Analizo: an Extensible Multi-Language Source Code Analysis and Visualization Toolkit

Antonio Terceiro[1], **Joenio Costa**[2], **João Miranda**[3], **Paulo Meirelles**[3], **Luiz Romário Rios**[1], **Lucianna Almeida**[3], **Christina Chavez**[1], **Fabio Kon**[3]

[1] Universidade Federal da Bahia (UFBA)

{terceiro,luizromario,flach}@dcc.ufba.br

[2]Universidade Católica do Salvador (UCSAL)

joenio@perl.org.br

[3]Universidade de São Paulo (USP)

{joaomm,paulormm,lucianna,fabio.kon}@ime.usp.br

Abstract. *This paper describes Analizo, a free, multi-language, extensible 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.*

1. Introduction

Software engineers need to analyze and visualize the software they create or maintain in order to better understand it. Software Engineering researchers need to analyze software products in order to draw conclusions in their research activities. However analyzing and visualizing large individual software products or a large number of individual software products is only cost-effective with the assistance of automated tools.

Our research group have been working with empirical studies that require large-scale source code analysis, and consequently we resort to source code analysis tools in order to support some of our tasks. We have defined the following requirements for the tool support we needed:

- Multi-language. The tool should support the analysis of different programming languages (in particular, at least C, C++ and Java), since this can enhance the validity of our studies.
- Free software. The tool should be free software[1], available without restrictions, in order to promote the replicability of our studies by other researchers.
- Extensibility. The tool should provide clear interfaces for adding new types of analyzes, metrics, or output formats, in order to promote the continuous support to our studies as the research progresses.

In this paper, we present Analizo, a toolkit for source code analysis and visualization, developed with the goal of fulfilling these requirements. Section 2 describes related work. Section 3 describes Analizo architecture. Section 4 presents Analizo features. Section 5 presents Analizo use cases. Finally, Section 6 concludes the paper and discusses future work.

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

2. Related work

While evaluating the existing tools to use in our research, we analyzed the following ones: CCCC [Littlefair 2010], Cscope [Steffen et al. 2009], LDX [Hassan et al. 2005], CTAGX [Hassan et al. 2005], and CPPX [Hassan et al. 2005]. Besides the research requirements described, we have included two practical requirements:

- The tool must be actively maintained. This involves having active developers who know the tool architecture and can provide updates and defect corrections.
- The tool must handle source code that cannot be compiled anymore. For example, the code may have syntax errors, the libraries it references may be not available anymore, or the used libraries changed API. This is important in order to be able to analyze legacy source code in software evolution studies.

The requirements evaluation for the tools are presented in Table 1. Since we only looked at tools that were free software, the table does not have a line for that requirement.

Requirement	**CCCC**	**Cscope**	**LDX**	**CTAGX**	**CPPX**
Language support	C++, Java	C	C, C++	C	C, C++
Extensibility	No	No	No	No	No
Maintained	Yes	Yes	No	No	No
Handles non-compiling code	Yes	No	No	No	No

Table 1. Found tools versus posed requirements

As it can be seen in Table 1, none of the existing tools we found fulfills all of our requirements. In special, none of the tools were able to analyze source code in all three needed languages, and none of them had documented extension interfaces that could be used to develop new analysis types or output formats.

3. Architecture

Analizo architecture is presented in Figure 1, using a Layered style [Clements et al. 2002]. Each layer in the diagram uses only the services provided by the layers directly below it.

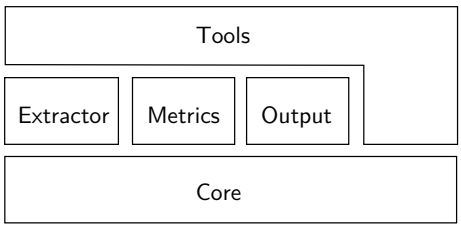


Figure 1. Analizo architecture, using the Layered Style [Clements et al. 2002]

The *Core* layer contains the data structures used to store information concerning the source code being analyzed, such as the list of existing modules[2], elements inside each module (attributes/variables, or methods/functions), dependency information (call,

[2]we used the "module" concept as a general term for the different types of structures used in software development, as classes and C source files

inheritance, etc). This layer implements most of Analizo business logic, and it does not depend on any other layer.

The *Extractors* layer comprises the different source code information extraction strategies built in Analizo. Extractors get information from source code and store them in the *Core* layer data structures. It requires only the creation of a new subclass to add a new type of extractor that interfaces with another external tool or provides its own analysis directly. Currently, there are two extractors. Both are interfaces for external source code parsing tools:

- *Analizo::Extractors::Doxyparse* is an interface for Doxyparse, a source code parser for C, C++ and Java developed by our group [Costa 2009]. Doxyparse is based on Doxygen[3], a multi-language source code documentation system that contains a robust parser.
- *Analizo::Extractors::Sloccount* is an interface for David A. Wheeler's Sloccount[4], a tool that calculates the number of effective lines of code.

The other intermediate layers are *Metrics* and *Output*. The *Metrics* layer processes *Core* data structures in order to calculate metrics. At the moment, Analizo supports a fair set of metrics (listed in Section 4). The *Output* layer is responsible for handling different file formats. Currently, the only output format implemented is the DOT format for dependency graphs, but adding new formats is simply a matter of adding new output handler classes.

The *Tools* layer comprises a set of command-line tools that constitute Analizo interface for both users and higher-level applications. These tools use services provided by the other layers: they instantiate the core data structures, one or more extractors, optionally the metrics processors, an output format module, and orchestrate them in order to provide the desired result. Most of the features described in Section 4 are implemented as Analizo tools.

Those tools are designed to adhere to the UNIX philosophy: they accomplish specialized tasks and generate output that is suitable to be fed as input to other tools, either from Analizo itself or other external tools. Some of the tools are implemented on top of others instead of explicitly manipulating Analizo internals, and some are designed to provide output for external applications such as graph drawing programs or data analysis and visualization applications.

4. Features

4.1. Multi-language source code analysis

Currently, Analizo supports source analysis of code written in C, C++ and Java. However, it can be extended to support other languages since it uses Doxyparse, which is based on Doxygen and thus also supports several different languages.

4.2. Metrics

Analizo 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

[3] `doxygen.org/`
[4] `dwheeler.com/sloccount/`

project-level, Analizo also provides basic descriptive statistics for each of the module-level metrics: sum, mean, median, mode, standard deviation, variance, skewness and kurtosis of the distribution, minimum, and maximum value. The following metrics are supported at the time of writing[5]:

- Project-level metrics: Total Coupling Factor, Total Lines of Code, Total number of methods per abstract class, Total Number of Modules/Classes, Total number of modules/classes with at least one defined attributes, Total number of modules/classes with at least one defined method, Total Number of Methods.
- Module-level metrics: Afferent Connections per Class, Average Cyclomatic Complexity per Method, Average Method LOC, Average Number of Parameters per Method, Coupling Between Objects, Depth of Inheritance Tree, Lack of Cohesion of Methods, Lines of Code, Max Method LOC, Number of Attributes, Number of Children, Number of Methods, Number of Public Attributes, Number of Public Methods, Response For a Class.

4.3. 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.

4.4. 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 and provide a CSV data file with the metrics values for each revision in which source code was changed in the project. Git and Subversion repositories are supported directly, and CVS repositories must be converted into Git ones beforehand.

4.5. Dependency Graph output

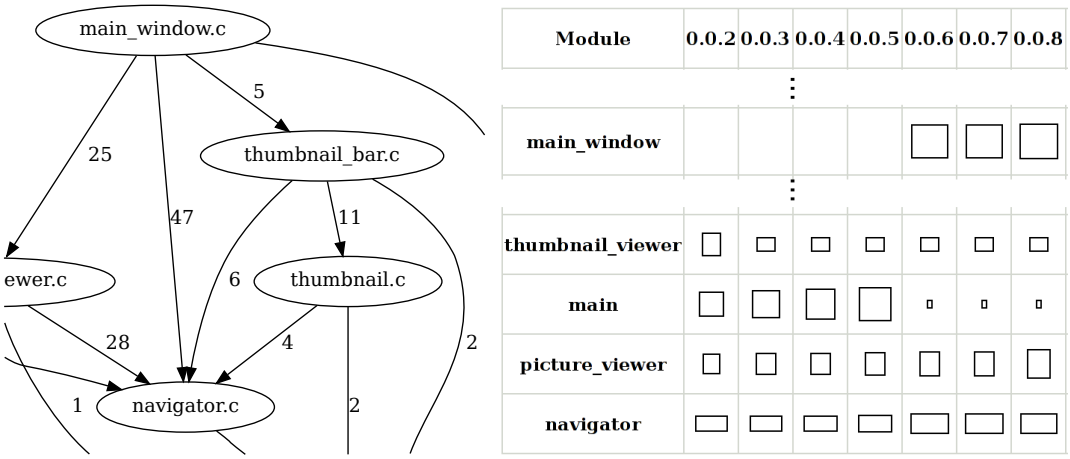
Analizo can output module dependency information extracted from a source code tree in a format suitable for processing with the Graphviz[6] graph drawing tools. Figure 2(a) presents a sample dependency graph obtained by feeding Graphviz' *dot* tool with Analizo graph output.

4.6. Evolution matrix

Another useful Analizo feature is generating evolution matrices [Lanza 2001]. After processing each release of the project (see Section 4.3), the user can request the creation of an evolution matrix from the individual data files. Figure 2(b) shows an excerpt of a sample evolution matrix produced by Analizo.

[5]References to literature on each metric were omitted because of space constraints.

[6]`graphviz.org/`



(a) Sample module dependency graph (b) Sample evolution matrix

Figure 2. Examples of Analizo features.

5. Use cases

Analizo has been validated in the context of research work performed by our group that required tool support with at least one of its features:

- [Amaral 2009] used Analizo module dependency graph output to produce an evolution matrix for a case study on the evolution of the VLC project. Later on, an evolution matrix tool was incorporated in Analizo itself.
- [Costa 2009] did a comparison between different strategies for extracting module dependency information from source code, leading to the development of Doxyparse – the Analizo Doxygen-based extractor.
- [Terceiro and Chavez 2009] used the metrics output on an exploratory study on the evolution of structural complexity in a free software project written in C.
- [Morais et al. 2009] used the Analizo metrics tool as a backend for Kalibro[7], a software metrics evaluation tool. Later on, Kalibro Web Service[8] was developed, providing an integration with Spago4Q[9] – a free platform to measure, analyze and monitor quality of products, processes and services.
- [Terceiro et al. 2010] used the metrics history processing feature to analyze the complete history of changes in 7 web server projects of varying sizes.
- [Meirelles et al. 2010] used Analizo metrics batch feature to process the source code of more than 6000 free software projects from the Sourceforge.net repository.

Most of the work cited above contributed to improvements in Analizo, making it even more appropriate for research involving source code analysis.

6. Final remarks

This paper presented Analizo, a toolkit for source code analysis and visualization that currently supports C, C++ and Java. Analizo has useful features for both researchers

[7] `softwarelivre.org/mezuro/kalibro/`
[8] `ccsl.ime.usp.br/kalibro-service`
[9] `spago4q.org`

working with source code analysis and professionals who want to analyze their source code in order to identify potential problems or possible enhancements.

Future work includes the development of a web-based platform for source code analysis and visualization based on Analizo. This project is current under development.

Analizo is free software, licensed under the GNU General Public License version 3. Its source code, as well as pre-made binary packages, manuals and tutorials can be obtained from `softwarelivre.org/mezuro/analizo`. All tools are self-documented and provide on-line manuals. Analizo is mostly written in Perl, with some of its tools written in Ruby and Shell Script.

This work is supported by CNPQ, FAPESB, the National Institute of Science and Technology for Software Engineering (INES), Qualipso project, and USP FLOSS Competence Center (CCSL-USP).

References

Amaral, V. (2009). Análise de evolucao de projetos de software livre através de matrizes de evolucao. Undergraduation course conclusion project, Universidade Federal da Bahia.

Clements, P., Bachmann, F., Bass, L., Garlan, D., Ivers, J., Little, R., Nord, R., and Stafford, J. (2002). *Documenting Software Architecture : Views and Beyond*. The SEI series in software engineering. Addison-Wesley, Boston.

Costa, J. (2009). Extração de informações de dependência entre módulos de programas c/c++. Undergraduation course conclusion project, Universidade Católica do Salvador.

Hassan, A. E., Jiang, Z. M., and Holt, R. C. (2005). Source versus object code extraction for recovering software architecture. In *Proceedings of the 12th Working Conference on Reverse Engineering (WCRE'05)*.

Lanza, M. (2001). The evolution matrix: recovering software evolution using software visualization techniques. In *IWPSE '01: Proceedings of the 4th International Workshop on Principles of Software Evolution*, pages 37–42, New York, NY, USA. ACM.

Littlefair, T. (2010). CCCC - C and C++ Code Counter. Available at `http://cccc.sourceforge.net/`. Last access on June 3rd, 2010.

Meirelles, P., Jr., C. S., Miranda, J., Kon, F., Terceiro, A., and Chavez, C. (2010). A Study of the Relationship between Source Code Metrics and Attractiveness in Free Software Projects. *Submitted*.

Morais, C., Meirelles, P., and Kon, F. (2009). Kalibro: Uma ferramenta de configuração e interpretação de métricas de código-fonte. Undergraduation course conclusion project, Universidade de São Paulo.

Steffen, J., Hans-Bernhard, and Horman, B. N. (2009). *Cscope*. http://cscope.sourceforge.net/.

Terceiro, A. and Chavez, C. (2009). Structural Complexity Evolution in Free Software Projects: A Case Study. In Ali Babar, M., Lundell, B., and van der Linden, F., editors, *QACOS-OSSPL 2009: Proceedings of the Joint Workshop on Quality and Architectural Concerns in Open Source Software (QACOS) and Open Source Software and Product Lines (OSSPL)*.

Terceiro, A., Rios, L. R., and Chavez, C. (2010). An Empirical Study on the Structural Complexity introduced by Core and Peripheral Developers in Free Software projects. *Submitted*.