README: How to experiment the artifacts

Phuong T. Nguyen, Juri Di Rocco, Davide Di Ruscio[1], Lina Ochoa, Thomas Degueule[2], and Massimiliano Di Penta[3]

[1] *Università degli studi dell'Aquila, L'Aquila, Italy*
{phuong.nguyen, juri.dirocco, davide.diruscio}@univaq.it
[2] *Centrum Wiskunde & Informatica, Amsterdam, Netherlands*
{lina.ochoa, thomas.degueule}@cwi.nl
[3] *Università degli Studi del Sannio, Benevento, Italy*
dipenta@unisannio.it

ABSTRACT

This document provides exhaustive instructions to get the datasets, to obtain, install, and execute the tools for replicating the experiments described in the paper entitled *"FOCUS: A Recommender System for Mining API Function Calls and Usage Patterns"* accepted in the Technical Track of ICSE 2019.

1 Introduction

FOCUS is a recommender system that assists software developers during the development phase by suggesting highly relevant API function calls (invocations) and patterns [5]. Given a developer who is working on a method declaration d_a of a software project p, FOCUS provides her with additional method invocations to include and complete d_a.

To pave the way for further explanations, in this section we give a short introduction on the functionalities of FOCUS. Figure 1 depicts an example of how FOCUS recommends API function calls. The declaration `handleEmplyBatch()`[1] consists of a number of invocations to perform a specific task. However, at the time of consideration, only some invocations are present, i.e., the developer has just finished from the beginning until `client.connect()` and all the invocations within the frame are not yet written. It is expected that FOCUS is capable of recommending these missing invocations. To this end, the system has been designed so as to provide two types of recommendation. First, it recommends ***a list of invocations*** related to d_a. Second, it suggests ***real code snippets*** that are useful for completing the declaration.

[1] `https://github.com/hnfgns/incubator-drill/blob/master/exec/java-exec/src/test/java/org/apache/drill/exec/physical/impl/mergereceiver/TestMergingReceiver.java`

```
public void handleEmptyBatch() throws Exception {
    final RemoteServiceSet serviceSet = RemoteServiceSet.getLocalServiceSet();
    try (final Drillbit bit1 = new Drillbit(CONFIG, serviceSet);
        final Drillbit bit2 = new Drillbit(CONFIG, serviceSet);
        final DrillClient client = new DrillClient(CONFIG, serviceSet.getCoordinator());) {
      bit1.run();
      bit2.run();
      client.connect();
      final List<QueryDataBatch> results =
             client.runQuery(org.apache.drill.exec.proto.UserBitShared.QueryType.PHYSICAL,
              Files.toString(FileUtils.getResourceAsFile("/mergerecv/empty_batch.json"),
                  Charsets.UTF_8));
      int count = 0;
      final RecordBatchLoader batchLoader = new RecordBatchLoader(client.getAllocator());
      // print the results
      for (final QueryDataBatch b : results) {
        final QueryData queryData = b.getHeader();
        batchLoader.load(queryData.getDef(), b.getData()); // loaded but not used, for testing
        count += queryData.getRowCount();
        b.release();
        batchLoader.clear();
      }
      assertEquals(100000, count);
    }
  }
```

Figure 1: An example: The code marked within the frame is not present and should be recommended.

In the accepted paper [5], we fulfilled the first type of recommendation, i.e., recommending a list function calls given a specific declaration. We leave the second type of recommendation as our future work, i.e., retrieving real code snippets to be shown directly in the IDE being used. Thus, in the experiments presented in the paper as well as in this artifact submission, we attempt to evaluate FOCUS with respect to the first type of recommendation by using different amount of input data as well as different datasets. Furthermore, we also compare the performance of FOCUS with that of PAM [3]. In this sense, the experiments are conducted to answer the following research questions:

- **RQ**$_1$: To what extent is FOCUS able to provide accurate and complete recommendations?
- **RQ**$_2$: What are the timing performances of FOCUS in building its models and in providing recommendations?
- **RQ**$_3$: How does FOCUS perform compared with PAM?

The artifacts include the FOCUS tool written in Java and four datasets as given in Table 1.[2] The datasets have been collected from the Software-Heritage archive[3] and the Maven Central repository[4]. The PAM tool can be downloaded directly from its authors' repository [2].

[2]The datasets' aliases are consistent with those presented in the paper

[3]`https://www.softwareheritage.org/`

[4]`https://mvnrepository.com/`

Nr.	Alias	Folder	Descriptions
1	SH_L	`SH_L`	This dataset contains 610 GitHub Java projects extracted from the SoftwareHeritage archive
2	SH_S	`SH_S`	The 200 smallest (in size) projects extracted from SH_L
3	MV_L	`MV_L`	A set of $3,600$ JARs extracted from Maven Central
4	MV_S	`MV_S`	A subset of MV_L and consists of $1,600$ JARs where, for each project, only one version is kept

Table 1: The datasets used in the paper

2 System Requirements

Table 2 specifies the hardware and software requirements that a testing system needs to meet in order to be eligible for the execution of the artifacts.

Name	Requirements
RAM	$\geq$ 8GB
Operating System	A modern Linux system[1]
Java JRE	$\geq$ Java 8
Apache Maven	$\geq$ Maven 3.0
Python	$\geq$ 2.7.12
Rascal[2]	$\geq$ 10.0
Git	$\geq$ 2.0

Table 2: Hardware and software requirements

[1] We developed and tested FOCUS on different Linux systems, although it should work without any issue on any operating system. In the instructions, we assume a Linux system and basic command line skills. [2] The use of Rascal is optional.

In our case, the testing platform is a PC with Linux 4.20.3, Intel Core i7-6700HQ CPU @ 2.60GHz and 16GB of RAM.

3 The validation process

This section gives a summary on the validation process applied in the paper. In Section 3.1, we briefly introduce the k-fold cross-validation technique. Afterwards, in Section 3.2 we recall the experimental configurations.

3.1 K-fold cross validation

We perform evaluation on the datasets presented in Table 1, i.e., SH_L, SH_S, MV_L, and MV_S using the process illustrated in Figure 2.

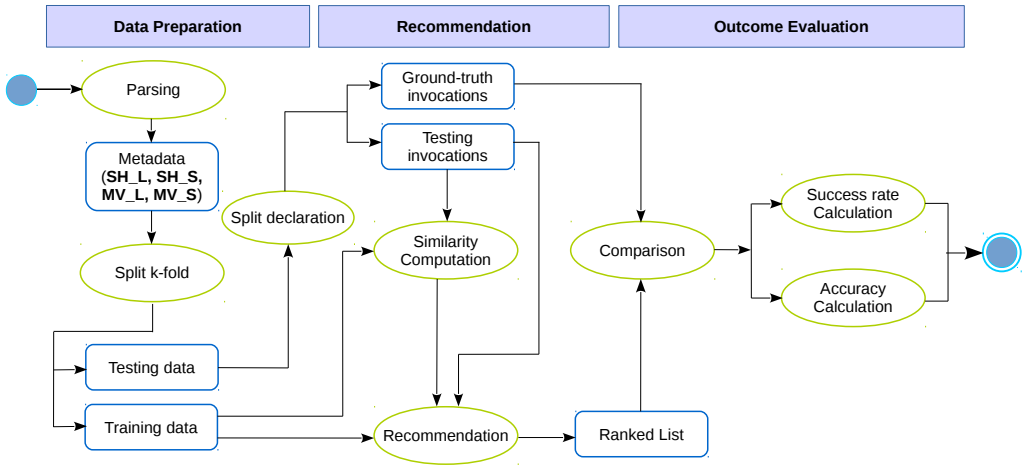

Figure 2: The evaluation process

For an experiment, a dataset, e.g., MV_L is split into two independent parts, namely a *training set* and a *testing set* (`Split k-fold`). In practice, the training set represents the OSS projects that are available as background data. Meanwhile, each item in the testing set represents the project being developed, or *the active project*. Our evaluation mimics a real development scheme: *the recommender system should produce recommendations for a project based on the data available from a set of existing projects*.

We opt for *k-fold cross validation* [4]: A dataset with n projects is divided into k equal parts, so-called *folds*. For each validation round, one fold is used as testing data and the remaining $k-1$ folds are used as training data. To address $\mathbf{RQ}_1$, ten-fold cross validation, i.e., $k = 10$ is applied on the MV_L and MV_S datasets. To address $\mathbf{RQ}_2$ and $\mathbf{RQ}_3$, leave-one-out cross validation, i.e., $k = n$ is applied on the SH_L and SH_S datasets.

3.2 Configurations

To study if FOCUS is applicable in real-world settings, we simulate different stages of a development process. To this end, some parts of p are removed to mimic an actual development. Given a testing project p, the total number of declarations it contains is called Δ. However, only δ declarations ($\delta < \Delta$) are used as input for recommendation and the rest is discarded. In practice, this corresponds to the situation when the developer already finished δ declarations, and she is now working on the *active declaration* d_a. For d_a, there are Π invocations, however to simulate the situation as illustrated in Figure 1, only the first π invocations ($\pi < \Pi$) are selected as testing and the rest is removed and saved as ground-truth data for future comparison (`Split declaration`). Figure 3 gives an intuition on how the extraction of a testing project is done.

The two parameters δ, π are used to simulate different development phases. In particular, we consider the configurations as given in Table 3.2.

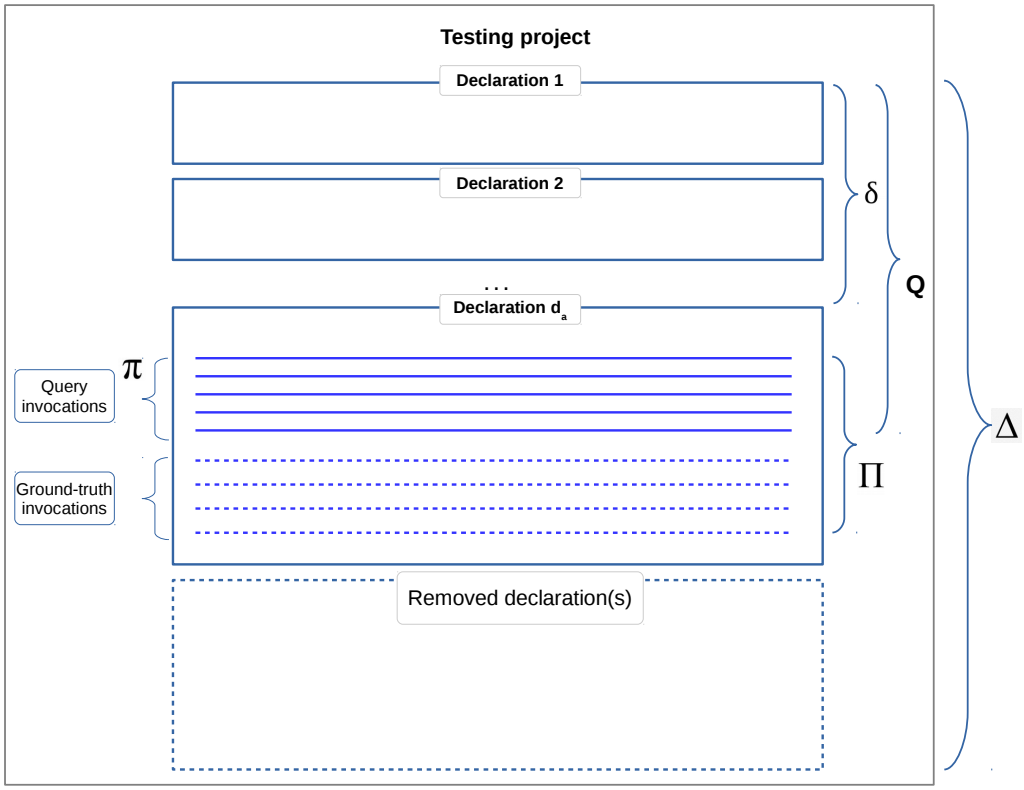

Figure 3: Extracting data for a testing project

4 Download the artifacts

On a Linux system, open a terminal and execute the following command to download all artifacts from a single GitHub repository:

```
$ git clone https://github.com/crossminer/FOCUS
```

This will create a new directory and we call it `<FOCUS_root>` in the scope of this presentation. An example of such a folder in Linux is `/home/admin/Desktop/FOCUS/`. The root directory has the following structure:

- The `tools` directory contains the implementation of the different tools we developed:
  - `Focus`: the Java implementation of FOCUS.
  - `FocusRascal`: a set of tools written in Rascal [1] that are used to transform raw Java source and binary code into FOCUS-processable and PAM-processable data.
  - `PAM`: utilities supporting the evaluation of PAM.
- The `dataset` directory contains the datasets described in the paper that we use to evaluate FOCUS:

Conf.	δ	π	Description
`C1.1`	$\Delta/2-1$	1	Almost the first half of the declarations ($\Delta/2-1$) is used as testing data and the second half is removed. The last declaration of the first half is selected as the active declaration d_a. For d_a, only the *first* invocation is provided as a query, and the rest is used as ground-truth data which we call `GT(p)`. This configuration mimics a scenario where the developer is at an early stage of the development process and, therefore, only limited context data is available to feed the recommendation engine
`C1.2`	$\Delta/2-1$	4	Similarly to `C1.1`, almost the first half of the declarations, i.e., $\Delta/2-1$ is kept and the second half is discarded. d_a is the last declaration of the first half of declarations. For d_a, the first *four* invocations are provided as query, and the rest is used as `GT(p)`
`C2.1`	$\Delta-1$	1	The last method declaration is selected as testing, i.e., d_a and all the remaining declarations are used as training data ($\Delta-1$). In d_a, the *first* invocation is kept and all the others are taken out as ground-truth data `GT(p)`. This represents the stage where the developer almost finished implementing p
`C2.2`	$\Delta-1$	4	Similar to `C2.1`, d_a is selected as the last method declaration, and all the remaining declarations are used as training data ($\Delta-1$). The only difference with `C2.1` is that in d_a, the first *four* invocations are used as query and all the remaining ones are used as ground-truth data `GT(p)`

Table 3: Experimental configurations.

- `jars`: There are 3,600 JAR files extracted from Maven Central.
- `SH_L`: metadata of the SH_L dataset.
- `SH_S`: metadata of the SH_S dataset.
- `MV_L`: metadata of the MV_L dataset.
- `MH_S`: metadata of the MV_S dataset.

More information about the artifacts can be found in `<FOCUS_root>/README.md`.

5 Execute the experiments

This section describes how to run FOCUS and PAM on the datasets using the configurations to reproduce the results presented in the paper. Please note that more information is available in various README.md files of the GitHub repository, e.g., `<FOCUS_root>/README.md`, `<FOCUS_root>/tools/Focus/README.md`.

5.1 Running FOCUS

Three parameters must be fed to FOCUS: the dataset (as shown Table 1), the configuration (as shown in Table 3.2), and the cross-validation method (10-fold or leave-one-out). These parameters can be specified in the file `<FOCUS_root>/tools/Focus/evaluation.properties`. The default `evaluation.properties` file runs 10-fold cross-validation on the SH_S dataset using configuration `C1.1`, and it is written as follows:

```
# Dataset directory (SH_L, SH_S, MV_L, MV_S)
sourceDirectory=../../dataset/SH_S/

# Configuration (C1.1, C1.2, C2.1, C2.2)
configuration=C1.1

# Validation type (ten-fold, leave-one-out)
validation=ten-fold
```

To run FOCUS using the `evaluation.properties` file, navigate to the directory containing the FOCUS's implementation and execute it through Maven as given below:

```
$ cd <FOCUS_root>/tools/Focus
$ mvn compile exec:java -Dexec.mainClass=org.focus.Runner
```

You need to replace `<FOCUS_root>` with your actual root directory.

The final results of the evaluation including success rate, precision, and recall for different cut-off values $N = \{1, 5, 10, 15, 20\}$ are printed directly in the console. Intermediate results are stored in the evaluation folder of the corresponding dataset's directory. For instance, the results for the SH_L dataset will be saved in `<FOCUS_root>/dataset/SH_L/evaluation/`. In this folder, there are ten sub-folders, i.e., `round1`, `round2`,..., `round10`. Each of them stores the evaluation results of one fold using the following folders:

- `APIUsagePatterns`: the folder stores the recommended real code snippets.
- `GroundTruth`: it contains all the invocations extracted as ground-truth data.
- `Recommendations`: the list of recommended invocations for each project is stored in this folder.
- `Similarities`: the folder contains the similarity scores for each project.
- `TestingInvocations`: the folder stores the invocations used as query.

5.2 Running PAM

To answer $\mathbf{RQ}_2$ and $\mathbf{RQ}_3$, it is necessary to run PAM on the SH_S dataset using leave-one-out cross validation. We already parsed the same data used by FOCUS to provide as input for PAM. All you have to do to get the recommendation outcomes by PAM is to run the following commands:

```
$ cd <FOCUS_root>/tools/Pam
$ mvn clean compile exec:java -Dexec.mainClass=org.focus.Runner -
    Dexec.args=confs/shs12.properties
```

The results are stored using a similar directory structure used by FOCUS as given below:

- `GroundTruth`: the folder contains the ground-truth invocations.
- `Recommendations`: the list of recommended invocations for each project is stored in this folder.
- `TestingInvocations`: the folder stores the invocations used as query.

> **NOTE**: The following instructions are dedicated for those who want to run PAM from the beginning. Thus, the execution presented in the rest of this section is purely optional.

For your convenience, we provide a Python script named `parsingPAM.py` to convert the input data used by FOCUS to be fed to PAM. Run the following commands:[5]

```
$ cd <FOCUS_root>/tools/PAM
$ python3 parsingPAM.py <FOCUS_root>/dataset/SH_S/ <FOCUS_root>/
    dataset/PAM/SH_S/
```

Then execute PAM on the given input data as follows:

```
$ git clone git@github.com:mast-group/api-mining.git <PAM_root>
$ cd <PAM_root>
$ mvn package
$ for f in <FOCUS_root>/dataset/PAM/SH_S/ *; do java -jar api-
    mining/target/api-mining-1.0.jar -f $f; done
```

where `<PAM_root>` is the directory where you have cloned the repository. In our case, we set it to `/home/admin/Desktop/PAM/`.

You can find more information about the execution of PAM at `<FOCUS_root>/tools/PAM/README.md`.

[5]The execution of the commands necessitates python3 which is available at `https://www.python.org/download/releases/3.0/`

5.3 Running FocusRascal

This section describes how to run the `Parsing` phase as illustrated in Figure 2 using Rascal to generate metadata. The use of Rascal is described in Section III.A of the accepted paper [5].

> **NOTE**: For the sake of reproducibility, we report all the related activities to generate metadata using Rascal. However, we completely parsed the metadata needed as input for both FOCUS and PAM at your disposal. The tasks described in this section require a long computation time as well as knowledge about Rascal, and we strongly suggest using the metadata as it is. Therefore, it is advisable not to perform the executions presented in Section 5.3.

The JAR libraries involved in this experiment can be found at `FOCUS/dataset/jars`. A snapshot of the related GitHub repositories is available online.[6]

Follow the instructions available at `<FOCUS_root>/tools/FocusRascal` to install and configure the Rascal environment. Afterwards, go to the Rascal Console and import the main module:

```
import focus::corpus::ExtractMetadata;
```

Since the MV_L and MV_S datasets have been extracted from byte code, whereas the SH_L and SH_S datasets are mined directly from source code, the `ExtractMetadata.rsc` module provides two different ways to parse the datasets. To extract metadata from source code, `processGithubRepos()` is invoked using the following Rascal command. By default, it will read the GitHub repositories metadata stored in `FocusRascal/config/github-repos\\.properties` and output the results in the `FocusRascal/data/m3/java-projects/` directory.[7]

```
processGithubRepos();
```

The other function, `processJARs(loc directory)` is used to create the M3 models from the JAR files contained in the specified directory. By default, it outputs the results in `FocusRascal/data/m3/jar-projects/`. The following command, for instance, parses all the JARs joined in this artifact submission (replacing `path/to/FOCUS` with `<FOCUS_root>`):

```
processJARs(file:///path/to/FOCUS/dataset/jars);
```

[6] `https://drive.google.com/open?id=1hQkh85XY4c0sh788G9g8B0yi5stJxSwc`

[7] More information can be found at `FOCUS/tools/FocusRascal`

6 Results

The following listing gives an example of how the experimental results may look like in practice. These are the results including success rate, precision and recall obtained by performing FOCUS on the MV_L dataset using Configuration `C1.1`. For instance, `successRate@1 = 73.30556` means that the success rate for the cut-off value $N = 1$ is approximate 73.30%. An increase in N will lead to an improvement of the evaluation scores. More information about the results is available at `<FOCUS_root>/tools/Focus`.

```
FOCUS: A Context-Aware Recommender System!
...
### 10-FOLDS RESULTS ###
successRate@1 = 73.30556
precision@1  = 0.73305553
recall@1     = 0.06737014
successRate@5 = 82.66667
precision@5  = 0.6486667
recall@5     = 0.2956125
successRate@10 = 86.69446
precision@10  = 0.5511667
recall@10     = 0.49201664
...
```

Troubleshooting

If you encounter any problem during the evaluation, please do not hesitate to contact us through one of the email addresses given above.

References

[1] B. Basten, M. Hills, P. Klint, D. Landman, A. Shahi, M. J. Steindorfer, and J. J. Vinju. M3: A General Model for Code Analytics in Rascal. In *1st International Workshop on Software Analytics*, pages 25–28, Piscataway, 2015. IEEE.

[2] J. Fowkes and C. Sutton. PAM: Probabilistic API Miner. `https://github.com/mast-group/api-mining`. last access 24.08.2018.

[3] J. Fowkes and C. Sutton. Parameter-free Probabilistic API Mining Across GitHub. In *24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 254–265, New York, 2016. ACM.

[4] R. Kohavi. A Study of Cross-validation and Bootstrap for Accuracy Estimation and Model Selection. In *14th International Joint Conference*

on Artificial Intelligence, pages 1137–1143, San Francisco, 1995. Morgan Kaufmann Publishers Inc.

[5] P. T. Nguyen, J. Di Rocco, D. Di Ruscio, L. Ochoa, T. Degueule, and M. Di Penta. FOCUS: A Recommender System for Mining API Function Calls and Usage Patterns - to appear. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, May 2019.