Automatically Categorizing Software Technologies

Mathieu Nassif, Christoph Treude and Martin P. Robillard

Abstract—Informal language and the absence of a standard taxonomy for software technologies 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 technology (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 six 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 increasingly relies 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 that becomes difficult to monitor across discussion channels. The unstructured data, informal nomenclature, and folksonomies used on social media forums make it difficult to reliably determine, for example, the list of all technologies of a certain type, or their popularity relative to this type. Questions such as "what is the most popular web application framework?" are important to many organizations, for example to decide which development tool to adopt at the start of a project, or which technology to develop a driver for. Answers to these questions are routinely proposed without any kind of supporting data (e.g., [35]), but sound empirical surveys are hard to find. To move towards a streamlined, 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), which led to the development of many automated hypernym extraction tools. 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.

To address this issue, we propose an *automated* approach for the categorization of software technologies. Our approach, called Witt, for *What Is This Technology*, takes as input a term 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.). Our approach involves three automated phases that each address a major technical challenge. First, we find the Wikipedia article (or article section) that best describes the input term. Then, we use the article to extract candidate hypernyms for the phrase. Finally, we extract general categories and related attributes for the hypernyms.

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 mentions to any of the technologies within one or more categories. It was intended to provide answers to questions such as "what are all the web application frameworks available?". An important aspect to consider is the ability to express queries with varying levels of precision. Already established development teams may want only web application framework *for a given language*, whereas the managers of a given framework could want to track the popularity of their product relative to all frameworks. Hence, we focused our efforts on generating a normalized and flexible categorization structure. Nevertheless, the success of such a task relies on the ability to accurately answer questions like "what is this new technology X?"

With Witt 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 application, Figure 1 shows the *proportion* of Stack Overflow posts tagged with a specific

- *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*

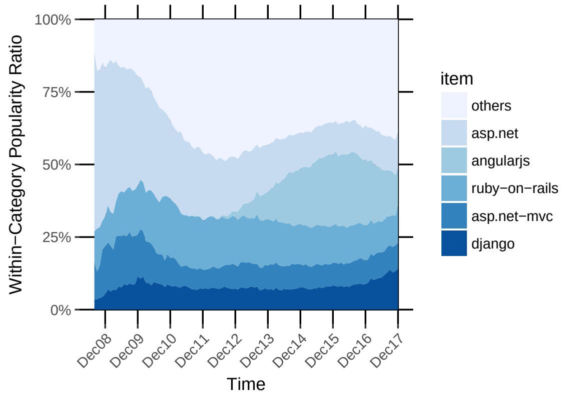

Fig. 1. Ratio of Stack Overflow posts tagged with a specific technology over all posts tagged with a technology in the same *automatically discovered* category (frameworks) and attribute (web application)

web application framework over all posts tagged with a *web application framework* – a pair of category and attribute that Witt automatically constructed. Without Witt, this kind of visualization would require a manually-crafted list of tags or keywords that match all technologies of interest. This visualization, however, is only one of numerous potential applications.

To evaluate the quality of the results and the value of the domain adaptation in general, we compared our approach to six available hypernym discovery tools, ranging from the venerable WordNet [25] to the most recent development, WebIsADb [32], and including the Google search engine. When queried for software terms, Witt was found to be superior to the six 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. We also 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 [25], a manually-constructed database of 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"' [25, 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 (e.g, "such as X") [13], and Caraballo extended the idea by aggregating hypernyms into a hierarchy [4]. These approaches were improved with the use of linguistic dependencies [33] and supervised machine learning algorithms [30]. All these approaches work by mining large text corpora. Among the latest such techniques is the WebIsA Database [32] from the Web Data Commons project, which extracts hypernyms from CommonCrawl,[1] a corpus of over 2.1 billion web pages.

In contrast to these previous works, our method only requires Stack Overflow tag information data and targeted Wikipedia searches. It creates a structure that links a single term to an attributed category that describes the term.

Wikipedia as a Source of Encyclopedic Knowledge

Previous 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 [46].

Several works have focused on automatically discovering and resolving ambiguous links between terms from textual documents and Wikipedia, a process known as *wikification*. Milne and Witten [26] used unambiguous links as contextual information to give as input to a trained classifier. Mihalcea and Csomai [24] 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. We cannot rely on these traditional approaches because the tag information on Stack Overflow is typically concise and often missing, and because we cannot make the assumption that a relevant link exists. 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. [27], 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 [9], [15].

Given the amount of links and meta-data available in Wikipedia, numerous other approaches to information ex-

1. http://commoncrawl.org/

traction are possible, such as leveraging similarities between words [44], keyword popularity [16], or HTML tables [6]. The Wikipedia Bitaxonomy Project (WiBi), extracts information from Wikipedia articles to add to the taxonomy of Wikipedia categories, and vice-versa [11], 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 [23]. Similar efforts are also under way in industry [42].

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 contextual information to find the relevant article. We overcome this lack of context by using domain-specific techniques.

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 [10]. In a similar vein, Nonnen et al. use heuristics to discover where concepts associated with an identifier are introduced or described in source code [28]. 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 [36], [37] and its meta-data [40], or from source code comments and identifiers [45]. 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.

Tagging in Software Engineering

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

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 [39]. 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 [40]. 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*. In addition, previous work has proposed a tag recommendation approach for projects hosted on Ohloh and Freecode [41] 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 [38]. Several approaches have been developed for recommending tags for Stack Overflow posts, including a discriminative model approach [31], a Bayesian probabilistic model [34], and an approach combining several techniques called TagCombine [43].

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 [1]. In contrast to this work, our approach automatically categorizes the software technologies represented by tags.

3 Overview of the Approach

Our approach takes as input a term to categorize. As a vocabulary for software technologies, we use the set of all Stack Overflow tags. With a cardinality of 51 109, these tags form a closed set that is a very good equivalent of the otherwise open set of all possible terms that describe software technologies. In the remainder of this paper, we thus consider a Stack Overflow tag and an input term to our approach to be equivalent concepts.

As a starting point, we use the *excerpt* and *information page* of Stack Overflow tags downloaded from the Stack Exchange API.[2] 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 that describes the tag, if it exists (Section 4). Then, we apply a new hypernym detection algorithm to the article that describes the tag (Section 5). Finally, we transform hypernyms into more abstract descriptors called *categories* and *attributes* (Section 6).

We developed the structure of `Witt` using a set of 6317 tags and their corresponding information downloaded in 2014. This set was used to generate the foundations of our approach and test different heuristics in an iterative, trial-and-error process. After this initial development phase, we downloaded all 51 109 tags, with their corresponding information, on January 10, 2018. We extracted two distinct subsets. We used the first subset, composed of 382 tags chosen uniformly among the entire set, as our *development set*, to refine our approach and select the values of the different parameters of our approach. The second subset, containing 984 tags, was exclusively used to evaluate `Witt`, and is described in greater details in Section 7. Throughout this paper, we report global statistics computed over all 51 109 tags from Stack Overflow.

4 Linking Tags to Articles

Several scenarios can make the linking process far from trivial. Some tags closely match a Wikipedia title, but for a different sense (e.g., the default article for `ant` refers to the insect, not the build tool). Other tags, such as `curl`, could reasonably be linked to more than one computer science

2. http://api.stackexchange.com/

related article. Also, some tags are only described within a section of a related article. For example, the tag `catalina` is described in 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.

We tackle these challenges with original solution elements related to software technology. Overall, our linking process is able to find a corresponding article for 30% 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 they do not represent a performance measure. Many tags, such as those representing a specific class like `uinavigationcontroller`, simply do not have any corresponding Wikipedia article.

Our linking process relies on the computation of a *similarity score* between a tag and a Wikipedia article, and involves some numeric parameters and a fixed list of programming keywords. We detail how we determined these components of the approach at the end of the section.

4.1 Search Process

The search process uses the information contained in the tag excerpt and information page. Missing tag information is common on Stack Overflow, which increases the difficulty of the search task: in the 2018 dataset, only 55% of the tags have a full information page and an excerpt.

We start by parsing the tag information page for hyperlinks to Wikipedia articles. If we do not find any match, we automatically perform a text search using the Wikipedia API.[3] This text search involves three steps. First, we transform the tag into proper query terms. Next, we pass the query terms to the search API, and use a similarity score to select zero or one article among the results. Finally, if we find an article, we parse it to identify which section describes the tag.

Inspecting Links in Tag Information Pages

We identified four situations in which a link to Wikipedia found in the information page is likely to be to the matching article. We proceed through the following cases in order and select the first match, if any:

1) There is a single hyperlink and it points to a Wikipedia article.
2) There are links to Wikipedia articles in the first paragraph. In this case, we select the article with the highest similarity score (see Section 4.2).
3) There is a single list containing a single link to a Wikipedia article. This case usually happens when the information page contains a list of references.
4) There is a block quote, with a reference to a Wikipedia article.

Creating Query Terms for the Wikipedia Search API

If the heuristics in the previous section do not find an article, we create well-formed query terms to perform a text search. This step is necessary because of the constrained format of Stack Overflow tags.[4]

3. http://en.wikipedia.org/w/api.php
4. The only valid characters are `[a-z0-9+-#.]`

First, we consider all hyphens in Stack Overflow tags as spaces, and we look at the tag's synonyms on Stack Overflow. Tag synonyms are secondary tags that redirect to the main tag. We use the synonym if the three following conditions are met: (1) the synonym is longer than the main tag (to avoid further contractions), (2) the main tag is not a substring of the synonym (to avoid more specific concept), and (3) all words of the synonym appear in the tag excerpt (to avoid distinct but related concepts).

Next, we apply a series of transformations specific to Wikipedia. We remove all version numbers, and look for a more descriptive name in the tag excerpt. 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 word of the excerpt: *AJAX (Asynchronous JavaScript and XML)*

Analyzing Search Results

The query terms are now passed to the Wikipedia search API. We select the first five results as candidate articles, and continue to add subsequent results to our list of candidates as long as at least half of the titles contain at least one of the query terms.

If we can find a disambiguation page[5] among the results, and if its title matches the query terms, we extract all articles on this page, except for articles under the `See also` section.[6] If the disambiguation page contains section headings with at least one programming keyword, we only take articles under these sections and the leading section. This new list of candidates will replace the one obtained from the Wikipedia search API.

Finally, we remove all disambiguation pages from the results, and apply a second category-based filter described in Section 4.2. We select the candidate article with the highest similarity to the tag among the remaining articles, unless the similarity score is below a minimum threshold min_{sim}, in which case we determine that there is no matching Wikipedia article.

Identifying Specific Sections

To identify whether the tag is described only in a specific section of the article, we start with the assumption that the leading section of the article describes it, and iterate through the sections of the article. We select a different section only if both components of the similarity score are higher than those computed with the leading section and article title.

4.2 Computing the Similarity Score

We define a similarity score measure *simscore* between a tag and a pair (article, section). The score estimates the

5. A page containing a list of articles sharing a similar title.
6. A section that typically appears at the end of Wikipedia pages, linking to related articles.

likelihood that a tag and a section of an article refer to the same concept. *Simscore* is composed of two components, *textsim* and *titlesim*. To compute the score, we add the first component (*textsim*) to the product of a small factor w_{title} and the second component (*titlesim*). Because both components take values in the unit interval, *simscore* values range from 0 to $1 + w_{title}$.

The following sections describe each component in details, followed by a justification of the constants employed.

Category-Based Filter

Each Wikipedia article belongs to a number of categories. We analyze the categories to estimate the relevance of the article to software technologies.

We defined the function $\mathrm{prog}(s)$ that takes as input any string s and returns 1 if the s is in camel case or contains at least one of our predefined programming keywords, and 0 otherwise. For each category, we compute both the result of prog on the category's name and the average of prog over all of its members. We average the two to create an individual score for the category. Finally, we determine that an article should be filtered out if the average score of its categories is less than a fixed threshold, min_{cat}.

Textual Similarity (textsim)

We calculate the textual similarity based on the *Q grams* similarity [12] between the first paragraph of the selected article section and the tag excerpt. This metric calculates the proportion of common sequences of q tokens between two strings. The outcome is a score between 0 (completely different) and 1 (exactly copies). We chose the commonly used value $q = 2$ for this metric, with words as tokens. We stemmed each word using the Porter Stemmer [29] and did not consider the letter casing.

We chose this metric because it is order-sensitive (as opposed to, e.g., the Jaccard index [14]) and do not greatly penalize insertions of long sequences such as propositions (as opposed to, e.g., the normalized Levenshtein similarity [18]).

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 runs out of sentences. 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 minimize variations between small values.

If either the tag excerpt or the first paragraph of the article is missing or empty, it is impossible to compute the textual similarity. In this case, we give a default textual score of $min_{sim} - w_{title}$ for this component. This default score is such that the *titlesim* must take the maximum value of 1 for *simscore* to pass the min_{sim} threshold.

Title Similarity (titlesim)

The title score component is also computed using a string similarity metric that returns a value between 0 (different) and 1 (identical). We compare the query terms and the article title. If there are other Wikipedia titles that redirect to the same article, we take the maximum similarity over all titles. However, if only a specific section of the article

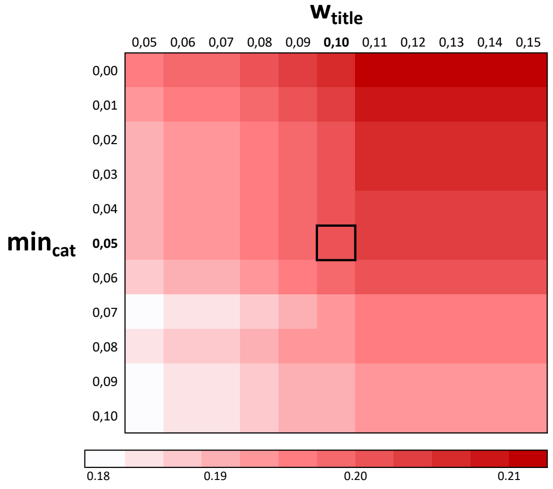

Fig. 2. Sensitivity of the w_{title} and min_{cat} parameters to small variation. Each cell represents the number of true positive rate obtained when using the corresponding pair of values, according to the legend below, for an optimal value of min_{sim} (possibly different for each cell).

describes the tag, we take the maximum value of only two alternatives: the section title and the concatenation of the article title and the section title.

For this component, we use the normalized Levenshtein similarity, which penalizes every insertion, deletion, displacement and substitution, because the small length of the titles would make the *Q grams* similarity very sensitive to small variations.

Here again, we use stemmed words as tokens for both input strings, and do not consider letter casing. Additionally, we remove any disambiguation part in parentheses before the comparison.

Selecting Values for the Parameters

Three numeric parameters are involved in the linking process: w_{title}, min_{cat} and min_{sim}, respectively the weight of the title similarity relative to the text similarity, the cut-off threshold of our category-based filter and the minimum score threshold to accept the final candidate article.

To choose the parameters' value, we manually created a benchmark of the Wikipedia articles for each tag in our development set. We then simulated the search with different values of each parameter and computed the true positive rate and accuracy of our approach. Because the balance between these metrics is arbitrary, we had to make an arbitrary final decision. However, we found out that the values of these parameters are not very sensitive to small variations. Figures 2 and 3 show the combined effect of varying w_{title} and min_{cat} on the true positive rate and accuracy. Combined variations of up to 50% of the value of w_{title} and 100% of the value of min_{cat} induced a maximal variation of 3% on the true positive rate and accuracy. Figure 4 shows the sensitivity of parameter min_{sim}, with the fixed values $w_{title} = 0.10$ and $min_{cat} = 0.05$. Variations of up to 20% of its value only modifies the true positive rate and accuracy by 2%.

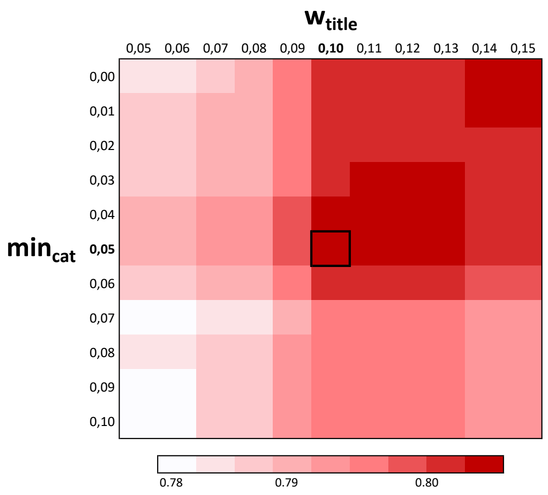

Fig. 3. Sensitivity of the w_{title} and min_{cat} parameters to small variation. Each cell represents the accuracy obtained when using the corresponding pair of values, according to the legend below, for an optimal value of min_{sim} (possibly different for each cell).

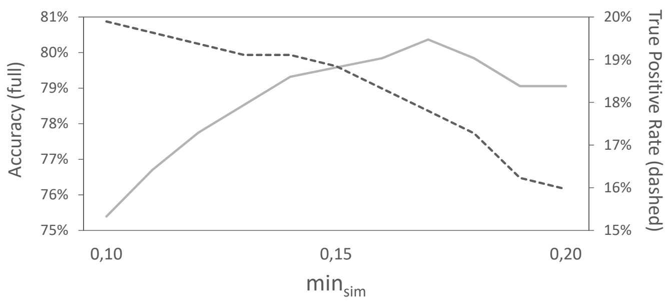

Fig. 4. Sensitivity of (min_{sim}) to small variations. For these simulations, the values of w_{title} and min_{cat} were fixed at 0.10 and 0.05.

Creating a List of Programming Stems

To generate the set of programming keywords used in the linking process, we extracted all words from all tag excerpts, stemmed them using the Porter Stemmer, and sorted them by decreasing number of occurrences. This produced a list of 28 013 stems. We automatically filtered out stems appearing less than 100 times (the most popular ones had over 10 000 occurrences). Next, we manually analyzed the remaining 950 stems. We removed all stems that either did not have a specific meaning in the programming domain, or that corresponded to a specific software technology, such as *java*, as opposed to a general programming concept. The final result is a list of 75 stems.

5 Hypernym Discovery

The next step of our approach consists of extracting hypernyms from the gathered resources (Wikipedia article and Stack Overflow tag information). We use four techniques that complement each other, and can each return zero, one, or multiple hypernyms. Two of those techniques are specific to Wikipedia articles, and the other two are used both on the Wikipedia articles and tag excerpt. The output of this process is the union of all six sets of discovered hypernyms.

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]...*[7], the technique returns nothing.

We only use the first uninterrupted sequence of wikilinks, separated only by spaces, commas, conjunctions and the articles *a*, *an* and *the*. Wikilinks that appear later in the sentence are rejected. We return the complete sequence as a hypernym. We also return each wikilink individually, as additional hypernyms.

When wikilinks are piped,[8] i.e., with a different text than the title of the linked article, we use the displayed text for the hypernym extracted from the whole sequence, but the exact titles for the hypernyms extracted from each individual wikilink.

Finally, we remove all disambiguation terms in parentheses, and reject all hypernyms consisting of only one word, as we found these to generally represent specific technologies instead of broader concepts.

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 *java program* is returned, as well as *Computer program*.

5.2 Infobox Values

This technique involves parsing the first infobox of the article, if there is one.

For this technique, we manually determined a small set of attributes that usually contain hypernyms, and are present in many templates. We created the list by looking at all programming-related infobox templates. This list consists exclusively of the four keys `genre`, `type`, `family` and `paradigm`.

For each key found in the infobox template, we return a normalized version of the associated value. The normalization consists of straightforward transformations that split all items in an enumeration and remove markup and elements such as hyperlinks and footnotes. For one-word values, we append to them the title of the infobox to form a complete hypernym. Finally, we add the infobox title alone as an additional hypernym. This title is usually general, but accurate.

5.3 Wikipedia Categories

We considered using the Wikipedia categories as hypernyms, but eventually rejected the idea because the concept of a category, as used in Wikipedia, is more general than the hypernymy relation. For example, the categories of the article `Chrome OS` include `Google` and `Google Chrome`.

7. As required by the Wikipedia Manual of Style.
8. https://en.wikipedia.org/wiki/Wikipedia:Piped_link

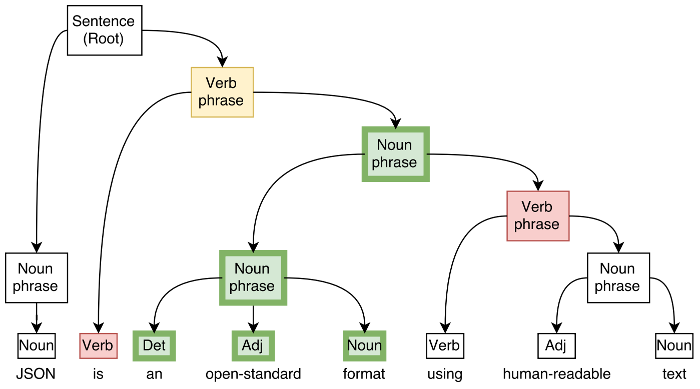

Fig. 5. An example of the Phrasal Group approach. The input sentence was *"JSON is an open-standard format using humain-readable text."* The structure of the phrasal groups is shown as returned by the parser. The analysis starts at the first verb phrase (in yellow), and parses the tree, keeping only the relevant phrasal groups (in green with thick border, the groups kept, in red with slim border, the groups rejected).

While those categories represent meaningful associations of articles, they do not represent a hypernym of their members.

As evidence, the WiBiTaxonomy Project attempts to extract hypernyms based on both the articles and categories linked to a Wikipedia article, and this resulted in a higher false positive rate (see Section 7).

5.4 Natural Language Processing

The other two techniques use NLP to select only the most pertinent words of the first sentence of an article (or section) to form 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. The two different techniques are based on the different outputs of the *Stanford Core NLP* library for Java [22].

Phrasal Groups 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 longest 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. We recursively parse all remaining phrasal groups, removing extraneous groups at each step. Figure 5 shows an example of this approach.

Grammatical Relations Approach

The second heuristic considers 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 pre-selected grammatical relations. We then take all new words, and look for more words related to them with the same relations, until there are no more words to add. The hypernym is composed of the selected words in the same order as they appear in the sentence. Some of

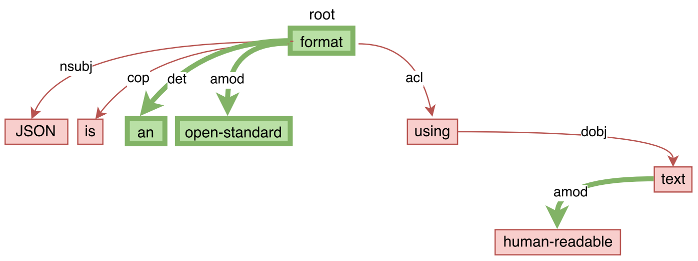

Fig. 6. An example of the Grammatical Relation approach, using the same sentence as in Figure 5. The approach starts with the root word and includes all words related by one of the chosen relations (in green with thick border, the words kept, in red with slim border, the words rejected).

TABLE 1
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

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 both groups. The accepted grammatical relations [8] are `poss`, `possessive`, `amod`, `mod`, `nn`, `det`, `predet`, `pobj`, `advmod` and `number`. The splitting relations are `conj`, `apos` and `dep`. Figure 6 shows an example of this approach.

6 Extracting Categories

The last step of our approach is to transform the list of hypernyms extracted as described in the previous section into a unified category structure. The final output is a set of category–tag pairs, each associated with a set of attributes.

6.1 Importance of Grouping Hypernyms

Table 1 shows some examples of hypernyms discovered, with their corresponding input tags. Even this small sample illustrates the limitations of using hypernyms to categorize similar technologies.

First, **attributes introduce variants of a concept** represented by a hypernym. For example, 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, because *and* connects two attributes of the concept *programming language*. Third, hypernyms sometimes **use an acronym** instead of the expanded form. This situation introduces some additional variations preventing a consistent grouping of the software technologies. 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 transformations such as putting all words in lower case, normalizing punctuation, removing disambiguation terms in parentheses, and transforming nouns into a common base form, i.e., removing grammatical variations like the plural forms and cultural variations like the American ending *or* instead of the British *our*. The last step is performed using WordNet's catalog as our reference of the base form of a word.

6.3 Detecting Acronyms and Compound Terms

The previous tokenization may split *compound terms* (e.g., *open source*). Compound terms should act as a unit, so we regroup them and treat them as one token before pursuing the transformation.

We start by aggregating all originally hyphenated word sequences back into a compound term. This has the added benefit of normalizing hyphenation. Hence, *open-source* and *open source* will both be considered as the same, single term.

We also replace expanded acronyms by their abbreviated form. This step both regroups compound terms and solves the problem of the having two expressions of the same concept (the acronym and its expansion).

Detecting acronyms is a hard research problem in itself, and most state-of-the-art methods were not usable with our set of hypernyms because they rely on a large corpus of text to detect acronyms and their expansion. We used a heuristic to generate a list of acronyms and their expansions: for each set of hypernyms associated with a given tag, and for each word w of those hypernyms (except stop-words), we look for a sequence of words in the other hypernyms such that by taking the first letter of each word in the sequence, and at most one additional letter by word, in order, we could reconstruct w. Stop words can be ignored or used to complete a match.

This simple technique produced surprisingly good results. We manually validated the list of all acronym–expansion pairs. Of the 174 candidate pairs automatically generated, 102 (59%) were retained. Most of the false positives were incorrect expansions of real acronyms for which the correct expansion was also found. We used the manually validated list in our approach.

Finally, we used a statistical approach to detect the remaining compound terms. As for the acronyms, current state-of-the-art approaches [21, Chapter 5] to detect compound terms cannot be used reliably on our corpus composed of many short, nominal sentences, with high redundancy due to popular categories of tags. Therefore, we used the following model to define and detect compound terms in our particular context. A set of terms forms a compound term if, whenever the terms in the set appear together in the same hypernym, they are placed one after the other, always in the same order.

Based on this model, if we could find a sequence of terms that, whenever they appear in the same hypernym, are placed one after the other in the same order with probability $1 - \epsilon$, we concluded that they formed a compound term. We chose the value of ϵ as the ratio of hypernyms with a grammatically incorrect structure, which we estimated using our development set. We used the statistical binomial test, with a confidence of 95%, to filter out coincidental events.

6.4 Extracting Categories and Attributes

The final step of the approach consists of transforming the hypernyms into a set of categories, possibly with some attributes.

We designed *categories* to represent *general hypernyms*, with a focus on coverage: *commercial ide for php* is a better (more precise) hypernym than *ide*, but the latter is a better category (higher coverage). The *attributes* are meant to provide a flexible way to express the information lost when transforming a hypernym into a category. They represent typical variants of the category, but would not constitute valid hypernyms on their own.

To transform a hypernym into a category with attributes, we start by removing all non-informative phrases like *name of* and *type of*. We also transform phrases indicating a collection, e.g., *set of*, into the attribute *collection of*, and remove it from the hypernym. We constructed a small list of such phrases based on our development set. If two or more occurrences of the word *of* or of the word *for* remain in the hypernym, we do not parse the hypernym, as its structure is possibly too complex for our simple heuristics.

If the remaining hypernym contains both the words *of* and *for*, we remove everything after and including *for*. This procedure tends to produce reliable hypernyms without removing too much information.

At this point, the hypernym may contain at most one occurrence of *of* or *for*. If this is the case, we consider the sequence that comes after *of* or *for* to be a series of attributes. We split this sequence on the conjunctions and commas, and remove trailing stop words. Each remaining group will form a new attribute.

We are left with a sequence of terms without *of* or *for*. To parse this sequence, we need three metrics to distinguish categories from attributes. We define the position of a word in a hypernym by counting from right to left, from 0. Hence, in the hypernym *markup language*, *language* has the position 0, and *markup* has the position 1. Then, for each word, we compute the mean of its position and its variance over all hypernyms. Finally, the support represents the number of times the word appears in any hypernym.

To parse the remaining sequence, we employ the following algorithm. The last term becomes a category, unless it has a support of at least 10 and a position variance of at least 2.5. However, if there was a group starting with *of* or *for* that created some attributes, this rule is not applied, and the last term always become a category. Then, for each word from right to left, if the word is a stop word, we ignore it, otherwise the word becomes an attribute, unless it has a support of at least 10, a mean position of at most 0.5, and the term to its right is a conjunction, in which case it becomes a category. Attributes are always assigned to the last discovered category. If no category is found in the hypernym, the hypernym is rejected.

We used three parameters in the conditions: the minimum variance of an attribute (2.5), the maximum average of a category (0.5), and the minimum number of occurrences to make a decision (10). These thresholds were determined by studying the hypernyms of the development set. For the stop words, we used the list from the NLTK project. [2]

Finally, for each tag that belongs to two categories, where one is a prefix or suffix of the other, only the longest category is kept, but all attributes are kept.

7 Evaluation

The complete output of Witt is a very large set of tag–category–attributes relations, and the categories discovered by the approach have technically open-ended extensions. For these reasons, it is not possible to compare the output with any specific oracle. Instead, we evaluated 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?*

7.1 Comparative Evaluation

We compared Witt with six taxonomy tools to determine how well they could extract valid hypernyms for software technologies. We found five active projects offering comparable functionality: *WebIsADb* [32], *WordNet* [25], *DBpedia Spotlight* [23], *WiBiTaxonomy* [11] and *THD* [9] (see Section 2). We found these tools by using the Google Scholar search engine, with queries such as *targeted hypernym OR hyponym extraction* and *automatic taxonomy OR ontology creation OR discovery*. We also looked at the tools described in the research articles and those used in the evaluation sections of these articles. We also compared Witt with *Google*'s definitions to make sure a simple automated Google search would not outperform our approach.

Evaluation Set

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. This resulted in 563, 15 673 and 34 873 popular, common and rare tags.

We 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 229 popular tags, 375 common tags, and 380 rare tags, for a total of 984.

Obtaining Hypernyms

We provided all the tags in our sample as input to the seven tools under evaluation. Because the comparison tools are not domain-specific, we injected additional information to contextualize the query to the programming domain. We manually verified that 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 the same list of programming word stems we created and used in the development of Witt, which is described in Section 4.2. Additionally, because the format of Stack Overflow tags may affect the efficiency of the tools, if a tool did not return any hypernym for a tag, we tried again with the normalized version of the tag we created in Section 4.1.

The following sections briefly describe each tool and detail the exact procedure employed to obtain hypernyms for sample tags. All procedures described were fully automated.

WiBiTaxonomy: The WiBiTaxonomy project (WiBi) [11] created a directed hypernymy graph from all of Wikipedia articles and categories, using NLP techniques and the existing links between articles and categories. As in our case, WiBi relies on the 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 of the online demo for the two parameters of the approach: a maximum page height of 2 and a category height of 3.

THD: Targeted Hypernym Discovery (THD) [9], [15] 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 the Linked Hypernyms Dataset 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 [25] 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 [7].

WebIsADb: The WebIsA Database was created by applying a refined and extended set of grammatical patterns, similar to the Hearst patterns, to the very large corpus of web documents, Common Crawl. In addition to finding hypernyms, WebIsADb uses *pre-modifiers* and *post-modifiers*, a concept similar to our attributes.

We gave as input the tag to WebIsADb, and collected the three hypernyms with the highest support. For each of these hypernyms, we added the three pre-modifiers or post-modifiers with the highest support. We prepended all pre-modifiers and appended all post-modifiers to their corresponding hypernym, thus creating only one larger hypernym. We restricted the number of results to three, because WebIsADb often returned a very large number of results for a tag, sometimes up to a few thousand results for a single tag. Not only would using all hypernyms have a significant cost in the evaluation, but the precision of lower results greatly reduces after the top ones. The same rationale stands for the modifiers. Without restricting the size of the output, we would end up with hypernyms composed of hundreds of words.

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 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 12596 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 source-blind 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 randomly ordered. This way, it was impossible for the evaluators to know which tool had produced which pair. Furthermore, 200 unidentified pairs were given to both evaluators, to support an assessment of the evaluator's agreement. The evaluators marked each pair as either *correct* (and in a sense related to software), or *incorrect*.

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 the *correct* 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). Evaluating hypernyms is a relatively low-subjectivity task, and the evaluators completed the task with a Cohen's kappa agreement of 0.719. According to a common scale this agreement can be considered substantial [17]. All disagreements were resolved by attributing the value *incorrect* to the pair.

Results

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 motivations for 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). Assume that the list returned for Java comprises 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%. Precision is thus 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: 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 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 2 provides the complete results, organized by subsample and by tool. The best performance for each subsample is in bold.

We find that Witt_H and Witt_{CA} offer better coverage than all other tools, for all three sets of tags. Given the 5% confidence interval, the superiority of Witt is statistically significant. Google is the only tool that consistently returned a lower number of false positives than Witt_{CA}, but at the cost of very low coverage. Otherwise, Witt_{CA} returned fewer false positives than all other tools (except for WordNet over the set of rare tags, but again, at the cost of very low coverage).

Among the variants of Witt, Witt_C generally produces worse results than Witt_H and Witt_{CA}. We explain this difference by the fact that category names are usually too general to represent a useful hypernym on its own. However, adding the attributes in Witt_{CA} solves this issue and provides major coverage improvements. Also, there is a slight loss of coverage from Witt_H to Witt_{CA}, but this also results in a large improvement of the false positive rate.

To get a better idea of the relative performance of the tools that takes 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.

TABLE 2
Evaluation Results

	Popular tags	Common tags	Rare tags
Proportion of tags with at least one correct hypernym			
Witt_H	**0.686**	**0.408**	**0.247**
Witt_{CA}	0.590	0.368	0.224
Witt_C	0.376	0.195	0.089
DBpedia	0.454	0.197	0.082
WiBi	0.293	0.160	0.089
THD	0.074	0.035	0.013
Google	0.118	0.040	0.003
WordNet	0.024	0.022	0.005
WebIsADb	0.328	0.187	0.084
Average number of wrong hypernyms per tag			
Witt_H	1.528	0.955	0.466
Witt_{CA}	0.913	0.584	0.295
Witt_C	1.262	0.827	0.458
DBpedia	3.135	2.683	2.600
WiBi	2.314	2.749	2.092
THD	1.419	0.979	0.708
Google	**0.751**	**0.379**	**0.095**
WordNet	2.288	1.080	0.126
WebIsADb	2.031	1.589	0.971

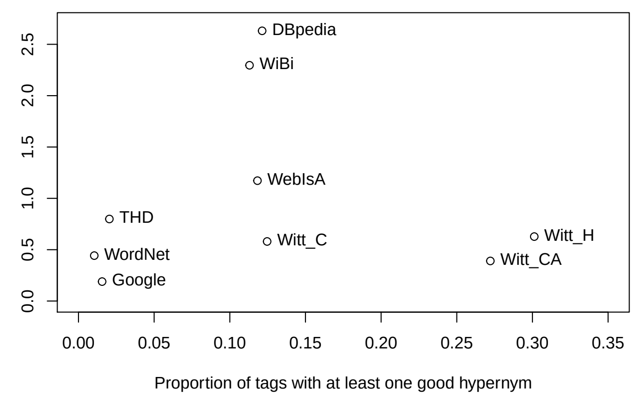

Fig. 7. Extrapolated aggregated performance. The bottom right corner indicates the overall best performance.

7.2 Categories

A major contribution of this work is the algorithm we use to abstract hypernyms into general categories (Section 6). 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 are much more useful than raw hypernyms, because they support automatically categorizing 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.

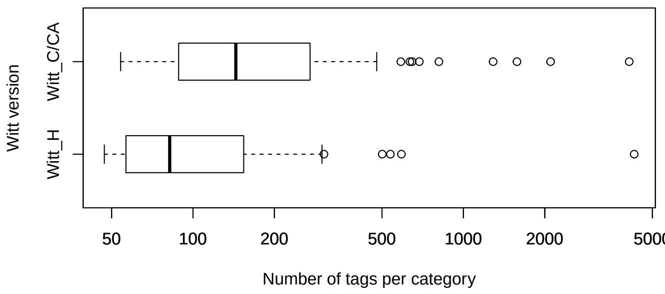

Fig. 8. Size of the 100 largest categories (log scale).

First, we found that after transforming hypernyms into more abstract categories, the total number of categories decreased from 31 760 to 2707. The proportion of singleton categories also decreased from 80.6% to 49.4%. 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 $\text{Witt}_C/\text{Witt}_{CA}$ 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 3 lists the 20 largest categories for each variant, with their membership size. As evidence: the second largest hypernym (*open source*) is incorrect; the shared entries (e.g., *programming language*) have much higher membership for categories, and the hypernyms show much more redundancy (five variations of *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 at least twelve other tags related to the Java programming language (e.g., `java-9`).

TABLE 3
Ten largest categories of each version

Witt_H		$\text{Witt}_C/\text{Witt}_{CA}$	
Hypernym	**Size**	**Category**	**Size**
software	4282	software	4104
open source	590	library	2101
programming language	537	framework	1578
company	501	tool	1288
file format	305	systems	812
process	300	programming-language	687
integrated development environment	256	platform	646
tool	255	company	634
library	250	language	587
web application framework	224	class	478
os	218	service	449
free software	209	plugin	430
functional programming language	200	component	420
imperative programming language	199	application	395
multi-paradigm programming language	199	api	395
software framework	185	program	374
open-source software	182	extension	349
website	179	functions	336
framework	178	file-format	318
object-oriented programming language	173	os	314

7.3 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 50 000 software terms. Furthermore, small variants in spelling or in the use of acronyms are eliminated by our normalization procedures, 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. Moreover, the sensitivity of the our parameters shown in Section 4.2 demonstrates that these thresholds are robust to small variations. Finally, 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 six existing taxonomy tools applied to a representative sample of Stack Overflow tags. The results show that 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 59% coverage with an average of 0.91 false positives. The closest matches are DBPedia Spotlight (45% coverage, at the cost of an average of 3.1 false positives) and Google (average 0.75 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] 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.

[2] S. Bird, E. Klein, and E. Loper, *Natural language processing with Python: analyzing text with the natural language toolkit*. O'Reilly Media, Inc., 2009.

[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] 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.

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

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

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

[14] P. Jaccard, "Étude comparative de la distribution florale dans une portion des alpes et des jura," *Bulletin de la Societe Vaudoise des Sciences Naturelles*, vol. 37, no. 142, pp. 547–579, 1901.

[15] 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.

[16] 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.

[17] 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.

[18] V. I. Levenshtein, "Binary codes capable of correcting deletions, insertions, and reversals," in *Soviet physics doklady*, vol. 10, no. 8, 1966, pp. 707–710.

[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 and H. Schütze, *Foundations of statistical natural language processing*. MIT press, 1999.

[22] 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.

[23] 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.

[24] 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.

[25] 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.

[26] 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.

[27] 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.

[28] 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.

[29] M. F. Porter, "An algorithm for suffix stripping," *Program*, vol. 14, no. 3, pp. 130–137, 1980.

[30] 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.

[31] 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.

[32] J. Seitner, C. Bizer, K. Eckert, S. Faralli, R. Meusel, H. Paulheim, and S. P. Ponzetto, "A large database of hypernymy relations extracted from the web," in *Proceedings of the 10th edition of the Language Resources and Evaluation Conference*, 2016, pp. 360–367.

[33] 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.

[34] 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.

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

[36] 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.

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

[38] 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.

[39] 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.

[40] 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.

[41] 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.

[42] 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.

[43] 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.

[44] 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.

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

[46] 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.

Mathieu Nassif I am a Master student at McGill University, Canada. My research interests include the evolution of software systems, information extraction techniques, and software documentation. I completed my B.Sc. in pure and applied mathematics at the Université de Montréal in June 2016.

Christoph Treude I am a faculty member and an ARC DECRA Fellow in the School of Computer Science at the University of Adelaide, Australia. I received my Diplom degree in Computer Science/Management Information Systems from the University of Siegen, Germany, and my PhD degree in Computer Science from the University of Victoria, Canada. Before joining the University of Adelaide, I worked as a postdoctoral researcher at McGill University in Montréal, Canada, and I conducted research in Brazil at DIMAp/UFRN in Natal as well as at IME/USP in São Paulo.

Martin P. Robillard Martin Robillard is a Professor of Computer Science at McGill University. His current research focuses on problems related to software evolution, architecture and design, and software reuse. He served as the Program Co-Chair for the 20th ACM SIGSOFT International Symposium on the Foundations of Software Engineering (FSE 2012) and the 39th ACM/IEEE International Conference on Software Engineering (ICSE 2017). He received his Ph.D. and M.Sc. in Computer Science from the University of British Columbia and a B.Eng. from École Polytechnique de Montréal.