TechSpaces: Identifying and Clustering Popular Programming Technologies

ABSTRACT

Background: Software ecosystems are becoming increasingly complex and large. Therefore, discovering and selecting the right libraries and frameworks for use in a project is becoming a challenging task. Existing commercial services that support this task rely on annual surveys with developers to provide a landscape of the most popular technologies in a given ecosystem. **Aims:** In this paper, we outline a semi-automated technique for this purpose, which we call TechSpaces. **Method**: Our proposal relies on community detection and well-known NLP algorithms to automatically extract groups of related technologies, using as primary data source tags associated with Stack Overflow questions. **Results:** We describe the first results of using our technique to identify popular and inter-related technologies in five programming language ecosystems. **Evaluation:** We compare our technique against two other tools in the literature. **Conclusions:** The proposed technique shows potential to assist IT professionals in taking technical decisions supported by crowd knowledge. However, further improvements are needed to make it a viable choice. For instance, we envision the usage of other data sources (e.g., GitHub and Wikipedia) can contribute to improve the accuracy and expressiveness of our graph representations.

KEYWORDS

Programming technologies, APIs, Community detection algorithms, Mining software repositories.

ACM Reference Format:

. 2022. TechSpaces: Identifying and Clustering Popular Programming Technologies. In *Proceedings of 16th Brazilian Symposium on Software Components Architectures and Reuse (SBCARS '22).* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Modern software development heavily depends on third-party APIs [5, 6, 13]. Nowadays, even simple apps use a variety of libraries and frameworks for front-end interaction, communication, security, persistence, basic data structures manipulation, among others [8]. For this reason, we claim that—without exaggeration—finding and selecting the right libraries and frameworks is a key decision in the path for delivering high-quality software with the right time to marketing.

SBCARS '22, October 03-07, 2022, Uberlândia, Brazil

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

However, it is not trivial to discover and select the tools and libraries for using in a project. For example, npm—the widely popular package repository for JavaScript—hosts more than 1.3 million packages in April 2020.

In fact, in recent years commercial services appeared to provide curated data about the most popular APIs in a given ecosystem. One example is the "State of JS" site whose goal is to "identify current and upcoming trends in the *javascript* ecosystem".[1] Particularly, this site provides information on the most common *javascript* libraries for front-end and back-end development, communication, testing, building, and mobile development. However, this data is obtained by runnnig surveys with the *javascript* community. Therefore, each year, a new survey is conducted to update the data and collect new trends. This strategy requires a significant effort and consequently it is not a surprise that in 2021 the service maintainers were not able to send the survey, which is happening only in 2022.[2]

In this paper, we outline the first implementation of a semi-automated technique to identify and cluster popular APIs in a given software ecosystem. The proposed technique has two key characteristics. First, it relies on the co-occurrence between Stack Overflow's tags to group the closely related ones into technology spaces. This abstraction aims to provide a comprehensive overview of the technologies adopted in a given ecosystem and to allow the users to choose the best ones according to their needs. Second, by using well-known NLP techniques, our technique annotates the selected technologies with higher-level alternative categories, so users can better understand the main purpose when adopting each technology. Finally, we use our technique to identify popular technologies in five popular programming languages ecosystems: *java*, *python*, *javascript*, *c#* and *php*.

The remainder of this paper is structured as follows. In Section 2, we outline the proposed technique. In Section 3, we report the first results of using the proposed technique in five popular ecosystems. In Section 4 we list usage scenarios and in Section 5 we discuss the evaluation. Section 6, discusses related work and Section 7 concludes the paper.

2 PROPOSED SOLUTION

The approach proposed in this paper can be used with any technology or programming language that has tags in Stack Overflow. However, to make our presentation more concrete and easy to follow, in this section we rely on examples from five major programming languages, as listed in the Introduction. A more detailed view of our results is presented in Section 3.

Figure 1 depicts the steps used to leverage our proposed solution. As we can observe, our approach is divided into three key steps. We start by collecting the co-occurrence information about Stack Overflow's tags. In parallel, we pre-process the textual description of each tag (known as excerpt) in order to extract their respective

[1] https://2020.stateofjs.com/en-US/technologies/
[2] https://2021.stateofjs.com/en-US/

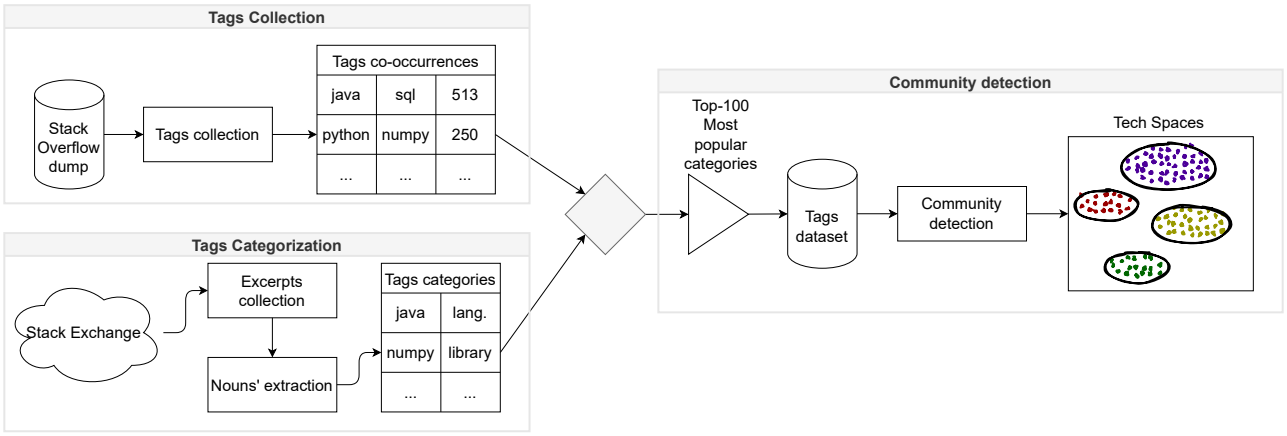

Figure 1: Proposed Solution.

technology category. The resulting data of the aforementioned processes are then merged and forwarded to the last step, where we rely on a community detection algorithm to cluster semantically related tags based on their co-occurrences to, at last, generate the techspaces. These steps are described in detail in the remaining of this section.

2.1 Tags Collection

Our approach depends on the co-occurrence among Stack Overflow tags to build up the techspaces. Therefore, we first gathered information about the co-occurrence relationship of all existing tags on Stack Overflow. For this, we downloaded the Stack Overflow dump provided by Stack Exchange on January, 2022[3] and generated a three-columns table as follows. The first two columns refer to tags that co-occur in Stack Overflow; this co-occurrence is considered whenever the two tags are used in the same question. Finally, the third column stores the number of co-occurrences of each tuple, which is used later to map the techspaces. In total, we identified 196,290,608 co-occurrences among 53,949,887 questions.

2.2 Tags Categorization

Along with the tags collection method described earlier, we implemented a procedure to automatically classify the Stack Overflow tags into higher level categories, such as *languages*, *libraries*, *frameworks*, etc. Such categorization might enrich our techspaces as it provides additional information about which kind of technology is frequently used in the same context with respect to software development. For instance, *javascript* and *python* have many libraries and frameworks on their techspaces, while *java* has more *tools* and *toolkits*.

To provide this classification, we first need a data source from where we can mine these high-level categories for each tag. In this work, we rely on the excerpts provided by the Stack Overflow for their tags. Basically, an excerpt is a short textual description maintained by the Stack Overflow community that explains what an specific tag is. Figure 2 presents an excerpt example, for the *spring* tag, which was then categorized as a *framework* by our approach.

[3]https://archive.org/details/stackexchange

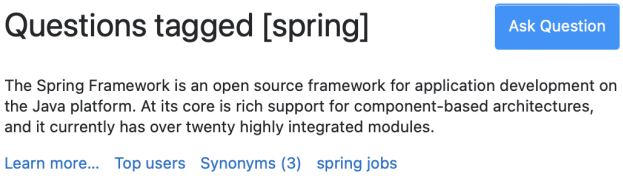

Figure 2: Excerpt for the *spring* tag.

Hence, we started by collecting all excerpts available in the Stack Overflow platform. This time, we gathered the data from the Stack Exchange Data Explorer[4] on Jan 10th, 2022. In total, we leveraged excerpts for 41,589 tags.

Next, we applied the *DepparseProcessor* from StanfordNLP to extract the category for each tag [11]. This algorithm works by determining the syntactic head of each word in a sentence and the dependency relation between the words.

Similar to other works [3, 9], we extract the first sentence from the excerpt as it generally contains the definition of the tag, and then apply the aforementioned NLP algorithm. Figure 3 illustrates this procedure for the excerpt associated to the *spring* tag. Each node represents one word of the sentence and the arrows establish a dependency relation, which type is shown on the side.

Despite having valid excerpts, our approach was not able to find categories for some tags, such as *angular*, *shell*, and *android-studio*. After investigated further, we noted that the excerpt of these tags does not comply with the structure expected by our NLP toolkit. For instance, *Angular* contains the following excerpt: "Questions about Angular (not to be confused with AngularJS), the web framework from Google". In this case, we failed to identify a noun representing a category for this tag (*framework*). Therefore, we opted for removing from our list all tags without any category identified by StanfordNLP. As a result, we classified 22,172 tags distributed in 2,109 high-level categories.

[4]https://data.stackexchange.com/

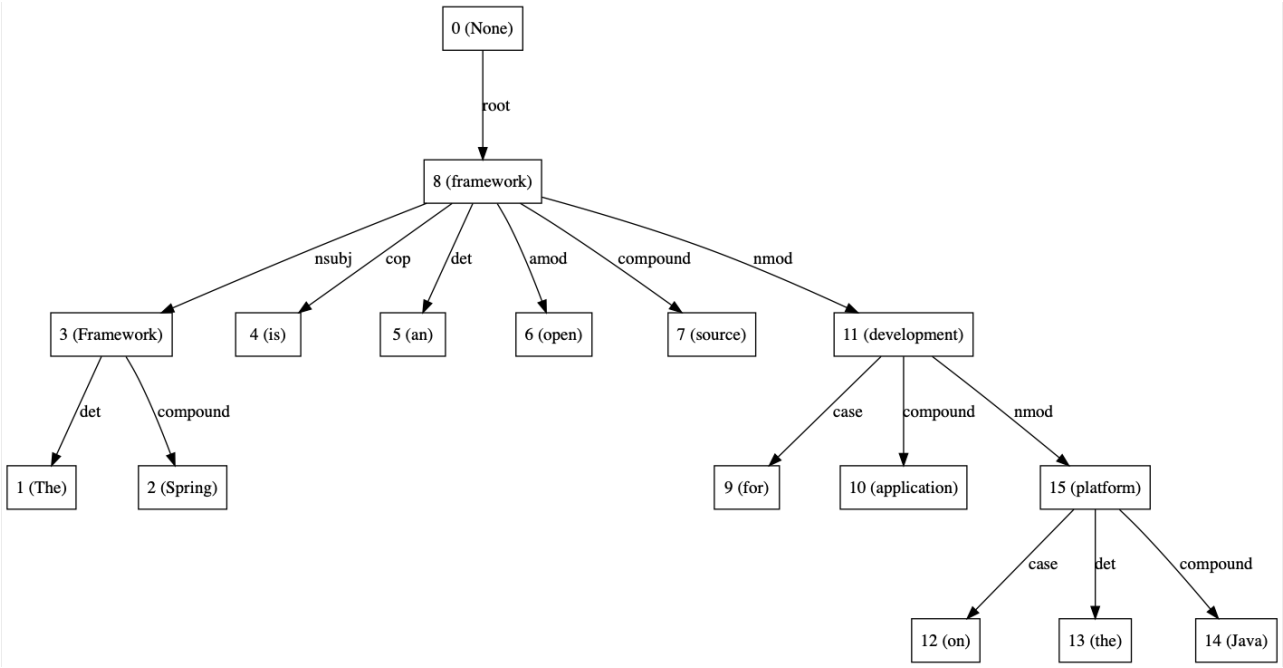

Figure 3: Extracting the category from *spring* excerpt using *DepparseProcessor* algorithm.

2.3 Community Detection

Ultimately, the techspaces proposed in this work should provide a roadmap with the major technologies adopted in each space, so developers can be aware and chose among them to implement the intended software project accordingly. In other words, these groups should be compact and must be formed mostly by essential technologies. In light of this, we performed another three-step filtering process. First, we ranked the categories based on their frequency, and then selected the top-100 most referenced ones. Next, the first author manually analyzed each of them, discarding categories not related to software technologies, such as *name*, *reference*, *sequence*, etc. Finally, we maintained the tags that belongs to one of the remaining categories, removing the other ones from the dataset. Our dataset ended up with 6,076 tags, associated with 14 categories.

Afterwards, we submitted this dataset to the Louvain method for community detection, which is an algorithm that identifies community structures from large networks [1]. Essentially, this algorithm uses a heuristic that computes the partition of the graph nodes which maximizes modularity, using a scale value between −0.5 (non-modular clustering) and 1.0 (fully modular clustering). The higher the value, more dense are the edges inside communities in comparison to edges outside their own communities.

For example, if *java* has a large number of co-occurrences with *spring* and *swing* tags, they will be placed in the same community, even if *spring* and *swing* do not have many co-occurrences themselves. Furthermore, although *java* has a large number of co-occurrences with *python*, they will stay in separate communities, since the other tags in *python*'s community do not have a strong connection with *java*.

The resulting communities represent groups of technologies that are frequently mentioned together in Stack Overflow's questions, which we named as **techspaces**. Therefore, they also tend to be used together during software development tasks. At the end of this procedure, our approach identified five techspaces.

3 RESULTS

We organized the technology spaces as a tree layout, where each node represent one of the top-15 relevant technologies and their respective categories. We also colored each node accordingly to their category, keeping closely related categories with similar colors, e.g., *framework* and *library*; these technologies were also gathered side-by-side in order to improve the techspace visualization. In the following paragraphs we briefly describe the results for five languages: *java*, *python*, *javascript*, *c#* and *php*.

3.1 Java TechSpace

Figure 4 shows *java*'s techspace, with their related frameworks, languages, tools and so on.

As we can see, the categories are well balanced in this ecosystem. *Tool* is the most popular one (three technologies), followed by *framework*, *language*, *system*, and *toolkit* (two, each). Lastly, three categories appear with one occurrence each: *ide*, *platform*, and *component*.

With respect to the technologies itself, we observe that most of them are important for different situations when implementing enterprise applications. For instance, *ant* and *maven* are widely adopted for dependency management in large scale projects. *spring* and *jsf* feature among the most popular web frameworks for Java; the same apply to *hibernate* for database ORM and *eclipse* for IDEs.

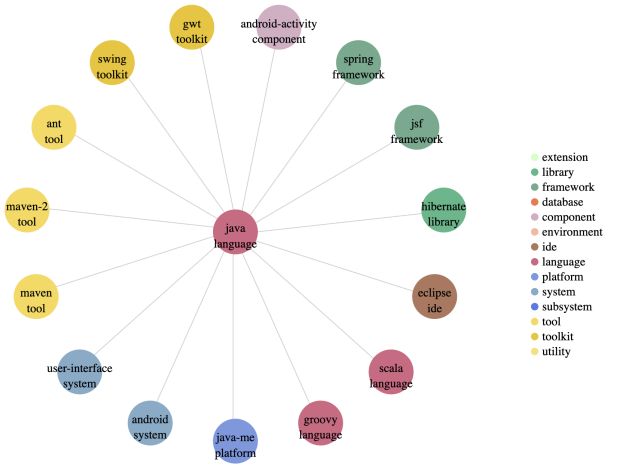

Figure 4: Java's techspace

Finally, *scala* and *groovy* are two programming languages created to address some limitations of the Java language, such as advanced type systems, elements of functional programming, closures, etc. Although both are distinct languages, with possibly their own and smaller ecosystems, we claim it is important for Java developers to known that they can easily integrate their programs with software written in these languages.

3.2 Python TechSpace

Figure 5 shows *python* techspace. Differently from *java*, the *python* has more technologies used as third-party components. For instance, *library*, *framework*, and *extension* contain eight out of the top-15 most relevant ones.

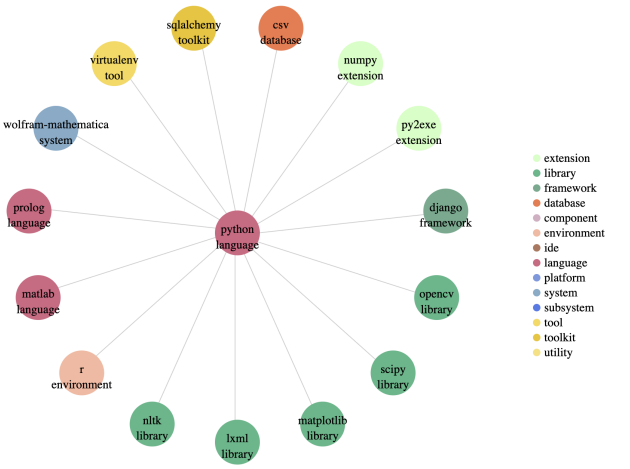

Figure 5: Python's techspace

Furthermore, these technologies include frameworks and libraries for web-development (*django* and *lxml*), computer vision (*opencv*) and science and statistics tasks (*numpy*, *scipy*, *matplotlib*, and *nltk*); it also includes two *languages* (*matlab* and *prolog*), and one technology for *system* (*wolfram-mathematica*), *tool* (*virtualenv*), *toolkit* (*sqlalchemy*), and *database* (*csv*). Interestingly, the majority of such technologies are largely used in scientific computing, which reinforce the role python is playing recently in software development [8, 10, 12].

3.3 JavaScript TechSpace

Similar to Python's, *JavaScript* techspace (Figure 6) has several technologies used as third-party components, with *library*, *framework*, and *extension* containing ten out of the top-15 most relevant ones.

It is also interesting to observe that these technologies are mostly for web-development, so as *css*, *html* and *xul*, categorized as languages. Thereby, the generated techspace confirms the most common use case of the language.

In comparison with "State of JS" website, even analyzing only the technologies usage data, the proposed techspace reflects a scenario of more well-established and longer-term technologies, which filters the enthusiasm of new trends from survey participants.

Even so, the obtained techspace still lacks the presence of one of the most used libraries in the JavaScript ecosystem: *react*. This is explained by the library name itself. Since "react" is an English verb, the proposed NLP solution fails to correctly categorize it. In future versions of our work, we plan to handle this problem, for example, by considered semantically enriched data sources, such as the Wikipedia.

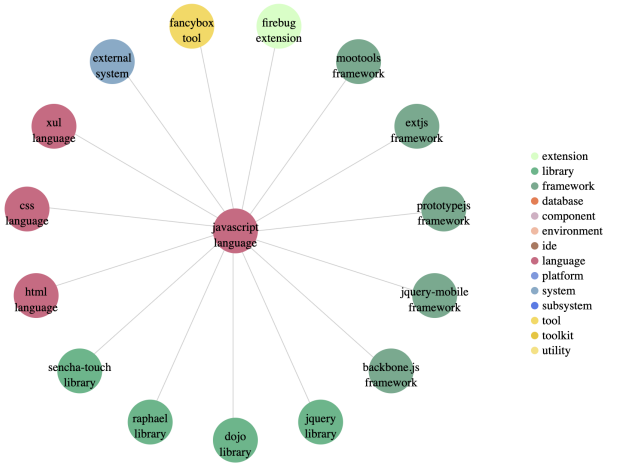

Figure 6: JavaScript's techspace

3.4 C# TechSpace

Figure 7 shows *C#*'s techspace, with its main frameworks, languages, platforms and so on. This space has a balanced number of different categories, with the most predominant being *languages*, with four members (*razor*, *F#*, *xaml* and *vb.net*). Subsequently, we have *frameworks*, *platforms*, *components* and *system* with three, two, two and two members, respectively. Lastly, the *subsystem* and *environment* appears with one member, each.

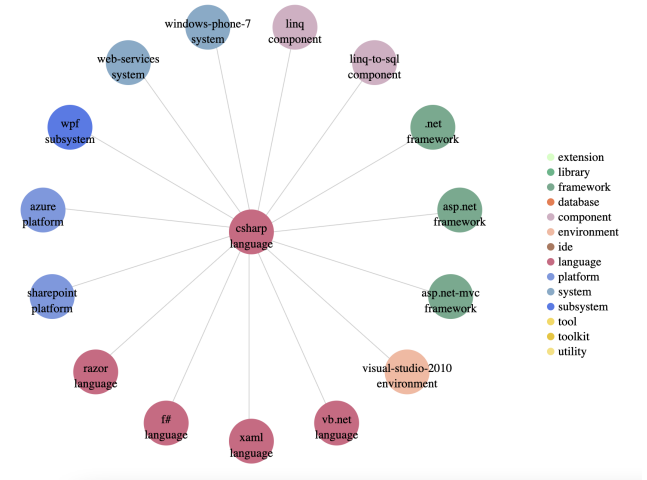

Figure 7: C#'s techspace

Regarding the technologies themselves, the presence of Microsoft technologies is notable, which is expected given the history of the language. For instance, we identified several technologies for different purposes used in *C#*'s applications context, such as *.net*'s family frameworks, *visual-studio-2010* development environment, *azure* cloud platforms and *f#* and *vb.net* languages.

3.5 PHP TechSpace

Lastly, Figure 8 shows *php*'s techspace. It majorly has *frameworks* (four terms), *systems* (four terms) and *platforms* (three terms), although it also contains a *tool*, a *language* and two *extensions*.

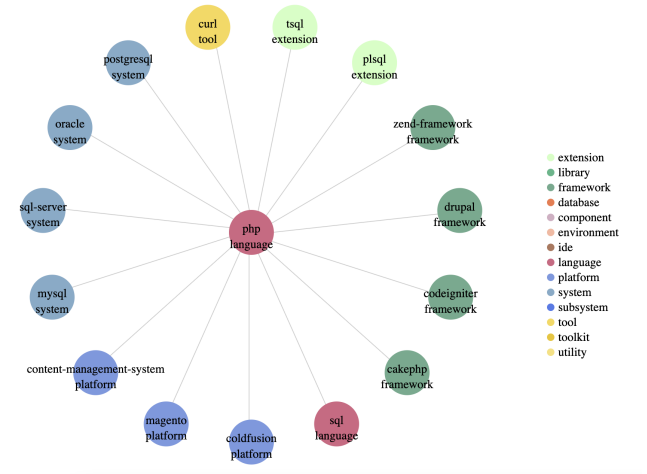

Figure 8: PHP's techspace

The main difference between *php*'s TechSpace in comparison to the other ones is the presence of a series of SQL-related and web-based terms, such as *mysql*, *sql-server*, *postgresql* and *content-management-system*. The reason is probably due to historical aspects of the evolution of the language and the web, since *php* was the most adopted language for the web for a long time.

4 USAGE SCENARIOS

TechSpaces can help IT professionals when taking technical decisions as it shows the most relevant technologies for each ecosystem, based on information provided by the software development community (currently, Stack Overflow users). Therefore, our proposed technique might play an important role as the technological landscape is dynamic and frequently changes overtime [7]. We illustrate some of these typical usage scenarios below:

Tech Stack Adoption: Our technique relies on the number of Stack Overflow's posts to leverage each technology space. In other words, the graph generated by our technique emphasizes popular technologies among the software development ecosystem. In this scenario, TechSpaces can assist developers by helping them to choose a technology stack for a new software project, or even by providing alternative or more modern software components for a predefined stack.

Technology Roadmaps: Currently, our technique's structure does not consider the technologies usage timeline. But once implemented, we might be able to visualize not only popular technologies, but also trending and outdated ones. In this context, TechSpaces can complement roadmap tools—e.g., Thought Works Technology Radar[5]—by relying on crowd knowledge to assist IT professionals in prospecting trending technologies, and in discarding outdated ones.

Career Transition: Finally, TechSpaces can also help IT professionals who are interested in changing the focus of their careers. For instance, a backend developer who is interested in becoming fullstack can use the JavaScript techspace as a reference for the most important technologies she must learn to become proficient in frontend, such as *html*, *css*, *jquery*, etc. Likewise, a professional who is interested in working as a data scientist can study the technologies present in Python techspace (e.g., *numpy*, *scipy*, *matplotlib*, and *nltk*, etc).

5 EVALUATION

We evaluate the TechSpace approach considering two major aspects: the presence of important technologies in the ecosystem and the categorization of these technologies. For this, we compare our technique against two other tools in the literature: TechGraph [2], which is a graphical representation created with association rules of a technology landscape from Stack Overflow question tags; and Witt [9], an automated approach for the categorization of software technologies. Their differences are reported in the remaining of this section.

5.1 Comparison with TechGraph

5.1.1 *TechGraph Overview.*

TechGraph relies on association rule mining and community detection to mine the technology landscape from Stack Overflow [2].

[5]https://www.thoughtworks.com/radar

It is noteworthy that, in our visualization, the category of each technology is immediately presented to the user. Additionally, TechGraph categorizes technologies into just three types: software library, programming language or general concept, while our solution has no restrictions on high-level categories. TechGraph visualizations are also grouped and colored by the association rule, whereas our tool does this through the categories of each technology.

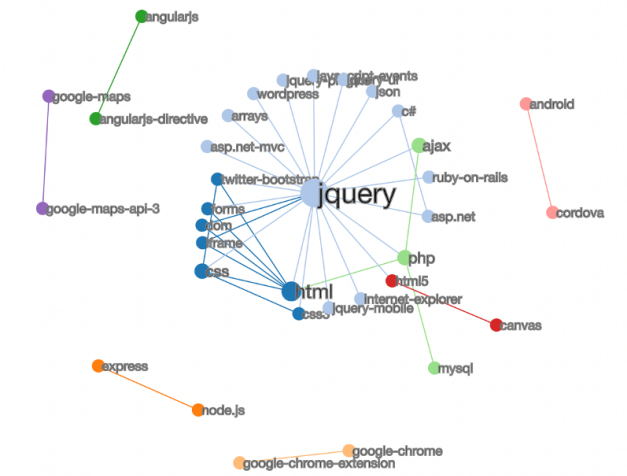

Figure 9: JavaScript's TechGraph

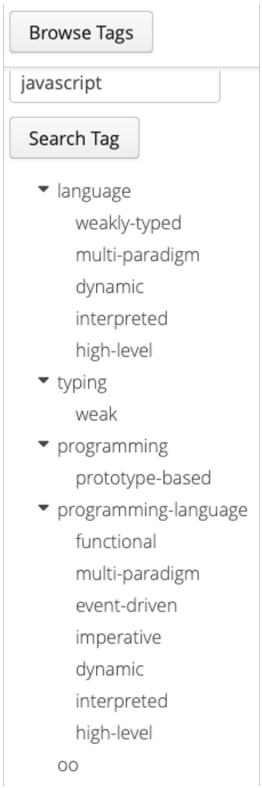

Figure 10: JavaScript's Witt result

Furthermore, TechGraph allows estimating the importance of a technology to an ecosystem by the size of the node in the graph, while our solution only shows the 15 most important technologies, without a defined hierarchy. Although the first approach brings a more detailed level of information, sometimes the generated visualization can be excessively low-level, making it confusing or even difficult to read. For instance, Figure 9 depicts *javascript*'s TechGraph. The subgraph in blue presents html related tags. As we can observe, some of them are excessively low-level, such as *dom*, *forms* and *frame*.

5.1.2 ***Results***.

We compare our approach to theirs by analyzing the graph obtained for the same technologies. Moreover, from the obtained TechGraphs, we select the most important nodes and compare their categorization with ours.

Java: Considering *java*'s TechGraph, we see four main technologies and their subsequent links: *android*, *spring*, *eclipse* and *swing*. With the exception of the last one, which is categorized as a software library, all other technologies are defined as general concept. On the other hand, our solution addresses these four technologies more specifically: *android* is a *system*, *spring* is a *framework*, *eclipse* is an *ide* and *swing* is a *toolkit*.

Python: In TechGraph's *python* ecosystem, the most prominent technologies are *django*, *numpy*, *pandas*, and *matplotlib*. The first two are categorized as general concepts, while the last two are seen as software libraries. In our solution, *django* appears as a *framework*, *numpy* as an *extension*, *matplotlib* as a *library* and *pandas* does not appear as one of the top 15 technologies (although it is categorized as a *library* by the NLP algorithm).
JavaScript: In the *javascript* ecosystem, TechGraph's three main technologies are *jquery*, *html*, and *css*. The first is categorized as a software library, the others two are programming languages. Likewise, our solution includes the three technologies among the top 15 and also presents them as *library* and *language*.
C#: In Techgraph's *C#* ecosystem, we see four main technologies: *asp.net*, *wpf*, *asp.net-mvc* and *.net*. While *wpf* is categorized as a software library, all other technologies are identified as general concepts. In our solution, *wpf* is categorized as a *subsystem* and the other three are classified as *frameworks*.
PHP: Finally, in the *php* TechGraph's ecosystem, the main categories are *mysql*, *javascript*, *html* and *jquery*. The first one is categorized as a general concept, *javascript* and *html* are programming languages and *jquery* is a software library.

In our approach, *mysql* is categorized as a *system*. More important, *javascript*, *html* and *jquery* appears in *javascript*'s TechSpace itself; instead of *php*'s ones.

Figure 11: Java's ecosystem Witt result.

5.2 Witt

5.2.1 *Witt Overview.*

Witt uses Stack Overflow and Wikipedia data to categorize software technologies using data mining and NLP [9]. The Witt approach to grouping technologies is very different from ours, since it is based on the categories of the presented term itself. For instance, Figure 10 presents categories generated by Witt for the *javascript* term. As we can see, it is categorized into main categories and most of these have subcategories. In general, the tags categorizes aspects of the language itself, such as *weakly-typed*, *interpreted* and *high-level*. Similarly, *java*, *python*, *C#*, and *php* present equivalent results, e.g., they are all considered *programming-languages*. While Witt's approach can be quite interesting to show to the user technologies with a similar purpose, it is less friendly when it comes to leveraging the ecosystem of a specific technology.

As categorization, Witt works with an expanded version of the hypernym discovery problem. This means that each technology can be categorized in many different ways, and these categories may contain even more specific subgroups. So, as Figure 10 depicts, *javascript* is a *programming-language*. More specifically, *javascript* is an *interpreted programming-language*. Although the technologies have more specific and hierarchical categorizations, some results do not match what was expected. For example, *swing* is not a programming language, despites being categorized that way.

Although Witt's categorization approach is very interesting to answer statements such as "show me all the interpreted programming languages", this structure does not fit our general goal with TechSpaces.

5.2.2 *Results.*

We compare our approach to Witt considering the categories presented for the same technologies analyzed in the previous subsection, i.e., the most important nodes of TechGraph.

Figure 11 shows Witt's results for *java*. As we can observe, Witt leverages several categories, some of them may be considered controversial, though. For instance, *swing* is categorized as a *toolkit* (like ours), but also as *programming-language* and *language*. The same happens with *spring*, which is classified as a *framework* (like ours), but also as *inversion*, *container*, and *software*.

In other words, Witt leverages many categories representing marginal characteristics of the term, such as oo for java, open-source for eclipse, servlet for spring, etc. This aspect limits its application in our context, since the proposed approach depends on categories capable of defining the type of the technology. As the results for the other languages have a very similar structure, we will only show the results for *java*.

6 RELATED WORK

In this work, we proposed a method to classify Stack Overflow tags into high-level technologies and to group them into technology spaces. In this section, we describe similar studies that contribute either by extending current tags vocabulary or by proposing techniques to group semantically related tags.

Zhu et. al. propose a machine learning method to create an hierarchical taxonomy for existing Stack Overflow tags [14]. Similarly to our approach, they use information on tags co-occurrence to

train semi-supervised models, but focused at identifying hypernym-hypernym relations. Chen et. al. propose another strategy to structure Stack Overflow tags [4]. The authors extract topics from the text of questions and answers and associate them with tags declared in each one. Once this map is completed, they use a topic modeling approach to generate a hierarchy of concepts. The taxonomy proposed by both works assume a topological hierarchy, i.e., a concept encompasses or is encompassed by another. Differently, we propose to associate concepts based on their co-occurrence without any hierarchical assumption. We claim this is a simple and easy to understand model. In fact, existing sites, such as The State of JS, also provide a flat organization of programming technologies.

Nassif et. al. focus their efforts towards automatically categorizing software technologies terms into high-level categories [9]. For this, they implemented a tool that, based on the content provided by Wikipedia articles and well-known NLP techniques, extract and link such categories with software technologies. Similarly, we also rely on NLP techniques to generate high-level categories, using the excerpts provided by Stack Overflow. However, the key advantage of our solution is that we leverage the tags available on Stack Overflow, which are carefully curated by the forum users and moderators.

TechGraph is an approach and tool for inferring technology clusters using data extracted from Stack Overflow question tags [2]. The tool relies on association rule mining and community detection algorithms. As a result, a so-called Technology Associative Network (TAN) is generated, which is essentially a graph whose nodes are technologies and the edges denote associations between technologies (i.e., they frequently appear together as tags in Stack Overflow questions). In addition to the community detection technique, we extended our methodology by adding supervised steps where we instrumented the technologies extracted from Stack Overflow in order to keep up with the most relevant ones in each ecosystem.

A closely related tool was proposed by the same authors to recommend similar libraries, called SimilarTech [3]. To make this possible, the authors merge tags in the same question into one sentence, and then apply well-known NLP and clustering algorithms to match equivalent libraries. Instead, our technique allows us to identify similar technologies in terms of their purpose in the software development process.

7 CONCLUSIONS

In this paper, we proposed a semi-automated approach that relies on NLP and on community detection algorithms to derive clusters of popular and related technologies in programming languages ecosystems, which we called techspaces. We used this approach to retrieve the techspaces of five major programming languages.

We also evaluated our approach against two other techniques by comparing the presence of important technologies in the TechSpace and the categorization of these technologies. The results show that our approach is capable of covering most of the important technologies, as well as providing a concise categorization suited to our context.

Our future work agenda includes:

- We plan to consider other data sources for tag's definitions such as Wikipedia, since we detected that not all excerpts provided by Stack Overflow follow the initially expected structure. For this reason, some important ecosystem tags were not successfully categorized by the NLP algorithm. Since Wikpedia's excerpts are better structured to what we expect, the categorization results may be more accurate.
- We plan to extend the current tags collection dataset to include data on GitHub usage or the clustering algorithm may also lead to interesting results.
- We also intend to evolve the techspace visualization, by adding hierarchies between tags and providing more context to the generated graph.
- We also plan to validate our results with practitioners through surveys and interviews.

Our data and code, including detailed installation instructions, are publicly available at: omitted due to TBR reviewing.

REFERENCES

[1] Vincent D. Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. 2008. Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment* 2008 (2008), P10008.

[2] Chunyang Chen and Zhenchang Xing. 2016. Mining Technology Landscape from Stack Overflow. In *International Symposium on Empirical Software Engineering and Measurement (ESEM)*. 1–10.

[3] Chunyang Chen and Zhenchang Xing. 2016. SimilarTech: automatically recommend analogical libraries across different programming languages. In *International Conference on Automated Software Engineering (ASE)*. 834–839.

[4] Hui Chen, John Coogle, and Kostadin Damevski. 2019. Modeling stack overflow tags and topics as a hierarchy of concepts. *Journal of Systems and Software* (2019), 283–299.

[5] Israel J. Mojica, Bram Adams, Meiyappan Nagappan, Steffen Dienst, Thorsten Berger, and Ahmed E. Hassan. 2014. A Large-Scale Empirical Study on Software Reuse in Mobile Apps. *IEEE Software* 31, 2 (2014), 78–86.

[6] Joao Eduardo Montandon, Luciana Lourdes Silva, and Marco Tulio Valente. 2019. Identifying Experts in Software Libraries and Frameworks Among GitHub Users. In *16th International Conference on Mining Software Repositories (MSR)*. 276–287.

[7] João Eduardo Montandon, Cristiano Politowski, Luciana Lourdes Silva, Marco Tulio Valente, Fabio Petrillo, and Yann Gaël Guéhéneuc. 2021. What skills do IT companies look for in new developers? A study with Stack Overflow jobs. *Information and Software Technology* 129 (2021).

[8] João Eduardo Montandon, Marco Tulio Valente, and Luciana L. Silva. 2021. Mining the Technical Roles of GitHub Users. *Information and Software Technology* 131 (2021).

[9] Mathieu Nassif, Christoph Treude, and Martin Robillard. 2018. Automatically Categorizing Software Technologies. *IEEE Transactions on Software Engineering* 5589 (2018), 1–14.

[10] João Felipe Pimentel, Leonardo Murta, Vanessa Braganholo, and Juliana Freire. 2019. A Large-Scale Study About Quality and Reproducibility of Jupyter Notebooks. In *16th International Conference on Mining Software Repositories (MSR)*. 507–517.

[11] Peng Qi, Timothy Dozat, Yuhao Zhang, and Christopher D. Manning. 2018. Universal Dependency Parsing from Scratch. In *Proceedings of the Association for Computational Linguistics*. 160–170.

[12] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. 2016. "Why Should I Trust You?": Explaining the Predictions of Any Classifier. *arXiv:1602.04938 [cs, stat]* (2016).

[13] Anand Ashok Sawant and Alberto Bacchelli. 2017. fine-GRAPE: fine-grained APi usage extractor – an approach and dataset to investigate API usage. *Empirical Software Engineering* 22, 3 (2017), 1348–1371.

[14] Jiangang Zhu, Beijun Shen, Xuyang Cai, and Haofen Wang. 2015. Building a Large-scale Software Programming Taxonomy from Stackoverflow. In *International Conference on Software Engineering and Knowledge Engineering (SEKE)*. 391–396.