Bachelor Thesis

June 18, 2019

Visual Analysis of TeX Repositories

Luka Volk

Abstract

Many scientific papers require collaboration of multiple authors across multiple institutions. To aid this process, a versioning system is often used. By far the most popular versioning system in the world is Git, and most scientific papers under version control are using it. Still, there is no real solution available tailored specifically to the needs of TeX repositories.

This paper presents TeXVis, a web application which analyzes Git repositories containing TeX papers, and gives a visual representation of the data, focusing on the roles each author plays in terms of ownership and contribution, as well as providing visualization of the TeX element tree of each file in the repository, and visualization of history of each individual TeX element.

Keywords: TeX; Git; Visual Analysis

Advisor:
Prof. Michele Lanza
Co-advisor:
Dr. Csaba Nagy

Approved by the advisor on Date:

Contents

List of Figures

1 Introduction

The core feature of a versioning system is the ability to track changes being made to the files in the repository. This is a particularly useful feature when there are many collaborators working together on a project since they are often interested in knowing who contributed what and when. It is most often used by developers who are developing code together, but it is not limited exclusively to that. Tracking of changes is useful for any text file regardless of the content. One of the prime examples of this are scientific papers written in TeX, a popular format for these publications.

Most scientific papers include more than one author, and it is not rare for a paper to have authors from different countries and even continents working together on a paper. As such, they benefit greatly from having files stored in a system which does all the bookkeeping for them.

1.1 Versioning System Popularity Comparison

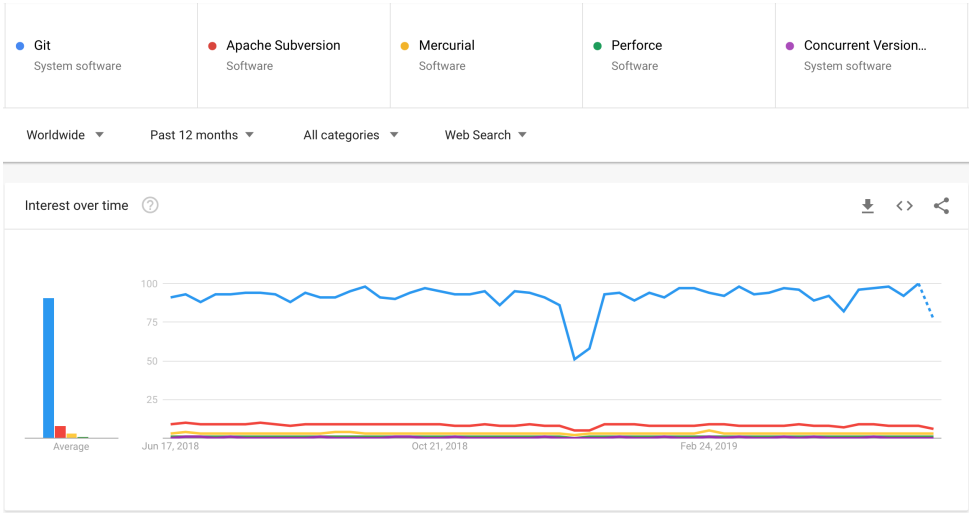

Figure 1. Google worldwide searches for different versioning systems in the last 12 months

There are many versioning systems available. To get an idea about popularity of some prominent versioning systems, a Google trends[1] query was performed, see Figure 1.

The Figure does not contain an exhaustive list of versioning systems. However, it offers some insight into how popular systems stack up against each other. As we can see, Git is the clear winner among the compared systems.

1.1.1 Git

Versioning systems have already been mentioned in this introduction, but what exactly is a versioning system?

According to the Pro Git book [2] "*Version control is a system that records changes to a file or set of files over time so that you can recall specific versions later.*"

Another quote from Pro Git book might be helpful in answering the question what makes Git so popular: "*Conceptually, most other systems store information as a list of file-based changes [...] Git doesn't think of or store its data this way. Instead, Git thinks of its data more like a series of snapshots of a miniature filesystem.*"

It is important to understand Git (or at least know what it is), as this project is making extensive use of it.

1.1.2 GitLab

GitLab is an online Git repository management tool.

Currently, this project has support only for repositories hosted on GitLab servers, as it is relying on GitLab API[2] to get information about the repository.

[1] Google trends: `https://trends.google.com/trends`
[2] GitLab API: `https://docs.gitlab.com/ee/api`

1.2 TeX

TeX is a typesetting system which is "*intended for the creation of beautiful books — and especially for books that contain a lot of mathematics*" [4]. Since its inception in 1978, it has become the de facto standard for scientific writing.

Many tools with the aim of helping collaborators in tracking software evolution exist. Among them are some well-known commercial solutions(e.g., GitHub and GitLab for Git) as well as tools that stem from academia [6]. Some of these tools are targeting specific programming languages, like Java [3]. However, none of these tools are designed for tracking the progress of TeX repositories.

Why is there a need for a tool which specifically targets TeX repositories? Consider the following scenario: a group of collaborators on a scientific paper is working on the introduction section together. When it is time to submit the paper, they would like to know who and when made changes to the introduction, how much of the introduction was contributed by which author, etc. Since the introduction is only a part of a certain file, there is no simple way of looking up this information.

This is just one possible use case where a contributor in a TeX repository would benefit from an application specifically designed for analysis of TeX repositories.

This paper introduces TexVis, a web based application which aims to satisfy the specific needs of collaborators in a TeX repository.

2 Overview

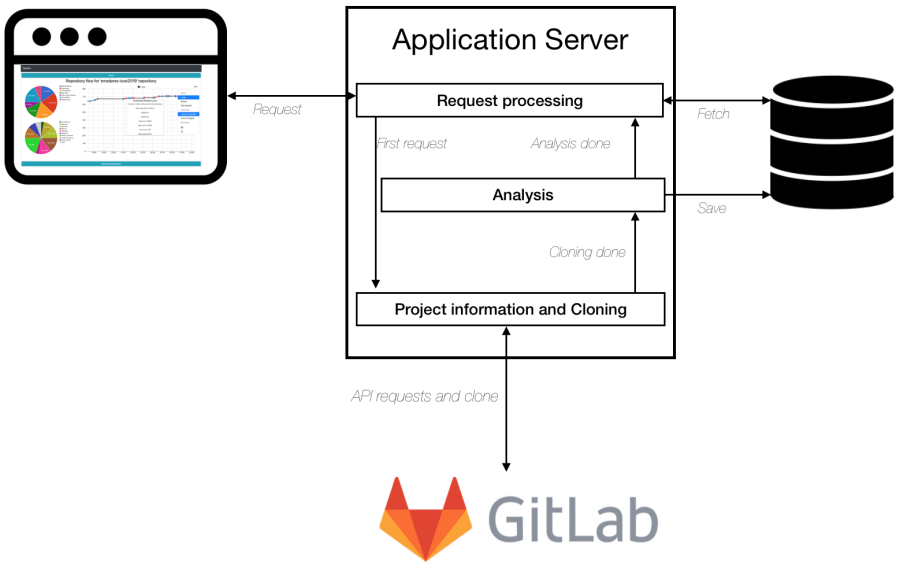

Figure 2. Application overview

The high-level overview of the application is shown in Figure 2.
The application consists of three parts:

- **Front end**. The user interacts with the application via the front end, which is displayed in the browser. Front end, in turn, communicates with the application back end, which serves the requested data. For rendering the front end, Thymeleaf[3] is used as a server side templating engine. Vue.js[4] is used in the browser to keep the user interface responsive.
- **Back end**. Back end serves the requested information to the front end, and is also responsible for analyzing the repository and persisting the data. This implies that it has to communicate with both the front end and the database. To analyze the repository, it first has to clone it, so it has to communicate with the GitLab server of the repository as well. The back end is built with the Spring Boot[5] framework. For mapping Java classes to the database, the application is relying on Hibernate[6] as the JPA implementation provider.
- **Database**. Because the server is written in Java, it is quite natural to map the entity objects to a relational database. The database management system in this project is MySql[7].

2.1 Feature Selection

In his PhD work, Voinea identifies the main question of a software evolution system as: "*How to enable users to get insight in the evolution of a software system?*" [5]. According to Voinea, the main challenges of such a system are scalability, intuitiveness, usability and integration.

2.2 Architectural Design

To tackle the problems of intuitiveness and usability, TeXVis is providing users with features that make it unique, but always in the frame of the general context of the repository.

The specific information that we are trying to provide is related to the TeX elements in the files. Specifically, the two must-have features are TeX element tree for a specific file, and TeX element history, since that is what is lacking most when we look at general solutions for repository tracking and visualization.

[3]Thymeleaf: `https://www.thymeleaf.org`
[4]Vue.js: `https://vuejs.org`
[5]Spring Boot: `https://spring.io/projects/spring-boot`
[6]Hibernate: `https://hibernate.org`
[7]MySql: `https://www.mysql.com`

To provide context, and ensure a smooth integration with the TeX-specific features, the application is conceptually split into three levels:

- **Repository level**. A user can see all the commits, files, and the most important commit-related information.
- **File level**. Similar to the repository level, but with the specific feature that allows the user to see the file text and TeX elements at each commit.
- **TeX element level**. This level is similar to the file level, with the difference being that it is displaying information related to each individual TeX element.

For the back end of the application, this implies storing all the commits, files, and all changes to the files and the TeX elements in the repository. TeX elements first have to be found in the text, and for this, we will need a TeX parser.

The front end of the application needs to present this information in an easy to understand manner, with the use of charts.

3 Back End

3.1 Selection

As we communicate with GitLab through its REST API, this requires that the server's address, the username, and an access token are provided. The token is a personal access token, which a user needs to previously create by logging into their GitLab account. The user must have at least reporter permissions for the specified project (GitLab provides several levels of authentication, in increasing order of privileges granted to them: Guest, Reporter, Developer, Maintainer, Owner)[8].

After this information has been provided, a user can decide to either manually input the project ID, or select one that is provided in the selection field. The elements in the selection are provided based on the information discussed above — the server address, username, and token. This information is obtained by the application with an AJAX call to the specified server address. Only the projects for which the user has the authorization to use the GitLab API will be shown. If the token (or the user) do not have the permission for any repository on the given server, the selection will be empty.

Another option is to manually input the project ID. The ID is a unique identifier of the project, on a server to server basis. A user can simply look it up on the dashboard of the project in GitLab.

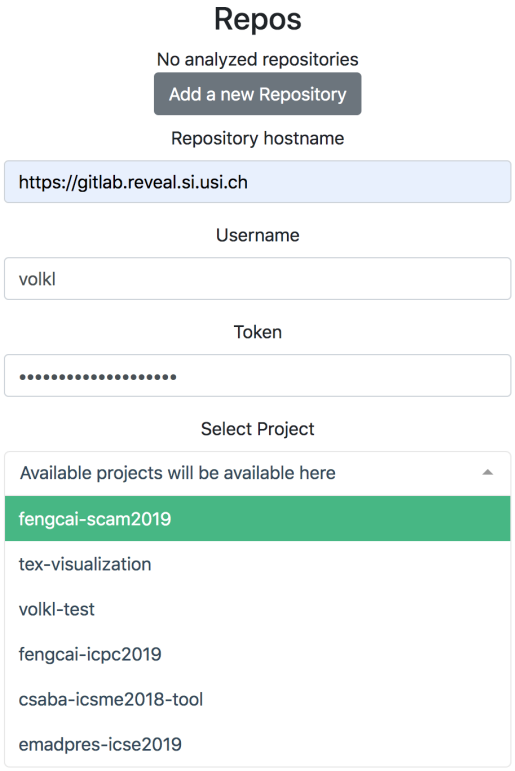

Figure 3. Project selection form

3.2 Cloning

After the selection process, the application clones the repository. This is done differently based on how the user selected the project to analyze.

If the manual selection was used, then information which was input into the form is passed to the server side of the application, and a request is issued to the server of the project. In the response of this request, we receive information about the URL, which is then used for cloning the repository.

On the other hand, if assisted selection was used, all the needed information has already been received by the front end of the application (in the AJAX response), so the front end does not pass the user provided information to the application server, but rather the information it has received from the project server. This spares the server side from making a request to the server (as the front end already made this same request). Figure 3 shows the form with which the users specify the information.

[8]GitLab permissions: `https://docs.gitlab.com/ee/user/permissions.html`

3.3 General Information Extraction

After the repository has been cloned, the server side of the application starts the analysis. This is done by issuing shell commands, utilizing the Git command line interface to get the information about the repository. The basic information that is extracted is the commits in the repository and the files changed in the specific commit.

According to Aghajani et al. "*Besides source code history, additional information such as code metrics are the most important resources to understand a system's evolution*" [1]. TeXVis extracts multiple metrics about each analyzed file, namely: number of lines, number of words, and size. Although these metrics are not as sophisticated as some of the metrics used in the referenced paper (e.g., number of methods, cyclomatic complexity), they still offer valuable insight into how files evolved over time.

After the necessary information is extracted, it is mapped to the Java entity classes, and saved in the database.

3.4 Database Structure

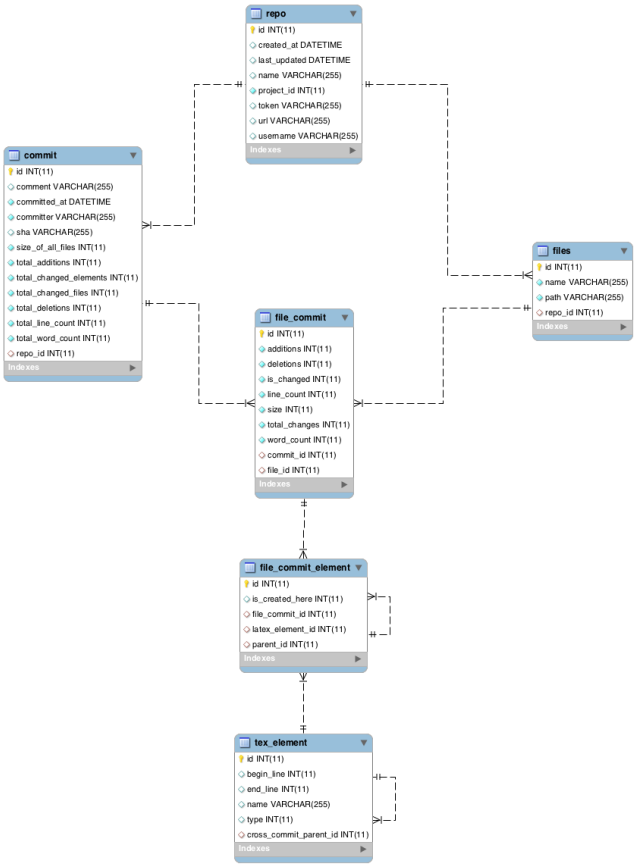

Figure 4. Database structure

Repo table is at the root of the database hierarchy. It stores the data passed by the user as well as other pertinent information, for example the creation date. *File* and *Commit* tables are self-explanatory. The *FileCommit* table models that a file is in the repository at the time of a commit. It has a flag which serves the purpose of knowing whether changes were made in the related commit. If the flag is false, then no other data is stored in the table row, since all the data is the same as in the previous commit and has already been stored in the database. If the flag is true, however, information (size, words, lines) about the file at the time of the commit is stored. The TeX elements of the file are stored as well. They are modeled by the tables *FileCommitElement* and *TeXElement*. What is interesting to note about these two tables is that both of them have a self looping parent-child relationship. This relationship in the *FileCommitElement* models the parent-child relationships in the element tree, and the one in *TeXElement* models the relationship between the previous and the new version of the element.

The structure of the database is shown in Figure 4.

3.5 File Parsing

Before the *TexElements* can be stored in the database, the files in the repository have to be parsed, to get the TeX elements of each file at each commit. This is done with the use of a third-party software called TeXlipse. After TeXlipse returns the elements it has found, the application stores the related information in the *FileCommitElement* and the *TeXElement* tables. Whether the element has been changed in the current revision is determined by utilizing the ability of Git to track history of lines in a file.

3.5.1 TeXlipse

TeXlipse is an Eclipse IDE plugin[9]. It was first developed as a software project at University of Helsinki. Later develpement took place as an open source project on SourceForge. Currently, the Eclipse foundation is the maintainer of the project. The source code is available on GitHub.[10]

[9] TeXlipse home page: `http://texlipse.sourceforge.net`
[10] TeXlipse repository: `https://github.com/eclipse/texlipse`

4 Front End

4.1 Repository Level Visualization

The repository level visualization is displaying general information about the repository on a commit to commit basis. Some of this information is not specific to TeX repositories, but it is one of the useful non-TeX features to have in the application, as already discussed.

4.1.1 Repository Level Charts

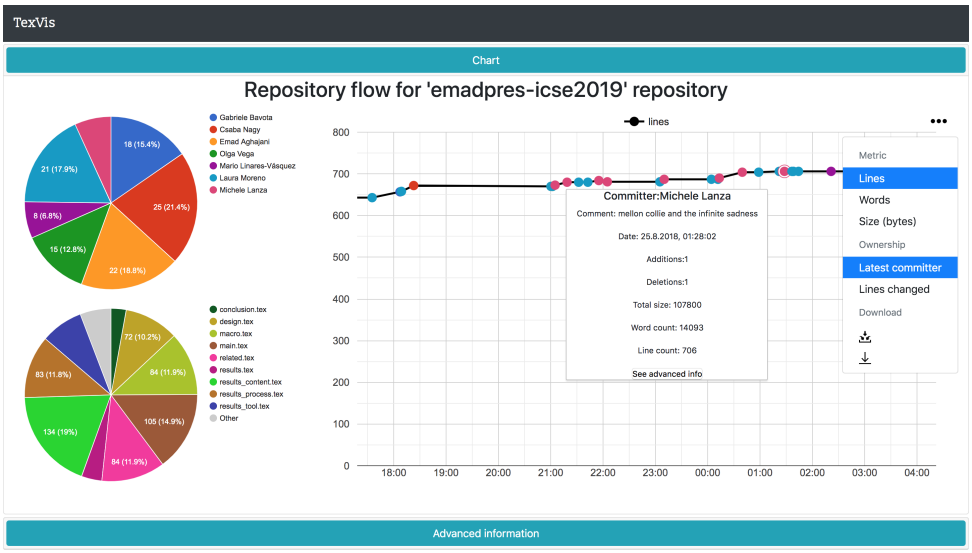

Figure 5. Repository level visualization

In the repository level visualization, three charts are displayed, see Figure 5. On the left side of the screen, there are two pie charts. They show the information about the contributors, and the files in the repository. By default, the contributor chart is showing commits per author. The file pie chart shows the relationships between files based on the number of lines in each file.

On the right side of the screen, we see a line chart, where the commits are plotted. The x axis represents time, and the y axis represents the selected metric. In the repository level visualization there are three metrics to choose from: lines, words, and characters. The default metric is “lines". The color of each person is the same in the contributor pie chart and the commit line chart.

An important feature of these charts is that they are always synchronized with each other. For example, the color of a commit in the line chart is the same as the one in the contributor pie chart for each committer. The charts are synchronized also when we want to make changes to the metrics being displayed.

There are two possible changes to all graphs by selecting the three dot symbol in the top right screen of the line chart.

- **Metric**. The chart can display count of words or characters instead of the default count of lines. When this metric is changed in the line chart, it is also changed in the file pie chart.
- **Ownership**. Another option is available regarding “ownership” of the repository. By default the commits are colored in the color of the committer. This color can be changed in the settings to show the color of the contributor who contributed the most to the repository up to the selected commit. This also redraws contributor pie chart so that it displays percentages based on the number of changes.

Another important feature is the tooltip. It is a quick way of obtaining general information about the commit, like the size, date committed...

4.1.2 Repository Extra Information

By folding the “accordion" element on the page, the user can switch to another screen, shown in Figure 6. The user sees the general level information that we have already seen in the tooltip, but more importantly, the files in the repository at the selected commit. The ones that were changed in this commit are colored green. The user can click

on each of them to see its TeX element tree. The tree elements in bold denote elements changed in currently selected commit. Note that for the files that were not changed in the current commit, no element can ever be changed, so their elements are never bolded.

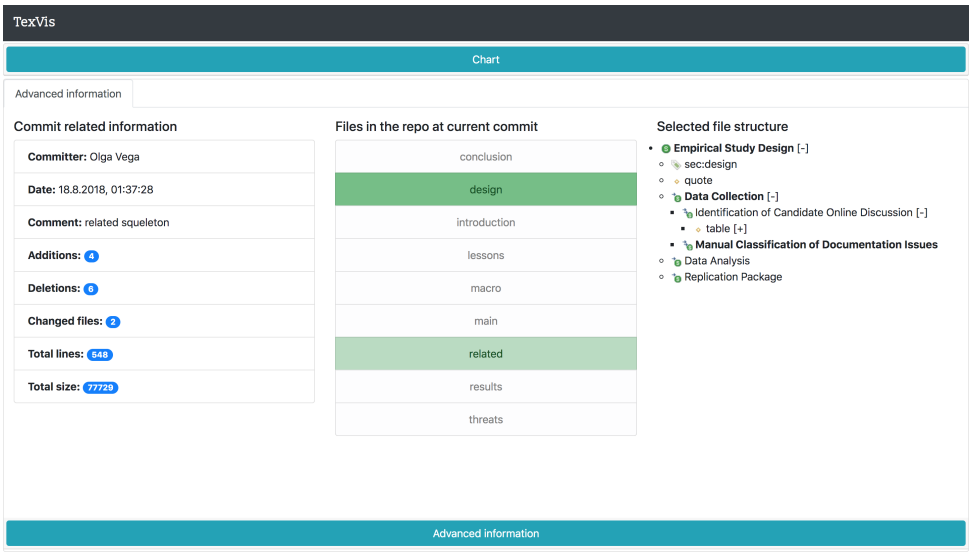

Figure 6. The extra information screen

By double clicking on the element, the user gets redirected to the TeX element history page. Likewise, by double clicking on the file, they can go to the page which displays its evolution.

4.2 File Level Visualization

The charts in the file level visualization are very similar to the repository level charts, with the exception of displaying information for an individual file instead of the whole repository, and that the pie chart is not displayed, since we are considering only one file here.

What is new in the file level repository is the text of the file with the related element tree. Refer to Figure 7 for more information.

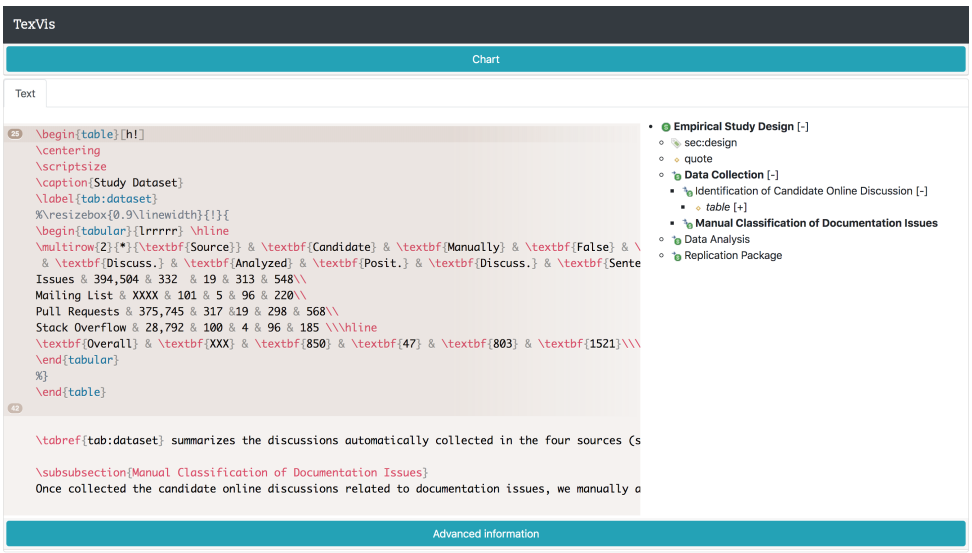

Figure 7. Text with element tree

Users can select the element so that the corresponding lines in the text are highlighted. As on the repository level, a double click takes the user to the element level page.

4.3 TeX Element Level Visualization

This level is very similar to the file level, except, of course, that it displays information about an element.

4.4 Other Features

Two other important features are the possibility to remove and to update a repository.

Remove operation deletes all the entries of the repository from the database, and also deletes the folder of the cloned repository.

The update function pulls the latest commits from the repository. If any of the new commits contain changes to the .tex files, they are analyzed and added as normally when analyzing the repository, and ignored otherwise.

These operations are available from the repository selection screen by clicking on the buttons in the corners of the tiles, as seen in Figure 8

The charts on the page are rendered using Google Charts[11]. They are represented as an svg element. Download of these svg elements is possible by clicking on the single arrow for the single dowload of the line chart, and the multiple arrow symbol for downloading all of the charts on the page, as seen on Figure 5.

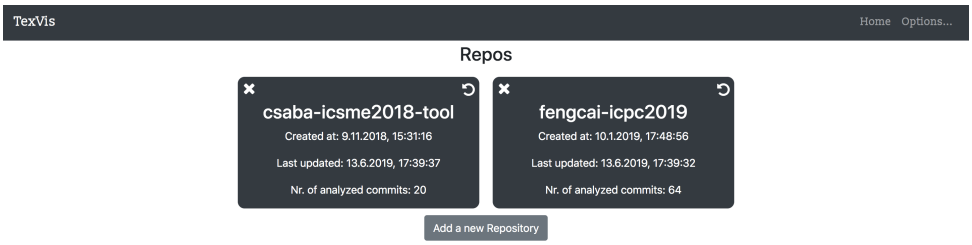

Figure 8. Selection screen after analysis

4.5 Transitions Between Levels

An important thing to note is that the user interface was designed with deep linking in mind — if a user selects a particular commit and saves the link, they will see the exact same thing the next time the link is opened.

What is special about transitions between different levels is that when a user selects a certain commit at, say, repository level and from there they want to see the file flow (as described in section Repository extra information) they will not be taken to the same commit in the file flow page, if that file was not changed by the requested commit. This is because that data point does not exist — if the file was not changed in a certain commit, then plotting it on a graph would result in a flat line. Most commits change only a small percentage of all files, and even a smaller percentage of TeX elements, which means that including these “dead" data points would make most charts very flat and not very informative.

When the data point does not exist, the revision displayed in the file flow page will be the last one where the changes were made. This rule applies across all levels of visualization.

[11]Google Charts: `https://developers.google.com/chart/`

5 Conclusions and Future Work

The project reached most of the goals it has set out to achieve. It is providing useful TeX non-specific information, but also offers TeX-specific selected features, like file tree structure and TeX element history. It does so with the use of charts, where the information is color coded, to ensure data representation is intuitive and easy to understand, which has also been one of the goals of the project.

There are several possible extensions which would enhance the functionality.

One possible extension would be to add support for different versioning systems, as currently only Git is supported. Doing that would also imply adding support for multiple different development platforms.

Support for other versioning systems can be added, even though this should not be a top priority, as the popularity of Git ensures that the application is applicable a large number of projects.

Perhaps the most important feature that is missing is analysis of other TeX files which contain useful information, such as the .cls and .bib files, as currently only .tex files are being analyzed.

References

[1] Emad Aghajani, Andrea Mocci, Gabriele Bavota, and Michele 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] Scott Chacon and Ben Straub. *Pro Git*. Apress. Online at: `https://git-scm.com/book/en/v2.html`.

[3] Christian Collberg, Stephen Kobourov, Jasvir Nagra, Jacob Pitts, and Kevin Wampler. A system for graph-based visualization of the evolution of software. In *Proceedings of the 2003 ACM Symposium on Software Visualization*, SoftVis '03, pages 77–ff, New York, NY, USA, 2003. ACM.

[4] Donald E. Knuth. *The TeXBook*. Addison-Wesley, 1984.

[5] Lucian Voinea. *Software Evolution Visualization*. PhD thesis, Technische Universiteit Eindhoven, 2007.

[6] Lucian Voinea, Alex Telea, and Jarke J. van Wijk. Cvsscan: Visualization of code evolution. In *Proceedings of the 2005 ACM Symposium on Software Visualization*, SoftVis '05, pages 47–56, New York, NY, USA, 2005. ACM.