Software Impacts 16 (2023) 100491

Contents lists available at ScienceDirect

Software Impacts

journal homepage: www.journals.elsevier.com/software-impacts

Original software publication

Annotation Visualizer: A software visualization tool for code annotations

Phyllipe Lima [a,*], Nathalya Stefhany Pereira [b], Everaldo Gomes [c], Eduardo Guerra [d], Paulo Meirelles [c]

[a] Federal University of Itajubá - IMC - UNIFEI, Brazil
[b] National Institute for Telecommunications – Inatel, Brazil
[c] University of São Paulo – IME-USP, Brazil
[d] Free University of Bolzano-Bolzen – UniBZ, Italy

ARTICLE INFO

Keywords:
Code annotations
Circle packing
Web application
Software visualization
Open-source tool

ABSTRACT

The Annotation Visualizer (AVisualizer) is a software visualization tool for displaying code annotations distribution in a given target Java-based software system. Implemented as a web application, it can extract annotations usage from the target source code and display it using a hierarchical circle packing approach. Using a dedicated suite of software metrics, it can display size-related information and code responsibilities associated with annotations usage. The tool provides three different views of the analyzed system, each with different granularity. The AVisualizer is a tool that helps improve code comprehension.

Code metadata

Current code version	v1.0.0
Permanent link to code/repository used for this code version	https://github.com/SoftwareImpacts/SIMPAC-2023-73
Permanent link to Reproducible Capsule	https://codeocean.com/capsule/6899091/tree/v1
Legal Code License	GNU Affero General Public License version 3 (AGPL-3.0)
Code versioning system used	git
Software code languages, tools, and services used	TypeScript, JavaScript, Java, JDK 11
Compilation requirements, operating environments & dependencies	JRE 11 or higher, Node v16.10.0, Yarn v1.22.17, Maven
If available Link to developer documentation/manual	https://github.com/metaisbeta/avisualizer#readme
Support email for questions	phyllipe@unifei.edu.br

1. Introduction

Code annotations are Java programming language feature to configure custom metadata directly on programming elements, such as methods and classes. Since annotations are inserted directly into the source code, they are a convenient and quick alternative to configure metadata. Our study performed in Java open source projects [1] identified at least one code annotation in 78% of the classes, reinforcing their popularity. Furthermore, our companion work [2] demonstrated that there is a relationship between code responsibility and code annotations.

In this context, the Annotation Visualizer (AVisualizer) is a tool that implements the CADV (Code Annotations Distribution Visualization), a software visualization approach for code annotations defined and validated in our companion work [2]. The CADV uses a circle-packing approach to display code annotations usage and distribution in a Java system under analysis, as well as the underlying hierarchical structure of packages and classes. It also uses colors to differentiate the annotations by their package and groups the code annotations according to the structure of the system under analysis.

DOI of original article: https://doi.org/10.1016/j.infsof.2022.107089.
The code (and data) in this article has been certified as Reproducible by Code Ocean: (https://codeocean.com/). More information on the Reproducibility Badge Initiative is available at https://www.elsevier.com/physical-sciences-and-engineering/computer-science/journals.
* Corresponding author.
E-mail addresses: phyllipe@unifei.edu.br (P. Lima), nathalya.stefhany@gec.inatel.br (N.S. Pereira), everaldogjr@gmail.com (E. Gomes), eduardo.guerra@unibz.it (E. Guerra), paulormm@ime.usp.br (P. Meirelles).

https://doi.org/10.1016/j.simpa.2023.100491
Received 14 February 2023; Accepted 6 March 2023

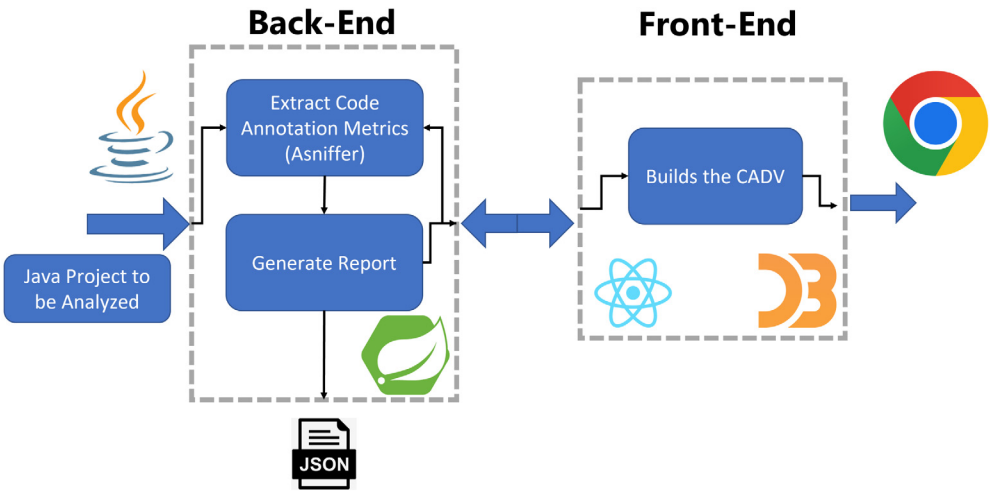

Fig. 1. AVisualizer basic building blocks.

```java
import org.junit.After;
import org.junit.Before;
import org.junit.Test;

public class TestClass {
    @Before
    public void setUp(){ //initializations }
    @Test
    public void testMethod(){ //Execute tests }
    @After
    public void cleanTest(){ //clear resources allocated during initialization }
}
```

Fig. 2. Example with *org.junit* annotation schema.

The work of Hasselbring et al. [3] comments that developers spend most of their time comprehending software before adding and maintaining features. To overcome this, one of the main goals of the CADV is to aid in software comprehension through software visualization. We implemented the AVisualizer as a web application. We also extended and developed a plugin integrated with the IntelliJ IDE [4], a popular Java development environment.

Even though code annotations are a popular feature, we could not find any software visualization approaches or tools focusing on them. The AVisualizer was developed to implement and validate the CADV approach in our companion work [2]. All empirical evaluations conducted for this approach used the AVisualizer web application.

2. Software description

The AVisualizer is a web application divided into two main components: (i) the backend responsible for extracting annotation metrics values from the target project; and (ii) the frontend responsible for rendering the circle packing (the CADV approach) using these metrics values. To render the circle packing, we used the D3.js JavaScript library [5]. Fig. 1 presents the basic building blocks of the AVisualizer. The backend is wrapped as a SpringBoot application, and the frontend is a React application. We configured a maven build to generate a single executable jar file with the application. Information about the build process is available in the public repository.[1]

2.1. Code annotation metrics extraction

The backend is mainly composed of a Java tool named Annotation Sniffer (ASniffer) [6], seen in the left part of Fig. 1. Our research group developed the ASniffer to collect code annotation metrics from Java source code. These are a suite of 7 metrics that extract code annotations size, complexity, coupling, and annotation schema. They were defined and validated in our previous work [1]. The ASniffer is an independent component used as a dependency in the backend of the AVisualizer.

An annotation schema of a code annotation is an important concept since it greatly influences the frontend of the AVisualizer tool. An annotation-based API, or metadata-based framework, define and expose a set of annotations so application developers can use them to configure programming elements and execute the desired behavior. This set of annotations that define a metadata structure of a given domain for an API is defined as **annotation schema** [1]. The code in Fig. 2 presents a Java class responsible for executing unit tests using the JUnit 4 framework. The annotations `@Test`, `@After`, `@Before` belong to the package `org.junit`. In this case, we refer to this annotation set as part of the `org.junit` annotation schema. In practical terms, an automated approach for extracting the annotation schemas is to identify imported packages for annotations. Another example of a popular annotation schema is the `javax.persistence`.

2.2. CADV rendering

After extracting the metrics values in the backend, the tool generates a JSON report suitable to serve the frontend module, the right part of Fig. 1. Developed using the React library and D3.Js, it reads this JSON report and builds the CADV visualization.

The CADV comprises three different software visualization views, each displaying the software under analysis in different granularity and scope. The (i) **System View** displays the complete system and annotations distributed by packages. The (ii) **Package View** displays a single package (and nested packages) with annotations distributed in the classes of the displayed package. Finally, the (iii) **Class View**

[1] https://github.com/metaisbeta/avisualizer

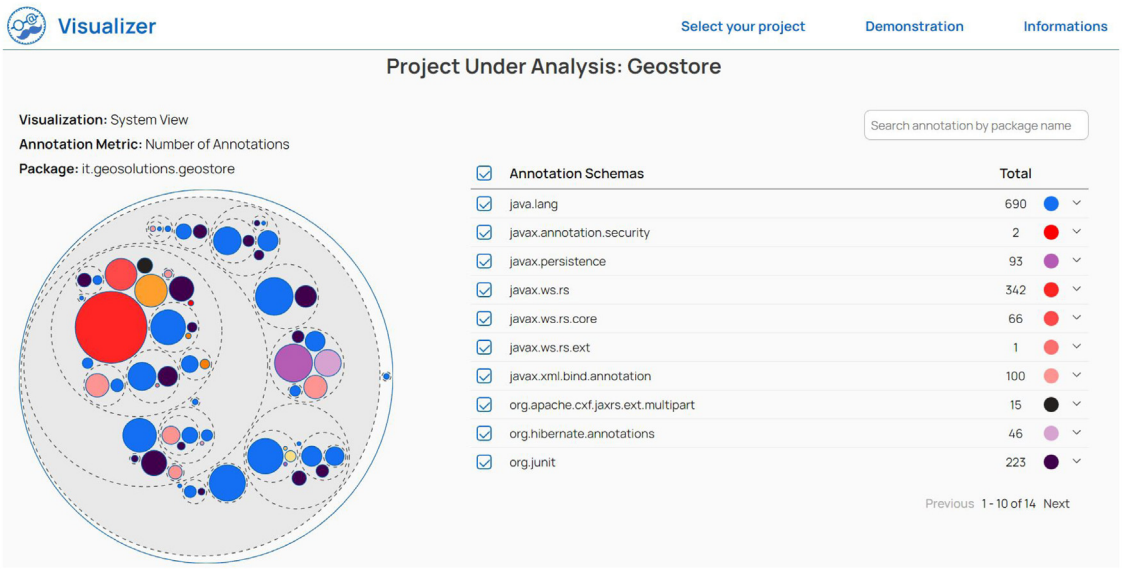

Fig. 3. AVisualizer application rendering geostore project.

displays a single class and the distribution of annotations in the code elements within this class. The views use colors to differentiate the type and grouping of annotations, i.e., the annotation schema. The size of these circles follows a chosen code annotation metric value. Our companion paper [2] contains the details and design of the CADV approach, as well as the evaluation carried out.

When executed, the AVisualizer presents the user with an example project under analysis, shown in Fig. 3. The default is the Geostore[2] project. The navigation bar has a *Demonstration* option with two other projects the user can display. The AVisualizer has three main areas that convey the information to the user: (i) The **Header**, (ii) The **View**, (iii) and The **Annotation Schema Table**.

The **Header** contains information to guide the user on what it is seeing and what part of the project it is currently inspecting. It informs what package/class is being displayed, what annotation metric is being used to determine the circle size, and which of the three views of CADV are currently being rendered, i.e., the **System**, **Packge** or **Class** View.

The **View** is the area that displays the actual visualization, being located below the **Header**. The **System View** is the visualization open as a default. Each dashed-line circle represents a package from the source code, and each colored circle represents an annotation, or group of annotations, from a specific annotation schema. Each annotation schema is assigned a color. The user can click on packages (dashed-line circles) or annotations (colored circles) to zoom in. During the zoom process, the visualization is changed to either the **Package View** or **Class View**, depending on how far the user is currently exploring. As mentioned, each view has different characteristics that complement each other for completeness in the visualization process. The AVisualizer tool allows navigating between all three views.

The **Annotation Schema Table** is a table with the annotation schemas found in the project under analysis. It displays the colors assigned to each annotation schema and the total number of annotations from that annotation schema in the project. The tool allows users to select only the annotation schemas they wish to visualize.

The user can visualize their own project by clicking on “Select your project” and informing the path to where the project resides on the local machine. It currently requires a Java project as input, and having a maven or gradle build file is not mandatory.

[2] https://github.com/geosolutions-it/geostore

3. Impact

As shown by our previous work [1,6], companion work [2] as well as others in the literature [7,8], code annotations are a popular feature, actively maintained, and helps to understand the role or responsibility of a class or package. Furthermore, languages such as Kotlin, C#, and TypeScript have similar features, reinforcing that this is a popular programming style not only for Java developers.

However, as mentioned, we could not find a tool or software visualization approach that targets the comprehension of how code annotations are used in a project. The visualizations in the literature focus on displaying size, complexity, and cohesion, with the types (classes, interfaces, enums) representing the main elements. Our approach using colors to identify and differentiate annotation schemas, according to our empirical study results [2], allowed developers to identify code responsibilities and architectural roles, not just size or complexity.

The AVisualizer tool is essential to our companion work and helped evaluate the benefits of visualizing code annotations. With the AVisualizer, we could evaluate that visualizing the annotations helps developers further improve their system knowledge, which helps maintenance and evolution. In short, the AVisualizer helped us achieve our primary goal of improving software comprehension through software visualization, focusing on an important language feature neglected by other tools.

4. Limitations and future improvements

The AVisualizer is focused on displaying code annotations, so it is suitable for Java systems that use APIs based on them. Even though they are popular, some Java projects do not use them. In this case, the AVisualizer will not contribute to the comprehension process. On the other hand, web applications are an example of applications that would benefit from our tool since they are notorious for extensively using annotations-based frameworks and APIs, such as Spring, JPA, CDI, etc.

The AVisualizer tool is an ongoing work with different types of limitations. The first one is related to the rendering of the **Class View**, which presents several low-level details of the class. This limitation comes from the CADV approach and not the tool itself. However, the tool can minimize such limitations by allowing users to choose the information they want in this view. In short, the tool should provide

more customization. The other limitation is that there is no way to navigate from the visualization to the source code directly. We will tackle this from the plugin version of AVisualizer available for IntelliJ IDE.

5. Publications enabled by the tool AVisualizer

- CADV: A software visualization approach for code annotations distribution. *Information and Software Technology*. 2023. DOI: https://doi.org/10.1016/j.infsof.2022.107089

CRediT authorship contribution statement

Phyllipe Lima: Conceptualization, Methodology, Software, Validation, Investigation, Writing – original draft, Writing – review & editing. **Nathalya Stefhany Pereira:** Software. **Everaldo Gomes:** Software. **Eduardo Guerra:** Conceptualization, Writing – review & editing. **Paulo Meirelles:** Conceptualization, Methodology, Writing – original draft, Writing – review & editing, Funding acquisition.

Acknowledgments

We would like to thank the support granted by Brazilian funding agency FAPESP (São Paulo Research Foundation) grant 2019/12743-4.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Appendix A. Supplementary data

Supplementary material related to this article can be found online at https://doi.org/10.1016/j.simpa.2023.100491.

References

[1] P. Lima, E. Guerra, P. Meirelles, L. Kanashiro, H. Silva, F. Silveira, A Metrics Suite for code annotation assessment, J. Syst. Softw. 137 (2018) 163–183, http://dx.doi.org/10.1016/j.jss.2017.11.024, URL http://www.sciencedirect.com/science/article/pii/S016412121730273X.

[2] P. Lima, J. Melegati, E. Gomes, N.S. Pereira, E. Guerra, P. Meirelles, CADV: A software visualization approach for code annotations distribution, Inf. Softw. Technol. 154 (2023) 107089, http://dx.doi.org/10.1016/j.infsof.2022.107089, URL https://www.sciencedirect.com/science/article/pii/S0950584922001987.

[3] W. Hasselbring, A. Krause, C. Zirkelbach, ExplorViz: research on software visualization, comprehension and collaboration, Softw. Impacts 6 (2020) 100034, http://dx.doi.org/10.1016/j.simpa.2020.100034, URL https://www.sciencedirect.com/science/article/pii/S2665963820300257.

[4] S. Abilio, P. Lima, E. Gomes, E. Guerra, P. Meirelles, Annotation Visualizer Plugin: An IDE-Integrated Tool for Code Annotations Visualization, Revis. de Sistemas de Inf. da FSMA (30) (2022) 28–37, URL http://www.fsma.edu.br/si/edicao30/FSMA_SI_2022_2_01_Guerra_en.html.

[5] M. Bostock, V. Ogievetsky, J. Heer, D3: Data-driven documents, IEEE Trans. Vis. Comput. Graphics 17 (12) (2011) 2301–2309, http://dx.doi.org/10.1109/TVCG.2011.185.

[6] P. Lima, E. Guerra, P. Meirelles, Annotation sniffer: a tool to extract code annotations metrics, J. Open Source Softw. 5 (47) (2020) 1960, http://dx.doi.org/10.21105/joss.01960.

[7] M. Aniche, G. Bavota, C. Treude, M. Gerosa, A. van Deursen, Code smells for model-view-controller architectures, Empir. Softw. Eng. J, (2017) http://dx.doi.org/10.1007/s10664-017-9540-2.

[8] Z. Yu, C. Bai, L. Seinturier, M. Monperrus, Characterizing the usage, evolution and impact of java annotations in practice, IEEE Trans. Softw. Eng. 47 (5) (2021) 969–986, http://dx.doi.org/10.1109/TSE.2019.2910516.