RepoVis Timeline Extension

Amel Hamidovic, Jakov Matic, Günther Kniewasser, Andreas Wöls

Graz University of Technology
A-8010 Graz, Austria

16 April 2018

Abstract

RepoVis as a software repository visualisation tool has several useful functionalities for displaying files and their changes but maneuvering inside of it and getting information over time is quiet a challenge. This paper gives an introduction to RepoVis and describes the concept of the timeline extension, which addresses said issues. Furthermore the used framework to develop said timeline extension will be discussed, as well as the implementation, the functionalities and a tutorial on how to use RepoVis.

Contents

List of Figures

Listings

Chapter 1

RepoVis

RepoVis is, by the time of writing this report, a PhD project for the visualisation of Git repositories inspired by SeeSoft[1]. Its purpose is to analyse Git repositories regarding the amount of code lines, the size, the contributors and the most recent commit informations of each file. Currently RepoVis analyses a selected repository, every time the user selects it. In the future it is planned that the server analyses the repository only once and when a user selects the repository, the user immediately gets the desired information and does not have to wait for the analysis process to finish. As for the technical background: RepoVis is built on Ruby with the framework Sinatra for preprocessing and uses HTML5 and WebGL for rendering.

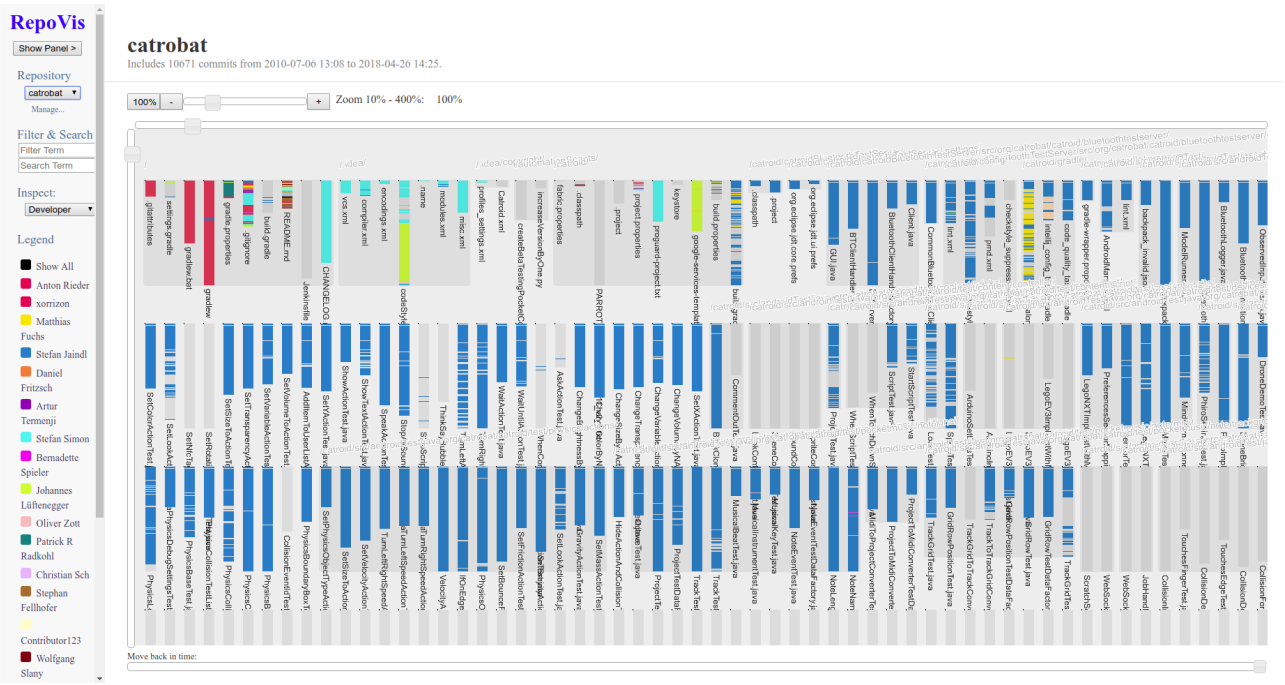

Figure 1.1: RepoVis with Pocket-Code [Ebner et al. 2017] repository selected. Screenshot created using RepoVis.

[1]SeeSoft was the first officially known visualisation tool for code, published by S.C. Eick, J.L Steffen and E.E. Summer in IEEE Transactions on Software Engineering (Volume: 18, Issue: 11, Nov 1992) Page: 957 - 968

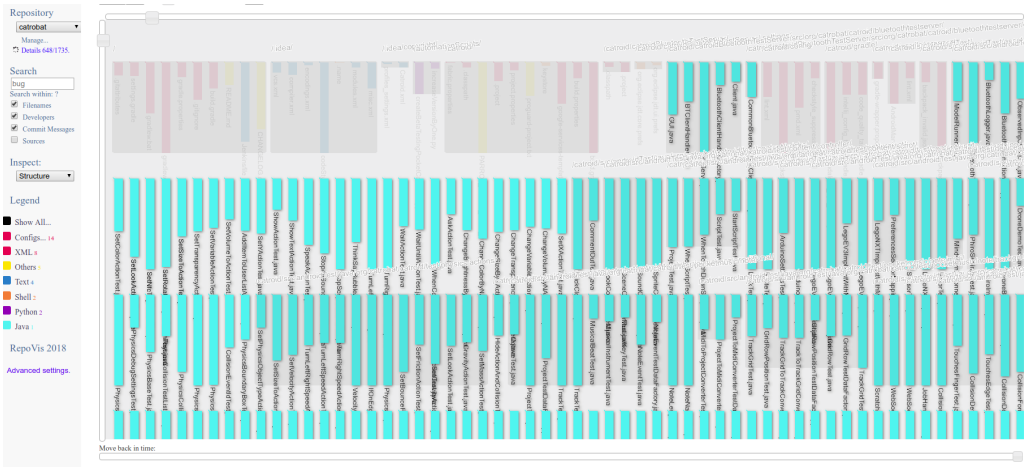

Figure 1.2: Highlight all Java files. Screenshot created using RepoVis.

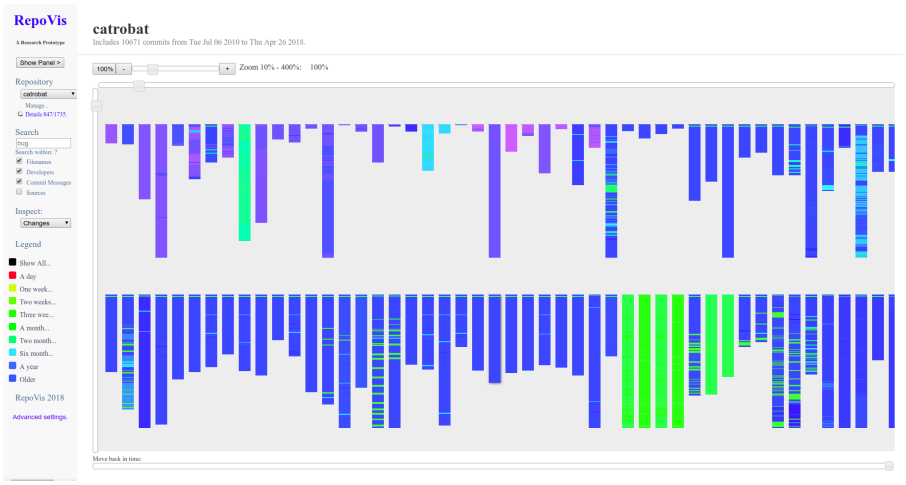

Figure 1.3: Files colored by when they were changed. Screenshot created using RepoVis.

1.1 Functions

RepoVis has some main functionalities which have the potential to be useful tools in the daily life of developers and project managers. The following features were tested on the Pocket Code [Ebner et al. 2017] repository of the Catrobat project of the Graz University of Technology.

1.1.1 Highlight

The first relevant feature is to highlight files in the repository by different presets like ”Last Modifications”, ”Developers”, ”File Types” and ”Issues”. In Figure 1.2 all Java files were highlighted and in Figure 1.3 all recently changed files.

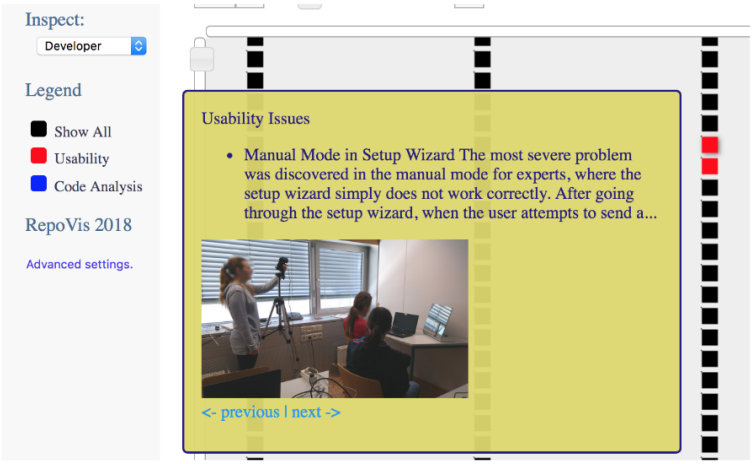

Figure 1.4: Usability findings can be inspected as they are shown on demand [Feiner and Andrews 2018]. Screenshot created using RepoVis.

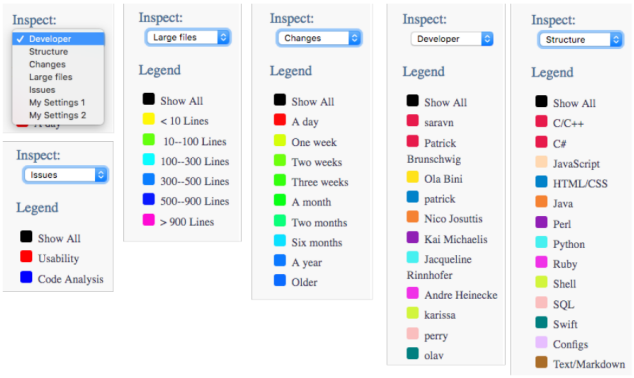

Figure 1.5: Presets enable fast switching between different views for different program comprehension tasks [Feiner and Andrews 2018]. Screenshot created using RepoVis.

1.1.2 Usability Issues

RepoVis has the unique feature to integrate external data into its metric system by collection data from users through questionairs about the program. This external data is then processed by RepoVis to show issues inside certain project files [Figure 1.4] to make it easier for developers to solve them.

1.1.3 Presets

Presets can create a general overview of the project with the help of various filter options. Figure 1.5 shows which metrics can be used as said filter options.

1.2 New Feature: Timeline

For enhancing the manoeuvrability in RepoVis and acquiring a better overview of commit quantity on certain days, the idea of creating a timeline extension emerged. The timeline should facilitate RepoVis's usability in a chronological manner as it is currently rather difficult to move between greater time periods and fulfil said need of a more general overview. The next chapter is going to present the concept of that timeline and the necessary requirements to realise it in the best suitable way.

Chapter 2

Timeline Concept

This section describes the functional and non-functional requirements. Generally, the slider differs in several aspects compared to the previous slider. The differences in implementation and usage will be explained in detail in this chapter.

2.1 Concept Design

Basically, RepoVis's main view represents a frame, in which the repository structure of a software project is shown in detail. A basic slider was the tool for navigating through the history of commits. Not being able to slide through the repository by commits, but only by days was a major weakness in RepoVis's usability. Therefore, a new slider with a more convenient usage was designed [Figure 2.1]. The core of the design is the slider sketch, that has a integrated chart in the background. The chart displays the relation between commits and days. In the front of the slider window a cursor can be dragged left and right. When the slider is moved left or right, the background updates and moves forwards or backwards in the timeline of the repository.

2.2 Slider Usage

Since the previous slider was not fitting the requirements, a new approach for navigating through the repository history was designed. In Figure 2.1 one can see the cursor, that lies on a bar, which represents the amount of commits on a specific day. The goal of the redesign was to improve the usability and intuitiveness of searching through the commit history. The need for the improvement of this specific element appeared, as the selection of commits with the previous slider were simply too difficult and seemed not precise. Additionally, the time it took to pick a repository commit was not appropriate.

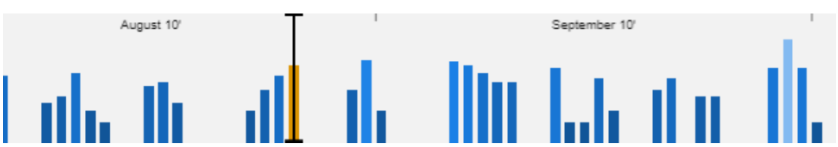

Figure 2.1: Illustration of slider and cursor selecting a day

2.3 Slider Functionality

The objective of the redesign was to improve the usability of navigation in RepoVis. Previously, a commit was selected by a simple horizontal slider, which lacked intuitiveness and accuracy. Another main problem that came along with the old version of the slider was the poor performance. Every time the slider has been used, the main frame was reloaded causing the repository illustration to reload.

2.3.1 Requirements of Slider

Confronted with the mentioned issues, the redesigned slider extension is capable of:

- Faster selection of commits
- Better accuracy of selection
- Easy and intuitive usage

2.3.2 Zoom Feature

The new zoom feature allows the user to zoom in and have a better view of the days in order to select a specific date. When scrolling with the mouse wheel, the background view updates and shows bigger and easier to select day bars, that consists of all commits. Additionally, one can move forward and backward when zoomed-in on the repository, by dragging the slider left or right.

2.3.3 Timeline Description

Another important element of usability within the zoom functionality, is the refreshing timeline description. As it can be seen in the presented Figures, the description of dates is implemented in a responsive manner. In Figure 2.2, the scale of zoom only allows a big scope of description. Therefore the year format with marks, representing a border between two years, was designed. In Figure 2.3, the scope of commit bars is slightly bigger, which enables an improved and more detailed specification of the timeline. In Figure 2.4, one can see that the year, the month and the days are visible.

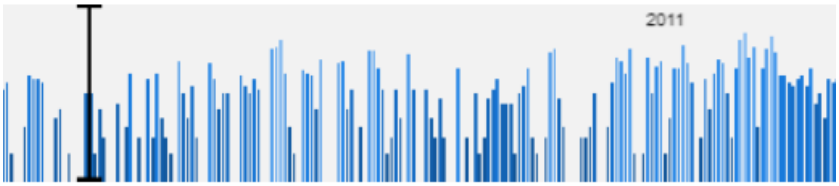

Figure 2.2: Background Chart with large-size description

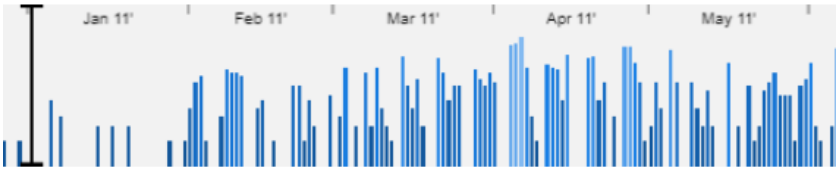

Figure 2.3: Background Chart with medium-size description

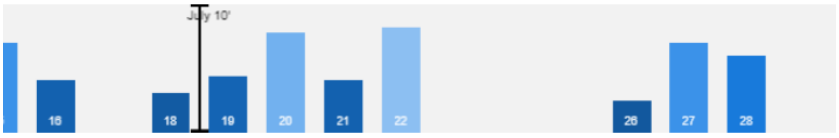

Figure 2.4: Background Chart with small-size description

Chapter 3

Creating the Timeline

For realising the previously described timeline we decided to use a JavaScript Framework called PixiJS in version 4.5.5. This chapter is going to give a brief overview of PixiJS's functionalities that are necessary to understand, to be able to comprehend how the timeline was created as well as how we used these functionalities to in fact create the timeline. All information about PixiJS come from the Github page Learning Pixi which is directly referenced by PixiJS's official web page.

3.1 PixiJS

PixiJS is a 2d renderer for web browsers. It uses WebGL so it is fast, however if WebGL is not available, it can also work on its own, but with a worse performance. It is a JavaScript library that can be easily integrated by including only one file. It is then possible to use all of its functionalities. PixiJS is licensed under the M.I.T. License.

3.1.1 Usage

With PixiJS one can add images to a website and modify these images. It is also possible to create 2d figures with PixiJS. If one wants to add an image the user should proceed as described in the following subsections.

3.1.2 Setting up PixiJS

To set up PixiJS it is simply necessary to add the file "pixi.min.js" to the project's folder. The only thing to do afterwards is to add the following line inside the HTML file: `<script src="pixi.min.js"></script>` and PixiJS is ready to be used.

3.1.3 Getting started

To get started drawing with PixiJS, first it is needed to create an application object, for example: `let app = new PIXI.Application({width: 256, height: 256});` this creates a black HTML canvas element with the size of 256 pixel width and 256 pixel height. After that, the user adds the canvas to the HTML document by adding the following line to the script: `document.body.appendChild(app.view);`.

3.1.4 Sprites and Primitives

As the Github page *Learning Pixi* states: "Sprites are basically just images that you can control with code. You can control their position, size, and a host of other properties that are useful for making interactive and

Figure 3.1: Example of drawn bars with different amount of commits.

animated graphics." [Github [no date]]. Primitives like lines, circles, triangles and rectangles can be easily created and modified according to size, shape, color and transparency and can be controlled in the same way as sprites. Both of these graphic types are made visible through adding them to the stage.

3.1.5 Stage

In the previously created application object a stage exists, which is a special container that renders all elements added to it by `app.stage.addChild(graphics);` automatically in the web browser. In general Pixi containers are objects that are used to group several Pixi sprites and primitives and perform operations on them simultaneously.

3.2 Timeline Implementation

For the timeline the canvas was created beforehand so the application could access it directly. Afterwards a "day bar" was created as a rectangle primitive for each commit day with a length that was proportional to the amount of commits of the respective day and a suitable coloring was chosen [Figure 3.1].

The width was determined according to the amount of days so every rectangle had enough space on the canvas. Each day rectangle was given an "on click" listener event to change the color of the respective day rectangle and move the slider to that position:

```
graphics.on('click', function (data) {
            _this.resetOldBarColor();
            _this.lastBarColor = heatColor;
            _this.selectedCommitDay = this.dayNr;
            this.graphicsData[0].fillColor = "0xdc9400";
            _this.updateDropDown();
            _this.slider.transform.position.x = this.hitArea.x + (this.
                hitArea.width / 2);
            _this.showCommitInfos(0);
        });
```

Listing 3.1: Listener to click on a day bar

The slider which has the functionality to move around in the timeline is a sprite that was imported from a PNG file. It was given an "on pointerup" listerner to choose the day rectangle that is colliding with the slider. For that collision detection we used the objects bounds and compared the x coordinates and width with some offset:

```
function isColliding(slider, bar) {
    let offset = bar.width / 4;
    let sliderBounds = slider.getBounds();
    let barBounds = bar.getBounds();
    return (sliderBounds.x + (sliderBounds.width / 2) > barBounds.x -
        offset && (sliderBounds.x + (sliderBounds.width / 2)) <
        barBounds.x + barBounds.width + offset);
}
```

Listing 3.2: Collision detection between slider and day bar

The zoom buttons, the commit selection and the commit details dialog were all realised in plain HTML with events triggering the corresponding functions in JavaScript:

```
<button id="zoomin" class="zoomPlus" onclick="timeline.zoomIn()">+</
    button>
<button id="zoomout" class="zoomMinus" onclick="timeline.zoomOut()">-</
    button>
<button id="zoomreset" class="zoomReset" onclick="timeline.setZoom(100)
    ">100%</button>
<form>
    <select id="subcategory" onchange="timeline.showCommitInfos(this.
        selectedIndex)"></select>
</form>
```

Listing 3.3: HTML zoom buttons and commit selection

Chapter 4

Technical Details

This chapter is going to explain the technical details of the implementation. The project was split into the following categories:

- Gather Data
- Parse Data
- Calculate Commits per Day
- Normalize Data to fit onto the Screen
- Zoom

4.1 Gather Data

The data that was necessary to visualize were gathered by an API call to RepoVis. A JSON formatted file is returned which can be parsed [Figure 4.1].

```
    {
        "tree": "11cbb5f548179db12b1fa541777e8a6423288a76",
        "author": "Max Mustermann",
        "message": "Merge pull request #2793 from 84n4n4/anotherRefactoringCleanup\n\n[REFACTORING] cleanup, fixed layout and some ux issues",
        "timestamp": "2018-04-18T16:40:07+02:00"
    },
    {
        "tree": "f342bb665cb66bbd5274a79a3e6c2ba5baa85d72",
        "author": "Max Mustermann",
        "message": "[REFACTORING] Changed StorageHandler to work with File Objects as params\n",
        "timestamp": "2018-04-24T13:00:56+02:00"
    },
    {
        "tree": "f342bb665cb66bbd5274a79a3e6c2ba5baa85d72",
        "author": "Max Mustermann",
        "message": "Merge pull request #2794 from thmq/storageHandler\n\n[REFACTORING] Refactored StorageHandler to work with File objects as â¦",
        "timestamp": "2018-04-24T13:44:11+02:00"
    },
    {
        "tree": "716fcee84a755d9cc55f3e0b52145c68c0eb5833",
        "author": "Peter Laggner",
        "message": "CAT-2683 Refactor FormularEditorCategoryListFragment\n\n- refactor FormulaEditorCategoryListFragment to class using RecyclerView\n-
        "timestamp": "2018-04-24T15:45:20+02:00"
    },
    {
        "tree": "08d5d2acec08c32f9e882346b86cfab3d783055c",
        "author": "Max Mustermann",
        "message": "Merge pull request #2773 from Tot333/CAT-89\n\nCAT-89 default values incosistent number format",
        "timestamp": "2018-04-25T17:01:34+02:00"
    },
    {
        "tree": "7025bdacb0a8c482d835d2324f0478f7dad18be5",
        "author": "Max Mustermann",
        "message": "Merge pull request #2786 from LPeteR90/LPeteR90\n\nCAT-2683 Refactor FormularEditorCategoryListFragment",
        "timestamp": "2018-04-26T16:25:21+02:00"
    }
]
```


Figure 4.1: Data from API call in JSON format

4.2 Parse Data

After the data is received, everything is then saved into a map to have access to the data for the whole duration of the program.

4.3 Calculate Commits per Day

To create the visualization with bars it is necessary to calculate how many of the commits happened on the same day. Therefore the program iterates over all commits and parses the timestamps to unixtime in seconds. After that it is counted how many of the commits did have the same unixtime.

$$\frac{\#Commits}{day} = \sum_{day}^{M} \sum_{commit[day]}^{N} 1 \tag{4.1}$$

After the calculations a map was created that has **day** as the key and **#Commits** as value.

4.4 Normalize Data to fit onto the Screen

To display the data on the screen in a sophisticated way the bars had to be normalized so that everything had enough space.

4.4.1 Normalize Y-Direction

To fit everything in the y-direction (height of the bars) the height of the screen was divided by the maximum of found commits per day.

Figure 4.2: Timeline with normal scale

Figure 4.3: Timeline with logarithmic scale

$$y_{scaling} = \frac{\text{screen height}}{\max\left(\frac{\#Commits}{day}\right)} \tag{4.2}$$

Each of the drawn bars got scaled by this value to make sure everything fits onto the screen.

$$y_{screen} = y * y_{scaling} \tag{4.3}$$

4.4.2 Normal Scaling vs. Logarithmic Scaling

In section 4.4.1 the bar height is normalized to fit the screen height. Without logarithmic scaling on the y-axis the bars with a lot of commits would dominate the others and everything looks squished together [Figure 4.2]. With logarithmic scaling each of the bars got more weight to it and hence was easier to read [Figure 4.3].

4.4.3 Normalize X-Direction

To fit everything in the x-direction (width of the bars) the width of the screen was divided by the number of days.

$$x_{scaling} = \frac{\text{screen width}}{days} \tag{4.4}$$

Since every bar would now only get 1 Pixel of space to fit everything onto the screen this value was divided by some pixel spacing value (in this case: 3) so that the bars can be bigger and still have some space between them.

$$x_{better\ scaling} = \frac{\text{screen width}}{days} * \frac{1}{\text{Pixel Spacing}} \tag{4.5}$$

Each of the drawn bars got scaled by this value to make sure everything fits onto the screen.

$$x_{screen} = x * x_{better\ scaling} \tag{4.6}$$

4.5 Zoom

With thousands of commits or projects that span over multiple years it is hard to see or click specific commits. Therefore the decision was made to build a zoom functionality which either zooms to the location of the cursor (zoom with buttons) or to the location of the mouse pointer (zoom with the mouse wheel). For the zoom the zoomed location is centered to the middle of the screen and everything is shifted to the sides.

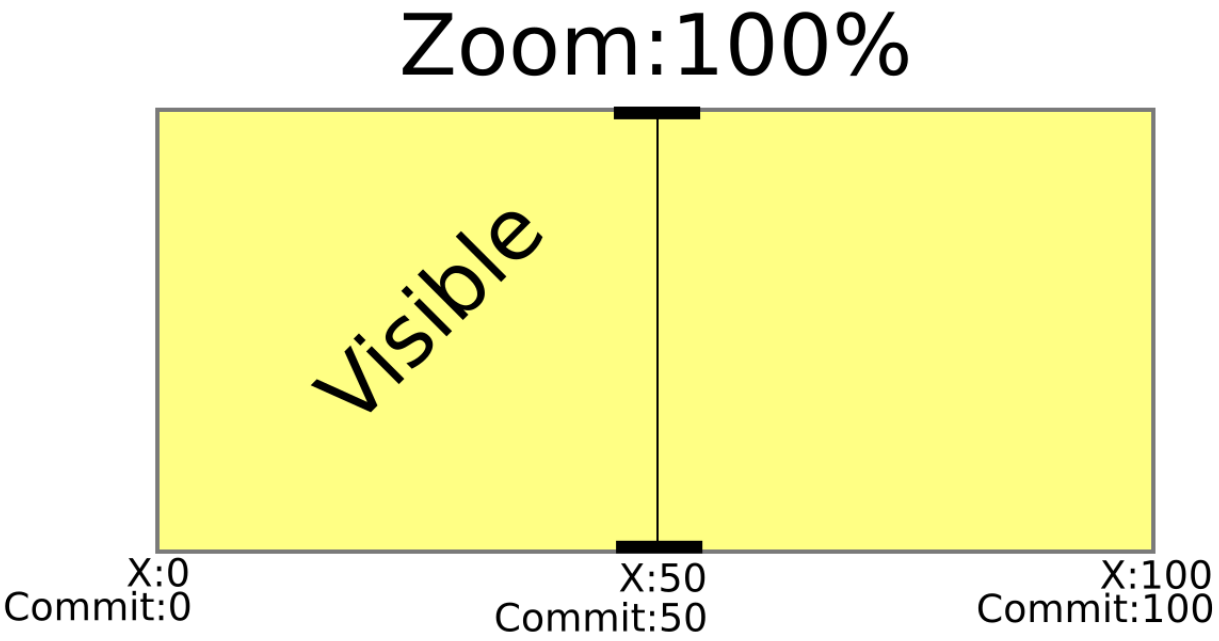

Figure 4.4: Zoom Illustration at 100%

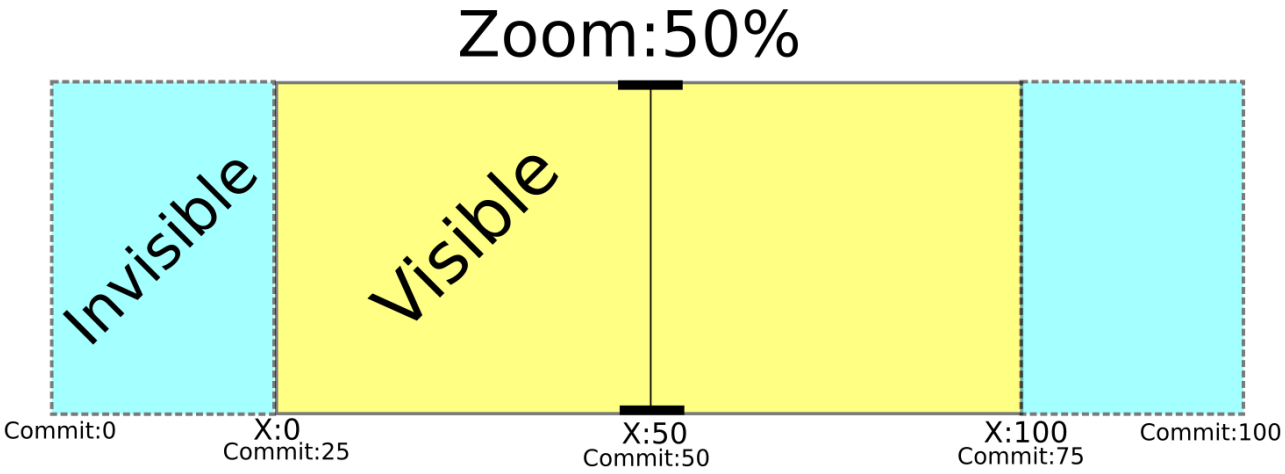

Figure 4.5: Zoom Illustration at 50%

Bibliography

Ebner, Martin, Stefan Janisch, Bettina Höllerbauer, Maria Grandl and Wolfgang Slany [2017]. "Pocket Code – Programmieren für Alle mit einem offenen Online-Kurs". 01 [Apr 2017], pages 16–19 (cited on pages 1–2).

Feiner, Johannes and Keith Andrews [2018]. "RepoVis: Software Repository Visualisation with Integrated Code Metrics and Usability Issues". 14th May 2018 (cited on page 3).

Github, kittykatattack. *Learning Pixi*. https://github.com/kittykatattack/learningPixi (cited on page 10).