DiffViz: A Diff Algorithm Independent Visualization Tool for Edit Scripts

Veit Frick, Christoph Wedenig, and Martin Pinzger
Software Engineering Research Group
Alpen-Adria-Universität Klagenfurt
Email: {veit.frick, christoph.wedenig, martin.pinzger}@aau.at

***Abstract*—A number of approaches and tools exist that extract and visualize the changes between two versions of a file and thereby help developers to understand them. DiffViz is an interactive visualization tool that visualizes the changes independent from the differencing algorithm. It supports, but is not limited to, a granularity on the level of abstract syntax trees. Furthermore, it provides several new features, such as node matching and the mini-map, to navigate and analyze the changes.**
A demo of the installation and example usage of the tool is available here: https://youtu.be/RF93ey9GYoc

I. Introduction

Most software systems are evolving and changing systems. To understand the changes made to such systems it is necessary to understand the changes made to the underlying source code. Instead of simply comparing two versions of the same source file, differencing algorithms can be used to analyze them and detect the changes made.

There exist a multitude of differencing algorithms, focusing on different programming and markup languages and using different approaches to generate edit scripts. Textual differencing algorithms, such as the commonly used diff, based on the algorithm presented by Hunt *et al.* [8] and its modifications [10] [11], compare two files on a purely textual level and are capable of detecting inserted and deleted lines of text. Algorithms like GumTree [4] and ChangeDistiller [5] use the AST representation of source code instead of the text of the source files as their basis. Hence, they output edit scripts consisting of changed nodes in the ASTs and are therefore able to compute more fine-grained results.

Edit scripts alone do not necessarily further the understanding of changes, their visualization plays an important factor as well. Many differencing approaches also provide tools that are able to visualize the resulting edit scripts by marking the edits in the corresponding source and destination files. GitHub, for example, provides a web-view, highlighting changes between two versions of a file, using a variant of diff. GumTree provides a tool that is able to produce a web-view of the edit scripts generated by the GumTree differencing algorithm.

DiffViz is an edit script visualization tool that provides the user with an easy-to-use and comprehensive way of generating and analyzing edit scripts. In comparison to existing tools, DiffViz is not tied to just one differencing algorithm but can be used to visualize edit scripts generated by any differencing algorithm that produces an output that can be converted into the format presented in Section III. As provided, DiffViz already includes the differencing algorithms GumTree, MTDIFF [3], and IJM [6], and was also tested with BuildDiffer [9]. Given the respective differencing algorithm, DiffViz can be used to visualize changes in any kind of text (*e.g.*, Source Code, XML, prose). DiffViz is Open Source and can be found on GitHub (github.com/W3D3/DiffVisualizer).

The remainder of this paper is structured as follows: Section II describes DiffViz and how it can be used. Section III gives an overview of the architecture and the requirements of DiffViz. In Section IV DiffViz is compared to other visualization approaches. Section V presents the related work. Finally, Section VI concludes this paper.

II. DiffViz

As edit script visualization tool, DiffViz highlights the changes between two versions of a file (in the following named diff-pair). As shown in Figure 1 DiffViz compares both versions of the diff-pair side-by-side. The left side shows the older version and the right side shows the newer version. DiffViz depicts changes in the diff-pair with colored markups. Changes that are present in both versions of the diff-pair (*e.g.*, updated nodes) are visually connected as explained later.

To analyze a diff-pair, the user has to provide DiffViz with the respective source files. This can be done in three different ways. First, there is the integrated editor which can be used to provide source code manually (Figure 1-①). Second, diff-pairs can be imported from GitHub using a wizard (Figure 1-②). As a third option, a JSON file that references one or more GitHub diff-pairs that are uniquely identified by username, project name, commit hash, parent-commit hash, filename, and parent-filename can be uploaded to import many diff-pairs at once (Figure 1-③). Every imported diff-pair is listed in the sidebar for the user to load (Figure 1-④). The whole list can be downloaded as a JSON file.

After importing, the user can choose a specific edit script from the sidebar to inspect. The source code is fetched through a proxy and sent to the backend where the selected differencing algorithm is applied. The backend and the algorithm can be changed at any time in the settings (Figure 1-⑤).

DiffViz receives the result and shows source (Figure 1-⑥) and destination (Figure 1-⑦) side-by-side with all the changes visualized as inline nodes. DiffViz implements further features for improved code navigation, such as:

- **Node Matching:** Updated and moved nodes have matching nodes in the source and the destination file. If the user selects such a node, DiffViz automatically displays the respective matching node on the opposite side and highlights both nodes.
- **Node Highlighting:** Nodes can be hard to distinguish from each other. For this reason, nodes that have been selected by the user are highlighted with a solid border and shadow. Nodes that the user hovers over are rendered with a dotted border. This makes it easier to pick the desired one.
- **Meta-Data:** The backend can return metadata for every node. For example, IJM supplies a node type description as well as the containing method and class. The user can inspect this data by double-clicking on a node.
- **Mini-Map:** When browsing an edit script with few or small changes that are spread across the file it can be hard to spot all the nodes without scrolling through the whole file. DiffViz provides a mini-map that enhances the scrollbar with a color encoded representation of all nodes and the currently visible area of the file to help the user navigate the changes. The mini-map can be seen in (Figure 1-⑧) .
- **Editor:** DiffViz offers the possibility to make changes to the already imported source code on the fly with the editor (Figure 1-①). The modified version is then diffed again.
- **Filter:** Focusing on a specific change type in a file with a high variety of change types can make it easier to find relevant changes. To achieve this, DiffViz offers an option to filter nodes based on their change type.
- **Syntax Highlighting:** Highlighted language keywords make the code easier to read. DiffViz tries to detect the programming language automatically and uses highlight.js[1] for syntax highlighting.
- **Jump to Line:** To help with navigating huge files there is a jump-to-line feature for the case when the user already knows where to look.
- **Code Search:** DiffViz overrides the default browser search with a custom one that only searches the source code. The user has the option of searching both revisions or to search only one explicitly.

A. *Bug fix scenario*

This example shows how DiffViz can be used to detect bugs in the Java class `Loops`. The class implements some simple array operations: sorting and printing an array in reverse. Since the latest commit, the program crashes when trying to run `doSomething()`. The developer therefore decides to analyze the commit using DiffViz and selects a suitable differencing algorithm. In this case, GumTree has been chosen.

The user is then presented with a change overview that is shown in Figure 1. Now that all the changes are highlighted, the user can reduce the scope of the potentially faulty code. If the bug was added with the commit, it was most likely introduced through one or more of the changes.

The left code panel of the DiffViz in Figure 1 shows that the whole `for`-loop used for reverse printing the array has been moved. Clicking on any moved node highlights the corresponding matching node as well. It is therefore possible to see that the loop got moved into the method `loopReverse()`.

The only node that was not moved from the original `for`-loop inside `loopReverse()` is the infix expression `this.arr.length - 1`. It got deleted (red background color) and replaced with `this.arr.length`. This uncovers the bug since this change leads to an `ArrayOutOfBounds` exception.

When viewing this commit in the traditional line-based diff view that GitHub provides as seen in Figure 2 the changes get interpreted as just four inserts and four deletes. Due to the long distance of the move, GitHub will not show us character-based differencing which makes it harder to spot that only the condition inside the `for`-loop got altered. While this specific bug would be easy to spot in reality without any tools, it is still exemplary for how DiffViz can improve the understandability of an edit script.

III. Architecture

The DiffViz project is composed of different modules for the sake of interchangeability and encapsulation.

A. *Backend*

The backend is a Java RESTful web-service, providing the source code analysis and consequently the data for the front-end to visualize. DiffViz's currently used backend includes GumTree [4], MTDIFF [3] and IJM [6] as differencing algorithms. However, this backend and its differencing algorithms are interchangeable and DiffViz can use any data source which complies with the following requirements:

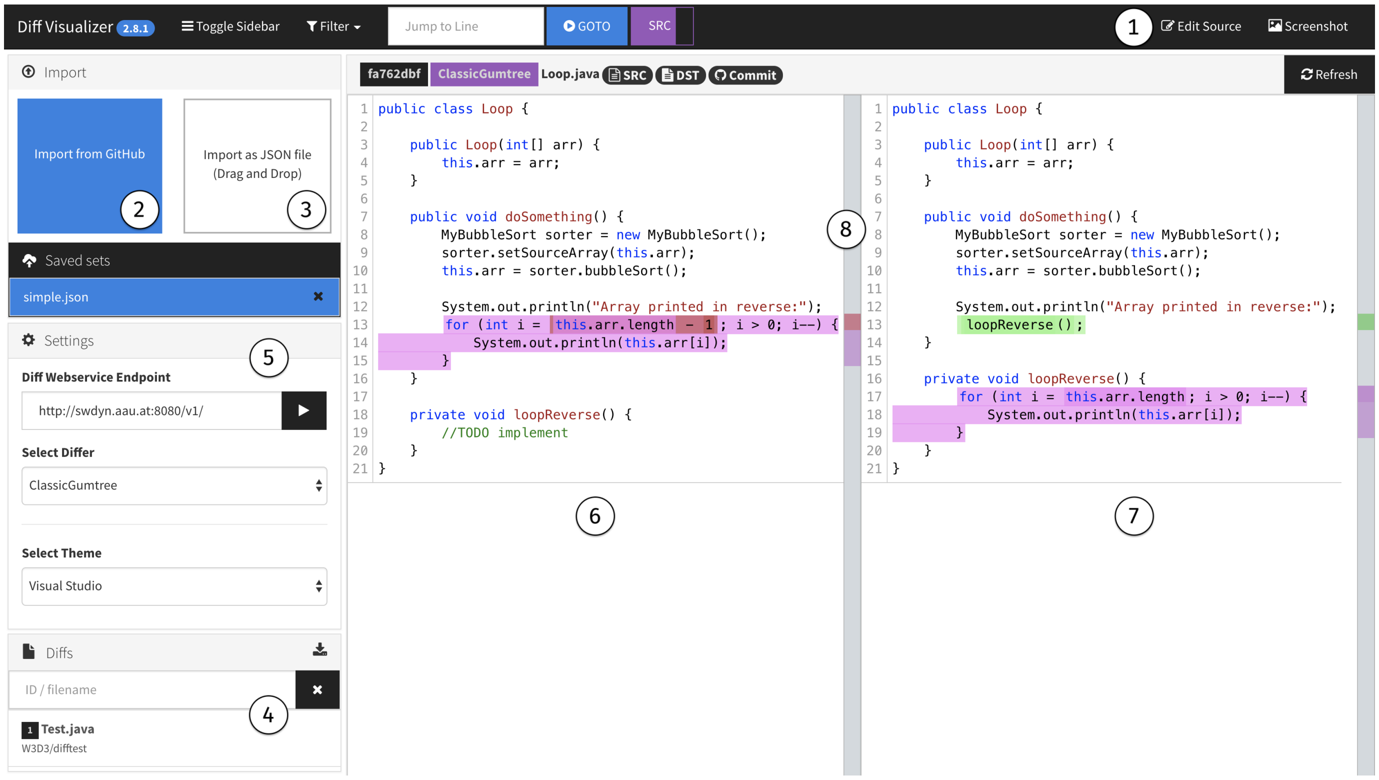
- Supplies a list of available matchers where every matcher has a name and a unique ID for further identification.
- Supplies all the nodes for a given source and destination source-code in combination with a unique matcher ID. A node can have the types INSERT, DELETE, UPDATE, MOVE and META. UPDATE and MOVE nodes need to have a corresponding node in both versions. META nodes can be used for additional highlighting. Every node needs to contain the relative start/end positions in the source/destination file (line number and offset). Nodes need to be properly nested and cannot be partially overlapping. Every node can contain a string of metadata.

B. *Client*

The client is written in ECMAScript 6. It aims to make it efficient and intuitive to interact with the backend. It uses Bootstrap 3 and jQuery for the UI. Microsoft's Monaco editor[2] is used for the code editor.
Together with the client, DiffViz also includes a small utility

[1]http://highlightjs.org/

[2]https://microsoft.github.io/monaco-editor/



simple.json

Diff Webservice Endpoint: http://swdyn.aau.at:8080/v1/

Select Differ: ClassicGumtree

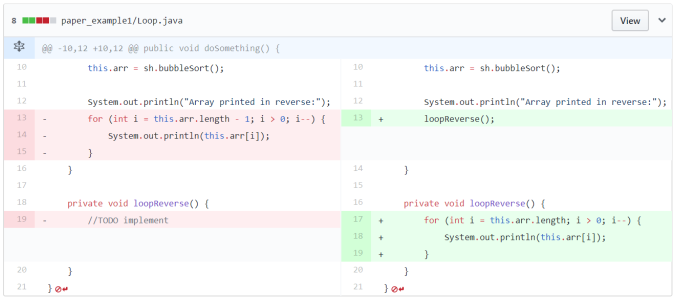
Select Theme: Visual Studio

Test.java
W3D3/difftest

```java
public class Loop {

    public Loop(int[] arr) {
        this.arr = arr;
    }

    public void doSomething() {
        MyBubbleSort sorter = new MyBubbleSort();
        sorter.setSourceArray(this.arr);
        this.arr = sorter.bubbleSort();

        System.out.println("Array printed in reverse:");
        for (int i = this.arr.length - 1; i > 0; i--) {
            System.out.println(this.arr[i]);
        }
    }

    private void loopReverse() {
        //TODO implement
    }
}
```

```java
public class Loop {

    public Loop(int[] arr) {
        this.arr = arr;
    }

    public void doSomething() {
        MyBubbleSort sorter = new MyBubbleSort();
        sorter.setSourceArray(this.arr);
        this.arr = sorter.bubbleSort();

        System.out.println("Array printed in reverse:");
        loopReverse();
    }

    private void loopReverse() {
        for (int i = this.arr.length; i > 0; i--) {
            System.out.println(this.arr[i]);
        }
    }
}
```

Fig. 1. Screenshot of DiffViz showing two versions of the Java class `Loop` where a bug was introduced in the newer version



```
@@ -10,12 +10,12 @@ public void doSomething() {
10         this.arr = sh.bubbleSort();                           10         this.arr = sh.bubbleSort();
11                                                               11
12         System.out.println("Array printed in reverse:");      12         System.out.println("Array printed in reverse:");
13 -       for (int i = this.arr.length - 1; i > 0; i--) {       13 +       loopReverse();
14 -           System.out.println(this.arr[i]);
15 -       }
16     }                                                         14     }
17                                                               15
18     private void loopReverse() {                              16     private void loopReverse() {
19 -       //TODO implement                                      17 +       for (int i = this.arr.length; i > 0; i--) {
                                                                 18 +           System.out.println(this.arr[i]);
                                                                 19 +       }
20     }                                                         20     }
21 }                                                            21 }
```

Fig. 2. GitHub diff of the Java `Loop` class

web-server. This web server hosts the client and is used to up- and download JSON files containing diff-pairs that are validated in this process. It also acts as a middleman between the GitHub REST API v3 to avoid exposing the login credentials to the client. It also proxies requests to the GitHub RAW CDN to circumvent CORS restrictions.

IV. Comparison

In this section we compare DiffViz to three other freely available and web-based visualization tools.

1) The GitHub diff visualization (GHDV) is based on diff. diff is a line-based differencing algorithm, which does not provide update or move actions. GHDV does, however, highlight the parts of the line that have been added or deleted. Due to its use of diff, the GHDV is language independent and available for all text revisions uploaded to GitHub.
2) The GumTree Webdiff tool (GTWD) is a web-service that visualizes edit scripts produced by GumTree. Hence, GTWD is able to show changes on a fine-grained level and supports move and update actions. GTWD shows the different actions in different colors and is able to highlight corresponding moved or updated nodes when the user clicks on them.
3) Mergely[3] is an open-source visualization tool that builds on diff as well. It is therefore line-based and does not support move or update actions but is language independent. It does not only serve as a diff visualization tool but can also be used to directly merge two files.

A. Usage Scenario

In Section II we described a bug finding usage scenario and explained the steps necessary in DiffViz to identify the changes that caused the bug. In the following, we provide an overview of how the same scenario would look like using GHDV, GTWD, and Mergely and compare the differences to our approach.

If the commits in question are already on GitHub, GHDV is a fast way to get a grasp on what changed. The user has to open the commit on the GitHub website. They are then presented with a line-based differencing view of the `Loop` class, shown in Figure 1. Deleted and added lines are visually

[3]http://www.mergely.com/

highlighted and can therefore help finding the bug, however, an indication that some of the lines are moved or partially updated is missing.

Mergely allows the user to directly copy-and-paste the two source files into their online editor. It then presents the user with an indication what lines have been added or deleted, similar to GHDV. An indication that some of the lines are moved or partially updated is missing as well.

GTWD is started from a command line, taking both files as arguments. It then starts a web-server that displays a side-by-side comparison of both source files. It highlights inserted, moved, updated, and deleted nodes in different colors. When clicking on a moved or updated node, the corresponding node in the other source file is highlighted as well. This representation allows the user to quickly find the bug, as described in Section II.

DiffViz, GHDV and Mergely are all easy to use and require only little effort to display the file differences. However, the line based nature of GHDV and Mergely may not be sufficient for a detailed analysis. GTDV on the other hand provides a detailed analysis but requires some effort to use and lacks further features to navigate the changes, as can be seen in Table I. DiffViz is easy to use and can provide a detailed analysis.

B. Feature Comparison

Table I compares the features provided by all four tools. It is important to note, that the four tools can be used for different scenarios, Mergely for example offers additional merge features that are not discussed here. In our selection of features, we focused on features that help with understanding the changes made to two versions of a source file. It is not a complete list of all features but is only aimed at providing an overview. Table I shows that DiffViz provides many features, aimed at helping developers understand changes, that are not provided by other visualization tools. It is also the only differencing tool that allows choosing between multiple differencing algorithms. We therefore think that DiffViz provides a useful set of features that can help developers to understand complex changes.

TABLE I
FEATURE COMPARISON OF THE DIFF VISUALIZATIONS BY GITHUB, GUMTREE, MERGELY, AND DIFFVIZ

Feature	GitHub	GumTree	Mergely	DiffViz
Source Highlighting	Yes	No	No	Yes
Moves & Updates	No	Yes	No	Yes
Search Function	Browser	Browser	Custom	Custom
Mini Map	No	No	Yes	Yes
Unchanged Code Collapse	Yes	No	No	No
Commenting	Yes	No	No	No
Code Editor	No	No	Yes	Yes
Side-by-Side view	Yes	Yes	Yes	Yes
Scroll to Node	No	No	No	Yes
Node Metadata	No	No	No	Yes
Multiple Differs	No	No	No	Yes
Filter Function	No	No	No	Yes
Jump to Line	No	No	No	Yes
Diff Summary	Yes	No	No	On import

V. RELATED WORK

Many different approaches have been developed to visualize the historical data of version control repositories. Voinea *et al.* [16] present an open framework to mine and visualize historical data from CVS repositories. Gource [2] and code_swarm [12] both generate animated histories of repositories. Ogawa *et al.* [13] present an approach to visualize the interaction between different developers of a repository in the form of software evolution storylines. Aghajani *et al.* [1] present an IDE plugin to explore the history of changes in a project. It is used to find interesting points in the history of a file using metrics from the version control system. The Chronos tool by Servant *et al.* [14] allows for the visualization of the history of specific lines of code.

In addition to the visualization of whole repositories, there also exist approaches to visualize the changes made by specific commits or revisions. With Torch, Gómez *et al.* [7] present a visualization tool that combines text-based diffs with metrics and a visual representation to support integration decisions. Telea *et al.* present Code Flows [15], a method to visualize the evolution of source code geared to the understanding of fine and mid-level scale changes across several file versions. With Chronicler, Wittenhagen *et al.* [17] present a history-graph base visualization of the evolution of code elements. Yoon *et al.* [18] present a tool to visualize fine grained code changes. DiffViz differs from these existing approaches in several ways: firstly, it visualizes the changes at a much more detailed level, namely the AST; secondly, it allows to switch the underlying differencing algorithm; finally, it provides several new features to browse and navigate the changes.

VI. CONCLUSION

In this paper we presented DiffViz, a tool to visualize code changes. DiffViz works with a variety of differencing algorithms and can be extended to use any approach as long as they conform with the requirements specified in Section III. We explain how DiffViz is built and how it can be used. We also compare it to three other visualization approaches (GitHub, Mergely, and GumTree), highlight their differences, and show the benefits of DiffViz.

ACKNOWLEDGEMENT

This work has been funded by the Austrian Science Fund (FWF) under project number 2753-N33.

REFERENCES

[1] E. Aghajani, A. Mocci, G. Bavota, and M. Lanza. The code time machine. In *Proceedings of the 25th International Conference on Program Comprehension*, ICPC '17, pages 356–359, Piscataway, NJ, USA, 2017. IEEE Press.

[2] A. H. Caudwell. Gource: Visualizing software version control history. In *Proceedings of the ACM International Conference Companion on Object Oriented Programming Systems Languages and Applications Companion*, OOPSLA '10, pages 73–74, New York, NY, USA, 2010. ACM.

[3] G. Dotzler and M. Philippsen. Move-optimized source code tree differencing. In *2016 31st IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 660–671, Sept 2016.

[4] J.-R. Falleri, F. Morandat, X. Blanc, M. Martinez, and M. Monperrus. Fine-grained and accurate source code differencing. In *Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering*, ASE '14, pages 313–324, New York, NY, USA, 2014. ACM.

[5] B. Fluri, M. Wuersch, M. Pinzger, and H. Gall. Change distilling:tree differencing for fine-grained source code change extraction. *IEEE Transactions on Software Engineering*, 33(11):725–743, Nov 2007.

[6] V. Frick, C. Wedenig, and M. Pinzger. Generating accurate and compact edit scripts using tree differencing. In *2018 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, in press.

[7] V. U. Gmez, S. Ducasse, and T. DHondt. Visually characterizing source code changes. *Science of Computer Programming*, 98:376 – 393, 2015. Special Issue on Advances in Dynamic Languages.

[8] J. W. Hunt and M. Douglas McIlroy. An algorithm for differential file comparison. 1975.

[9] C. Macho, S. Mcintosh, and M. Pinzger. Extracting build changes with builddiff. In *Proceedings of the 14th International Conference on Mining Software Repositories*, MSR '17, pages 368–378, Piscataway, NJ, USA, 2017. IEEE Press.

[10] W. Miller and E. W. Myers. A file comparison program. *Software: Practice and Experience*, 15(11):1025–1040, 1985.

[11] E. W. Myers. An O(ND) difference algorithm and its variations. *Algorithmica*, 1(1-4):251–266, Nov 1986.

[12] M. Ogawa and K. L. Ma. code_swarm: A design study in organic software visualization. *IEEE Transactions on Visualization and Computer Graphics*, 15(6):1097–1104, Nov 2009.

[13] M. Ogawa and K.-L. Ma. Software evolution storylines. In *Proceedings of the 5th International Symposium on Software Visualization*, SOFTVIS '10, pages 35–42, New York, NY, USA, 2010. ACM.

[14] F. Servant and J. A. Jones. Chronos: Visualizing slices of source-code history. In *2013 First IEEE Working Conference on Software Visualization (VISSOFT)*, pages 1–4, Sept 2013.

[15] A. Telea and D. Auber. Code flows: Visualizing structural evolution of source code. In *Proceedings of the 10th Joint Eurographics / IEEE - VGTC Conference on Visualization*, EuroVis'08, pages 831–838, Chichester, UK, 2008. The Eurographs Association &#38; John Wiley &#38; Sons, Ltd.

[16] L. Voinea and A. Telea. An open framework for cvs repository querying, analysis and visualization. In *Proceedings of the 2006 International Workshop on Mining Software Repositories*, MSR '06, pages 33–39, New York, NY, USA, 2006. ACM.

[17] M. Wittenhagen, C. Cherek, and J. Borchers. Chronicler: Interactive exploration of source code history. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, CHI '16, pages 3522–3532, New York, NY, USA, 2016. ACM.

[18] Y. Yoon, B. A. Myers, and S. Koo. Visualization of fine-grained code change history. In *2013 IEEE Symposium on Visual Languages and Human Centric Computing*, pages 119–126, Sept 2013.