A Systematic Approach to Extract Knowledge Types in API Reference Documentation: A Survey

K.Srilakshmi
Department of Information Technology
Sree Vidyanikethan Engineering College
Tirupati, India
katari.srilakshmi@gmail.com

O.Obulesu
Department of Information Technology
Sree Vidyanikethan Engineering College
Tirupati, India
oobulesu681@gmail.com

Abstract - Developing computer programs with an Application Program Interface (API) is an important part in software Engineering. The API Reference Documentation is a key role in this process. The problem for developers is to select the optimized API reference documentation to implement programs quickly. By using suitable API reference documentation, removes the low value content in the web. By using knowledge types we improve the quality of the Reference documentation during the product development. The Reference documentation is also plays a crucial role in how developers learn and use an API, and developers have high expectations about the information they should find therein. Most technology platforms expose APIs provide a documentation system with a uniform structure and look and-feel for presenting and organizing the API documentation. The main idea of this paper is to extract knowledge types in API Reference Documentation and to study the gap between information seekers & information providers.

Keywords: API Documentation, Content Analysis, Java, Knowledge Types, .NET, Textual Mining.

I. INTRODUCTION

Software Engineering is the study of engineering to the design, development, and maintenance of software. Software engineering approach with the connotations of predictability, precision, mitigated risk and professionalism. The reference documentation is a type of document that outlines past procedures, actions or strategies as they relate to a particular activity. Application programming interfaces (APIs) access the reuse of libraries and frameworks in software development.

An important and challenging programming activity is using Application Programming Interfaces (APIs), frameworks, toolkits and libraries. Programmers implementing new functionality need to determine which APIs to use and how to combine them. Programmers reading or modifying code need to understand how existing code that calls APIs works, what assumptions the code makes, and how to change or add to the code without breaking these assumptions.

We define API reference documentation as a set of documents indexed by API element name, where each document specifically provides information about an element (class, method, etc.). Reference documentation is a necessary and significant part of a framework. Most technology platforms exposing APIs provide a documentation system with a uniform structure and look and-feel for presenting and organizing the API documentation. Phase 1 addressed the first research question using a combination of grounded and analytical methods to derive taxonomy of knowledge types. In Phase 2, we used the taxonomy as a coding guide and had 17 trained coders review a random sample of documentation units to assess whether each unit contained knowledge of the different types in our taxonomy. Each knowledge type became a variable that had to be rated with the value True (if it is present in the unit) or False (if not). Application Programming Interfaces (APIs) are a means of code reuse. They provide an interface to features and functionality in existing frameworks and libraries, such as the Java Standard Edition libraries or the .NET framework. Reusing APIs saves time and mitigates the risk of defects in implementing an equivalent feature from scratch. Using large APIs, however, is often challenging to many programmers. This challenge can be attributed to factors like interdependencies between multiple APIs, obscure API naming convention, low cohesion of an API, or lack of information on how to use them efficiently. Providing extensive documentation for the APIs can help programmers understand the APIs better. Documentation is thus an important constituent of APIs in particular and software projects in general. There exist different types of software documentation, such as reference documentation, code comments, tutorials, and white papers. Each of these types of documentation serves a specific purpose. For instance, API reference documentation, such as Javadoc, provides information specific to individual API elements, whereas a tutorial, such as the Java Tutorial, provides information used to accomplish an end-to-end task.

We use the expression rating a unit to mean rating all variables for the unit. In this phase each documentation unit was independently rated by two randomly assigned coders. The result was a database of ratings which also contained disagreements for some variables in some documentation units (e.g., for the documentation unit of method m coder 1 rated the presence of knowledge type T as True and coder 2 coded it as False). In Phase 3, we systematically analyzed the disagreements to 1) evaluate the work of the coders, 2) evaluate the quality of the guide, and 3) design a scheme to resolve disagreements. This analysis allowed us to answer our second research question. After applying the data cleaning scheme, each rated variable in a unit was reconciled into a single value: True or False.

The main **objectives** of this paper are:

i A systematic identification of knowledge types in reference documentation.
ii To study the gap between information seekers and information providers to provide qualitative documentation units that present unusual combinations of features in the web.
iii To improve the quality of software project development via good reference documentation.
iv To improve the quality of research works in the industry.

II. RELATED WORK

In 1977, J.R. Landis proposed a measurement of agreement for categorical data[6], it explains the measurement of agreement are used to assess the reproducibility of a new assay or instrument, the acceptability of a new or generic process, methodology or method comparison. Also identified the several different alternatives to Cohen's kappa and weighted kappa coefficients. It focused mainly on the investigate whether there are others forms of matrix functions that can be applied to the multivariate kappa.

In 1993, James D. Herbssleb and Eiji kuwana introduced preserving knowledge in design projects: what designers need to know, it describes the design of technology support and new procedural methods for software design. And also identified data analysis method and research for design tools and methods are discussed [4].

In 1999, Douglas Kramer proposed API documentation from source code comments: a case study of Javadoc to explain the process we went through to determine the goals, principles, audience, content and style for writing comments in source code for the Java platform at the Java Software division of Sun Microsystems [5]. This includes how the documentation comments evolved to become the home of the Java platform API specification, and the guidelines we developed to make it practical for this document to reside in the same files as the source code.

In 2007, Strauss, Anselm L.; Corbin, Juliet M. proposed Basics of Qualitative Research: Techniques and Procedures for Developing Grounded Theory [1] explains the different techniques and procedures. And also explained theoretical sampling is data gathering driven by concepts derived from the evolving theory and based on the concept of making comparisons.

Again in 2007, Brian Ellis, Jeffrey Stylos, and Brad Myers introduced The Factory Pattern in API Design: A Usability Evaluation usability of software APIs is an important and infrequently researched topic. This explained the future research should explore the similarities and differences between class clusters and factories from the API developer's point of view as well [3]. Also explains the relative ease of debugging objects created using factories as opposed to constructors.

In 2008, Jonathan Sillito proposed Asking and Answering Questions during a Programming Change Task, is known about the specific kinds of questions programmers ask when evolving a code base. This also explains the catalog of 44 types of questions programmers ask and a categorization of those questions into four categories based on the kind and scope of information needed to answer a question, and a description of important context for the process of answering questions, and a description of support that is missing from current programming tools[12].

Again in 2008, Jeffrey Stylos, Brad A. Myers proposed The Implications of Method Placement on API Learnability [13] explained to better understand what makes Application Programming Interfaces (APIs) hard to use and how to improve them. Here Javadoc could also be changed to make appropriate starting classes easier to find. And also used different prototype alternative designs to the flat alphabetical class list.

In 2009, Martin P. Robillard introduced What Makes APIs Hard to Learn? Answers from Developers [9] explains the study of obstacles that professional Microsoft developers faced when learning to use APIs uncovered challenges and resulting implications for API users and designers.

In 2010, Barthelemy Dagenais and Martin P. Robillard proposed Creating and Evolving Developer Documentation: Understanding the Decisions of Open Source Contributors [2] identified the decisions that contributors make, the factors influencing these decisions and the consequences for the project. Also we would like to report our results on the other decisions made by open source contributors and pursue our analysis of the documentation needs of users.

In 2011, Chris Parnin, Christoph Treude proposed Measuring API Documentation on the Web, developer forums and Q&A websites are changing the way software is documented [10]. Introduced the method of one particular API |jQuery | are documented on the Web. And also we need to understand what can be done to help developers find documentation more effectively and how tool support can help those creating documentation using social media.

Again in 2011, Lin Shi, Hao Zhong, Tao Xie, and Mingshu Li proposed An Empirical Study on Evolution of API Documentation with the evolution of an API library, its documentation also evolves [11].

In 2012, Martin Monperrus, Michael Eichberg, Elif Tekes, Mira Mezini proposed what should developers be aware of? An empirical study on the directives of API documentation, are exposed to developers in order to reuse software libraries. And also identified work related software tools [7].

Again in 2012, Dennis Pagano and Walid Maalej proposed How do open source communities' blog? [8] They report on an exploratory study, which aims at understanding how software communities use blogs compared to conventional development infrastructures. They introduced two research methods hypothesis-driven and a content analysis line. Hypothesis-driven research enables us to explore the role of social media and allows for a need-driven integration of these media into development processes and tools. A content analysis research enables a more in-depth analysis of the knowledge shared in blogs, giving more reliable results on the roles, efficiency, and the quality of blogs and blogging.

III. CONTENT ANALYSIS METHODOLOGY

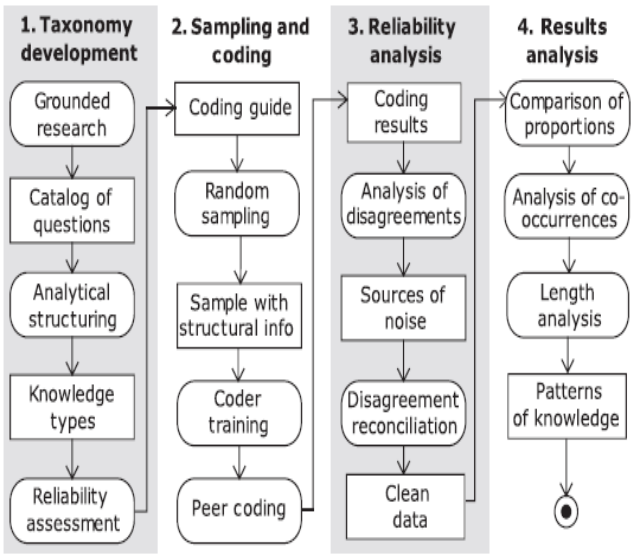

Fig. 1. The overview of process methodology

The objective is to produce a taxonomy that would be:

1. Reliable, in that different people consistently come to the same conclusion about the knowledge types contained in a documentation unit.
2. Meaningful, listing knowledge types relevant to the practice of software development.
3. Fined-grained, providing more than just a few high-level categories.

The outcome of this process was a detailed taxonomy of knowledge types usable as a coding guide for the quantitative analysis of the content of API documentation. Our research is based on content analysis, a methodology for studying the content of recorded human communications. The most challenging part of this research project consisted of describing the different knowledge types commonly found in API reference documentation. Many authors have discussed the different types of knowledge used in various software engineering contexts, and in some cases provided empirically grounded taxonomies. Unfortunately, a careful review of previous work showed that existing taxonomies are neither directly applicable to API documentation nor sufficiently detailed to be directly used as knowledge types definitions for our purpose. We thus elaborated a taxonomy of knowledge types for API reference documentation through an iterative refinement process.

IV. PROPOSED METHODOLOGY

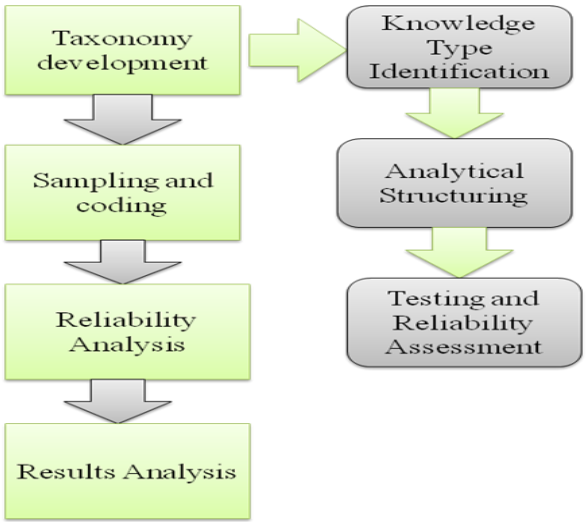

Fig. 2. A Systematic research process

A. *Knowledge Type Identification*

In the first step, we used a grounded approach to elicit a preliminary list of knowledge types present in API reference documentation. Each author independently selected sentences from the reference documentation of two different open-source systems: Http Components and Jena. These systems were selected for their mature and extensive reference documentation. To select sentences, we employed a process inspired by theoretical sampling. This involves refining and adjusting the sampling procedure as data is collected.

B. *Analytical Structuring*

A limitation of the grounded approach is that it does not guarantee that all knowledge types will be uncovered. In a second step, we refined our catalog with a detailed review of the literature and, through analytical reasoning, expanded all variation points for a question template.

As a part of this process, we assessed the reliability of our catalog by independently coding individual sentences in randomly selected sets of API elements in open-source APIs others that those distributed as part of the JDK 6.0 and .NET 4.0.5 The goal of this evaluation was to determine if any obvious questions had been left out, and assess the ease of associating sentences with questions that model knowledge types

C. *Testing and Reliability Assessment*

In the last step, we tested the reliability of our taxonomy by iteratively coding various random samples of 50 units, studying the disagreements, and making improvements based on the findings. As a part of this process, we added an increasing number of clarifications to the coding guide about how to code different variables. The samples consisted of documentation units which were to be coded for about 12 variables that represent to what degree knowledge of different types was present.

V. CONCLUSION

We found that Functionality knowledge is pervasive and Structure is common, while other types, such as Concepts and Purpose, are much rarer. We also found that Non information, a deviant type of knowledge representing low-value content, is prevalent in the documentation of methods and fields of the JDK and .NET APIs. Comparisons of patterns of knowledge types in different populations revealed many significant differences on, for example, how classes are documented versus methods, how knowledge types tend to co-occur, and how these patterns take different forms in different technology platforms. Collections of knowledge patterns applicable to a cohesive subset of API documentation unit can be seen as a form of documentation style.

The findings can inform software development practice in four different ways. First, they allow practitioners to evaluate the content of their API documentation in relation to well-defined knowledge types. Second, they can guide the development of documentation templates that are adapted to the knowledge commonly associated with different API elements types. Third, our taxonomy provides a vocabulary that can facilitate discussions about the content of API documentation. Finally, they document the extent of low-value content in documentation which we hope will serve as a motivation for curtailing this practice.

The study also motivates additional research in at least three areas. First, our taxonomy provides a foundation for the automated classification of knowledge types in API documentation. Second, our results help in studying the gap between the knowledge provided by different types of documents and the information needs of developers. Finally, classifying documentation according to knowledge types supports quantitative analyses linking patterns of knowledge with more subjective quality features.

REFERENCES

1. J. Corbin and A. Strauss, Basics of Qualitative Research: Techniques and Procedures for Developing Grounded Theory, third ed. Sage Publications, 2007.
2. B. Dagenais and M.P. Robillard, “Creating and Evolving Developer Documentation: Understanding the Decisions of Open Source Contributors,” Proc. 18th ACM SIGSOFT Int’l Symp.Foundations of Software Eng., pp. 127-136, Nov. 2010.
3. B. Ellis, J. Stylos, and B. Myers, “The Factory Pattern in API Design: A Usability Evaluation,” Proc. 29th ACM/IEEE Int’l Conf.Software Eng., pp. 302-312, May 2007.
4. J.D. Herbsleb and E. Kuwana, “Preserving Knowledge in Design Projects: What Designers Need to Know,” Proc. Joint INTERACT ’93 and CHI ’93 Conf. Human Factors in Computing Systems, pp. 7- 14, 1993.
5. D. Kramer, “API Documentation from Source Code Comments: A Case Study of Javadoc,” Proc. Conf. ACM Special Interest Group for Design of Comm., pp. 147-153, 1999.
6. J.R. Landis and G.G. Koch, “The Measurement of Observer Agreement for Categorical Data,” Biometrics, vol. 33, no. 1, pp. 159-174, Mar. 1977.
7. M. Monperrus, M. Eichberg, E. Tekes, and M. Mezini, “What Should Developers Be Aware of? An Empirical Study on the Directives of API Documentation,” Empirical Software Eng., vol. 17, no. 6, pp.703-737, 2012.
8. D. Pagano and W. Maalej, “How Do Open Source Communities Blog?” Empirical Software Eng., pp. 1- 35, 2012.
9. M.P. Robillard, “What Makes APIs Hard to Learn? Answers from Developers,” IEEE Software, vol. 26, no. 6, pp. 26-34, Nov./Dec.2009.
10. M.P. Robillard and R. DeLine, “A Field Study of API Learning Obstacles,” Empirical Software Eng., vol. 16, no. 6, pp. 703-732, 2011.
11. L. Shi, H. Zhong, T. Xie, and M. Li, “An Empirical Study on Evolution of Documentation,” Proc. Conf. Fundamental Approaches to Software Eng., pp. 416-431, 2011.
12. J. Sillito, G.C. Murphy, and K.D. Volder, “Asking and Answering Questions during a Programming Change Task,” IEEE Trans.Software Eng., vol. 34, no. 4, pp. 434-451, July/Aug. 2008.
13. J. Stylos, B. Graf, D.K. Busse, C. Ziegler, and R.E.J. Karstens, “A Case Study of API Re design for
14. Improved Usability,” Proc. Symp. Visual Languages and Human-Centric Computing, pp.189-192, 2008.