# GitHub Developer Profiler Based on Latent Dirichlet Allocation

By

FADI YILMAZ

fyilmaz@cs.fsu.edu

A PROJECT PRESENTED TO THE GRADUATE SCHOOL OF FLORIDA STATE UNIVERSITY IN PARTIAL FULLFILMENT OF THE REQUIREMENTS FOR THE DEGREE OF MASTER OF COMPUTER SCIENCE

FLORIDA STATE UNIVERSITY

2014

Copyright 2014

By

Fadi Yilmaz

# ACKNOWLEDGEMENTS

I am using this opportunity to express my gratitude to everyone who supported me throughout this Master of Computer Science project, especially to my advisor Dr. Haiduc. I am thankful for her enlightening advices and sincere approaches to me during every step of implementing the project. I am also honored to have Dr. Zhao and Dr. Burmester in my committee.

I would like extend my sincerest appreciation to my advisor Dr. Liu and Dr. Mascagni to guide me during my graduate education. Without their guidelines, I could not even have the opportunity of writing this report and getting one more step closer to my diploma.

Last, I would like offer my thanks to TURKISH REPUBLIC MINISTRY OF NATIONAL EDUCATION and TURKISH CONSULATE GENERAL IN NEW YORK for being my sponsors.

# TABLE OF CONTENT

**TITLE PAGE** ..... **1**

**ACKNOWLEDGEMENT** ..... **3**

**CHAPTERS**

**1 ABSTRACT** ..... **6**

**2 INTRODUCTION** ..... **7**

**3 GITHUB** ..... **8**

3.1 EXTRACTING COMMITS OF EXISTING PROJECT ..... 8

3.2 COMMIT FORMAT ..... 9

**4 LATENT DIRICHLET ALLOCATION** ..... **9**

4.1 MALLET ..... 10

**5 DEVPROFILER** ..... **12**

5.1 INSTALLATION OF MALLET ..... 13

**6 OVERVIEW OF DEVPROFILER** ..... **13**

6.1 DOWNLOADING A GIT PROJECT INTO ECLIPSE WORKSPACE ..... 14

6.2 CREATING BATCH FILE ..... 14

6.3 CREATING SUBDIRECTORIES AND PROFILE FILES ..... 15

6.4 PARSING COMMITS ..... 16

6.5 THRESHOLDS IN WRITING COMMITS TO PROFILE FILES ..... 16

6.6 STEMMING PROCESS ..... 17

6.7 INVOKING MALLET WITH PROFILE FILES ..... 19

6.8 CREATING CHART FROM MALLET OUTPUT FILES ..... 20

**7 CREATING AN ECLIPSE PLUG-IN** ..... **21**

7.1 CREATING MANIFEST.MF ..... 21

7.2 CREATING PLUGIN. XML ..... 22

**8 HOW TO USE DEVPROFILER** ..... **24**

8.1 CALCULATING STATISTICAL SIMILARITIES OF DEVELOPER PROFILE FILES TO EACH TOPIC ..... 24

8.2 SUBDIRECTORIES AND PROFILE FILES ..... 25

8.3 INTERPRETATION OF EXCEL CHART ..... 26

**9 CASE STUDY – IMAGEJ** ..... **27**

**10 TEST RESULTS AND COMMENTS** ..... **28**

**11 REFERENCES** ..... **31**

# 1. ABSTRACT

It is frequently happening event that a developer leaves and a new developer is hired or assignment of a developer is changed, in the software companies that have many employees. After that, some role assignments are happened and newcomers or developers, who are assigned to the project that they have not worked on, might have trouble with catching how the project goes on. In these cases, they need to ask help from the more experienced colleagues, but also they need to know expertise of the colleagues to get a useful answer.

On the other hand, pair programming is an agile software development technique in which two programmers work together and nowadays, it is used very common in software companies. The project manager is responsible for deciding pairs. Consistency of assigned pairs is one of the most important factors that affect efficiency. The project manager needs to know developers expertise deeply to make the best decision for the company.

In this project, I aim to implement such a tool which creates developers' profiles using source codes and commit messages of each developer in project repositories. I am looking at repositories to obtain history of the project which means to have far more information, instead of looking at final version of the project. To illustrate, while final version of the study case, Imagej[1], has 136 thousand lines of source code, its repository has more than 6 millions lines of information.

# 2. INTRODUCTION

Knowing expertise of developers makes some works of a project manager easier, such as deciding pair programming teams, assigning new role to existing developers etc. Besides, a developer needs to ask a question about an issue in which a colleague might have expertise. Since a developer can not know all of expertise of his or her colleagues', the developer should be given expertise of the colleagues or uses a tool to create their profiles to find the correct colleague to ask help.

*DevProfiler* is an eclipse plug-in which creates developers' profiles based on Latent Dirichlet Allocation (LDA) using source code and commit messages of a project in its repository. DevProfiler extracts commits in Git repositories for each developer and gives an excel file contains chart where topics and percentages of the expertise on the topics are shown. The topics are chosen using LDA by processing all information in the repository. Therefore; the topics are continual for the project and number of the topics is optional. However; if the number of the topics is not chosen wisely, the topics will be shallow or meaningless because LDA tries to cover whole information within given number of topic. The number must not be too big to make topics shallow or too small to make topics meaningless.

A topic has a set of words which are relevant each other. Words in the set must be considered as a collection not as an individual. For instance; a topic, Florida State University, can be demonstrated with 'J.J.Love', 'Tallahassee', 'Seminoles', 'Garnet', 'Gold', 'Strozier' and 'Dirac'. When a person reads the words in the set, that person is able to claim that these words are about Florida State University, easily. However; if the word 'Gold' is shown to the person only, he thinks the word is about a metal or economics most probably. The person might think that the word is about Florida State University but least probably. Therefore; LDA gives percentages to each developer on each topic.

This report is structured as follow. First I discuss the problem and how our project helps to solve this problem. Next I discuss basic features of *GitHub* [2], Git repository service, and how the information is pulled out from it. Then, I mention LDA and *mallet* [3], the LDA tool. Finally, I state our project, DevProfiler in detail with the project called *ImageJ* as the case study.

# 3. GITHUB

GitHub is a Git repository web-based hosting service offering additional features. While a classic Git is strictly a command-line tool, GitHub provides a graphical interface, desktop and mobile integration. Allowing non-registered users to browse public repositories is one of the most beloved features of GitHub. Registered users are allowed creating ‘forks’ to an existing project in their own repositories. In addition, GitHub provides a social network graph to display how developers work on their ‘forks’ with dates of them. Therefore; GitHub stores history of a project by containing every commit.

### 3.1. Extracting Commits Of Existing Project

In this project, I use desktop integrated version of GitHub to access its features via windows command prompt. First, git must be added to ‘Environment Variables’ through Windows OS advance system settings. Otherwise; the windows command prompt does not recognize git commands. Second, GitHub must be integrated Eclipse IDE to download source code and create new project with this source code because DevProfiler is an eclipse plug-in, and works on ‘Project Explorer’ window of eclipse. Final, DevProfiler needs to a windows batch file to invoke the process of git [8]. The git command is “git log -p > [filepath]” and this command must be called on where ‘HEAD’ file of the project is because GitHub decides what project is chosen by looking for HEAD file of projects [4]. [filepath] is the path of the text file where extracted commits are stored. Furthermore; ‘git’ command can extract history of single file which is usually a class file in java, if only information of a class is needed instead of whole project.

### 3.2. Commit Format

When a commit is pushed to repository, GitHub assigns a unique hash code to the commit. Thus; commits are separated to each other and accessible through that hash code. In Figure 1, a partial commit output is shown.

```
commit b1c5328661d7f6f8220d01630d424b0bb325ee24
Author: Curtis Rueden <ctrueden@wisc.edu>
Date:   Wed Nov 5 12:17:04 2014 -0600

    ToplevelImageJApp: use ImageJApp.PRIORITY constant

    This is more surefire -- if the priority of ImageJApp itself changes,
    the ToplevelImageJApp will still be prioritized correctly.

diff --git a/src/main/java/net/imagej/app/ToplevelImageJApp.java b/src/main
index 077be62..24f3c5a 100644
--- a/src/main/java/net/imagej/app/ToplevelImageJApp.java
+++ b/src/main/java/net/imagej/app/ToplevelImageJApp.java
@@ -31,10 +31,8 @@

 package net.imagej.app;

-import net.imagej.app.ImageJApp;
 import net.imagej.legacy.LegacyService;
```

Figure 1 - A commit output from repository

The first line of a commit output is “commit [hash code]”. The following line has format of “Author Name: [registered user name] <[registered email address]>”. The next line is for date and it has format of “Date: [time of commit pushing]. The next line is empty line and then commit message takes place. The following lines show modification on source code, if any. Otherwise; the commit is ended. The modification starts with path of old file before modification and new file after modification. The lines start with ‘-’ were removed and the lines start with ‘+’ were added. GitHub gives extra three lines before and after each change to make the changed lines more meaningful.

# 4. LDA

LDA [5] is an example of ‘Topic Model’ which is a type of statistical model for discovering topics that occur in collection of documents in machine learning and natural language processing. The aim of the topic modeling is to explain similarity of unobserved collections with topics by generating trained model of observed collections with the topics. Mallet is a command-

line based LDA tool [4], working on windows command prompt. Mallet has several features such as statistical natural language processing, document classification, clustering, information extraction, and topic modeling. The feature topic modeling is the reason why mallet is used since after extracting commits from a repository, the commits are stored in a text based document for each project. That document is divided into a collection of documents according to authors' name. After that point, topic modeling is become applicable to our data.

### 4.1. Mallet

Mallet is an LDA tool offering topic modeling and working on windows command prompt. Therefore; it must be added to 'Environmental variables' via 'Windows OS advance system settings'. The path must be 'C:\mallet' and all commands must be written by declaring the path 'C:\mallet\bin\mallet.bat'. mallet.bat is the executable batch file of mallet and used with other parameters.

Mallet is a memory-friendly tool. Even though, its output contains every word lies in the input files, the memory requirement is approximately 1/12 of the input files since it uses a compressing method called gz.

Mallet is implemented in two steps because of nature of topic modeling. In the first step, mallet is trained with the collection of documents. In this project, the collection is text files which contain commit parts which are separated in terms of their owners, author names, and their roles. Mallet observes the parts one by one to be trained itself. After the training, an output file which has information about observation is stored for each part. The file is named as [type of part of collection]-tutorial.mallet. For example; after observing the collection of commit texts, the file commitText-tutorial.mallet is created and it can be used in the second step.

In the second step, the file, *-tutorial.mallet, is read and according to parameters similarity of each file for each topic is calculated. Each part is processed with their own collection. Thus; it is prevented that a far bigger part makes a smaller part ignorable or unworthy to be considered. It is obvious that the part contains only class names is far smaller than the part contains all texts. After the second part, mallet gives three output files. These are a state file of which format is '.gz', a topic keys file of which format is '.txt' and a composition file which has the same format with topic keys file. The state file contains every word in the collection and the topic it belongs.

The topic keys file shows the key words for the each top topic with given number and the composition file gives the similarity percentage of each file to each topic. The total similarity is always 1 since the collection comprises the history.

I want to specify some parameters of mallet. These parameters are used in the first step.

- import-dir : this parameter must be followed by path of a directory where the collection of documents is.
- --input : declares the next parameter is the path of input collection and must be followed by the path.
- --output : shows the the next parameter is the path of output file created by mallet contains information about observation of the collection and must be followed by the path.
- --keep-sequence : is used if the second step is going to be performed so mallet writes the output file for further works.
- --remove-stopwords : removes the words which can not define a topic such as ‘a’, ‘the’, ‘more’, ‘since’. Mallet has its own stop words collections in English, French, German, Japanese, and Finnish but I am not using this parameter because the text files in my collection are written in java language and needs extra stop words such as ‘string’, ‘abstract’ and ‘final’. Also I need English stop words collection since commit texts are written in English and may include stop words.

To sum up, the command to start the first step of mallet is in following format:

“*C:\mallet\bin\mallet.bat import-dir --input [file path of input directory] -- output [file path of the output file] --keep-sequence*”

The parameters are used in the second step are as followed:

- train-topics : opens the tutorial.mallet file and runs the topic model operations. This parameter declares the second step is started and followed by path of the tutorial.mallet file.
- --input : declares the next parameter is the path of the tutorial.mallet file and must be followed by the path.

- --num-topics : shows number of the top topics to be considered and must be followed by the number.
- --output-state : declares the next parameter is the path of the state file and must be followed by the path.
- --output-topic-keys : declares the next parameter is the path of topic keys file and must be followed by the path.
- --output-doc-topics : declares the next parameter is the path of tutorial composition file and must be followed by the path.
- --optimize-interval : should be used to get better result. The parameters gives a statistical margin between the topics and not constant for all case. The best number can be found trying only. I do not use this parameter since I think default variable gives a decent result. The parameter must be followed by the number.

# 5. DEVPROFILER

DevProfiler is an eclipse plug-in which creates developer profiles by extracting project histories from repositories. A developer profile contains information about what topics the developer worked on and the similarities of the works of the developer to the other work which in the project. DevProfiler uses git repositories to have project history which covers the evolution steps of the project from the beginning to its current version. These steps provide far more information about the project and developers who make contribution, than last version of the project. To make a comparison, while last version of the project, ImageJ which is the case study, has 136,000 lines of source code, the history of the project contains more than 6,300,000 lines of code and commits [6]. Whole process takes more than forty minutes for ImageJ, which is not a good performance for daily life usage, but the process needs to be performed once for every project since the output files are stored in mallet workspace folder. Then, users can bring out information about any developers in seconds. Therefore; usage of the plug-in in daily life can be accepted as reasonable.

### 5.1. Installation of DevProfiler

Since DevProfiler is an eclipse plug-in, it must be installed to an existing eclipse IDE directly. DevProfiler can be installed on any eclipse version after 3.5 by copying the jar file of the plug-in into 'dropins' folder of the eclipse. When eclipse is restarted, the plug-in is ready to be used. In addition, there must be integration between GitHub and eclipse since DevProfiler uses git repositories to extract project histories and it works on eclipse 'Project Explorer' window. Actually, DevProfiler can be used without the integration. In that case, there are extra works to be done to make projects to be shown in the project explorer window such as creating new package, giving the name of the project to the package created before and being shared eclipse workspace directory and GitHub workspace directory. Moreover; GitHub must be installed in the path "C:\GitHub" because DevProfiler looks for the path of GitHub as if it is integrated with eclipse and if GitHub is not found, DevProfiler looks for the default path which is C:\GitHub.

# 6. OVERVIEW OF DEVPROFILER

DevProfiler controls every step after downloading a git project to disk, till having similarity charts. The only thing users need to do is to start the plug-in. It navigates the project, creates batch file to invoke other tools it uses, creates required directories and subdirectories, makes performance boosters in I/O operations, performs modifications on text files given by the other tools and creates charts to show legible and easily understandable results to end users. The simplified overview of DevProfiler is shown in Figure 2.

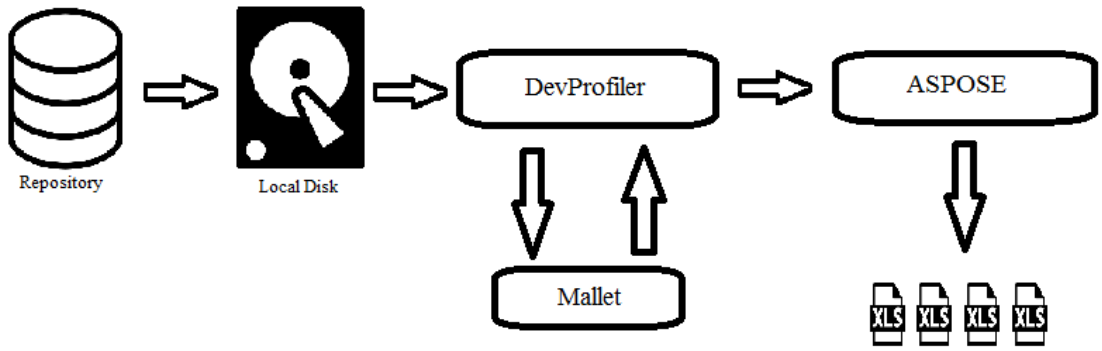



Figure 2 - Overview of DevProfiler

Basically, git project is downloaded to the local disk using eclipse git file import tool. The project history is extracted using GitHub and parsed by DevProfiler itself. The parsed information is sent to mallet to be trained mallet. Mallet performs one more operation to calculate statistical similarities among topics and developers' works to express which topic a developer worked on. The results are taken and sent to *Aspose* [7] to create excel files which contain charts, to make results easily understandable. Finally Aspose creates the excel files and DevProfiler navigates them and shows to users automatically.

### 6.1. Downloading a Git Project into Eclipse Workspace

Git is a very useful and commonly used distributed version control system where open source projects can be stored, modified and cloned by any user. However; to upload new modifications to the original repository, users must be registered and create a branch instead of making changes on master branch. Therefore; users can follow who is responsible for modifications. On the other hand, a branch which is not useful can be demoted by voting of the other contributors, thus, git projects are very reliable projects.

To download a reliable project into eclipse workspace, File ⟶ Import ⟶ Git ⟶ Projects from Git ⟶ Clone URI. Then, a git URI must be entered next to 'URI:' label and the branches to be downloaded must be chosen. The protocol must be chosen as git. If GitHub and eclipse are integrated the project is downloaded into eclipse workspace which is the same with GitHub workspace. Otherwise; before completing download, git asks to path to download the project. In that step, eclipse workspace must be entered manually without changing name of the project and the path of subdirectories.

### 6.2. Creating Batch File

A batch file contains collection of commands to be performed by one by in windows command prompt. DevProfiler creates the batch file to be performed after choosing a git project from project explorer window. Since git commands require to be performed when the command prompt shows path of the project and, the path contains name of the project, each project requires a different batch file which is created as the first step. After getting name of the chosen project, the batch file is created to extract commits. The batch file contains navigation for the path of the

git project which should be the same with eclipse workspace and perform git command, ‘git log’. The batch file is invoked by java.lang.ProcessBuilder class used to create operation system processes. After the commands in the batch file are finished, the ProcessBuilder class gives a text file as output which contains whole history of the chosen project. As reminder, extracting commit of existing project is stated in Chapter 3, Section 1.

### 6.3. Creating Subdirectories and Profile Files

Subdirectories are needed to store separate type of information in the history. During parsing output of the batch file, the information must be clustered and stored correctly. The first created subdirectory is called ‘developers’ where name of the developers is hold in a text file. The text file is called ‘developers.txt’ and name of the developers are written line by line. The text file is so crucial for DevProfiler since DevProfiler creates profile of developers and it must know how many developers contributed to the project and their contributions individually. In addition, after whole operation is completed, users can extract any information about developers and they can choose name of the developers from that file.

Since each commit contains name of author, it is easy to detect all developers name. When a new developer is detected, his or her name is written in the text file and profile files are created. The files are created in four other subdirectories in the developers directory. The names of subdirectories are ‘all’, ‘changedLineTexts’, ‘classNameTexts’ and ‘commitTexts’. These subdirectories are created after the batch file finishes.

The profile files are stored in relevant subdirectories under the developers directory. After parsing a commit, information about the source code must be stored into relevant profile files. For a developer, there are four profile files, one in each subdirectory. Format of names of the profile files is [name of developer]-[relevant subdirectory name].txt. Therefore, DevProfiler creates number of developers times four profile files at this point.

The profile files require having another subdirectory for stemming process. Although stemming can be implemented on the same file since it has a feature of ‘keep-original-file’, size of the project files does not allow to working on the same file. When keep-original-file is selected as false, after stemming a file, the original file is deleted and a new file is created. This process requires one I/O operation but the profile files are too big to be handled in one I/O operation. Therefore; keep-original-file must be selected as true and the stemmed information must be

written into a new file. This new file is stored as the subdirectory of stemming process which called ‘Eliminated’. The format of stemmed files is the same with the format of original files but they are stored in another subdirectory. Stemming process is mentioned following chapters.

### 6.4. Parsing Commits

Parsing a commit depends of its format. Since GitHub has a constant format for all commits, same parsing steps can be applied to all commits. First, each commit is separated by finding their commit hash line. Then, author name is parsed and the commit is sent to the author log files. Second, the rest of commit is divided into four files for each developer. The first file is called commit text file and all commit texts are stored in a single file for each developer. The second file is called class name file and only name of classes where the modification takes places is stored. A commit can contain more than one class and more than one modification take place in the same class. In that case, repeated class names are ignored to provide consistency of LDA processing. The third file is named as changed line file and the only lines marked with ‘-’ and ‘+’ are sent there. The last file is called all text. After removing commit hash line, author name line and date line, the commit sent there without any modification. Basically, all text files contain all information in the other three files and unmarked source codes additionally.

### 6.5. Thresholds in Writing Commits to Profile Files

The collection of commits is read line by line to pull each commit out. The commit is stored in a java.util.ArrayList and each 400 commits are written to the disk. The number, 400, is a threshold of number of commits to be written into disk. This threshold prevents memory restrictions and provides better I/O performance. Either a commit can contain only five lines of source code or the commit can contain more than 100,000 lines of source codes since git thinks that adding a file to a project is same with adding the lines in the file to the project. Even if a name of file is changed in a project, the output of that commit includes all lines in the file with its previous name as removed lines and the same lines in the file with its new name as added lines. Thus; a small change in file naming causes a huge output as two times size of the file of which name is changed. Therefore; there is another threshold, which is 3,000, as maximum line of source code for a commit. I experienced that that threshold provides more accurate result and better performance since rest of commit is ignored.

### 6.6. Stemming Process

Stop words elimination provides more accurate result for our topic modeling since stop words do not refer a useful keyword and they are removed from collection. The elimination has seven steps and the order of the step must be followed.

- Blank Elimination: It is obvious that a text has blanks and the blanks must be removed to pull words out from the text. In java, there are some blank characters except space character ' '. These characters are represented as '\t', '\r' and '\n'. '\t' refers eight-character-block. '\r' refers carriage return character and is used to send line pointer to the left-hand side. '\n' refers line feed character and is used to move line pointer to the next line. To illustrate a text "Tallahassee is the best place to live (\t) in.(\n)" becomes "Tallahassee is the best place to live in.". In the example double spaces are reduced to the single spaces and '\t', '\r' and '\n' are removed if any. Single spaces must be preserved because of a method of java.lang.String class, .split(param_1). The method takes param_1 as a splitter token and return an array contains every word in the string as an element individually. If the text contains double spaces, this method splits double spaces and assumes that there is a string with no characters. Therefore; double spaces or triple spaces or more spaces must be reduced to single spaces.

- Number Elimination: Even if some numbers have so identifying importance such as dates, they are not accepted as useful key words. Therefore; numbers are removed from the collection.

- Non Literals Elimination: Characters other than letters and numbers such as '.', ';', '$', '[' are frequently used in java language. These non literals must be removed to pull the meaningful words out. For example; the line "System.out.println("That is great!");" is transformed in "System out println That is great".

- Splitting Identifiers: Identifiers are used to give name to objects, methods and variables in java language. If the word contains a number or a capital letter among small letters or the character '_', the word is an identifier most probably. This is a common programming rule [9]. For example; an identifier with two words mostly written as format of the first word with all small letters and the second word with only capital initial letter without space or the character '_' is between them.

- Keywords Elimination: In this step, there are words that are able to be a key word in the collection only. Java language requires using many keywords such 'public', 'extends', 'import'. These words are used with a meaning different from their original meaning. Even if their original meaning can be a good key word, eliminating them gives more accurate results. List of the keywords is stored in the disk and the list can be found in Oracle Documentations [1]. In addition some words in English are not good for to be key words. Therefore; additional elimination must be performed. Mallet has its own English keywords list and also this list can be found in many web services. Some of the words in the keywords list are 'despite', 'less', 'ourselves'.

- Stemming: Stemming is an operation used for getting root of words. It does not need to give morphological root but it must map the related word to the same root even if the root is not a valid word. For example; the words 'cats', 'catlike', 'catty' are stemmed to 'cat'. As another example with not a valid root, the words 'argue', 'argues', 'arguing' are stemmed to 'argu'. Although stemming may give some meaningless words, the mapping related words to the same root makes it very useful since the topic modeling assumes 'argue' and 'arguing are different from each other and their statistical percentages divided into number of different related words. On the other hand, output of topic modeling is going to be read by people and a person can understand that the word 'argu' is the root of 'argue' and 'arguing'. There is a possibility of two different words are mapped to same root but this risk must be taken at it is obviously least probable then having related words with the same root.

- Short String Elimination: After having roots of the words, the roots with length of less than three are removed to have more accurate results.

### 6.7. Invoking Mallet with Profile Files

Mallet is an LDA tool providing topic modeling and working on windows command prompt. Therefore; DevProfiler invokes mallet using the java.lang.ProcessBuilder class same with invoking the batch file. Different mallet process is invoked for each type of profile files since they are using different subdirectories. As mentioned before, input and output paths in the format of mallet command are changed for each type.

Since mallet needs to be completed in two steps, two processes are invoked by DevProfiler. The first process is for training mallet and the second step is for calculating similarities. The second step must wait for the output of the first step. Therefore; pipelining is not applicable in this case. However; each mallet process can be performed individually. Thus; DevProfiler invokes more than one process at the same time. Mallet requires more than forty minutes to be completed totally although the processes are invoked at the same time.

#### 6.7.1. Format of Output Files of Mallet

Mallet gives three files as output. The first file is state file which contains every word in the collection and the topic it belongs. The second file is topic keys file which shows the key words for the each top topic with given number and the last files is composition file which gives similarity percentages of each file to each topic. Their format is permanent, so using the same parser for every project is viable approach.

The first step creates tutorial.mallet file which contains information about words, topic, developers, basically everything in the history. It is not executable or legible and only has meaning for mallet itself.

The state file is the biggest file that mallet gives as output. It is the compressed text file and the decompressed text file contains every word in the collection. The most remarkable feature of mallet is actually to create that file since the other outputs are created using the state file. To express the format of the file easier, a line from it copied to Figure 3. As it is shown, the first character, which is 2, is a number of the file in the collection and the path of the file, which is C:\mallet\workspace\developers\changedLineTexts\Eliminated\Barry DeZonia-changedLines.txt, follows it. The next numbers, which are 216299 and 93, are the order of the word ‘chang’ and its

place where it first appears. The last number, which is 9, shows the topic number that the word ‘chang’ below. Here ‘chang’ is the stemmed version of related words of ‘change’.

```
2 C:\mallet\workspace\developers\changedLineTexts\Eliminated\Barry DeZonia-changedLines.txt 216299 93 chang 9
```

Figure 3 - A line from state file

The next output is tutorial_composition file. This file represents profile files of every developer with one line. The line shows the similarity percentages of the profile files to topics. The similarity percentages are given in descending order and DevProfiler uses the top five topics to make the chart eye candy. If DevProfiler considers all topics and assume that there are twenty five topics and twenty contributors, the chart has five hundreds variables that makes read the file hard and complicated.

Topics are assigned a number and the tutorial_keys file shows that assignments. Figure 3 shows a line from tutorial_keys file. The first number, is which is 10, is the assigned number of the topic. The next number, which is 3.33333, is the mallet coefficient. This coefficient is given with the parameter of --optimize-interval. If this parameter is not given, mallet calculates it as dividing 50 to the number of topics that is 15 in Figure 4. The following words are key words. The key words are written in random order. Even in different iteration assigned number of topics and order of words may be different. However; the set of key words of a topic is protected.

```
10  3.33333 file except schindelin updat johann site legaci servic plugin loader upload version limit script stream imag mode action pom
```

Figure 4 - A line from tutorial_keys file

### 6.8. Creating Chart from Mallet Output Files

Aspose is a java api (application programming interface) provides I/O operation and perform excel commands on excel files. It is easily installed to an eclipse plug-in since only adding path of .jar file of the library to the plug-in is sufficient. Aspose provides a worksheet object which behaves like an excel file and performs I/O operation after the worksheet object is completely ready. If the same worksheet object is called one more time, it assumes that it is working on an empty file.

Aspose provides access to each cell in excel files with a method called .getCells(). The method returns image of the current table of the worksheet object. Then it access a cell using the method called .get(param_1) where param_1 is coordinate of the cell. The method .putValue(param_1) is used to write a value into the cell. If the value is an integer, Aspose converts the cell type to number, and if the value is a string, it converts the cell type to text automatically. To create a chart with existing values on the worksheet objects, the coordinates of a rectangle where the values lies in, type of chart and the coordinates of the chart must be given to Aspose then it creates the chart. However; the worksheet object must be stored to disk after the chart is created, otherwise the object assumes that an empty file is called and the data in the previous object are destroyed.

# 7. CREATING AN ECLIPSE PLUG-IN

DevProfiler is an eclipse plug-in that means it is written to work on eclipse platform only and contribute to it. A plug-in uses the same syntax with a java project and the all java language rules are valid for it. However; a plug-in has some extra files than an ordinary java project. These files are manifest.mf and plugin.xml. While manifest.mf contains some configurations, states the dependencies of the plug-in and the runtime requirements; plugin.xml contains information about how the plug-in contributes to the platform.

## 7.1. Creating Manifest.mf

One of the most useful features of eclipse is to have plug-in development wizard. Users need to choose their configuration among given settings instead of writing them into manifest.mf. The wizard allows setting ID, version and name of the plug-in at first. Then, users are required setting the activator of the plug-in and if the plug-in is a singleton or not by clicking relevant checkboxes. For DevProfiler, ID and name are ParsingUserInterface and version is 1.0.0.qualifier. DevProfiler need not to have an activator since it is activated when eclipse starts up. Being a singleton means that bundle only have one version could exist in the OSGi runtime and for DevProfiler it is true.

DevProfiler requires other plug-ins for the operation. These plug-ins are org.eclipse.jdt.core, org.eclipse.core.resources, org.eclipse.core.runtime, org.eclipse.core.expressions,

org.eclipse.core.commands and org.eclipse.ui. There are also three constitutor libraries specified in runtime tab. After setting these configurations, manifest.mf is completed. Manifest.mf is given in the related codes section.

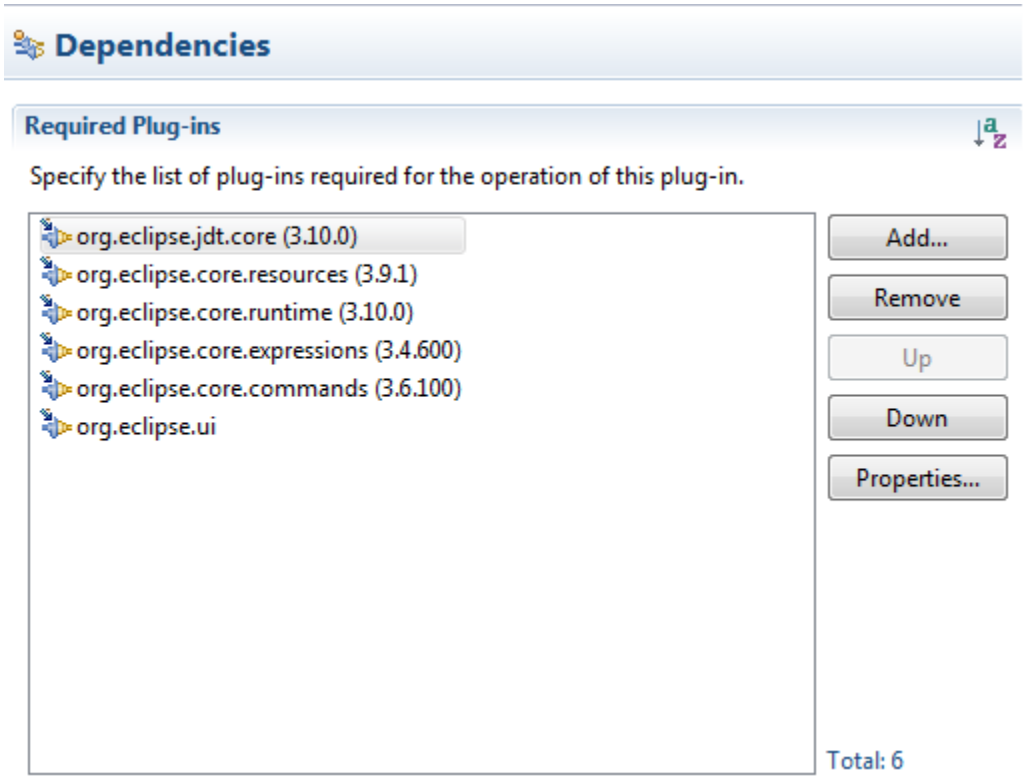



Figure 5 - Dependencies of DevProfiler

### 7.2. Creating Plugin.xml

Plugin.xml indicates how the plug-in contributes to eclipse platform. DevProfiler takes place in the right click menu of a project in the project explorer window. Eclipse has its own command structure to specify contribution of a plug-in and DevProfiler uses org.eclipse.ui plug-in to take place at the right click menu. At the first step; a command is created with unique ID and a default handler option. Basically, a command contains name and path of the 'runnable' class of the plug-in. A class being runnable means that the operation starts with that class. At the second steps; menu contribution of the plug-in is set using org.eclipse.ui.menus. Since DevProfiler takes places at the project explorer window, its location is popup:org.eclipse.jdt.ui.ProjectExplorer. ID of the command created at the first step is entered as command ID of the menu contribution and the

label of the tab is set. For DevProfiler commandID is ParsingUserInterface, default handler is ParsingUserInterface.runApplication and the label is DevProfiler. ParsingUserInterface.runApplication is implemented in runApplication.java. In Figure 5, a screenshot of creating plugin.xml window is given.

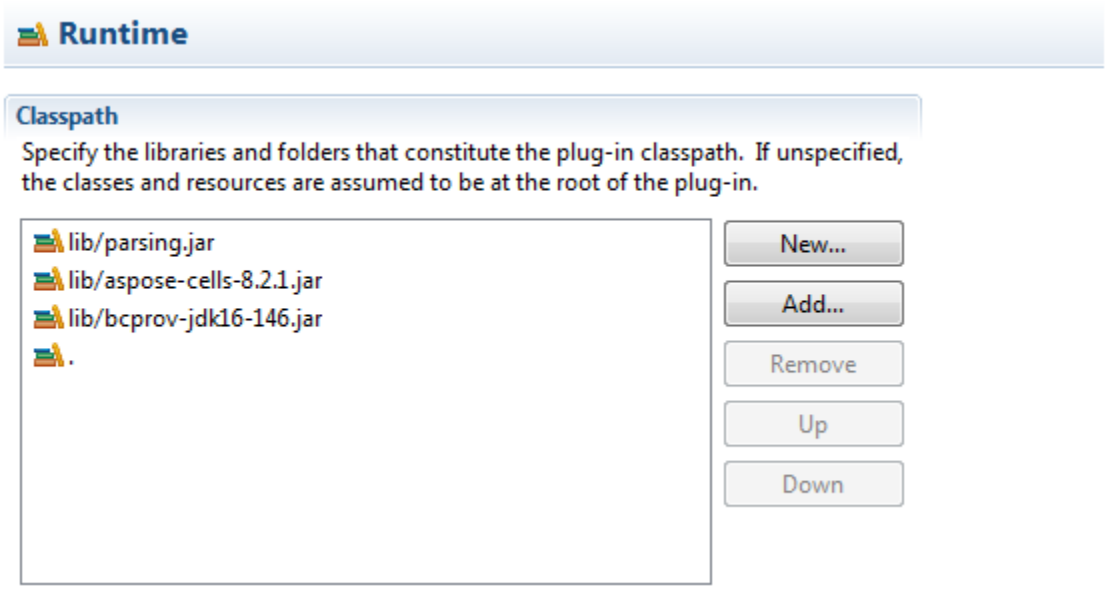



Figure 6 - Runtime requirements for DevProfiler

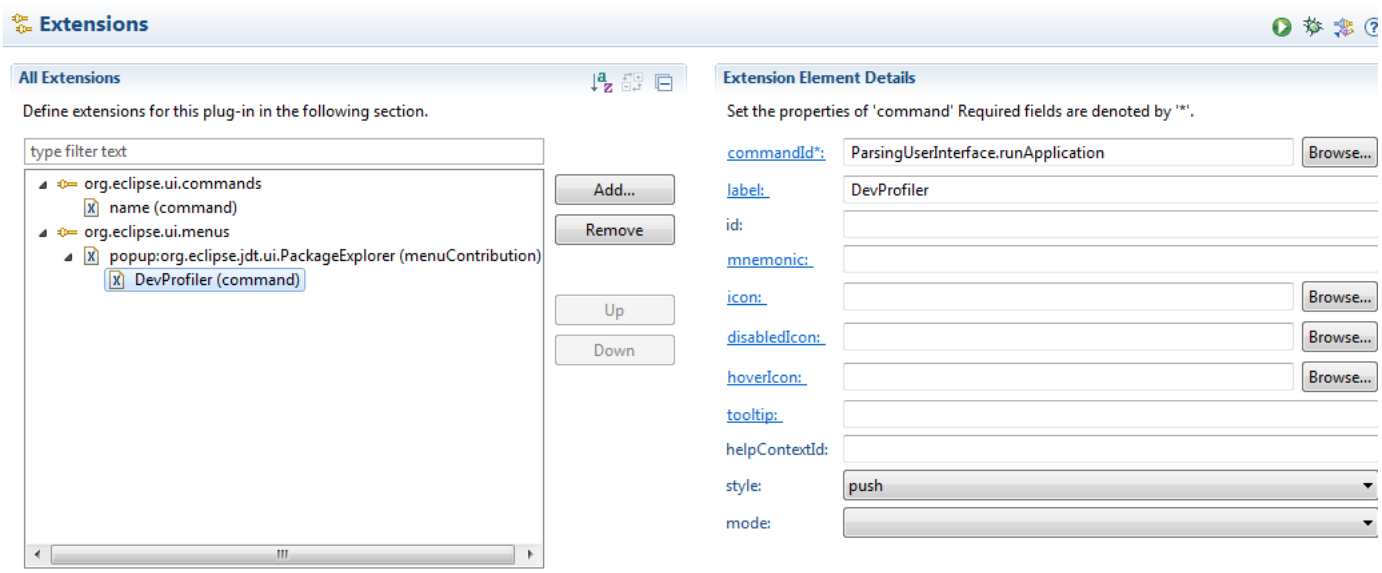



Figure 7 - Creating plugin.xml window

# 8. HOW TO USE DEVPROFILER

DevProfiler performs in two steps. At the first step, statistical similarities of developer profile files to each topic are calculated and stored into the disk. The first step takes more than forty minutes for the test case. At the second step, desired similarity or similarities are written in excel chart and shown to users. Although the first step takes more than forty minutes, it is one time job and after that the second step is performed anytime since the similarities are stored into the disk. The second step takes seconds.

## 8.1. Calculating Statistical Similarities of Developer Profile Files to Each Topic

DevProfiler is an eclipse plug-in which offers a graphic user interface. After the installation, DevProfiler can be used by right clicking on a git project in project explorer window. The label of DevProfiler appears in the menu and by clicking its name the plug-in is activated. DevProfiler can be used for only a git project since it requires a git repository where history of the project is stored. After activating DevProfiler, its configuration window pops up. In Figure 9, the place of DevProfiler is shown.

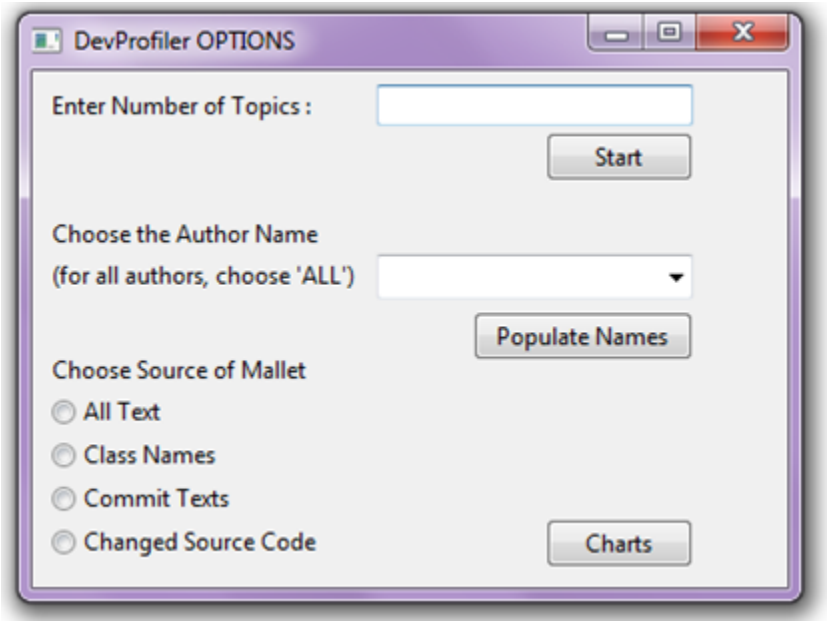



Figure 8 - DevProfiler interface

At top of the configuration window, there is a text area to enter number of topics. The number of topics can be any number, but according to tests, 35 as the number of topics gives decent results for the case study. By clicking start button, the operation is started. In Figure 8, the interface is shown. During the calculation, the plug-in is frozen to prevent users to make any misclicks.

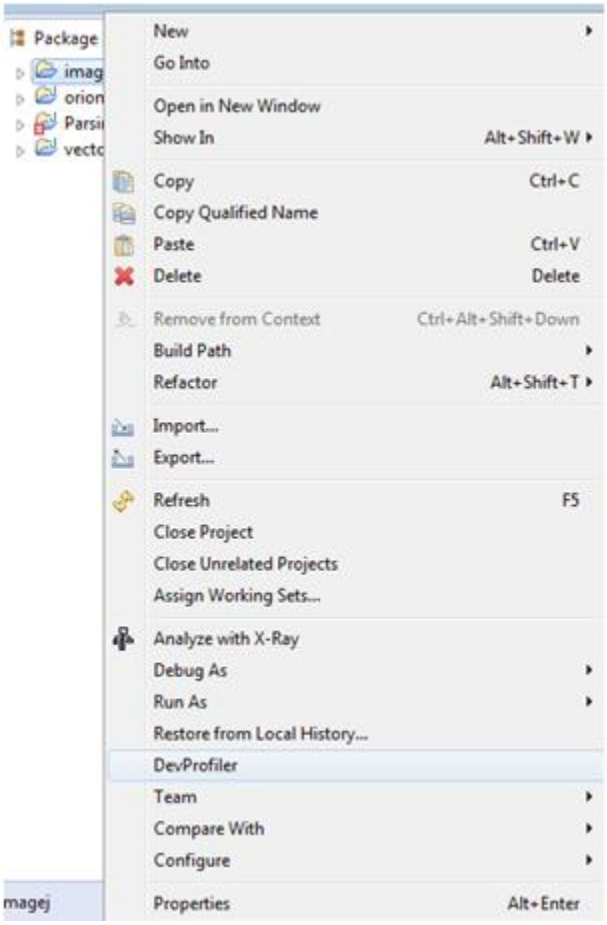
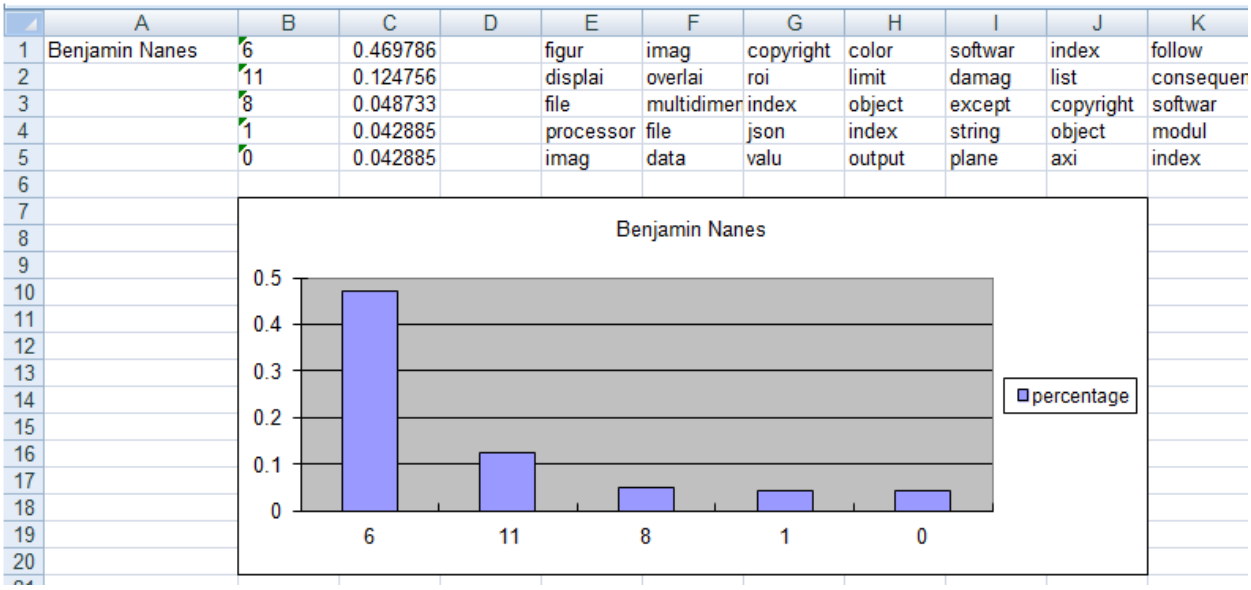



Figure 9 - Right click menu of a project in project explorer window

### 8.2. Creating Excel Chart

The statistical similarities given by the mallet are not easily understandable. Therefore; DevProfiler creates an excel chart for each query. After the excel chart is created, DevProfiler navigates it and then open it. To make a query, the combobox in the interface is used. If DevProfiler is used first time for a project, the combobox filled automatically, after the calculation of the similarities. If DevProfiler has already calculated the similarities, the

combobox is filled by clicking the button named ‘Populate Names’. After the filling the combobox, users can choose a developer name to see what topic the developer worked on. In addition, users can choose ‘ALL’ item to see similarities of all developers in the same time. It was mentioned that mallet has four different types of source and there are radio buttons at left bottom side of the interface to select what type of similarities is desired. After selecting type, users need to click the button named ‘Chart’ at the right bottom of the interface to start the operation of creating charts.




| | A | B | C | D | E | F | G | H | I | J | K |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | Benjamin Nanes | 6 | 0.469786 | | figur | imag | copyright | color | softwar | index | follow |
| 2 | | 11 | 0.124756 | | displai | overlai | roi | limit | damag | list | consequer |
| 3 | | 8 | 0.048733 | | file | multidimer | index | object | except | copyright | softwar |
| 4 | | 1 | 0.042885 | | processor | file | json | index | string | object | modul |
| 5 | | 0 | 0.042885 | | imag | data | valu | output | plane | axi | index |

Figure 10 - An example for excel charts

### 8.3. Interpretation of Excel Charts

After creating an excel chart, the file is opened by DevProfiler automatically. In Figure 10, an example for excel charts is shown. The cell A1 stores name of the developer. The numbers in the ‘Column B’ are topic numbers in the tutorial_keys file. Each topic is assigned a number and represented with that number in the chart. The words above the chart are the key words set of the topics. To clarify, the words ‘figur’, ‘imag’, ‘copyright’, ‘color’ and ‘software’ are the elements of the key words set of the topic 6. Applications which use topic modeling can not say what the topic is with a word but the key words set is sufficient to make an accurate guess. In the example; topic 6 must be about coloring an image file. Actually, when the whole set is read, a better guess can be made. Therefore; I am giving the whole set below:

"*figur imag copyright color softwar index follow line provid condit abstract event binari view name diff draw redistribut tool*" are all elements in the key words set of topic 6 and now, topic 6 is about a tool provides modifications on an image or figure. The excel file contains top five topic to simplify the chart. Otherwise; there would be 35 columns in the chart and 35 rows of key words set. The numbers in 'Column C' are percentages of the statistical similarities. As the example shows, works of developer Benjamin Nanes are related to the topic is in 46.9%. The next most related topic is topic 11 and the percentage is 12.4% and so on.

# 9. CASE STUDY – IMAGEJ

ImageJ is an open source java image processing program designed for scientific multidimensional images inspired by 'NIH image' for the Machintosh. The program can be used as an applet or be downloaded as application. ImageJ can display, edit, analyze, save, process and print up to 32-bit images. It is multithreaded so the time consuming operations such as reading an image file can be performed used pipelining. It is able to make so many calculations such as area of chosen part of images, pixel value statistics, measuring distance and angles, creating density hologram and line profile plots. In addition, it provides basic image processing operation such as edge detection, sharpening and median filtering.

ImageJ is highly extensible with thousand of plug-ins and macros for performing so many different varieties of tasks. This extensibility is provided by many contributors' effort. While the last version of the program has 136,000 lines of source code, the history of it has more than 6 millions lines of code. There are twenty contributors in the project currently. Number of total commits is more than 9,500 and for every 400 commits, DevProfiler performs I/O operations to store them into disk. Although the project itself can be accepted as a medium size project, its history is highly big. Thus; creating topic model of the project is the most time consuming operation, which takes more than forty minutes, in DevProfiler.

# 10. TEST RESULTS AND COMMENTS

Success of DevProfiler depends on giving suitable number of topics as parameter. Since DevProfiler examines whole history, the number of topics must be big enough covering the actual topics in the history and small enough preventing having irrelevant or repeated topics. After applying DevProfiler to ImageJ, I believe that '30' gives decent results for ImageJ as number of topics.

Since DevProfiler creates profile of developers, its result strictly depends on contributions of a developer to project. The test case, ImageJ, is an open source java program and there is no strict rule on contributing to it and this no-rule situation affects performance of DevProfiler because DevProfiler works on some assumptions such as a developer can have one account or every developer should make enough contribution to have a profile. However; some developers have more than one account and some commits only ten lines of source codes in ImageJ project. In these cases, DevProfiler assumes that each account belongs to one developer and creates separated profiles for them and this multi accounting distorts coherence of work that developer did. Furthermore; a developer having a real short contribution history, prevents calculating similarities of the contribution to topics. In that case, the developer contribution has almost same similarities to each topic.

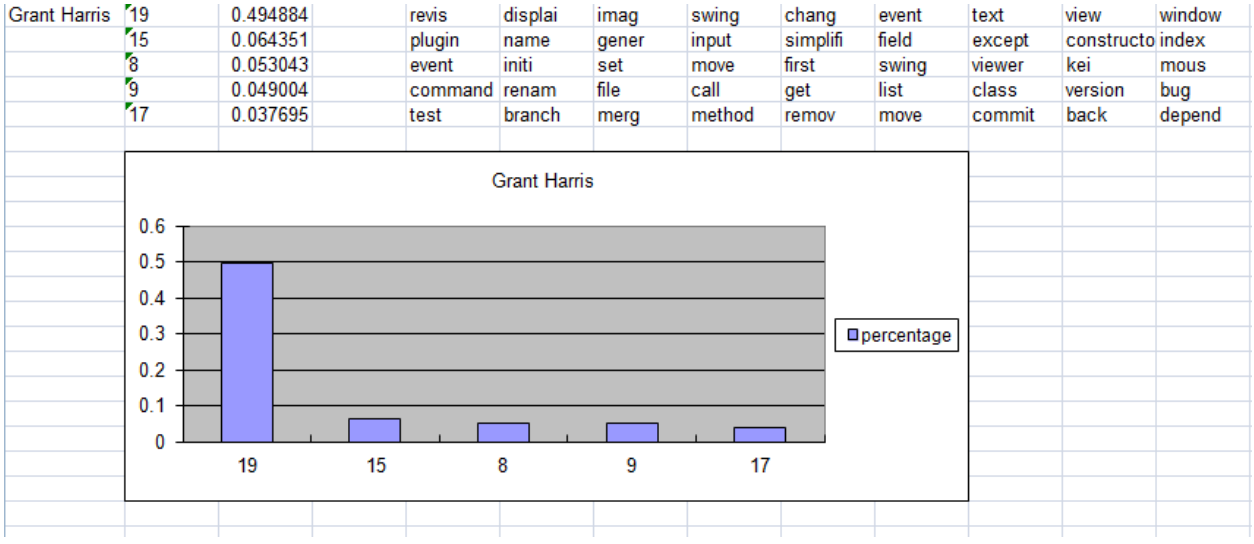



Figure 11 – A Result from Commit Text

In Figure 11, the result of topic modeling for only commit texts is shown for the developer, Grant Harris. The topic number 19, with the key word set, ‘revis’, ‘displai’, ‘imag’, ‘swing’, ‘chang’, ‘event’, ‘text’, ‘view’ and ‘window’ has more than 49% similarity with the contribution of the developer. 49% is quite significant percentage since the total similarity is 100% and it is divided into 30 topics. Even a person, who does not know anything about ImageJ, can claim that Grant Harris has worked on the interface of a program which is responsible for displaying images.

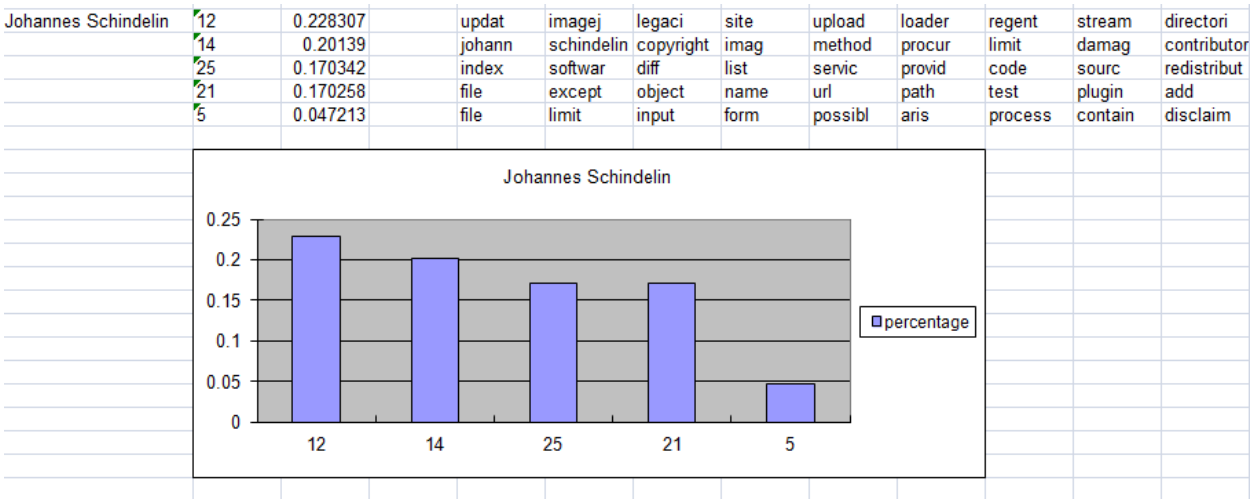



Figure 12 – A Result from All Text

In Figure 12, a result of topic modeling for all text is shown for the developer, Johannes Schindelin. The topics number 12, 14, 25 and 21 have considerable similarities to the contribution of the developer. Having more than one considerable similarity means the developer has worked on more than one topic. That name of the developer is chosen as key words might be seemed weird but the reason of that is the author signed every commit he made with his name. Therefore, the name is written at least number of commit that the developer made.

In Figure 13, a result of topic modeling for only class names is shown for the developer, Melissa Linkert. The figure does not have any significant result because of some factors. One of the factors is that class names are the smallest part in the history since it contains only the class names where the modifications are implemented. As another factor, the developer is responsible for the xml files, and the number of xml files is not big enough having a profile.

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Melissa Linkert | 25 | 0.051282 | | displai | swing | view | hot | panel | drop | navig | statu | edit |
| | 17 | 0.051282 | | pom | imag | releas | consol | save | bright | plugin | posit | interact |
| | 29 | 0.032051 | | legaci | imagej | object | editor | java | helper | engin | hot | adapt |
| | 28 | 0.032051 | | util | option | draw | test | lut | zoom | widget | imagej | shadow |
| | 27 | 0.032051 | | command | text | tabl | panel | creator | mode | bezier | dummi | client |

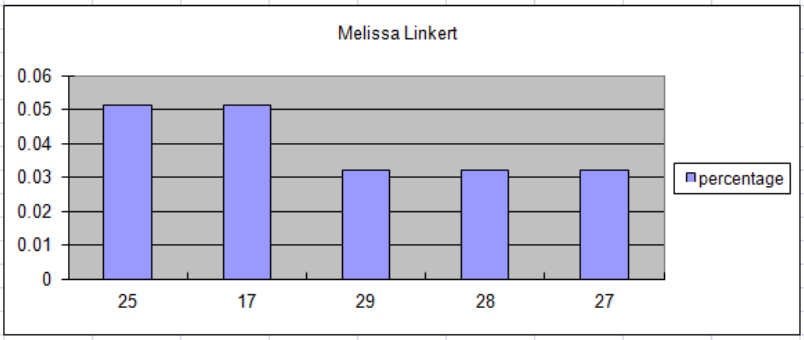



Figure 13 - A Result from Class Name

In Figure 14, a result of topic modeling for only changed lines is shown for the developer, Samuel Inverso. The figure shows that the developer has worked on topic 22 obviously as it has 37.5% similarity. The other topics have smaller similarity but not 0% similarity because the words used in coding or writing a commit, are English words and they must share the same words most probably.

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Samuel Inverso | 22 | 0.375209 | window | menu | item | procur | plugin | damag | right | button | list |
| | 29 | 0.118928 | plugin | load | except | chang | call | size | compon | version | content |
| | 24 | 0.053601 | valu | output | test | dimens | info | type | data | plane | current |
| | 11 | 0.048576 | imag | index | event | color | name | menu | zoom | file | licens |
| | 4 | 0.033501 | index | mnemon | todo | weight | test | diff | persist | plane | kei |

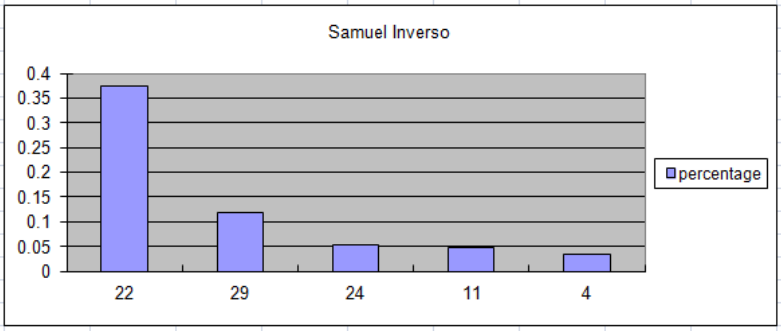



Figure 14 – A Result from Changed Line

# 11. REFERENCES


[1] http://imagej.net/Welcome

[2] https://github.com/

[3] http://mallet.cs.umass.edu/index.php

[4] http://www.vogella.com/tutorials/Git/article.html

[5] http://machinelearning.wustl.edu/mlpapers/paper_files/BleiNJ03.pdf

[6] https://www.openhub.net/p?ref=homepage&q=imagej

[7] http://www.aspose.com/docs/display/cellsjava/Aspose.Cells+for+Java+API+Reference

[8] http://www.microsoft.com/resources/documentation/windows/xp/all/proddocs/en-us/batch.mspx?mfr=true

[9] http://www.oracle.com/technetwork/java/codeconvtoc-136057.html

# 12. RELATED CODES

parsing.java

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package parsing;
import parsing.developers.*;
import java.io.*;
import java.util.*;
import com.aspose.cells.*;
import parsing.elimination.*;
/*
 *
 * @author fadi
 */
public class Parsing {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args)
    {
        run(args[0],args[1], args[2]);
    }
    public static void run(String numberOfTopic,String projectName, String authorName)
    {

    }
    public static void createFiles()
    {
```

```
        File folder = new File("C:\\mallet\\workspace\\commitFiles\\");
        File[] fileList = folder.listFiles();

        ArrayList<Commit> commitList = new ArrayList<Commit>();
        for(File file : fileList)
        {
               if(file.getPath().contains(".txt"))
            {
               ArrayList<Commit> temp =Commit.execute(file.getPath());
                for(Commit c: temp)
                {
                      commitList.add(c);
                }
            }
        }

        for(Commit commit : commitList)
        {
            Developer.createDeveloper(commit);
        }

    }
    public static void malletClassNames(String numberOfTopic)
    {
       try
       {
               Process import_dir = Runtime.getRuntime().exec("C:\\mallet\\bin\\mallet.bat
import-dir --input C:\\mallet\\workspace\\developers\\classNameTexts\\Eliminated"+
                       " --output C:\\mallet\\workspace\\Mallet\\className\\className-
tutorial.mallet --keep-sequence");
               synchronized (import_dir) {
```

```
                    import_dir.waitFor();

                    import_dir.destroy();

              }

               Process trainMallet = Runtime.getRuntime().exec("C:\\mallet\\bin\\mallet.bat
train-topics --input C:\\mallet\\workspace\\Mallet\\className\\className-tutorial.mallet"+

                     " --num-topics "+numberOfTopic+

                     " --output-state C:\\mallet\\workspace\\Mallet\\className\\className-
topic-state.gz "+

               "--output-topic-keys C:\\mallet\\workspace\\Mallet\\className\\className-
tutorial_keys.txt --output-doc-topics "+

               " C:\\mallet\\workspace\\Mallet\\className\\className-
tutorial_composition.txt");

               synchronized(trainMallet)

               {

                 trainMallet.wait(300000);

                 trainMallet.destroy();

               }




       }

           catch(InterruptedException ioe)

           {


           }

              catch(IOException ie)

              {


              }

    }

    public static void malletCommit(String numberOfTopic)

    {

       try

        {
```

```
            Process import_dir = Runtime.getRuntime().exec("C:\\mallet\\bin\\mallet.bat
import-dir --input C:\\mallet\\workspace\\developers\\commitTexts\\Eliminated"+

                  " --output C:\\mallet\\workspace\\Mallet\\commit\\commit-tutorial.mallet -
-keep-sequence");

            synchronized (import_dir)

            {



                   import_dir.waitFor();

                   import_dir.destroy();

            }

          Process trainMallet = Runtime.getRuntime().exec("C:\\mallet\\bin\\mallet.bat
train-topics --input C:\\mallet\\workspace\\Mallet\\commit\\commit-tutorial.mallet"+

                 " --num-topics "+numberOfTopic+

                 " --output-state C:\\mallet\\workspace\\Mallet\\commit\\commit-topic-
state.gz "+

          "--output-topic-keys C:\\mallet\\workspace\\Mallet\\commit\\commit-
tutorial_keys.txt --output-doc-topics "+

          " C:\\mallet\\workspace\\Mallet\\commit\\commit-tutorial_composition.txt");

          synchronized(trainMallet){



               trainMallet.wait(300000);

               trainMallet.destroy();

         }

      }



      catch(IOException ioe)

      {



      }

      catch(Exception e)

      {



      }
```

```
    }

    public static void malletChangedLines(String numberOfTopic)

    {

       try

        {



              Process import_dir = Runtime.getRuntime().exec("C:\\mallet\\bin\\mallet.bat
import-dir --input C:\\mallet\\workspace\\developers\\changedLineTexts\\Eliminated"+

                    " --output C:\\mallet\\workspace\\Mallet\\changedLine\\changedLine-
tutorial.mallet --keep-sequence > C:\\bakbak.txt");

              synchronized (import_dir)

              {



                     import_dir.waitFor();

                     import_dir.destroy();

              }

           Process trainMallet = Runtime.getRuntime().exec("C:\\mallet\\bin\\mallet.bat train-
topics --input C:\\mallet\\workspace\\Mallet\\changedLine\\changedLine-tutorial.mallet"+

                  " --num-topics "+numberOfTopic+

                  " --output-state C:\\mallet\\workspace\\Mallet\\changedLine\\changedLine-
topic-state.gz "+

           "--output-topic-keys C:\\mallet\\workspace\\Mallet\\changedLine\\changedLine-
tutorial_keys.txt --output-doc-topics "+

           " C:\\mallet\\workspace\\Mallet\\changedLine\\changedLine-tutorial_composition.txt
>> C:\\bakbak.txt");

           synchronized(trainMallet){



                 trainMallet.wait(300000);

                 trainMallet.destroy();

           }

        }



        catch(IOException ioe)

        {
```

```
        }
        catch(Exception e)
        {

        }
    }
    public static void malletAll(String numberOfTopic)
    {
       try
        {
              Process import_dir = Runtime.getRuntime().exec("C:\\mallet\\bin\\mallet.bat
import-dir --input C:\\mallet\\workspace\\developers\\all\\Eliminated"+
                    " --output C:\\mallet\\workspace\\Mallet\\all\\all-tutorial.mallet --keep-
sequence");
              synchronized (import_dir)
              {

                    import_dir.waitFor();
                    import_dir.destroy();
              }
           Process trainMallet = Runtime.getRuntime().exec("C:\\mallet\\bin\\mallet.bat train-
topics --input C:\\mallet\\workspace\\Mallet\\all\\all-tutorial.mallet"+
                  " --num-topics "+numberOfTopic+
                  " --output-state C:\\mallet\\workspace\\Mallet\\all\\all-topic-state.gz "+
           "--output-topic-keys C:\\mallet\\workspace\\Mallet\\all\\all-tutorial_keys.txt --
output-doc-topics "+
           " C:\\mallet\\workspace\\Mallet\\all\\all-tutorial_composition.txt");
           synchronized(trainMallet){

                trainMallet.wait(300000);
                trainMallet.destroy();
           }
```

```
    }

    catch(IOException ioe)
    {

    }
    catch(Exception e)
    {

    }
}
public static void malletElimination()
{
   File folder = new File("C:\\mallet\\workspace\\developers\\changedLineTexts");
    File[] changedLineFileList = folder.listFiles();

    folder = new File("C:\\mallet\\workspace\\developers\\classNameTexts");
    File[] classNameFileList = folder.listFiles();

    folder = new File("C:\\mallet\\workspace\\developers\\commitTexts");
    File[] commitFileList = folder.listFiles();

    folder = new File("C:\\mallet\\workspace\\developers\\all");
    File[] allFileList = folder.listFiles();

    elimination(changedLineFileList);
    elimination(classNameFileList);
    elimination(commitFileList);
    elimination(allFileList);
}
public static void createExcelChart(String authorName, boolean [] selection)
{
```

```
    if(selection[3])
    {
            File folder = new File("C:\\mallet\\workspace\\mallet\\changedLine");
             File[] fileList = folder.listFiles();
            creatingExcelChart(fileList, authorName);
    }
    if(selection[0])
    {
            File folder2 = new File("C:\\mallet\\workspace\\mallet\\all");
             File[] fileList2 = folder2.listFiles();
             creatingExcelChart(fileList2, authorName);
        }
     if(selection[2])
    {
            File folder3 = new File("C:\\mallet\\workspace\\mallet\\commit");
            File[] fileList3 = folder3.listFiles();
            creatingExcelChart(fileList3, authorName);
    }
     if(selection[1])
     {
            File folder4 = new File("C:\\mallet\\workspace\\mallet\\className");
             File[] fileList4 = folder4.listFiles();
             creatingExcelChart(fileList4, authorName);
     }
}
public static void elimination(File[] fileFolder)
{
    for(File file : fileFolder)
        {
           int numberOfLineRestriction=100;
            if(file.isFile()){
                try
```

```
                    {
                      String path =file.getPath();

       path=path.substring(0,path.lastIndexOf('\\')+1)+"Eliminated\\"+path.substring(path.last
IndexOf('\\')+1);
                        String text = "";
                        BufferedReader br = new BufferedReader(new FileReader(file));
                        System.out.println(path);
                        String line = br.readLine();

                        do{
                             BufferedWriter bw = new BufferedWriter(new
FileWriter(path,true));
                             int lineNumber=1;
                             text="";
                             while(lineNumber%numberOfLineRestriction!=(Integer.valueOf("0"))
&& line!=null)
                             {
                                    lineNumber++;
                                   text += line+"\n";
                                   line = br.readLine();
                             }
                               String eliminatedText = Elimination.run(text);
                               bw.write(eliminatedText);
                               bw.close();
                               if(line==null)
                               {    br.close();

                               }
                           }
                        while(line!=null);
                    }
                    catch(IOException ioe)
```

```
                {


                }
            }
        }
}

public static void creatingExcelChart(File[] fileList, String authorName)
{

    ArrayList<String []> topicNumberList = new ArrayList<String []>();
    boolean allAuthors = false;
    if(authorName.equals("ALL"))
    {
            allAuthors = true;
    }
    //Instantiating a Workbook object
    int lineNumber=1;

    Workbook workbook = new Workbook();
    Worksheet worksheet = workbook.getWorksheets().get(0);



    for(File f : fileList)
    {
            if(f.getName().contains(".txt"))
            {
                    try
                    {
                            String filePath = f.getPath();
                            String text="";
                            String temp = filePath.substring(0,filePath.lastIndexOf("-"));
```

```
                String xlsFilePath=temp+".xls";
                BufferedReader br = new BufferedReader(new FileReader(f));
            String line = br.readLine();

            while(line!=null)
            {
                text += line+"\n";
                line = br.readLine();
            }
            br.close();

            String [] cells = text.split("\n");//divide text into rows.
            for(int i=0;i<cells.length;i++)
            {
               String [] cellsLine=null;
               if(filePath.contains("tutorial_composition"))
                    {
                      cellsLine = cells[i].split("\t");//divide lines into cells
                      if(cellsLine.length>1)
                      {

                             String [] topicNumbers = new String[5];
                             Double [] percentages = new Double[5];
                             cellsLine[1]=cellsLine[1].replaceAll("%20", " ");
                             String fileName = cellsLine[1];
                             if(allAuthors || cellsLine[1].contains(authorName))
                             {
                                    fileName =
fileName.substring(fileName.lastIndexOf('/')+1,fileName.lastIndexOf("-"));
                                    for(int k=0;k<10;k++)
                                    {
                                           if(k%2==0)
```

```
                                                {
                                                        topicNumbers[(k/2)]=cellsLine[k+2];
                                                }
                                                else
                                                {
                                                        percentages[((int)Math.floor(k/2))]
= Double.valueOf(cellsLine[k+2]);
                                                }
                                        }


                                        lineNumber = createChart_Composition(fileName,
topicNumbers, percentages, lineNumber, worksheet);
                                        //workbook.save(xlsFilePath);
                                        topicNumberList.add(topicNumbers);
                                }
                                if(authorName.equals("ALL"))
                                {
                                       lineNumber++;
                                }
                          }
                        }
                        else
                        {
                          lineNumber = 1;
                          for(String[] topicNumberArray : topicNumberList)
                          {
                                for(int k=0;k<topicNumberArray.length;k++)
                                {

cells[Integer.valueOf(topicNumberArray[k])]=cells[Integer.valueOf(topicNumberArray[k])].replac
e('\t', ' ');
                                        cellsLine =
cells[Integer.valueOf(topicNumberArray[k])].split(" ");//divide lines into cells
```

```
                                        for(int j=0;j<cellsLine.length;j++)
                                          {
                                                if(j!=1 && j!=0)

worksheet.getCells().get(convertIntToCell(j+3)+String.valueOf(lineNumber)).putValue(cellsLine[
j]);


                                          }
                                        lineNumber++;
                                    }
                                    lineNumber +=17;
                                 }


                               }


                        }
                        workbook.save(xlsFilePath);


                          }
                          catch(Exception ioe)
                {


                }
                }
        }


    }
    public static int createChart_Composition(String title, String [] topicNumbers, Double []
percentages,
                int lineNumber, Worksheet worksheet)
    {
        int dikeyStartPoint = lineNumber;
        int dikeyPointer = lineNumber;
```

```
        int dikeyEndPoint=lineNumber;

        int yatayStartPoint = 2;

        int yatayEndPoint = 1;

        int yatayPointer =1;


        worksheet.getCells().get(convertIntToCell(yatayPointer)+String.valueOf(dikeyPointer)).p
utValue(title.replace("%20"," "));

        yatayPointer++;



        for(int i=0;i<topicNumbers.length;i++)

        {

                yatayPointer=yatayStartPoint;


        worksheet.getCells().get(convertIntToCell(yatayPointer)+String.valueOf(dikeyPointer)).p
utValue(topicNumbers[i]);

                yatayPointer++;


        worksheet.getCells().get(convertIntToCell(yatayPointer)+String.valueOf(dikeyPointer)).p
utValue(percentages[i]);

                dikeyPointer++;

        }

        yatayEndPoint = yatayPointer;

        dikeyEndPoint=dikeyPointer;

        int chartIndex = worksheet.getCharts().add(ChartType.COLUMN,dikeyStartPoint+5,
yatayEndPoint-2, dikeyEndPoint+14, yatayEndPoint+7);

        //Accessing the instance of the newly added chart



        Chart chart = worksheet.getCharts().get(chartIndex);


        chart.getNSeries().add(convertIntToCell(yatayStartPoint+1)+String.valueOf(dikeyStartPoi
nt)+":"+

                        convertIntToCell(yatayStartPoint+1)+String.valueOf(dikeyEndPoint-
1),true);



        chart.getNSeries().setCategoryData(convertIntToCell(yatayStartPoint)+String.valueOf(dik
eyStartPoint)+":"+
```

```
                convertIntToCell(yatayStartPoint)+String.valueOf(dikeyEndPoint-1));


    chart.getTitle().setText(title);
    Series aSeries = chart.getNSeries().get(0);
    aSeries.setName("percentage");
    dikeyEndPoint = dikeyEndPoint+16;
            return dikeyEndPoint;
}
public static String convertIntToCell(int lineNumber)
{
    String line="";
    if(lineNumber>26)
    {
            line+=convertIntToCell(lineNumber/26);
            line+=convertIntToCell(lineNumber%26);
    }
    else{
            switch(lineNumber)
            {
                    case(1):
                    {
                            line = "A";
                            break;
                    }
                    case(2):
                    {
                            line = "B";
                            break;
                    }
                    case(3):
                    {
                            line = "C";
```

```
        break;
}
case(4):
{
        line = "D";
        break;
}
case(5):
{
        line = "E";
        break;
}
case(6):
{
        line = "F";
        break;
}
case(7):
{
        line = "G";
        break;
}
case(8):
{
        line = "H";
        break;
}
case(9):
{
        line = "I";
        break;
}
```

```
case(10):
{
        line = "J";
        break;
}
case(11):
{
        line = "K";
        break;
}
case(12):
{
        line = "L";
        break;
}
case(13):
{
        line = "M";
        break;
}
case(14):
{
        line = "N";
        break;
}
case(15):
{
        line = "O";
        break;
}
case(16):
{
```

```
        line = "P";
        break;
}
case(17):
{
        line = "Q";
        break;
}
case(18):
{
        line = "R";
        break;
}
case(19):
{
        line = "S";
        break;
}
case(20):
{
        line = "T";
        break;
}
case(21):
{
        line = "U";
        break;
}
case(22):
{
        line = "V";
        break;
```

```
                }
                case(23):
                {
                        line = "W";
                        break;
                }
                case(24):
                {
                        line = "X";
                        break;
                }
                case(25):
                {
                        line = "Y";
                        break;
                }
                case(26):
                {
                        line = "Z";
                        break;
                }

            }

    }
    return line;
}
public static void createBatchFile(String projectName)
{
    try
    {
```

```
            BufferedWriter outputStream = new BufferedWriter(new
FileWriter("C:\\mallet\\workspace\\commitFiles\\"+projectName+".bat"));


       outputStream.write("cd..\ncd..\ncd..\ncd..\ncd..\ncd..\ncd..\ncd..\ncd..\ncd..\nc
d..\ncd..\n"

                           + "cd Users\ncd fadi\ncd Documents\ncd GitHub\ncd
"+projectName+"\ngit log -p > "

                           + "C:\\mallet\\workspace\\commitFiles\\"+projectName+".txt
2>&1");

            outputStream.close();

            ProcessBuilder pb = new
ProcessBuilder("C:\\mallet\\workspace\\commitFiles\\"+projectName+".bat");

            Process p = pb.start();

            p.waitFor();



       }

       catch(Exception ioe)

       {



       }

    }

    public static String createCommitFiles(String projectName)

    {

       createBatchFile(projectName);

       Developer.readDeveloperFile();

       createFiles();

       String authorsName = getAuthorsName(projectName);

       return authorsName;

    }

    public static String getAuthorsName(String projectName)

    {

       String authorsName="";

       try

        {
```

```
            File file = new File("C:\\mallet\\workspace\\developers\\developers.txt");
            BufferedReader inputStream = new BufferedReader(new FileReader(file));
            String line = inputStream.readLine();

            while(line!=null)
            {
                authorsName+=line+"\n";
                line=inputStream.readLine();
            }
            inputStream.close();
        }
        catch (IOException ioe)
        {

        }
        return authorsName;
    }
    public static void malletOperations(String numberOfTopic)
    {
        malletElimination();
        malletClassNames(numberOfTopic);
        malletChangedLines(numberOfTopic);
        malletCommit(numberOfTopic);
        malletAll(numberOfTopic);
    }
}
```

Commit.java

```
package parsing;

import java.util.ArrayList;
import java.io.*;
```

```
import parsing.developers.Developer;

public class Commit

{



    public ArrayList<String>  commitHash;

    public ArrayList<String>  author;

    public ArrayList<String>  date;

    public String commitText;

    public ArrayList<String>  classNames;

    public String addedLines;

    public String removedLines;

    public String modifiedLines;



    public Commit()

    {


    }

    public static ArrayList<Commit> execute(String filePath)

    {

       ArrayList <Commit> commitList = readFile(filePath);

        return commitList;

    }

    public Commit(ArrayList<String> commitHash, ArrayList<String> author, ArrayList<String>
date,String commitText ,ArrayList<String> classNames,

                String addedLines, String removedLines, String linesInCommit)

    {


            this.commitHash = commitHash;

            this.author = author;

            this.date = date;
```

```
    this.commitText = commitText;
    this.classNames = classNames;
    this.addedLines = addedLines;
    this.removedLines = removedLines;
    this.modifiedLines = linesInCommit;

}
public static ArrayList<Commit> readFile(String filePath)
{
    ArrayList <Commit> commitList = new ArrayList<Commit>();

   try
   {

        FileInputStream fis = new FileInputStream(filePath);

        DataInputStream dis = new DataInputStream(fis);

        BufferedReader br = new BufferedReader (new InputStreamReader(dis));


        String line = br.readLine();


        while(line!=null) //loop for text file
        {

            ArrayList<String> commitHash = new ArrayList<String>();
            ArrayList<String> author = new ArrayList<String>();
            ArrayList<String> date = new ArrayList<String>();
            String commitText = "";
            ArrayList<String> classNames = new ArrayList<String>();
```

```
                String addedLines ="";

                String removedLines ="";

                String linesInCommit = "";




                if(line.length()>1)

                {




                    while(line!=null&&line.length()>7 &&
!line.substring(0,6).equals("Author")&&

                           !line.substring(0,6).equals("Date: ") &&
!line.substring(0,3).equals("@@ "))

                           //commit loop, there can be more than one commit hash line!

                    {

                        if(line.substring(0,7).equals("commit "))

                        {

                            commitHash.add(line.replace("commit ", ""));

                            line=br.readLine();



                        }

                        else if(line!=null && line.substring(0,7).equals("Merge: "))

                        {

                             line=br.readLine();



                        }

                    }

                    while(line!=null && line.length()>7 &&
!line.substring(0,6).equals("Commit")&&

                           !line.substring(0,6).equals("Date: ") &&
!line.substring(0,3).equals("@@ "))

                        //Author loop, there can be more than one author line!

                    {
```

```
                        if(line.substring(0,8).equals("Author: "))

                        {

                                author.add(line.replace("Author: ", ""));

                                line=br.readLine();



                        }

                    }

                    while(line!=null && line.length()>7 &&
!line.substring(0,6).equals("Commit")&&

                           !line.substring(0,8).equals("Author: ") &&
!line.substring(0,3).equals("@@ "))

                        //Date loop, there can be more than one date line!

                    {

                        if(line.substring(0,6).equals("Date: "))

                        {

                                date.add(line.replace("Date: ", ""));

                                line=br.readLine();



                        }

                    }

                    /*while(line.length()>8 && !line.substring(0,6).equals("Commit")&&

                           !line.substring(0,8).equals("Author: ") &&
!line.substring(0,3).equals("@@ ")

                            &&!line.substring(0,5).equals("diff "))*/

                    //boolean length = false;



                    while(line!=null && line.length()<8 || (!line.substring(0,5).equals("diff
") && !line.substring(0,7).equals("commit ")))

                         //Commit Text loop, there can be more than one commit text line!

                    {

                        commitText+=(line+"\n");

                        line=br.readLine();
```

```
                    }
                    if(!line.substring(0,6).equals("commit"))
                    {



                    while(line!=null && line.length() >3 && !line.substring(0,3).equals("@@
"))
                        //class names loop. there can be more than one class name!
                    {


                        if(line.substring(0, 3).equals("+++") || line.substring(0,
3).equals("---"))
                        {
                           if(line.contains("."))
                           {
                               int i = line.lastIndexOf(".");
                               int j = line.lastIndexOf("/");
                               if(i>j+1)
                                    classNames.add(line.substring(j+1,i));
                               else
                                    classNames.add(line.substring(j+1));
                           }
                           else
                           {
                               classNames.add(line.substring(4));
                           }
                        }
                        line=br.readLine();


                    }
```

```
                    int line_counter = 0;

                       while(line!=null && (line.length()<8 ||
!line.substring(0,7).equals("commit ")) && line_counter<3000)

                       {

                             if(line.length()>1)

                           {

                             if(line.charAt(0)=='+')

                               {

                                  addedLines+=line+"\n";

                               }

                               else if(line.charAt(0)=='-')

                               {

                                  removedLines+=line+"\n";

                               }

                               linesInCommit+=line+"\n";



                           }



                             line=br.readLine();

                        line_counter++;

                       }

                       if(line_counter>=3000)

                       {

                             System.out.println(line_counter);

                             do

                             {

                                    System.out.println(line_counter);

                                    line=br.readLine();

                                    line_counter++;

                        }

                             while(line!=null&& !(line.contains("commit") &&
line.substring(0,6).equals("commit")));

                       }
```

```
                        System.out.println(commitList.size()+"-"+commitHash);
                    }
                        commitList.add(new
Commit(commitHash,author,date,commitText,classNames,addedLines,removedLines,linesInCommit));
                        if(commitList.size()>400)
                        {
                              for(Commit c : commitList)
                              {
                                    Developer.createDeveloper(c);
                              }
                              commitList=new ArrayList<Commit>();
                        }
                  }


            }
            br.close();
        }
        catch(IOException ioe)
        {
               System.out.println(ioe.getMessage());
    }
        catch(Exception e)
        {
               System.out.println(e.getMessage());
        }
        return commitList;
    }


}
```

Developer.java

```
/*
 * To change this template, choose Tools | Templates
```

```
 * and open the template in the editor.
 */
package parsing.developers;
import java.util.ArrayList;
import java.io.*;

import parsing.Commit;
/**
 *
 * @author fadi
 */
public class Developer {

    public String name;
    public String DOB;

    public ArrayList<Commit> commitList;

    public final static ArrayList<Developer> developerList=new ArrayList<Developer>();

    public Developer(String name, String DOB, String commitlist)
    {
        this.name = name;
        this.DOB= DOB;




    }
    public Developer(String name)
    {
        this.name = name;
```

```
    }
    public static void createDeveloper(Commit commit)
    {
        for(String authorName : commit.author)
        {
            boolean newDeveloper = true;
            for(Developer dev : developerList)
            {
                if(authorName.equalsIgnoreCase(dev.name))
                {
                    newDeveloper = false;
                    break;
                }
            }
            if(newDeveloper)
            {
               System.out.println("bura 3");
                try
                {
                      BufferedWriter outputStream = new BufferedWriter(new
FileWriter("C:\\mallet\\workspace\\developers\\developers.txt",true));
                    outputStream.write(authorName+"\n");
                    outputStream.flush();
                    outputStream.close();
                    readDeveloperFile();
                    createDeveloperLogFile(authorName);
                }
                catch (IOException ioe)
                {

                }
            }
```

```
            addCommit(authorName, commit.commitHash,commit.commitText);

            addClass(authorName, commit.commitHash, commit.classNames);

            addModifiedLines(authorName, commit.commitHash, commit.modifiedLines);

            addAll(authorName,commit.commitText,commit.classNames,commit.modifiedLines);

        }

    }
    public static void readDeveloperFile()
    {
        developerList.clear();
        try
        {
            File file = new File("C:\\mallet\\workspace\\developers\\developers.txt");
            BufferedReader inputStream = new BufferedReader(new FileReader(file));
            String line = inputStream.readLine();

            while(line!=null)
            {
                developerList.add(new Developer(line));
                line=inputStream.readLine();
            }
            inputStream.close();
        }
        catch (IOException ioe)
        {

        }
```

```
    }

    public static void createDeveloperLogFile(String name)

    {

        String trimmedName = name.substring(0, name.indexOf(" <"));

        try

        {

            BufferedWriter outputStreamCommit = new BufferedWriter(new
FileWriter("C:\\mallet\\workspace\\developers\\commitTexts\\"+trimmedName+"-commit.txt"));

            outputStreamCommit.write("");

            outputStreamCommit.flush();

            outputStreamCommit.close();




            BufferedWriter outputStreamClass = new BufferedWriter(new
FileWriter("C:\\mallet\\workspace\\developers\\classNameTexts\\"+trimmedName+"-
classNames.txt"));

            outputStreamClass.write("");

            outputStreamClass.flush();

            outputStreamClass.close();




            BufferedWriter outputStreamMethod = new BufferedWriter(new
FileWriter("C:\\mallet\\workspace\\developers\\changedLineTexts\\"+trimmedName+"-
changedLines.txt"));

            outputStreamMethod.write("");

            outputStreamMethod.flush();

            outputStreamMethod.close();



            BufferedWriter outputStreamAll = new BufferedWriter(new
FileWriter("C:\\mallet\\workspace\\developers\\all\\"+trimmedName+"-all.txt"));

            outputStreamAll.write("");

            outputStreamAll.flush();

            outputStreamAll.close();

            readDeveloperFile();
```

```
        }
        catch(IOException ioe)
        {

        }
    }
    public static void addCommit(String authorName, ArrayList<String> commitHash, String
commitText)
    {
        authorName = authorName.substring(0, authorName.indexOf(" <"));

        try
        {

            BufferedWriter outputStream = new BufferedWriter(new
FileWriter("C:\\mallet\\workspace\\developers\\commitTexts\\"+authorName+"-commit.txt",true));

            outputStream.write(commitText+"\n");
            outputStream.flush();
            outputStream.close();
        }
        catch(IOException ioe)
        {

        }
    }
    public static void addAll(String authorName,String commitText, ArrayList<String>
className, String modifiedLines)
    {
       authorName = authorName.substring(0, authorName.indexOf(" <"));

       try
```

```
        {




            String classNames ="";

            for(String c : className)

            {

                if(!classNames.contains(c)){

                    classNames+=c+"\n";

                }

            }

            BufferedWriter outputStream = new BufferedWriter(new
FileWriter("C:\\mallet\\workspace\\developers\\all\\"+authorName+"-all.txt",true));



            outputStream.write(commitText+"\n"+classNames+"\n"+modifiedLines+"\n");

            outputStream.flush();

            outputStream.close();

        }

        catch(IOException ioe)

        {



        }

    }

    public static void addClass(String authorName, ArrayList<String> commitHash,
ArrayList<String> className)

    {

         authorName = authorName.substring(0, authorName.indexOf(" <"));



         try

         {



            String classNames ="";

            for(String c : className)

            {
```

```
                if(!classNames.contains(c)){

                    classNames+=c+"\n";

                }

            }


            BufferedWriter outputStream = new BufferedWriter(new
FileWriter("C:\\mallet\\workspace\\developers\\classNameTexts\\"+authorName+"-
classNames.txt",true));



            outputStream.write(classNames+"\n");

            outputStream.flush();

            outputStream.close();

        }

        catch(IOException ioe)

        {



        }

    }

    public static void addModifiedLines(String authorName, ArrayList<String> commitHash,
String modifiedLines)

    {

        authorName = authorName.substring(0, authorName.indexOf(" <"));



        try

        {









            BufferedWriter outputStream = new BufferedWriter(new
FileWriter("C:\\mallet\\workspace\\developers\\changedLineTexts\\"+authorName+"-
changedLines.txt",true));



            outputStream.write(modifiedLines+"\n");

            outputStream.flush();
```

```
            outputStream.close();

        }
        catch(IOException ioe)
        {

        }
    }
    public void removeDeveloper()
    {

    }
    public void modifyDeveloper()
    {

    }
}
```

Elimination.java

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package parsing.elimination;

import java.io.*;

/**
 *
 * @author fadi
 */
public class Elimination {
```

```
    public static String run(String text)
    {
        String keywords="";
        try
        {
            File file = new File("keywordsIncludingJavaAndC++.txt");

            BufferedReader br = new BufferedReader(new FileReader(file));
            String line = br.readLine();

            while(line!=null)
            {
                keywords+=line+"\n";
                line=br.readLine();
            }

        }
        catch(IOException ioe)
        {

        }
        String txt = "";
        txt = Utils.eliminateBlanks(text);
        txt = Utils.eliminateNumbers(txt);
        txt = Utils.eliminateNonLiterals(txt);
        txt = IdentifierSplitter.splitIdentifiers(txt, false);
        txt = Utils.eliminateKeywords(txt,
"C:\\Users\\fadi\\Documents\\Eclipse\\Parsing\\src\\parsing\\elimination\\keywordsIncludingJav
aAndC++.txt");
        txt = GeneralStemmer.stemmingPorter(txt);
        txt = Utils.eliminateShortStrings(txt);
        return txt;
    }
```

```
}
```

Utils.java

```
package parsing.elimination;

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.ObjectInputStream.GetField;
import java.util.ArrayList;
import java.util.StringTokenizer;

public class Utils {



	public static String eliminateBlanks(String stringIn)
	{
		String stringOut = stringIn.replaceAll("\n"," ");
		stringOut = stringOut.replaceAll("\r"," ");
		stringOut = stringOut.replaceAll("\t"," ");
		int pos = stringOut.indexOf("  ");
		while (pos>0 )
		{
			stringOut = stringOut.replaceAll("  ", " ");
			pos = stringOut.indexOf("  ");
		}
		return stringOut;
	}



	public static String eliminateNumbers(String stringIn)
	{
		String stringOut = "";
		String [] words = stringIn.split("[ \t\n]");
```

```
        for (int j=0; j<words.length; j++)
        {
                StringTokenizer st = new StringTokenizer(words[j]);
                String strReturn = "";

                while ( st.hasMoreTokens() ){
                        String word = st.nextToken();
                        if (word.matches("[a-zA-Z_]+[a-zA-Z_0-9]*"))
                        {
                                stringOut += word + ' ';
                        }
                }
        }
        return stringOut;
}


public static String eliminateNonLiterals(String stringIn)
{
        String stringOut = new String();
        stringOut = stringIn.replaceAll("[^a-zA-Z0-9_]", " ");
        stringOut = eliminateBlanks(stringOut);
        return stringOut;
}
public static String eliminateKeywords(String stringIn, String fileName)
{
        String stringOut = new String();
        ArrayList<String> keywords = getKeywordsFromFile(fileName);
        String [] words = stringIn.split("[ \t\n]");
        for (int i=0; i<words.length; i++)
                if (!keywords.contains(words[i].toLowerCase().trim()))
                        stringOut += words[i].trim()+ " ";
```

```
        return stringOut;
}



private static ArrayList<String> getKeywordsFromFile(String fileName)
{
        ArrayList<String> list = new ArrayList<String>();
        BufferedReader in;
        try {
                in = new BufferedReader(new FileReader(fileName));

        String line;
        while((line = in.readLine())!=null)
        {
                String [] words = line.split("[ \t\n]");
                for (int i=0; i<words.length; i++)
                        if (!list.contains(words[i].toLowerCase().trim()))
                                list.add(words[i].toLowerCase().trim());
        }
        in.close();
        }
        catch (Exception e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
        }
        return list;
}

public static String eliminateShortStrings(String stringIn)
{
        String stringOut = new String();
        String [] words = stringIn.split("[ \t\n]");
```

```
		for (int i=0; i<words.length; i++)
			if (words[i].trim().length()>=3)
				stringOut += words[i].trim()+ " ";
		return stringOut;
	}
}
```

GeneralStemmer.java

```
package parsing.elimination;

// general class that provides access to the stemming functions of different stemmers
// for now, only the Porter stemmer is available; future versions will offer more stemmers

public class GeneralStemmer {

	public static String stemmingPorter(String stringIn)
	{
		String stringOut = new String();

		String [] words = stringIn.split("[ \t\n]");
		for (int i = 0; i<words.length; i++)
			{
				PorterStemmer stemmer = new PorterStemmer();
				String stem = "";
				String word = words[i];
				for (int j = 0; j<word.length(); j++)
					stemmer.add(Character.toLowerCase(word.charAt(j)));
				stemmer.stem();
				stem = stemmer.toString();

				stringOut += stem.trim()+ " ";

			}

		return stringOut;
	}

}
```

ContentRemover.java

```
package parsing.elimination;


import java.io.File;

import java.io.FileReader;

import java.io.FileWriter;

import java.io.StreamTokenizer;


public class CommentsRemover {
```

```
public static String removeComments(String source)
{
        String result = new String();
        try{

               FileWriter fwTemp = new FileWriter("c:/tempComments.txt");
               fwTemp.write(source);
               fwTemp.close();

         FileReader rd = new FileReader("c:/tempComments.txt");

         StreamTokenizer st = new StreamTokenizer(rd);

         // Prepare the tokenizer for Java-style tokenizing rules
         st.wordChars('_', '_');
         st.eolIsSignificant(true);

         // If whitespace is not to be discarded, make this call
         st.ordinaryChars(0, '_');

         // These calls caused comments to be discarded
         st.slashSlashComments(true);
         st.slashStarComments(true);

         // Parse the file
         int token = st.nextToken();
         while (token != StreamTokenizer.TT_EOF) {
             switch (token) {
             case StreamTokenizer.TT_WORD:
                 // A word was found; the value is in sval
                 String word = st.sval;
```

```
    result += st.sval;
    break;
case '"':
    // A double-quoted string was found; sval contains the contents
    String dquoteVal = st.sval;
    if (st.sval!=null)
    {
    result += st.sval;
    }
    else
          result += "\"";
    break;
case '\'':
    // A single-quoted string was found; sval contains the contents
          String squoteVal = st.sval;
                if (st.sval==null)
                       result += '\'';
                else
                       result += st.sval;
    break;
case StreamTokenizer.TT_EOL:
    // End of line character found
   result += "\r\n";
    break;
case StreamTokenizer.TT_EOF:
    // End of file has been reached
    break;
default:
    // A regular character was found; the value is the token itself
    char ch = (char)st.ttype;
    result += ch;
    break;
```

```
                }

                token = st.nextToken();

            }

            rd.close();


            File f = new File ("c:/tempComments.txt");

            boolean success = f.delete();

        }

            catch(Exception e)

            {

                   e.printStackTrace();

            }

            return result;

    }

}
```

IdentifierSplitter.java

```
package parsing.elimination;

import java.util.StringTokenizer;

public class IdentifierSplitter {

    public static String splitIdentifiers(String stringIn, boolean keepOriginal)
    {
        String stringOut = new String();

        String [] words = stringIn.split("[ \t\n]");
        for (int j=0; j<words.length; j++)
        {
            StringTokenizer st = new StringTokenizer(words[j]);
            String strReturn = "";

            while ( st.hasMoreTokens() ){
                String word = st.nextToken();
                String temp_word = new String();
                int len = word.length()-1;

                boolean lastCharWasNumber = false;
                boolean lastCharWasSmallLetter = false;
                boolean lastCharWasBigLetter = false;
                boolean lastCharWasNoneOfAbove = false;
                boolean letterFoundBeforeNumber = false;
```

```
                        int i = 0;
                        while (i < len)
                        {
                              if (word.charAt(i) >= '0' && word.charAt(i) <= '9')
                              {

                                    temp_word += word.charAt(i);
                  /*                if (lastCharWasNumber == false)
                                    {
                                          if (i<len)
                                                if ((word.charAt(i+1) >= 'A' &&
word.charAt(i+1) <= 'Z') ||word.charAt(i+1) >= 'a' && word.charAt(i+1) <= 'z'  );
                                                      temp_word += ' ';

                                    }
                              /*          else
                                    {
                                          temp_word += word.charAt(i);
                                    }*/
                                    lastCharWasNumber = true;
                                    lastCharWasBigLetter = false;
                                    lastCharWasSmallLetter = false;
                                    lastCharWasNoneOfAbove = false;
                              }
                              else
                              if (word.charAt(i) >= 'A' && word.charAt(i) <= 'Z' )
                              {
                                    if (lastCharWasBigLetter == false)
                                    {
                                          if (i>0)
                                                temp_word += ' ';
                                          temp_word += word.charAt(i);
                                    }
                                    else
                                    {
                                          if (i<len-1 && ((word.charAt(i+1) >=
'a' && word.charAt(i+1) <= 'z')))
                                          {
                                                if (i>0)
                                                      temp_word += ' ';
                                                temp_word += word.charAt(i);
                                          }
                                          else
                                          {
                                                temp_word += word.charAt(i);
                                          }
                                    }
                                    lastCharWasBigLetter = true;

                                    lastCharWasNumber = false;
                                    lastCharWasSmallLetter = false;
                                    lastCharWasNoneOfAbove = false;
                                    letterFoundBeforeNumber = true;
                              }
                              else
```

```
                                if ((word.charAt(i) >= 'a' && word.charAt(i) <=
'z'))
                                {
                                        if (lastCharWasNoneOfAbove ||
lastCharWasBigLetter || lastCharWasSmallLetter)
                                        {
                                                temp_word += word.charAt(i);
                                        }
                                        else
                                        {
                                                if (i>0)
                                                        temp_word += ' ';
                                                temp_word += word.charAt(i);
                                        }
                                        lastCharWasBigLetter = false;

                                        lastCharWasNumber = false;
                                        lastCharWasSmallLetter = true;
                                        lastCharWasNoneOfAbove = false;
                                        letterFoundBeforeNumber = true;
                                }
                                else
                                if ( word.charAt(i) == '_' )
                                {
                                        if (i>0)
                                                temp_word += ' '; //Replace underscore
by space
                                        lastCharWasBigLetter = false;
                                        lastCharWasNumber = false;
                                        lastCharWasSmallLetter = false;
                                        lastCharWasNoneOfAbove = true;
                                        letterFoundBeforeNumber = false;
                                }
                                else
                                {
                                        temp_word += word.charAt(i);
                                        lastCharWasBigLetter = false;
                                        lastCharWasNumber = false;
                                        lastCharWasSmallLetter = false;
                                        lastCharWasNoneOfAbove = true;
                                }

                                i++;
                        }

                        if ( word.charAt(i) != '_' ) // so that if the last
character is an _, it is ignored
                                temp_word += word.charAt(i);

                        strReturn += temp_word;
                        strReturn = strReturn.toLowerCase().trim();
                        strReturn += ' ';
                        if ( keepOriginal == true  &&
(!temp_word.trim().equals(word.trim()))){
                                strReturn += word;
```

```
                                strReturn += ' ';
                            }
                        }
                        stringOut += strReturn;
                }
                return stringOut;
        }
}
```

PorterStemmer.java

```
package parsing.elimination;
/*

Porter stemmer in Java. The original paper is in

    Porter, 1980, An algorithm for suffix stripping, Program, Vol. 14,
    no. 3, pp 130-137,

See also http://www.tartarus.org/~martin/PorterStemmer

History:

Release 1

Bug 1 (reported by Gonzalo Parra 16/10/99) fixed as marked below.
The words 'aed', 'eed', 'oed' leave k at 'a' for step 3, and b[k-1]
is then out outside the bounds of b.

Release 2

Similarly,

Bug 2 (reported by Steve Dyrdahl 22/2/00) fixed as marked below.
'ion' by itself leaves j = -1 in the test for 'ion' in step 5, and
b[j] is then outside the bounds of b.
```

```
Release 3

Considerably revised 4/9/00 in the light of many helpful suggestions
from Brian Goetz of Quiotix Corporation (brian@quiotix.com).

Release 4

*/

import java.io.*;

/**
* Stemmer, implementing the Porter Stemming Algorithm
*
* The Stemmer class transforms a word into its root form.  The input
* word can be provided a character at time (by calling add()), or at once
* by calling one of the various stem(something) methods.
*/

public class PorterStemmer
{  private char[] b;
private int i,     /* offset into b */
            i_end, /* offset to end of stemmed word */
            j, k;
private static final int INC = 50;
                  /* unit of size whereby b is increased */
public PorterStemmer()
{  b = new char[INC];
   i = 0;
   i_end = 0;
}
```

```
/**
 * Add a character to the word being stemmed.  When you are finished
 * adding characters, you can call stem(void) to stem the word.
 */

public void add(char ch)
{  if (i == b.length)
   {  char[] new_b = new char[i+INC];
      for (int c = 0; c < i; c++) new_b[c] = b[c];
      b = new_b;
   }
   b[i++] = ch;
}


/** Adds wLen characters to the word being stemmed contained in a portion
 * of a char[] array. This is like repeated calls of add(char ch), but
 * faster.
 */

public void add(char[] w, int wLen)
{  if (i+wLen >= b.length)
   {  char[] new_b = new char[i+wLen+INC];
      for (int c = 0; c < i; c++) new_b[c] = b[c];
      b = new_b;
   }
   for (int c = 0; c < wLen; c++) b[i++] = w[c];
}

/**
 * After a word has been stemmed, it can be retrieved by toString(),
```

```
 * or a reference to the internal buffer can be retrieved by getResultBuffer
 * and getResultLength (which is generally more efficient.)
 */
public String toString() { return new String(b,0,i_end); }

/**
 * Returns the length of the word resulting from the stemming process.
 */
public int getResultLength() { return i_end; }

/**
 * Returns a reference to a character buffer containing the results of
 * the stemming process.  You also need to consult getResultLength()
 * to determine the length of the result.
 */
public char[] getResultBuffer() { return b; }

/* cons(i) is true <=> b[i] is a consonant. */

private final boolean cons(int i)
{  switch (b[i])
   {  case 'a': case 'e': case 'i': case 'o': case 'u': return false;
      case 'y': return (i==0) ? true : !cons(i-1);
      default: return true;
   }
}

/* m() measures the number of consonant sequences between 0 and j. if c is
   a consonant sequence and v a vowel sequence, and <..> indicates arbitrary
   presence,

      <c><v>       gives 0
```

```
      <c>vc<v>     gives 1
      <c>vcvc<v>   gives 2
      <c>vcvcvc<v> gives 3
      ....
*/

private final int m()
{  int n = 0;
   int i = 0;
   while(true)
   {  if (i > j) return n;
      if (! cons(i)) break; i++;
   }
   i++;
   while(true)
   {  while(true)
      {  if (i > j) return n;
            if (cons(i)) break;
            i++;
      }
      i++;
      n++;
      while(true)
      {  if (i > j) return n;
         if (! cons(i)) break;
         i++;
      }
      i++;
    }
}

/* vowelinstem() is true <=> 0,...j contains a vowel */
```

```
private final boolean vowelinstem()
{  int i; for (i = 0; i <= j; i++) if (! cons(i)) return true;
   return false;
}

/* doublec(j) is true <=> j,(j-1) contain a double consonant. */

private final boolean doublec(int j)
{  if (j < 1) return false;
   if (b[j] != b[j-1]) return false;
   return cons(j);
}

/* cvc(i) is true <=> i-2,i-1,i has the form consonant - vowel - consonant
   and also if the second c is not w,x or y. this is used when trying to
   restore an e at the end of a short word. e.g.

      cav(e), lov(e), hop(e), crim(e), but
      snow, box, tray.

*/

private final boolean cvc(int i)
{  if (i < 2 || !cons(i) || cons(i-1) || !cons(i-2)) return false;
   {  int ch = b[i];
      if (ch == 'w' || ch == 'x' || ch == 'y') return false;
   }
   return true;
}

private final boolean ends(String s)
```

```
{  int l = s.length();
   int o = k-l+1;
   if (o < 0) return false;
   for (int i = 0; i < l; i++) if (b[o+i] != s.charAt(i)) return false;
   j = k-l;
   return true;
}

/* setto(s) sets (j+1),...k to the characters in the string s, readjusting
   k. */

private final void setto(String s)
{  int l = s.length();
   int o = j+1;
   for (int i = 0; i < l; i++) b[o+i] = s.charAt(i);
   k = j+l;
}

/* r(s) is used further down. */

private final void r(String s) { if (m() > 0) setto(s); }

/* step1() gets rid of plurals and -ed or -ing. e.g.

       caresses  ->  caress
       ponies    ->  poni
       ties      ->  ti
       caress    ->  caress
       cats      ->  cat

       feed      ->  feed
       agreed    ->  agree
```

```
       disabled  ->  disable


       matting   ->  mat
       mating    ->  mate
       meeting   ->  meet
       milling   ->  mill
       messing   ->  mess


       meetings  ->  meet


*/


private final void step1()
{  if (b[k] == 's')
   {  if (ends("sses")) k -= 2; else
      if (ends("ies")) setto("i"); else
      if (b[k-1] != 's') k--;
   }
   if (ends("eed")) { if (m() > 0) k--; } else
   if ((ends("ed") || ends("ing")) && vowelinstem())
   {  k = j;
      if (ends("at")) setto("ate"); else
      if (ends("bl")) setto("ble"); else
      if (ends("iz")) setto("ize"); else
      if (doublec(k))
      {  k--;
         {  int ch = b[k];
            if (ch == 'l' || ch == 's' || ch == 'z') k++;
         }
      }
      else if (m() == 1 && cvc(k)) setto("e");
  }
```

```
}

/* step2() turns terminal y to i when there is another vowel in the stem. */

private final void step2() { if (ends("y") && vowelinstem()) b[k] = 'i'; }

/* step3() maps double suffices to single ones. so -ization ( = -ize plus
   -ation) maps to -ize etc. note that the string before the suffix must give
   m() > 0. */

private final void step3() { if (k == 0) return; /* For Bug 1 */ switch (b[k-1])
{
    case 'a': if (ends("ational")) { r("ate"); break; }
              if (ends("tional")) { r("tion"); break; }
              break;
    case 'c': if (ends("enci")) { r("ence"); break; }
              if (ends("anci")) { r("ance"); break; }
              break;
    case 'e': if (ends("izer")) { r("ize"); break; }
              break;
    case 'l': if (ends("bli")) { r("ble"); break; }
              if (ends("alli")) { r("al"); break; }
              if (ends("entli")) { r("ent"); break; }
              if (ends("eli")) { r("e"); break; }
              if (ends("ousli")) { r("ous"); break; }
              break;
    case 'o': if (ends("ization")) { r("ize"); break; }
              if (ends("ation")) { r("ate"); break; }
              if (ends("ator")) { r("ate"); break; }
              break;
    case 's': if (ends("alism")) { r("al"); break; }
              if (ends("iveness")) { r("ive"); break; }
```

```
            if (ends("fulness")) { r("ful"); break; }
            if (ends("ousness")) { r("ous"); break; }
            break;
    case 't': if (ends("aliti")) { r("al"); break; }
            if (ends("iviti")) { r("ive"); break; }
            if (ends("biliti")) { r("ble"); break; }
            break;
    case 'g': if (ends("logi")) { r("log"); break; }
} }

/* step4() deals with -ic-, -full, -ness etc. similar strategy to step3. */

private final void step4() { switch (b[k])
{
    case 'e': if (ends("icate")) { r("ic"); break; }
            if (ends("ative")) { r(""); break; }
            if (ends("alize")) { r("al"); break; }
            break;
    case 'i': if (ends("iciti")) { r("ic"); break; }
            break;
    case 'l': if (ends("ical")) { r("ic"); break; }
            if (ends("ful")) { r(""); break; }
            break;
    case 's': if (ends("ness")) { r(""); break; }
            break;
} }

/* step5() takes off -ant, -ence etc., in context <c>vcvc<v>. */

private final void step5()
{   if (k == 0) return; /* for Bug 1 */ switch (b[k-1])
    {  case 'a': if (ends("al")) break; return;
```

```
        case 'c': if (ends("ance")) break;
                  if (ends("ence")) break; return;
        case 'e': if (ends("er")) break; return;
        case 'i': if (ends("ic")) break; return;
        case 'l': if (ends("able")) break;
                  if (ends("ible")) break; return;
        case 'n': if (ends("ant")) break;
                  if (ends("ement")) break;
                  if (ends("ment")) break;
                  /* element etc. not stripped before the m */
                  if (ends("ent")) break; return;
        case 'o': if (ends("ion") && j >= 0 && (b[j] == 's' || b[j] == 't')) break;
                                    /* j >= 0 fixes Bug 2 */
                  if (ends("ou")) break; return;
                  /* takes care of -ous */
        case 's': if (ends("ism")) break; return;
        case 't': if (ends("ate")) break;
                  if (ends("iti")) break; return;
        case 'u': if (ends("ous")) break; return;
        case 'v': if (ends("ive")) break; return;
        case 'z': if (ends("ize")) break; return;
        default: return;
    }
    if (m() > 1) k = j;
}

/* step6() removes a final -e if m() > 1. */

private final void step6()
{  j = k;
   if (b[k] == 'e')
   {  int a = m();
```

```
      if (a > 1 || a == 1 && !cvc(k-1)) k--;
   }
   if (b[k] == 'l' && doublec(k) && m() > 1) k--;
}

/** Stem the word placed into the Stemmer buffer through calls to add().
 * Returns true if the stemming process resulted in a word different
 * from the input.  You can retrieve the result with
 * getResultLength()/getResultBuffer() or toString().
 */
public void stem()
{  k = i - 1;
   if (k > 1) { step1(); step2(); step3(); step4(); step5(); step6(); }
   i_end = k+1; i = 0;
}

/** Test program for demonstrating the Stemmer.  It reads text from a
 * a list of files, stems each word, and writes the result to standard
 * output. Note that the word stemmed is expected to be in lower case:
 * forcing lower case must be done outside the Stemmer class.
 * Usage: Stemmer file-name file-name ...
 */
public static void main(String[] args)
{
   char[] w = new char[501];
   PorterStemmer s = new PorterStemmer();
   for (int i = 0; i < args.length; i++)
   try
   {
      FileInputStream in = new FileInputStream(args[i]);

      try
```

```
{ while(true)

  {  int ch = in.read();
     if (Character.isLetter((char) ch))
     {
        int j = 0;
        while(true)
        {  ch = Character.toLowerCase((char) ch);
           w[j] = (char) ch;
           if (j < 500) j++;
           ch = in.read();
           if (!Character.isLetter((char) ch))
           {
              /* to test add(char ch) */
              for (int c = 0; c < j; c++) s.add(w[c]);

              /* or, to test add(char[] w, int j) */
              /* s.add(w, j); */

              s.stem();
              {  String u;

                 /* and now, to test toString() : */
                 u = s.toString();

                 /* to test getResultBuffer(), getResultLength() : */
                 /* u = new String(s.getResultBuffer(), 0, s.getResultLength()); */

                 System.out.print(u);
              }
              break;
           }
```

```
            }
          }
          if (ch < 0) break;
          System.out.print((char)ch);
        }
      }
      catch (IOException e)
      {  System.out.println("error reading " + args[i]);
         break;
      }
   }
   catch (FileNotFoundException e)
   {  System.out.println("file " + args[i] + " not found");
      break;
   }
}
}
```

Statistics.java

```
package parsing.elimination;

import java.io.BufferedReader;
import java.io.FileReader;
import java.util.ArrayList;

public class Statistics {

        private int getNumberNonUniqueWordsInFile(String fileName)
        {
                int numberWordsTotal = 0;
                BufferedReader in;
                try {
                        in = new BufferedReader(new FileReader(fileName));
```

```
                String line;
                while((line = in.readLine())!=null)
                {
                        String [] words = line.split("[ \t\n]");
                        for (int i=0; i<words.length; i++)
                                numberWordsTotal++;
                }
                in.close();
        }
        catch (Exception e)
        {
                e.printStackTrace();
        }

        return numberWordsTotal;
}



private int getNumberUniqueWordsInFile(String fileName)
{
        ArrayList<String> list = new ArrayList<String>();
        BufferedReader in;
        try {
                in = new BufferedReader(new FileReader(fileName));

                String line;
                while((line = in.readLine())!=null)
                {
                        String [] words = line.split("[ \t\n]");
                        for (int i=0; i<words.length; i++)
                                if (!list.contains(words[i].trim()))
                                        list.add(words[i].trim());
```

```
                }
                in.close();
        }
        catch (Exception e)
        {
                e.printStackTrace();
        }

        return list.size();
}



private int getNumberLinesOfTextInFile(String fileName)
{
        BufferedReader in;
        int numberLOT = 0;
        try {
                in = new BufferedReader(new FileReader(fileName));

                String line;
                while((line = in.readLine())!=null)
                {
                        if (line.trim().length()>0)
                                numberLOT++;
                }
                in.close();
        }
        catch (Exception e)
        {
                e.printStackTrace();
        }
```

```
                return numberLOT;

        }

}
```

ParsingUI.java

```
package ParsingUserInterface;

import org.eclipse.swt.SWT;

public class ParsingUI extends Shell {
        private Text text;
        public Label lblNewLabel;
        public Button btnNewButton;
        public Button btnOk;
        public Label lblNewLabel_1;
        public Label lblNewLabel_2;
        public Combo combo;
        public static String projectName =null;
        public Button btnRadioButton;
        public Button btnRadioButton_1;
        public Button btnRadioButton_2;
        public Button btnRadioButton_3;
        private Label lblNewLabel_3;
        /**
         * Launch the application.
         * @param args
         */
        public static void main(String args[]) {
                try {
                        Display display = Display.getDefault();
                        ParsingUI shell = new ParsingUI(display,args[0]);
                        shell.open();
                        shell.layout();
                        while (!shell.isDisposed()) {
                                if (!display.readAndDispatch()) {
                                        display.sleep();
                                }
                        }
                } catch (Exception e) {
                        e.printStackTrace();
                }
        }

        /**
         * Create the shell.
         * @param display
         */
        public ParsingUI(Display display, String projectName) {
                super(display, SWT.SHELL_TRIM);
                ParsingUI.projectName = projectName;
                lblNewLabel = new Label(this, SWT.NONE);
                lblNewLabel.setBounds(10, 10, 137, 15);
                lblNewLabel.setText("Enter Number of Topics :");
```

```
text = new Text(this, SWT.BORDER);
text.setBounds(175, 7, 161, 21);

btnNewButton = new Button(this, SWT.NONE);

btnNewButton.setBounds(261, 31, 75, 25);
btnNewButton.setText("Start");

lblNewLabel_1 = new Label(this, SWT.NONE);
lblNewLabel_1.setBounds(10, 75, 137, 15);
lblNewLabel_1.setText("Choose the Author Name");
lblNewLabel_1.setVisible(true);
lblNewLabel_2 = new Label(this, SWT.NONE);
lblNewLabel_2.setBounds(10, 96, 159, 15);
lblNewLabel_2.setText("(for all authors, choose 'ALL')");
lblNewLabel_2.setVisible(true);
combo = new Combo(this, SWT.NONE);
combo.setBounds(175, 93, 161, 23);
combo.setVisible(true);
btnOk = new Button(this, SWT.NONE);


btnOk.setBounds(261, 227, 75, 25);
btnOk.setText("Charts");
btnOk.setVisible(true);

btnRadioButton = new Button(this, SWT.RADIO);
btnRadioButton.setBounds(10, 165, 90, 16);
btnRadioButton.setText("All Text");

btnRadioButton_1 = new Button(this, SWT.RADIO);
btnRadioButton_1.setBounds(10, 187, 90, 16);
btnRadioButton_1.setText("Class Names");

btnRadioButton_2 = new Button(this, SWT.RADIO);
btnRadioButton_2.setBounds(10, 209, 90, 16);
btnRadioButton_2.setText("Commit Texts");

btnRadioButton_3 = new Button(this, SWT.RADIO);
btnRadioButton_3.setBounds(10, 231, 171, 16);
btnRadioButton_3.setText("Changed Source Code");

lblNewLabel_3 = new Label(this, SWT.NONE);
lblNewLabel_3.setBounds(10, 144, 171, 15);
lblNewLabel_3.setText("Choose Source of Mallet");

Button btnNewButton_1 = new Button(this, SWT.NONE);

btnNewButton_1.setBounds(224, 122, 112, 25);
btnNewButton_1.setText("Populate Names");
createContents();

btnNewButton.addSelectionListener(new SelectionAdapter() {
      @Override
```

```
                public void widgetSelected(SelectionEvent e) {

                        String [] args = new String [2];
                        args [0] = text.getText();
                        args [1] = ParsingUI.projectName;
                        String authorsName = Parsing.createCommitFiles(args[1]);
                        String [] authorsNameArray = authorsName.split("\n");
                        Parsing.malletOperations(args[0]);
                        combo.add("ALL");
                        for(int i=0; i<authorsNameArray.length; i++)
                        {

        combo.add(authorsNameArray[i].substring(0,authorsNameArray[i].indexOf(" <")));
                        }

                        lblNewLabel_1.setVisible(true);
                        lblNewLabel_2.setVisible(true);
                        combo.setVisible(true);
                        btnOk.setVisible(true);
                        lblNewLabel.setVisible(false);
                        text.setVisible(false);
                        btnNewButton.setVisible(false);
                }
        });
        btnOk.addSelectionListener(new SelectionAdapter() {
                @Override
                public void widgetSelected(SelectionEvent e) {
                        String authorName = combo.getText();
                        boolean [] radioButtons = new boolean[4];
                        radioButtons[0] = btnRadioButton.getSelection();
                        radioButtons[1] = btnRadioButton_1.getSelection();
                        radioButtons[2] = btnRadioButton_2.getSelection();
                        radioButtons[3] = btnRadioButton_3.getSelection();


                        Parsing.createExcelChart(authorName, radioButtons);
                        boolean fileCreated = false;
                        Desktop dt = Desktop.getDesktop();
                        if(radioButtons[0])
                        {
                                do
                                {
                                        File folder = new
File("C:\\mallet\\workspace\\mallet\\all");
                                  File[] fileList = folder.listFiles();
                                  for(int i = 0; i<fileList.length;i++)
                                  {

        if(fileList[i].getPath().equals("C:\\mallet\\workspace\\mallet\\all\\all.xls")
)
                                                fileCreated = true;
                                  }
                                }
                                while(!fileCreated);
                                try
```

```
                                    {
                                          dt.open(new
File("C:\\mallet\\workspace\\mallet\\all\\all.xls"));
                                    }
                                    catch(IOException ioe)
                                    {

                                    }
                              }
                              if(radioButtons[1])
                              {
                                    try
                                    {
                                          dt.open(new
File("C:\\mallet\\workspace\\mallet\\className\\className.xls"));
                                    }
                                    catch(IOException ioe)
                                    {

                                    }
                              }
                              if(radioButtons[2])
                              {
                                    try
                                    {
                                          dt.open(new
File("C:\\mallet\\workspace\\mallet\\commit\\commit.xls"));
                                    }
                                    catch(IOException ioe)
                                    {

                                    }
                              }
                              if(radioButtons[3])
                              {
                                    try
                                    {
                                          dt.open(new
File("C:\\mallet\\workspace\\mallet\\changedLine\\changedLine.xls"));
                                    }
                                    catch(IOException ioe)
                                    {

                                    }
                              }


                        }
                  });
                  btnNewButton_1.addSelectionListener(new SelectionAdapter() {
                        @Override
                        public void widgetSelected(SelectionEvent e) {
                              String [] authorsName =
Parsing.getAuthorsName(ParsingUI.projectName).split("\n");
                              combo.removeAll();
```

```
                        combo.add("ALL");
                        for(int i=0; i<authorsName.length; i++)
                        {

        combo.add(authorsName[i].substring(0,authorsName[i].indexOf(" <")));
                        }

                }
        });
        }


        /**
         * Create contents of the shell.
         */
        protected void createContents() {
                setText("DevProfiler OPTIONS");
                setSize(400, 300);

        }

        @Override
        protected void checkSubclass() {
                // Disable the check that prevents subclassing of SWT components
        }
}
```

runApplication.java

```
package ParsingUserInterface;

import java.awt.BorderLayout;


import org.eclipse.swt.widgets.Event;


import javax.swing.*;


import java.awt.*;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.awt.event.WindowAdapter;

import java.awt.event.WindowEvent;


import org.eclipse.swt.SWT;

import org.eclipse.swt.widgets.Button;
```

```
import org.eclipse.swt.widgets.Display;
import org.eclipse.swt.widgets.Label;
import org.eclipse.swt.widgets.Shell;
import org.eclipse.swt.widgets.Text;
import org.eclipse.swt.widgets.Listener;
import org.eclipse.core.resources.IProject;
import org.eclipse.jface.viewers.IStructuredSelection;
import org.eclipse.jface.viewers.ISelection;
import org.eclipse.jface.action.IAction;



import javax.swing.*;

import parsing.*;
import ParsingUserInterface.ParsingUI;
import org.eclipse.core.commands.AbstractHandler;
import org.eclipse.core.commands.ExecutionEvent;
import org.eclipse.core.commands.ExecutionException;
import org.eclipse.swt.widgets.Shell;
import org.eclipse.ui.handlers.HandlerUtil;
import org.eclipse.jface.dialogs.MessageDialog;

public class runApplication extends AbstractHandler  {

         public JTextField textField = null;
         public JLabel projectNameLabel = null;
         public JButton button = null;
         public JFrame frame = null;
         public Object execute(ExecutionEvent event) throws ExecutionException
         {

                String projectName =null;
```

```
                Shell shell = HandlerUtil.getActiveShell(event);
                ISelection sel = HandlerUtil.getActiveMenuSelection(event);
                IStructuredSelection selection = (IStructuredSelection) sel;
                Object firstElement = selection.getFirstElement();
                projectName =firstElement.toString();
                if(projectName.contains(" "))
                {
                        projectName = projectName.substring(0,projectName.indexOf(' '));
                }
                if(projectName.contains("/"))
                {
                        projectName = projectName.substring(projectName.indexOf("/")+1);
                }
                String [] args = new String [1] ;
                args[0] = projectName;
                ParsingUI.main(args);

                return null;
        }

}
```

MANIFEST.MF

```
Manifest-Version: 1.0
Bundle-ManifestVersion: 2
Bundle-Name: ParsingUserInterface
Bundle-SymbolicName: ParsingUserInterface;singleton:=true
Bundle-Version: 1.0.0.qualifier
Bundle-RequiredExecutionEnvironment: JavaSE-1.7
Require-Bundle: org.eclipse.jdt.core;bundle-version="3.10.0",
 org.eclipse.core.resources;bundle-version="3.9.1",
 org.eclipse.core.runtime;bundle-version="3.10.0",
 org.eclipse.core.expressions;bundle-version="3.4.600",
 org.eclipse.core.commands;bundle-version="3.6.100",
 org.eclipse.ui
Bundle-ClassPath: lib/parsing.jar,
 lib/aspose-cells-8.2.1.jar,
 lib/bcprov-jdk16-146.jar,
 .
```

Plugin.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<?eclipse version="3.4"?>
<plugin>
   <extension
         point="org.eclipse.ui.commands">
      <command
            defaultHandler="ParsingUserInterface.runApplication"
            id="ParsingUserInterface.runApplication"
            name="name">
      </command>
   </extension>
   <extension
         point="org.eclipse.ui.menus">
      <menuContribution
            allPopups="false"
            locationURI="popup:org.eclipse.jdt.ui.PackageExplorer">
         <command
               commandId="ParsingUserInterface.runApplication"
               label="DevProfiler"
               style="push">

         </command>
      </menuContribution>
   </extension>

</plugin>
```