Fundamentals of Software Analytics

Summer Term 2019
Assignment Sheet # 1

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

Objectives

With these assignments you will learn how to

- gather structured information from a git repository using command–line interfaces,
- use data processing, normalization, and relation to augment a dataset,
- use basic visualization techniques to gain insights into datasets, and
- detect and correct flaws in datasets for improved automated workflows.

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

Remarks This exercise sheet can be used to get familiar with GNU command-line tools like `awk`, `cut`, `echo`, `grep`, `head`, `sed`, `sort`, `tail`, `tr`, `uniq`, `wc`, `xargs`, scripting, and pipes. When working on Windows, we recommend the use of the Windows Subsystem for Linux (WSL).

Dataset An example dataset that can be used to test the programming tasks 1.2–1.4 is the open source Qt base repository `https://code.qt.io/qt/qtbase.git`. It can be accessed by cloning it:

```
1  > git clone https://code.qt.io/qt/qtbase.git
```

Task 1.1: CSV File Format Audit (4 Points)

Automated processing of data requires documented, reliable, and easy-to-parse communication w.r.t. datasets and interfaces. As such, the comma-separated values format allows for unified parsing and handling of relational data in tabular format using rows and columns. Rows are delimited by a newline character (`\n`) and columns are separated by either comma (“,”), semicolon (“;”), or other visual separators such as tabulator (`\t`). To get an overview of a CSV-formatted dataset, tools as Visual Studio Code (via addon), Microsoft Excel or LibreOffice Calc can be used.

Task Review a given dataset file in CSV format. This CSV is an intermediate result within an automated software analytics processing pipeline. Compile (in means of *create*) a text file with short descriptions of at least four flaws that impedes a straight-forward and automated processing of the dataset. This task is of theoretical nature, i.e., an implementation is not required.

The text file should adhere to the following format:

```
<flaw 1>: <description \
using multiple lines>
- <excerpt from dataset>
+ <improved value>

<flaw 2>: <another description \
using multiple lines>
- <excerpt from dataset>
+ <improved value>

[...]
```

Dataset The dataset is hosted in the course moodle system at `https://moodle.hpi3d.de/mod/resource/view.php?id=3653`. This CSV is an intermediate result within an automated software analytics processing pipeline. The row delimiter is the newline character (`\n`) and the column delimiter is a semicolon (";"). The first line contains the header with column names and derivable semantics (e.g., if the column name includes "name", then the value is a string).

Deliverables The submission should contain the following artifacts within the directory `1_1_error_detection`.

Error Report The text file containing the error types, a descriptive text and a correct example. The error type and description may be written in English or German.

Task 1.2: Line Count (3 Points)

The *Lines of Code* metric is a measure for the size or *volume* of a source code repository. Looking at this metric in isolation won't provide insights but *Lines of Code* are useful for weighting other measures and gaining an overall impression of the size of a software system.

Task Implement a command-line tool that processes all files within the current git repository and computes the number of lines per file. The output should contain one line per file with the relative file path from the repository root and the number of lines within that file, separated by a semicolon (";").

Example The tool should output the information using the following format:

```
> cd qtbase
> ./line-count.sh
.gitattributes;4
.gitignore;325
.qmake.conf;7
.tag;1
INSTALL;10
LICENSE.FDL;450
```

Deliverables The submission should contain the following artifacts within the directory `1_2_line_count`.

Source Code the sources of the tool

Task 1.3: Author Consolidation (3 Points)

While working with version control systems, development activity of developers is recorded. That data enables for process-based metrics such as *Number of Developers*. Unfortunately, some developers provide inconsistent information for straight-forward tracking (e.g., differing email addresses). A dataset containing all distinct authors and all their email addresses enable for consolidation of the development activity and thus, more precise and consistent process metrics.

Task Implement a command-line tool that processes a list of author names and their email addresses. The tool should process a number of lines from standard input where each line contains the author name (semicolon-separated) and the author email address within angle brackets. The output of the tool is a list of consolidated author names and their joined email addresses. The author lines should be considered as same author if the author name matches[1]. Email addresses are not part of the author identity matching.

[1] The minimum requirement is a full match. Optionally, a matching algorithm using heuristics—such as the edit distance—can be used, too.

Example The tool should output the information using the following format:

```
> cd qtbase
> git log --format="%an;<%ae>" | ./author-consolidation.sh
Aaron Kennedy <aaron.kennedy@intopalo.com> <aaron.kennedy@nokia.com> <aaron.kennedy+qt@gmail.com>
Aaron Linville <aaron@linville.org>
Aaron McCarthy <aaron.mccarthy@jollamobile.com> <aaron.mccarthy@nokia.com> <amccarthy@sigtec.com> <mccarthy.
    aaron@gmail.com>
aavit <eirik.aavitsland@digia.com> <qt_aavit@ovi.com> <qt-info@nokia.com>
ABBAPOH <ABBAPOH@nextmail.ru>
Adam Majer <adamm@zombino.com>
Adam Reichold <adam.reichold@t-online.de>
```

Deliverables The submission should contain the following artifacts within the directory `1_3_author_consolidation`.

Source Code the sources of the tool

Task 1.4: Repository Mining (8 Points)

The version control system *git* allows to access information that are gathered throughout the development of a software system (refer to `git log`). In order to provide efficient and automated analytics workflows, the information should be gathered, filtered, and reformatted.

Task Implement a command-line tool (script or compilable executable) that gathers information about a git repository. The tool should gather the information from the current commit and its ancestor graph. Unreachable commits are not considered. After processing, the tool should output the following information to the standard output:

Number of authors The total number of different commit author emails
Author consolidation from Task 1.2 is not considered here. Be aware of the differentiation of commit authors and committers as only the former are subject to this metric.

Most active author within last year The name and email address of the author with most commits within the last 365 days
Author consolidation from Task 1.2 is not considered here.

Time range of development The number of days since the oldest ancestor commit
The number of days should be extended by a more human-readable output in terms of commenced years and months, whereby the year and month information should get dropped if if they are zero (examples: “started 3 months ago”, “started 2 years ago”, and “started 10 years, 1 month ago”).

Share of maintenance The fraction of commits targeting maintenance activities in relation to the total number of commits (print as percentage round to one decimal)
Commits that target maintenance are identified by any of the following keywords in their commit message: “fix”, “rewrite”, “doc”, “test”, or “improve”
This number should exclude merge commits and cherry-picks.

Share of stale code The fraction of files that were not changed within the last year (print as percentage round to one decimal)
A file is considered changed if it is mentioned in *diff* reports from the git repository (e.g., `git diff` or `git log`).

Top 10 commit message keywords within last month The top 10 most used keywords within the last 30 days in commit messages
A keyword is an alpha-numeric character list. Symbols are excluded except the hash symbol “#” as it is commonly used in commit messages to reference issues. The measure should exclude words from the commit body. Further, the following words should be excluded: “a”, “an”, “at”, “for”, “from”, “in”, “is”, “of”, “on”, “the”, “to”, “use”, “with”, and “when”.

Example The tool should output the information using the following format:

```
> cd qtbase
> ./repository-mining.sh
Number of authors: 1288
Most active author within last year: Tor Arne Vestbo <tor.arne.vestbo@qt.io>
Time range of development: 2911 days (started 8 ago)
Share of maintenance: 19.1%
Share of stale code: 79.6%
Top 10 commit message keywords last month: fix, and, add, qmake, windows, test, remove, tst, qpa, macos
```

Deliverables The submission should contain the following artifacts within the directory `1_4_repository_mining`.

Source Code the sources of the tool

Task 1.5: ASCII Art Visualization (5 Points)

Task Implement a command-line tool that reads a metrics file with keys and values from the command line standard input and output a bar chart in ASCII art with prepending the key to standard output. Additional functional requirements:

- The ASCII art symbol should be easily readible and discernible, e.g., `+`.
- The command line option `--attribute` should allow to select the attribute. If no attribute is selected, use the first one after the key.
- The command line option `--sort` should allow to sort the output by attribute value. Allow options to sort in ascending order and descending order.
- The a command line option `--limit` should allow to limit the number of lines of the output.
- For the output, the key should be interpreted as file path. If the command line option `--hierarchical` is passed, print the full file path. If this command line option is not passed (or `--flat` is passed), print only the file name part of the file path. Use the longest printed file path to left-pad the others to the same length.
- Separate the text from the ASCII chart depiction using a visual delimiter, e.g., `|`.
- The ASCII bar chart should use all remaining characters up to the column width. The default column width for the output should be 80 columns and be configurable via the command-line option `--columns`.

If the input exceeds the output constraints, the program behavior is undefined.

Dataset The dataset is hosted in the course moodle system at `https://moodle.hpi3d.de/mod/resource/view.php?id=3767`.

Example The resulting output should have the following structure:

```
> ./bar-chart.sh --flat --attribute=RLoC --sort=desc < merged-metrics.csv
     Server.cpp | ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
     Client.cpp | ++++++++++++++++++++++++++++++
FileWatcher.cpp | ++++++++++++++++++
       Rect.cpp | +++++++

> ./bar-chart.sh --hierarchical --attribute=Mccc --sort=desc --columns=40
  < merged-metrics.csv
             src/Client.cpp | ++++++++++
             src/Server.cpp | ++++++
       src/private/Rect.cpp | +++
src/private/FileWatcher.cpp | ++
```

Deliverables The submission should contain the following artifacts within the directory `1_5_bar_chart`.

Source Code the sources of the tool

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_1_matrikNr1.zip` or `assignment_1_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 #1 are due on Mai 13th, 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.