HPC-completeCycle.sh

This script is in charge of the entire cycle, so to the user a simple command can bootload and start the process

0-ProjectRefactorer.jar

In order to comply with Pinot's requirements, this Java project will refactor all the .java files inside a repository to refactor occurences of elements which are not supported by Pinot

4-PomFileManipulator.jar

To support older versions of Maven, a java project was created to refactor and "clean" old pom files, so that more dependencies could be captured for Pinot.

Copy Maven Dependencies

After the preprocessing stage before, a command is executed to obtain the Maven dependencies of the project in a specific commit

Pinot Software

This software is the one that will produce an analysis for each of the commits in a GitHub repo

Loops for each commit

HPC-blank-error-validChecker.sh

"Debug" script, that checks all the outputs from Pinot, to write information regarding how many files contain valid analysis, or error/blank files. This is helpful to isolate possible conflicts with certain commits.

Outputs

$ProjectName-blanks.list (hashes of commits where the output was a blank file).

$ProjectName-errors.list (hashes of commits where the output was a blank file).

$ProjectName-valid.list (hashes of commits where the output was a blank file).

$ProjectName-finalAnalysis.txt (count of how many blanks, errors and valids in a cycle).

pinotAnalysisProgressChecker.jar

Another "debug" stage, where the java project will create a timeline of the output of pinot according to the three possible states: valid, error or blank. This is just an informative project which aims to create a way for the user to understand how Pinot behaves over time on a given project.

Outputs

$ProjectName-finalAnalysis.csv (prints a representation of the evolution of the result of the pinot analysis), an example:

1,3,valid
2,2,error
3,1,blank

The first number represents the commit counter, the second number represents either blank (1), error(2) or valid(3).

pinotOutputComparator.jar

Very important step in the cycle. Here is where a java project will compare two consecutive commits and compare for differences in the number of outputted patterns. This project outputs a file named "VALID-COUNTER" which will have information on the differences in the number of patterns.

Outputs for each comparison (two consecutive commits)

Pattern changes caused by commit: 5351ccfb462570c6cc9d9ab50ae7fab40369fcbf

From: Facade-22
To: Facade-23

From: Flyweight-14
To: Flyweight-15

From: Mediator-52
To: Mediator-53

HPC-issueTagExtractor.sh

In order to obtain the correspondence between commit and issue from JIRA, for each commit, a git log needs to be printed to find that same information.

Outputs for each file created by pinotOutputComparator, a new file with a copy of the information + the git log

Pattern changes caused by commit: 5351ccfb462570c6cc9d9ab50ae7fab40369fcbf

From: Facade-22
To: Facade-23

From: Flyweight-14
To: Flyweight-15

From: Mediator-52
To: Mediator-53

==========
NEW GIT LOG
==========

This commit refers to file: VALID-14.txt

commit 5351ccfb462570c6cc9d9ab50ae7fab40369fcbf
Author: Niklas Therning <niklas@apache.org>

Added StreamWriteFilter

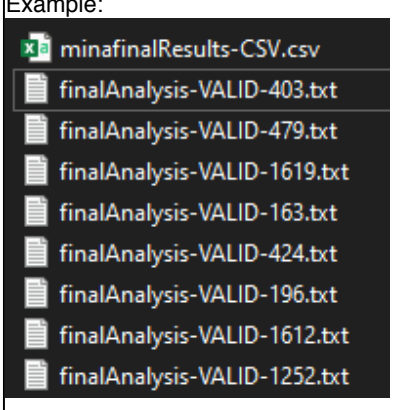
JiraIssueParser.jar

Last step of the cycle. Here is where a java project will read all the files created by HPC-issueTagExtractor, obtain an issueKey from the commits and write the information of that issue + the changes that occured in the patterns in that commit, to a csv file which will hold the final results for the entire project.
Besides this, it will print to this same csv file information on the pattern changes for commits which do not contain issueKeys in the Git log, so that they can be manually inserted afterwards. **(Output 1)**
The way the information from the issues is retrieved is by using HTTP connections to obtain and parse XML information from JIRA's issues repository.
It also creates, for each issue key, a detailed file containing the same information that is present in a row of the csv file, but with the addition of comments (since these would be impossible to write to a csv). **(Output 2)**
At last, it scans all the issues in a project and creates a file for each issue key containing all the comments/descriptions which contain possible information regarding patterns discussion **(Output 3)**.
A "condensed" txt file is created with the same logic as the ones created for all issues, but this one contains all the possible discussions of patterns in the issues that had their issue keys in commits that contained pattern changes. **(Output 4)**

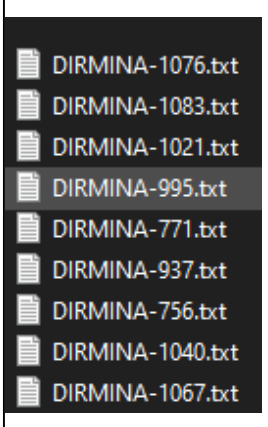
outputs

finalResults-$ProjectName-issueTags - This folder contains the final csv file, named $ProjectNamefinalResults-CSV.csv **(Output 1)**, as well as the files for each valid analysis **(Output 2)**.

Example:



AllIssues-$ProjectName-issueTags - Folder that contains a file for each issue of that project **(Output 3)**.
patternsMentionedInIssues.txt corresponds to **Output 4**.



On issue key DIRMINA-209 the singleton pattern might have been discussed, namely here:
==========
SocketIoProcessor singleton causes performance wall under load
==========

Cycle Finished!