Developing Stop Word Lists for Natural Language Program Analysis*

Benjamin Klatt, Klaus Krogmann
FZI Research Center for Information Technology
Haid-und-Neu-Str. 10-14,
76131 Karlsruhe, Germany
{klatt,krogmann}@fzi.de

Volker Kuttruff
CAS software AG
CAS-Weg 1-5,
76131 Karlsruhe, Germany
volker.kuttruff@cas.de

1 Introduction

When implementing a software, developers express conceptual knowledge (e.g. about a specific feature) not only in program language syntax and semantics but also in linguistic information stored in identifiers (e.g. method or class names) [6]. Based on this habit, Natural Language Program Analysis (NLPA) is used to improve many different areas in software engineering such as code recommendations or program analysis [7]. Simplified, NLPA algorithms collect identifier names and apply term processing such as camel case splitting (i.e. "MyIdentifier" to "My" and "Identifier") or stemming (i.e. "records" to "record") to subsequently perform further analyzes [10]. In our research context, we search for code locations sharing similar terms to link them with each other. In such types of analysis, filtering stop words is essential to reduce the number of useless links.

Just collecting, splitting, and stemming the identifier names, can result in a list of terms with divergent grade of usefulness. For example, the terms "get" and "set" are used in most Java application due to common coding practices and not to express any conceptual knowledge. These terms corrupt the program analysis leading to unreasonable findings.

To reduce this noise, a typical approach in natural language processing is to filter terms known as useless (aka "stop words"). For natural languages, many stop word lists are publicly available. However, as Høst et al. [5] identified, developers use a more specific vocabulary than in general spoken language. So common stop word lists are not reasonable to be used in program analysis, they even depend on domain, application type, developing company, and project settings.

In this paper, we propose an approach to develop reusable stop word lists to improve NLPA. We i) propose to distinguish different scopes a stop word list applies to (i.e. programming language, technology, and domain) and ii) recommend types of sources for terms to include. Our approach is closely related to the work of Ratiu [8], who recommended considering domain knowledge for program analysis in general. We propose a specific application of the concept as guidelines for developing stop word lists. Our approach is not limited to a specific technology but, according to our research context, examples in this paper are based on Java technology.

*Acknowledgment: This work was supported by the German Federal Ministry of Education and Research (BMBF), grant No. 01IS13023 A-C.

2 Stop Word Scopes

In general natural language processing, it is common sense, that different languages have different stop words. For example, a stop word list for analyzing English language contains the term "and", while for German language it would contain the term "und". Furthermore, a stop word list for a scientific domain would differ from a list for a sport domain. These lists cannot be reused between the domains' scopes but reused within the same.

In a similar way, we propose to not develop a global stop word list, but to build stop word lists for different scopes. More specific, we recommend distinguishing three different dimensions: Programing language, technology and domain.

Programming Language Most state-of-the-art programming languages allow to use nearly all characters in identifier names (e.g. Java accepts all UTF-8 characters except of digits at the beginning of identifiers). However, for each programming language, there are a lot of common sense naming conventions. For example, in Java, methods to access an object's attribute should start with the term "get". We propose to build programming language specific stop word lists, that reflect those common sense terms used as part of identifier names to express specific technical but not conceptual knowledge.

Technology On top of programming languages, developers build different types of applications (e.g. desktop or web applications) with according technologies, each of them leading to specific terms used in identifiers. For example, the terms "dialog" or "button" are often used in desktop applications with a Swing-based Graphical User Interface (GUI). A stop word list containing those terms could be reused for analyzing other Swing-based GUI-applications.

Domain Every domain has common terms that are used frequently and are not specific to implemented concepts. For example, in applications for literature management, the term ISBN might be used quite frequently as global identifier with low contribution to specific features. Developing a domain specific stop

word list (e.g. for a company or a product), would allow to improve the NLPA for this domain and could be reused and evolve within this context.

3 Stop Word Sources

Developing a stop word list depends on the scope and context it is used in. On the one side, developing a list for the context of analyzing arbitrary applications in the scope of a specific programming language, allows for using generic and publicly available sources and studies. On the other side, developing a stop word list to be used in the context of an analysis project within a company developing niche products allows to use less publicly available but even more specific sources such as interviewing developers.

So far, we have identified different sources for developing stop word lists for different scopes and contexts. As scopes and contexts vary, there will be no finite list of sources. However, we identified types of sources that one could use for developing or reusing stop word lists. The types of sources presented below result from investigating systems in our research project. Accordingly, we do not claim for completeness but for a reasonable set to consider.

Programming Language Naming Conventions Almost every programming language offer either explicit or common sense coding conventions including terms to be used, such as the "get" prefix in Java mentioned before (e.g. JavaBeans Specification [3]). One should exploit such conventions when developing a stop word list for a specific technology.

Programming Habit Observations In the context of NLPA, researchers have analyzed existing applications to identify typical programming habits during identifier naming. For example, Høst et al. [4], Sajaniemi et al. [9], and Caprile et al. [1] analyzed programs to identify frequently used terms expressing more technical and less conceptual knowledge.

Design Patterns When implementing design patterns, developers tend to identify the role of a class or interface within the pattern as part of its identifier (e.g. "Account*Model*", or ""Account*View*"). These role names can be received from pattern catalogs as provided by Gamma et al. [2]. Nowadays, many pattern catalogs exist for general purpose as well as specific applications. One must use a catalog that matches the scope of the stop word list developed.

Framework and Library Terms Frameworks and programming libraries cause developers to use terms in identifier names resulting from specific framework or library features. For example, developing OSGi[1] components includes classes to control the life-cycle of a component. The identifier of such a class typically includes the term "Activator". This is additionally stimulated by development environments (IDE) supporting a framework or library. It is recommended to check the framework, library, or IDE support for terms that are typically used when developing a stop word list for analyzing applications based on them.

Company Guidelines Explicitly documented or not, most companies have naming guidelines or cultures. They can range from acronyms to shorten identifiers up to terms resulting from code libraries shared within a company. Collecting such terms requires to interview experts because they are rarely documented.

4 Limitations & Future Work

Stop word lists are only one technique used in NLPA. However, we rate filtering stop words as a valuable and traceable one. We claim stop word list development to be significantly improved by considering programming language, application type or domain-specific scopes and sources as presented in this paper.

In the context of the KoPL project[2], we are working on such stop word lists, planing to publish those for common programming languages and technologies and to build a company internal repository. This work is still in progress, but results are promising and will also be published as part of our future work. To evaluate different lists, we are going to facilitate them in our analyzes and present the results to developers for assessing them with a focus on which terms have been filtered out and which have not.

References

[1] B. Caprile and P. Tonella. Nomen est omen: analyzing the language of function identifiers. In *Proceedings of WCRE 1999*. IEEE.

[2] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design patterns: elements of reusable object-oriented software*. Addison-Wesley, 1995.

[3] G. Hamilton. Sun Microsystems JavaBeans. Technical report, Sun Microsystems, 1997.

[4] E. W. Høst and B. M. Østvold. The Programmer's Lexicon , Volume I : The Verbs. In *Proceedings of SCAM 2007*. IEEE.

[5] E. W. Høst and B. M. Østvold. The Java Programmer's Phrase Book. In *Software Language Engineering*, volume 5452 of *Lecture Notes in Computer Science*, pages 322–341. Springer, 2009.

[6] A. Kuhn, S. Ducasse, and T. Girba. Semantic clustering: Identifying topics in source code. *Information and Software Technology*, 49(3):230–243, 2007.

[7] L. Pollock, K. Vijay-Shanker, E. Hill, G. Sridhara, and D. Shepherd. Natural Language-Based Software Analyses and Tools for Software Maintenance. In *ISSSE 2009-2011*, pages 102–134, 2012.

[8] D. Ratiu. Domain Knowledge Driven Program Analysis. *Softwaretechnik-Trends*, 29(2), 2009.

[9] J. Sajaniemi and R. N. Prieto. An Investigation into Professional Programmers' Mental Representations of Variables. In *Proceedings of IWPC 2005*. IEEE.

[10] P. van der Spek, S. Klusener, and P. van de Laar. Towards Recovering Architectural Concepts Using Latent Semantic Indexing. In *Proceedings of CSMR 2008*. IEEE.

[1] www.osgi.org

[2] http://www.kopl-project.org