NAME OF ARTIFACT

TraceLab Components for Reproducing Source Code Summarization Experiments

AUTHORS' CONTACT DETAILS

Breno D. Cruz <bdantesc@nd.edu>
Paul W. McBurney <paulmcb@seas.upenn.edu>
Collin McMillan <cmc@nd.edu>

SUMMARY

This artifact is a reproducibility package for experiments in source code summarization (for automatic documentation generation). The artifact is implemented as a set of components for the TraceLab research infrastructure. We have converted two implementations of state-of-the-art source code summarization into prepackaged and easily-reusable TraceLab components. Prior to this conversion, the implementations were accessible but difficult to use, being scattered across numerous scripts in various languages with many dependencies.

A KEY DEFINITION

Source code summarization is the process of creating natural language descriptions of source code. The input to a source code summarization technique is source code plus other software artifacts. The output is natural language text which describes the behavior of, purpose behind, and/or details in that source code.

SCORECARD

Insightful

Timely (i.e., addresses a problem that is most current and most pressing)?: This artifact is timely because it enables researchers to reproduce state-of-the-art techniques in source code summarization, and to test new ideas against the state-of-the-art approaches. Source code summarization is a relatively new research area and working implementations of state-of-the-art approaches are relatively rare. The result is that progress is slowed during the infancy of this research area, as scientists must reconstruct earlier work in order to evaluate new ideas.

Makes researchers "smarter" in some way (e.g., fills some significant gap in prior work)?: What this artifact does is provide researchers with a complete tool kit for not only reproducing two existing techniques, but also tinkering with the internals of those techniques, or creating entirely novel techniques out of the pieces of the existing ones. While this is possible in theory already, this artifact greatly simplifies the process by resolving all dependencies and porting existing code into one language in a unified framework.

Useful

Serves a useful purpose?: The purpose of this artifact is to provide easy access to recent source code summarization techniques, so that other researchers can modify and improve those techniques. Source code summarization is a key technology in research on automatic documentation generation. For the state-of-the-art to advance, it is critical that existing techniques be reproducible.

Serves a purpose that would otherwise be tedious, prolonged, awkward, or impossible?: The techniques reproduced by this artifact have been published before, and the implementations are available through online appendices. However, the existing implementations are complex and involve a variety of languages and temperamental dependencies. This artifact streamlines the implementations by porting them to one language and packaging them with the data and dependencies.

Cost-effective?: The artifact has no financial cost to the user. Support for development is provided by the United States NSF grant CNS-1510329.

Usable

Easy to understand?: Specific steps we have taken to simplify the artifact include: 1) porting all code to Java so there is only a single programming language, 2) use the graphical interface provided by the TraceLab research infrastructure, 3) provide step-by-step instructions with screenshots, and 4) include two virtual machine images as examples.

Accompanied by tutorial notes?: We provide notes in two forms: 1) step-by-step instructions online, and 2) README files for every component.

Details for Executable Artifacts:

- *Easy to download, install, or execute?*
  The components can be installed to the TraceLab research infrastructure, or may be used via one of the provided virtual machine images.
- *Available in a virtual machine image?*
  We provide working copies of our artifact inside two images created for VirtualBox 5: 1) a Debian image containing only the bare minimum necessary for operation, and 2) an Ubuntu image with standard features and development tools enabled.
- *Available online?*
  All resources are available via:
  http://www.cse.nd.edu/~cmc/tracelab/icsme16/
- *Supported by configuration management tools to permit easy updates?*
  Configuration management support is provided via TraceLab. To update the artifact, a user would only need to copy the new component files into TraceLab's components directory.

TraceLab Components for Reproducing Source Code Summarization Experiments

Breno Dantas Cruz*, Paul W. McBurney†, and Collin McMillan*
*Dept. of Computer Science and Engineering
University of Notre Dame
Notre Dame, IN, USA
{bdantesc, cmc}@nd.edu
†Dept. of Computer and Information Science
University of Pennsylvania
Philadelphia, PA, USA
paulmcb@seas.upenn.edu

***Abstract*—This artifact is a reproducibility package for experiments in source code summarization. The artifact is implemented as a set of components for the TraceLab research infrastructure. We have converted two implementations of state-of-the-art source code summarization into prepackaged and easily-reusable TraceLab components. Prior to this conversion, the implementations were accessible but difficult to use, being scattered across numerous scripts in various languages with many dependencies. We provide the components, detailed tutorials, and two example virtual machine images via our online appendix.**

I. Introduction / Background

Source code summarization is the process of creating natural language descriptions of source code. The input to a source code summarization technique is source code plus other software artifacts. The output is natural language text which describes the behavior of, purpose behind, and/or details in that source code.

Source code summarization is a somewhat new research area and working implementations of state-of-the-art approaches are relatively rare. The result is that progress is slowed during the infancy of this research area, as scientists must reconstruct earlier work in order to evaluate new ideas.

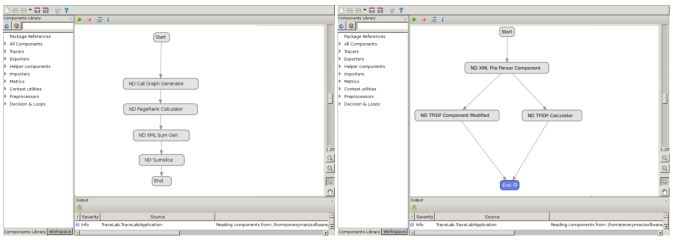
Broadly speaking, there are two strategies for summarization: abstractive and extractive. This paper presents an artifact containing reproducible implementations for one abstractive and one extractive technique. The abstractive technique was created by McBurney *et al.* [2] and the extractive technique by Rodeghero *et al.* [3]. We felt these were good choices because both techniques have been published in IEEE Transactions on Software Engineering within the previous year.

II. The Artifact

This artifact contains two "`teml`" files and seven "components." A `teml` file is a file describing an experiment in the TraceLab research framework [1]. An experiment includes the implementation of a research prototype, and specifies the input and output data of the implementation. The implementation is made up of "components" whose interaction is controlled by the `teml` file. TraceLab provides a platform for executing the experiments. To reproduce an experiment, a scientist only needs to load a `teml` file, set input parameters (if different ones from the default are desired), and click run. TraceLab executes the components presents results.

We built one `teml` file for each of the two techniques we reimplemented. We also built four components for the technique by McBurney *et al.* and three components for the technique by Rodeghero *et al*. The artifacts are mature in the sense that they faithfully reimplement the previous work. We note that while existing implementations of the previous work are available, they are not straightforward to reuse because they rely on a variety of programming languages, and have several temperamental dependencies. This artifact resolves these problems by using on one language (Java) and packaging dependencies via TraceLab.

The following figure shows a birds-eye view of both experiments and the seven components side-by-side in TraceLab:



On the right are the three components for the Rodeghero *et al.* experiment:
File Parser creates a markup of the subject source code.
TF/IDF implements the baseline comparison approach.
TF/IDF MOD implements Rodeghero's approach.

On the left are the four components for the McBurney *et al.* experiment:
Call Graph Gen. generates a call graph for source code.
PageRank Calc. calculates the PageRank of each vertex.
XML Preparer prepares source code data for analysis.
SumSlice implements McBurney's approach.

The experiments, components, detailed tutorials with screenshots, and two example virtual machine images are available via:

http://www.cse.nd.edu/~cmc/tracelab/icsme16/

Acknowledgment: This work is supported in part by the NSF CCF-1452959 and CNS-1510329 grants. Any opinions, findings, and conclusions expressed herein are the authors and do not necessarily reflect those of the sponsors.

References

[1] J. Cleland-Huang. TraceLab. *Online.* http://www.coest.org/.
[2] P. W. McBurney and C. McMillan. Automatic source code summarization of context for java methods. *IEEE Trans. on Software Engineering*, 42(2):103–119, Feb 2016.
[3] P. Rodeghero, C. Liu, P. W. McBurney, and C. McMillan. An eye-tracking study of java programmers and application to source code summarization. *IEEE Trans. on Software Engineering*, 41(11):1038–1054, Nov 2015.