Software Repository Visualisation

Amel Hamidovic, Jakov Matic, Günther Kniewasser, Andreas Wöls

Graz University of Technology
A-8010 Graz, Austria

16 April 2018

Abstract

Acquiring knowledge about specific data of a software project can be really challenging. One way to get an overview of and facilitate your entry into a project is to visualise its files and folders, with some additional information. In this paper we are dealing with the means of visualising whole repositories and the ways users can interact with these visualisations as well as their pros and cons. We will be looking at the history of software repository visualisation starting with early tools like Seesoft and then explain some of the currently existing programs. We will discuss the different categories of software repository visualisation like, "dependency structure", "files, folders and lines of code", "metrics", "project monitoring" and "evolution of code over time". Furthermore, we will present some programs for these categories. The programs are: "SolidTA", "RepoVis", "CodeFlower", "Gource", "GitStats".

© Copyright 2018 by the author(s), except as otherwise noted.

This work is placed under a Creative Commons Attribution 4.0 International (CC BY 4.0) licence.

Contents

List of Figures

Listings

Chapter 1

History

This section is about the first software repository visualisation program. It was the inspiration for many other programs that followed. A difficult problem in software visualisation is to find a way to represent the given information. Before 1992 there were already programs to analyse program codes from version control systems, static analysers and project management tools. The amount of code can make it difficult to gain insight. Visualisation software had to reduce the given information to a degree which was useful for the user to analyse.

The Seesoft software tool[Eick et al. 1992] was developed in 1992 to visualise line oriented software statistics. It could analyse up to 50 000 lines of code simultaneously by mapping each line of code into a thin row. The success of the software came from four major key-ideas.

- **Reduced representation**: Display files as line of code in thin rows.
- **Colouring by statistic**: Each line has its own color depending on the use case(e.g color by latest-change).
- **Direct manipulation**: The user can directly manipulate the data to find interesting patterns.
- **Read actual code**: Users can open the actual code by positioning the cursor over the reduced representation which will zoom into the content.

Figure 1.1: Seesoft showing zoom function. Screenshot taken from the paper[Bacher et al. 2017]

Chapter 2

Tool Categories

Software repositories can be visualized in different ways for different needs. With the progress of these visualisation tools, a loose categorisation for these tools can be made:

- Dependency structure
- Files, Folders and Lines of code
- Metrics
- Project monitoring
- Evolution of code

2.1 Dependency Structure

With the rise of complexity of (object oriented) software projects the need emerged to have an overview of all class structures. Therefore programs that visualise dependency structure were developed, with which it is possible to see the class inheritances and class memberships at a glance. These programs help to introduce new classes, because it immediately visible from which existing classes to inherit and prevent the creation of duplicates. Furthermore it is possible to refactor the already existing structure and remove redundancies. One of the most important points would be that it is then possible to solve ciruclar dependencies and preserve overall code quality.

2.2 Files, Folders and Lines of Code

The objective of this category is to get an bird's-eye-view of the whole project in regards to the individual parts (files, folders and lines of code). It makes it possible to see files and folder dependencies as well as the lines of code of these files in detail. With the help of corresponding programs a large application can be refactored, as file sizes can be reduced and folders can be reorganised. It could also be possible to see commits and contributions of the project.
Things that can be analysed:

- last changes
- commits over time
- authors

2.3 Metrics

Metrics allow the user to get a deep insight into the project, by analysing and visualising code content. The corresponding software preserves code quality by test coverage inspection as well as bug and code smell recognition in most cases. It influences directly the development process of a software project instead of just creating overviews, and changes dynamically with every commit. The main goal for visualising software repository metrics was to have a -more or less- live inspection of the current project status enhanced with additonal in-depth informations. By continuously calling attention of the developers, these informations should in the long run save time and money for everyone involved in the development of the ongoing project.

2.4 Project Monitoring

Project monitoring software calculates continuously key paramaters like time and commits for productivity, contributors and commits for participation as well as lines of code and time for code complexity. With such tools it is possible to monitor the project over time and manage its further development. Moreover the user is able to analyse the project and identify all general aspects like the total amount of files, all contributors, tags and files types of the project.

2.5 Evolution of Code

Several software repository visualisation tools faced the need to have a global view on file changes over a certain time period. In most cases these periods were between relevant commit or deployment versions leading to a more specific approach in that case. With these tools it was now possible to exaclty examine how the project evolved and which previous decisions led to the current code state. That could be rather important in either acqouiring a better understanding about the project or solve curcial problems that occured through repeatedly made misdecisions. Software which addresses evolution of code is mostly used for software maintainance, as it needs a certain history of changes made in a project to show differences between versions.

Chapter 3

Software

In the previous chapters the different categories of software repository visualisation were described. In the following chapters there will be programs presented, which contain the functionalities of these categories. The programs differ a lot from each other. They have a broad variety of functionalities and can be used for a wide range of use cases. Some tools like CodeFlower and Gource have only a limited set of features and are mainly used to have some sort of visualisation that is nice to look at but the analysing features have not much depth. On the other hand some tools like SolidTA, RepoVis and GitStats are less focused on the aspect of presentation, but rather want to give the user a lot of features and options in analysing his or her repository and provide the user with helpful information.
The following programs will be discussed:

- SolidTA
- RepoVis
- Gource
- GitStats
- CodeFlower

Figure 3.1: SolidTA: User interface. Screenshot created using SolidTA.

3.1 SolidTA

SolidTA is a program that can be associated with the two categories "project monitoring" and "metrics". According to SolidTA's website: "SolidTA is an integrated tool for visually analyzing the evolution of software projects stored in [...] repositories."[Groningen [no date]] The program gives a great overview over a project, its contributors and its change over time. The general user interface of SolidTA can be seen in Figure 3.1.

3.1.1 Setup

SolidTA is downloadable on the website of the University of [Groningen [no date]] but only available for Windows. After following the installation wizard and starting the application one can simply create a new project by clicking on the new button and filling in the required information (project type, protocol, url) to access the needed repository. To get first data from the repository it is required to load the created project and update the file list. Afterwards it is possible to acquire version data by updating the version list and then get the most useful informations by updating the contents. Finally to be able to work with the acquired data it is needed to compute metrics, so that one can activate filters and start analyzing.

3.1.2 Functionalities

SolidTA has a variety of functionalities. You can import projects from Git, SVN and CVS repositories. These projects can be analysed and then the different sections inspected. In the following paragraphs, we will describe the different features and analysed parts in detail.

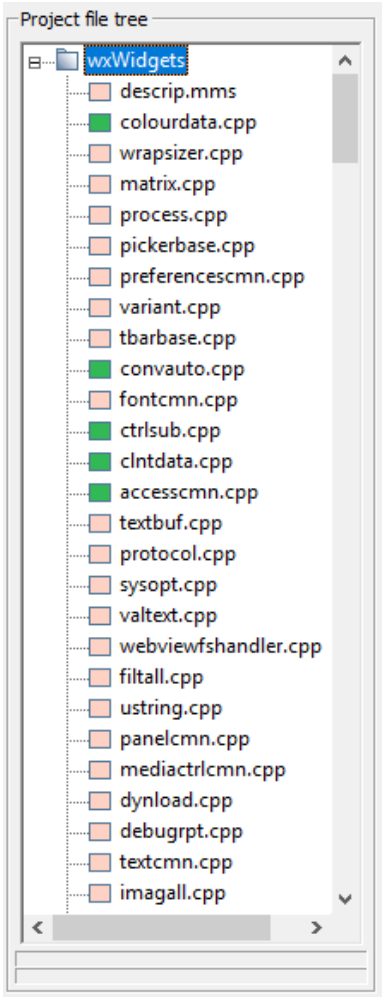


Figure 3.2: SolidTA: Project file tree. Screenshot created using SolidTA.

3.1.2.1 Project File Tree

On the left side of the UI there is a project file tree, where all the folders and files inside the repository can be seen [Figure 3.2]. When the box beside the file is white, then this means the file was only loaded into SolidTA, but you cannot get any information from it. If the box is light red, then this means some basic information is available, like version changes, commits and author data. If the box is green, then this means that all the information is available. So you can additionally filter by complexity, code size, lines of text and find specific methods.

3.1.2.2 Project Files Overview

In the middle of the SolidTA program all files of the loaded project are shown. You can see when these files where created and when they where changed, indicated by a yellow mark [Figure 3.3].

3.1.2.3 Project Filters

It is possible to filter though the project by: authors, file type, text, folder, complexity, methods, code size and lines of text [Figure 3.4]. If you select for example the filter category "authors" than all the authors are shown

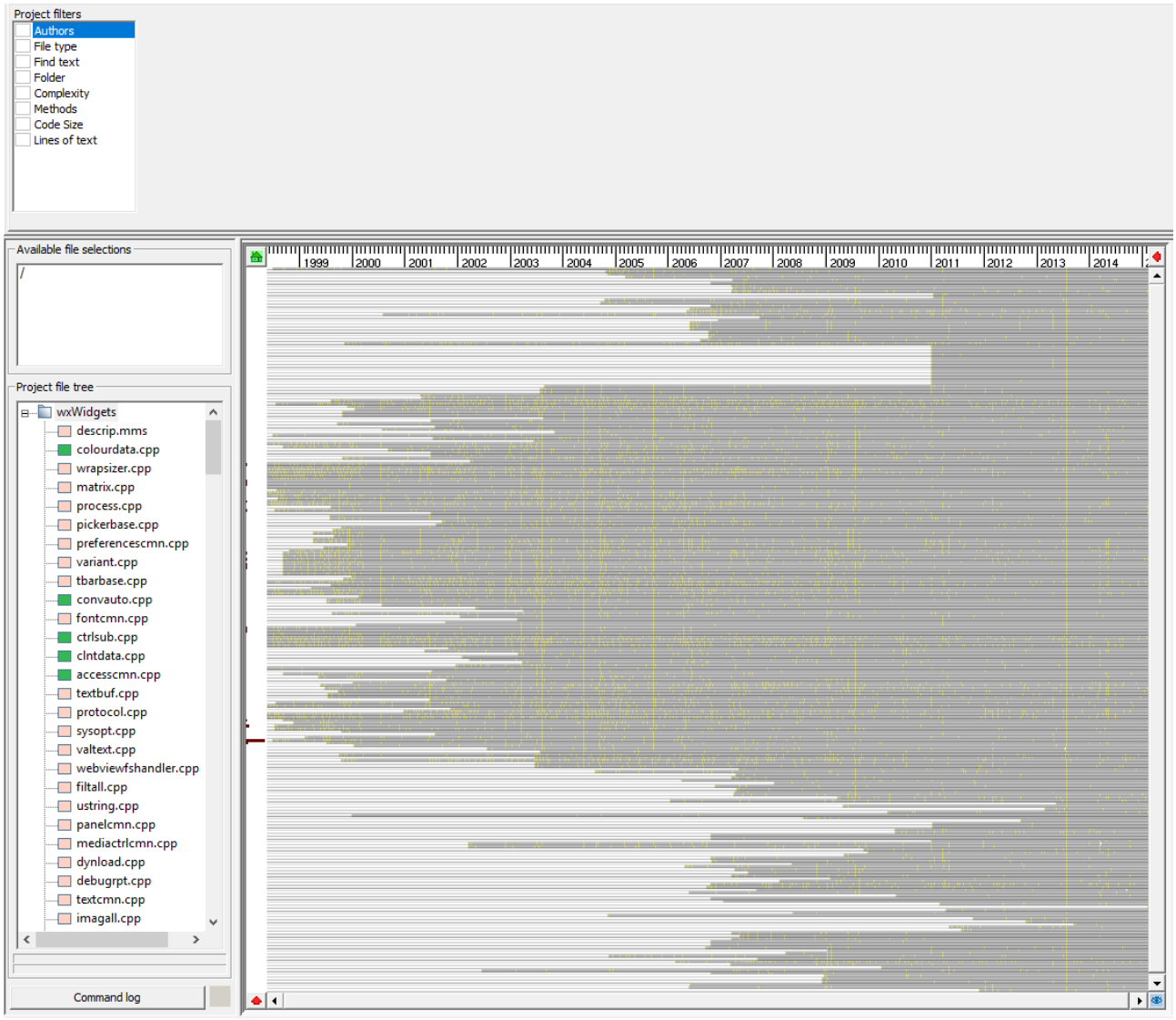


Figure 3.3: SolidTA: Project files overview. Screenshot created using SolidTA.

in color. Every author has its own color [Figure 3.5]. It is also possible to select one specific author and see when and on which files the selected author has worked on [Figure 3.6].

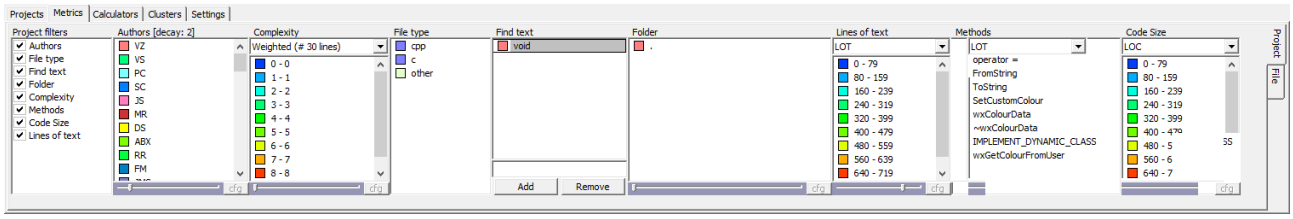


Figure 3.4: SolidTA: All filters selected. Screenshot created using SolidTA.

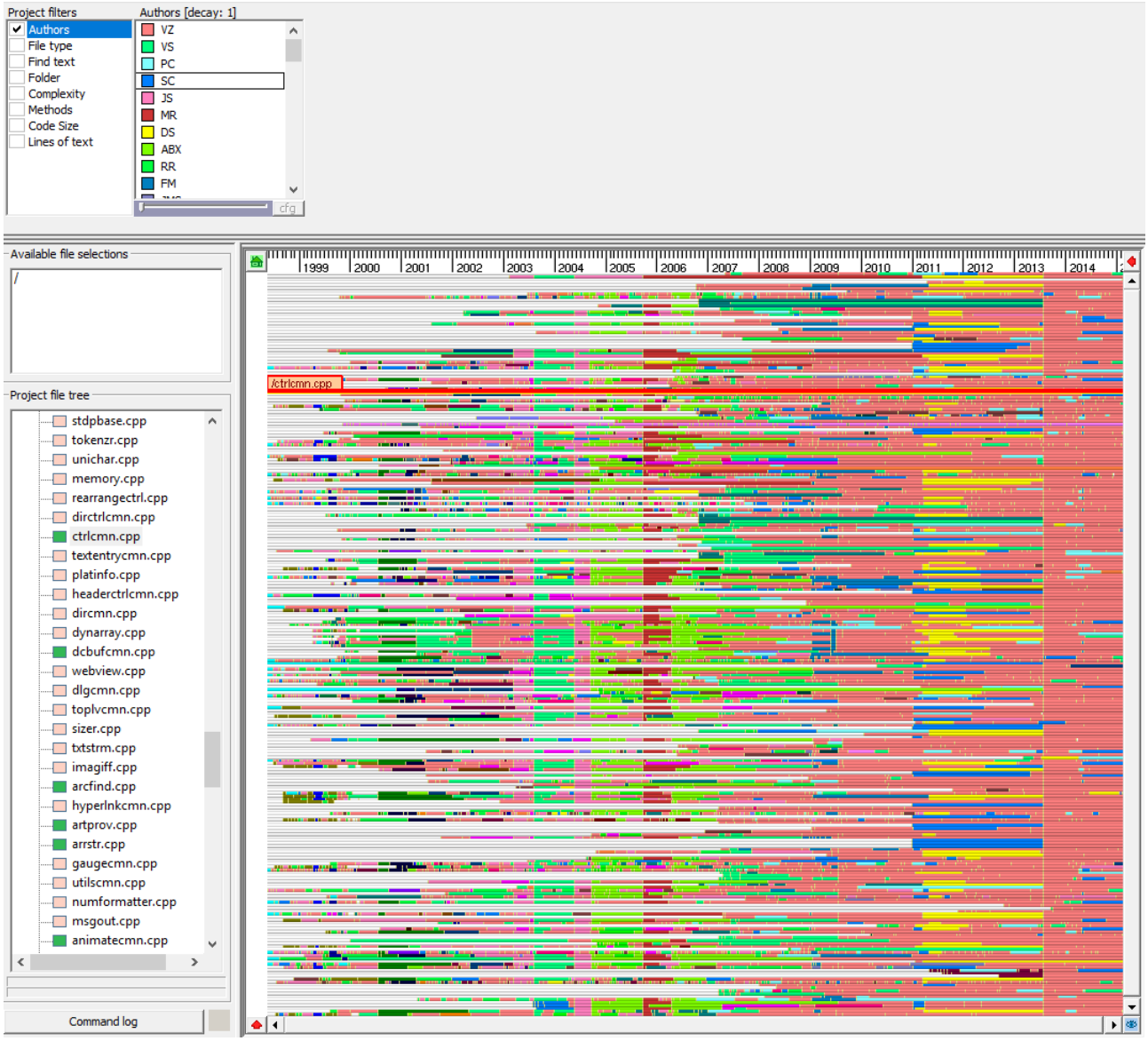


Figure 3.5: SolidTA: Filtered by authors. Screenshot created using SolidTA.

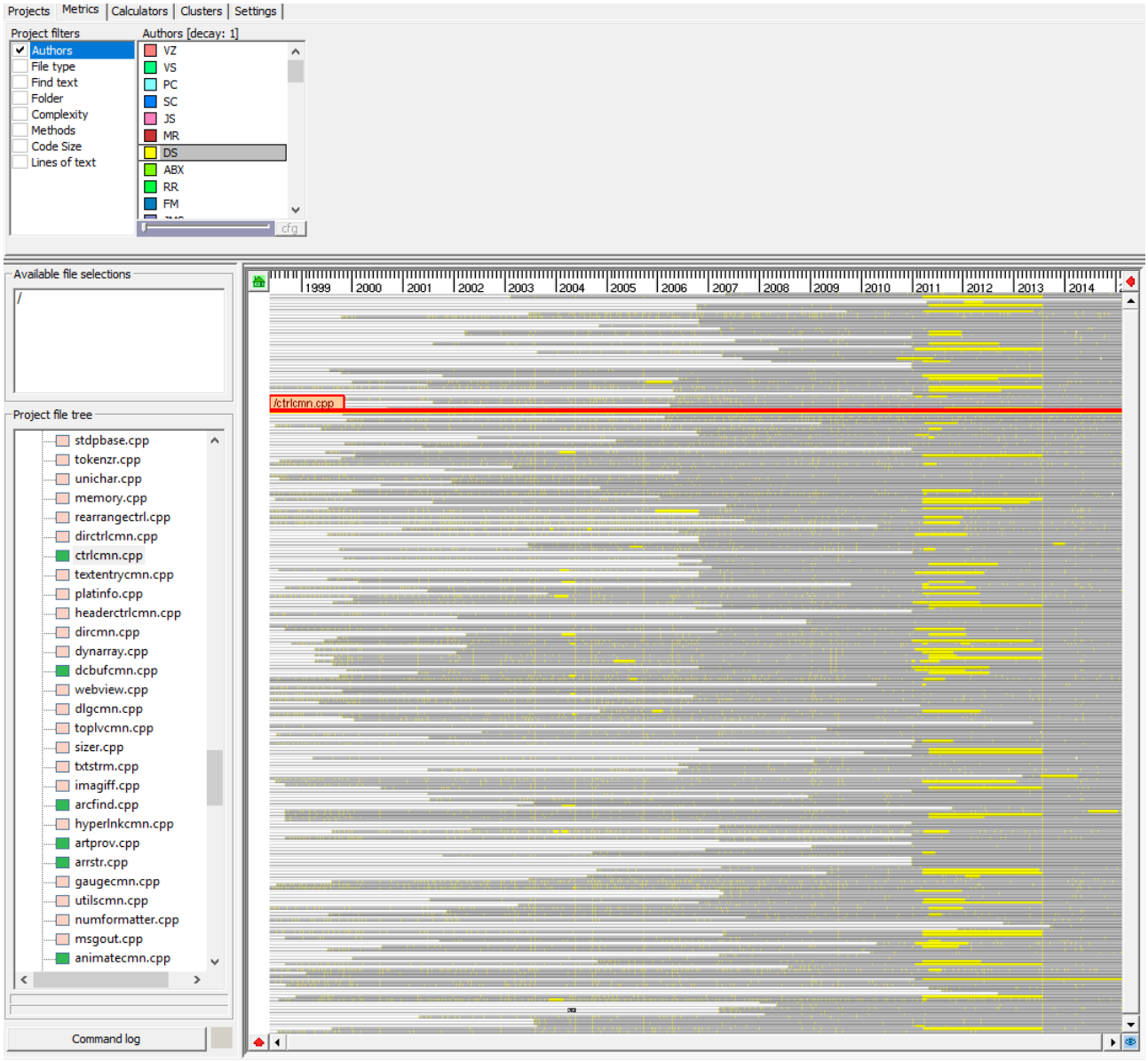


Figure 3.6: SolidTA: Filtered by a specific author. Screenshot created using SolidTA.

3.2 Gource

Gource is a flexible tool that can be used to display activity from your repository using a video visualization. It is available on all operating systems and can be redistributed and modified under the terms of the GNU General Public Licence. Gource generates a video from your versioning log and can then be used to navigate through the timeline. Therefore it belongs to the category of **Evolution of Code over Time**.

3.2.1 Structure

Software projects are displayed by Gource as:

- An animated tree with the root directory of the project at its centre.
- Directories appear as branches with files as leaves.
- Developers can be seen working on the tree at the times they contributed to the project.

Gource includes built-in log generation support for Git, Mercurial, Bazaar and SVN. Gource can also parse logs produced by several third party tools for CVS repositories.

3.2.2 Options

Gource has the following options to customize:

- `-- path /path/to/repo` (or omit and run Gource from the top level of the repo dir)
- `- f` (show full screen)
- `-- logo "InfoVisLogo.png"`
- `-- title "InfoVis"`
- `-- key` (shows colored legend for file types)
- `-- start-date '10.05.2018'`
- `-a 1` (auto skip to next entry if nothing happens in x seconds ? default 3)
- `- s .05` (speed in seconds per day: default 10)
- `-- camera-mode track` (follows the current active users or the selected user)

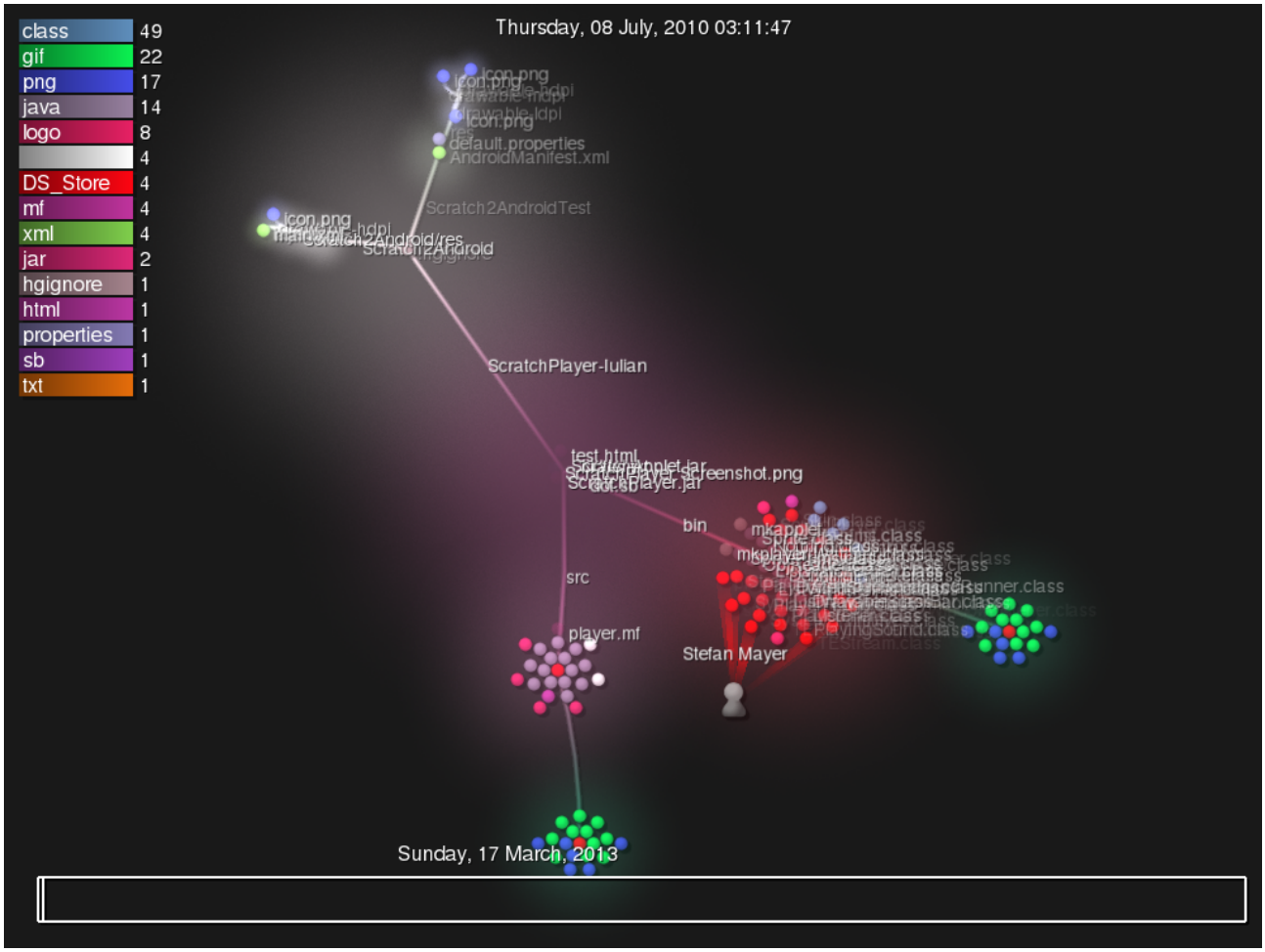


Figure 3.7: Gource on the Pocket-Code [Ebner et al. 2017] repository. Screenshot created using Gource.

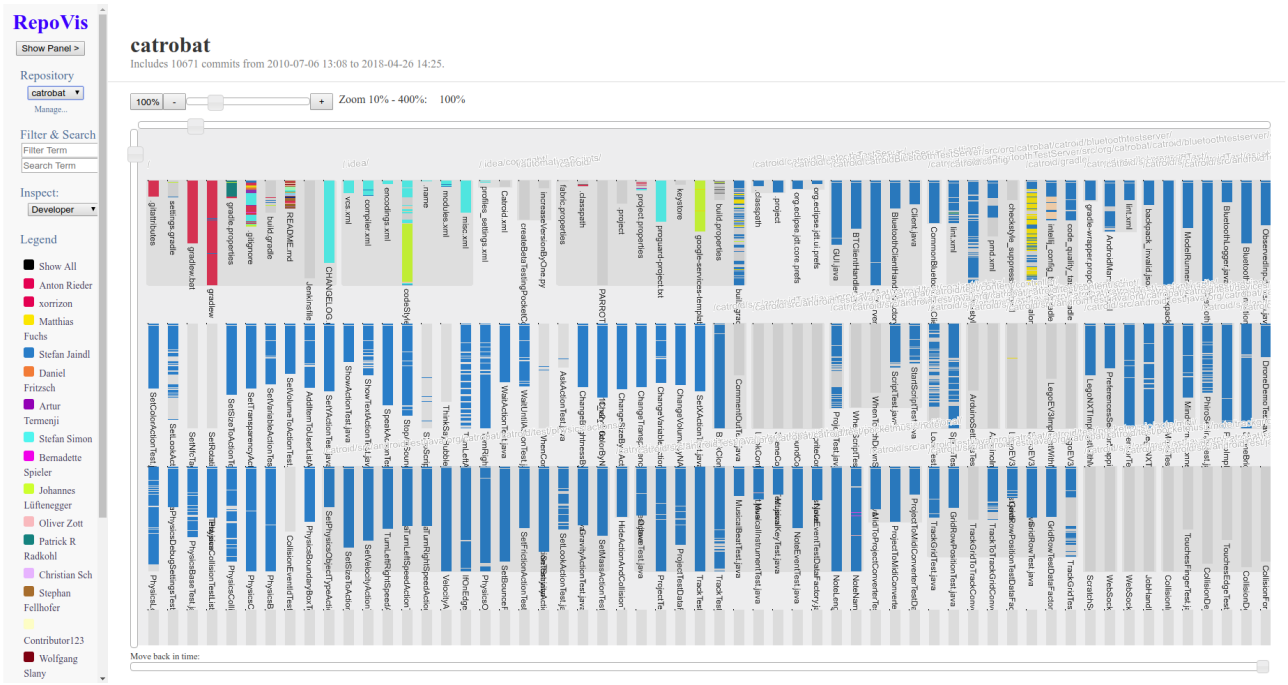


Figure 3.8: RepoVis on the Pocket-Code [Ebner et al. 2017] repository.Screenshot created using RepoVis.

3.3 RepoVis

3.3.1 Structure

RepoVis is by the time of writing this survey a PhD. project for the visualisation of Git repositories. It takes the traditions of Seesoft and improves it on the following aspects:

- Internal metrics:
  - Code(classes)
  - Authors
  - Lines of Code
- External metrics:
  - Usability issues: Group answers questions to the project to find problems.

It is implemented as a web application with a server-client structure. The reason for the server-client is that people should be able to upload their projects to the server which will be analyzing the repository in the background and then clients can access this analyzed data from each of the devices. The backend is built on Ruby with the framework Sinatra for preprocessing and the frontend is written in HTML5 and WebGL. RepoVis belongs to multiple categories.

3.3.2 Functions

In this section we want to show a few functions of the repository visualization tool RepoVis. It has potential to be a useful tool in the daily life of developers and project managers. We tested the following features on the

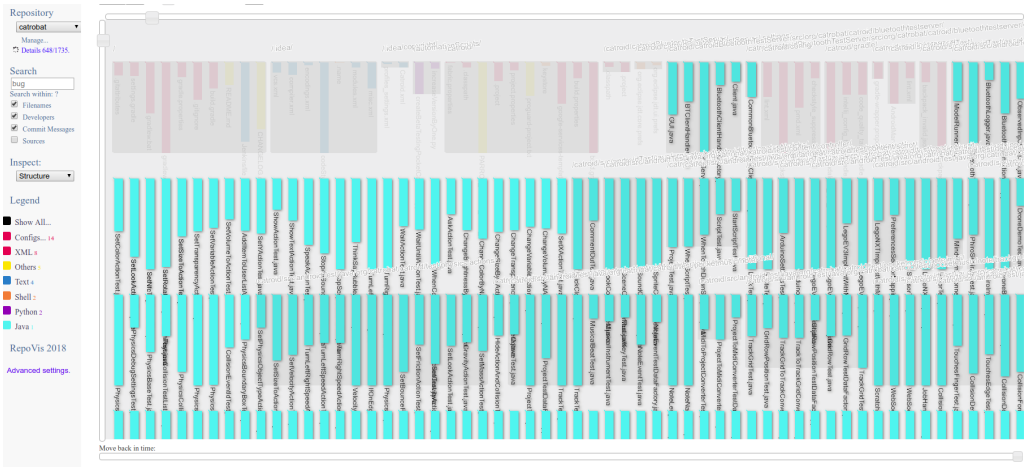


Figure 3.9: Highlight all java files. Screenshot created using RepoVis.

Figure 3.10: Files colored by latest change time. Screenshot created using RepoVis.

Pocket Code[Ebner et al. 2017] repository of the TU Graz.

3.3.2.1 Highlight

One big feature is to highlight files in the repository by different metrics. In the figure below we sorted our repository for Java-files.

3.3.2.2 Display Files by Latest Change

3.3.2.3 Usability Issues

RepoVis has a unique feature to integrate external data into its metric system. People can be asked to answer some questions about the program and find some issues. This issues can than be seen in RepoVis to make it

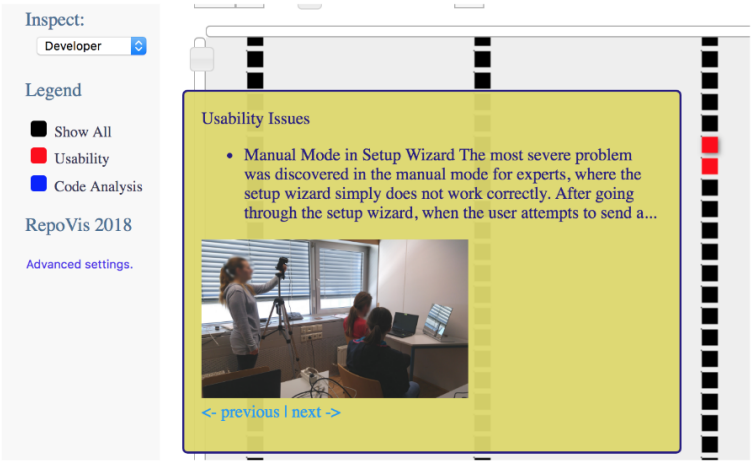


Figure 3.11: Usability findings can be inspected as they are shown on demand [Feiner and Andrews 2018]. Screenshot created using RepoVis.

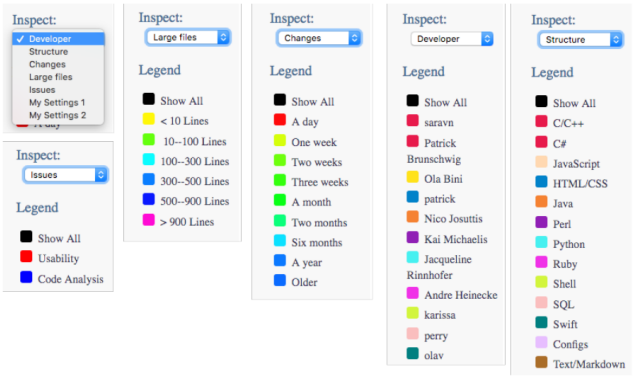


Figure 3.12: Presets enable fast switching between different views for different program comprehension tasks [Feiner and Andrews 2018]. Screenshot created using RepoVis.

easier for the developers to resolve them.

3.3.2.4 Presets

To make it easier for developers to get an overview of the project one can define presets with different filters. In the following figure we show which metrics can be used.

3.3.2.5 Others

In this section we just want to show some features which are still in development and some nice to have features:

- Timeline: Possible to go back in time and check project development.

- Customization: Change width, height, rotation of the display.
- Zoom: Slider to change zoom level.
- Actual code: Open panel with actual code content.

3.4 GitStats

GitStats is a History Statistics Generator, which aims to analyze, organize and display software repository data. GitStats is published under GNU General Public License v2. The date of publication is the 22 of August in 2007 on sourceforge. The application code is written in Python and the intended audience are developers.

3.4.1 Installation & Usage

The program is available on Linux as well as MacOS machines and can be downloaded as tar ball or directly installed via the apt-get package manager. The Gitstats repository is hosted on Gitorious and GitHub. There are 3 different options provided: A repository clone by a terminal command git clone git://github.com/hoxu/gitstats.git. Secondly, the download of a tarball https://github.com/hoxu/gitstats/tarball/master Last but not least, the installation and execution on a UNIX-based machine is recommended via the following terminal entries:

```
sudo apt-get install gitstats // Installation
gitstats [options] <src_gitpath> <dest_path> // Execution
options: -c key=value // Override configuration file
```

Listing 3.1: Gitstats: Recommended installation and execution command of GitStats

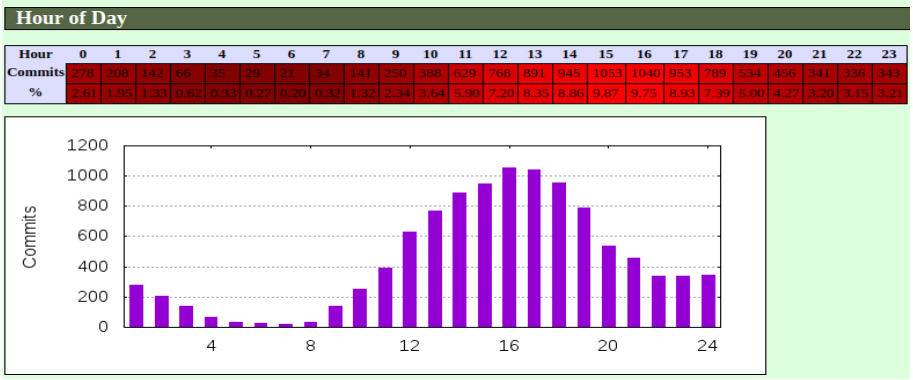
As mentioned in listing 3.1, GitStats is a terminal-executed program for Linux and MacOS platforms. The programs execution is achieved by a command expecting 2 parameters. First, the target repository path and secondly, a target-analyzing-repository path. This evokes the calculation of the ouput directory, which consists of js, .plot, .png, .dat and html-files and can be found under the path, specified at the previous execution. The content can be made visible via all standard web browsers. Gitstats produces line and bar charts as well as heatmaps over gnuplot 5.0 and table illustrations. A table sorting script included in the analyzing directory enables the user to sort all generated tables.

3.4.2 Functionality

This section is about the functional actions that are provided by Gitstats. Generally the features can be categorized in 5 parts. For simplification reasons, the explanation in the following text has merged categories.

3.4.2.1 General & Activity

General is the first section and allows the user to see general aspects such as project name, age, total files, lines of code, commits and authors. The activity category follows the general category and illustrates the commit statistic of the project team. Out of the commit data many different graphics of different chart types are derived. E.g. commits per hour per day, per week, per year. The following graphic illustrates the cumulated lines of code per author. The x-axes represents the time, the y-axes represents lines of code. [Figure 3.13]



Hour of Day

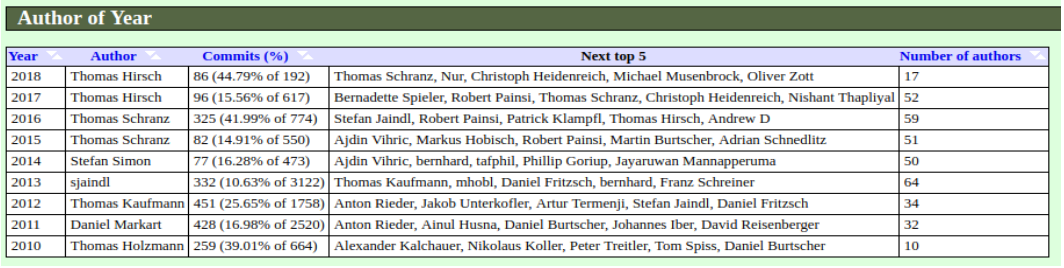
Hour	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
Commits	278	208	142	66	35	29	21	34	141	250	388	629	768	891	945	1053	1040	953	789	534	456	341	336	343
%	2.61	1.95	1.33	0.62	0.33	0.27	0.20	0.32	1.32	2.34	3.64	5.90	7.20	8.35	8.86	9.87	9.75	8.93	7.39	5.00	4.27	3.20	3.15	3.21

Figure 3.13: GitStats: Bar chart out of repository metadata. Screenshot created using GitStats

3.4.2.2 Authors

In addition to activity-based features, gitstats has a category with author-based statistics. A list of authors, cumulated lines of code per author, commits per author or a feature called author of the month are provided. Moreover the described section covers a author of the year and a illustration of commits by domains.

On the image above the author with the highest commit ratio is listed. In addition, the next top 5 following contributors are listed. The table has 6 columns in total, every row represents an expired year. Moreover, the table provides a sorting button on every column heading. [Figure 3.14]



Author of Year

Year	Author	Commits (%)	Next top 5	Number of authors
2018	Thomas Hirsch	86 (44.79% of 192)	Thomas Schranz, Nur, Christoph Heidenreich, Michael Musenbrock, Oliver Zott	17
2017	Thomas Hirsch	96 (15.56% of 617)	Bernadette Spieler, Robert Painsi, Thomas Schranz, Christoph Heidenreich, Nishant Thapliyal	52
2016	Thomas Schranz	325 (41.99% of 774)	Stefan Jaindl, Robert Painsi, Patrick Klampfl, Thomas Hirsch, Andrew D	59
2015	Thomas Schranz	82 (14.91% of 550)	Ajdin Vihric, Markus Hobisch, Robert Painsi, Martin Burtscher, Adrian Schnedlitz	51
2014	Stefan Simon	77 (16.28% of 473)	Ajdin Vihric, bernhard, tafphil, Phillip Goriup, Jayaruwan Mannapperuma	50
2013	sjaindl	332 (10.63% of 3122)	Thomas Kaufmann, mhobl, Daniel Fritzsch, bernhard, Franz Schreiner	64
2012	Thomas Kaufmann	451 (25.65% of 1758)	Anton Rieder, Jakob Unterkofler, Artur Termenji, Stefan Jaindl, Daniel Fritzsch	34
2011	Daniel Markart	428 (16.98% of 2520)	Anton Rieder, Ainul Husna, Daniel Burtscher, Johannes Iber, David Reisenberger	32
2010	Thomas Holzmann	259 (39.01% of 664)	Alexander Kalchauer, Nikolaus Koller, Peter Treitler, Tom Spiss, Daniel Burtscher	10

Figure 3.14: GitStats: Table showing the author of the year. Screenshot created using GitStats

3.4.2.3 Files & Tags

The final section is a summarization of the last 2 categories. The files category has sparse content, displaying one table with file extension types and one line chart containing files per date. he The ffinal category is the lines category with a sum of total lines and a lines of project evolution chart.

3.5 CodeFlower

The CodeFlower source code visualisation project is basically a science experiment of François Zaninotto. His project publication is under MIT license and is dated on the 4th of March 2013 and is mainly written in JavaScript and with regard to D3. The objective of CodeFlower is to visualise software repositories with a connected, interactive tree.

3.5.1 Installation & Usage

The implementation of CodeFlower is a procedure of three consecutive steps: Inclusion of D3 and CodeFlower libraries. Construction of JSON data and initialization of CodeFlower with structured data. Beginning with d3.js and CodeFlower.js file inclusion, the first step is trivial. Secondly, in order to create a correct repository visualisation, one has to generate a data file containing files, folders & lines of code. Achievable by different ways, this paper recommends cloc, a tool for counting lines of code in source code repositories. Listing 3.2 shows the exact steps for construction of cloc files.

After execution, cloc produces an exact image of all flower-relevant metadata in form of a cloc file. Feeding the output to a cloc to JSON converter on the project website (redotheweb.com/codeflower), the codeflower data is preprocessed. Finally, the codeflower instance has to be initialized. The constructor expects three parameters. A css selector, a width and a height. The update function has to be fed with the created JSON data and constructs the corresponding SVG element as you can see in listing 3.3

```
# Using git clone and cloc (slow, accurate)
$ git clone git://https://github.com/Catrobat/Catroid.git
$ cloc symfony --csv --by-file --report-file=Catroid_analyzed.cloc
```

Listing 3.2: Codeflower: Creation of cloc file with repository metadata

```
var myFlower = new CodeFlower("#visualization", 300, 200);
//CodeFlower(<CSS selector>, <"width">, <"height">)
myflower.update(jsonData);
//JsonData represents the the converted cloc file content
```

Listing 3.3: Codeflower: Initialization and implementation of the codeflower svg element

3.5.2 Functionality

The CodeFlower generates a bird's-eye-view over the software project folder. This overview is achieved by a constructed, interactive tree with a defined amount of nodes and edges in different sizes and different colours. A CodeFlower is an undirected graph in which any two vertices are connected by exactly one path. The root node represents the top-level-directory. By hovering over a bubble, one can see the lines of code in an absolute number. By clicking on a node, all its children nodes are merged to a bigger node. Another click on the newly created bubble reconstructs the previous node constellation.

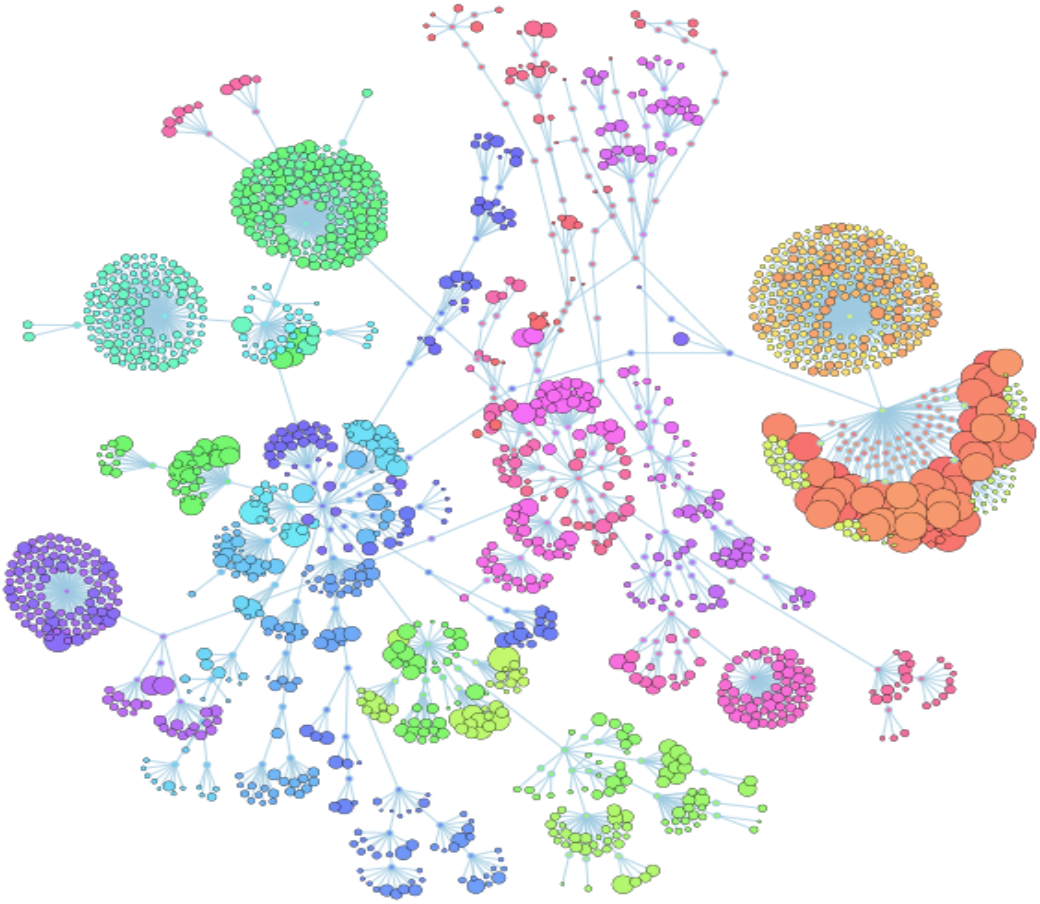


Figure 3.15: CodeFlower: Catroid project as flower with all files and directories within the project.

The Codeflower above is derived from the Catroid project. Every file represents a node with colour and a size attribute. All of these nodes are coloured according to their directory belonging. The size of a bubble is dependent on the amount of code lines. [Figure 3.15]

Bibliography

Bacher, Ivan, Brian Mac Namee and John Kelleher [2017]. “The Code-Map Metaphor - A Review of Its Use Within Software Visualisations”. In: Feb 2017 (cited on page 2).

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 12–14).

Eick, S.C., Joseph L. Steffen and Eric E. Sumner Jr [1992]. “Seesoft-A Tool for Visualizing Line Oriented Software Statistics”. 18 [Dec 1992], pages 957–968 (cited on page 1).

Feiner, Johannes and Keith Andrews [2018]. “RepoVis: Software Repository Visualisation with Integrated Code Metrics and Usability Issues”. 14th May 2018 (cited on page 15).

Groningen, University. *SolidTA tool*. `http://www.cs.rug.nl/svcg/SME/Tools` (cited on page 6).