SoftwareX 7 (2018) 222–225

Contents lists available at ScienceDirect

SoftwareX

journal homepage: www.elsevier.com/locate/softx

Original software publication

From source code to publication: Code Diary, an automatic documentation parser for SAS

Christiaan H. Righolt *, Barret A. Monchka, Salaheddin M. Mahmud

Vaccine and Drug Evaluation Centre, Department of Community Health Sciences, University of Manitoba, 337–750 McDermot Avenue, Winnipeg MB, R3E 0T5 Canada

ARTICLE INFO

Article history:
Received 15 December 2017
Received in revised form 16 May 2018
Accepted 11 July 2018

Keywords:
Documentation
Up-to-date
Parser
Comments
SAS
Code-publication match

ABSTRACT

Team members do not always review, or understand, all source code and the decisions that are made in it. Code developers and maintainers should have tools available to easily write, maintain, collate and share source code documentation. Institutional security demands often limit the types of software that researchers can install on their systems. It is, therefore, necessary to run a documentation tool natively within programs that are already installed. We developed Code Diary, an automatic documentation parser for SAS 9.3 and up. Code Diary provides a way to generate documentation natively from SAS source code.

Software metadata

Current software version	*1.0.4*
Permanent link to executables of this version	https://github.com/VaccineAndDrugEvaluationCentre/code-diary-sas/releases/tag/v1.0.4; https://zenodo.org/record/1244766#.WvRli5dOmUk
Legal Software License	*GNU General Public License v3.0*
Computing platform/Operating System	*Any system with SAS installed*
Installation requirements & dependencies	*Requires SAS*
If available Link to user manual - if formally published include a reference to the publication in the reference list	*N/A*
Support email for questions	VDEC@umanitoba.ca, *issues on GitHub might be answered quicker*

Code metadata

Current Code version	*v1.0.4*
Permanent link to code/repository used of this code version	https://github.com/ElsevierSoftwareX/SOFTX_2018_45
Legal Code License	*GNU General Public License v3.0*
Code Versioning system used	*git*
Software Code Language used	*SAS*
Compilation requirements, Operating environments & dependencies	*None, only SAS*
If available Link to developer documentation/manual	*N/A*
Support email for questions	VDEC@umanitoba.ca, *issues on GitHub might be answered quicker*

1. Introduction

Advances in technology since the mid-20th century have resulted in a dramatic rise in the use of programmable computer tools in scientific and medical research [1]. Most medical research projects involve manipulating data using statistical (e.g., SAS, R, Stata) or general-purpose programming languages (e.g., Python) where often lengthy and complicated programs or scripts (source code) are developed and maintained. Research teams working on such projects have an internal division of labor. The content experts and investigators responsible for research design, writing papers and disseminating results may not be developing the source

* Corresponding author.
E-mail addresses: Christiaan.Righolt@umanitoba.ca (C.H. Righolt), Barret.Monchka@umanitoba.ca (B.A. Monchka), Salah.Mahmud@gmail.com (S.M. Mahmud).

https://doi.org/10.1016/j.softx.2018.07.002

code, or even have the expertise to fully review it to ensure that it matches the project requirements and achieves the research objectives [2].

In this paper, we describe the software requirements, implementation and use of Code Diary, a documentation tool that automatically generates a report of what the code does, based on the standard SAS comment structure.

2. Problems and background

Source code documentation provided by the code writer (often a statistician, analyst, programmer or graduate student) is often the only written information available for future users and maintainers of the code. In our experience, source code documentation is also often the only written information available to the rest of the scientific team that would explain what was done to arrive at the results. This documentation is typically a stand-alone document that needs to be maintained and expanded continuously to remain synchronized with the code. Programmers might also make small decisions, e.g., how to handle missing values, without remembering to include that in an external document and without consulting the primary investigator. It is imperative that code developers and maintainers have tools available to simplify the tasks of writing, maintaining, collating and sharing source code documentation. This enables the research team to be aware of all decisions, including minor decisions, in the code and to adjust the design of the code easily if required. It also makes it easier and quicker to maintain code while simultaneously improving output fidelity, directly increasing productivity. Source code documentation should be as close to the source code itself as possible, ideally in the same repository. This lowers the barriers to write and locate documentation. Documentation can then evolve, using standard versioning and branching tools, and remain in sync with the rest of the repository.

Several documentation tools (e.g., Doxygen, Haddock, JSDoc, Sphinx, and Read the Docs) have been developed to support developers working with general-purpose languages. The programmer uses certain conventions and keywords to document their code's routines and public interfaces (functionality, inputs and outputs, error codes, etc.). They then run a program (essentially a code parser) to extract and assemble the documentation text using a pre-defined template. The results can then be output into a standard format (e.g., HTML, RTF or PDF) for sharing and publication. Although sophisticated and elaborate, these tools are rarely suitable for use in many data-intensive medical research projects. First, existing tools are better at describing programming constructs (e.g., interface signature, function parameters and return values) than on documenting the decisions made during data preparation and analysis that are needed to interpret the results correctly. It is also common that research institutions limit the types of software that can be installed on systems used for analyzing clinical data, due to strict privacy and confidentiality rules and fear of security breaches [3]. On such closed systems, it is not unusual to be limited to the statistical analysis tools installed on the system. Faced by this situation in our own work, we developed a documentation tool that runs natively in one of our statistical analysis tools: SAS.

3. Requirements

In discussions with programmers, data analysts and investigators, we agreed on the following requirements. The tool should:

1. run natively in SAS v9.3 (SAS Institute, Cary, North Carolina), without any other dependencies. SAS is a data analysis software suite, initially released in 1976, which is widely used especially in medical research. A 2011 study estimated that SAS was used in approximately 40% of health research studies in the United States [4]. SAS does not have built-in documentation facilities for user-written code.
2. generate its output using a human-readable non-proprietary text format that can be converted afterwards into different standard publishing formats such as Microsoft Word, HTML or PDF.
3. automatically extract blocks of documentation from the source file. Documentation blocks would be placed right before the relevant code block to minimize maintenance overhead as the programmer can easily update the documentation block to reflect changes made in the associated code. Compared to maintaining the documentation block in a separate document, this approach should increase the chance that the programmer will remember to update the documentation.
4. enable the organization of the resulting documentation in a logical way, i.e., documentation blocks can be grouped thematically regardless of their location in the code. For instance, it should be possible to extract a list of all exclusion and inclusion criteria of study subjects even if the source code used to enforce these criteria is scattered in separate files.
5. enable the programmer to exclude specific documentation blocks (e.g., comments, reminders, task lists) from appearing in the final output.
6. make it easy to find the location of documentation back in the source code. This will facilitate maintenance and corrections of both the source code and its documentation.

4. Implementation

Our implementation consists of a language with a simple syntax (the Code Diary language) and a generator tool that produces a human-readable structured document containing the documentation.

4.1. Syntax and document structure

Code Diary syntax, loosely based on Doxygen syntax, is simple and easy-to-learn as it uses standard SAS comment syntax [5] allowing the user to write both single line comments and comment blocks. A Code Diary single line comment starts with "**" and ends with ";" whereas a comment block starts with "/**" and ends with "*/." The addition of the "*" to the standard SAS comment syntax is what identifies a comment as a Code Diary comment.

A logically structured document is much more useful than a collection of points in arbitrary order. To organize the comments thematically, the user flags each comment as belonging to a specific section or subsection using the @section.subsection syntax, entered as the first word in the comment. For example, a comment flagged with "@methods.population.matching" would be placed in the generated documentation under section Methods, subsection Population and subsubsection Matching. In the generated document, each Code Diary comment is formatted as a separate bullet point. This is useful for succinct documentation that can be read quickly and expanded easily into a full text methods section for a report or manuscript. See below for directions to an example SAS file with Code Diary comments.

4.2. Document generator

The document generator, implemented as a SAS macro, is invoked with the file path to the desired SAS script. The parser