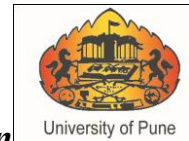




Knowledge Discovery in Java API Reference Documentation



Ms. M. J. Metkar, Prof. S. M. Kamalapur

Department of Computer Engineering, KKWIEER, Nashik-422003, Maharashtra, India.

Abstract Application Program Interface (API) allows programmers to use predefined functions instead of writing them from scratch. Description of API elements is provided through API Reference Documentation. Hence API Reference Documentation acts as a guide to user or developer to use API's. The work focuses on a study of the nature and organization of Knowledge contained in the reference documentation of the hundreds of API's provided as part of Java technology platform. Also, it involves the development of a classification of Knowledge automatically into Knowledge Types. And then classifying API Reference Documents according to the Knowledge Types.

Index Terms— **API, API Reference Documentation, Knowledge Types.**

I. INTRODUCTION

An Application Programming Interface (API) is a set of commands, functions, and protocols. It also specifies the interaction between the software components. In most procedural languages, an API specifies a set of functions or routines that accomplish a specific task or are allowed to interact with a specific software component.

For example:

```
static InputStream in
```

In above example, "in" is used for standard input stream.

Whenever user or developer is referring to an API and has planned to use it for specific purpose API Reference documentation works as a guide. API Reference Documentation is an important part of programming with APIs and it complements the API by providing information about the API. So, it plays a crucial role in how developers learn and use an API, and developers will expect the information about API elements they should find therein.

By considering the above example, if new developer wishes to use "in" field in Java Program, he can refer to API Reference Documentation of Java and he will find the description of "in" field in Field Summary as:

The "standard" input stream.

In above example Java Documentation is considered and Java APIs are documented through Javadocs which is a set of web pages such that one for each package or type in the API. Also, documentation for Python modules can be generated with the pydoc utility, and MSDN provides documentation for MSDN Technologies.

To enhance the quality of API reference documentation and the efficiency with which the relevant information it contains can be accessed, it's necessary to first understand its contents by analyzing it. Therefore, to reason about the quality and value of API reference documentation, focus should be about what knowledge it contains, and how this knowledge is organized. Because Knowledge refers to retrieve useful information from data and then use this knowledge for specific purpose. By analyzing the contents of API Reference Documentation Knowledge can be generated and this knowledge can be categorized further.

Previous work focused separately on Studies of Knowledge Categorization and of API reference Documentation and Knowledge retrieval was done based on Experience, Observations and Analysis.

So proposed system will focus on automated classification of knowledge types in API reference documentation.

Section II focuses on Literature Review. Section III gives Implementation Details with Block Diagram, Concept with Example and Algorithms are highlighted in. Data Sets required for the System are discussed in Section IV of Results. The paper ends with concluding remarks.

II. LITERATURE SURVEY

The previous work mainly focused on the Knowledge Categorization and API Reference Documentation Separately.

1. API Reference Documentation

Study of documentation needs for a domain-specific API, using surveys and interviews of developers was done by Nykaza

et al.[7] This study identified, among other requirements and the importance of an overview section in API documentation.

Jeong et al. [8] conducted a lab study with eight participants to assess the documentation of a specific service-oriented architecture. This study identified 18 guidelines they believe would lead to increased documentation quality for the system under study, including “explaining starting points” for using the API.

Robillard and DeLine [9] identified the obstacles faced by developers when trying to learn new APIs through surveys and interviews with Microsoft developers. The study showed that many obstacles were related to aspects of the documentation, but did not include the systematic analysis of API documentation content.

Similarly, Shi et al. [11] studied API documentation evolution. The authors apply data mining techniques over the source repository of five open-source APIs. Their study provides various quantitative measures of which parts of the API documentation are most frequently revised, and how often API documentation is changed consistently with the corresponding elements.

2. Knowledge Categorization based on Manual Methods

Researchers have applied Knowledge from one field to other field, they also studied which are the different questions raised in Software Project Development.

Mylopoulos et al.[1] discussed how knowledge representation techniques from the field of Artificial Intelligence can be applied to software engineering. The authors presented a categorization of different knowledge types, presumably derived from their experience.

Requirement and Design are the important stages in Software Project Development. Herbsleb and Kuwana[2] classified questions asked in design meetings to study the kinds of knowledge that may benefit from explicit capture at the requirements and design stages based on their general experience.

Hou et al.[4] studied 300 questions related to two specific Swing widgets (JButton and JTree) posted on the Swing forum. They then mapped the questions to the different design features of the widgets. Their classification focuses more on the target of the question and less on discovering the different types of knowledge provided to and sought by API users.

More recently, Ko et al.[3] observed 17 developers at Microsoft for a 90 minutes session each, studying their information needs as they perform their software engineering tasks. From the observation data the authors collected 334

specific information needs, which they abstracted into 21 general information needs.

Kirk et al.[5] investigated the knowledge problems faced by them and their students when trying to develop applications by extending the JHotDraw framework.

Similarly to Ko et al.’s study, Sillito et al.[6] produced a catalog of 44 types of questions developers ask during software evolution tasks. The questions in the catalog do not focus exclusively on API usage, but rather relate to software evolution and maintenance tasks.

So, researchers focused on how different stages of Software Project Development and tools required can be analyzed in different ways and they classified the Questions raised in different phases into different categories based on their Experience, Observations. Knowledge Types was not generated automatically.

Here, authors referred and studied API Reference Documentation in different ways. So, Separate study of Knowledge Categorization and API Reference Documentation was done previously.

The proposed work focuses on automation of Knowledge Types from API Reference Documentation.

III. DETAILS OF DISSERTATION WORK

This section explains Mathematical Model, the Proposed System, Concept in detail with example and Algorithms.

A. Mathematical Model

Let S be a Knowledge Discovery System, that discovers the Knowledge from API Reference Documents in Java such that,
 $S = \{D, T, I, K\}$

Where, D represents the set of API Reference Documents;

$D = \{D_0, D_1, D_2, D_3, \dots, D_n\}$

T represents the set of HTML Tags of respective API Reference Documents derived from all HTML Tags

$T = \{T_0, T_1, T_2, T_3, \dots, T_m\}$

I represents the set of Description of respective document derived from all HTML tags

$I = \{I_0, I_1, I_2, I_3, \dots, I_k\}$ where, $k < m$

K represents the set of Knowledge Types derived from the Description of HTML tags $K = \{K_1, K_2, K_3, K_4, \dots, K_j\}$ where $1 < j < 12$

Activities:

Activity 1: Let F_1 be a rule of D into T such that for given API Reference Document (HTML File) it gives HTML Tags.

$F_1(D) \rightarrow T$

Activity 2: Let F_2 be a rule of T such that for given HTML Tags it returns the description of Tags (those who have description) using Regular Expression

$F_2(T) \rightarrow I$

Activity 3: Let F3 be a rule of I such that for the given Description of Tags it returns the Knowledge Type using Knowledge Discovery Technique.

$$F3(I) \dashrightarrow K$$

Activity 4: Let F4 be a rule of D such that for the given API Reference Documents it classifies them according to the Knowledge Types.

$$F4(D) \dashrightarrow K$$

Functional Dependency Graph:

Functional Dependency Graph of the above functions is shown below:

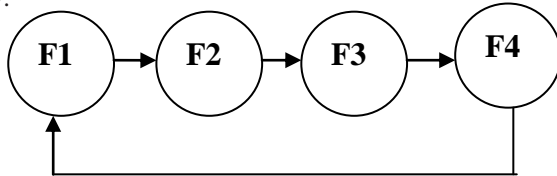


Fig 1. Functional Dependency Diagram

Functional Dependency Matrix:

Functional Dependency Matrix of the above functions is shown below:

TABLE I. FUNCTIONAL DEPENDENCY MATRIX

	F1	F2	F3	F4
F1	1	0	0	1
F2	1	1	0	0
F3	1	1	1	0
F4	1	1	1	1

B. Block Diagram of the Proposed System

The following figure explains the Block Diagram of Proposed System:

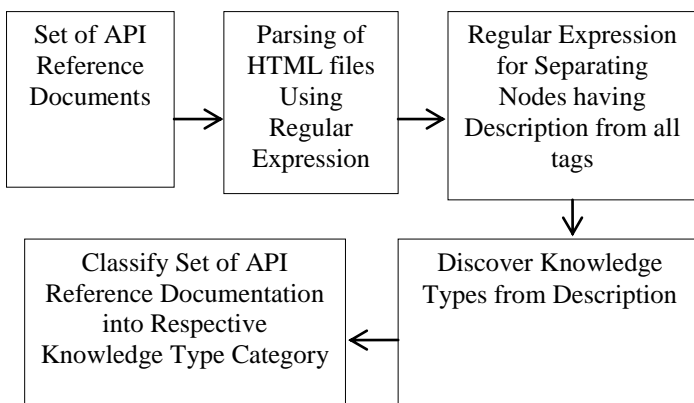


Fig 2. Block Diagram of the Proposed System

Proposed system focuses on API Reference Documents of Java that is Javadocs. Input to the system will be API Reference Document of Java that is HTML Page.

The API Documentations are parsed using regular expressions and HTML Tags are separated.

In the next step, Knowledge Discovery Technique will be applied to the Description of API elements and then Knowledge Types will be generated automatically.

After generating the Knowledge Types, set of API Reference Documents will be classified into the respective

Knowledge Types. Here one API Reference Document may be classified into more than one knowledge types.

Following are the 12 Knowledge Types:

1. Functionality and Behavior: This Knowledge Type describes functionality and features of API. And also specifies what happens when the API is used.

e.g.: protected boolean enabled

Specifies whether action is enabled; the default is true.

2. Concepts: Concepts explains the meaning of the terms used to name or describe an API element, or describes design or domain concepts used or implemented by API.

e.g.: public class AccessibleAttributeSequence extends Object
This class collects together the span of text that share the same contiguous set of attributes, along with that set of attributes. It is used by implementors of the class AccessibleContext in order to generate ACCESSIBLE_TEXT_ATTRIBUTES_CHANGED events.

3. Directives: It is related to accessibility that is what users are allowed or not allowed to do with the API element. Directives are clear contracts.

e.g.: public class AccessException extends RemoteException
An AccessException is thrown by certain methods of the java.rmi.Naming class (specifically bind, rebind, and unbind) and methods of the java.rmi.activation.ActivationSystem interface to indicate that the caller does not have permission to perform the action requested by the method call. If the method was invoked from a non-local host, then an AccessException is thrown.

4. Purpose and Rationale: This Knowledge Type specifies the purpose of providing an element or the rationale of a certain design decision. It is having this information that answers a "why" question that is Why is this element provided by the API? Why this is designed this way? Why would we want to use this?

e.g.: protected static class AbstractRegionPainter.PaintContext extends Object

A class encapsulating state useful when painting. Generally, instances of this class are created once, and reused for each paint request without modification. This class contains values useful when hinting the cache engine, and when decoding control points and bezier curve anchors.

5. Quality Attributes and Internal Aspects: This describes quality attributes of the API, also known as non-functional requirements, for example, the performance implications.

Also applies to information about the API's internal implementation that is only indirectly related to its observable behavior.

e.g.: public abstract class AbstractWriter extends Object
AbstractWriter is an abstract class that actually does the work of writing out the element tree including the attributes. In terms of how much is written out per line, the writer defaults to 100. But this value can be set by subclasses.

6. Control-Flow: How the API (or the framework) manages the flow of control is described by this knowledge type. For example by stating what events cause a certain callback to be triggered?

e.g.: `protected void fireIntervalRemoved(Object source, int index0, int index1)`

`AbstractListModel` subclasses must call this method after one or more elements are removed from the model.

7. Structure: Structure specifies the internal organization of a compound element (e.g. important classes, fields, or methods), information about type hierarchies, or relation of elements with each other.

e.g.: `Set<String> getSupportedAnnotationTypes()`

If the processor class is annotated with `SupportedAnnotationTypes`, return an unmodifiable set with the same set of strings as the annotation.

8. Patterns: It describes how to accomplish specific outcomes with the API, for example, how to implement a certain scenario, how the behavior of an element can be customized, etc

e.g.: `public abstract class AbstractColorChooserPanel extends JPanel`

This is the abstract superclass for color choosers. If you want to add a new color chooser panel into a `JColorChooser`, subclass this class.

9. Code Examples: Code examples are provided for how to use and combine elements to implement certain functionality or design outcomes.

e.g.: `public abstract class AbstractExecutorService extends Object implements ExecutorService`

Provides default implementations of `ExecutorService` execution methods. This class implements the `submit`, `invokeAny` and `invokeAll` methods using a `RunnableFuture` returned by `newTaskFor`, which defaults to the `FutureTask` class provided in this package. For example, the implementation of `submit(Runnable)` creates an associated `RunnableFuture` that is executed and returned. Subclasses may override the `newTaskFor` methods to return `RunnableFuture` implementations other than `FutureTask`.

Extension example. Here is a sketch of a class that customizes `ThreadPoolExecutor` to use a `CustomTask` class instead of the default `FutureTask`:

```
public class CustomThreadPoolExecutor extends
ThreadPoolExecutor {
    static class CustomTask<V> implements RunnableFuture<V>
    { ... }
    protected <V> RunnableFuture<V>
    newTaskFor(Callable<V> c) {
        return new CustomTask<V>(c);
    }
    protected <V> RunnableFuture<V> newTaskFor(Runnable r,
    V v) {
        return new CustomTask<V>(r, v);
    }
    // ... add constructors, etc.
}
```

10. Environment: Aspects related to the environment in which the API is used is described in this type, but not the API directly, e.g., compatibility issues, differences between versions, or licensing information.

e.g: `public abstract class AbstractElementVisitor7<R,P> extends AbstractElementVisitor6<R,P>`

A skeletal visitor of program elements with default behavior appropriate for the `RELEASE_7` source version.

11.External References: It includes any pointer to external documents, either in the form of hyperlinks, tagged "see also" reference, or mentions of other documents (such as standards or manuals).

e.g: `public interface DOMLocator`

`DOMLocator` is an interface that describes a location (e.g. where an error occurred).

See also the Document Object Model (DOM) Level 3 Core Specification.

12. Non-information: A section of documentation containing any complete sentence or self-contained fragment of text that provides only uninformative boilerplate text.

e.g: 1. `DefinitionKindHelper()`

2. `static DefinitionKind extract(Any a)`

C. Concept in detail with example

a) Consider following HTML file as Input: In this example, one of the class of Javadocs , named `void` is taken into consideration.

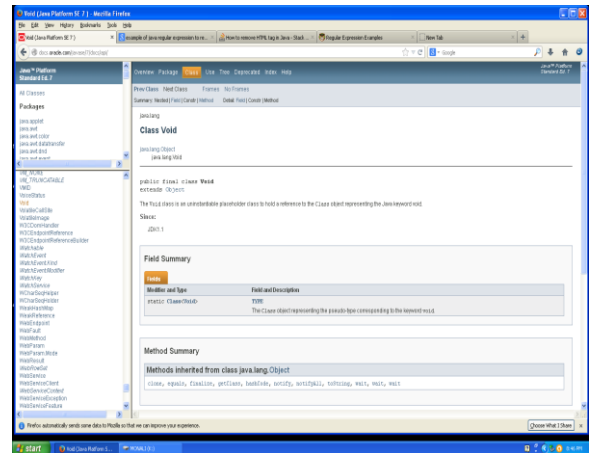


Fig 3: Example of the API Reference Document

b) Parsing of HTML document is done using following two techniques:

1. In the first technique, HTML file is considered as Input and DOM Tree is constructed for the HTML tags. From the DOM Tree, Regular Expression will be created. But Computational Time required for the DOM Tree Construction and then creation of Regular Expression from DOM tree is more and hence second technique will be considered.

2. In the second technique, the HTML page is taken as input , and parsing of this HTML page will be done by forming Regular Expression to fetch the tags having description.

Let us consider following example of generation of Regular expression for the title tag in Java Regex:

Pattern `p = Pattern.compile("<title>(.*)</title>");`

As shown in the above pattern , here the regular expression is generated for separating title tag from HTML page.

Using the same principal, Regular Expression Pattern will be formed for Parsing HTML document and separating the tags having the description of API elements.

c) After separating the tags having the description, next step is to Apply Knowledge Discovery Technique for generating the Knowledge Types for the given API Reference Document. Here description of one API elements on the API Reference Document may fall under more than one Knowledge Types.

d) After generating the Knowledge Types, set of API Reference Documents will be classified to respective Knowledge Types. Here one document may fall under more than one Knowledge Types.

D. Algorithms

1) Parsing of HTML Files to fetch Description:

a) Initially, one of the Javadocs pages that is API Reference Document which is to be processed is taken as input.

b) Source code of the Javadocs is HTML tags and hence the actual input to the first step is HTML and JavaScript tags.

c) After taking HTML tags as input, the next step is to parse the HTML tags to fetch the tags having description.

d) So, to fetch the description of API element from the current page Regex package in Java will be used.

e) Here, all HTML tags are scanned and then the Regular Expression is created to locate the tags having the description of an API element.

f) That is, Pattern and Matcher will be used for fetching the nodes having the description of an API element.

For example, to fetch the Method Details from all the tags, we will create a pattern of Regular Expression under the heading tag where the word “Method Detail” is specified. And then using Matcher we will fetch the description.

2) For Generation of Knowledge Types for the Description of API elements:

a) After fetching the description of API elements in second step, next step is to process this description.

b) To process the description of the API elements, the patterns of the Descriptions are observed, that is whether the descriptions are having some common words in them or they are starting with same words or having some common format.

c) So, after finding some common patterns in the descriptions, the Knowledge Representation Technique will be applied for generation of Knowledge Types.

That is description will be classified to the appropriate Knowledge Type. For Example, Description of all API elements will have common Knowledge Type as Functionality and Behavior.

IV. RESULT

A. Data Set

The Data Set for the proposed system are set of API Reference Documents. The jdk-7u51-apidocs.zip file contains the set of API Reference Documents for Java. The above said file can be obtained by using following link:

<http://www.oracle.com/technetwork/java/javase/documentation/java-se-7-doc-download-435117.html>

B. Expected Results

Consider the API Reference Documentation in Fig.4 for Class AbstractListModel<E> :

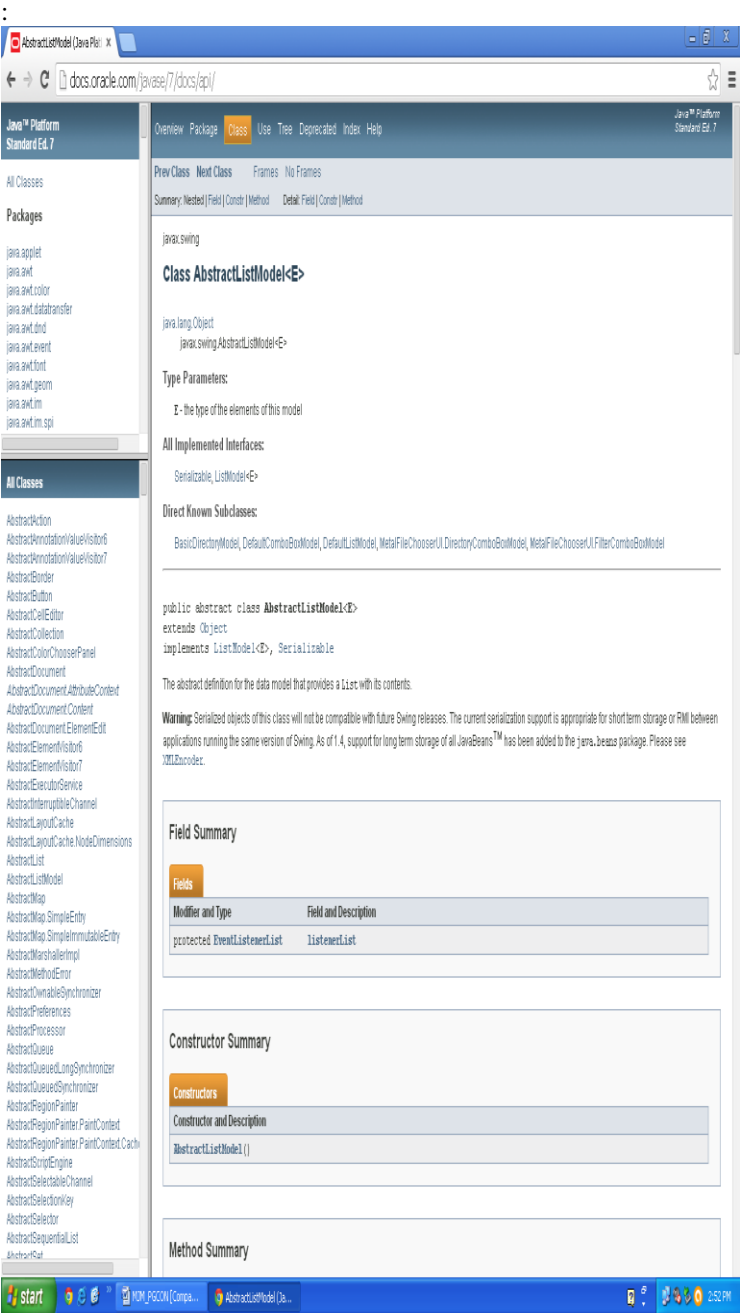


Fig 4: Example of the API Reference Document

For above API Reference Document, following Knowledge Types will be generated:

TABLE II: TABLE SHOWING RESULT SET FOR CLASS ABSTRACTLISTMODEL<E>

Other Details of Class	Description	Knowledge Types that are true for the Description (From all 12)

	Class Description: The abstract definition for the data model that provides a List with its contents.	1.Functionality and Behavior 2.Purpose and Rationale
Fields under above class		
protected EventListenerList	.	Non-information
Methods in above class		
void addListener(DataListener listener)	Adds a listener to the list that's notified each time a change to the data model occurs	1.Functionality and Behavior 2.Purpose and Rationale
protected void fireContentsChanged(Object source, int index0, int index1)	AbstractListModel subclasses must call this method after one or more elements of the list change.	1.Functionality and Behavior 2.Purpose and Rationale 3.Control-Flow
void addListener(DataListener listener)	Adds a listener to the list that's notified each time a change to the data model occurs	1.Functionality and Behavior 2.Purpose and Rationale
protected void fireIntervalAdded(Object source, int index0, int index1)	AbstractListModel subclasses must call this method after one or more elements are added to the model.	1.Functionality and Behavior 2.Purpose and Rationale 3.Control-Flow
protected void fireIntervalRemoved(Object source, int index0, int index1)	AbstractListModel subclasses must call this method after one or more elements are removed from the model.	1.Functionality and Behavior 2.Purpose and Rationale 3.Control-Flow
ListDataListener[] getListDataListeners()	Returns an array of all the list data listeners registered on this AbstractListModel	1.Functionality and Behavior 2.Purpose and Rationale
<T extends EventListener> T[] getListeners(Class<T> listenerType)	Returns an array of all the objects currently registered as <i>FooListeners</i> upon this model.	1.Functionality and Behavior 2.Purpose and Rationale

Void removeListener(DataListener listener)	Removes a listener from the list that's notified each time a change to the data model occurs	1.Functionality and Behavior 2.Purpose and Rationale
Constructor in above class		
AbstractListModel()		Non-information

V. CONCLUSION

API's are used as interface for using predefined functions, packages, classes etc. Developers read API reference documentation to learn how to use the API and answer specific questions they have during development tasks. Thus API Reference Documentation provides guide to user for referring to API. API Reference Documentation contains description of API elements; this description will be analyzed for generating Knowledge. Proposed system focuses on classification of description of API elements into different Knowledge Types. And then set of API Reference Documentations will be categorized in Knowledge Types.

REFERENCES

- [1] J. Mylopoulos, A. Borgida, and E. Yu, "Representing software engineering knowledge," *Automated Software Engineering*, vol. 4, no. 3, pp. 291–317, 1997.
- [2] J. D. Herbsleb and E. Kuwana, "Preserving knowledge in design projects: what designers need to know," in *Proceedings of the Joint INTERACT '93 and CHI '93 Conferences on Human Factors in Computing Systems*, 1993, pp. 7–14.
- [3] A. J. Ko, R. DeLine, and G. Venolia, "Information needs in collocated software development teams," in *Proceedings of the 29th International Conference on Software Engineering*, 2007, pp. 344–353.
- [4] D. Hou, K. Wong, and J. H. Hoover, "What can programmer questions tell us about frameworks?" in *Proceedings of the 13th International Workshop on Program Comprehension*, 2005, pp. 87–96.
- [5] D. Kirk, M. Roper, and M. Wood, "Identifying and addressing problems in object-oriented framework reuse," *Empirical Software Engineering*, vol. 12, pp. 243–274, June 2007.
- [6] J. Sillito, G. C. Murphy, and K. D. Volder, "Asking and answering questions during a programming change task," *IEEE Transactions on Software Engineering*, vol. 34, no. 4, pp. 434–451, July–August 2008.
- [7] J. Nykaza, R. Messinger, F. Boehme, C. L. Norman, M. Mace, and M. Gordon, "What programmers really want: Results of a needs assessment for SDK documentation," in *Proceedings of the 20th Annual ACM SIGDOC International Conference on Computer Documentation*, 2002, pp. 133–141.
- [8] S. Y. Jeong, Y. Xie, J. Beaton, B. A. Myers, J. Stylos, R. Ehret, J. Karstens, A. Efeoglu, and D. K. Busse, "Improving documentation for eSOA APIs through user studies," in *Proc. 2nd Int'l Symp. on End-User Development*, ser. LNCS, vol. 5435. Springer, 2009, pp. 86–105.
- [9] M. P. Robillard and R. DeLine, "A field study of API learning obstacles," *Empirical Software Engineering*, vol. 16, no. 6, pp. 703–732, 2011.
- [10] Walid Maalej and Martin P. Robillard, *Patterns of Knowledge in API Reference Documentation*. IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. 39, NO. X, XXXXXXXX 2013
- [11] L. Shi, H. Zhong, T. Xie, and M. Li, "An empirical study on evolution of API documentation," in *Proceedings of the Conference on Fundamental Approaches to Software Engineering*, 2011, pp. 416–431.