Project Number 732223

D2.8 API Analysis Components

Version 1.1
30 June 2019
Final

Public Distribution

Centrum Wiskunde & Informatica (CWI)

Project Partners: Athens University of Economics & Business, Bitergia, Castalia Solutions, Centrum Wiskunde & Informatica, Eclipse Foundation Europe, Edge Hill University, FrontEndART, OW2, SOFTEAM, The Open Group, University of L’Aquila, University of York, Unparallel Innovation

Every effort has been made to ensure that all statements and information contained herein are accurate, however the CROSSMINER Project Partners accept no liability for any error or omission in the same.

© 2019 Copyright in this document remains vested in the CROSSMINER Project Partners.
# Project Partner Contact Information

<table>
<thead>
<tr>
<th>Athens University of Economics &amp; Business</th>
<th>Bitergia</th>
</tr>
</thead>
<tbody>
<tr>
<td>Diomidis Spinellis</td>
<td>José Manrique Lopez de la Fuente</td>
</tr>
<tr>
<td>Patision 76</td>
<td>Calle Navarra 5, 4D</td>
</tr>
<tr>
<td>104-34 Athens</td>
<td>28921 Alcorcón Madrid</td>
</tr>
<tr>
<td>Greece</td>
<td>Spain</td>
</tr>
<tr>
<td>Tel: +30 210 820 3621</td>
<td>Tel: +34 6 999 279 58</td>
</tr>
<tr>
<td>E-mail: <a href="mailto:dds@aueb.gr">dds@aueb.gr</a></td>
<td>E-mail: <a href="mailto:jsmanrique@bitergia.com">jsmanrique@bitergia.com</a></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Castalia Solutions</th>
<th>Centrum Wiskunde &amp; Informatica</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boris Baldassari</td>
<td>Jurgen J. Vinju</td>
</tr>
<tr>
<td>10 Rue de Penthèvre</td>
<td>Science Park 123</td>
</tr>
<tr>
<td>75008 Paris</td>
<td>1098 XG Amsterdam</td>
</tr>
<tr>
<td>France</td>
<td>Netherlands</td>
</tr>
<tr>
<td>Tel: +33 6 48 03 82 89</td>
<td>Tel: +31 20 592 4102</td>
</tr>
<tr>
<td>E-mail: <a href="mailto:boris.baldassari@castalia.solutions">boris.baldassari@castalia.solutions</a></td>
<td>E-mail: <a href="mailto:jurgen.vinju@cwi.nl">jurgen.vinju@cwi.nl</a></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Eclipse Foundation Europe</th>
<th>Edge Hill University</th>
</tr>
</thead>
<tbody>
<tr>
<td>Philippe Krief</td>
<td>Yannis Korkontzelos</td>
</tr>
<tr>
<td>Annastrasse 46</td>
<td>St Helens Road</td>
</tr>
<tr>
<td>64673 Zwingenberg</td>
<td>Ormskirk L39 4QP</td>
</tr>
<tr>
<td>Germany</td>
<td>United Kingdom</td>
</tr>
<tr>
<td>Tel: +33 62 101 0681</td>
<td>Tel: +44 1695 654393</td>
</tr>
<tr>
<td>E-mail: <a href="mailto:philippe.krief@eclipse.org">philippe.krief@eclipse.org</a></td>
<td>E-mail: <a href="mailto:yannis.korkontzelos@edgehill.ac.uk">yannis.korkontzelos@edgehill.ac.uk</a></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>FrontEndART</th>
<th>OW2 Consortium</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rudolf Ferenc</td>
<td>Cedric Thomas</td>
</tr>
<tr>
<td>Zászló u. 3 I/5</td>
<td>114 Boulevard Haussmann</td>
</tr>
<tr>
<td>H-6722 Szeged</td>
<td>75008 Paris</td>
</tr>
<tr>
<td>Hungary</td>
<td>France</td>
</tr>
<tr>
<td>Tel: +36 62 319 372</td>
<td>Tel: +33 6 45 81 62 02</td>
</tr>
<tr>
<td>E-mail: <a href="mailto:ferenc@frontendart.com">ferenc@frontendart.com</a></td>
<td>E-mail: <a href="mailto:cedric.thomas@ow2.org">cedric.thomas@ow2.org</a></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SOFTEAM</th>
<th>The Open Group</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alessandra Bagnato</td>
<td>Scott Hansen</td>
</tr>
<tr>
<td>21 Avenue Victor Hugo</td>
<td>Rond Point Schuman 6, 5th Floor</td>
</tr>
<tr>
<td>75016 Paris</td>
<td>1040 Brussels</td>
</tr>
<tr>
<td>France</td>
<td>Belgium</td>
</tr>
<tr>
<td>Tel: +33 1 30 12 16 60</td>
<td>Tel: +32 2 675 1136</td>
</tr>
<tr>
<td>E-mail: <a href="mailto:alessandra.bagnato@softeam.fr">alessandra.bagnato@softeam.fr</a></td>
<td>E-mail: <a href="mailto:s.hansen@opengroup.org">s.hansen@opengroup.org</a></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>University of L’Aquila</th>
<th>University of York</th>
</tr>
</thead>
<tbody>
<tr>
<td>Davide Di Ruscio</td>
<td>Dimitris Kolovos</td>
</tr>
<tr>
<td>Piazza Vincenzo Rivera 1</td>
<td>Deramore Lane</td>
</tr>
<tr>
<td>67100 L’Aquila</td>
<td>York YO10 5GH</td>
</tr>
<tr>
<td>Italy</td>
<td>United Kingdom</td>
</tr>
<tr>
<td>Tel: +39 0862 433735</td>
<td>Tel: +44 1904 325167</td>
</tr>
<tr>
<td>E-mail: <a href="mailto:davide.diruscio@univaq.it">davide.diruscio@univaq.it</a></td>
<td>E-mail: <a href="mailto:dimitris.kolovos@york.ac.uk">dimitris.kolovos@york.ac.uk</a></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Unparallel Innovation</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Bruno Almeida</td>
<td></td>
</tr>
<tr>
<td>Rua das Lendas Algarvias, Lote 123</td>
<td></td>
</tr>
<tr>
<td>8500-794 Portimão</td>
<td></td>
</tr>
<tr>
<td>Portugal</td>
<td></td>
</tr>
<tr>
<td>Tel: +351 282 48502</td>
<td></td>
</tr>
<tr>
<td>E-mail: <a href="mailto:bruno.almeida@unparallel.pt">bruno.almeida@unparallel.pt</a></td>
<td></td>
</tr>
</tbody>
</table>
Document Control

<table>
<thead>
<tr>
<th>Version</th>
<th>Status</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>Initial outline</td>
<td>3 June 2019</td>
</tr>
<tr>
<td>0.2</td>
<td>Full update</td>
<td>18 June 2019</td>
</tr>
<tr>
<td>1.0</td>
<td>Release for internal review</td>
<td>25 June 2019</td>
</tr>
<tr>
<td>1.1</td>
<td>Final version</td>
<td>30 June 2019</td>
</tr>
</tbody>
</table>
# Table of Contents

## I  FOCUS: a Recommender System for Mining API Function Calls and Usage Patterns

1  Introduction

2  Using FOCUS in standalone mode

3  Integration with the CROSSMINER platform

## II  MARACAS: a Framework for API Evolution Analysis and Client Code Migration

4  Introduction

5  User Scenarios

6  Getting Started

7  Additional Features

8  Integration with the CROSSMINER platform
Executive Summary

This deliverable reports on the (open-source) software output of Task 2.4.

Task 2.4: API Analysis  This task will produce analyses to identify API and API usage information. The goal is to learn from existing projects about typical usage to be able to concretely advise software engineers about the use of open-source software components. The resulting analyses should be amenable to bespoke extensions.

The output of this task mainly consists of two software artifacts, together with their corresponding integration as components of the CROSSMINER platform, Knowledge Base, and IDE:

- **FOCUS**, a context-aware collaborative-filtering recommendation system that exploits cross-relationships between software projects to suggest API method calls and API usage patterns that assist developers in learning and using APIs; FOCUS is meant to be integrated with the CROSSMINER IDE to provide developers with recommendations tailored to the particular context they are faced with. FOCUS is available at [https://github.com/crossminer/focus](https://github.com/crossminer/focus)

- **MARACAS**, a source code and bytecode analysis framework that automatically analyses changes in the APIs of software projects to assist developers in the migration of their code when the APIs they rely on evolve; MARACAS is meant to be integrated in both the CROSSMINER platform and dashboards (as a set of metrics allowing to understand how OSS projects evolve) and the CROSSMINER IDE (to provide recommendations regarding API migration directly while working on a software project). MARACAS is available at [https://github.com/crossminer/maracas](https://github.com/crossminer/maracas)

In this deliverable, we focus on the user documentation, developer documentation, and integration of FOCUS and MARACAS. We refer the reader to the companion deliverable D2.7 – Framework and API Analysis – Final Report for a detailed analysis of the research questions we address, our scientific contributions, the evaluation of the tools, and the project requirements they fulfill.

Concretely speaking, FOCUS and MARACAS are two complementary parts of the new API Miner Component of the CROSSMINER platform described in D8.1 – Architecture Specification of the Integrated Platform (cf. Section 3.3.2 and Figure 24).

This deliverable is organized as follows:

- In Part I, we present FOCUS and its integration in the CROSSMINER IDE;
- In Part II, we present MARACAS and its integration in the CROSSMINER platform, dashboards, and IDE.

All the components presented here, along with the scientific contributions introduced in D2.7 were created from scratch specifically for CROSSMINER and were not part of the previous OSSMETER project and platform.
Part I

FOCUS: a Recommender System for Mining API Function Calls and Usage Patterns

1 Introduction

FOCUS is a context-aware collaborative-filtering recommendation system that exploits cross-relationships between software projects to suggest API method calls and usage patterns [3]. In this section, we show how FOCUS can be manipulated as a standalone project, and how it is integrated with other components of the CROSSMINER platform.

FOCUS has already been assessed by a committee of software experts as part of the Artifact Evaluation process of the 41st International Conference on Software Engineering and received seals of approvals regarding its availability and reusability.¹

2 Using FOCUS in standalone mode

The FOCUS tool is publicly available on our shared GitHub organization at the URL https://github.com/crossminer/focus. The precise version of FOCUS that we used for the evaluation in [3], along with the corresponding datasets, are also available with a dedicated DOI on the public archive Zenodo [2].

FOCUS relies on two main sub-components: FOCUS itself, which implements the context-aware recommendation system, and FocusRascal which implements the static Java source code analyzers that provide the necessary information about method declarations and method invocations to FOCUS.

2.1 FocusRascal

FocusRascal is a Rascal project used to mine Java projects (in source code or binary JAR form) to extract method invocations and declarations information used by the FOCUS recommender system. FocusRascal extracts $M^3$ models from source code and JARs, which are then turned into FOCUS-compatible models.

**Installation**  FocusRascal can be implemented following these steps:

1. Install an appropriate version of Eclipse RCP and RAP according to your platform/distribution from this URL:

2. In Eclipse, navigate to Help -> Install New Software... and install Rascal using the following URL:
   https://update.rascal-mpl.org/unstable/

3. Restart Eclipse to complete the installation.

4. Clone locally the ‘rascal-java-build-manager’ project which is a required dependency of FocusRascal and import it into Eclipse: https://github.com/cwi-swat/rascal-java-build-manager

5. In Eclipse, import the two projects ‘FocusRascal’ and ‘rascal-java-build-manager’ in the workspace

**Configuration**  FocusRascal must be configured with a path to the local installation of Apache Maven and paths to the projects or datasets from which method declarations and invocations should be extracted. These parameters can be configured in the file FocusRascal/src/focus/corpus/Configuration.rsc:

```rascal
// Pointer to mvn executable, used to build the classpath of Maven projects
loc mavenLocation = |file:///usr/bin/mvn|;

// Where the GitHub repositories are stored and/or cloned
loc datasetLocation = |file:///FOCUS_ROOT/tools/FocusRascal/dataset/|;

// Where the M3 and FOCUS models are stored
loc javaM3sLocation = |project://FocusRascal/data/m3/java-projects|;

// Where the FOCUS models built from JARs are stored
loc jarsM3sLocation = |project://FocusRascal/data/m3/jar-projects|;

// Where information about the GitHub projects to analyze is stored
loc githubConfigFile = |project://FocusRascal/config/github-repos.properties|;
```

**Quick Start**  All manipulations must be done in a running Eclipse workbench:

1. First, open the Rascal perspective: Window -> Perspective -> Open Perspective -> Rascal

2. Right-click on the FocusRascal project and select Rascal Console, this opens a new Rascal console in the lower part of the IDE

3. In the console, import the ExtractMetadata module:

   ```rascal
   > import focus::corpus::ExtractMetadata;
   ```

   From here, the two main functions to invoke from the console are:
• processGithubRepos(), which reads the GitHub repositories stored in FocusRascal/config/github-repos.properties, clone all repositories, build their M³ models, and write the output FOCUS files in FocusRascal/data/m3/java-projects/. By default, the github-repos.properties file points to the 17 libraries depicted in Table 1 of [3]. This process takes a long time to complete. To use it, simply type in the console:

```java
> processGithubRepos();
```

• processJARs(loc directory) which takes as input the path to a directory containing a list of JARs, and writes as output the FOCUS models of those JARs in FocusRascal/data/m3/jar-projects/. For instance, to parse, analyse, and compute the M³ and FOCUS models of the 3,600 JARs extracted from Maven Central (dataset MV in [3]), type the following in the Rascal console (replacing /path/to/FOCUS with the path to your clone of the FOCUS repository): 

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

### 2.2 FOCUS

The core FOCUS component exploits the meta-data extracted by FocusRascal to implement the recommender system.

**Requirements**  
FOCUS requires the following dependencies:

- Apache Maven >= 3.0
- Java >= 1.8

**Running the tool**  
To start the evaluation of FOCUS using the default evaluation.properties file (see below) which analyses the datasets used in [3], run the following command:

```
mvn clean compile exec:java -Dexec.mainClass=org.focus.Runner
```

To use a different .properties file, use the -Dexec.args argument:

```
mvn clean compile exec:java -Dexec.mainClass=org.focus.Runner -Dexec.args=confs/shs12.properties
```

Results (success rate, precision, recall) for different cut-off values are displayed in the console directly as follows:

Intermediate results for every fold (recommended invocations, groundtruth invocations, usage patterns, etc.) are stored in the evaluation folder of the corresponding dataset (e.g., ../../dataset/SH_S/evaluation/).

Note that the evaluation might take a considerable amount of time and resources. The table below gives reference time for 10-fold cross-validation on a Linux 4.20.3 with Intel Core i7-6700HQ CPU @ 2.60GHz and 16GB of RAM.
FOCUS: A Context-Aware Recommender System!
Running ten-fold cross validation on ../../dataset/MV_L/ with configuration C1.1
Fold [9/10]: SimilarityCalculator took 454s
Fold [8/10]: SimilarityCalculator took 455s
Fold [2/10]: SimilarityCalculator took 466s
Fold [7/10]: SimilarityCalculator took 467s
Fold [9/10]: ContextAwareRecommendation took 17s
Fold [8/10]: ContextAwareRecommendation took 17s
Fold [0/10]: SimilarityCalculator took 476s
Fold [1/10]: SimilarityCalculator took 476s
Fold [6/10]: SimilarityCalculator took 477s
Fold [4/10]: SimilarityCalculator took 477s
Fold [7/10]: ContextAwareRecommendation took 13s
Fold [2/10]: ContextAwareRecommendation took 15s
Fold [1/10]: ContextAwareRecommendation took 7s
Fold [0/10]: ContextAwareRecommendation took 9s
Fold [6/10]: ContextAwareRecommendation took 9s
Fold [4/10]: ContextAwareRecommendation took 11s
Fold [3/10]: SimilarityCalculator took 231s
Fold [5/10]: SimilarityCalculator took 234s
Fold [3/10]: ContextAwareRecommendation took 6s
Fold [5/10]: ContextAwareRecommendation took 9s
### 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
successRate@15 = 88.22222
precision@15 = 0.4440371
recall@15    = 0.57449764
successRate@20 = 89.13888
precision@20 = 0.3645417
recall@20    = 0.6169203
10-fold took 716s
### Dataset Configuration Time (seconds)

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Configuration</th>
<th>Time (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SH_S</td>
<td>C1.1</td>
<td>4</td>
</tr>
<tr>
<td>SH_S</td>
<td>C1.2</td>
<td>4</td>
</tr>
<tr>
<td>SH_S</td>
<td>C2.1</td>
<td>4</td>
</tr>
<tr>
<td>SH_S</td>
<td>C2.2</td>
<td>5</td>
</tr>
<tr>
<td>SH_L</td>
<td>C1.1</td>
<td>314</td>
</tr>
<tr>
<td>SH_L</td>
<td>C1.2</td>
<td>312</td>
</tr>
<tr>
<td>SH_L</td>
<td>C2.1</td>
<td>298</td>
</tr>
<tr>
<td>SH_L</td>
<td>C2.2</td>
<td>345</td>
</tr>
<tr>
<td>MV_S</td>
<td>C1.1</td>
<td>135</td>
</tr>
<tr>
<td>MV_S</td>
<td>C1.2</td>
<td>124</td>
</tr>
<tr>
<td>MV_S</td>
<td>C2.1</td>
<td>139</td>
</tr>
<tr>
<td>MV_S</td>
<td>C2.2</td>
<td>135</td>
</tr>
<tr>
<td>MV_L</td>
<td>C1.1</td>
<td>716</td>
</tr>
<tr>
<td>MV_L</td>
<td>C1.2</td>
<td>732</td>
</tr>
<tr>
<td>MV_L</td>
<td>C2.1</td>
<td>741</td>
</tr>
<tr>
<td>MV_L</td>
<td>C2.2</td>
<td>768</td>
</tr>
</tbody>
</table>

**The evaluation.properties file**  The evaluation of FOCUS is configured with a .properties file that specifies (i) the dataset (ii) the configuration and (iii) the validation technique to be used. For instance, the default evaluation.properties runs 10-fold cross-validation on SH_S using the C1.1 configuration:

```properties
# 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
```

This file can be edited to point to a different dataset in the `../../dataset/` directory, to select a different configuration, or to switch between 10-fold and leave-one-out cross-validation. The repository also include .properties files for every dataset/configuration pair, which can be selected using the method described above.

### 3 Integration with the CROSSMINER platform

As described in D2.7 – Framework and API Analysis – Final Report and in Section 2.1, FOCUS employs static source code analysis techniques to extract the list of method declarations and method invocations from OSS projects, which are then encoded in 3D rating matrices, which are then analyzed to infer a ranked list of method invocations that match the current context in which a developer is working.

In the context of CROSSMINER, FOCUS extracts information about method declarations and invocations from two sources: the source code currently being developed in the CROSSMINER IDE (locally) and the source code already analyzed by the CROSSMINER Knowledge Base (remotely). The CROSSMINER IDE
D2.8 API Analysis Components

communicates with the CROSSMINER Knowledge Base to gather actionable information about projects currently being developed, to allow taking the current context of the developer into account. As FOCUS is intended to recommend additional method calls and usage patterns when the developer is using a particular API, we decided to integrate FOCUS as part of the Knowledge Base.

Instead of duplicating the information about the integration in two documents, we refer the interested reader to deliverable D6.5 – The CROSSMINER Knowledge Base – Final Version for more information about this integration. In particular, Sections 10.5.10 and 10.6.11 describe the two REST endpoints exposed by the Knowledge Base that support the recommendation of API function calls and usage patterns, which are:

/api/recommendation/focus/ “Given a project source code as input this resource returns a recommended list of API function calls”.

/api/recommendation/recommended_API_call “Given a project code as input this resource returns a list of patterns that matches with the current code”.

More information about the information that must be supplied to these endpoints, with concrete examples, is given in D6.5. An important part of the input is the list of method declarations, the list of method invocations, and the current declarations in the code currently being developed by the user of the the CROSSMINER IDE. To enable the IDE to compute this information and supply it to the Knowledge Base, we implemented a small tool that makes the extraction of M^3 models from the code currently being developed in the CROSSMINER IDE possible. The IDE thus first builds the M^3 models of the project being developed locally, uses it to extract the list of method declarations and invocations, and invokes the corresponding Knowledge Base endpoints to get accurate recommendations for new function calls and usage patterns. Naturally, the Knowledge Base uses the FOCUS tool described in this document to build such recommendations internally.
Part II

MARACAS: a Framework for API Evolution Analysis and Client Code Migration

4 Introduction

MARACAS is a framework for API evolution and migration developed at Centrum Wiskunde & Informatica and written in the Rascal programming language. Its primary purpose is to support the co-evolution of APIs and client code: how to help developers migrating the client code that uses an API that has evolved? The current version of MARACAS is hosted and actively developed under the CROSSMINER organization umbrella on GitHub (https://github.com/crossminer/maracas).

In this document, we introduce MARACAS from a user perspective: how to set it up, how to interact with it and manipulate it, etc. Details of the approach implemented by MARACAS, the underlying models it builds, and the algorithms we use are available in the companion deliverable D2.7 – Framework and API Analysis – Final Report.

5 User Scenarios

MARACAS considers two types of users: (i) API developers and (ii) developers of client projects that use the APIs. In this section, we give a high-level overview of the questions MARACAS can answer, providing insights to both kinds of users.

5.1 Using MARACAS as an API Developer

In the case of API developers, MARACAS can be used to get insightful information about the evolution of an API itself by computing the number and detailed list of changes between two arbitrary versions of an API. Version should be understood here in a broad sense. MARACAS is flexible enough to compute changes between any two states of the source code or bytecode of an API: between two major or minor versions, between two commits, etc. Moreover, MARACAS distinguishes between breaking and non-breaking changes. We consider that gathering information about both breaking and non-breaking changes is valuable for the API developers, although only breaking changes have an impact on the clients of their API. It enables API developers to get a complete overview of the changes introduced between two versions of the API and, most importantly, to analyze how these changes might affect their users. Some of the questions API developers are faced with that can be answered by MARACAS are as follows:

- Which changes have been introduced between two versions of an API?
- How many and which of them are breaking changes?
- Which declarations have been deprecated in the latest release?
- What are the new features introduced in the latest release of an API?
D2.8 API Analysis Components

• Which parts of an API have been the most/the least affected by breaking changes in the recent past?

MARACAS can also analyze client projects to infer which parts of an API are used in practice (using usage models) and how the breaking changes introduced in an API affect the clients using this API (using detection models), cf. D2.7 – Framework and API Analysis – Final Report. Both of these analyses provide valuable feedback to API developers to better understand how their APIs are used and how the changes they introduce affect their users.

5.2 Using MARACAS as a Client Project Developer

Software project developers care the most about how the evolution of the APIs they are using might affect their code. In this case, we consider a client project that uses an old version of a given API and wants to migrate to a newer version. MARACAS is then meant to support client project developers during this migration process. The main goal is to reduce the amount of effort that developers need to invest in the migration activity, which, by itself, does not add any value to the client project in terms of new functionality. These questions are especially relevant in the context of CROSSMINER and are at the core of several use cases (Softeam, FrontEndArt), cf. D1.1 – Project Requirements. Currently, the tool computes and stores the API changes between two versions in a $\Delta$-model. These changes are then considered to detect the locations in the client project that have been affected. MARACAS considers both breaking changes and deprecated elements, which must be analysed and managed by the client project developers. To this aim, the tool computes a Detection model that pinpoints the physical locations in the client project that use a modified API access point (i.e., type, method, or field). Thus, client project developers do not need to manually search for API changes. They can actually consider MARACAS mappings to replace their code with the suggested one. These suggestions are computed directly from the source code with complete accuracy, or by means of using similarity functions that map old and new API elements with a confidence score. In general, client project developers can answer questions such as:

• Which parts of the client project have been affected by the API migration?
• What are the API suggested changes?
• What are the newly introduced features in a given version of the API?

Furthermore, we can use MARACAS to compute $\Delta$-models and Detection models in a codebase of client projects using a given API. With this information we can crosscheck our current suggestions against actual mappings obtained from migrated client projects. These suggestions can be refined with the previous information by means of using other techniques related to mining code repositories and recommender systems (cf. deliverable D6.5 – The CROSSMINER knowledge base - Final Version).

6 Getting Started

In this section, we present a quick guide to get started with the MARACAS framework. We consider both using it within Eclipse (cf. Section 6.2) and as a pure Java pipeline executed through Apache Maven (cf. Section 6.3).
6.1 Dependencies

MARACAS is implemented in Rascal, a functional programming language particularly suited for source code analysis and transformation [1]. It depends on a number of frameworks and libraries:

**Rascal** (version 0.12.0-SNAPSHOT and up) to parse and analyse source code and bytecode. The framework can be found at [https://update.rascal-mpl.org/unstable/](https://update.rascal-mpl.org/unstable/)

**java-string-similarity** (version 1.1.0) for computing text distance and similarity through different methods and functions (e.g., Levenshtein, Jaro-Winkler). The library can be found at [https://github.com/tdebatty/java-string-similarity](https://github.com/tdebatty/java-string-similarity)

**java-build-manager** (version 0.0.1) for extracting the classpath of Maven and OSGi-based Java projects during M3 models creation. The library is hosted at [https://github.com/cwi-swat/rascal-java-build-manager](https://github.com/cwi-swat/rascal-java-build-manager)

6.2 Using MARACAS in Eclipse

Using MARACAS in Eclipse enables users to manipulate the tool through a Read-Eval-Print Loop (REPL) which offers the best flexibility. Here, we present the main requirements to use MARACAS in the Eclipse IDE, and the main steps to benefit from the MARACAS API.

6.2.1 Requirements

Setting up MARACAS to be used in Eclipse is straightforward and involves the following steps:

1. Download and extract Eclipse RCP/RAP from the following URL: [https://www.eclipse.org/downloads/](https://www.eclipse.org/downloads/)
2. Install the current unstable version of the Rascal meta-programming environment using the following update site URL: [https://update.rascal-mpl.org/unstable](https://update.rascal-mpl.org/unstable)
3. Clone the Rascal project java-build-manager from the following URL and and import it into the Eclipse workspace: [https://github.com/cwi-swat/rascal-java-build-manager](https://github.com/cwi-swat/rascal-java-build-manager)

6.2.2 Step-by-step Guide

Once the Eclipse environment is ready, follow the steps below to import MARACAS, compute $\Delta$-models and detection models, and render them in user-friendly HTML reports:

1. Clone the MARACAS project from the following URL and import it into the Eclipse workspace: [https://github.com/crossminer/maracas](https://github.com/crossminer/maracas)
2. Run the Rascal REPL by right-clicking the maracas project and selecting Rascal Console.
3. In the REPL (> denotes the REPL prompt), import the org::maracas::Maracas module:

   > import org::maracas::Maracas;

4. Define the following loc variables which point, respectively, to the initial version of the API to be analyzed, its updated version, and a client project. Don’t forget to change the absolute paths to the projects:

   > loc apiv1 = |file:///Users/your/path/myjar-0.1.0.jar|;
   > loc apiv2 = |file:///Users/your/path/myjar-0.2.0.jar|;
   > loc client = |file:///Users/your/path/client-of-myjar-0.1.0.jar|;

5. Build the ∆-model between the two versions of the API using the MARACAS API. The resulting ∆-model can be subsequently filtered to restrict the list of changes to, respectively, classes and interfaces, methods, or fields, as shown below:

   > Delta delta = delta(apiv1, apiv2);
   > cdelta = classDelta(delta);
   > mdelta = methodDelta(delta);
   > fdelta = fieldDelta(delta);

6. To compute the list of detections (i.e., parts of the client code that are impacted by changes in the ∆-model), use the detections function as depicted below:

   > cdetection = detections(client, cdelta);
   > mdetection = detections(client, mdelta);
   > fdetection = detections(client, fdelta);

7. Optionally, ∆-models can be visualized in user-friendly HTML reports:

   > import org::maracas::delta::vis::Visualizer;
   > writeHtml(|file:///Users/your/path/ClassDelta.html|, cdelta);
   > writeHtml(|file:///Users/your/path/MethodDelta.html|, mdelta);
   > writeHtml(|file:///Users/your/path/FieldDelta.html|, fdelta);

6.3 Using MARACAS from Java

To ease the integration of MARACAS with the remainder of the CROSSMINER platform, we have implemented a Java API that can be employed to access MARACAS features from regular Java code, without having to rely on Eclipse and the REPL environment of Rascal. This API is specified in the Java class org.maracas.Maracas. This class also defines an entry point main() which can be directly invoked from Maven to ease the analysis of a whole dataset of APIs and projects. The arguments expected by the pipeline are as follows:

Usage: maracas <lib1Jar> <lib2Jar> <clientsPath> <reportPath>

Where lib1Jar is the path to the JAR file of the initial version of the considered API, lib2Jar is the JAR of the updated version of the API, clientsPath is a directory containing a list of JARs that use the API lib1Jar, and reportPath is a directory where the results of the MARACAS analysis will be serialized.

The full pipeline involves creating a ∆-model, filtering it to only consider breaking changes, and computing the Detection model of all the supplied clients. The models, as well as the HTML-based reports, are serialized for
future analysis and are stored within a pre-selected folder. The pipeline can conveniently be invoked directly from Maven. For instance, using the pipeline to analyze API evolution and migration of clients between Guava 18.0 and Guava 19.0 is realized as follows:

```
mvn package exec:java -Dexec.mainClass="org.maracas.Maracas" -Dexec.args="/path/to/guava-18.0.jar /path/to/guava-19.0.jar /path/to/guava-clients /path/to/guava-report"
```

### 7 Additional Features

Besides the creation of $\Delta$-models, *Detection* models, and *Migration* models, MARACAS exposes a number of convenient features, such as: filtering a $\Delta$-model to only consider breaking changes, non-breaking changes, changes at the type, method, or field levels; generating user-friendly HTML reports; customizing MARACAS code matchers; and extending MARACAS with new matchers and similarity functions. We give an overview of these features in this section.

#### 7.1 Built-in $\Delta$-model Filters

As shown in Section 6.2, a $\Delta$-model can be filtered to only get type-, method-, or field-level changes. This filtering is performed by means of creating the corresponding $\Delta$-model and then passing it as an argument to filtering functions such as `classDelta(Delta)`, `methodDelta(Delta)`, and `fieldDelta(Delta)`. Additionally, the $\Delta$-model can also be filtered to only consider the *breaking* changes, i.e., changes that break clients of the API:

```
> Delta delta = delta(apiv1, apiv2);
> bc = breakingChanges(delta);
```

#### 7.2 Generating $\Delta$-model Reports

MARACAS is able to generate HTML reports that present $\Delta$-model information in a user-friendly manner. As shown in the listing below, we invoke the `writeHTML(loc, Delta)` function from the `org::maracas::delta::vis::Visualizer` module. The first parameter, of type `loc`, points to the physical location where the HTML report should be generated. The second parameter corresponds to the $\Delta$-model itself.

```
> import org::maracas::delta::vis::Visualizer;
> writeHtml(|file:///Users/your/path/Delta.html|, cdelta);
```

The HTML report for an API such as Guava (versions 18.0 and 19.0) is showcased in Figures 1 and 2. Figure 1 presents some general statistics related to the content of the $\Delta$-model, mainly the number of tuples per relation in the model. Then, the report describes the content of every single relation and attribute in the $\Delta$-model. For instance, Figure 2 shows an excerpt of the HTML report related to the `abstractModifiers` relation. The table shows the location of the element that is impacted by a change in the $\Delta$-model (`column Old`), and then the mapping that has been detected. In this case, the mapping is either from an `abstract` method to a non-`abstract` method, or the other way around. The report also displays the confidence score (`Score`) that indicates how confident a user should be in the accuracy of the mapping. This confidence score is of particular importance when checking the renamed, moved, and deprecated relations where similarity and distance functions are involved.
Delta model between guava-18.0.jar and guava-19.0.jar

Statistics

<table>
<thead>
<tr>
<th>Type</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Method parameters changed</td>
<td>20</td>
</tr>
<tr>
<td>Static modifiers changed</td>
<td>0</td>
</tr>
<tr>
<td>Class/Interface implementation changed</td>
<td>23</td>
</tr>
<tr>
<td>Access modifiers changed</td>
<td>25</td>
</tr>
<tr>
<td>Abstract modifiers changed</td>
<td>12</td>
</tr>
<tr>
<td>Removed elements</td>
<td>494</td>
</tr>
<tr>
<td>Added elements</td>
<td>951</td>
</tr>
<tr>
<td>Renamed elements</td>
<td>55</td>
</tr>
<tr>
<td>Final modifiers changed</td>
<td>22</td>
</tr>
<tr>
<td>Deprecated elements</td>
<td>16</td>
</tr>
<tr>
<td>Moved elements</td>
<td>968</td>
</tr>
<tr>
<td>Field and method types changed</td>
<td>16</td>
</tr>
<tr>
<td>Class extension changed</td>
<td>122</td>
</tr>
</tbody>
</table>

Figure 1: General statistics about the Δ-model.

Abstract modifiers changed

<table>
<thead>
<tr>
<th>Old</th>
<th>From</th>
<th>To</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>[java+method:[//com/google/common/io/BaseEncoding/decodingStream(java.io.Reader)]</td>
<td>default()</td>
<td>abstract()</td>
<td>1.0</td>
</tr>
<tr>
<td>[java+method:[//com/google/common/collect/ImmutableSortedMap/tailMap(java.lang.Object,boolean)]</td>
<td>abstract()</td>
<td>default()</td>
<td>1.0</td>
</tr>
<tr>
<td>[java+class:[//com/google/common/reflect/ClassPath$Scanner]</td>
<td>default()</td>
<td>abstract()</td>
<td>1.0</td>
</tr>
<tr>
<td>[java+method:[//com/google/common/collect/ImmutableMapEntry/getNextInKeyBucket()]]</td>
<td>abstract()</td>
<td>default()</td>
<td>1.0</td>
</tr>
<tr>
<td>[java+method:[//com/google/common/collect/ImmutableSortedMap/values()]]</td>
<td>abstract()</td>
<td>default()</td>
<td>1.0</td>
</tr>
</tbody>
</table>

Figure 2: Detailed description of changes in abstract modifiers reported in the Δ-model.
7.3 Tuning Matchers Configuration

MARACAS includes a configuration file that allows to tune the analysis process when matching old and new API elements. This is needed specially when mapping removed and added elements for the renamed, moved, and deprecated relations of the Δ-model. To specify your preferences, go to the config.properties file within the config folder in the MARACAS project. Identify the matchers key and assign the set of matchers that you would like to use to perform the API matching. Currently, MARACAS includes two built-in matchers: levenshtein and jaccard. However, customised matchers can be hooked to the tool (cf. Section 7.4). Besides, it is also possible to compute all possible matches between old and new API elements using both Levenshtein and Jaccard matchers by including the following line in the configuration file: matchers = jaccard, levenshtein. In this case, both matchers will be applied to all elements in the Δ-model, with their own confidence score. If no matcher is declared in the configuration file, the tool defaults to Jaccard.

7.4 Hooking New Matchers

MARACAS allows developers to hook new similarity functions and their corresponding data representations. All the modules to add or modify are contained in the org::maracas::match package. Hereafter, we present the main steps to follow in order to add a new method to map declarations between two versions of the same API.

1. Reuse or add (if needed) a data representation. The first thing to check is if the data representation that you need for your similarity or distance function is already defined within MARACAS. Data representations are declared in the org::maracas::match::data::Data module. Listing 1 depicts the Data datatype that represents the data structure used by a set of similarity and distance functions. In this case, we consider two representations: sets (set) and strings (str). Both of them include a threshold that might be used by similarity and distance functions, which defaults to 0.0; and two map structures, namely from and to. Each of these maps includes a loc elem that references an API element, and a repr that represents the given element (either as a set[loc] or as a str in our two scenarios). The from map gathers all the removed declarations from the API with their corresponding representation, and the to map gathers all the added declarations to the new version of the API.

```
data Data
    = \set ( 
        real threshold = 0.0, 
        map[loc elem, set[loc] repr] from = {}, 
        map[loc elem, set[loc] repr] to = {} ) 
    | string ( 
        real threshold = 0.0, 
        map[loc elem, str repr] from = {}, 
        map[loc elem, str repr] to = {} ) 

Listing 1: The Data datatype.
```

Implementation a new representation, say a vector-based representation, would be possible by defining a new constructor in the Data datatype similar to the one depicted in Listing 2.

```
vector ( 
```

Confidentiality: Public Distribution
2. **Reuse or add (if needed) a similarity or distance function.** Once your needed data representation is defined and customised for your needs, you can reuse or add a new similarity or distance function. Built-in functions are declared in the org::maracas::match::fun package. This package already contains the SetSimilarity and StringSimilarity modules. The former considers the jaccardSimilarity function which, given two sets, computes a Jaccard score of type real (cf. Listing 3). The latter contains the levenshteinSimilarity function which, given two strings, computes the normalized Levenshtein similarity score, also of type real. If you want to add a new similarity or distance function, make sure you create a new function (within one of the modules of the target package) that returns a value of type real and compares only two elements of your chosen type. This type should be the same you use for the repr attribute in the constructor of the Data datatype. This type could be more generic as it is the case of the jaccardSimilarity function, which declares two parameters of type set[value].

```cpp
real jaccardSimilarity(set[value] x, set[value] y)
= (size(x) > 0 && size(y) > 0) ? jaccard(x, y) : -1.0;
```

Listing 3: jaccardSimilarity function.

3. **Create your own matcher.** At this point, you have both the data representation you need as well as the similarity or distance function that operates with your data. Then, you can create your own matcher. Matchers are located in the org::maracas::match::matcher package and all of them rely on the org::maracas::match::matcher::Matcher module. We need two elements from this module, namely the Matcher data type depicted in Listing 4 and the match(M3Diff, Data, real (&T, &T)) function. Every matcher should provide an implementation of the match function within the Matcher data type. In most cases, this implementation relies on the match(M3Diff, Data, real (&T, &T)) function of the Matcher module.

```cpp
data Matcher = matcher(
    set[Mapping[loc]] (M3Diff diff, real threshold) match);
```

Listing 4: Matcher data type.

```cpp
set[Mapping[loc]] jaccardMatch(M3Diff diff, real threshold)
= match(diff, createData(diff, threshold), jaccardSimilarity);
```

Listing 5: Jaccard matcher.

To illustrate the aforementioned statements, let us consider the implementation of the match function within the org::maracas::match::matcher::JaccardMatcher. Listing 5 shows this implementation. The jaccardMatch function declares the two expected parameters, namely a M3Diff computed in the Delta module, and a real that represents a threshold that might be used by the similarity or distance function. This declaration matches the expected function of the Matcher data type. The body of the declaration is an invocation of the match(M3Diff, Data, real (&T, &T)) function.
provided by the Matcher module. In this case, we create our Data value with the createData(M3Diff, real) function, and we pass the jaccardSimilarity(set[value], set[value]) as second argument. The match function computes the similarity of all the removed and added declarations contained in the Data value under the hood. In the end, we get a set of mappings pointing from an old API element to a new one. All these mappings are meant to have a higher score than the one defined with the threshold. This depends on the implementation of the chosen similarity or distance function.

4. Include your matcher as a new mapping option. Finally, to hook your matcher, add a new case to the switch statement within the applyMatchers(M3Diff, map[str,str], str) function in the org::maracas::delta::DeltaBuilder module. That is, select a string label for the matcher and hook the match function implementation to the matcher (cf. Listing 6). The selected label can then be used within the MARACAS configuration file (cf. Section 7.3).

```java
Matcher currentMatcher;
switch (matcherLabel) {
    case "levenshtein": currentMatcher = matcher(levenshteinMatch);
    case "jaccard": currentMatcher = matcher(jaccardMatch);
    default: currentMatcher = matcher(jaccardMatch);
}
```

Listing 6: applyMatchers switch statement in the DeltaBuilder module.

8 Integration with the CROSSMINER platform

8.1 Platform and Dashboard Integration

MARACAS is incorporated as a component of the CROSSMINER platform under the name org.eclipse.scava.maracas, publicly available on our central repository https://github.com/crossminer/scava. This plug-in enables any metric provider of the platform to invoke the creation of delta models, usage models, and detection models which can then be analyzed to compute metrics of interest. More specifically, the plug-in exposes the MARACAS API described in Section 6.2 to metric providers of the platform. The CROSSMINER platform works by analyzing projects every day. Our integration of MARACAS complies with this requirement by allowing to analyze the changes introduced in a project and its API on a daily basis. Specifically, one of the main access points of the API is the function computeDelta which can be invoked to compute the daily delta model from the M3 model of current date and the M3 model of the previous date, as depicted in Listing 7. All arguments passed to this function are automatically provided by the platform infrastructure at run time: a metric’s developer does not have to worry about them.

```java
tuple[M3 old, M3 new, Delta delta] computeDelta(ProjectDelta projectDelta,
    rel[Language, loc, M3] m3s,
    map[loc, loc] scratchFolders) {
    M3 previousM3 = loadPreviousM3(scratchFolders, projectDelta.date);
    M3 newM3 = systemM3(m3s, delta = projectDelta);
    Delta d = deltaFromM3(previousM3, newM3);
    serializeNewM3(scratchFolders, newM3, projectDelta.date);
    return <previousM3, newM3, d>;
}
```
Listing 7: The computeDelta function computes the daily *delta model*

We used the features exposed through the MARACAS API to implement various metrics related to API analysis in the CROSSMINER platform. These metrics are implemented in a dedicated plug-in `org.eclipse.scava.metricprovider.trans.rascal.api`. Listing 8 depicts the implementation of three metrics that have been implemented atop the MARACAS infrastructure introduced above.

```java
@metric{numberOfChanges}
@friendlyName{Number of changes}
@appliesTo{java()}
@resetOnEmptyDelta{}
@historic{}
int numberOfChanges(ProjectDelta projectDelta = ProjectDelta::empty(),
                     rel[Language, loc, M3] m3s = {},
                     map[loc, loc] scratchFolders = {})
{
    tuple[M3 old, M3 new, Delta delta] t = computeDelta(projectDelta, m3s, scratchFolders);

    return countChanges(t.delta);
}

@metric{numberOfBreakingChanges}
@friendlyName{Number of breaking changes}
@appliesTo{java()}
@resetOnEmptyDelta{}
@historic{}
int numberOfBreakingChanges(ProjectDelta projectDelta = ProjectDelta::empty(),
                             rel[Language, loc, M3] m3s = {},
                             map[loc, loc] scratchFolders = {})
{
    tuple[M3 old, M3 new, Delta delta] t = computeDelta(projectDelta, m3s, scratchFolders);

    return countChanges(breakingChanges(t.delta, t.old, t.new));
}

@metric{changedMethods}
@friendlyName{Changed methods}
@appliesTo{java()}
@resetOnEmptyDelta{}
@historic{}
set[loc] changedMethods(ProjectDelta projectDelta = ProjectDelta::empty(),
                        rel[Language, loc, M3] m3s = {},
                        map[loc, loc] scratchFolders = {})
{
    tuple[M3 old, M3 new, Delta delta] t = computeDelta(projectDelta, m3s, scratchFolders);

    Delta methodDelta = getMethodDelta(t.delta);

    return domain(methodDelta.accessModifiers) +
           domain(methodDelta.finalModifiers) +
           domain(methodDelta.staticModifiers) +
           domain(methodDelta.abstractModifiers) +
```
### 8.2 IDE Integration

Next to its use in the CROSSMINER dashboards, MARACAS is intended to be used directly within the CROSSMINER IDE to support developers as they are facing API evolution and migration scenarios.

In a similar fashion as for FOCUS (cf. Section 3), MARACAS is integrated as part of the CROSSMINER Knowledge Base, enabling the CROSSMINER IDE to query recommendations about API evolution and migration through a REST API. Thus, we refer the reader to **D6.5 – The CROSSMINER Knowledge Base – Final Version** for a complete presentation of how the Knowledge Base uses MARACAS to answer these queries.

The main endpoints that can be accessed are as follows:

<table>
<thead>
<tr>
<th>Endpoint</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>/api/api-migration/documentation/</td>
<td>“This resource provides a list of StackOverflow posts related to API migration”.</td>
</tr>
<tr>
<td>/api/api-migration/detection/</td>
<td>“This resource provides a list of client locations that are impacted by the evolution on the specific library”.</td>
</tr>
<tr>
<td>/api/api-migration/recommend/</td>
<td>“This resource provides a list of code snippet that other clients use to support the breaking changes introduced by the adoption of a new version of a library”.</td>
</tr>
</tbody>
</table>
References

