Automated Example Oriented REST API Documentation at Cisco

S M Sohan
Security Group
Cisco Systems Ltd.
Calgary, Canada
sosohan@cisco.com

Craig Anslow
Department of Computer Science
University of Calgary
Calgary, Canada
canslow@ucalgary.ca

Frank Maurer
Department of Computer Science
University of Calgary
Calgary, Canada
frank.maurer@ucalgary.ca

***Abstract*—API documentation presents both a problem and an opportunity for API usability. Representational State Transfer, or more commonly known as REST based APIs are used by software developers to interconnect applications over HTTP. Developers are required to publish and maintain the documentation of their REST APIs so that other developers can learn and use the APIs as intended. This poses the problem of identifying an efficient and effective process of generating and maintaining the documentation of REST APIs. In this paper, we have discussed our lessons learned from a case study comprising of the production use of an automated example oriented REST API documentation approach using a tool called SpyREST at Cisco over a period of eighteen months. We have observed that continuously updated documentation can be achieved by using automated test code against a REST API. Practitioners can leverage the insights shared in this paper to improve the state of their REST API documentation process. Researchers and tool developers can incorporate the ideas from this case study to extend the example oriented documentation approach to APIs beyond the realm of REST APIs.**

***Keywords*-API; REST; Documentation; Tool; Case study; Test; Automation; HTTP; Web API;**

I. Introduction

Application Programming Interfaces, commonly known as APIs, are used to express a software component in terms of its operations, inputs, outputs, and their types[1]. Robillard describes API as follows: An API is the interface to implement functionality that developers can access to perform various tasks [**?**] [**?**].

REST APIs are a subclass of APIs that use standard web technologies for interconnectivity over HTTP. Fielding defined Representational State Transfer or REST as an architectural style for developing distributed hypermedia systems [**?**]. REST APIs provide an abstraction of the underlying system by using system specific resources such as documents, images, etc. and unique identifiers to access the resources. Using REST APIs, systems perform actions on the resources by transferring a representation of the resources between various systems. For example, the GitHub REST API[2] has a resource called $Repository$ to denote a code repository hosted on GitHub.

[1]https://en.wikipedia.org/wiki/Application_programming_interface
[2]https://developer.github.com/v3/repos/#create

Researchers identified the documentation of APIs as both the primary source of information as well as the key obstacle for API usability [**?**]. To this regard, researchers have identified the qualities of "good API documentation" as follows: complete, correct, includes thorough explanations and code examples, provides consistent presentation and organization [**?**], [**?**]. In our previous work, we introduced a novel technique and SpyREST, an implementation, based on an HTTP proxy server to automatically intercept example REST API calls and synthesize the data to produce REST API documentation to meet the aforementioned qualities.

The case study in this paper presents an evaluation of SpyREST in the industry. SpyREST is being used in production at Cisco for the documentation of a commercial REST API of a cloud based Cyber security product that the first author of this paper is affiliated with. It provides us with a unique opportunity to analyze the impact of the industry adoption of a tool developed in research. Production usage over an eighteen month period also allows us to understand the problem and opportunities presented by SpyREST in depth. Despite the popularity and adoption of REST APIs, there is a lack of published work about the API documentation techniques used by today's internet companies. Our core contributions from this case study are as follows:

- **Test driven REST API documentation**. For practitioners, we discuss a reusable technique for producing example oriented REST API documentation as a byproduct from automated API test code.
- **Evolution of API documentation**. For practitioners, we discuss a viable technique for maintaining the evolution of API documentation as the API evolves without duplicating effort.
- **Implications for future work**. We show a practical evidence that API documentation can be generated by intercepting and transforming example API calls. Researchers can leverage this technique to improve tool support for the documentation of other forms of APIs beyond REST.

The remainder of this paper is organized as follows: in the following section we present a literature review to discuss the current state of research on REST API documentation. Then,

we provide a brief overview of our REST API documentation technique and the tool, SpyREST, followed by a case study of using SpyREST at Cisco. Then, we discuss our lessons learned and the limitations of this case study.

II. Related Work

A. API Usability and Documentation

API usability is a qualitative concept derived from the characteristics that make an API easy to use. Several papers in the existing literature have focused on identifying the characteristics that make an API usable based on case studies. Robillard studied API usability by surveying 83 software developers at Microsoft [**?**]. Robillard found that 78% of the survey participants read API documentation to learn the APIs, 55% used code examples, 34% experimented with the APIs, 30% read articles, and 29% asked colleagues. While API documentation is used as the principal source of information about how to use an API, Robillard et al. found that the most severe API learning obstacles are related to the API documentation. For API usability, Robillard suggested the following requirements as must-have for API documentation: include good examples, be complete, support many example usage scenarios, be conveniently organized, and include relevant design elements. Zibran et al. analyzed bug repositories for 562 API usability related bugs from five different projects and found that 27.3% of the reported bugs are API documentation bugs [**?**]. Scheller et al. provided a framework for objectively measuring API usability based on the number and types of different objects and methods that the API provides [**?**].

Kuhn performed a user study with 19 professional software developers to understand requirements for tool development to support API learnability [**?**]. Kuhn recommended the following as requirements for API documentation: trustworthiness, confidentiality, lack of information overload and the need for code examples as first-class documentation artifacts. Shi et al. observed a large number of API documentation changes related to polishing the custom content, (i.e. fix typos, and API usage examples) [**?**]. They recommended API documentation tools to support editors for custom content to provide usage tips and simple ways to include API usage examples without syntax errors. Ko et al. found that thorough introductions to the concepts, standards and ideas in API documentation are a prerequisite for developers to be able to effectively use an API [**?**].

The strong relationship between the documentation and usability of the API as discussed in the aforementioned papers also applies to the context of REST APIs.

B. Usage Examples in API Documentation

Several authors introduced tool support for including usage examples with API documentation. Hoffman et al. recommended using executable examples in API documentation [**?**]. They introduced Roast test as tool support to combine prosaic descriptions of Java APIs along with executable code examples in a unified API documentation. Montandon et al. developed APIMiner as a search tool for Java and Android APIs and recommended providing production-like API usage examples in the API documentation [**?**]. They observed that 35% of API related web searches performed on APIMiner included the term example inferring that developers search for source code examples while using API documentation. Zhu et al. developed an Eclipse plugin called UsETeC to extract API usage examples by automatically synthesizing JUnit test code of the APIs [**?**]. Stylos presented Jadeite as an IDE plug-in that combines a few techniques to provide developers with faster access to relevant API documentation [**?**]. Jadeite uses placeholders for API elements such as classes and method names that developers commonly expect to exist, but the actual classes or methods are named differently. When developers search for placeholder API elements, Jadeite shows links to the API documentation of the actual API elements and finds relevant usage examples from Google code search.

Several authors presented techniques for linking official API documentation with crowd-sourced API usage examples that is otherwise fragmented. Nasehi et al. recommended mining knowledge repositories such as StackOverflow and developer forums should be considered for retrieving useful code examples [**?**]. Parnin et al. found that examples of 87.9% of all jQuery API methods are found by searching software development blogs and forums [**?**]. Wu et al. presented an Eclipse plugin called CoDocent that can automatically find code examples using various online code search engines and link with the relevant official API documentation [**?**]. Chen et al. presented a technique to automatically link official documentation with crowd-sourced documentation by recording the API related web searches that are performed by developers [**?**]. Dagenais et al. presented a technique and a tool called RecoDoc to link code-like elements from API mailing lists and developer forums with their corresponding code elements in the API documentation [**?**]. Treude et al. presented a machine learned based technique called SISE to augment useful information from StackOverflow to API documentation by using text similarity of API elements and StackOverflow content [**?**].

As mentioned above, we observed that the research on API documentation related tools have focused on local APIs such as Java library APIs. We found a lack of published work on the tool support for including usage examples with REST API documentation.s

C. REST API Documentation

Maleshkova analyzed the state of RESTful APIs and found that most RESTful APIs are manually documented which results in API underspecification, and a lack of support for common tasks and reusable tools [**?**]. Myers et al. performed a user study on the usability of a complex API for enterprise SOA [**?**]. They recommended providing a consistent look-and-feel with explanation for the starting points and an overall map comprising of both text and diagrams, providing a browsing experience with breadcrumb trail following a hierarchy, an effective search interface, providing example code and a way to exercise the examples online without writing code. In a

case study, we found the documentation of RESTful APIs are performed manually or using bespoke tools [**?**]. We observed the documentation of the studied RESTful APIs to commonly include summary information and API examples, with optional description of the structure of API requests and responses.

Several authors have suggested machine readable specification languages for REST APIs that can be used to transform into API documentation and auto generated API client code. Hadley proposed WADL (Web Application Description Language) [**?**]. Mangler et al. proposed RIDDL as an extension of WADL to support composition and evolution of REST APIs [**?**]. Kopecky et al. proposed hRESTS, as an HTML based mico-format to describe RESTful APIs [**?**]. Verborgh et al. proposed RESTdesc as a language to define RESTful APIs using resources and links that connects the resources [**?**]. Danielsen presented a vocabulary to describe RESTful APIs using WIfL(Web Interface Language) [**?**]. Lei et al. presented OmniVoke as a tool to abstract out multiple RESTful APIs under a unified interface following a specification [**?**]. Polak proposed a specification format for REST API using the Model-Driven Architectural principle where the evolution of a REST API can be programmatically managed from it's model representation [**?**].

In addition to the existing literature, several REST API description languages have been proposed by industry practitioners such as RAML[3], Blueprint[4], and Swagger[5]. There are tools that can automatically convert and publish the description of RESTful APIs from these languages into HTML based RESTful API documentation.

The primary advantage of these specification languages is code generation and automatic transformation into REST API documentation. On the other hand, we found a lack of automated tool support for producing the API specifications for REST APIs.

TABLE I
REST API DOCUMENTATION

Desirable Property	**Current State**
Detailed introduction	Manually edited contents are commonly used.
Includes Examples	Commonly include manually generated API examples.
Executable Examples	Bespoke tooling is used to provide API explorers.
Automated	Tools rely on manually written specifications.
Consistent Presentation	Includes access information, resources, actions, request and response structures and API examples.

Table I contrasts the current state of tool support for REST API documentation against a set of properties that researchers identified as required for API usability. In summary, practitioners and researchers have attempted to solve the problem of RESTful API documentation by proposing candidate specifications to standardize the vocabulary and format of describing RESTful APIs. Manual work is needed by REST API developers to generate and maintain their API specifications as it evolves. These specification formats support describing the structure of different API elements (the syntax), but no support is provided for API usage examples and executable API examples (the semantics), an important attribute for API learnability as identified by the researchers. Also, we found a lack of published papers on the effectiveness of the aforementioned specification languages in an industry setting. In our work with SpyREST, instead of relying on specification, we have focused on automatically intercepting and transforming example REST API calls into usable documentation. In this paper we have shared the lessons learned of using this new technique at Cisco.

[3]http://raml.org/
[4]https://apiblueprint.org/
[5]http://swagger.io/

III. CONCLUSION

The conclusion goes here. this is more of the conclusion

ACKNOWLEDGMENT

The authors would like to thank... more thanks here