Journal of Visual Languages and Computing 34-35 (2016) 11–24

Contents lists available at ScienceDirect

Journal of Visual Languages and Computing

journal homepage: www.elsevier.com/locate/jvlc

Proposing and assessing a software visualization approach based on polymetric views

Rita Francese [a,*], Michele Risi [a], Giuseppe Scanniello [b], Genoveffa Tortora [a]

[a] *University of Salerno, Italy*
[b] *University of Basilicata, Italy*

ARTICLE INFO

Article history:
Received 11 July 2015
Received in revised form
27 April 2016
Accepted 6 May 2016
Available online 19 May 2016

Keywords:
Empirical evaluation
Qualitative study
Software visualization
Polymetric-views

ABSTRACT

In this paper, we present an approach for the visualization of object-oriented software. This approach has been implemented in MetricAttitude, a visualization tool based on static analysis that provides a mental picture of a software implemented in Java by means of polymetric views. The approach graphically represents a suite of object-oriented design metrics (e.g., Weighted Methods per Class) and “traditional” code-size metrics (e.g., Lines Of Code). To assess the validity of our proposal, we have conducted two users' studies with students in Computer Science and professional software developers. The used empirical method is qualitative. To assess MetricAttitude and its underlying approach, we conducted questionnaire-based surveys. Results suggest that MetricAttitude is a viable means to deal with existing objects-oriented software and to comprehend their source code, in particular.

1. Introduction

Many activities in software engineering involve existing software. Software maintenance, testing, quality assurance, reuse, and integration are only a few examples of software processes that involve existing software [1]. In software maintenance field, visualization [2,3] is extensively investigated and successfully adopted to deal with existing software and to improve software comprehensibility [4].

Researchers have proposed approaches/techniques and supporting tools based on 2D and 3D environments [4–7]. These approaches visualize either static information, performing static analysis (e.g., [8]), or dynamic information, relying on software execution traces (e.g., [9]). The weakness of dynamic based approaches is that visualization strongly depends on the scenarios chosen to execute software [10]. Whatever the information of a software to visualize, a relevant concern is how it is presented to the developer. It might happen that a visualization offered by a tool is cryptic or vague, so failing to improve software comprehension. The choice of a proper visualization is straightforward, when a well known and highly adopted representation is used, e.g., the notations of the UML [11]. In other cases, a visualization is the essence of a technique because it has to highlight relevant information at a proper level of detail.

In this respect, empirical evaluations with developers appear to be essential in order to assess the usefulness of a software visualization approach. Nevertheless, software visualization tools are rarely empirically assessed by means of users' studies. For example, Wettel et al. [12] conducted a controlled experiment with software professionals to assess the validity of CodeCity [13], namely a 3D software visualization approach based on a city metaphor, where classes are buildings and packages districts. This empirical study was essentially unique in the genre. The main reason for a lack of users' studies in software visualization is related to the high risk of their failure.

In this paper,[1] we show a software visualization approach [14] that provides a mental picture by viewing an object-oriented (OO) software by means of polymetric views [8], i.e., lightweight software visualizations enriched with software metrics. The approach has been implemented in a prototype of a supporting tool, intended as an Eclipse Rich Client Platform (RCP). We named this prototype MetricAttitude [15]. It provides a large-scale understanding of a software system visualizing all classes together and handles class relationships (e.g., delegations and hierarchies). Our

* Corresponding author at: Dipartimento di Informatica Università di Salerno Via Giovanni Paolo II, 132 84084 Fisciano, SA.
E-mail addresses: francese@unisa.it (R. Francese), mrisi@unisa.it (M. Risi), giuseppe.scanniello@unibas.it (G. Scanniello), tortora@unisa.it (G. Tortora).

[1] Please read the paper on-screen or as a color-printed paper version, we make extensive use of color pictures.

http://dx.doi.org/10.1016/j.jvlc.2016.05.001
1045-926X/© 2016 Elsevier Ltd. All rights reserved.

tool also provides a fine-grained representation of individual classes (e.g., the name of the class and its methods). Accordingly, the developer can identify classes with specific characteristics, such as a large number of code lines or the class position in the inheritance tree [15]. To assess the validity of both the approach and MetricAttitude, we have conducted an empirical investigation with students and software professional developers [16]. This investigation was based on questionnaire-based surveys.

This paper is partially based on our previous works, in which we proposed our approach [14] and its preliminary assessment [16]. The main contributions of the current paper can be summarized as follows:

- An improved description of our solutions. In particular, we provide some visualization examples, also specifying steps needed to use MetricAttitude;
- an extended and improved data analysis. For example, we here analyze responses to all the questions in our questionnaire-based surveys, while a subset in [16];
- the discussion of results has been improved and extended also thank to the modification before;
- related work section has been extended by also referring to some commercial software products.

The remainder of the paper is organized as follows. In Section 2, we present results of a review conducted on software visualization approaches and techniques. In Section 3, we describe our approach, while we present MetricAttitude in Section 4. In Section 5, we provide details about our users' studies. In Section 6, we outline obtained results. We conclude with final remarks and possible future directions for our research.

2. Related work

There are a number of approaches/techniques for software visualization based on Synthetic Natural Environment (SNE) (e.g., [17–21]). In the first subsection, we discuss these techniques. Software visualization based on graph representations and on polymetric views have been also proposed [8,22,23]. We present some of them in the second subsection. Due to the number of available software visualization approaches, a deep and exhaustive discussion of related work is not possible and it is also out of the scope of the paper. Price et al. [24] proposed an extensive taxonomy of software visualization tools and approaches. For further details, we redirect the interested reader to that taxonomy.

2.1. SNE based techniques

Among the SNE based techniques and approaches, the city metaphor is one of the most known and used in software visualization (e.g., [17–20]). Wettel and Lanza [13] propose such a kind of metaphor for the comprehension of object-oriented software. In their proposal, classes are represented as buildings and packages as districts. The authors implement the metaphor in the CodeCity tool. Software metrics are mapped onto the size and the type of buildings. The same authors [12] present a controlled experiment to assess the validity of both city metaphor and CodeCity. The experiment was conducted with software professionals. The results indicate that the proposed solutions lead to a statistically significant improvement in terms of task correctness. Results also suggested that task completion time statistically decreases when using CodeCity.

Graham et al. [25] propose a different SNE based approach. In particular, they propose a software where each sun represents a package and planets are classes. Orbits represent the inheritance level of a class within its package. Such metaphor is used to analyze either static or evolving code and to show suspected risk parts of the code.

To visualize the integrated representation of software development processes, Martínez et al. [26] suggest a metaphor based on landscape. This metaphor is conceived to describe a number of aspects related to the development of a software, but it is not focussed on source-code. Therefore, the main difference with respect to our proposal is that source-code is not the subject of software visualization.

Ghandar et al. [27] also present a jigsaw puzzle metaphor. Each component of a software is represented as a piece of a jigsaw puzzle. The surface of a piece is used to graphically show the complexity of software components. The metaphor does not provide a view at class granularity level.

Erra and Scanniello [7] present an approach based on a forest metaphor to ease the comprehension of OO software. A software is depicted as a forest of trees. Each tree is a class. Trunks, branches, leaves, and their color indicate characteristic of classes (e.g., a method is a branch of the tree that in turns represents a class). CodeTrees is the name of the tool prototype that implements the metaphor. To improve realistic aspect of trees, the authors exploited and adapted the Weber and Penn approach [28]. Successively, the same authors extended CodeTrees to allow the visualization of evolving software [29]. The same authors have recently proposed a novel SNE based metaphor that takes advantages of concepts such as archipelagos, atolls, and palms [30]. Each package is represented as an atoll, while palms on it graphically depict salient information of classes contained in the package associated to that atoll. The entire software is represented as a set go atolls. Authors speculated that this metaphor can be considered simpler than that implemented in CodeTree for naive users and more pleasant than that implemented in CodeCity. The validity of this new metaphor is not assessed through an empirical assessment with actual users.

A botanical tree metaphor is also proposed in [31]. The authors suggest forests of trees for the visualization of huge hierarchical structures and apply the proposed metaphor to the visualization of directory structures. Directories, files, and their relations are visualized using trees. The approach is basically a natural visual metaphor for information hierarchically structured.

Gall and Jazayeri [32] show a 3D visual representation for analyzing software release histories. The approach is based on a retrospective analysis technique to evaluate architectural stability, based on the use of colors to depict changes in different releases. Differently, Tu and Godfrey [33] propose an approach that makes an integrated use of software metrics, visualization, and origin analysis. Girba et al. [34] suggest an approach based on the notion of history to analyze how changes appear in software. The authors propose a tool for visualizing histories of evolving class hierarchies. The main difference between these approaches and ours is that we are able to show both large- and low- scale understanding of software.

Several are the differences among our proposal and approaches discussed before. The most remarkable one is that our approach enriches lightweight visualizations (i.e., polymetric view) with metrics information and relationships among classes. Relationships are statically inferred to approximate run-time types of class receivers. This represents another remarkable difference between our proposal and visualization approaches introduced before. In addition, we perform users' studies with both students and professional software developers. These studies in the large part qualitative even if quantitative information have been gathered and analyzed.