An Efficient ROS Package Searching Approach Powered By Knowledge Graph

Long Chen[1, 2], Xinjun Mao[1, 2*], Yinyuan Zhang[1, 2], Shuo Yang[1, 2], Shuo Wang[1, 2]
[1]College of Computer,National University of Defense Technology, Changsha, China
[2]Key Laboratory of Software Engineering for Complex Systems, National University of Defense Technology, Changsha, China
{chen_long, xjmao, yinyuanzhang, yangshuo11, wangshuo15} @nudt.edu.cn

***Abstract*—Over the past several years, the Robot Operating System (ROS), has grown from a small research project into the most popular framework for robotics development. It offers a core set of software for operating robots that can be extended by creating or using existing packages, making it possible to program robotic software that can be reused on different hardware platforms. With thousands of packages available per stable distribution, encapsulating algorithms, sensor drivers, etc., it is the *de facto* middleware for robotics. However, finding the proper ROS package is a nontrivial task because ROS packages involve different functions and even with the same function, there are different ROS packages for different tasks. So it is time-consuming for developers to find suitable ROS packages for given task, especially for newcomers. To tackle this challenge, we build a ROS package knowledge graph, ROSKG, including the basic information of ROS packages and ROS package characteristics extracted from text descriptions, to comprehensively and precisely characterize ROS packages. Based on ROSKG, we support ROS packages search with specific task description or attributes as input. A comprehensive evaluation of ROSKG shows the high accuracy of our knowledge construction approach. A user study shows that ROSKG is promising in helping developers find suitable ROS packages for robotics software development tasks.**

***Index Terms*—Knowledge Graph, ROS package searching, NLP**

I. Introduction

Writing software for robots is difficult, particularly as the scale and scope of robotics continue to grow. As a framework for building robotics software, ROS is designed with the promise of making development easier through modular design and code reuse[1]. It offers an abstraction layer between the hardware and application layers, providing hardware manipulation primitives that hide the heterogeneity of the underlying hardware, as well as helping manage the communication between robots.

ROS also provides a package management system to simplify code reuse, so developers can contribute their own applications back to ROS in the form of packages. It is widely used by robotics developers and contains 6,191 packages across its 13 distributions[2]. The ROS architecture and package system have led to the success of ROS: ROS is considered as the *de facto* standard for robot programming [4].

When developers need a new feature, developers use a search engine to look for an existing package that implements the feature. But for beginners, since they do not know the ecosystem well, they may choose the inappropriate ROS package during the reuse process, causing the task to fail [4]. For ROS-based robotics software development, ROS Wiki[3] is the most commonly used knowledge search community, especially for newcomers. It provides the most basic keyword search function to meet the most basic requirements for developers to find knowledge to solve tasks. Like many other search engines, what it returns are related web links, not direct answers to the task, so developers need to find relevant software packages from the detail page of these web links. Moreover, it can return a limited number of links (Related ROS packages, Q&A posts, tutorials), and requires the developer to click on the link and read the details page to obtain relevant information. Once the searched keywords are not included in the relevant ROS package, it becomes difficult to retrieve and developers need to constantly adjust according to the existing retrieval information, which is time-consuming.. So the current search for ROS package knowledge does not meet the needs of developers well.

In this paper, we focus on extracting the rich semantic expression of ROS package and its related information(e.g., related dependencies and messages, etc.). Based on this, we can more effectively characterize the ROS package to better support the recommendation of the related ROS package. Specifically, we firstly design a web crawler framework to obtain ROS package descriptions and some structured information (e.g., sensor, motor and robot which ROS package belongs to). Then we use natural language processing methods to parse the description text to obtain more fine-grained features. In order to make up for the lack of description information, we extract and analyze features from the ROS package names. Finally, we apply our approach to the Kinetic distribution of the ROS packages, and obtain 25,484 entities and 62,854 relationships. All the above information serves as the foundation for constructing a comprehensive Knowledge Graph of ROS package (ROSKG) to enable efficient ROS

DOI:10.18293/SEKE2021-063
* Corresponding author.
[1]http://wiki.ros.org/ROS/Introduction
[2]https://index.ros.org/stats/
[3]http://wiki.ros.org/

package search. We conduct two experiments to evaluate our approach. A comprehensive evaluation of ROSKG shows the high accuracy of our knowledge construction approach. A user study shows that ROSKG is promising in helping developers find suitable ROS package for robotics software development tasks.

Our main contributions are summarized as follows:

- We introduce the idea of using knowledge graph based on semantics representation of package information for the task of ROS package searching. To the best of our knowledge, this is the first study to build a knowledge graph of ROS package and address the problem of ROS package selection.
- We leverage techniques of relation linking and text processing to convert semi-structured and unstructured ROS-related knowledge into a knowledge graph, and develop a search engine, which uses natural language as query input, to solve ROS package searching problems.
- We evaluate the quality of the key steps for ROS package knowledge graph construction and the usefulness of the knowledge graph on ROS package searching.

The rest of the paper is organized as follows. In the next section, we review some related works. The details of our approach are presented in Section III. We provide the evaluation of our work in Section IV. We discuss the threats of validity in Section V. Finally, we conclude our work in Section VI.

II. Related Work

A. ROS

In recent studies, researchers pay more attention to the ecology of ROS and the dependencies between ROS packages, etc. Pichler et al. studied the interdependencies between ROS packages on GitHub, BitBucket, and the rosdistro, and how quality propagates through the dependency network [10]. In an empirical study consisting of interviews and a survey with ROS developers, Estefo et al. investigated the difficulties that ROS users encounter when reusing ROS packages, main contribution bottlenecks in ROS ecosystem [4]. In a separate prior study, Estefo et al. studied code duplication in ROS packages [3]. Alami et al. conducted a qualitative study to better understand quality assurance practices within the ROS community [1]. Kolak et al. focused on ecosystem structure, collaboration, code reuse, and ecosystem health. They found that the most widely used ROS packages belong to a small cluster of foundational working groups (FWGs) [6].

The above is mainly concerned with the problems of ROS ecosystem and the status quo of ROS packages reuse but does not involve how to better realize ROS packages reuse.

B. Knowledge Graph in Robotics Development

Recently, knowledge graphs as a form of structured knowledge have drawn great research attention from both the academia and the industry [5], but few researchers have studied the knowledge graph about robotics development. Zamanirad et al. designed a bot programming platform that dynamically synthesizes natural language user expressions into API invocations and constructed an API knowledge graph to encode and evolve APIs to help robot understand the natural language spoken by humans [14]. Although their knowledge graph is applied in the field of robot, it is not specific to the robotics software development, what we are more concerned about.

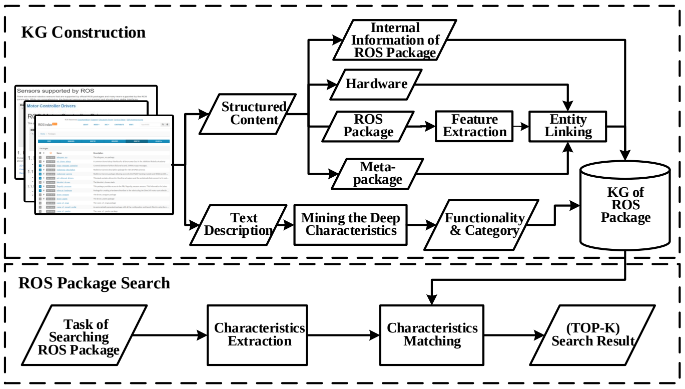

Fig. 1: The Overall Framework of Our Approach

III. Methodology

We propose a knowledge graph based approach to overcome the barriers mentioned above. Fig. 1 presents the key steps in our approach, which contains two main parts: mining ROS package knowledge graph from official website page information and searching ROS package based on the mined knowledge graph. We first use the popular web crawler tool Scrapy[4] to crawl structured content and text description. Then we extract package-related knowledge from structured content and establish connections, including the relevant content of the software package what it provides (launch file, service, plugin and message, etc), metapakckage (A set of ROS packages related to a certain function) and the hardware to which it belongs (sensor, motor or robot, etc). In order to get more characteristics information about ROS packages, we use natural language processing methods to obtain category and functionality from the description text to express ROS packages more abundantly. Since the description information of the ROS package is not very complete, and some even do not have it at all, therefore we extract features from the package name. At last, we build a knowledge graph called ROSKG based on the knowledge above for retrieval.

A. ROS Package Characteristic Extraction

Category and functionality: In order to dig out more information about the ROS package, we parse the package description sentence to extract the category and functionality of the package. We first use Stanford CoreNLP tool [9] to obtain the Part-of-Speech(POS) tags of the description sentence. Tokenization is used to break the text into words, phrases, or symbols. POS will represent the category of words which has similar grammatical properties. Then we

[4]https://scrapy.org/

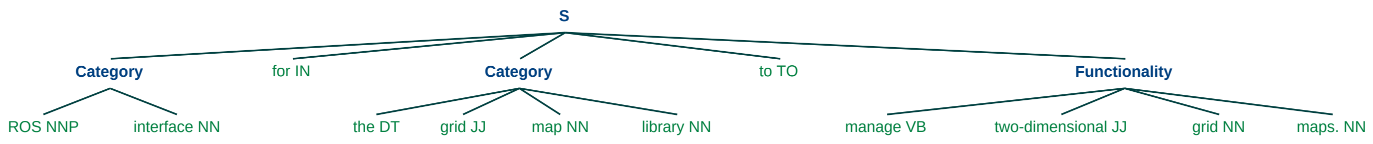

Fig. 2: Result of Category and Functionality Chunking

use rule-based chunking technique [15] to chunk category and functionality of the package. In our system, the category and functionality are identified by the regular expressions as shown in Table I. The terms in the table have the same meaning as [15]. Specifically, we extract verb (or verb phrase) followed by noun (or noun phrase) as functionality and noun (noun phrase) as category. We stipulate that noun and noun phrase in functionality do not belong to category. For example, The description of ROS package "grid_map_ros", "ROS interface for the grid map library to manage two-dimensional grid maps.", we chunk category "ROS interface" and "the grid map library"; functionality "manage two-dimensional grid maps" as shown in Fig. 2.

TABLE I: Regular Expression of Different Chunks

Name	Regular Expression
Functionality	(MD)*(VB.*)+(CD)*(DT)?(CD)*(JJ)*(CD)*-(VBD|VBG)*(NN.*)*(POS)*(CD)*(VBD|VBG)*-(NN.*)*(VBD|VBG)*(NN.*)*(POS)*(CD)*(NN.*)+
Category	(CD)*(DT)?(CD)*(JJ)*(CD)*(VBD|VBG)*(NN.*)*-(POS)*(CD)*(VBD|VBG)*(NN.*)*-(VBD|VBG)*(NN.*)*(POS)*(CD)*(NN.*)+

Feature: Not all ROS packages have description information, which are even incomplete or missing. It is not enough to rely on description information to characterize the ROS package. We notice that the words that make up a package name can well reflect its characteristics. For example, *turtlebot3* in the package *turtlebot3_navigation* reflects this package belonging to the robot *turtlebot3*, and *navigation* reflects that it is a package of the navigation type.

After the above analysis, we extract the last word of the package name of all ROS packages and analyze. However, there are many words that are just different in expression, but the actual meaning is the same in the robot software development process. So we classify words with the same meaning into one category and use one of the words to represent them. For example, "*msg*" is the abbreviation of "*message*", "*message*" is the singular form of "*messages*", so we use "*message*" to represent them. After the above processing, we select words that appear more than 5 times and analyze their actual meaning. Then we give their definitions. As shown in table II, we show the most frequent words (top 10) and their definitions.

TABLE II: Definitions of ROS Package Features (TOP 10)

Feature	Definition
Message	packages related to message types, which contains specific message definitions, e.g., geometry_msgs
Description	packages related to URDF description, e.g., heron_description
Config	packages automatically generated with all the configuration, including launch files and scripts, e.g., hironx_moveit_config
Driver	packages related to driver of hardware,such as camera, sensor, robot, e.g., dynamixel_driver
Gazebo	packages related to gazebo simulation, e.g., turtlebot3_gazebo
Control	packages related to controller to ensure the safe operation, e.g., roch_safety_controller
Support	packages to support to realize certain functions, e.g., choreo_kr5_arc_support
ROS	packages related to ROS wrapper, e.g., packml_ros
Tool	packages related to additional tool kit, e.g., nodelet_topic_tools

B. Linking Entity to ROS Package

Since the category and functionality of ROS package is originally extracted from the description of ROS package, the connection is naturally established. In this phase, we mainly introduce how to link the extracted entities to ROS package, including package feature introduced in Section III-A and hardware entities.

Hardware linking: Through the analysis of the package name above, we find that in general, the first word of the package name is likely to represent hardware information. So we construct an entity dictionary, which contains all hardware words, and we match hardware entities crawled from ROS Wiki with the dictionary to establish the linking. In order to more accurately match and establish relationship with ROS package, we have also sort out synonyms for hardware entities. For example, "*Velodyne HDL-64E 3D LIDAR*" is often referred to simply as "*Velodyne*". Furthermore, we take the hardware dictionary to match the description to establish potential relationship.

Feature linking: According to the source of the feature, it is mainly reflected in the package name. But we find that some packages contain more than one feature, such as "ainstein_radar_gazebo_plugins", which includes *gazebo* and *plugin* feature. So we split the package name according to the underscore and use the constructed feature dictionary to match to establish a relationship with ROS package. Hardware entities are generally more domain-oriented vocabularies, and feature entities are some commonly used vocabularies with a higher frequency, so hardware can establish potential relationships through description information. If the feature entities do the same, it may establish many relationships that don't

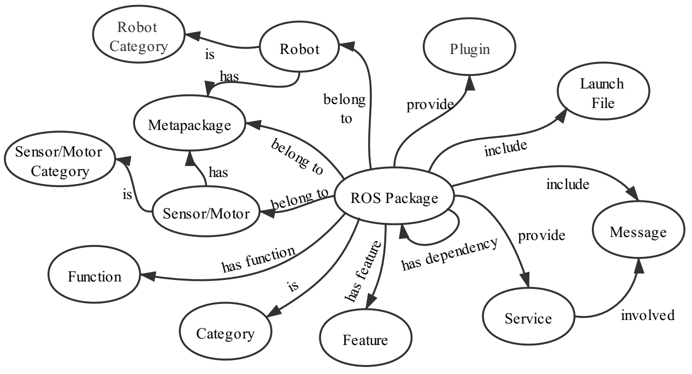

Fig. 3: Conceptual Schema of ROS Package Knowledge Graph

exist.

The other relationship can be naturally established by structured data, such as dependency, metapackage which the package belongs to. These relationships can establish relationships between packages that seem to be unconnected on the surface, for example, the ROS package "turtlebot3_navigation" depends on the other ROS package "amcl".

C. ROSKG Empowered ROS Package Search

For a specific ROS package searching, the ROSKG can group the ROS package that have the same attributes, e.g., sensor, robot, metapackage, etc, so developers can perform search tasks based on these attributes and view the related content, such as service, message, etc.

For ROS package searching related to specific tasks, given a query phrase or sentence in natural language description, which may not involve entities such as package names, we first use the parsing techniques mentioned in Section III-B to detect functionality and category in the query. We use the start-of-the-art pre-trained word embedding BERT [2] to separately vectorize the components in the query and the components in the description sentence in the knowledge graph. The functionalities and categories extracted from the description sentence have been vectorized in advance. We then compute the phrase similarity by the cosine similarity of phrase embedding. After matching the functionalities and categories compartments of the query sentence and the components in the ROSKG, we rank the ROS package in the ROSKG by the sum of the similarity of the matched counterparts. Finally we give top 10 ROS packages according to the similarity under the premise that above the user-specified threshold (generally 0.8).

D. Proof-of-Concept Implementation

We apply our knowledge graph construction methods described in Section III-B and Section III-C to the Kinetic distribution and use Neo4j[5], a graph database, to construct the knowledge graph for the ROS package. The resulting ROSKG consists of **25,484** entities and **62,854** relationships. Related concepts and their relations can be explained by the conceptual schema shown in Fig. 3.

[5]http://neo4j.com/

IV. EVALUATION

We conduct evaluations to explore the following research questions:

- **RQ1**: How is the intrinsic quality of the knowledge captured in the constructed ROSKG?
- **RQ2**: How does ROSKG perform in ROS package searching task compared with ROS Wiki engine?

A. RQ1: Quality of The Constructed ROSKG

The knowledge is extracted from two main sources: structured document content and textual description. Since knowledge extracted from structured information is intrinsically accurate, we mainly evaluate the accuracy of the knowledge extraction from text (i.e., functionality extraction, category extraction, feature linking).

1) Protocol: Similar to previous studies [8] [11] [13], we adopt a sampling method [12] to ensure that the ratio observed in the sample is within 5 confidence interval, and is extended to the population at 95% confidence level. We randomly select 349 functionalities, 369 categories, and 342 feature links to conduct the experiment.

Two developers (who are not involved in this study and familiar with ROS) independently perform the examination. All decisions are binary (the accuracy rates are *Acc1* and *Acc2* respectively). For the data instances that the two annotators disagree, they have to discuss and come to a consensus. We compute the final accuracy after resolving the disagreements (*AccF*) and compute Cohen's Kappa agreement (*Kap.*) [7] to evaluate the inter-rater agreement. Based on the consensus annotations, we evaluate the quality of the created knowledge about ROS package.

2) Results and Analysis: The results are shown in Table III. We can see the agreement rate are all above 0.78, indicating substantial or almost perfect agreement. The accuracy is generally high (above 0.91) except for the category extracted from description sentences(above 0.73).

TABLE III: Accuracy of ROS Package Knowledge

Aspect	Acc1	Acc2	AccF	Kap.
Functionality	92.3%	91.7%	92.0%	0.78
Category	77.0%	73.4%	75.3%	0.80
Feature Linking	100.0%	100.0%	100.0%	1.00

Typical problems of ROS package characteristics extraction include: 1) POS tagging or dependency parsing error, e.g., *"This package"* from sentence *"This package contains numerous examples of how to use SMACH"* is tagged as a noun; 2) meaningless characteristics, e.g., *"is a ROS-Package"* from sentence *"This is a ROS-Package for libviso2 a library for visual odometry.."* is extracted as a functionality of the ROS package *"libviso2"*; 3) incomplete sentences caused by incorrect HTML parsing or sentence splitting, e.g., "The move_base package provides an implementation of an action (see the"; 4) overly simplistic description sentence, e.g., description sentence just repeats the ROS package name, but is extracted as a category.

TABLE IV: Statistic of 11 Tasks including Subtasks in "ROS Robotics Projects"

Task	Subtask	Related package
1.Use ROS, opencv and dynamixel servo servos for face detection and tracking	1-1 driver for V4L USB camera	usb_cam
	1-2 servo motor for dynamixel	dynamixel_motor
	1-3 opencv	vision_opencv
2.Build a chatbot like Siri in ROS	2-1 to translate commands	sound_play
3.Use ROS to control embedded circuit boards	3-1 ROS for arduino platforms	rosserial_arduino
	3-2 ROS for TivaC Launchpad boards	rosserial_tivac
4.Operate the robot remotely using gestures	4-1 turtlebot simulation	turtlebot_gazebo
5.Object detection and recognition	5-1 object detection and recognition	find_object_2d object_recognition
6.Use ROS and TensorFlow for deep learning	6-1 convert ROS message to OpenCV image data type	cv_bridge
	6-2 use OpenCV capture object to capture camera image	cv_camera
7.Run ROS on MATLAB and Android	7-1 Android development package	android_core rosjava
8.Building an autonomous mobile robot	8-1 to generate maps	map_server
9.Use ROS to create self-driving cars	9-1 Velodyne HDL-64E 3D LIDAR	velodyne
10.Use VR headsets and Leap Motion to remotely control robots	10-1 gesture sensor	leap_motion
	10-2 visualization tool	rviz
11.Control the robot through the network	11-1 websocket interface	rosbridge_suite rosbridge_server
	11-2 to set and publish joint state values for a given URDF	joint_state_publisher
	11-3 HTTP Streaming of ROS Image in Multiple Formats	web_video_server

The feature linking have 100% accuracy, which is unsurprising because the features are extracted form package name and manually checked and filtered. So the feature linking can maintain a better result.

Our ROS package characteristics and relationship extraction methods for constructing ROS package knowledge graph are basically accurate, which can support practical use.

B. RQ2: Usefulness Evaluation

We evaluate the usefulness of ROSKG in ROS package searching tasks, that is, choosing the most suitable ROS package for the specific task.

1) Task: We extract the main tasks from the book "*ROS Robotics Projects*"[6] written by Lentin Joseph, which is an introductory book for ROS learners and has a high authority. As shown in Table IV, the book mentions 11 ROS robotics development tasks, and each task involves several subtasks, each of which contains 1-2 ROS packages. Finally, we summarize 18 subtasks, involving 21 ROS packages. The participants can formulate any query they wish based on the search task descriptions and the hints from previous search results.

2) Baseline: We use ROS Wiki's search engine as the baseline tool. For ROS-based robotics development, ROS Wiki is the most commonly used knowledge search community, especially for newcomers.

3) Protocol: We recruit 6 master students from our school and all of them have almost no ROS-based robotics software development experience. We believe these students are qualified for our study. Furthermore, they also simulate the target audience that our tool aims to assist, i.e., developers who may lack relevant knowledge in finding suitable ROS packages for the specific task. Then we randomly allocate them into two equivalent groups: the control group uses the ROS Wiki's search engine (P1-P3), while the experimental group uses our ROSKG to complete the tasks (P4-P6).

[6]http://wiki.ros.org/ROS_Robotics_Projects

TABLE V: Accuracy of ROS Package Knowledge

Experimental Group		
	Ave Task Time(seconds)	#Correct Answers
P1	55.4	13
P2	53.9	14
P3	56.4	16
Ave±stddev	55.2±1.0	
Control Group		
P4	79.8	12
P5	83.6	12
P6	88.2	14
Ave±stddev	83.9±3.4	

4) Results and Analysis: Table V shows the average task completion time and the number of correct answers by each participant in the two groups. We can see that two groups have the similar answer correctness but the experimental group complete the task faster with narrower standard deviation than the control group. The participant in the experimental group completed the tasks 52.0% faster (55.2 seconds on average) than the control group.

Through interviews with participants, we know that the control group participants often have to scroll the document back and forth in ROS Wiki and compare several documents to pinpoint and cross-validate the function of ROS packages. In contrast, the experimental group participants can view ROS package information in a more structured way, which can help them understand the function of the ROS package faster. We also look into the correct answer rate for each task, for some tasks, both search engines perform well, such as Task#6/11. But neither our search engine nor ROS Wiki performs well on Subtask#5-1. That's because the information about the ROS package “find_object_2d” cannot be obtained from our query statement.

ROS Wiki can return very relevant online documents for user queries, including tutorials, Q&A post and documents, but not everyone can find the correct ROS package through these resources. For example, the answer to Subtask#8-1 is hidden in a Q&A post, P6 finds it but P4&P5 don’t.

We find that the participants have difficulty in choosing since there are many similar packages related to the task. For example, P5 searches both “rviz” and “octovis” for Subtask#10-2. The relevant information for the two result actually contain “visualization tool”, but “octovis” is a specialized tool for “OctoMap”, which may not meet the current task. P3 makes the similar mistake when carrying out the Subtask#1-2.

One of the biggest problems with ROS Wiki search is that its search method is only keyword matching and only show one page. Once the searched keywords are not included in the relevant ROS package, it becomes difficult to retrieve and time-consuming. For example, participants in control group all return the wrong answer for Subtask#2-1, because the information of the highest ranked ROS package only contains keywords “commands” and “translate” separately, not phrases “translate commands”. But for our search engine, we use the fuzzy query method, even if it is not able to match keywords completely, we can return the relevant software package based on the similarity.

Although by no means conclusive due to the small-scale of our study, our pilot user study demonstrates that our approach significantly decreases the amount of time developers need for ROS package search tasks.

V. Threats to Validity

A threat to internal validity is that some software packages do not have descriptive information, that is to say, no textual information is provided, which will make it impossible to link to the corresponding software package via natural language. Another threat to internal validity is that our database is not complete enough to include all distributions of ROS packages. In the future, we will continue the collection work through automatic methods, which will contribute to further development.

The major threat to external validity is the generalization of our results and usefulness study is small scale. In the future, we will reduce this threat by applying our approach to more open tasks related to robotic software development and release our knowledge graph for public evaluation.

VI. conclusion

In this paper, we propose an efficient ROS package search approach based on knowledge graph. We leverage advanced NLP techniques for extracting the rich characteristics to better represent ROS packages. Our evaluation confirms the quality of different kinds of knowledge in the knowledge graph, and the usefulness of the generated ROS package search results. In the future, we will refine text processing techniques and design more rules to select meaningful characteristics to improve and extend our approach.

Acknowledgement

This work was supported in part by the National Key Research and Development Program of China under Grant 2018YFB1004202.

References

[1] Adam Alami, Yvonne Dittrich, and Andrzej Wasowski. Influencers of quality assurance in an open source community. In *2018 IEEE/ACM 11th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE)*, pages 61–68. IEEE, 2018.

[2] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[3] Pablo Estefó, Romain Robbes, and Johan Fabry. Code duplication in ros launchfiles. In *2015 34th International Conference of the Chilean Computer Science Society (SCCC)*, pages 1–6. IEEE, 2015.

[4] Pablo Estefo, Jocelyn Simmonds, Romain Robbes, and Johan Fabry. The robot operating system: Package reuse and community dynamics. *Journal of Systems and Software*, 151:226–242, 2019.

[5] Shaoxiong Ji, Shirui Pan, Erik Cambria, Pekka Marttinen, and Philip S Yu. A survey on knowledge graphs: Representation, acquisition and applications. *arXiv preprint arXiv:2002.00388*, 2020.

[6] Sophia Kolak, Afsoon Afzal, Claire Le Goues, Michael Hilton, and Christopher Steven Timperley. It takes a village to build a robot: An empirical study of the ros ecosystem. In *2020 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 430–440. IEEE, 2020.

[7] J Richard Landis and Gary G Koch. An application of hierarchical kappa-type statistics in the assessment of majority agreement among multiple observers. *Biometrics*, pages 363–374, 1977.

[8] Hongwei Li, Sirui Li, Jiamou Sun, Zhenchang Xing, Xin Peng, Mingwei Liu, and Xuejiao Zhao. Improving api caveats accessibility by mining api caveats knowledge graph. In *2018 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 183–193. IEEE, 2018.

[9] Christopher D Manning, Mihai Surdeanu, John Bauer, Jenny Rose Finkel, Steven Bethard, and David McClosky. The stanford corenlp natural language processing toolkit. In *Proceedings of 52nd annual meeting of the association for computational linguistics: system demonstrations*, pages 55–60, 2014.

[10] Marc Pichler, Bernhard Dieber, and Martin Pinzger. Can i depend on you? mapping the dependency and quality landscape of ros packages. In *2019 Third IEEE International Conference on Robotic Computing (IRC)*, pages 78–85. IEEE, 2019.

[11] Xiaoxue Ren, Xinyuan Ye, Zhenchang Xing, Xin Xia, Xiwei Xu, Liming Zhu, and Jianling Sun. Api-misuse detection driven by fine-grained api-constraint knowledge graph. In *2020 35th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 461–472. IEEE, 2020.

[12] Ravindra Singh and Naurang Singh Mangat. *Elements of survey sampling*, volume 15. Springer Science & Business Media, 2013.

[13] Chong Wang, Xin Peng, Mingwei Liu, Zhenchang Xing, Xuefang Bai, Bing Xie, and Tuo Wang. A learning-based approach for automatic construction of domain glossary from source code and documentation. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 97–108, 2019.

[14] Shayan Zamanirad, Boualem Benatallah, Moshe Chai Barukh, Fabio Casati, and Carlos Rodriguez. Programming bots by synthesizing natural language expressions into api invocations. In *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 832–837. IEEE, 2017.

[15] Xuejiao Zhao, Zhenchang Xing, Muhammad Ashad Kabir, Naoya Sawada, Jing Li, and Shang-Wei Lin. Hdskg: Harvesting domain specific knowledge graph from content of webpages. In *2017 ieee 24th international conference on software analysis, evolution and reengineering (saner)*, pages 56–67. IEEE, 2017.