Fundamentals of Software Analytics

Summer Term 2019
Assignment Sheet # 3

Faculty of Digital Engineering
Computer Graphics Systems Group
Prof. Dr. Jürgen Döllner

Objectives

With these assignments you will learn how to:

- Extract per-commit repository data and process data,
- Use statistics on source code to derive the degree of similarity,
- Use charting and visualization libraries for depiction of data, and
- Use dimensionality reduction techniques to prepare high-dimensional data for visualization.

For your programming tasks, you may use one of the following languages: Bash, Python, Batch, PowerShell, C, C++, or Java.

Task 3.1: De-Facto Coupling Matrix (4 Points)

For new features and adjustments to the software, often multiple source code files has to get changed. This introduces implicit – but de-facto – coupling between those source code files.

Task Implement a command-line tool that operates on a git Repository (the repository can be expected to be in the current working directory). The tool should perform the following steps in its data analytics and visualization pipeline:

a) Analyze each commit and its time window of 7 days[1] for files that are changed by the same author (identified by author email). The resulting data is a graph of source code modules as nodes and edges between source code modules that are changed together regarding this time window.

b) The resulting data should be visualized using a heatmap (see Figure 1). Thereby, the source code files are used for both columns and rows and each matrix cell has the mapped number of occurrences as its color value.

c) The visualization should use a sequential color scheme[2]; the color map legend is optional. The symmetry in the matrix is expected.

The result should be written as PDF to the file path provided as the first command-line argument (default is “result.pdf”).

Dataset Use the open source Github repository `https://github.com/cginternals/webgl-operate`. It can be accessed by cloning it:

```
> git clone https://github.com/cginternals/webgl-operate.git
```

Example

```
> cd webgl-operate
> ./3-1-de-facto-coupling-matrix output.pdf
> evince output.pdf
```

The result should be similar to the contents of Figure 1.

[1]An example: if a commit was issued on 2019-05-23 16:05:33, then the time window should include commits since 2019-05-20 00:00:00 until 2019-05-26 23:59:59.

[2]For more information, use `http://colorbrewer2.org/#type=sequential`.

Remarks You should not implement the visualization from scratch. Use a charting or visualization library that supports this kind of depiction. For this example, the python library *matplotlib* with its feature to depict 2d raster data using color maps was used.

Deliverables The submission should contain the following artifacts within the directory `3_1_de_facto_coupling_matrix`.

Source Code the sources of the tool

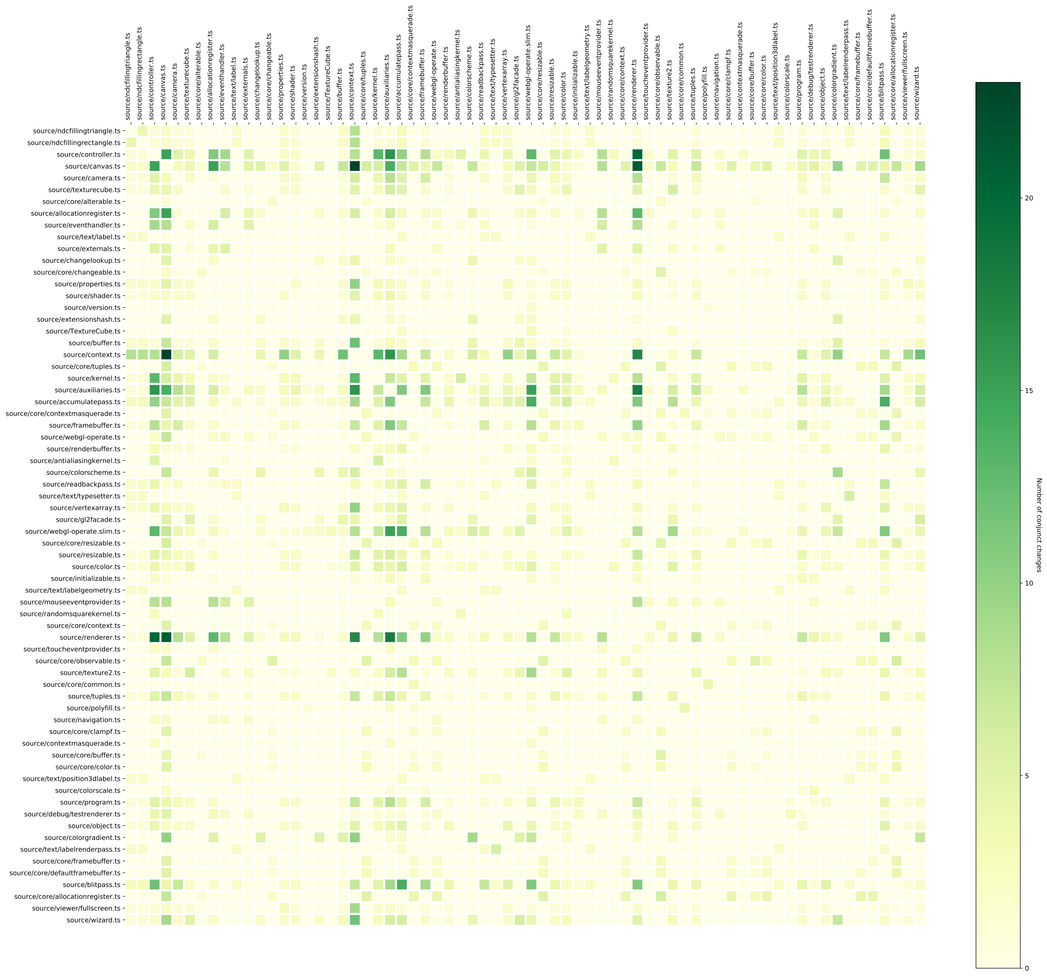

Figure 1: A heatmap visualization depicting the number of occurrences source code files got changed together using a 7-day time window.

Task 3.2: Commit Similarity Plot (5 Points)

The syntactic or semantic similarity of commits can be used to measure the homogenity in every-days programming. For example, clusters and outliers can provide insights on typical tasks and programming patterns.

Task Implement a command-line tool that operates on a git Repository (the repository can be expected to be in the current working directory). The tool should perform the following steps in its data analytics and visualization pipeline:

a) Create a vocabulary by collecting all changed lines over all commits and extracting the top 256 *words*[3].

b) Compute the Bag-of-Words vector for each commit by counting the occurrences of each *word* from the vocabulary. Use the Bag-of-Word vector together with the author of the commit for subsequent processing.

c) Use principal component analysis (PCA) to project the high-dimensional Bag-of-Word vectors to two dimensions.

d) Plot the results using a scatterplot and assign a per-author unique color to the plotted dots.

The resulting depiction should be similiar to Figure 2. The color map legend is optional as are the axes, the labels, and the text. The result should be written as PDF to the file path provided as the first command-line argument (default is “result.pdf”).

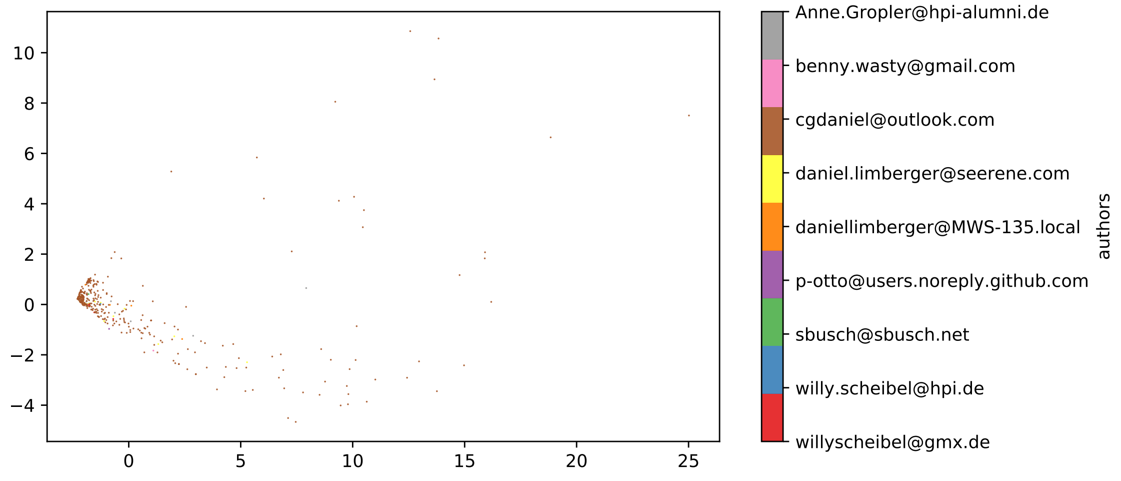

Figure 2: A depiction of commit similarity. Similar commits by means of similar changes are in close proximity whereas dissimilar commits are positioned far in the layout.

Dataset Use the open source Github repository `https://github.com/cginternals/webgl-operate`. It can be accessed by cloning it:

```
> git clone https://github.com/cginternals/webgl-operate.git
```

Example

```
> cd webgl-operate
> ./3-2-commit-similarity-plot output.pdf
> evince output.pdf
```

The result should be similar to the contents of Figure 2.

[3] A word may include words from natural languages as well as typical operators from programming languages. Only the natural word definition is required.

Remarks You should not implement the data processing and visualization from scratch. Use a charting or visualization library that supports this. For this example, the python library *scikit-learn* was used to count the words, extract the vectors, and perform the principal component analysis, whereas *matplotlib* was used to render the scatterplot.

Deliverables The submission should contain the following artifacts within the directory `3_2_commit_similarity_plot`.

Source Code the sources of the tool

Task 3.3: Topic Maps (5 Points)

Not using the inherent structure of the modules or the file system to derive a layout for visualization but the similarity between source code modules can result in other insights on the source code. For example, code clones and semantically near modules are placed in proximity and build easy-to-spot clusters.

This task should exemplify the processing pipeline that is used to create topic maps. The main result are the projected positions of source code modules. A more sophisticated visualization would improve on importance-based labeling and guidance to encode and decode the distances (e.g., using isolines).

Task Implement a command-line tool that operates on a git Repository (the repository can be expected to be in the current working directory). The tool should perform the following steps in its data analytics and visualization pipeline:

a) Create a vocabulary by collecting all source code from all current files and extracting the top 256 *words*[3].

b) Compute the Bag-of-Words vector for each source code module by counting the occurrences of each *word* from the vocabulary.

c) For each high-dimensional Bag-of-Word vector, use multi-dimensional scaling (MDS) to project it down to two dimensions.

d) Plot the results using a scatterplot and label each dot with the file name. You don't have to resolve overlaps and occlusion.

The resulting depiction should be similiar to Figure 3. The result should be deterministic (i.e., with no change in the input data, the result should be the same); displaying the axes is optional. The result should be written as PDF to the file path provided as the first command-line argument (default is "result.pdf").

Dataset Use the open source Github repository `https://github.com/cginternals/webgl-operate`. It can be accessed by cloning it:

```
> git clone https://github.com/cginternals/webgl-operate.git
```

Example

```
> cd webgl-operate
> ./3-3-topic-map output.pdf
> evince output.pdf
```

The result should be similar to the contents of Figure 3.

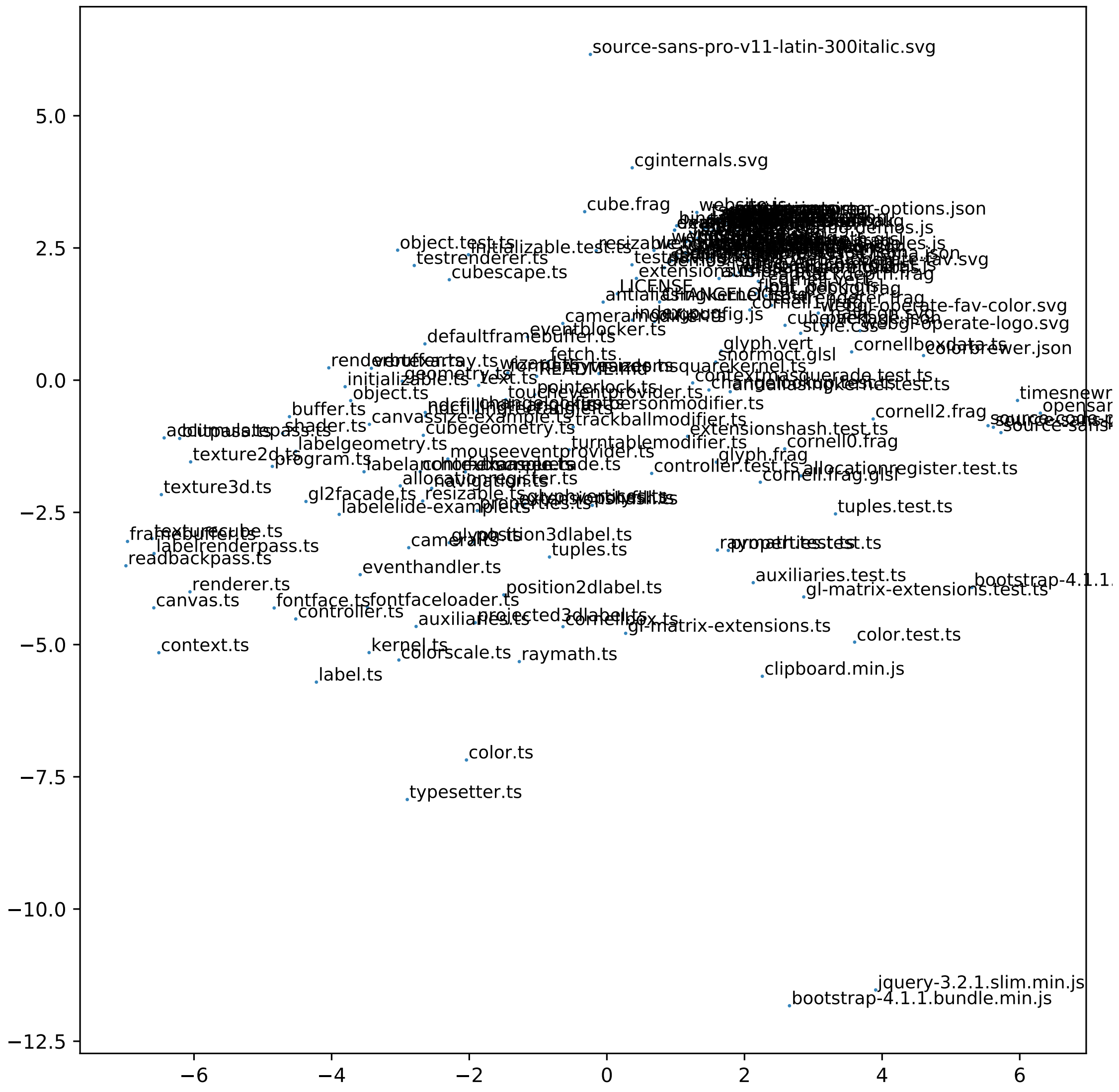

Figure 3: A topic map created by multi-dimensional scaling of Bag-of-Words feature vectors of the source code files.

Remarks You should not implement the data processing and visualiazion from scratch. Use a charting or visualization library that supports this. For this example, the python library *scikit-learn* was used to count the words, extract the vectors, and perform the multi-dimensional scaling, whereas *matplotlib* was used to render the scatterplot and text.

Deliverables The submission should contain the following artifacts within the directory `3_3_topic_map`.

Source Code the sources of the tool

Task 3.4: Author-File Relations Bundle View (4 Points)

Besides the visualization of measures and amounts, the depiction of relations allows to visualize structure and dependencies. As an example, the visualization of the edit dependencies of authors and source code files can help to identify process patterns and additional team structures.

Task Implement a command-line tool that operates on a git Repository (the repository can be expected to be in the current working directory). The tool should perform the following steps in its data analytics and visualization pipeline:

a) Iterate over all commits and collect each changed file and the author of the change (a bipartite graph of authors and modules as nodes).

b) Further collect the following metrics:
   - Number of Changes (NoC) per author
   - Number of Changes (NoC) per source code module
   - Mean Time Between Changes (MTBC) per author
   - Mean Time Between Changes (MTBC) per source code module

c) Output the graph using the following graph encoding format (see also Listing 1):
   - The file is structured into separate chunks. A chunk can introduce a set of nodes or a set of edges.
   - A chunk of nodes is structured as follows:
     - The line starts with `hierarchy` followed by the hierarchy identifier and the number of nodes to follow, each separated by semicolon ";".
     - For each node there follows a line structured as follows:
       - The line starts with `node`, followed by a type identifier of the node.
       - After that, the identifier of the node follows. The front slash "/" is used to encode the hierarchy. Parent nodes are implicitly created.
       - Then, two metric values must be provided, later mapped to weight and color.
       - All values are separated by semicolon ";".
   - A chunk of edges is structured as follows:
     - The line starts with `edges` followed by the edge list identifier and the number of edges to follow, each separated by semicolon ";".
     - For each edge there follows a line structured as follows:
       - The line starts with `edge`, followed by a type identifier of the edge.
       - After that, the identifier of the edge source follows.
       - Then, the identifier of the edge target follows.
       - All values are separated by semicolon ";".

You can use the output of the tool as input for the multi-hierarchical circular bundle view implementation provided on moodle[4]. An exemplary output is seen in Figure 4.

Dataset Use the open source Github repository `https://github.com/gorhill/uBlock`. It can be accessed by cloning it:

```
> git clone https://github.com/gorhill/uBlock.git
```

[4]The python script can be downloaded at `https://moodle.hpi3d.de/mod/resource/view.php?id=4003`.

Example

```
> cd uBlock
> ./3-4-author-file-relations-bundleview > graph.txt
> cat graph.txt | python3 ./mhcbv.py > output.pdf
> evince output.pdf
```

The result should resemble the contents of Listing 1 and a subsequent visualization should resemble Figure 4.

Listing 1: Excerpt from a graph file that is suitable for input of the multi-hierarchical circular bundle view implementation.

```
hierarchy;authors;5
node;author;rhill@raymondhill.net;15958;0.998771
node;author;noelle_leigh@fastmail.com;1;1.000000
node;author;okiehsch@users.noreply.github.com;1;1.000000
node;author;gwarser@users.noreply.github.com;3;0.914856
node;author;22801430+ZaphodBeebblebrox@users.noreply.github.com;1;1.000000
hierarchy;modules;5
node;module;platform/chromium/vapi-background.js;165;0.989456
node;module;dist/firefox/updates.json;219;0.997886
node;module;dist/version;247;0.998127
node;module;src/js/redirect-engine.js;45;0.969537
node;module;assets/assets.json;74;0.987823
edges;edits;5
edge;edit;rhill@raymondhill.net;platform/chromium/vapi-background.js
edge;edit;rhill@raymondhill.net;dist/firefox/updates.json
edge;edit;rhill@raymondhill.net;dist/version
edge;edit;rhill@raymondhill.net;src/js/redirect-engine.js
edge;edit;rhill@raymondhill.net;assets/assets.json
```

Deliverables The submission should contain the following artifacts within the directory `3_4_author_file_relations_bundle_view`.

Source Code the sources of the tool

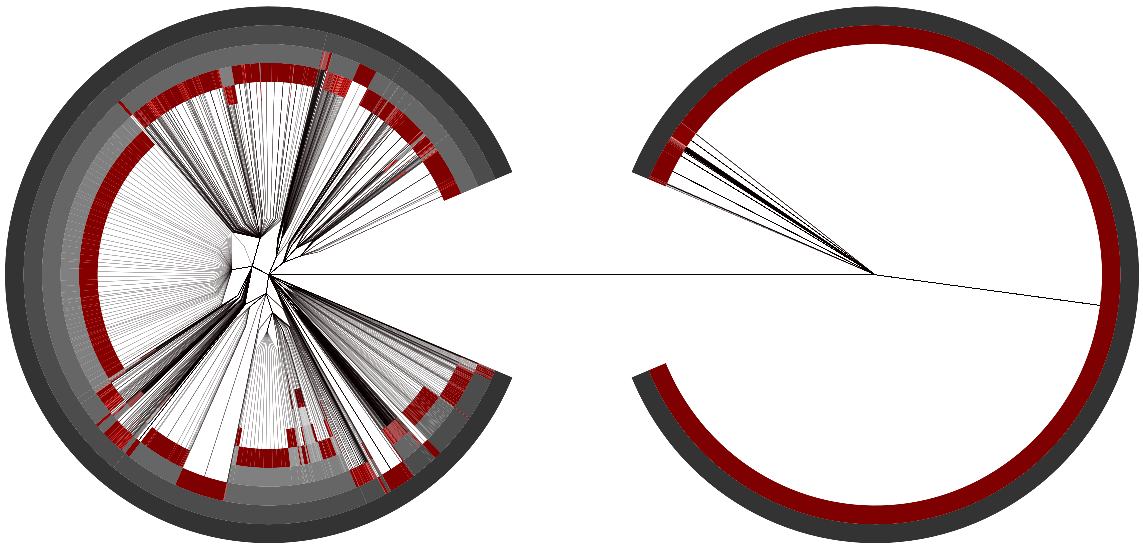

Figure 4: An example visualization of two hierarchies as hierarchical circular bundle view.

Instructions

Pair Programming On these assignments, you are encouraged (not required) to work with a partner provided you practice pair programming. Pair programming "is a practice in which two programmers work side-by-side at one computer, continuously collaborating on the same design, algorithm, code, or test." One partner is driving (designing and typing the code) while the other is navigating (reviewing the work, identifying bugs, and asking questions). The two partners switch roles every 30–40 minutes, and on demand, brainstorm.

Upload Results All described artifacts are submitted to the course moodle system `https://moodle.hpi3d.de/course/view.php?id=119` as a zipped archive with the following naming convention: `assignment_3_matrikNr1.zip` or `assignment_3_matrikNr1_matrikNr2.zip` whether or not pair programming was applied. Compiled, intermediate, or temporary files should not be included. If pair programming is used, the results are turned in only once. The assignments #3 are due on June 24th, 11:00 a.m.

Violation of Rules a violation of rules results in grading the affected assignments with 0 points.

- Writing code with a partner without following the pair programming instructions listed above (e.g., if one partner does not participate in the process) is a serious violation of the course collaboration policy.
- Plagiarism represents a serious violation of the course policy.