Automatically Categorizing Software Technologies

Journal:	*Transactions on Software Engineering*
Manuscript ID	TSE-2017-06-0169
Manuscript Type:	Journal First
Keywords:	H.3.3.a Clustering < H.3.3 Information Search and Retrieval < H.3 Information Storage and Retrieval < H Information Technology a, H.3.3.h Search process < H.3.3 Information Search and Retrieval < H.3 Information Storage and Retrieval < H Information Technolo, H.3.1.d Linguistic processing < H.3.1 Content Analysis and Indexing < H.3 Information Storage and Retrieval < H Information Tech

Automatically Categorizing Software Technologies

Mathieu Nassif, Christoph Treude and Martin P. Robillard

Abstract—Informal language and the absence of a standard taxonomy for software technology make it difficult to reliably analyze technology trends on discussion forums and other on-line venues. We propose an automated approach called Witt for the categorization of software technologies (an expanded version of the hypernym discovery problem). Witt takes as input a phrase describing a software technology or concept and returns a general category that describes it (e.g., integrated development environment), along with attributes that further qualify it (commercial, php, etc.). By extension, the approach enables the dynamic creation of lists of all technologies of a given type (e.g., web application frameworks). Our approach relies on Stack Overflow and Wikipedia, and involves numerous original domain adaptations and a new solution to the problem of normalizing automatically-detected hypernyms. We compared Witt with five independent taxonomy tools and found that, when applied to software terms, Witt demonstrated better coverage than all evaluated alternate solutions, without a corresponding degradation in false positive rate.

1 Introduction

Software development relies increasingly on reusable components in the forms of frameworks and libraries, and the programming languages and tools to use them. Considered together, these *software technologies* form a massive and rapidly-growing catalog of building blocks for systems. For example, at least 135 different web application frameworks have been developed and made publicly available [39].

Given the continual emergence of new software development technologies, it is increasingly difficult to monitor how these technologies are mentioned and discussed by various stakeholders. The unstructured data, informal nomenclature, and folksonomies used on social media forums make it difficult to reliably determine, for example, the relative popularity of a certain type of technology. Answers to questions such as "what is the most popular web application framework?" are routinely proposed without any kind of supporting data (e.g., [32]). To move towards an evidence-based approach to monitoring the use of software technologies, we need to be able to *automatically* classify and group named mentions of software technologies.

An important step toward the machine understanding of terminology is hypernym discovery, i.e., the discovery of the more general concept in a *is-a* relationship (e.g., AngularJS *is a* web application framework). Unfortunately, discovering valid hypernyms is not sufficient to support the detection and monitoring of comparable software technologies. For example, *commercial cross-platform IDE for PHP* is a valid hypernym for PhpStorm, but the expression is too specific to constitute a useful category of technologies. Categorizing software technologies is a much more complex problem that requires additional abstraction and normalization.

We propose a *completely automated* approach for the categorization of software technologies. Our approach, called Witt, for *What Is This Technology*, takes as input a phrase such as PhpStorm and returns a general category that describes it (e.g., *integrated development environment*), along with attributes that further qualify it (*commercial*, *php*, etc.). Like many modern information extraction approaches, we rely on external web resources, in our case Wikipedia and Stack Overflow. Our approach involves three phases that each address a major technical challenge. First, we find the Wikipedia article (or article section) that best describes the input phrase. Then, we use the article to extract candidate hypernyms for the phrase. Finally, we extract general categories and related attributes for the hypernyms.

With our approach used on the set of all tags from Stack Overflow, we were able to automatically classify software technologies and, by extension, create dynamic lists of all technologies of a given type (e.g., web application frameworks). As an example, the list of programming languages includes less widespread languages such as ChucK, an audio language, and Bluespec, a hardware description programming language. Using the extensions of software category technologies, we can automatically find, for example, all recent Stack Overflow posts that refer to *any* web application framework. For example, Figure 1a) shows the *proportion* of Stack Overflow posts tagged with a specific web application framework over all posts tagged with a web application framework – a category (and categorization function) that Witt *automatically constructed*. For example, we can see that starting in May 2012 the number of posts about the AngularJS framework quickly grew from negligible to over 25% of all posts on web application frameworks. Currently this kind of analysis is not possible without a manually-crafted list of tags or keywords that match all technologies of interest. Figure 1b) is another similar example from which we can learn that mobile operating systems

- *M. Nassif is with the School of Computer Science, McGill University, Montréal, QC, Canada*
  E-mail: mnassif@cs.mcgill.ca
- *C. Treude is with the School of Computer Science, University of Adelaide, Adelaide, SA, Australia*
  E-mail: christoph.treude@adelaide.edu.au
- *M.P. Robillard is with the School of Computer Science, McGill University, Montréal, QC, Canada*
  E-mail: martin@cs.mcgill.ca

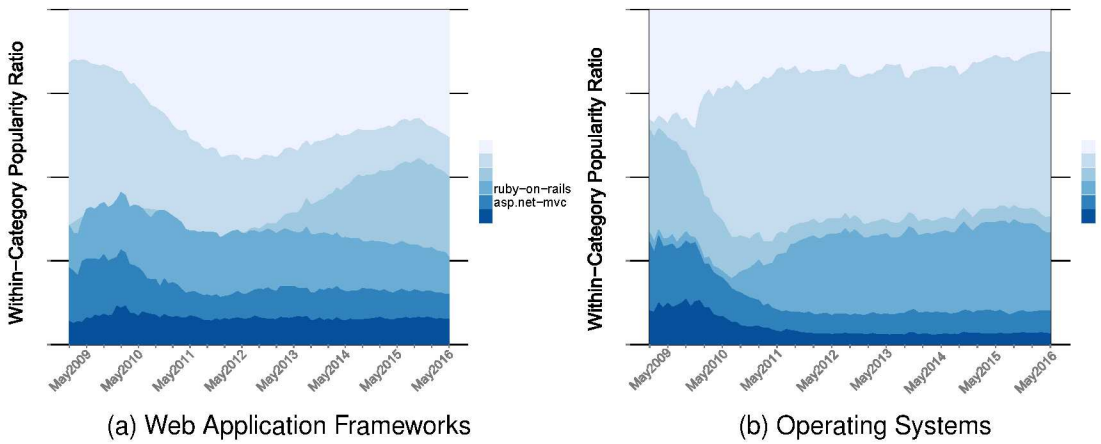

Fig. 1. Ratio of Stack Overflow posts tagged with a specific technology over all posts tagged with a technology in the same *automatically discovered* category (web application frameworks, and operating systems)

like Android or iOS are the target of the vast majority all the Stack Overflow posts about operating systems.

As another example of an application made possible by Witt, one can implement a simple Reddit parser that takes all titles in a subreddit as input and attaches categories based on a lookup of the words in the titles in our categories database. Using this application, one can filter, for example, only the posts related to operating systems. After implementing this strategy, we found that of the most recent 957[1] posts in the `programming` subreddit,[2] 38 contained an operating system name in their title. Without Witt, this analysis would have required a complete list of all operating system names. Such an application could be used to automatically group content on Reddit by category, making it easier to browse related topics.

These applications, however, are only some of numerous potential applications. The contribution of this paper is on the categorization technique, not a specific application.

Witt involves numerous domain-specific specializations of information extraction techniques. To evaluate the quality of the results and the value of the domain adaptation in general, we compared our approach to five available hypernym discovery tools, ranging from the venerable WordNet [24] to the most recent development, WiBi [12], and including the Google search engine. When queried for software terms, Witt was found to be superior to the five alternatives in terms of combined coverage and number of false positives.

The contributions of this paper include an automated approach for categorizing software technologies. The approach expands numerous information extraction techniques with adaptations for the software domain. The approach also includes a new algorithm for the abstraction of hypernyms of software terms into general categories.

The approach was developed with the goal of grouping a large set of software technologies into categories all at once, so that one can track all of the instances of one of the categories easily. It was intended to provide answers to questions such as "what are all the web application frameworks available?", but relies on the ability to answer questions like "what is this new technology X?"

Finally, we provide the first comparative evaluation of six existing hypernym discovery tools as applied to software terminology.

In Section 2 we present the background on information extraction and the related work in software engineering. Sections 3–6 describe the approach, starting with a general overview. Our comparative evaluation is presented in Section 7, followed by the conclusion in Section 8.

On-line Appendix. This paper is complemented by an on-line appendix that contains sample input data, a coding guide for labelling data, and text processing resources. The appendix is available at http://www.cs.mcgill.ca/~swevo/witt/.

2 Background and Related Work

This work builds on previous efforts on the construction of general taxonomies and relates to efforts in the development of lexicographic resources for software engineering, and to studies aimed at understanding tagging and other knowledge structuring practices in software engineering.

Background on Taxonomy Construction

Work on the automated construction of linguistic relations is usually traced to the development of WordNet [24], a digital lexicographic resource that includes manually-constructed semantic relations such as hypernymy and synonymy. Particularly relevant to our work is Miller et al.'s definition of hyponym (and, conversely, hypernym): "A concept [...] x is said to be a hyponym of the concept [...] y if native speakers of English accept sentences constructed from such frames as 'An x is a (kind of) y"' [24, p.8].

Foundational work on automating the construction of word relations through text mining then followed: Hearst proposed a series of lexico-syntactic patterns that usually indicate hyponymy (the so-called Hearst patterns, e.g, "such as X") [15], and Caraballo extended the idea by aggregating hypernyms into a hierarchy [4]. These basic approaches

1. This arbitrary number is a consequence of the Reddit API limitations.
2. https://www.reddit.com/r/programming/

were later improved by taking into account the use of linguistic dependencies [30] and supervised machine learning algorithms [28]. All these approaches work by mining large text corpora.

Contrary to these previous works, our method only requires Stack Overflow tag information data and targeted Wikipedia searches. It takes as input a set of words, and attempts to find hypernyms for each of these words. It also searches for hypernyms outside the initial input, as opposed to projects like WiBi, which finds hypernymy relations inside its input.

Wikipedia as a Source of Encyclopedic Knowledge

More recent work often leverages Wikipedia as the main resource to guide taxonomy construction, as it is commonly agreed to be the largest freely available collection of encyclopedic knowledge [44].

Several works have focused on automatically discovering and resolving ambiguous links between terms from textual documents and Wikipedia, a process known as *wikification*. One of the many challenges in such a task is to disambiguate between many similarly named entities, such as the Java programming language and the Indonesian island of the same name. Milne and Witten [25] used unambiguous links as contextual information to give as input to a trained classifier. Mihalcea and Csomai [23] evaluated two disambiguation approaches, the first one also using a machine learning classifier, but with the words surrounding the target term as input rather than other links, and the second one comparing the context in which the target term is found to the potential definitions. Because the tag information on Stack Overflow is typically concise, and because we cannot make the assumption that the relevant link exists, we cannot rely on these traditional approaches. Thus, we implemented a novel linking approach based on different information and domain-specific conditions.

Others have worked on the extraction of semantic relations between Wikipedia articles. Nakayama et al. [26], for example, retrieved semantic relations of any kind by discovering both the related terms and the predicate of the relation, such as *is bordered by* when linking *Djibouti* and *Eritrea*. Closer to our work, Targeted Hypernym Discovery (THD) attempts to find hypernyms for a query using the Linked Hypernyms Dataset generated from Wikipedia [10], [16].

Given the amount of links and meta-data available in Wikipedia, numerous other approaches to information extraction are possible, such as leveraging similarities between words [42], keyword popularity [17], or HTML tables [7]. One of the most recent work in the area, the Wikipedia Bitaxonomy Project (WiBi), extracts information from Wikipedia articles to add to the taxonomy of Wikipedia categories, and vice-versa [12], with the goal of improving the quality of the resulting knowledge structure.

Finally, some of the structured data extraction efforts on Wikipedia make their way into DBPedia [3], a crowd-sourced database of structured information that can also be queried for hypernyms through various tools and APIs [22]. Similar efforts are also under way in industry [40].

Our work contrasts from those efforts in that we search hypernyms for targeted terms, rather than extracted entities from a text, and thus must deal with very limited context to find the relevant article. We overcome this lack of context by using domain-specific heuristics.

Lexicographic Resources for Software Engineering

Falleri et al. used natural language processing to extract important concepts from identifiers defined in source code, aggregating them into a WordNet-like structure that includes their hypernymy relation [11]. In a similar vein, Nonnen et al. use heuristics to discover where concepts associated with an identifier are introduced or described in source code [27]. In both of these approaches all terms are derived from source code elements, information which cannot generally be used to categorize software technologies.

A major limitation of WordNet for applications to software engineering is the lack of support for specialized terminology. A number of projects have targeted the design of lexical databases that include a *word similarity* relation (a variant of synonymy). This relation can be computed from co-occurrences in the context of a forum post [33], [34] and its meta-data [37], or from source code comments and identifiers [43]. Similarity relations can be especially helpful to support *searching*, either for source code or related resources, because they help bridge the vocabulary gap between queries and documents [5]. Our work is different from these efforts in that we attempt to detect and organize hypernyms, which is a different type of semantic relation.

Hypernym Aggregation and Similarity Measures

To obtain a useful classification, similar hypernyms must be aggregated. Sentence aggregation have been developed a lot in the context of single- and multi-documents summarization, such as review summarization on e-commerce platforms. Several types of algorithms, from machine learning [14] to graph-based [45], have been used to infer similarity measures between sentences. More recently, Aliguliyev [1] developed a novel clustering approach based on the normalized Google distance [6].

The added difficulty of parsing small fragment rather than full sentences and the negative impact of domain-specific terms and expressions on the quality of NLP tools output prevent us from using these traditional aggregation methods. Thus, we implemented our own heuristics based on our domain-specific knowledge.

Tagging in Software Engineering

Tags often function as descriptors for software technologies, and may thus act as hypernyms.

In a study on the use of tags in a work item management system, Treude and Storey found that software developers had developed implicit and explicit mechanisms to manage tag vocabularies [36]. For a larger tag vocabulary on software project hosting websites such as Freecode, Wang et al. proposed a similarity metric to infer semantically related tags and build a taxonomy [37]. Li et al. proposed an agglomerative hierarchical clustering framework which relies only on how similar every two tags are, using data from Ohloh [19]. It is important to note that their work does not produce a hypernym hierarchy: for example, the term *hibernate* is clustered as a subnode of *java*, rather than

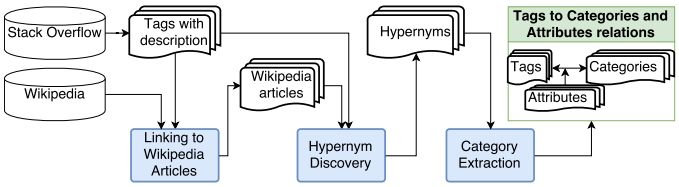

Fig. 2. Overview of the approach. The data is extracted using the Stack Overflow and Wikipedia APIs, and gradually transformed through three independent stages (in blue) to generate a set of categories and attributes.

TABLE 1
Development Sets

Set	Date	Questions	Tags	Instances
dev-1	2 May 2014	10 000	5515	29 677
dev-2	7 May 2014	100	207	303
dev-3	5 Jun 2014	500	740	1467
All SO (evaluation)	13 Aug 2014	7 818 961	37 892	23 071 454
All SO (metrics)	14 Jun 2016	11 814 545	45 476	35 159 658

as a subnode of a more specific and meaningful hypernym such as *object-relational mapping library*. In addition, previous work has proposed a tag recommendation approach for projects hosted on Ohloh and Freecode [38] and an approach to find similar applications based on the SourceForge tags [20].

On Stack Overflow, tags are mostly used to indicate programming languages, frameworks, environment concerns, domains, and non-functional issues [35]. Several approaches have been developed for recommending tags for Stack Overflow posts, including a discriminative model approach [29], a Bayesian probabilistic model [31], and an approach combining several techniques called TagCombine [41].

Using topic modeling to discover the main topics on Stack Overflow, Barua et al. found a wide range of topics and identified those that increased and decreased over time [2]. In contrast to this work, our approach automatically categorizes the software technologies represented by tags.

3 Overview of the Approach

Our approach requires downloading all the tag information from Stack Overflow (SO). We download the *excerpt* and *information page* for each tag using the Stack Exchange API.[3] The tag excerpt is a short, unformatted summary of the tag, and the information page is a more complete, html-formatted, documentation page. The tag information is user-generated and can be missing.

Once the tag data is available, our approach begins by automatically selecting the Wikipedia article (or article section) that describes the tag, if it exists (Section 4). Then, we apply a new hypernym detection algorithm to the article describing each tag (Section 5), which generates a list of hypernyms for the tag. Finally, we transform hypernyms into more abstract descriptors called *categories* and *attributes* (Section 6). The complete approach is illustrated in Figure 2, and Figure 5 (Section 6.5) illustrates the final data structure.

Our approach is heuristic and involves different threshold values as necessary components of the algorithms we developed. From the point of view of heuristic development, the thresholds fall into two categories: Some thresholds can be rationalized in terms of domain concepts (e.g., 1.0 weight for an exact lexical match). Other thresholds are necessary as a consequence of design choices, and cannot be linked to specific domain concepts. Whenever applicable, we justify the choice of threshold values, but in general the values were determined through iterative experimentations on a development set, using a hill-climbing approach. The main steps of the approach are stable with respect to the threshold values (see Section 7.3). Furthermore, the heuristics used for each sub-problem work independently, so that improving one of the heuristics or even replacing it will not impact on the others. The complete list of all the values used in each step of our approach can be found in Tables 2, 4 and 5.

We used three different sets of tags to develop our approach (dev-1,2,3 in Table 1). The sets were generated by downloading the N most recent questions on Stack Overflow (at the time of collection) and collecting all tags associated with these questions. Because these three sets are derived from recent questions, they contain mostly popular tags, thus limiting the number of tags in our development sets without any information. To evaluate our approach, we downloaded all the tags of Stack Overflow. We used all of the tags to evaluate our abstraction process from hypernyms to categories, but only the subsets described in Section 7.1 for the comparative evaluation. Finally, the metrics provided in Sections 4 and 5 have been calculated using the most recent set of all the tags from Stack Overflow at time of writing. All the sets are described in Table 1.

4 Linking Tags to Articles

In many cases, a tag has a corresponding article on Wikipedia with a title that closely matches the tag (e.g., `C++`). In such cases linking the two is trivial; however, for other cases the links between tags and articles are far less obvious.

One challenge is that some tags closely match a Wikipedia article for a different sense. For example, the article for `Maven` concerns the noun meaning an expert (the tag is really associated with the article entitled `Apache Maven`). A second challenge is that some tags, such as `curl`, could reasonably be linked to more than one computer science related article, namely `Curl (programming language)` and `cURL`. A third challenge is that some tags are only described in a section of a related article, and do not have their own Wikipedia entry. For example, the tag `catalina` refers to the section `Catalina` of the article `Apache Tomcat`. These three challenges are compounded by the fact that we cannot assume that there is a Wikipedia article or article section for every tag.

These particular challenges involve domain-specific elements related to software technologies that have not been addressed by related work. We tackle the linking problem

3. http://api.stackexchange.com/

TABLE 2
Summary of the Parameters Used in the Linking Process

Ref.	Parameter	Value
4.1 (*Analyzing*)	Unparsed section of disambiguation page	*"See also"*
4.1 (*Analyzing*)	Disambiguation page keywords	*"technology", "computing"*
4.1 (*Analyzing*)	Search results discount factor	0.8
4.1 (*Analyzing*)	Minimum score	0.24
4.1 (*Sections*)	Leading section bonus	0.1
4.1 (*Sections*)	Discounted sections penalty	0.2
4.1 (*Sections*)	Discounted sections keywords	*"Version", "History", "List"*
4.2	α (textsim weight)	12/22
4.2	β (titlesim weight)	5/22
4.2	γ (catsim weight)	5/22
4.2 (*titlesim*)	Partial match (2)	0.75
4.2 (*titlesim*)	Partial match (3)	0.25
4.2 (*catsim*)	Programming keywords	*"comput", "program", "framework", "software", "library"*
4.2 (*infobox*)	Infobox templates	*"Software infobox templates", "Computing infobox templates"*

with a search process (Section 4.1) that relies on the computation of a *similarity score* (Section 4.2) between a tag and a Wikipedia article. Overall, our process is able to find a corresponding article for 32% of the tags. We point out that this ratio is useful as a general indication of the coverage of tags by wikipedia articles, and that it does not represent a performance measure. Many tags, such as those representing a specific class like `uinavigationcontroller`, simply do not have any corresponding Wikipedia article. A meaningful way to evaluate coverage is to do so comparatively, as described in Section 7.

4.1 Search Process

On Stack Overflow, each tag has a possibly empty information page describing the tag. Missing tag information is common on Stack Overflow, which increases the difficulty of the search task: as of June 2016, 29% of the tags did not have any description associated with them, 15% only have an excerpt, and only 55% of the tags have a full information page and an excerpt. Fewer than 1% of the tags have an information page, but no excerpt. To find the correct Wikipedia article, we first look within this information page for hyperlinks to Wikipedia articles. If we find nothing, we perform a text search using the Wikipedia API[4] and attempt to find the correct article among the results. The details of each step are provided below.

Inspecting Links in Tag Information Pages

Some tag information pages contain hyperlinks to relevant resources. Sometimes, one of the links points to the Wikipedia article about the technology described. We identified four situations in which a link to Wikipedia is likely to be to the correct article. We proceed through the following cases in order and select the first match:

1) If there is a single hyperlink in the information page and it points to a Wikipedia article, we select that link.
2) We look for links in the first paragraph. If at least one exists, we select the article with highest similarity (as defined in Section 4.2).
3) If we find a single list containing a single link to a Wikipedia article, we select this article. This case usually happens when the information page contains a list of references.
4) We look for block quotes with a reference, as the description may contain text cited from Wikipedia. If the reference is given, we select that article.

Of all the tags with an information page, only about 5% contained at least one reference to a Wikipedia article. However, those references are usually accurate and using them greatly simplifies the search.

Performing a General Wikipedia Search

If the heuristics in the previous section do not find an article, we perform a general search. Because the Stack Overflow tags are often abbreviations or acronyms, and must respect a certain format,[5] we must transform them into proper query terms.

We replace every hyphen by a space and remove all version numbers. The tag `ruby-on-rails-3`, for example, becomes `ruby on rails`. We also try to find in the tag excerpt a more descriptive name for the tag. If the tag contains a version number, we look at the excerpt of the corresponding tag without any version number. We recognize three patterns where another phrasing for the tag can be found:

1) The excerpt starts with a series of capitalized words: *Windows Communication Foundation is . . .*
2) The excerpt starts with "Stands for" or "[tag] stands for", followed by a series of capitalized words: *SUP stands for Sybase/SAP Unwired Platform*
3) There are parenthesis containing a series of capitalized words after the first or the second item: *AJAX (Asynchronous JavaScript and XML) is . . .*

In each case, we ensure that the alternative phrasing consists of more than one word, and that it is longer than the normalized term. Otherwise, we reject the alternative.

Finally, we expand a few common abbreviations. These changes are hard-coded, but the list contains only five entries, all of which can be found in Table 3.

Analyzing Search Results

Once the search terms have been created, they are passed to the Wikipedia search API. For efficiency reasons, we start by looking only at the first ten results. We first try to find a disambiguation page[6] with a title completely matching the search terms or the tag within the first ten results. The next step depends on whether a disambiguation page is found.

4. http://en.wikipedia.org/w/api.php

5. Limited to 25 characters consisting of `[a-z0-9+-#.]`

6. A page containing a list of articles sharing a similar title.

TABLE 3
Replacements for Search Terms

Original Sequence	Replacement
ms	microsoft
vb.net	visual basic .net
vb	visual basic
vc	microsoft visual c++
[*single letter*]	[*single letter*] (programming)

If a disambiguation page is found we extract all articles that the disambiguation page refers to. If there is a *See also* section, we reject all articles in the *See also* section and below. If the page contains a section with the word *technology* or *computing*, only the articles in those sections and in the *leading section*[7] are extracted. The articles are compared with the tag description as described in Section 4.2 and the article with the highest score is selected.

If no disambiguation page is found we parse the first ten results to see if at least one has a title or a *redirect from* title that matches completely or partially the search terms (our definitions of *complete* and *partial* match are explained in Section 4.2). If we find some matches, we continue with them. Otherwise, we take the first 50 results. Again, this limit is set for efficiency results. Lower results are very unlikely to obtain a better score than the first ones, especially because of the cumulative discount factor we apply. We then calculate the similarity score for each article, and multiply it by a cumulative discount factor of 0.8^{r-1} where r is the rank of the article in the search results. The article with the highest resulting score is then selected. If this article has a score less than a threshold (experimentally set to 0.24), we reject it and conclude that there are no articles related to the tag on Wikipedia.

Identifying Specific Sections

Some tags are only described in a specific section of an article on a more general topic. To identify which section really represents the tag, we first look at the title of the article. If it entirely contains the tag *or* the search terms, then we simply take the leading section. Otherwise, if the title contains only a part of the tag, and a section title contains the other, this section is added to a list of candidate sections. When all candidate sections have been extracted, we select the one with the highest similarity between its first paragraph and the tag excerpt (using the text similarity definition in the following section).

Within the list of candidate sections, we boost the similarity score of the leading section by a factor of 0.1 and discount sections containing one of the words *Version*, *History* or *List* by a factor of 0.2. These factors encode the fact that the leading section usually describes the technology, and the discounted sections usually only contain meta-data about it. If no section title matches the tag or the search terms, we select the leading section.

7. The preamble section before the first titled section.

4.2 Computing the Similarity Score

The similarity score $\mathrm{sim}(t, a)$ between a tag t and an article a is computed based on five factors:

$$\begin{aligned}\mathrm{sim}(t,a) \quad = \quad & \mathrm{disamb}(a) \cdot \mathrm{infobox}(a) \cdot \\ & (\alpha \cdot \mathrm{textsim}(t,a) + \beta \cdot \mathrm{titlesim}(t,a) + \\ & \gamma \cdot \mathrm{catsim}(a))\end{aligned}$$

disamb and $\mathrm{infobox}$ are two filtering functions that remove uninteresting results by taking either the value 1 or 0. $\mathrm{disamb}(a)$ takes a value of 0 if a is a disambiguation page, and 1 otherwise. It thus filters out disambiguation pages, which are not useful for discovering hypernyms. $\mathrm{infobox(a)}$ is a more complex filter described at the end of this section. The other three linear components are all estimations of the relevance of a to t and are described below. We experimentally set the parameters as $\alpha = 12/22$, $\beta = 5/22$, and $\gamma = 5/22$.

Textual Similarity (textsim)

We calculate textual similarity based on the *Q grams* distance [13] between the first paragraph of the selected article section and the tag excerpt using words as tokens. This metric calculates the proportion of common sequences of specific length between two strings.

The similarity is calculated for the first sentence of both texts, then the first two sentences, the first three, etc., until one of the inputs run out of sentences. Each progressive subset induces a comparison unit with its corresponding *Q grams* metric and the best similarity score is kept as representative of the overall similarity between the two inputs. Finally, we return the square of the best metric value, to amplify the importance of high similarity matches.

Article Title (titlesim)

We consider how well the title of an article matches the tag and consider four cases.

1) The title *is* the search query (complete match). Returns 1. E.g. `java` completely matches `Java (programming language)`.
2) The title contains the search query (partial match). Returns 0.75. E.g. `maven` is contained in `Apache Maven`.
3) The title is contained in the search query (partial match). Returns 0.25. E.g. `microsoft sql server integration services` contains `SQL Server Integration Services`.
4) None of the above (no match). Returns 0.

The weights of Cases (1) and (4) can be trivially assigned. The rationale for the two other weights is that usually, (2) happens when the article title includes the organization overseeing the technology, e.g. `Apache Maven`, but the tag does not, e.g. `maven`, whereas (3) usually leads to articles that are too general.

All comparisons are case-insensitive. Also, if a title contains a part in parentheses to distinguish its subject from another with the same name, this part is removed before the comparison (e.g., `Hibernate (Java)` becomes `Hibernate`). Titles that redirect to an article as well as its title are compared to the tag and the best match is kept.

Categories (catsim)

Each Wikipedia article belongs to a number of categories (found at the bottom of the page). We analyze the categories to estimate the relevance of the article to software technologies. Each category c receives an individual score, computed as:

$$\text{score(c)} = 0.1 \cdot \text{title(c)} + 0.9 \cdot \frac{|\text{programming(members(c))}|}{|\text{members(c)}|}$$

where *title(c)* returns 1 if the title of category c contains at least one *programming keyword*,[8] and 0 otherwise, *members(c)* is the set of articles that are members of category c, and *programming(A)* is the subset of the set A of articles that are related to programming. We have $a \in \text{programming(A)}$ iff $a \in A$ and either the title of a is in CamelCase or it contains a programming keyword (footnote 8). The total score for an article a with a set C of visible[9] categories is calculated as:

$$\text{catsim}(a) = \min\left(0.25, \frac{\sum_{c \in C} score(c)}{|C| + 1}\right)$$

We add 1 to the denominator to give more importance to articles with more categories. We also take a ceiling of 0.25 for this value to avoid the noise caused by outliers.

Infobox

Infoboxes are tables with concise information in the form of keys and values that sometimes appear at the beginning of an article. Of all the articles parsed in our process, 48% had at least one infobox. Of those 48%, a small minority (7.5%) had more than one infobox. Infoboxes are based on some *templates*, which are assigned categories just as the articles. We manually selected two categories containing the infobox templates related to computing subjects: `Software infobox templates` and `Computing infobox templates`. Those two categories were selected by looking at the structure of the infobox templates and categories.[10] If an article has an infobox, and its template is not in the specified categories, we return $\text{infobox}(a) = 0$, otherwise, we return 1. If a section was selected for an article, we consider only the infoboxes in the selected section, if there are any. If an article has more than one infobox inside the targeted section, the first one is used. In our data set, this filter allows us to reject 80% of the articles with at least one infobox as inapplicable.

5 Hypernym Discovery

Once a Stack Overflow tag is linked to its corresponding Wikipedia article, we parse the article to discover hypernyms for the tag. We use three techniques that target different sections of the article and can be used independently. Each technique can return zero, one, or multiple hypernyms for any tag.

8. *comput*, *program*, *framework*, *software* or *library*

9. Some categories representing meta-data remain hidden.

10. A list of all infobox templates and infobox template categories can be found at https://en.wikipedia.org/wiki/Wikipedia:List_of_infoboxes

TABLE 4
Summary of the Parameters Used in the Hypernym Discovery Heuristics

Ref.	Parameter	Value
5.2	Infobox templates	*"Software"*, *"Programming language"*, *"OS"*, *"File format"*
5.2	Infobox keys	*"genre"*, *"paradigm"*, *"family"*
5.3 (*first*)	Phrasal groups	Noun, adjectival, prepositional, conjunction, single words
5.3 (*second*)	Grammatical relations	*poss, possessive, amod, mod, nn, det, predet, pobj, advmod, number*
5.3 (*second*)	Splitting relations	*conj, apos, dep*

The first technique uses the internal hyperlinks (wikilinks) in the article. The second relies on the infobox values, if there is one. The third takes the first sentence of the article and employs Natural Language Processing (NLP).

We did not use the Wikipedia categories attached to the articles as hypernyms for two reasons. First, the idea of a category is more general than the hypernym relation. For example, Wikipedia contains many categories of the type *Programming languages created in YYYY* or *Free Software programmed in XX*. Those categories represent different types of metadata rather than hypernyms. Second, the WiBiTaxonomy Project already attempts to extract hypernyms based on both the articles and categories linked to a Wikipedia article, and we found that our approach achieved higher coverage at a lower false positive rate (see Section 7).

5.1 Wikilinks

This technique involves inspecting the first wikilinks of the article. If the first wikilink is not in the first sentence or if the first sentence is not in the form *[Subject] is [a—an—the] [description]...* [11], the technique fails, as the pertinence of the link becomes unpredictable.

If there is at least one link in the first sentence, which is the case for 93% of the articles selected during the linking process, we use the following rule. The first link is taken, as well as every subsequent link that is separated from the previous one only by spaces, commas, conjunctions (*and*, *or*), and articles (*a*, *an*, *the*). From the selected links, we remove those pointing to an article with a title consisting of only one word. This particular decision is motivated by the observation that many Wikipedia articles related to technologies and consisting of only one word concern specific technology products, such as `Hibernate (framework)` and `JSON`, whereas broader concepts are usually expressed with at least two words, e.g., `Software framework` and `Programming language`. These single-word links often appear as attributes to the hypernym, rather than the hypernym itself, as in *JSON-based open standard*, when using this method. Therefore, we excluded them to reduce the number of false positives. Naturally, this heuristic can also remove valuable links, but given the different sources of hypernyms included in our approach, we opted for a more conservative.

11. As required by the Wikipedia Manual of Style. Over 75% of the articles selected during the linking process followed this recommendation.

For example, the first sentence of the article `Java servlet` is *A java servlet is a Java program that extends the capabilities of a server.* In this sentence, *Java* links to `Java (programming language)`, *program* links to `Computer program` and *server* links to `Server (computing)`. The hypernym *Computer program* is returned, but *Java* is rejected because it consists of only one word, after removing the disambiguation terms in parentheses, and *Server* is rejected because it is not among the first group of wikilinks.

The hypernym candidates returned with this technique are the *titles* of the Wikipedia articles referred to by the links, and not the text of the link. In addition to individual hypernyms produced for each link deemed valid as explained above, we also produce an aggregated hypernym for the entire list of links (if there is more than one link). In this case, the aggregate hypernym includes all words in the *text* of the links. In the example above, the hypernym *java program* would also be returned.

5.2 Infobox Values

This technique involves parsing the infobox of the article. If there is more than one infobox, only the first one is used. If there is no infobox, the method returns nothing. For this technique, we manually determined the most pertinent infobox templates and keys. If the key is not present, or if the infobox template is not in the list of selected templates, we return the template name of the infobox as a hypernym. This template name is usually general, but accurate. Otherwise we return a normalized version of the value for selected keys as hypernyms. The normalization consists of straightforward transformations to split all items in an enumeration, and remove markup and elements such as hyperlinks and footnotes. We selected `Software`, `Programming language`, `OS`, and `File format` as the relevant templates, and `genre`, `paradigm`, and `family` as the relevant keys. These infobox templates are the most commonly seen among the technology-related infoboxes, and were selected by looking at the list of infobox templates (footnote 10). For the `Programming language` and the `OS` infoboxes, we append the title of the infobox at the end of each hypernym, because the values are typically adjectives. If a key is present, then its value is parsed and formatted before being returned as a hypernym.

5.3 Natural Language Processing

The final technique is to use NLP to decompose the first sentence and group only the most pertinent words into a hypernym. We assume that this sentence is in the format *[Subject] is [a—an—the] [description]...*, although slight variation can still be accepted if parsed properly. Otherwise no hypernym is returned. Within this technique, we implemented two different heuristics. Those heuristics are based on the different outputs of the *Stanford Core NLP* library for Java [21].

The NLP heuristics are also applied to the tag excerpt obtained from Stack Overflow.

Phrasal Group Approach

The first heuristic starts from the whole sentence and reduces it to a single hypernym by removing extraneous phrasal groups. A phrasal group can be a noun phrase, a verb phrase, etc. Phrasal groups are made of single words and other phrasal groups. For this heuristic, we start with the first largest verb phrase of the sentence where *is* is the main verb, and we remove all extraneous phrasal groups. Extraneous groups are those that are not single words, noun phrases, adjectival phrases, prepositional phrases or conjunction phrases. Those phrasal groups were chosen to represent those that would most likely compose a valid hypernym. For example, we did not include verbal groups, as a hypernym does not normally contain a verb. We then recursively parse all remaining phrasal groups, removing extraneous groups at each step.

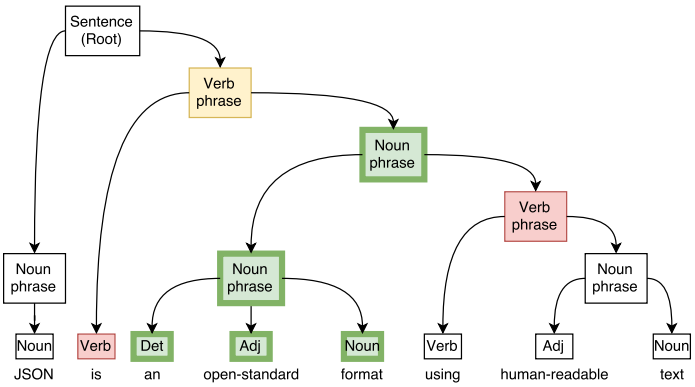

Fig. 3. An example of the Phrasal Group approach. The input sentence was *"JSON is an open-standard format using humain-readable text."* The structure of the phrasal groups is shown as returned by the parser. The analysis starts at the first verb phrase (in yellow), and parses the tree, keeping only the relevant phrasal groups (in green with thick border, the groups kept, in red with slim border, the groups rejected).

Consider for example the sentence *"Java is a runtime environment and an object-oriented programming language developed by Oracle."* Assuming the sentence is correctly parsed by the NLP engine, this heuristic would begin by taking the outermost verbal phrase consisting of everything except *Java*, and then examine the inner phrasal groups. Directly inside the verbal group are the two noun phrases linked by a conjunction. The first noun group, *a runtime environment*, would be retained in its entirety, because all inner groups are nouns and adjectives. The second group would be again examined, and the verbal phrase *developed by Oracle* would be removed, since it is not one of the target group types. The returned candidate is therefore *a runtime environment and an object-oriented programming language*. Figure 3 shows another example of this approach.

Grammatical Relation Approach

The second heuristic is to consider the grammatical relations between the words in the sentence and, starting from the word identified as the root of the sentence, adding all words related by some chosen grammatical relations. We then take all new words, and look for more words related to them with the same relations. When there are no more words to add, the selected words are put in the same order as they were in the sentence. Some of the relations, such as the coordinating relations, will split the output, generating new hypernyms. If this happens, all previous words in the group are copied over to the other, and we continue with

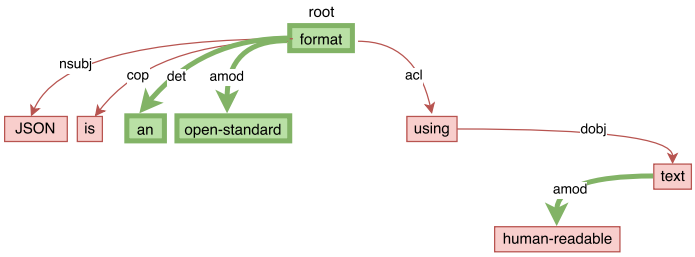

Fig. 4. An example of the Grammatical Relation approach, using the same sentence as in Figure 3. The approach starts with the root word and includes all words related by one of the chosen relations (in green with thick border, the words kept, in red with slim border, the words rejected).

TABLE 5
Summary of the Parameters Used to Group the Hypernyms into Categories

Ref.	Parameter	Value
6.1	Acronyms	see Table 7
6.3	N-gram popularity	1.1%
6.3	Neighbours popularity	20%
6.4	*vSingle* z-score	2
6.4	*vInFront* z-score	2
6.4	Nb. of occurrences of potential attributes	4
6.4	Nb. of occurrences of potential category descriptors	3
6.4	Proportion of popular hypernyms for successive iterations	10%-20%-30%-40%

all groups. The grammatical relations [9] accepted are `poss`, `possessive`, `amod`, `mod`, `nn`, `det`, `predet`, `pobj`, `advmod` and `number`. The splitting relations are `conj`, `apos` and `dep`.

Using the previous example sentence (*"Java is a runtime environment and an object-oriented programming language developed by Oracle."*), this heuristic would first select the root word, in this case *language*. This word is linked with *a* by the relation `det`, *runtime* by the relation `amod` and *language* by the relation `conj`. Therefore, those words are kept, but the output for *language* will create a distinct hypernym. Next, the words *a runtime environment* do not have any more additional relations, but *language* is linked to *an*, *object-oriented*, *programming* and *developed* by the relations `det`, `amod`, `amod` and `act` respectively. Thus, we keep *an object-oriented programming language*, but we drop *developed*. Because of the conjunction, there are two resulting hypernyms: *a runtime environment* and *an object-oriented programming language*. Figure 4 shows another example of this approach.

6 Extracting Categories

The hypernym detection phase produces, for a given tag, a list of candidate hypernyms. However, many raw hypernyms are semantically related variants of a general concept and distinguishing between them complicates the tracking of equivalent technologies. We address this issue with an algorithm that abstracts hypernyms into a *category*, and retains the *attributes* associated with the category–tag relation.

TABLE 6
Sample Hypernym Detection Output

	Tag	Detected Hypernym
1	html	markup language
2	python	imperative and functional programming language
3	objective-c	general-purpose programming language
4	objective-c	object oriented programming language
5	ruby	general-purpose open-source dynamic object oriented reflective programming language
6	java	open-source, dynamic, reflective, object oriented, general-purpose programming language
7	java	class-based, object oriented, strongly typed, reflective language and run-time environment
8	silverlight	run time environment and multimedia framework
9	django	open source web application framework
10	xcode	integrated development environment
11	phpstorm	commercial ide for php
12	phpstorm	commercial php integrated development environment

6.1 Importance of Grouping Categories

Table 6 shows some examples of hypernyms discovered, with their corresponding input tags. Even this small sample illustrates the challenges of interpreting hypernyms.

First, **attributes introduce variants of a concept** represented by a hypernym. For example, in Table 6, five hypernyms (2, 3, 4, 5 and 6) mention *programming languages*. On one hand, grouping these would lose the distinction between different types of programming languages. On the other, not grouping them would mean these tags would not share a hypernym, making direct comparisons difficult. Simply grouping by common terms is also problematic: some tags are said to be *open source* (5, 6 and 9), but creating an *open source* category would not respect the hypernymy relation.

Second, some **compound hypernyms** represent multiple concepts. For example, hypernym 7 implies that `java` is both a *programming language* and a *run-time environment*. In this case, the hypernym should be split around the word *and*. However, hypernym 2 is not in this situation, even if it contains the word *and*. In this case, the conjunction connects two attributes of the concept *programming language*.

Third, instead of the complete term, a hypernym will sometimes **contain an acronym** that needs to be expanded. We manually created a small list of common acronyms to achieve this goal. The Table 7 shows the entire list. The first entry is necessary to prevent incorrectly replacing the sequence *os x* with *operating system x*.

Finally, some hypernyms differ only in **word order** (e.g., hypernyms 11 and 12, after expansion).

6.2 Hypernym Tokenization

The first step for abstracting hypernyms into categories is to tokenize and normalize the hypernyms, which we do through straightforward transformations such as putting all words in lower case, normalizing punctuation, removing disambiguation terms in parentheses, transforming plural words to the singular form, and changing the British ending

TABLE 7
List of Computer Science Acronyms

Acronym	Replacement
os x	osx
os	operating system
ide	integrated development environment
dbms	database management system
rdbms	relational database management system
api	application programming interface
ui	user interface
gui	graphical user interface
sdk	software development kit

our to the American *or*. Some of the steps are assisted by the use of the Porter stemmer.[12]

6.3 Detecting Compound Terms

We also detect *compound terms* (e.g., *open source*) to discriminate the main concept represented by a hypernym from attributes, and to distinguish attributes from each other.

First, all originally hyphenated word sequences are simply aggregated back into a compound term. This has the added benefit of normalizing hyphenation. Hence, *open-source* and *open source* will both be considered as a single term.

We then use a statistical approach to detect compound terms. For this purpose we look at common n-grams (bigrams and trigrams only) among the set of all hypernyms detected for all Stack Overflow tags. This approach is divided into two steps, the first one based on global n-gram frequencies, and the second one on relative frequencies.

We first compute the number of occurrences of each n-gram and calculate its global popularity by dividing this number by the sum of occurrences of all *n-grams*. If this ratio is above an experimentally-determined threshold (0.011), the n-gram is considered to be a compound term. However, if two or more n-grams have one or more words in common, they are not accepted as compound terms. For example, if 10 000 n-grams were formed from all the hypernyms, and the most common ones are *open source* (200 occurrences), *java library* (150 occurrences), *python library* (120 occurrences) and *runtime environment* (100 occurrences), the first bigram (open source), would be aggregated, but the following two would not, since they share a common term (library), and the other ones would also be left as separate terms since they do not occur often enough. Once compound terms are detected, we repeat the process, treating all compound terms as a single term, until a fixed point is reached.

The second step detects compound terms based on relative frequencies. For each ordered pair of words (v, w), we count the number of occurrences of v immediately followed by w in the set of all hypernyms. We then compute two ratios: the sum of occurrences divided by the number of appearances of v in all hypernyms, and the sum of occurrences divided by the number of appearances of w. If both ratios are above 0.20, we consider the pair to form a compound term. This method is helpful to detect unpopular compound terms. For example, *file format* may not be detected by the previous heuristic if these terms do not appear in many hypernyms, but if *file* is followed by *format* at least 20% of the time, and similarly *format* is preceded by *file* at least 20% of the time, they will be aggregated. As for global n-grams, we repeat the process until a fixed point is reached.

12. http://tartarus.org/martin/PorterStemmer/def.txt

6.4 Discriminating Category Names

To discriminate the category names from attributes, the principle we follow is to discover, from the words used in hypernyms, those that *usually* act as attributes from those that *usually* describe a category. This distinction is different than the one that a part-of-speech (POS) tagger would make. Considering the hypernym *a php integrated development environment*, the POS tagger would normally tag *php*, *development* and *environment* as nouns and *integrated* as a verb (or an adjective). However, in our approach, we would like to infer that *integrated development environment* is usually a category descriptor (acting like a noun) and *php* is usually an attribute (acting like an adjective).

To implement this principle, we start by obtaining a small set of terms to be analyzed by a first independent heuristic, then use them as the initial seeds for a recursive heuristic that is executed until a fixed point is reached. Since our recursive heuristic can reach a low fixed point quickly, we re-seed it using progressively more terms. Each step is explained in details below.

The first seeds consist of the set W_s of all hypernyms that consist of only one word (counting compound terms as one). To analyze this group, we then calculate two values for each $w_{s,i} \in W_s$: the number of tags that have this single-term hypernym (*vSingle*), and the number of hypernyms in the set of all hypernyms where $w_{s,i}$ appears anywhere but at the last position (*vInFront*). We convert both values to z-scores. Words in W_s with a *vSingle* z-score above 2 are then assumed to be *category descriptors*, whereas words with a *vInFront* z-score above 2 are classified as *attributes*. If both z-scores for a word are above 2, the word is classified as an attribute. In all cases, we used a z-score limit of 2, which is a commonly agreed threshold for detecting extreme values (outliers).

Those preliminary lists of category descriptors and attributes are then used as seeds in an iterative process to find new category descriptors and attributes. The iterative process consists of expanding the list of attributes and category descriptors by identifying two situations where terms are likely to be category descriptors or attributes. First, we look at all hypernyms that contain a term from the preliminary list of category descriptors. If the term is in the last position, all terms before it are added to a list of potential attributes. After inspecting all hypernyms, the potential attributes that had been detected more than four times are added to the list of attributes. Then, for all attributes in the list of attributes, if the attribute can be found in a hypernym that consists of only two terms, where it is the first term, the second term is considered to be a potential category descriptor. After all hypernyms have been inspected, the potential category descriptors that had been detected at least three times are added to the list of category descriptors. If a word meets the criterion for the attributes as well as the category descriptor, the priority is given to the category descriptor. This step is repeated until a fixed point is reached.

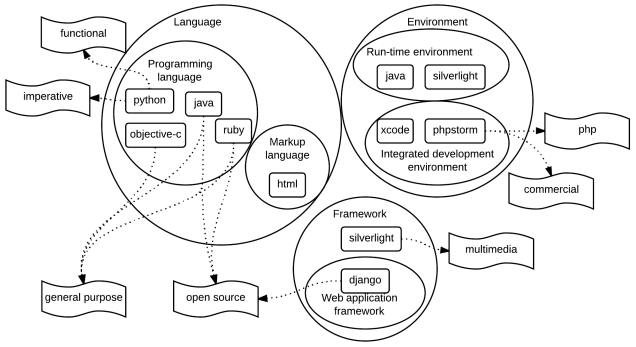

Fig. 5. Representation of the output for the information in Table 6. The categories are represented by circles, the tags are the rounded rectangles, and their attributes are the wavy rectangles. A tag inside a category means that the tag belongs to this category, and an arrow from a tag to an attribute means that the instance of the tag has that attribute. Some attributes have been left out for clarity.

After the fixed point is reached the entire heuristic is re-seeded, with a certain proportion of the most popular terms in the corpus of all hypernyms. Any already-processed term is removed from the seed set. We do four additional iterations in this way, starting with the 10% most popular terms, and adding the following 10% at each iteration. At the end of this process, many words will not have been classified. However, most of the hypernyms will contain at least one classified word, which is what we need for the next step.

6.5 Creating the Category Structure

The final step of our approach is to create the category structure (e.g., Figure 5). This structure consists of a series of *categories*, each containing the related tags and their attributes. A tag can be found in more than one category. Each tag in a category can have different attributes. If a tag is found in more than one category, it can have different attributes in each category. Finally, some categories are a subset of larger categories.

Due to the compact nature of typical hypernyms, a typical part-of-speech (POS) tagger does not return high quality POS tags. Therefore, we implemented our own algorithm to extract categories and attributes from hypernyms. The algorithm is based on the general shape of the sequence of terms in a hypernym, with the following building blocks: Attribute (A), Category Descriptor (C), unclassified term (U), special terms (S). The list of special terms is limited to the four terms: *for*, *of*, *and* and *or*. The shapes are defined as regular expressions.

The algorithm consists of a series of consecutive rules that are applied to the hypernyms. Each rule is composed of a shape and an action. They are presented in Table 8, with some examples, and the actions are explained in Table 9. The input to a rule is a sequence of terms (originally, the hypernym), and the output is a possibly empty list of transformed sequences of terms, unless otherwise specified. If the input of the rule matches its shape, the action is applied to the input, and the result is given to the next rule, unless otherwise specified. If the input does not match the shape, it is handed over to the next rule. The first six rules deal with the special terms, and the last two distinguish the category descriptor from the attributes. The final output is a list of category descriptors and their corresponding list of attributes.

When the category descriptors and the attributes have been distinguished, we assemble the category structure based on the category descriptors. If the category descriptor contains multiple words, we create a category name for the rightmost word, and for each preceding word, we create a specialized sub-category. For example, the category *web application framework* is a sub-category of *application framework*, which is a sub-category of *framework*.

When the hierarchy is completed the tags originally used to generate hypernyms are simply associated with their most specific category. We complete the process by scanning for tags linked to both a sub-category and some of its parents. This situation can happen because multiple hypernyms are produced for a tag. In such cases, we remove the association between a tag and the super-category to preserve the internal consistency of the structure. By default, the tag will transitively be a member of the parent category.

7 Evaluation

The output of Witt is a discovered attributed category structure for a very large number of tags. Moreover, the categories discovered by our approach have technically open-ended extensions. For these reasons, it is not possible to compare the output with any specific oracle. Instead, we evaluate the approach by comparison, and decompose the evaluation to individually target the two major steps on which the quality of the results is dependent: the ability to provide good hypernyms for each input term, and the ability to group similar hypernyms together. Specifically, we sought to answer those two questions: *how does the performance of* Witt *compare with existing taxonomy tools?*, and *how effective is the new hypernym abstraction phase for grouping equivalent technologies?* In our evaluation, we considered tags in Stack Overflow as proxies for software terms that might be of interest to software development stakeholders. This assumption is not unrealistic, as Stack Overflow tags form a vocabulary of over 45 000 programming related terms, as of July 2016.

7.1 Comparative Evaluation

We compared Witt with four taxonomy tools to determine how well they could extract valid hypernyms for software technologies. We found four active projects offering comparable functionality: *WordNet* [24], *DBpedia Spotlight* [22], *WiBiTaxonomy* [12] and *THD* [10] (see Section 2). Those were all available hypernym generation tools we could find at the time we conducted the experiment. To our knowledge, prior to Witt, there was no domain-specific tool for programming terms. We also compared Witt with *Google*'s definitions to make sure a simple automated Google search would not outperform our approach.

To conduct the experiment, we extracted a sample of the tags from Stack Overflow stratified by popularity. We provided the tags as input to each tool under evaluation,

TABLE 8
Rules used to extract the categories from the hypernyms.

Rule	Shape	Action	Example
1	`.*(for.*){2,}`	Stop.	`a software for creating access for users`
2	`[^for]*for[^for]*`	Split on *for*, with f_1 being the main fragment.	`commercial ide for php` Continue with `commercial ide` and `php` Outcome : `ide` as the category descriptor, `commercial` and `(for) php` as the attributes.
3	`.*((and|or|of).*){3,}`	Stop.	`a family of databases and servers of high quality`
4	`[^S]*of[^S]*(and|or)[^S]*`	Split on *of*, and split f_2 into $f_{2,1}$, $f_{2,2}$, ... using Rule 5. Append f_1 to each of the $f_{2,i}$ and continue with these fragments.	`implementations of java, ruby and python` Continue with `implementations of java`, `implementations of ruby`, and `implementations of python`.
5	`[^S]*(and|or)[^S]*of[^S]*` or `[^S]*((and|or)[^S]*)+`	Split on every *and*, *or*, and comma.	`programming language and runtime environment` Continue with `programming language` and `runtime environment` independently.
6	`[^S]*(of[^S]*){1,2}`	Split on every *of*, with f_2 being the main fragment.	`a large set of open source libraries of mathematics` Continue with `a large set`, `open source libraries`, and `mathematics`. Outcome : `libraries` as the category descriptor, `open source`, `set (of)`, and `(of) mathematics` as the attributes.
7	`A+`	Return all words as attributes, with no category descriptor. Thus, the tag and the attributes are associated with the root category.	`open source object oriented general-purpose`
8	Any shape	Return as category descriptors the last term and all immediately preceding terms that were marked as category descriptors. All of the other terms are returned as attributes, except for determinants (e.g., *the*), which are removed.	`an application framework popular feature` Outcome : `feature` as the category descriptor, `application framework` and `popular` as the attributes.

TABLE 9
Explanation of the actions taken by the rules in Table 8.

Action	**Explanation**
Split on ...	Split the input into fragments f_1, f_2, ... Unless otherwise specified, each fragment continues with the remaining rules independently. If a *main fragment* is designated, at the end of the process, the category descriptor found in the other fragments are added as attributes to the main fragment. The special term on which the Split action is done is added to the attribute for grammatical consistency, but the attribute will be considered the same with or without this special term. Only the possibly expanded result of the main fragment is returned. Otherwise, the result of each fragment is returned.
Stop	Stop the process and return the whole input as a category descriptor which will be placed directly under the root (unlabelled) category. This category descriptor will not be transformed into an attribute if the input is a fragment from a previous Split action.
Return ...	Stop the process and return the indicated category descriptor and attributes.

and collected the results. The results were then manually validated by the second and third authors as correct (the result is a valid hypernym for a software technology) or incorrect (it is not).

Typical evaluations report on the precision and recall of a given method. However, given the nature of our situation,

and the diversity of tools we used, these metrics would not properly measure the performance of our approach.

Recall: To evaluate recall, one must know the full set of expected outcomes of the evaluated tool. In our case, it would be impossible to determine the set of all hypernyms of a term, first because such a taxonomy does not exists (which is one of the motivation of Witt), and second because the many possible variations of a natural language phrase further increase the ambiguity of recall. For example, possible hypernyms for the term `Java` include:

1) multi paradigm concurrent object oriented general purpose programming language
2) object oriented programming language
3) general purpose programming language
4) multi paradigm programming language
5) concurrent programming language
6) programming language

Although this list is far from exhaustive, we can see from this example that all of these hypernyms overlap to some extent, and common hypernym tools (which we experimented with as reported on in the paper) are simply not designed to return exhaustive combinations of phrases.

Precision: In contrast to recall, we can actually compute precision because both the numerator and denominator are known. The problem in this case is that the resulting number is heavily subject to noise, so not acceptably meaningful. Using the same examples of hypernyms for `Java`, we can see that a tool could produce exponentially more hypernyms by generating minor variations of the same concept (#1 adds no information to #2-5). So, let's assume that the list returned for Java is the most specific term (item 1) and "island in the Pacific". Here for our purpose (categorizing software technology), the precision would be $1/2 = 50\%$. But then any tool could be tweaked to spuriously return variants of the output terms. For example, we could add, without increasing the amount of information, items 2 and 3 in the list. Then precision would jump to $3/4 = 75\%$. So in the end precision is not a proper performance measure, but an artifact of design decisions in individual tools, and as the example illustrates, in this scenario a tool with a higher precision would be, actually, less precise.

Alternatives: To replace these metrics, for each tool, we measured *coverage*, and *relative number of false positives* (defined below).

Extracting the Sample

As explained in Section 3, we were conscious that taxonomy tools in general work better on popular concepts. We accounted for this assumption by partitioning the population of all tags into three groups, *popular*, *common* and *rare*, and stratifying our sample accordingly. We defined popular tags as tags that had been used on Stack Overflow at least 10,000 times, common tags as having been used between 100 and 9,999 times, and rare tags as everything else. We obtained the entire tag data from Stack Overflow on August 13, 2014. At that point there were 301, 10 381 and 27 208 popular, common and rare tags, respectively.

We then randomly sampled tags in each strata of the population. The size of each subsample was computed so that ratios observed for the subsample would have a confidence interval of 5% at the 0.95 level. Our sample thus consisted of 167 popular tags, 370 common tags, and 379 rare tags, for a total of 918.

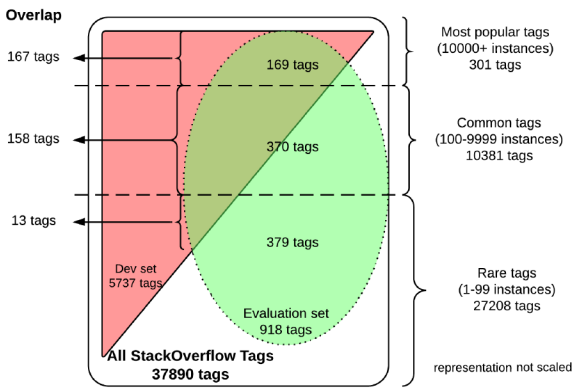

Fig. 6. Evaluation samples

To fulfill our sampling logic, it was not possible to exclude the tags in the development set. Given the size and selection strategy employed for the development sets (see Section 3), excluding the tags in the development set would have meant excluding almost all of the popular tags and half the common tags. Figure 6 shows the population stratas and corresponding subsamples. In this case, the threat of evaluating development artifacts is mitigated by the complexity of the approach, the size and diversity of the development set, and the statistical representativeness of the sample.

Obtaining Hypernyms

We provided all the tags in our sample as input to the six tools under evaluation. Because the comparison tools are not domain-specific, we injected additional information to contextualize the query to the programming domain. This made the tools perform better, resulting in a fairer comparison. We provided the contextual information in two different ways. For some tools, we appended the term *programming* to the input tag. For others, we used a list of the most common programming word stems containing the top words in each of the 40 top topics discovered by applying LDA to Stack Overflow data, taken from the work of Barua et al. [2]. The following sections detail the exact procedure employed with each tool to obtain output hypernyms for sample tags. All procedures described were fully automated.

WiBiTaxonomy: The WiBiTaxonomy project (WiBi) [12] created a directed hypernymy graph from all of Wikipedia articles and categories, using NLP techniques and the existing links between articles and categories. WiBi relies on the same assumption that the first sentence of a Wikipedia article defines the subject of the article.

Because the taxonomy is created between Wikipedia pages only, it requires an existing Wikipedia article as input. To get this article, we used the Wikipedia search engine, providing the tag as the input, and taking the first article containing at least one of the programming stems in its leading section. If we found no article, we considered that this tool would not return any hypernym for this tag. If an

article was found, we gave it to WiBi as input. The output of the tool is a set of articles and categories. We took the titles of all articles and categories as hypernyms. The title of the article given as input, however, was not considered as a hypernym. We used the default values for the two parameters of the approach: a maximum article height of 2 and a category height of 3.

THD: Targeted Hypernym Discovery (THD) [10], [16] uses hand-crafted lexico-syntactic patterns to discover hypernyms from targeted knowledge sources (Wikipedia articles). It automatically detects named entities from an input text, and returns a list of hypernyms for each entity detected.

For our search, we gave one tag at a time to THD. We did not append the word *programming* at the end of the tag, because preliminary experimentation with THD showed that this actually lowers the quality of the results because *programming* is then extracted as the entity (instead of the input tag). THD offers many options. We selected Live Wikipedia as the knowledge base, which gave the best results in the preliminary experimentation, we chose to extract all entities (named entities and common entities), and selected all sources. If the tag was extracted as an entity, we took all hypernyms returned by THD. Those hypernyms were from three sources: THD, DBpedia and Yago.

WordNet: WordNet [24] is a lexical database containing, among other information, hypernymy and hyponymy relations. The database was manually crafted, and is considered a golden standard in many linguistic applications.

With WordNet, we first retrieved all words that matched the tag. We removed a result if its gloss did not contain at least one of the programming stems. For all the remaining results, we analyzed the words listed as the hypernyms of the result. We considered all of those hypernyms for the evaluation.

DBpedia Spotlight: DBpedia [3] is a crowd-sourced database populated by structured information from Wikipedia. The DBpedia entries contains, among other information, hypernymy relations. DBpedia Spotlight is a tool for annotating text documents with DBpedia entries. It takes as input a free-formed text, and automatically extracts DBpedia entries.

We provided as input a piece of text composed of the tag and the word *programming*. If an entity was extracted, we verified that it covered the tag, and not the injected term "programming". Then, we retrieved the DBpedia resource, and took all of the terms listed under the official *dcterms:subject* key [8].

Google: The Google search engine defines a *define:* operator. When used, the search engine will try to find a definition of the word following the operator. If at least one definition is found, it will appear in a specific box on top of the web links.

We performed the search on Google using the *define:* operator. If some definitions were found, we used those definitions as hypernyms, but only if the tag was parsed as a noun. We only used the numbered definitions, and not the variants or the examples.

Witt: For the evaluation, we considered three variants of our approach. One variant returns only the raw hypernyms extracted as described in Section 5 (Witt_H). Another variant returns only the names of the general categories, without any attribute attached, and considered the returned categories to be hypernyms (Witt_C). The third variant returns, for a given tag, the corresponding category and all attached attributes (Witt_{CA}). Those variants were needed to answer the second question: *how effective is the new hypernym abstraction phase for grouping equivalent technologies?*

Evaluating the Output

Applying the tag sample to all tools produced 8964 tag–hypernym pairs that needed to be validated, following the logic *is the hypernym returned a true hypernym of the tag in the sense of software technology?* The last two authors acted as judges of the validity of a hypernym. The first author independently compiled all the results and provided each evaluator with a list of tag–hypernym pairs ordered alphabetically. This way, it was impossible for the evaluators to know which tool had produced which pair. Furthermore, 841 unidentified pairs were given to both evaluators, to support an assessment of the evaluator's agreement. The evaluators gave one of four values to each pair: *correct* (and in a sense related to software), *incorrect* (not a hypernym in any sense), *alternate* (a correct hypernym but for a sense unrelated to software), and *general* (an overly general term, such as *concept*, that could be a hypernym for almost anything).

We emphasize that we were interested in learning whether the tools could explain the software technology sense. For a developer interested in build tools, it is not useful to compare `Ant` with other insects. As a consequence, some tags could not receive anything better than the *alternate* mark since they are not directly related to software. For example, a `histogram` is a statistical concept, and it does not have a software specific sense.

The evaluators worked independently and followed an explicit evaluation guide (see the on-line appendix, the reference is in the last paragraph of the introduction). Evaluating hypernyms is a relatively low-subjectivity task, and the evaluators completed the task with a Cohen's kappa agreement of 0.721 if we consider only the binary split *correct* vs. everything else. According to a common scale this agreement can be considered substantial [18]. All disagreements were consensually resolved to produce the final benchmark.

Results

We measured two aspects of the performance of each tool: the proportion of tags for which at least one *correct* hypernym is found (*coverage*), and the average number of wrong hypernyms (*incorrect*, *alternate*, or *general*) returned by the tool for a tag (*average number of false positives*). The first measure provides a sense of the breadth of the terminology spectrum that can be handled by a tool, while the second is intended to capture the usability degradation caused by false positives. Table 10 provides the complete results, organized by subsample and by tool. The best performance for each subsample is in bold.

We find that the variants of Witt performed better than all of the other tools for both metrics except in two cases, where Google produces fewer false positives for common and rare tags (at the cost of very low coverage). Given the 5% confidence interval, the superiority of Witt for the

TABLE 10
Evaluation Results

	Popular tags	Common tags	Rare tags
Proportion of tags with at least one correct hypernym			
Witt_H	**0.562**	0.273	**0.169**
Witt_{CA}	0.538	**0.292**	0.161
Witt_C	0.414	0.173	0.084
DBpedia	0.391	0.122	0.066
WiBi	0.379	0.157	0.158
THD	0.172	0.030	0.037
Google	0.118	0.041	0.013
WordNet	0.024	0.022	0.005
Average number of wrong hypernyms per tag			
Witt_H	**0.710**	0.468	0.311
Witt_{CA}	0.822	0.668	0.385
Witt_C	1.030	0.838	0.475
DBpedia	2.373	1.186	0.712
WiBi	3.793	3.078	2.937
THD	1.071	0.484	0.380
Google	0.793	**0.395**	**0.182**
WordNet	2.396	1.389	0.459

coverage metric is statistically significant for both popular and common tags.

To get a better idea of the relative performance of the tools taking into account both aspects under evaluation, Figure 7 plots the performance of the tools in two dimensions. For this graph, we combined the three subsamples with a linear extrapolation that takes into account the relative sizes of the population stratas.

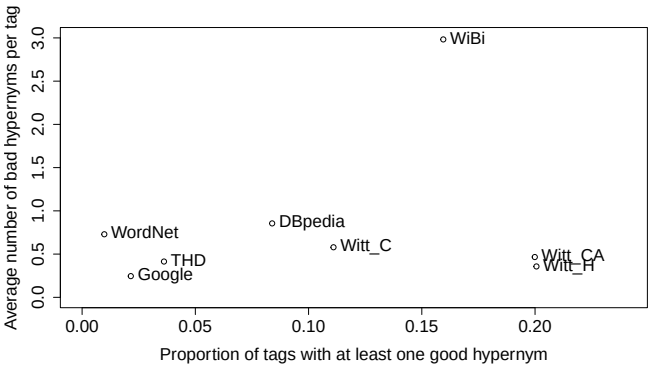

Fig. 7. Extrapolated aggregated performance

In the evaluation, we see that Witt_C generally produces worse results than Witt_H and Witt_{CA}. We explain this difference by the fact that category names were usually considered too general (e.g., *library*). However, the attributes, added in Witt_{CA}, provide major coverage improvements.

7.2 Categories

A major contribution of this work is the algorithm we use to abstract hypernyms into general categories (Section 6). We sought an estimate of the value of this approach. We saw in the previous section that Witt_H and Witt_{CA} show very similar performance, which confirms that the categories we abstract are also valid hypernyms. However, categories

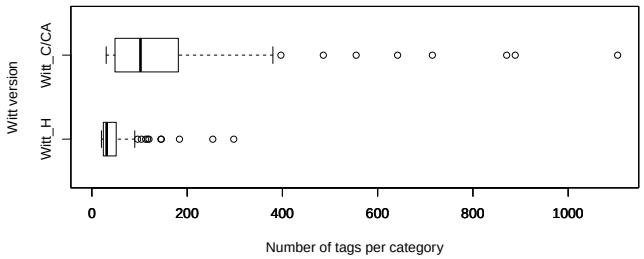

Fig. 8. Size of the 100 largest categories.

are much more useful than raw hypernyms, because they support automatically grouping equivalent technologies to produce analyses such as those showcased in Figure 1.

As a measure of the aggregating power of the category structure, we computed the membership size of each category for all variants of Witt. We considered each hypernym as a category for Witt_H, and only the category names for Witt_C and Witt_{CA}. Therefore, the categories are exactly the same for the last two versions. We used the complete output for all of the Stack Overflow tags for this comparison (as opposed to the limited number in the sample).

First, we found that after transforming hypernyms into more abstract categories, the total number of categories decreased from 14 633 to 6741. The proportion of singleton categories also decreased from 82.9% to 56.5%. When looking at the 100 most popular categories, we also found that those categories contained more tags. Figure 8 shows the distribution of the size of those categories.

We also reviewed the 50 largest categories for each variant and confirmed that the categories of Witt_C/Witt_{CA} more systematically reflect general classes of technologies than those of Witt_H. This observation can be confirmed by reviewing the complete lists on our on-line appendix. As an illustration, Table 11 lists the 10 largest categories for each variant, with their membership size. As evidence: the hypernym with the largest membership (*open source*) is incorrect; the shared entries (e.g., *programming language*) have much higher membership for categories, and some of the most common hypernyms do not have a less constrained superset (e.g., *functional programming language* vs. *programming language*). In case the membership numbers seem staggering, we note that many software technologies are described by multiple different tags. For example, in addition to the tag `java`, there are eight other tags related to the Java programming language (e.g., `java-7`).

To evaluate the correctness of our category structure, we extracted from the set of all 3744 super-category–sub-category relations a sample of 348 relations, for a confidence interval of 5% at the 0.95 level. We then asked two external annotators to answer the following question: "Does the concept indicated under 'sub-category' truly represent a sub-category of the concept indicated under 'super-category'?" We used only two external annotators (instead of a large group) to be able to resolve disagreements through consensus, rather than treating each decision as a vote, which intro-

TABLE 11
Ten largest categories of each version

Witt_H		Witt_C/Witt_{CA}	
Hypernym	**Size**	**Category**	**Size**
open source	298	library	1104
company	254	tool	889
process	184	framework	871
software	146	system	715
programming language	145	programming language	642
multi-paradigm programming language	120	company	555
web application framework	117	language	486
functional programming language	113	process	396
integrated development environment	104	program	380
imperative programming language	96	platform	332

duces a risk of shared misunderstanding. Additionally, an increased number of evaluators makes agreement measures much more complex to interpret. The annotators initially answered either by "Yes", "No" or "Don't Know". This initial classification was completed with a Cohen's kappa agreement of 0.355, and the disagreements and "Don't Know" were resolved in a separate meeting with the second author. In total, 251 relations (72.1%) were marked as correct by both annotators after the meeting, 73 relations (21.0%) were marked as incorrect, and for 24 relations (6.9%), the disagreement could not be resolved. All initial "Don't Know" responses were resolved after the meeting. After inspecting the results, most of the incorrect relations occurred when separating multiple terms used as an idiom, such as *third party*. In this case, a *third party framework* is not a sub-category of a *party framework*. However, the fact that the category *party framework* does not contain any tag, except for those in its sub-category *third party framework*, mitigates the negative effects of this incorrect structure.

Finally, we point out that without the type of *automatic* broad categorization supported by Witt, it is simply not possible to produce analyses of the type shown in Figure 1. To illustrate the extent to which Witt is more effective at matching similar software technologies, we ran the five tools described in Section 7.1 on all of the 209 tags Witt categorized as Web Application Frameworks. Table 12 shows the category with most tags returned by each tool. For example, the contender with the best coverage, WiBi, finds "Software Framework" for 36% of the tags while other tools such as Google, THD, and WordNet are unable to identify a single category that contains more than three tags. Clearly, in this case none of the tools would support automatically grouping similar technologies into a shared category.

7.3 Sensitivity Analysis

Because our approach relies on a number of thresholds, we performed a sensitivity analysis on the numeric thresholds used for linking tags to their Wikipedia article (Section 4). In this analysis we recorded the effects of variations on the thresholds on the sample data described in Section 7.1. We were interested in whether the article linked to the tag would be different or not.

TABLE 12
Dispersion of the Web Application Framework tags

Tool	**Most Popular Category**	**#Tags**	**%**
WiBi	Software Frameworks	75	36%
DBpedia	Web Application Framework	47	22%
WordNet	Move	3	1%
Google	*(no category with more than 2 tags)*		
THD	*(no category with more than 2 tags)*		
Witt	Web Application Framework	209	100%

TABLE 13
Minimal Variation of the Parameters to Induce a Change in the Linked Article for at Least 1% of the Tags

Parameter	**Sensitivity**
Discount factor	4%
Minimum score	2%
Leading section bonus	90%
Discounted section penalty	95%
α	12%
β	5%
γ	12%
Partial match (2)	32%
Partial match (3)	2%

The results of our sensitivity analysis can be found in Table 13. Refer to Table 2 for a description of the parameters. For all but four thresholds, a variation of at least 12% would be needed to change the linked article for more than 1% of the input tags. The following four thresholds were more sensitive than the others. For the weight of the title (β) in the similarity score and the discount factor (0.8) used when comparing similarity scores, we found that it required a 5% and 4% change in threshold value to change the linked article for 1% of the input tags. For the score given to a partial match of type 3 and the minimum similarity score needed to accept the Wikipedia article, a 2% change in threshold value was necessary to observe a 1% change in results. The latter observation is not surprising because this threshold is used to discriminate whether to accept an article or not. Overall, the sensitivity analysis shows the algorithm to be stable.

To evaluate the sensitivity of our approach to the words chosen as the *programming keywords*, we recorded the effect of removing each keyword one by one. For each set of tags (popular, common and rare), removing any keyword did not change the outcome for more than three tags. Removing

TABLE 14
Number of Articles Changed by Removing Each of the *Programming Keywords*

Keyword removed	**Popular** (169 tags)	**Common** (370 tags)	**Rare** (379 tags)
comput	3	3	1
program	3	1	2
framework	0	1	2
library	0	1	1
software	1	3	3

library or *framework* had the lowest impact, and removing *comput* had the highest. *Program* typically impacted more the popular tags, and *software* impacted more the rare tags. The complete set of results is shown in Table 14.

7.4 Threats and Limitations

Our use of a representative random sample for three stratas of the entire population of Stack Overflow tags means that our results are expected to generalize within their strata with 95% confidence. However, we cannot make any claim about the performance of the approach for general input queries. In practice, however, Stack Overflow tags already make up a vocabulary of over 45 000 software terms. Furthermore, small variants in spelling or in the use of acronyms are eliminated by our normalization heuristics, which effectively broadens the input space to include a much larger number of supported queries.

The implementation of Witt relies on thresholds that were manually selected during the development of the approach. Choosing different values will naturally impact the performance of the tool. However, the overall combination of heuristics reduces the impact of specific thresholds, and the set of evaluated tags differed from the set of tags used to develop Witt and fix its parameters. Also, none of the thresholds are directly related to the complete text or popularity of an input tag, so it would be impossible to predictably bias the results through threshold selection.

8 Conclusion

Motivated by the intention to better track references to categories of equivalent technologies in informal documentation, we developed a domain-specific technique to automatically produce an attributed category structure describing an input phrase assumed to be a software technology. We implemented our technique into a tool called Witt, which relies on the Stack Overflow and Wikipedia data sources. With Witt, we contribute a solution to three technical problems: *1)* to find the Wikipedia article (or article section) that best describes a Stack Overflow tag, if available; *2)* to extract valid hypernyms for a tag from a Wikipedia article (or section); *3)* to abstract hypernyms into general and uniform categories that group similar technologies.

We evaluated our technique by comparing its performance to that of five existing taxonomy tools applied to a representative sample of Stack Overflow tags. The results show that, at least for popular and common tags, Witt has significantly better coverage than the next best technology, while keeping a low relative false positive rate. For popular tags (the most likely to be queried), Witt shows 54% coverage with an average of 0.82 false positives. The closest matches are DBPedia Spotlight (39% coverage, at the cost of an average of 2.4 false positives) and Google (average 0.79 false positives, at the cost of 12% coverage).

We do not claim that our solution is universally superior to existing taxonomy tools. Indeed, it was developed with the goal of performing well for the software domain, and for this reason it encodes many software-specific rules. Nevertheless, the experiment gives us confidence that to automatically categorize software technologies, Witt is currently the best option available.

References

[1] R. M. Aliguliyev, "A new sentence similarity measure and sentence based extractive technique for automatic text summarization," *Expert Systems with Applications*, vol. 36, no. 4, pp. 7764–7772, May 2009.

[2] A. Barua, S. W. Thomas, and A. E. Hassan, "What are developers talking about? An analysis of topics and trends in Stack Overflow," *Empirical Software Engineering*, vol. 19, no. 3, pp. 619–654, 2014.

[3] C. Bizer, J. Lehmann, G. Kobilarov, S. Auer, C. Becker, R. Cyganiak, and S. Hellmann, "DBpedia - a crystallization point for the web of data," *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 7, no. 3, pp. 154–165, Sep. 2009.

[4] S. A. Caraballo, "Automatic construction of a hypernym-labeled noun hierarchy from text," in *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics on Computational Linguistics*, 1999, pp. 120–126.

[5] N. R. Carvalho, J. J. Almeida, M. J. V. Pereira, and P. R. Henriques, "Probabilistic SynSet Based Concept Location," in *1st Symposium on Languages, Applications and Technologies*, 2012, pp. 239–253.

[6] R. L. Cilibrasi and P. M. B. Vitanyi, "The google similarity distance," *IEEE Transactions on Knowledge and Data Engineering*, vol. 19, no. 3, pp. 370–383, March 2007.

[7] B. B. Dalvi, W. W. Cohen, and J. Callan, "WebSets: Extracting sets of entities from the web using unsupervised information extraction," in *Proceedings of the Fifth ACM International Conference on Web Search and Data Mining*, 2012, pp. 243–252.

[8] *DCMI Metadata Terms*, DCMI Usage Board, June 2012, http://dublincore.org/documents/2012/06/14/dcmi-terms/?v=terms#subject.

[9] M.-C. de Marneffe and C. D. Manning, *Stanford typed dependencies manual*, The Stanford Natural Language Processing Group, Sep. 2008, http://nlp.stanford.edu/software/dependencies_manual.pdf.

[10] M. Dojchinovski and T. Kliegr, "Entityclassifier.eu: Real-time classification of entities in text with wikipedia," in *Machine Learning and Knowledge Discovery in Databases*, ser. Lecture Notes in Computer Science, H. Blockeel, K. Kersting, S. Nijssen, and F. elezn, Eds. Springer, 2013, vol. 8190, pp. 654–658.

[11] J.-R. Falleri, M. Huchard, M. Lafourcade, C. Nebut, V. Prince, and M. Dao, "Automatic extraction of a WordNet-like identifier network from software," in *18th IEEE International Conference on Program Comprehension (ICPC)*, 2010, pp. 4–13.

[12] T. Flati, D. Vannella, T. Pasini, and R. Navigli, "Two is bigger (and better) than one: the Wikipedia Bitaxonomy Project," in *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics*, 2014.

[13] L. Gravano, P. G. Ipeirotis, H. V. Jagadish, N. Koudas, S. Muthukrishnan, L. Pietarinen, and D. Srivastava, "Using q-grams in a DBMS for approximate string processing," *IEEE Data Engineering Bulletin*, vol. 24, no. 4, December 2001.

[14] V. Hatzivassiloglou, J. L. Klavans, M. L. Holcombe, R. Barzilay, M.-Y. Kan, and K. R. McKeown, "Simfinder: A flexible clustering tool for summarization," in *Proceedings of the NAACL workshop on automatic summarization*, vol. 1, 2001.

[15] M. A. Hearst, "Automatic acquisition of hyponyms from large text corpora," in *Proceedings of the 14th Conference on Computational Linguistics*, 1992, pp. 539–545.

[16] T. Kliegr, V. Svatek, K. Chandramouli, J. Nemrava, and E. Izquierdo, "Wikipedia as the premiere source for targeted hypernym discovery," in *Proceedings of the ECML PKDD Workshop Wikis, Blogs, Bookmarking Tools: Mining the Web 2.0*, 2008.

[17] Z. Kozareva and E. Hovy, "A semi-supervised method to learn and construct taxonomies using the web," in *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 2010, pp. 1110–1118.

[18] J. R. Landis and G. G. Koch, "The measurement of observer agreement for categorical data," *Biometrics*, vol. 33, no. 1, pp. 159–174, March 1977.

[19] X. Li, H. Wang, G. Yin, T. Wang, C. Yang, Y. Yu, and D. Tang, "Inducing taxonomy from tags: An agglomerative hierarchical clustering framework," in *Advanced Data Mining and Applications*, ser. Lecture Notes in Computer Science, S. Zhou, S. Zhang, and G. Karypis, Eds. Springer Berlin Heidelberg, 2012, vol. 7713, pp. 64–77.

[20] D. Lo, L. Jiang, and F. Thung, "Detecting similar applications with collaborative tagging," in *Proceedings of the International Conference on Software Maintenance*, 2012, pp. 600–603.

[21] C. D. Manning, M. Surdeanu, J. Bauer, J. Finkel, S. J. Bethard, and D. McClosky, "The stanford CoreNLP natural language processing toolkit," in *Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, 2014, pp. 55–60.

[22] P. N. Mendes, M. Jakob, A. García-Silva, and C. Bizer, "DBpedia Spotlight: Shedding light on the web of documents," in *Proceedings of the 7th International Conference on Semantic Systems*, 2011, pp. 1–8.

[23] R. Mihalcea and A. Csomai, "Wikify!: Linking documents to encyclopedic knowledge," in *Proceedings of the Sixteenth ACM Conference on Conference on Information and Knowledge Management*, 2007, pp. 233–242.

[24] G. A. Miller, R. Beckwith, C. Fellbaum, D. Gross, and K. J. Miller, "Introduction to Wordnet: An on-line lexical database," *International Journal of Lexicography*, vol. 3, no. 4, pp. 235–244, 1990.

[25] D. Milne and I. H. Witten, "Learning to link with Wikipedia," in *Proceedings of the 17th ACM Conference on Information and Knowledge Management*, 2008, pp. 509–518.

[26] K. Nakayama, T. Hara, and S. Nishio, "Wikipedia link structure and text mining for semantic relation extraction." in *Proceedings of the Workshop on Semantic Search, 5th European Semantic Web Conference*, 2008, pp. 59–73.

[27] J. Nonnen, D. Speicher, and P. Imhoff, "Locating the meaning of terms in source code: Research on "term introduction"," in *Proceedings of the 18th Working Conference on Reverse Engineering*, 2011, pp. 99–108.

[28] A. Ritter, S. Soderland, and O. Etzioni, "What is this, anyway: Automatic hypernym discovery," in *Proceedings of the AAAI Spring Symposium: Learning by Reading and Learning to Read*, 2009, pp. 88–93.

[29] A. K. Saha, R. K. Saha, and K. A. Schneider, "A discriminative model approach for suggesting tags automatically for Stack Overflow questions," in *Proceedings of the 10th Working Conference on Mining Software Repositories*, 2013, pp. 73–76.

[30] R. Snow, D. Jurafsky, and A. Y. Ng, "Learning Syntactic Patterns for Automatic Hypernym Discovery," in *Proceedings of the 18th Annual Conference on Neural Information Processing Systems*, 2004.

[31] C. Stanley and M. D. Byrne, "Predicting tags for StackOverflow posts," in *Proceedings of the 12th International Conference on Cognitive Modelling*, 2013, pp. 414–419.

[32] "Most popular web application frameworks," Blog, http://www.hurricanesoftwares.com/most-popular-web-application-frameworks/.

[33] Y. Tian, D. Lo, and J. Lawall, "Automated construction of a software-specific word similarity database," in *Proceedings of the IEEE Conference on Software Maintenance, Reengineering, and Reverse Engineering*, 2014, pp. 44–53.

[34] ——, "SEWordSim: Software-specific word similarity database," in *Companion Proceedings of the 36th International Conference on Software Engineering*, 2014, pp. 568–571.

[35] C. Treude, O. Barzilay, and M.-A. Storey, "How do programmers ask and answer questions on the web? (NIER Track)," in *Proceedings of the 33rd International Conference on Software Engineering*, 2011, pp. 804–807.

[36] C. Treude and M.-A. Storey, "Work item tagging: Communicating concerns in collaborative software development," *IEEE Transactions on Software Engineering*, vol. 38, no. 1, pp. 19–34, 2012.

[37] S. Wang, D. Lo, and L. Jiang, "Inferring semantically related software terms and their taxonomy by leveraging collaborative tagging," in *Proceedings of the 28th International Conference on Software Maintenance*, 2012, pp. 604–607.

[38] T. Wang, H. Wang, G. Yin, C. X. Ling, X. Li, and P. Zou, "Tag recommendation for open source software," *Frontiers of Computer Science*, vol. 8, no. 1, pp. 69–82, 2014.

[39] "Comparison of web application frameworks," Wikipedia page, Verified 6 July 2016., https://en.wikipedia.org/wiki/Comparison_of_web_application_frameworks.

[40] W. Wu, H. Li, H. Wang, and K. Q. Zhu, "Probase: A probabilistic taxonomy for text understanding," in *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 2012, pp. 481–492.

[41] X. Xia, D. Lo, X. Wang, and B. Zhou, "Tag recommendation in software information sites," in *Proceedings of the 10th Working Conference on Mining Software Repositories*, 2013, pp. 287–296.

[42] I. Yamada, K. Torisawa, J. Kazama, K. Kuroda, M. Murata, S. D. Saeger, F. Bond, and A. Sumida, "Hypernym Discovery Based on Distributional Similarity and Hierarchical Structures," in *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 2009, pp. 929–937.

[43] J. Yang and L. Tan, "SWordNet: Inferring semantically related words from software context," *Empirical Software Engineering*, pp. 1–31, 2013.

[44] T. Zesch, C. Mller, and I. Gurevych, "Extracting lexical semantic knowledge from wikipedia and wiktionary," in *Proceedings of the Conference on Language Resources and Evaluation, electronic proceedings*, 2008.

[45] H. Zha, "Generic summarization and keyphrase extraction using mutual reinforcement principle and sentence clustering," in *Proceedings of the 25th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2002, pp. 113–120.