A Concept for the Design of Learning Resources for Application Programming Interfaces of Content Management Systems

Sirma Gjorgievska
Faculty of Informatics, Technical University Munich
Email: sirma.gjorgievska@tum.de

***Abstract*—Application Programming Interface (API) is a set of routines, protocols, and tools for building software applications. Insufficient or inadequate explanation and examples of the APIs structure and its usage present an obstacle for developers trying to learn an API. As a result, well documented APIs that enhance the experience for developers, became an essential requirement for defining an API's success. The most common way to document APIs today is to produce documentation that accurately lists the API endpoints, and describes the allowable operations on each, making the APIs more precise and readable. There are various tools that allow you to do this in an automated or semi-automated way. However, there is a lack of a common and proven concepts of how to introduce the developers to an API and train them how to integrate APIs into their systems respectively. Therefore, we propose A Concept for the Design of Learning Resources for APIs of Content Management Systems (CMS) which will allow developers to easily comprehend and learn how to use the specific API endpoints. The evaluation of our proposal, has been made on a next generation CMS system SocioCortex and its REST API, which we have extended with our proposed model.**

I. Introduction

Application Programming Interface is the interface to implemented functionality that developers can access to perform various tasks [1]. API allows one program to access the data and services provided by another program, which makes programming easier. Because of that in recent years, the number of web applications offering Application Programming Interfaces has increased dramatically [2]. Nowadays most of the software projects reuse components exposed through APIs. In fact, current-day software development technologies are becoming inseparable from the large APIs they provide. Beside all of the significant benefits APIs offer in the software development, learning and using an API presents one of the biggest challenges for developers. The most common way for learning an API is through API documentation, which is a technical writing on functionalities and usages of API elements like classes and methods. The quality of API documentation is varying a lot, but it can be critical for software developers. Documentation, on one hand, can help developers work efficiently and it can even serve to promote the API. On the other hand, documentation that does not meet its readers expectations in the worst case can lead to abandonment of the API. Although providing high-quality API documentation is obviously desirable, studies suggest that current approaches to developing and delivering documentation may not provide software developers with what they need. Most of the API's documentation today have one or more of the following problems: incompleteness, ambiguity, unexplained examples, obsoleteness, inconsistency and incorrectness. Because of that there are numerous researches analyzing the advantages and disadvantages of the current approaches of designing an API documentation. However, there is a lack of a common and proven concept of how to introduce the developers to an API and train them how to integrate APIs into their systems respectively. Therefore, we propose a conceptual model of API documentation. The model defines the guidelines for creating an API documentation that will be easy to use and will provide developers with the tools for more efficiently integrating specific API into their system. We have structured our proposed model into seven high-level guidelines: up-to date documentation, documentation of the API's high-level design, quickstart, tutorial, best practices, API reference and multiple ways of navigation. In addition, we have developed a prototype implementation in order to assess our proposed model. As part of the prototype, we have implemented a contemporary web application, which is based on AngularJS and integrates with well-established CMS (Content Management System) platform SocioCortex. A CMS is a computer application that supports the creation and modification of digital content using a common user interface and thus usually supporting multiple users working in a collaborative environment.

II. Related Work

A Concept for the Design of Learning Resources for Application Programming Interfaces of Content Management Platforms has not been addressed yet in a way comparable to the approach proposed in this paper. However, some research that elaborates the current approaches to design and present API documentation has already been made.

The work of Christopher Scaffidi [3] focused on four specific challenges related to learning and using APIs. The discussion of each challenge includes an outline of strategies that API users employ for dealing with that challenge, as well as strategies that API designers utilize for helping their respective API users. Even though this article elaborates why using an API is often difficult and emphasize the challenge of inadequate documentation, it does not address the exact shortcomings of the current API documentations.

The research conducted by Martin P. Robillard [1] address the obstacles that Microsoft developers faced when learning to use APIs. A major result of the survey is that learning resources topped the list of obstacles to learning APIs. In particular, insufficient or inadequate examples, the lack of reference on how to use the API to accomplish a specific task as well as having insufficient documentation on the high level aspect of the API were among the most commonly found obstacles. The study presented results at a high level of abstraction (e.g., inadequate documentation on high-level design), but failed to detail what these attributes mean in practice (e.g., what makes for inadequate).

Another study of people's needs and preferences with respect to documentation was conducted by David G. Novick and Karen Ward [4]. The study tries to find an answer to the one of the most important questions "What do users want in a documentation". In order to answer this question the authors conducted series of interviews with 25 computer users representing a cross-section of uses in work settings. The study's participants indicated that they preferred documentation that is easy to navigate, provides explanations at an appropriate level of technical detail, enables finding as well as solving problems through examples and scenarios, and is complete and correct. While this study addressed many of the preferences of users for API documentation, it did not clearly provide guidelines of how to design good API documentation.

In addition, Mitchell [5] reported the results of a comprehensive study conducted by IBM. The study had three research thrusts: supporting issues that influence attitudes toward technical documentation, attitudes toward documentation itself, and importance of and satisfaction with key documentation attributes. It found that users were very irritated with a lack of concrete examples, difficulty in understanding the documentation, a lack of relevant information, problems finding the information they need, and failure to address the why and how of a specific task. Mitchell concluded that users want information that is clear, accurate, and loaded with examples and scenarios.

One aspect that none of these studies covered is that even if the documentation is well written, most of the readers begin to try to use the system without reading the whole manual. As, Brockmann, in his paper "The Why, Where, and How of Minimalism" [6] notices that people avoid instructions and simple steps and can only understand through the effectiveness of their actions in the world. According to Brockmann adult learners are impatient and want to get started quickly on something productive, they skip around in manuals and online documents and rarely read them fully, make mistakes but learn most often from correcting such mistakes and they are best motivated by self-initiated exploration. Accordingly, our goal in this study was to propose a solution how to keep readers engaged and help them learn more quickly and efficiently.

III. Shortcomings of current API learning resources

While some earlier studies have provided a partial view of the shortcomings of the current API learning resources, none of the current researches give a consolidated, complete list of all shortcomings with more detailed explanation of each. Based on reviewing related research as well as current state of art practices, we identified five main shortcoming categories of current API learning resources, namely: out of date documentation, incomplete and incorrect documentation, ambiguous documentation, difficulty finding the right information and poorly presented documentation. You can see the list of all shortcomings in the below Table I.

First, one of the most commonly find problems with the current API learning approaches is that they are often out of date and do not reflect the most current changes. Learning and using APIs can be difficult for reasons stemming from the very nature of software. Software can evolve quickly, which means that APIs can rapidly become outdated [3] [7] [8] [9] [10] [11].

Second, documentation is often incomplete and incorrect. Documentation can be incomplete due to many different reasons. In the trivial case, it can be because of the lack of effort invested in it. Auto-generated documentation can sometimes be example of such incompleteness, with no documentation at all for many classes or methods. However, even when earnest effort is invested in documenting an API, it can be difficult to anticipate all the ways it can be used. When an API element might be involved in complex interactions with other API elements, understanding how to use it might require more than the description of its functionality [4] [5] [8] [11] [12]. Moreover, there is schedule pressure due to a desire to be first to market, so designers lack time for creating as much API documentation as might be desired. Because of that reason many documentations lack concrete code examples, which fail to explain developers the specific usage of an API element [4] [1]. Even if there are some code examples, they often do not have an adequate explanation [12]. Furthermore, the progress of the user can be hindered with no sufficient knowledge of the API's high level design. In fact, if readers do not understand the design intents behind the API and the overall architecture, they can have problems to choose among alternative ways to use the API, structure their code accordingly, and employ it as efficiently as possible. Additionally moving beyond trivial usage involves many types of decisions that can be informed by high-level design and still most of the today's approaches lack this important component [1].

Third, learning resources for API can be ambiguous with unclear description of API elements. In this case, the documentation appears to cover a topic of interest but leaves out important details so that multiple interpretations are possible, which leads to confusion or incomprehension [12]. Moreover, many readers have problems with documentation clarity, because it tends to be too complex for novice users with unfamiliar technical jargon and extraneous information that

does not help solve their problem. Developers that are in a process of learning the API can have difficult time making the right choice among many alternative usages of an API along with determining, which functions are optional and which are required in order to achieve complex effects [1]. On the other hand, the content should not become very basic and simplified, because then documentation can easily become too general without offering more detailed and specific knowledge of the usage and functionality of the API elements [4].

Fourth, many APIs learning resources have grown very large and diverse, making it harder for the readers to easily find information they need. Many large documentations fail to provide good navigation and developers have difficulty finding useful search terms and locating solutions to problems quickly [4] [5] [11]. Furthermore, users of the APIs are often under time pressure, so they are unwilling to invest time in wandering through API documentations. Because of that reason, it is very important for the documentation to have good navigation system, enabling users to more effectively use the API.

The last shortcoming we found in today's API documentation approaches is their poor presentation. Presentation of the documentation is sometimes bloat, which means that the description of an API element or topic is verbose or excessively extensive. The risk with associating large chunks of text to a specific element or topic is that readers will not readily be able to determine whether the text provides the information they seek, especially when the title or header is general [12]. Another problem is fragmentation of documentation. In fact, developers have problems understanding a documentation, when the information related to a specific element is fragmented or scatted over too many pages. When developers have to click through multiple pages of an API document to learn the functionality and use of an API element, they found the separation of the descriptions at such a micro level to be unnecessary [12]. Moreover, if the documentation contains instructions presented in some help displays and text editor where the user should enter some commands, users can have difficulty to both read and carry out the instructions at the same time. In this case the instructions should not cover or be covered by the application window.

IV. A CONCEPTUAL MODEL FOR API DOCUMENTATION

Based on the identified shortcomings and our own experiences in using several real-world APIs, we have proposed a novel conceptual model for API documentation that addresses the before mentioned concerns. We have structured our proposed model into seven high-level guidelines: up-to date documentation, documentation of the API's high-level design, quickstart, tutorial, best practices, API reference and multiple ways of navigation.

A. Up-to date documentation

First and the most basic guideline is that documentation should be up to date with the latest changes. If the documentation does not reflect the most recent changes, the users may not find the information they need, which in worst case can lead to abandonment of the API.

Furthermore, the documentation has to be complete and correct, by providing information that includes everything necessary to learn an API and to prepare the developer to successfully use the API and resolve problems [5].

B. Documentation of the API's high-level design

Documentation should present the API's high-level design. API's high-level design explains the architecture that is used for developing the API. There can be multiple ways of presenting the high-level design leveraging UML approach: architectural diagram, use case diagram, component diagram and class diagram.

The architectural diagram provides an overview of an entire system, identifying the main components that are developed and their interfaces. More detailed is the class diagram, which describes the structure of a system by showing the system's classes, their attributes, methods and the relationships among objects. In addition, user's interaction with the system can be presented using the use case diagram that represents the relationship between the user and the different use cases in which the user is involved. A use case diagram can identify the different types of users of a system and the different use cases and will often be accompanied by other types of diagrams as well.

The purpose of presenting the API's high-level design in a documentation is to help developers choose among alternative ways to use the API, structure their code accordingly, and employ it as efficiently as possible.

C. Quickstart

Developers who use the API for the first time should have instructions for installing and setting up the development environment. These instructions should be step-by-step guiding through the process of installation and should be described in a comprehensive manner, so readers from different level of expertise can understand them. It is crucially important for the documentation to provide help for all the users of the API and it should not assume that developers have the same level of knowledge, which means that the information should not be either too high-level or too basic [4].

D. Tutorial

As mentioned earlier in Section II, people avoid instructions and simple steps and can only understand through the effectiveness of their actions in the world. Furthermore, developers most of the time learn best by self-initiated exploration and by making and correcting mistakes. Because of that a documentation should offer readers a possibility to interactively try out the API invocations directly in the browser. This can be done by providing tasks which users have to solve. The level and difficulty of the provided tasks should be different and should be gradually increased.

Furthermore, the interface for interactive tutorial should be designed in the manner to make it easy for the developers to both read and carry out the instructions at the same time.

Category	Shortcomings	Research
Out of date documentation	• A documentation is obsolete	• Why are APIs Difficult to Learn and Use [3] • A study of the documentation essential to software maintenance [7] • Measuring API Documentation on the Web [8] • Live API Documentation [9] • Investigating Web APIs on the World Wide Web [10] • How software engineers use documentation state of practice [11]
Incomplete/incorrect documentation	• Lack of concrete examples • Unexplained examples • Lack of knowledge of API's high-level design	• What users say they want in documentation [4] • What Do Users Really Want From Computer Documentation [5] • Measuring API Documentation on the Web [8] • How software engineers use documentation state of practice [11] • How API Documentation Fails [12] • What Makes APIs Hard to Learn? Answers from Developers [1]
Ambiguous documentation	• Unclear description of elements • Too complex description for novice users • Too general documentation • Choose right function • Determine which functions are optional	• How API Documentation Fails [12] • What users say they want in documentation [4] • What Do Users Really Want From Computer Documentation [5] • What Makes APIs Hard to Learn? Answers from Developers [1] • Why are APIs Difficult to Learn and Use [3]
Finding the right information	• Difficulty locating solutions to problems quickly	• What users say they want in documentation [4] • What Do Users Really Want From Computer Documentation [5] • How software engineers use documentation state of practice [11]
Poorly presented documentation	• Bloat presentation • Fragmented documentation • Difficulty finding right information	• How API Documentation Fails [12] • What users say they want in documentation [4] • What Makes APIs Hard to Learn? Answers from Developers [1]

TABLE I: Categorization of shortcomings

E. Best practices

Documentation should include relevant examples, with explanations at a length and level of complexity that is appropriate to the users knowledge [4] [5]. Users benefit a lot from examples that show the best practices of an API's use [1]. These examples should demonstrate actual (non-theoretical) uses of the API and should be commented appropriately. If developers can see the code snippets for particular programming language, and know that the code works to produce the desired result, then he/she can immediately put that into their code and modify as needed.

F. API reference

API reference presents a list of all API's functionalities. The API reference can be organized by resource type, where each resource type has one or more methods. Moreover, there should be a detailed description for each class and method. The description of API elements and topics in the documentation should be short and precise, and therefore allow the readers to easily and quickly determine whether the text provides the information they seek. Furthermore, as mention in the Section III a common mistake made in the current API documentation approaches is that the information is fragmented or scatted over too many pages [12]. To avoid that, the description related to a specific API element should be presented in a single page.

G. Multiple ways of navigation

As APIs are growing bigger, it is essentially important for the users to be able to easily navigate through the documentation. In fact, users do not want to spend too much time wandering through the APIs, unable to find the information they need. Because of that documentation should provide multiple ways of navigation and by that enable users to locate solutions to problems quickly and easily [4] [1] [5]. In order to do so documentation should include:

- Overview page
  - Quickstart
  - Tutorials
  - Best practices
  - API reference
- Categories
- Full text search
- Link to related resources

The overview page is designed for developers who are first time encountering with the API documentation. This page presents the main, before mentioned categories in the

documentation and offers the user to easily choose which one he/she wants to start with. There are four categories presented: quickstarts, tutorials, best practices and API reference. Depending on the user level of expertise and how good he/she wants to learn the API, different categories are recommended. The purpose of the overview page is to provide appropriate level of help for novice as well as for expert users. Novice users should first start with quickstart, which gives instruction for installing and setting the development environment. However, expert users who already know those steps, can directly go to API reference, which gives a comprehensive listing of all functionalities of the API.

Furthermore, for easy navigation, the sidebar should be divided by few categories and should have full text search facility. Besides that, good online documentation should include pointers to additional sources of information [4].

V. Prototype

We have assessed our proposed model by developing a prototype implementation. As part of prototype implementation, we have developed a documentation module as a supplement to a well-established Content Management Platform SocioCortex [13]. The documentation module has been developed as an AngularJS web application that integrates with the SocioCortex backend.

AngularJS is an open-source web application framework mainly maintained by Google and by a community of individuals and corporations to address many of the challenges encountered in developing single-page applications. It aims to simplify both the development and the testing of such applications by providing a framework for client-side ModelViewController (MVC) and ModelViewViewModel (MVVM) architectures, along with components commonly used in rich Internet applications. The AngularJS framework works by first reading the HTML page, which has embedded into it additional custom tag attributes. AngularJS interprets those attributes as directives to bind input or output parts of the page to a model that is represented by standard JavaScript variables. The values of those JavaScript variables can be manually set within the code, or retrieved from static or dynamic JSON resources.

SocioCortex is the next generation of the collaborative information system, which was developed at the SEBIS chair for years. This system integrates proven features of SocioCortex's predecessor Tricia with approaches to end-user-oriented quantitative model analysis and the support for knowledge-intensive processes [13]. By exposing its features via a standardized API, the SocioCortex platform can serve as a foundation for the development of context-and project-specific applications. Our prototype application connects to SocioCortex backend. In order to access SocioCortex REST API we used sc-angular [14], which wraps the access to the REST API and furthermore provides some convenience functions.

Furthermore, the look and feel of our web application is based on the modified Read The Docs framework [15]. In the SocioCortex server we defined the content according to the seven guidelines from our proposed model. Our web application then dynamically pulls the documentation data from the SocioCortex server and visualizes the content.

For example, you can see the visualization of the 2nd guideline (Documentation of the API's high level design) in the Fig. 1 below. The figure presents a Class Diagram, which describes the structure of the system by showing the system's classes, their attributes, methods and the relationships among objects. UML Class Diagrams are a de facto standard in the design stages of a Software Development Process [16] and are therefore suitable solution for a high-level design of API documentation.

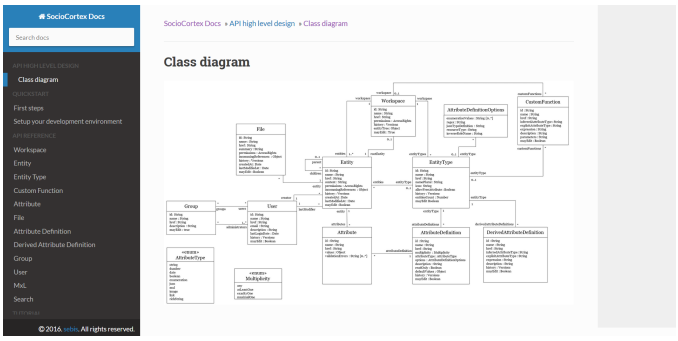

Fig. 1: Example of the 2nd guideline: Documentation of the API's high-level design

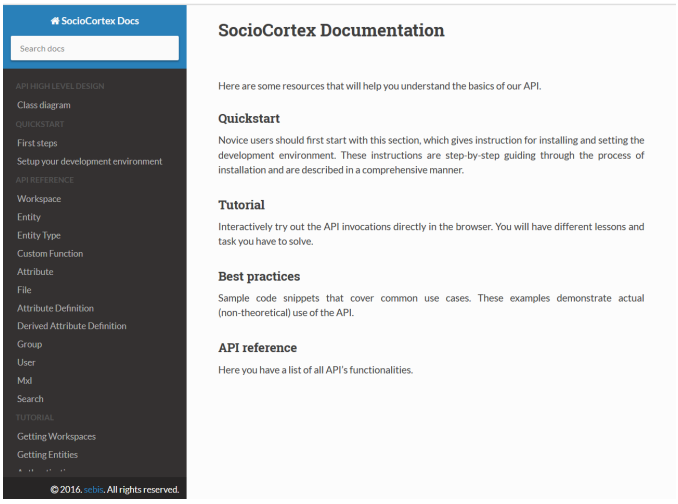

Fig. 2: Example of the 7th guideline: Multiple ways of navigation

The Fig 2 depicts the 7th guideline (Multiple ways of navigation). This guideline outlines the Overview page, Categories, Full text search and Link to related resources. As presented in the figure, the overview page is further structured into Quickstart, Tutorials, Best practices and API references. Moreover, the sidebar is divided by few categories and have full text search facility.

Furthermore, you can see the visualization of the 5th guideline (Tutorial) in the Fig. 3 below. As you can see on

the picture there is a task given to a user and an interactive code editor with console which offers the user possibility to try out the API invocations directly in the browser.

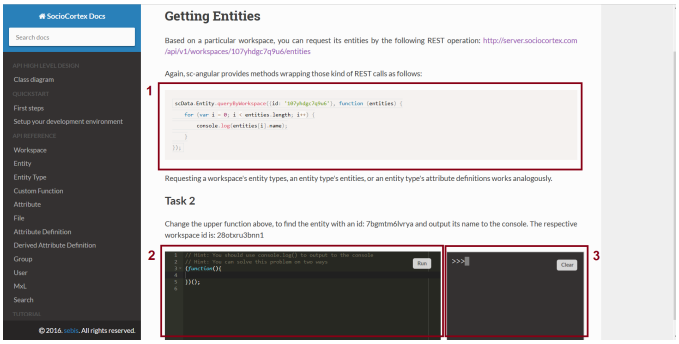

Fig. 3: Example of the 5th guideline: Tutorial

The demarcation 1 from the Fig. 3 shows a code snippet, presenting a typical usage of the SocioCortex REST API. The style of the syntax has been made using prism.js library. The demarcation 2 from the same figure depicts a code editor, where user has to enter the code according to a task that is given to him/her. When the user completes the assignment, he can see the results of his code by pressing the run button. Results are then presented in the console, shown in demarcation 3.

For example, the Fig. 4 shows a wrong result, which is marked in red.

Fig. 4: Wrong result of API call

On the other hand, the Fig. 5 below, presents how the output looks like in the console, when the user entered the correct code.

Fig. 5: Correct result of API call

For the code editor we have used Ace which is an embeddable code editor written in JavaScript [17]. It matches the features and performance of native editors such as Sublime, Vim and TextMate. Ace is maintained as the primary editor for Cloud9 IDE and is the successor of the Mozilla Skywriter project. Moreover, for the console implementation we have used jq-console that is a jQuery terminal plugin written in CoffeeScript, which tries to simulate a low level terminal by providing (almost) raw input/output streams as well as input and output states [18].

Furthermore, an example of the 6th guideline (API reference) is shown on the Fig. 6. On the picture we can see the integration of Swagger UI visualization into our prototype application. It was very important to incorporate Swagger into our proposal, since it has become de facto standard for describing and documenting REST APIs.

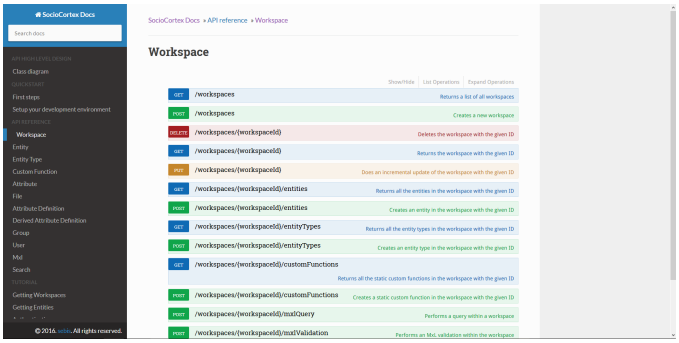

Fig. 6: Example of the 6th guideline: API reference

VI. Conclusion

In this paper, we proposed a conceptual model of API documentation, which will allow developers to easily comprehend and learn how to use the specific API endpoints. The model has defined the guidelines for creating an API documentation that will be easy to use and will provide developers with the tools for more efficiently integrating specific API into their system. Based on the identified shortcomings and our own experiences in using several real-world APIs, we proposed a novel conceptual model for API documentation that addresses the described concerns. We structured our proposed model into seven high-level guidelines: (1) up-to date documentation, (2) documentation of the API's high-level design, (3) quickstart, (4) tutorial, (5) best practices, (6) API reference and (7) multiple ways of navigation. We additionally developed a prototype implementation in order to assess our proposed model. As part of the prototype, we implemented a contemporary web application, which is based on AngularJS and integrates with well-established CMS platform SocioCortex.

Although we believe that applying our principles can improve the existing domain of API economy, the implementation of our model faces several technical challenges. The first challenge is related with the integration complexities between our model and a given solution. For instance, we need to fully understand the system (software) architecture on which we want to apply our model, and then find a way how to implement our proposal. The second challenge is how

to implement a mechanism that will make sure to keep the documentation up to date and manage all the versions of specific API. The third challenge is how to efficiently evaluate our proposed model, since not all users possess same programming skills and level of knowledge. For instance a beginner would require some basic examples on how to invoke a REST service, while an advanced user would just like to jump directly to advanced topics, e.g. the list of API functionalities. Therefore an advanced user might assess the model as too basic, while the beginner might not fully understand the documentation due to a high complexity.

As part of our future work we tend to extend our proposed model with advanced interactive tutorial leveraging virtual assistant, and additionally evaluate our proposal on a well-established open-source API Management solutions, such as JBoss apiman or WSO2 API Manager. Although we have derived our guidelines from the extensive literature review, the practicability and usefulness of our prototypical implementation and case studies will have to be presented. As part of future work we will also conduct a Quantitative Empirical Study, stating which documentations comply with our guidelines and which weaknesses current documentations have.

REFERENCES

[1] M. P. Robillard, "What makes apis hard to learn? answers from developers," *IEEE computer science*, 2009.

[2] J. J. S. J. H. S. R. Watson, M. Stamnes, "Api documentation and software community values: A survey of open-source api documentation," 2013.

[3] C. Scaffidi, "Why are apis difficult to learn and use?" *Crossroads*, 2006.

[4] K. W. David G. Novic, "What users say they want in documentation," 2006.

[5] G. E. Mitchell, "What do users really want from computer documentation?" *International Professional Communication Conference*, 1993.

[6] R. J. Brockmann, "The why, where and how of minimalism," *Proceeding SIGDOC '90 Proceedings of the 8th annual international conference on Systems documentation*.

[7] C. T. Chris Parnin, "A study of the documentation essential to software maintenance," *Proceedings of the 23rd annual international conference on Design of communication: designing for pervasive information*, 2005.

[8] C. Parnin and C. Treude, "Measuring api documentation on the web," *Web2SE*, 2011.

[9] R. H. Siddharth Subramanian, Laura Inozemtseva, "Live api documentation," *ICSE'14*, 2014.

[10] M. Maleshkova, "Investigating web apis on the world wide web," *Open Research Online*, 2010.

[11] A. F. T. Lethbridge, J. Singer, "How software engineers use documentation state of practice," *IEEE Computer Society*, 2003.

[12] M. P. R. Gias Uddin, "How api documentation fails," *IEEE Software*, 2015.

[13] "Sociocortex," https://wwwmatthes.in.tum.de/pages/13uzffgwlh8z4/SocioCortex.

[14] "sc-angulalr repository," https://github.com/sebischair/sc-angular.

[15] "Read the docs," http://read-the-docs.readthedocs.io/en/latest/.

[16] J. M. J. L. Karina Robles, Anabel Fraga, "Towards an ontology-based retrieval of uml class diagrams," *Information and Software Technology*, 2012.

[17] "Ace editor," https://ace.c9.io/.

[18] "jq-console github repository," https://github.com/replit/jq-console.

[19] R. Watson, "Development and application of a heuristic to assess trends in api documentation," *SIGDOC12*, 2012.