Science of Computer Programming 96 (2014) 121–140

Contents lists available at ScienceDirect

Science of Computer Programming

www.elsevier.com/locate/scico

# Verifiable source code documentation in controlled natural language

Tobias Kuhn [a], Alexandre Bergel [b,*]

[a] Chair of Sociology, in particular of Modeling and Simulation, ETH Zurich, Switzerland
[b] Pleiad Lab., Department of Computer Science (DCC), University of Chile, Chile

## HIGHLIGHTS

- This paper highlights a deficiency in the way documentation is usually written.
- Using controlled natural language is a viable approach.
- A prototype as well as its benefits are described.

## ARTICLE INFO

*Article history:*
Received 7 September 2012
Received in revised form 12 November 2013
Accepted 1 January 2014
Available online 9 January 2014

*Keywords:*
Controlled natural language
Metamodelling
Moose
Pharo

## ABSTRACT

Writing documentation about software internals is rarely considered a rewarding activity. It is highly time-consuming and the resulting documentation is fragile when the software is continuously evolving in a multi-developer setting. Unfortunately, traditional programming environments poorly support the writing and maintenance of documentation. Consequences are severe as the lack of documentation on software structure negatively impacts the overall quality of the software product. We show that using a controlled natural language with a reasoner and a query engine is a viable technique for verifying the consistency and accuracy of documentation and source code. Using ACE, a state-of-the-art controlled natural language, we present positive results on the comprehensibility and the general feasibility of creating and verifying documentation. As a case study, we used automatic documentation verification to identify and fix severe flaws in the architecture of a non-trivial piece of software. Moreover, a user experiment shows that our language is faster and easier to learn and understand than other formal languages for software documentation.



## 1. Introduction

Software documentation is commonly understood as any description of software attributes, and documenting software is a critical factor to success [1]. Glancing at the website of the ten most popular SourceForge software programs[1] easily demonstrates that: all software comes with a fair amount of documentation, including screenshots, tutorials, and FAQs.

However, scrutinizing the documentation of these popular programs reveals that the offered documentation, with extensive installation procedures and problem troubleshooting instructions, is essentially focused on end users. None of the projects we studied present a clear description of their internal structure. The source code of eMule, for example, weighs

* Corresponding author.
*E-mail address:* kuhntobias@gmail.com (T. Kuhn).
*URLs:* http://www.tkuhn.ch (T. Kuhn), http://bergel.eu (A. Bergel).

[1] VLC, eMule, Vuze, Ares Galaxy, Smart package, 7-Zip, Notepad++ Plugin Manager, PortableApps.com, FileZilla, and MinGW are the most downloaded software at the time this article is being written. See http://sourceforge.net/.

0167-6423/$ – see front matter 
http://dx.doi.org/10.1016/j.scico.2014.01.002

more than 8 MB with 260 000 lines of code. The extensive website of eMule contains many documentation files,[2] but none of them are targeted toward developers.

This situation is not an exception, especially when the documenting of the software internals is not a strong requirement [2], as is often the case with open source software. There are good reasons for this, including the lack of resources, the fragility of the documentation [3,4], and the lack of immediate benefits. Keeping the internal documentation up to date with the source code is particularly difficult [5]. This article proposes an effective and convenient solution to write documentation that is interpreted as a formal model, is automatically checked, and is kept in sync with the source code.

We propose the use of controlled natural language [6,7] to represent software structure and its relation to the outside world, which leads to documentation that is automatically verifiable. The idea is to let developers write statements in a subset of English, describing the design and the environment of software that is otherwise nowhere explicitly expressed in the source code (e.g., *Simon is a developer*, *Shapes is a component*, *Every subclass of MOShape belongs to Shapes*, and *Simon maintains every class that belongs to Shapes*).

Among the different kinds of controlled natural languages [7], our approach relies on the type of languages that provide a natural and intuitive representation for formal notations. The statements of such languages look like natural language (such as English), but are controlled on the lexical, syntactic, and semantic levels to enable automatic and unambiguous translation into formal logic notations. We propose to use such natural-looking statements to write documentation statements about the source code structure. Ideally, these documentation statements should be next to the source code, e.g. in code comments, and written, read, and evaluated in the IDE of the developer. In this way, changes in the source code or its documentation that would introduce an inconsistency can be automatically detected. Furthermore, the logical structure of the documentation is accessible to query engines, via questions written in the same controlled natural language.

We evaluate our approach using the language Attempto Controlled English (ACE) [8,9] and the semantic wiki engine AceWiki [10]. To test the feasibility of our approach, we apply it to document Mondrian, an open-source visualization engine, and to improve its design. In addition, we conduct a user experiment to test how controlled natural language compares to other formal languages for source code documentation and querying.

In this article, we investigate the following two research questions:

A – *Can controlled English be efficiently used to document software systems?*
B – *Is controlled English easier to understand than other formal languages for software documentation?*

A growing amount of empirical evidence [11] shows that programmers spend a significant amount of time understanding source code and navigate between source code of software components. The generality of available software engineering tools and their poor performance are often pointed out as the culprit for engineering tasks being carried out in a suboptimal fashion [12,13]. Informal, broken, or poorly conceived documentation seem to be at the root of many difficulties encountered in software engineering tasks [11,14,15]. The two research questions we have stated above explore a new direction: documentation should be written in restricted natural language and automatically verified against the actual source code.

We tackle the first research question by documenting a significant amount of source code in ACE using AceWiki. The second research question is investigated by conducting a user experiment with 20 participants.

We would like to stress that this article is *not* about providing yet another solution for querying source code. Effective solutions have been proposed already, using logic programming [16] or natural language [17]. Reverse-engineering and reengineering environments typically offer sophisticated way to query and extract information from source code. Our approach combines source code modeling techniques with general logic expressions for documentation, all accessed via a control natural language and accessible to formal reasoning. ACE has been successfully applied in areas such as the Semantic Web, however its application to software documentation is new and has not been considered as far as we are aware of.

We also would like to point out that our approach focuses on *internal* code documentation. Software documentation can describe a wide variety of aspects, including system requirements, architecture, design decisions, and implementation details. In this article, we focus on the low level source code structure where documentation statements talk about source code elements such as classes and methods, but express things that are not visible in the source code. This includes connections to the outside world, such as who is responsible for which classes and what external devices a particular piece of code depends on.

In order to answer the research questions introduced above, this article makes the following contributions:

i – it presents a concrete approach to document source code with a controlled natural language;
ii – it shows how source code modeling can be combined with controlled natural language and automated reasoning;
iii – it evaluates the approach on a case study of documenting a medium-sized application;
iv – it presents the results of a user experiment for the comparison of the understandability of different languages for source code documentation.

[2] http://www.emule-project.net/home/perl/help.cgi?l=1.