor that shows the source code by double clicking on a class in the graph.

In MetricAttitude++, results are organized in a two-dimensional space, where many information on classes are also

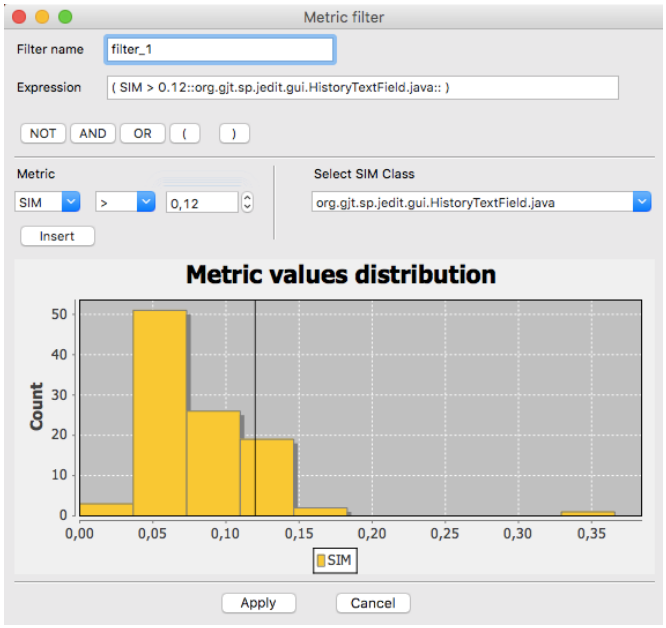

Fig. 2. The filtering operation in MetricAttitude++.

[4]https://eclipse.org/gef/
[5]https://lucene.apache.org/

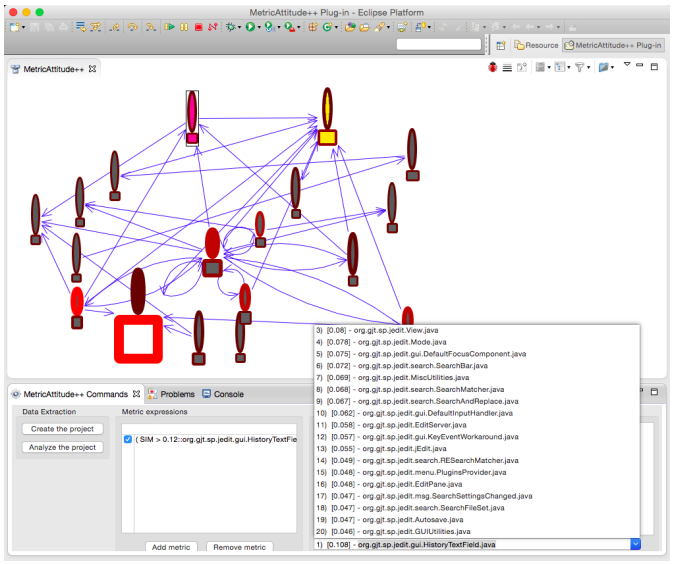

Fig. 3. MetricAttitude++ snapshot.

available (and filtered-out if needed). For example, Figure 3 shows the graph for the tool JEdit obtained by applying $SIM > 0.12$ as the threshold for the filtering operation (depicted in Figure 2). We also choose as the query a bug description[6] from the bug tracking system of JEdit. The first class of the ranked list is colored in magenta (top). The navigation of the graph might start from this class and follows any path from it to any buggy class whose methods are in the original change set of the bug. In Figure 3, we depict such a class in yellow (on the top). We highlighted this class to ease the description of Figure 3; MetricAttitude++ is clearly not able to predict buggy classes. If we search the buggy class in the ranked list of Figure 3, we find it at the 20^{th} place, while along the graph we have to follow only one edge. Clearly this does not mean that MetricAttitude++ reduces the time/effort to execute concept location. The number of possible alternative paths is high and many of them move away from buggy classes. In addition, the class containing the bug(s) could be not physically linked with the class returned on the top of the ranked list. However, the visual information related to each class as well as the supported filtering operations might make easier the identification of class/es containing bug/s. Unfortunately, we cannot make any postulation on how actual software engineers deal with concept location tasks supported by MetricAttitude++. To investigate on this point, we plan to conduct controlled experiments with actual software engineers.

IV. Conclusion

Software visualization allows studying multiple aspects of complex problems. To deal with these issues, visualization techniques based on graph representations and on polymetric views have been proposed [4], [14]. Lanza and Ducasse proposed polymetric views fro the first time in [4]. Reniers *et al.* [14] presented SolidSX, an integrated tool for the visualization of large software. Filters can be applied to hide useless information. A deep and exhaustive discussion of related work can be found in [15].

In this paper, we have presented MetricAttitude++. It provides a mental picture by viewing a software through several views based on metrics and relationships among classes. MetricAttitude++ also integrates the polymetric-based software visualization with an IR engine to support comprehension tasks. Although some similarities with the approach presented in [4], [14] and [16], MetricAttitude++ provides metric-based filtering features and supports concept location and impact analysis by exploiting IR-based filtering.

[6] *On Linux (unlike on Windows) using JVM java version "1.5.0_04" Java(TM) 2 Runtime Environment, Standard Edition (build 1.5.0_04-b05) Java HotSpot(TM) Client VM (build 1.5.0_04-b05, mixed mode, sharing), when trying to search, the focus is not always at the "find" field (the field where the text to be searched is to be entered). There is an about 50% chance to get the focus. Expected behaviour is: There should be a 100% chance to get the focus. This bug is reproducible.*

References

[1] R. K. Fjeldstad and W. T. Hamlen, "Application program maintenance study: Report to our respondents," in *Procs. of GUIDE 48*, 1983.

[2] R. Koschke, "Software visualization in software maintenance, reverse engineering, and re-engineering: A research survey," *Journal of Software Maintenance*, vol. 15, no. 2, pp. 87–109, 2003.

[3] M. Lanza, "CodeCrawler - Polymetric views in action," in *Procs. of International Conference on Automated Software Engineering*. IEEE CS Press, 2004, pp. 394–395.

[4] M. Lanza and S. Ducasse, "Polymetric views - A lightweight visual approach to reverse engineering," *IEEE Transactions on Software Engineering*, vol. 29, no. 9, pp. 782–795, 2003.

[5] P. Martin, G. Harald, and F. Michael, "Towards an integrated view on architecture and its evolution," *Electronic Notes in Theoretical Computer Science*, vol. 127, no. 3, pp. 183–196, 2005.

[6] M. Risi and G. Scanniello, "MetricAttitude: A visualization tool for the reverse engineering of object oriented software," in *Procs. of International Working Conference on Advanced Visual Interfaces*, 2012, pp. 449–456.

[7] M. Risi, G. Scanniello, and G. Tortora, "MetricAttitude," in *Procs. of Conference on Software Maintenance and Reengineering*, 2013, pp. 405–408.

[8] R. Francese, M. Risi, G. Scanniello, and G. Tortora, "Proposing and assessing a software visualization approach based on polymetric views," *Journal of Visual Languages and Computing*, vol. 3435, pp. 11–24, 2016.

[9] X. Xie, D. Poshyvanyk, and A. Marcus, "Support for static concept location with sv3D," in *Procs. of 3rd IEEE International Workshop on Visualizing Software for Understanding and Analysis*, 2005, pp. 1–6.

[10] A. Marcus, L. Feng, and J. I. Maletic, "3D representations for software visualization," in *Procs. of ACM Symposium on Software Visualization (SoftVis)*. ACM, 2003, pp. 27–ff.

[11] D. Poshyvanyk, A. Marcus, Y. Dong, and A. Sergeyev, "IRiSS - A source code exploration tool," in *Procs. of 21st IEEE International Conference on Software Maintenance (ICSM)*, 2005, pp. 69–72.

[12] S. R. Chidamber and C. F. Kemerer, "A metrics suite for object oriented design," *IEEE Transactions on Software Engineering*, vol. 20, pp. 476–493, 1994.

[13] B. Dit, M. Revelle, M. Gethers, and D. Poshyvanyk, "Feature location in source code: a taxonomy and survey," *Journal of Software Maintenance and Evolution: Research and Practice*, 2011.

[14] D. Reniers, L. Voinea, and A. Telea, "Visual exploration of program structure, dependencies and metrics with SolidSX," in *Procs. of IEEE Vissoft*. IEEE CS Press, 2011, pp. 1–4.

[15] B. A. Price, R. M. Baecker, and I. S. Small, "A principled taxonomy of software visualization," *Journal of Visual Languages and Computing*, vol. 4, no. 3, pp. 211–266, 1993.

[16] O. Nierstrasz, S. Ducasse, and T. Gîrba, "The story of moose: An agile reengineering environment," *SIGSOFT Softw. Eng. Notes*, vol. 30, no. 5, pp. 1–10, 2005.