Project Application Phase

Experiment Description

In this experiment data is fetched from the ‘GitHub-Archives’, then it is filtered to extract data of interest and afterwards the accumulated data of interest is used conduct statistical analysis. As a result two plots are created visualising the statistics.
Before describing the workflow in more detail, the aim of the experiment is explained. GitHub has a huge number of users and is widely used across the world. Thus there is continuous interaction of people and the system with each interaction generating an “system event”. These events are not only logged by GitHub but also archived openly, thus enabling everybody to access the list of collected events since 2015. More general, it is possible to extract all kind of information about github-usage or project management from this list of events.

This experiment aims to assess some basic statistics about GitHub usage as well as more detailed habits of GitHub users regarding commits. The basic statistics concern the percentage of different Event-Types in respect to the total number of captured Events. Thereby, the following Event-Types have been selected:

- Push-Events
- Pull-Request-Events
- Create-Events
- Delete-Events
- Fork-Events
- Watch-Events
- (Others)

For the analysis of user behavior regarding commits, Push-Events have been looked at more closely. Push-Events were extracted from the data and the number of commits within each Push-Event was studied. After iterating over all Push-Events, the following values regarding the number of commits per Push-Event were calculated:

- Average
- Minimum
- Maximum
- Median

After analysis and generation of statistics the results were visualized using a pie-chart for the basic data and a bar-chart.

As it may be interesting to compare different periods of time regarding those aspects, the experiment is not conducted on the whole archive but on a small part of it.

Implementation and Execution

The following figures are UML-Activity-Diagrams[1] depicting the workflow of the experiment. The basic workflow (shown in the first diagram) only consists of two activities and the following steps:

1. Data for the period of interest (specified by the user) is fetched from githubarchives.org
2. Data is Analyzed, Plot created

For each of these two activities (Fetch Data and Analyze Data) a Python3 script was written. These were not combined to enable accumulating data for multiple periods of interest before analyzing it. (The scripts should also be attached to this report)

These activities are described and shown in more detail:

1. Fetching Data
    a. The period of interest is split into queryable parts, as githubarchives.org only provides the events for one specific hour within one packed json-file when accessed via the HTML client. This means to collect the data for one day, 24 json-collections have to be retrieved and unpacked.

2. Analyze Data
    a. For analysis, each GitHub-Event has to be looked at individually. This is done in the activity “Process GitHub-Event” which gives a list with categorized event-data. This data-list is used to generate the statistics. The statistics are again used to generate the plots, which are saved as files.

3. Process Github Event
    a. As only some GitHub-Event-Types are of interest, irrelevant events are ignored (“Get Event Type”). For Push-Events the number of commits was extracted using a json-parser. For the simple Event-Types only the occurrence was noted.

4. Generate Statistics
    a. For the filtered event-types the percentage was calculated in regards to the total number of events
    b. For the commit-statistics average, minimum, maximum and median were calculated

E.g. execute the experiment for the time-period: April 1st 2015 00:00 - April 1st 2015 with each day from 8am to 1pm

[1] As taught in the lecture “Objektorientierte Modellierung”

[*commandline call: ./ue3_experiment_getData.py 2015 4 1 8 1 1 30 6*]
[*commandline call: ./ue3_experiment_analysis.py*]

The results given in the folder were created by examining only 01.01.2015 00:00 to 01:00. This is also the default time-period when calling getData.py without command-line parameters.