Github Focus Analyzer v1.0

Technical Specification

Author: Dorota Oleszczuk
University of Kent
July, 2018

Index

Overview

The GitHub Focus Analyzer (GFA) contains a series of scripts that save and analyze data for the predefined repositories in order to analyze developers' focus.

The GFA was created to download the information used in the *Are Good Developers More Focused?* Dissertation. This document details only the logic and processing aspects of the tool. For more details on the dissertation, contact the author.

Pre-Requisites

The GFA uses

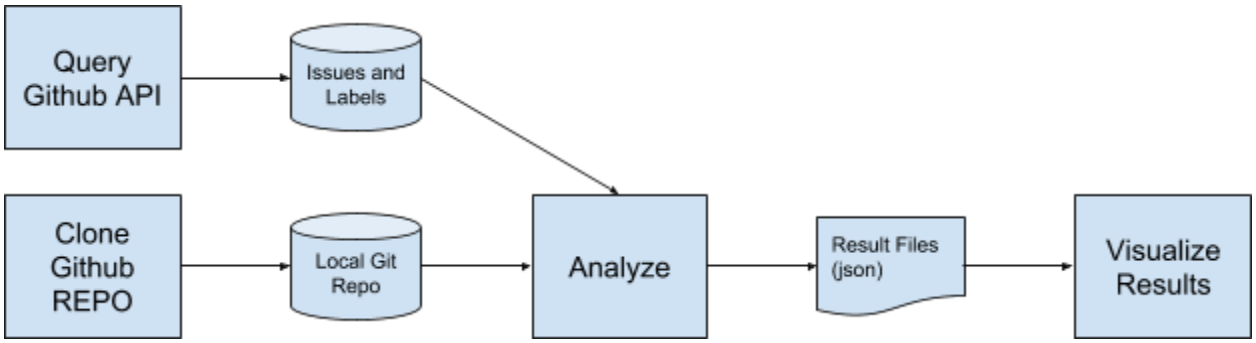
- Github REST API v3 which is available online and it is open for free use
- Git 2.18
- Python v3.0
- Javascript/HTML
- HighCharts.js

Scope

Once the Github connectivity is provided, the GFA will allow:

- Download the repository data from Github
- Download the labels associated with the repository
- Download the commits associated with the repository
- Download the issues associated with the repository
- Identify more information based on issues and commits
    - Identify *bugs*
    - Identify commits that are *fixes*
    - Identify commits that introduce *bugs*
    - Identify *good* and *bad* developers based on *fixes* and *bugs*
- Calculate the level of *focus* for a developer
- Test the information downloaded
- Visualize Results

Process



Technical Specifications

Download Data from Github

Clone Github Repository

Clones repos to the repos directory in the main folder. Includes the list of repositories to clone.

File	`/tools/load_repos.py`
Usage	`python3 load_repos.py`
Data Format	`Downloaded repository will be available in GIT`

By default, the following repositories are included:

- Elasticsearch
- Ansible
- Servo
- Bitcoin
- Selenium
- Spring-boot
- Rust
- TypeScript
- Symfony
- Rails

Download Labels

Download labels for each repository in the folders.

File	`/tools/labels.py`
Usage	`python3 labels.py -u <repo_user> <project>`
Storage	`/data/<project>/labels.json`
Arguments	`u : Repo owner` `project : Project name`

Data Format	`JSON`
	`{`
	` <label_name>: { score : <rate_unit> }`
	`}`

This script will also rate (rate unit) if a label is indication of a *bug* using a list of terms related to a *bug*.

Download Issues

Download issues for each repository in the folders.

File	`/tools/download_issues.py`
Usage	`python3 download_issues.py -u <repo_user> <project>`
Storage	`/data/<project>/issues.json`
Arguments	`u : Repo owner` `project : Project name`

Data Format	`JSON`
	`Full issue data returned by GitHub API`
	`GitHub API Issues`

Extract Bugs

Select issues that are *bugs* using rated labels. Issues with score > 0 are considered *bugs*.

File	/tools/extract_bugs.py
Usage	python3 extract_bugs.py <project>
Storage	/data/<project>/bugs.json
Arguments	project : Project name

Data Format	JSON
	{
	"assignee": <assignee>,
	"body": <issue message>,
	"closed_at": <closed date>,
	"created_at": <created date>,
	"labels": [
	{
	<full label data>
	}
	],
	"number":<issue number>,
	"state": <issue state>,
	"title": <issue title>
	}

Inputs	/data/<project>/issues.json, /data/<project>/labels.json

If test is True, then it will save ./data/<project>/test/test_bugs.json with positive and negative results for verification.

Extract Commits

Save commits data for the time period : 01.01.2014 - 31.12.2017

File	/tools/extract_commits.py
Usage	python3 extract_commits.py <project>
Storage	/data/<project>/commits.json
Arguments	project : Project name
Data Format	JSON [{ "author_email": <email>, "author_login": <login>,

	"commit": <commit sha>, "message": <title + message>, "time": <unix time> }]

Extract Fixes

Extract commits that are *fixes*. It performs syntactic and semantic analysis according to the SZZ algorithm. It creates a tuple (commit, message, number(possible bug), syntactic score, author email, semantic score).
Each of the following criteria increases the syntactic score by one:

- Is bug number or a hash number
- Is a plain number or contains a keyword

If a test or a rollback of a commit is found, the syntactic score is set to 0.
Semantic analysis verifies whether the number in the link is a bug number, then it increases the score by one for each of the following criteria:

- if the issues has been marked as fixed or closed,
- If the bug assignee is the author of the commit,
- If the bug title is included in the commit message

A commit is considered a *fix* if:
`( syntactic score > 0 and semantic score = 1 )  or  semantic score > 1`

File	/tools/extract_fixes.py
Usage	python3 extract_fixes.py <project>
Storage	/data/<project>/fixes.json
Arguments	project : Project name
Data Format	JSON [<commit sha>]
Inputs	/data/<project>/commits.json, /data/<project>/bugs.json

If test is True, then it will save /data/<project>/test/test_fixes.json with positive and negative results for verification.

Extract Buggy Commits

Extract commits that introduced the *bugs*.
The file will contain the commit sha and the number of buggy lines across the files of the commit.
The script performs the following steps according to the SZZ algorithm:

- For each fix get the deleted lines and corresponding file, and the parent revision
- Run `git blame` to obtain commit that inserted the lines
- Save the blamed commit with the lines and corresponding file

File	`/tools/extract_buggy_commits.py`
Usage	`python3 extract_buggy_commits.py <project>`
Storage	`/data/<project>/buggy_changes.json`
Arguments	`project : Project name`

Data Format	`JSON`
	`{`
	`  <commit sha> : [<line number>, <file>]`
	`}`

Inputs	`/data/<project>/commits.json,` `/data/<project>/fixes.json`

If test is True, then it will save test_buggy_lines.json, test_buggy_lines_dups.json, test_buggy_fix_diffs.json for verification.

Annotate Commits

The script annotates each commit from commits.json with number of good and buggy lines, list of files modified and the changes per file.

File	`/tools/annotate_commits.py`
Usage	`python3 annotate_commits.py <project>`
Storage	`/data/<project>/commit_changes.json`
Arguments	`project : Project name`
Data Format	`JSON` `{`

	"author_email": <email>,
	"author_login": <login>,
	"buggy": <number of buggy lines in the commit>,
	"Commit": <sha>,
	"files": [
	{
	"deleted": <list of deleted lines per file>,
	"inserted": <list of inserted lines per file>,
	"new": <filename>,
	"old": <filename>
	},
	"good":<number of good lines in the commit>,
	"inserted":<number of inserted lines in the commit>,
	"message": <commit message>,
	"time": <unix time>
	}

Inputs	/data/<project>/commits.json, /data/<project>/buggy_changes.json

Prepare Developers

Group the annotated commits per author, per date and remove the non-prolific developers according to the cutoff value : bottom 35%.

File	/tools/prepare_devs.py
Usage	python3 prepare_devs.py <project>
Storage	/data/<project>/authors.json
Arguments	project : Project name

Data Format	JSON
	{
	<author email>: {
	<date>: {
	"changes": [
	<number of buggy lines>,
	<number of good lines>,
	<number of inserted lines>,
	<daily ratio>
	],

	"commits": [
	{
	<sha>: [
	{
	"deleted": <deleted lines>,
	"inserted": <inserted lines>,
	"new": <filename>,
	"old": <filename>
	}
	] # end of files
	} # end of commit
	] # end of commits
	} # end of date
	}

Inputs	/data/<project>/commit_changes.json

If test is True, then it will save /data/test/<project>/test_sums.json for verification.

Combine Developers

Combine authors from different repositories into a single file and get the buggy-to-inserted ratios for all. Some authors appear across the repositories, they are combined based on the email.

File	/tools/combine.py
Usage	python3 combine.py
Storage	/data/authors_combined.json, /data/authors_ratio.json
Arguments	-

Data Format	Combined JSON
	Same format as authors.json
	
	Ratio JSON
	[
	{
	"commits": {
	"buggy": <sum of buggy lines across commits>,
	"daily_ratio": <average daily ratio>,
	"good": <sum of good lines across commits>,
	"ratio": <overall ratio>,
	"sum": <sum of inserted lines across commits>
	},

	"dev": <author email> }]
Inputs	/data/<project>/authors.json

Select Developers

Group developers into “good” and “bad” according to the ratio limit. Default limit is 0.

File	/tools/select_devs.py
Usage	python3 select_devs.py
Storage	/data/<project>/selected.json
Arguments	-l: Ratio limit for good developers

Data Format	JSON
	{
	bad: [<author email>, …],
	good: [<author email>, …]
	}

Inputs	/data/authors_combined.json, /data/authors_ratio.json

If test is True, then it will save /data/test/selected_test.json for verification.

Analyze Focus

Runs analysis of developers focus based on extensions. The analysis is done for different time periods : day, week, year, 4 years.
The analysis is done based on entropy. Entropy is computed by fixing either developer-date pair, or developer-extension pair.
The following files are used by the visualization scripts to create charts:

- data/entropy_daily.json,
- data/entropy_yearly.json,
- data/entropy_per_year.json,
- data/entropy_per_dev.json,
- data/entropy_all.json,
- data/entropy_ranges_vis.json,
- data/entropy_ext_distribution.json
- data/entropy_exts_popular

File	/tools/analyze_focus.py
Usage	python3 analyze_focus.py
Storage	data/days_good.json, data/days_bad.json, data/entropy_daily.json, data/entropy_yearly.json, data/entropy_per_year.json, data/entropy_per_dev.json, data/entropy_all.json, data/entropy_ranges.json, data/entropy_ranges_vis.json, data/entropy_weekly.json, data/entropy_weeks_analyzed.json, data/entropy_extensions.json, data/entropy_groups.json, data/entropy_ext_distribution.json data/entropy_exts_popular.json
Arguments	-

Data Format	**days_good JSON**
	{
	  <author_email>: [
	  [<date>, <list of extensions per day>]
	  ]
	}
	
	days_bad JSON
	Same format as days_good
	
	entropy_daily JSON
	{
	  <group_name>: [
	  {
	  "color": <color>,
	  "data": [
	  [<date>, <daily entropy>]
	  ],
	  "name": <author email>
	  }
	  ]
	}
	entropy_yearly JSON
	{

```
  <group_name> : [
    {
      "color": <color>,
      "data": [ <entropy 2014>, <entropy 2015>,
<entropy 2016>, <entropy 2017> ],
      "name": <author email>
    }
  ]
}
```

entropy_per_year JSON

```
{
  <year> :
    {
      <group_name>: [
        [<author_email>, <yearly entropy>]
      ]
    }
}
```

entropy_per_dev JSON

```
{
  <group_name> : [
    [
      <buggy ratio per developer>,
      <overall entropy per developer>
    ]
  ]
}
```

entropy_all JSON

```
{
  <group_name> : [
    [<author email>, <overall entropy>]
  ]
}
```

entropy_ranges JSON

```
{
  <group_name>: {
    "0.001": <number of devs in a range>,
    "0.01": <number of devs in a range>,
    "0.1": <number of devs in a range>,
    "1": <number of devs in a range>,
    "2": <number of devs in a range>,
    "3": <number of devs in a range>,
    "4": <number of devs in a range>,
    "5": <number of devs in a range>
```

```
  }
}
```

entropy_ranges_vis JSON

```
{
  <group_name>: [
    <number of devs in 1st range>,
    <number of devs in 2nd range>,
    <number of devs in 3rd range>,
    <number of devs in 4th range>,
    <number of devs in 5th range>,
    <number of devs in 6th range>,
    <number of devs in 7th range>,
    <number of devs in 8th range>
  ],
  "categories": [
    "0-0.001",
    "0.0011-0.01",
    "0.011-0.1",
    "0.101-1",
    "1.001-2",
    "2.001-3",
    "3.001-4",
    "4.001-5"
  ]
}
```

entropy_weekly JSON

```
{
  <group_name>: {
    <author email>: {
      <year>: {
        <week number>: <weekly entropy>
      }
    }
  }
}
```

entropy_weeks_analyzed JSON

```
{
  <group_name>: {
    <author email>: {
      "different": <number of weeks with weekly entropy
different than sum of daily entropies for the same
week>,
      "same": <number of weeks with weekly entropy the
same as sum of daily entropies for the week>,
```

```
      "similar": <number of weeks with weekly entropy
similar sum of daily entropies for the week>
    }
}
```

entropy_extensions JSON

```
{
  <group_name>: {
    <author email>: {
      <extension>: <extension entropy>
    }
  }
}
```

entropy_groups JSON

```
{
  <group_name>: {
    <author email>: {
      <date>: {
        "changes": [
          <number of buggy lines>,
          <number of good lines>,
          <number of inserted lines>,
          <daily ratio>
        ],
        "commits": [
          {
            <sha>: [
              {
                "deleted": <deleted lines>,
                "inserted": <inserted lines>,
                "new": <filename>,
                "old": <filename>
              }
            ] # end of files
          } # end of commit
        ] # end of commits
      } # end of date
    } # end of author
  } # end of group
}
```

entropy_ext_distribution JSON

```
{
  <group_name>: [
    [
      <author - extension pair>,
```

	<entropy for the author-extension pair>
	]
	]
	}
	
	Entropy_exts_popular JSON
	{
	<extension>: {
	<group_name>: {
	"color": <color>,
	"data": [<author email>, <extension entropy>],
	"name": <name>
	}
	}
	}

Inputs	/data/authors_combined.json, /data/selected.json

If test is True, then it will save data/test/test_years_good.json, data/test/test_years_bad.json, data/test/test_overall_bad.json, data/test/test_overall_good.json, data/test/test_years_count.json, data/test/test_weekly.json, data/test/test_unique.json for verification.

Other Functions

Helper Functions

Auxiliary functions that allow multiple actions on the repositories, files and data.

File	/tools/helper.py
Usage	from helper import <function>

Logger

Auxiliary script that will log details of the execution to a console or a text file.

File	/tools/logger.py

Usage	`from logger import log_debug`
	`log_debug(<project>, <message>, <data tuple>)`
	`log_info(<project>, <message>, <data tuple>)`
	`log_error(<project>, <message>, <data tuple>)`

Storage	`github-debug.log` `github-error.log`

Visualizations

Visualize Developers

Create charts with inserted lines per developer for the selected repository. One chart depicts all developers and amount of lines inserted over 4 years. Remaining charts depict timeline of developer's commits. Each commit is divided into good and buggy lines.

File	`/tools/visualization/developers.html`
Script	`/tools/visualization/scripts/developers.js`
Usage	`Open html page in the browser. Full usage instructions are explained on the html page.`

Visualize Developers Ratios

Create charts with ratio of developer's buggy to all lines inserted.

File	`/tools/visualization/developers-ratios.html`
Script	`/tools/visualization/scripts/developers-ratios.js`
Usage	`Open html page in the browser. Full usage instructions are explained on the html page.`

Visualize Focus

Create charts to visualize the focus analysis results.
Charts include:

- Overall good vs bad - Developers graphed according to their buggy ratio and overall entropy

- Overall area - Good and bad developers graphed with their entropy for 4 years
- Overall ranges - Entropy ranges and the number of developers falling into each range
- Four yearly graphs - Good and bad developers with their yearly entropy values
- Two daily entropy graphs - One for good and one for bad developers with their daily entropy values
- Extension entropy distribution for developer - Graph for both good and bad developers. It shows developer-extension pai on the x-axis and entropy for the given pair on the y-axis.
- Focus for most popular extensions - Graph for most popular extensions depicting all developers who work on the extension and their corresponding entropy values.
- Focus for most popular extensions with logarithmic y axis - same as above but with logarithmic y axis.

File	`/tools/visualization/developers-focus.html`
Script	`/tools/visualization/scripts/developers-focus.js`
Usage	`Open html page in the browser. Full usage instructions are explained on the html page.`

Test Cases

The GFA includes a library of unit tests that will confirm if the data downloaded and the analysis are correct.

Testing Buggy Lines

Methods that verify the results from extract_buggy_lines.py and annotate_commits.py

File	`/scripts/tests/test_buggy_lines.py`
Usage	`Run the script from tools directory.` `python3 test/test_buggy_lines.py`
Data Required	`commits_changes.json` `buggy_changes.json`

Tests

Test	Description
`test_buggy_annotated`	Test whether the buggy lines were correctly accounted for in the annotated changes.
`test_annotated_sums`	Test annotated lines sums = good + buggy
`test_buggy_duplicates`	Out of all the buggy commits, if there is a duplicate line, test what are the file names and if the same, note as error. Those are not really errors, just the duplicates that need to be investigated so that all the commits don't have to be investigated for duplicates, just the suspects.
`test_buggy_lines`	Test that all the buggy lines were the deleted lines in a fix.
`test_buggy_lines_single`	Test specific commit's buggy lines with pre-verified data.
`test_annotated_inserted`	Verify that number of inserted lines is correct.
`test_ranges`	test ranges in the buggy lines

Testing Focus

Unit tests for analyze_focus.py and select_devs.py.

File	`/scripts/tests/test_focus.py`
Usage	`Run the script from tools directory.` `python3 -m unittest tests/test_focus.py`

Data Required	`entropy_groups.json,`
	`data/test/selected_test.json,`
	`data/test/test_years_good.json,`
	`data/test/test_years_bad.json,`
	`data/days_bad.json,`
	`data/days_good.json,`
	`data/test/test_overall_good.json,`
	`data/test/test_overall_bad.json,`
	`data/entropy_ranges.json,`
	`data/entropy_ranges_vis.json`

Testing Authors

Unit tests for prepare_devs.py and combine.py.

File	`/scripts/tests/test_authors.py`
Usage	`Run the script from tools directory.` `python3 tests/test_authors.py`
Data Required	`data/<repository>/authors.json` `data/authors_combined.json`

Testing Fixes

Unit tests for extract_fixes.py.

File	`/scripts/tests/test_fixes.py`
Usage	`Run the script from tools directory.` `python3 tests/test_fixes.py`
Data Required	`data/<repository>/fixes.json` `data/<repository>/test/test_fixes.json`