Analizo User Guide

João Miranda
Paulo Meirelles
Lucianna Almeida
Vinícius Daros
Fabio Kon
University of São Paulo (USP)

Antonio Terceiro
Joênio Costa
Luiz Romário Rios
Christina Chavez
Federal University of Bahia (UFBA)

15/06/2010

This work is partially funded by EU under the grant of IST-FP6-034763 – QualiPSo Project

http://www.qualipso.org

Executive Summary

This document contains information about the use of the Analizo toolkit. It is directed to users and provides useful steps to run and use Analizo.

See also installation guide and architeture guide documentations.

Table of Contents

1 INTRODUCTION

Analizo[1] is multi-language source code analysis and visualization toolkit. It supports the extraction and calculation of a fair number of source code metrics, generation of dependency graphs, and software evolution analysis. It efficiently parses source code written in C, C++ and Java and reports useful information.

Source code metrics have been proposed since the 1970s, but it is not have been extensively used and disseminated among the developers. The causes of this can be a hard learning and cost of a metrics specialist in a development team. The lack of tools and systematic method to automate the evaluation of source code also collaborate with a few use of metrics source code as an indicator in evaluating the software.

Our research group are working in these problems and developing different studies that involve source code metrics. In this context, our ongoing research depends on the analysis and visualization of source code, and that requires the appropriate tools.

We have defined requirements for the tool support we needed. We could not find any tool in the literature that fulfils all our following requirements:

- Multi-language: Being able to analyze different programming languages is desired once it can provide a wider validity to our studies. In particular, at least support for C, C++ and Java is required.
- Free software: In order to ensure the replicability of our studies, other researchers should have access to all procedures applied. Therefore, our tool of choice should also be free software[2], providing an unrestricted use of it.
- Extensibility: It should provide clear interfaces for adding new types of analyzes, metrics, or output formats. Hence, the tool will be able to continue supporting our studies as the research progresses.

[1]softwarelivre.org/mezuro/analizo

[2]In our work, we consider the terms “free software” and “open source software” equivalent.

2 Using Analizo

2.1 Analizo configuration

Analizo is a simple command line application which receives files and directories as inputs and reports information on the standard output. Being so, it does not require any configuration nor setups. Once it is installed, there are no further steps other than its use.

2.2 Running Analizo

Currently Analizo is able to analyze source code written in C, C++ and Java. However, it could also support other languages once Doxyparse[3] is based on Doxygen[4], which supports several different ones. The following section illustrates the Analizo use cases.

2.2.1 Analizo

First all, it is necessary to acknowledge that Analizo is a toolkit. Each of the tools are accessed through the main script called `analizo`.

With that being said, the first use case is to simply run this script through the following command. As an output, there is small manual on how to use Analizo and which tools are available:

```
$ analizo
```

2.2.2 Analizo Metrics

Analizo Metrics is responsible for all source code metrics. It receives a file or directory as input and reports a set of global metrics and another set for each module/class. Analizo Metrics reports both project-level metrics, which are calculated for the entire project, and module-level metrics, which are calculated individually for each module. On the project-level, Analizo also provides basic descriptive statistics for each of the module-level metrics.

Input

To run Analizo Metrics on a file called HelloWorld.java, for instance, use the command line and enter:

```
$ analizo metrics <filepath>/HelloWorld.java
```

[3]softwarelivre.org/mezuro/doxyparse

[4]stack.nl/~dimitri/doxygen

The same tool can be used on two or more files. If there are two files named HelloWorld.java and Main.java, enter the command below:

```
$analizo metrics <filepath>/HelloWorld.java
<filepath>/Main.java
```

Another alternative is to use a directory as input. If HelloWorld.java and Main.java are in a folder called MyProject, to analyze all files on a directory use:

```
$ analizo metrics <folderpath>/MyProject/
```

Output:

Analizo Metrics has a simple YAML output containing **global metrics** concerning the whole project and **module metrics** for each module analyzed. The global metrics are divided in two subsets:

- Statistical metrics take into account the metric values of every module to calculate average, standard-deviation, variance, maximum, minimum, median, skewness, kurtosis and sum of all of them.
- The Non-Statistical Metrics are project-wide metrics meaning that it is calculated just considering the source code information of the entire project and not the values of module metrics. They are:

  - Total Number of Abstract Classes
  - Total Coupling Factor
  - Total Effective Lines of Code
  - Total Methods Per Abstract Class
  - Total Modules/Classes
  - Total Modules with Defined Attributes
  - Total Modules with Defined Methods
  - Total Number of Methods

As for the module metrics, Analizo calculates:

- Afferent Conexions per Class
- Average Cyclomatic Complexity per Method
- Average Method Lines of Code
- Average Number of Parameters per Method
- Coupling Between Objects
- Depth of Inheritance Tree
- Lack of Cohesion of Methods

- Lines of Code
- Maximum Method Lines of Code
- Number of Attributes
- Number of Children
- Number of Methods
- Number of Public Attributes
- Number of Public Methods
- Response For a Class

Options:

`--extractor <extractor>`

Defines which extraction method will be used to parse and analyse the input source codes. The default is obviously Doxyparse, but it is possible to extend Analizo Metrics in order to use other parsers. When using the Doxyparse extractor (default), all files matching the languages supported by Doxyparse are processed.

`--list | -l`

Displays the metric list above described.

`--output <file>`

Use a file as output

`--global-only | -g`

Don't output the details about modules: only output global (project-wide) metrics.

`--help`

Displays a help message on how to use Analizo.

2.2.3 Analizo Graph

Analizo Graph is responsible for the construction of the applications call graph. It can output module dependency information extracted from a source code tree in a format suitable for processing with the Graphviz[5] graph drawing tools. Figure 1 presents a sample dependency graph obtained by feeding Graphviz' DOT[6] tool with Analizo's graph output. For each module, it reports what are its dependencies throught an edge between its function and a external one.

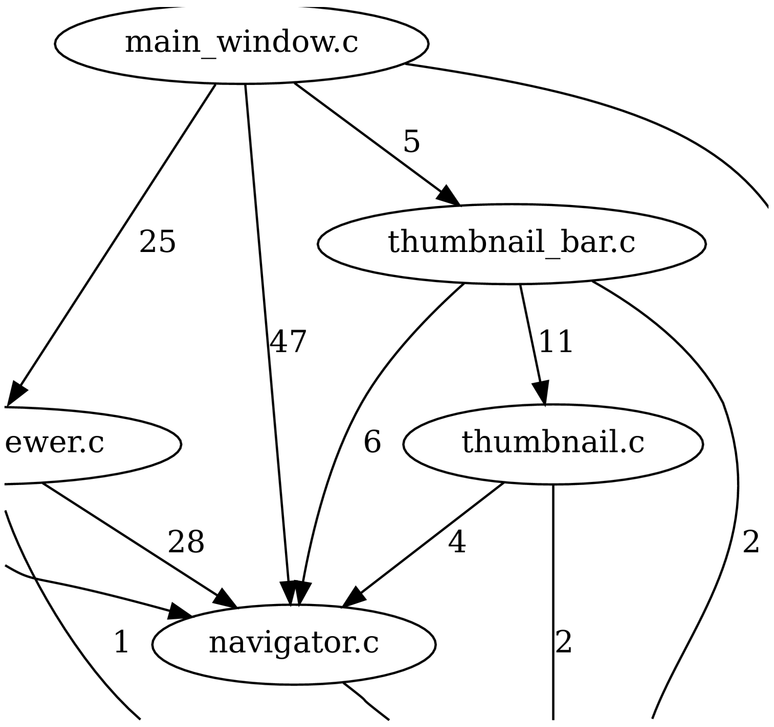

Figure 1: Sample module dependency graph

Input:

To obtain the call graph of a project at the folder named MyProject, use:

```
$ analizo graph <folderpath>/MyProject/
```

Output:

Analizo Graph output is a DOT digraph. Each module function is represented by a node and every function use is an arc. The following examples illustrate a possible output.

Suppose module A has a function called `foo()` and module B has one called `foobar()` and a variable called `B::var`. Also suppose that `A::foo()` calls `B::foobar()` and `A::foo()` uses the variable `B::var`. The output digraph contains:

[5]graphviz.org

[6]en.wikipedia.org/wiki/DOT_language

- A node called `A::foo()`
- A node called `B::foobar()`
- An arc from `A::foo()` to `B::foobar()`. This arc will be marked with `[style=solid]` since it is a function dependency.
- An arc from `A::foo()` to `B::var` marked with `[style=dashed]` because it is a variable dependency.

Options

--omit <functions>

Omit the given functions from the call graph. Multiple function names may be given separated by commas.

--cluster

Cluster the functions into files, so you can see in which files are the calling and called functions.

--modules

Group the functions by modules(files), and only represent calls between modules. This is useful to see the dependencies between the modules of the program, intead of focusing on specific functions. The arrows between the modules will be labelled with a number that represents the number of different places in which the calling module calls functions in the called module (i.e. how many times module A calls module B).

It doesn't make much sense to use --modules together with –cluster.

--extractor <extractor>

Defines which extraction method will be used to parse and analyse the input source codes. The default is obviously Doxyparse, but it is possible to extend Analizo Metrics in order to use other parsers.

When using the Doxyparse extractor (default), all files matching the languages supported by Doxyparse are processed.

--output <file>

Use the specified file as output

--help

Displays help on command line syntax and options.

2.2.4 Metrics batch processing

In most quantitative studies on Software Engineering involving the acquisition of source code metrics on a large number of projects, processing each project individually is impractical, error-prone and difficult to repeat. Analizo can process multiple projects in batch and produce one comma-separated values (CSV) metrics data file for each project, as well as a summary CSV data file with project-level metrics for all projects. These data files can be easily imported in statistical tools or in spreadsheet software for further analysis. This can also be used to analyze several releases of the same project, in software evolution studies.

```
$ analizo metrics-batch <folderpath> -s dataset.csv
```

2.2.5 Metrics history

Sometimes researchers need to process the history of software projects on a more fine-grained scale. Analizo can process a version control repository (CVS, Subversion, Git, etc) and provide a CSV data file with the metrics values for each revision in which source code was changed in the project.

```
$ analizo metrics-history <repositotypath>
```

2.2.6 Evolution matrix

Another useful Analizo feature is generating evolution matrices. After processing each release of the project (see section 2.2.3), the user can request the creation of an evolution matrix from the individual data files. Figure 2 shows an excerpt of a sample evolution matrix.

```
$ analizo evolution-matrix <folderpath>
```

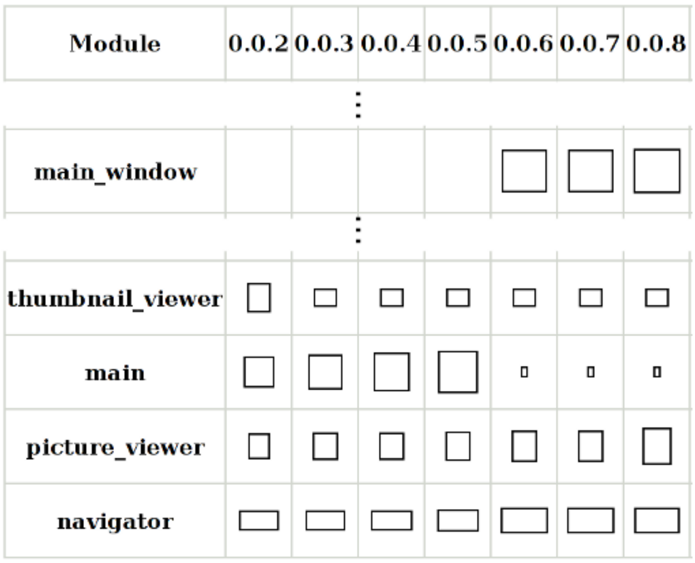

Figure 2: Sample evolution matrix produced by Analizo

3 Troubleshooting

Analizo has no reported bugs or troubles up to date. If the user occasionally finds one, please report it on:

- http://github.com/terceiro/analizo/issues
- Mailing list:
  - mezuro@listas.softwarelivre.org
  - http://listas.softwarelivre.org/cgi-bin/mailman/listinfo/mezuro

STD_COSTRAINT_FOR

The user may encounter the following message while using Analizo. Note that this does not cause any failure on the source code analysis nor on the output results. This is not an Analizo error, but a warning concerning the Getopt::Euclid Perl Module:

```
Variable "%STD_CONSTRAINT_FOR" will not stay shared at
/usr/share/perl5/Getopt/Euclid.pm line 208.
```

4 License

5 ACKNOWLEDGMENT

This tool also is supported by CNPQ, FAPESB, the National Institute of Science and Technology for Software Engineering (INES), and USP FLOSS Competence Center.