Fundamentals of Software Analytics

Summer Term 2019
Assignment Sheet # 2

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

Objectives

With these assignments you will learn how to:

- use a metric computation tool and visualization platform to gather insights into changes in source code,
- use different data sources to implement tailored metrics, and
- use text-based source code parsing to extract basic metrics.

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

Task 2.1: Per-Module Metrics Computation (7 Points)

The fine-grained measurement of source code and the version control system allow for a detailed assessment of the software, and, thus, allow for management and steering of its development. Software and its source code is usually structured in a hierarchical manner where multiple level of granularity are possible. Next to function, class, or namespace levels, the file-level is established as a shared level across all programming languages.

Task Write a command-line tool that, for each C and C++ source-code file within the current working directory and its subdirectories (recursively), computes the following metrics:

Mean Time Between Changes (MTBC) This represents the mean time in days between two revisions of a the source code file.
The period under consideration is the last 365 days of the current commit.

Number of Collaborators (NoC) This represents the number of different authors (identified through their email) for a source code file.
The period under consideration is the last 365 days of the current commit.

Botch Factor (BF) The botch factor (BF) is an ad-hoc compound metric of the *Mean Time Between Changes* (MTBC) and the *Number of Collaborators* (NoC) metric.
It is computed using $\mathrm{BF} = \mathrm{NoC}^2 \,/\, \mathrm{MTBC}$.

Object Size per LoC (OSpLoC) This is the ratio of the byte size of a compiled object file to the lines of code of its origin source code.
Your command-line tool may assume a compile commands database to match source code files to object files[1].

Share of Vulkan Code (SoVkC) The share of Vulkan code is the ratio of symbols that start with `vk` to the full list of symbols. This check is case-insensitive and counts every occurrence of a symbol per source code file. As a simplification, the symbols of a source code file are determined by extracting consecutive occurrences of alpha-numeric characters including the underscore[2].

Thereby, C and C++ source code files are files with a file extension being ".h", ".hpp", ".c", or ".cpp". The command-line tool should output the collected metric values to the standard output using the CSV format. The delimiter should be the semi-colon (";"). The first line of output is the header for a CSV-encoded file (see example header). The tool should output one line for each

[1] For example, CMake generates such a compile command database when setting the variable `CMAKE_EXPORT_COMPILE_COMMANDS` to `On`.

[2] In resemblance to the definition of an identifier in, e.g., C. See also `https://en.cppreference.com/w/cpp/language/identifiers`.

parsed file with its relative file name as the first column and the corresponding metric values for the succeeding columns.

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

```
> git clone https://github.com/SaschaWillems/Vulkan.git Vulkan-Examples
> cd Vulkan-Examples
> git submodule init
> git submodule update
```

This project depends on the Vulkan and Assimp libraries to get built. On Debian-based systems (e.g., Ubuntu), you can install it with

```
sudo apt install libvulkan-dev libassimp-dev
```

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

```
> cd Vulkan-Examples
> cmake . -DCMAKE_EXPORT_COMPILE_COMMANDS=On
> make
> ./2-1-per-module-metrics
# filename;MTBC;NoC;BF;OSpLoC;SoVkC
src/util/xmlpool/t_options.h;20;3;0.15;0;0
src/vulkan/util/vk_util.c;3;21;7;0;0.7
src/vulkan/util/vk_util.h;2;20;10;25.7;0.5
```

Deliverables The submission should contain the following artifacts within the directory `2_1_per_module_metrics`.

Source Code the sources of the tool

Task 2.2: Function Length per Parameter Visualization Tool (4 Points)

Task Implement a command-line tool that parses a list of given Python source code files (passed as standard input) and provides a depiction of their function body length (by means of Lines-of-Code) to parameter count distribution. The possible types of depictions should be dot plots and box plots including axis labels (see Fig. 1). The output is a PDF file containing the respective plot. Thereby, the program should be configurable and adhere to the following command line parameters:

`--output <filename>` The file name of the output PDF file (default is `output.pdf`).

`--type <type>` Define the plot type (either `box` or `dot`, default is `dot`).

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

```
> git clone https://github.com/pytorch/pytorch.git
```

Example The tool should be usable using the following command-line interface:

```
> cd pytorch
> find * -iname "*.py" | ./2-2-function-analysis --output result-dot.pdf --type dot
> find * -iname "*.py" | ./2-2-function-analysis --output result-box.pdf --type box
```

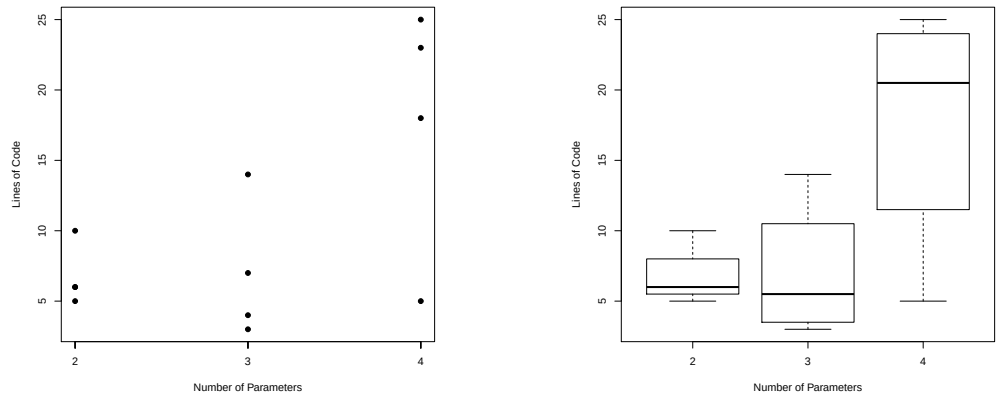

Figure 1: Possible outputs for the tool. A dot plot of the distribution of parameter count and function body length (left) and a corresponding boxplot (right).

Implementation Hints For a straight-forward implementation, the following assumptions and design decisions can be taken:

- ASCII is the only file encoding.
- A function always starts with the keyword `def` followed by a space character.
- A function starts at indentation level 0.
- The list of parameters is enclosed by parentheses and the parameters are comma-separated.
- The function declaration may contain line breaks.
- The function body is completed with the first line on indentation level 0.
- The output PDFs can be implemented using the R language with its core functionality to load CSV data (`read.csv`), to generate dot plots (`plot`) and boxplots (`boxplot`) and perform axis configuration (`axis`), and to execute the scripts with `Rscript`.

Deliverables The submission should contain the following artifacts within the directory `2_2_function_analysis_tool`.

Tool Source Code the sources of the tool

Task 2.3: Seerene Local Analyzer and Transparency Platform (4 Points)

The Seerene Transparency Platform is a Web service for software analytics. The Web service provides visualization, charting and analytics into pre-mined source code repositories. As part of this process, the Seerene Local Analyzer is used as the software metrics extraction tool.

Task Get familiar with the Local Analyzer and Transparency Platform workflows. Therefore, use the Local Analyzer to extract software metrics from an Open Source Github repository. Upload the resulting dataset to the Transparency Platform and analyze the changes to the software repository within the last 4 weeks. The result is a compilation of the top 10 source code files with respect to number of changes[3].

Dataset Use the per-group Open Source Github repository as stated in the moodle system at `https://moodle.hpi3d.de/course/view.php?id=119`.

[3]There may be multiple candidates for the last position—choose one—and there may be too few changed files—compile a shorter list.

Information on Seerene Local Analyzer and Transparency Platform A basic documentation with step-by-step instructions can be found on the moodle at `https://moodle.hpi3d.de/course/view.php?id=119`.

Deliverables The submission should contain the following artifacts within the directory `2_3_seerene_analysis`.

Report The list of the top 10 most-often changed source code files as basic text file.

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_2_matrikNr1.zip` or `assignment_2_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 #2 are due on June 3rd, 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.