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 [1] [2].

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 [3]. 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 [2]. 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 [2], [4]. 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 [1]. 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 [5]. 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 [6].

Kuhn performed a user study with 19 professional software developers to understand requirements for tool development to support API learnability [7]. 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) [8]. 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 [9].

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 [10]. 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 [11]. They observed that 35% of API related web searches performed on APIMiner included the term âĂIJexampleâĂİ 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 [12]. Stylos presented Jadeite as an IDE plug-in that combines a few techniques to provide developers with faster access to relevant API documentation [13]. 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 [14]. Parnin et al. found that examples of 87.9% of all jQuery API methods are found by searching software development blogs and forums [15]. 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 [16]. 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 [17]. 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 [18]. 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 [19].

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 [20]. Myers et al. performed a user study on the usability of a complex API for enterprise SOA [4]. 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 [21]. 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) [22]. Mangler et al. proposed RIDDL as an extension of WADL to support composition and evolution of REST APIs [23]. Kopecky et al. proposed hRESTS, as an HTML based mico-format to describe RESTful APIs [24]. Verborgh et al. proposed RESTdesc as a language to define RESTful APIs using resources and links that connects the resources [25]. Danielsen presented a vocabulary to describe RESTful APIs using WIfL(Web Interface Language) [26]. Lei et al. presented OmniVoke as a tool to abstract out multiple RESTful APIs under a unified interface following a specification [27]. 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 [28].

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. OVERVIEW OF SPYREST

We provide a brief overview of SpyREST, the REST API documentation tool and the underlying technique used in this case study. In our previous papers, we have discussed the design and implementation of this tool in greater detail [29], [30].

At the heart of the technique is a pass-through HTTP proxy server which acts as an interceptor between an API client and the API. This allows the proxy server to inspect the raw HTTP request and response data from the example API calls. However, for usable API documentation the raw HTTP data needs to be furthered processed and enriched with meta data. For example, given the following HTTP request and response data from an example API call to create a blog post:

```
Request Verb: POST
Request URL: /v2/posts
Request Headers:
  Content-Type: application/json
  Authorization: Basic dXNlcjpwYXNzd29yZA==
Request Body:
  {
    "title":"My New Blog post",
    "content": "This is a new blog post"
  }

Response Headers:
  Location: "/v2/posts/1"
  host: "blog.example.com"
```

A series of transformation needs to take place to produce a usable API documentation. The proxy server used by SpyREST is customized to record and synthesize such example API calls. The transformation process involves the following analyzers:

- **API version analyzer**. The version analyzer inspects the URL and a request header named *Accept* : to automatically infer the API version used by the example API call. From the aforementioned example, the version analyzer auto-detects the API version as v2 based on the URL. It uses regular expressions to match the URL and *Accept* header against the commonly used API versioning formats.

- **API resource analyzer**. To generate a hierarchical representation of the API elements, it's important to group multiple API actions that correspond to a single API resource under a single hierarchy. API resource analyzer parses the API URL and detects the rightmost non-numeric part of the URL path as the API resource. For the given example, SpyREST detects $post$ as the API resource.
- **API action analyzer**. Each API resource can be accessed or modified via the API actions. The API action analyzer infers the API action from example API calls by combining the HTTP verb (e.g. $GET, POST$) with an auto-templated representation of the path. For the given example, the action analyzer identifies $POST/v2/posts$ as the API action. If the path contains numeric or uuid parameters, the a templated path is used. For example, the API action analyzer detects a GET request to the path $/v2/posts/1/comments/10$ as the API action $GET/v2/posts/ : post_id/comments/ : comment_id$. The templated API actions offer a more meaningful representation of the intent, and allows multiple API examples to be grouped under a single API action.
- **API query parameter analyzer:**. The query parameter analyzer records each query parameter that is used by the example API calls and automatically infers the data types such as integer, string, timestamp, etc. For each API action, a query parameter table is shown where each row displays the name, auto-detected data type, and example values for each query parameter.
- **API request header analyzer**: The request header analyzer strips the $Authorization$ header to remove confidential credentials from the API documentation. Additionally, the request header analyzer automatically detects the type of authorization used, e.g. Basic, Token, etc. For the given example, the API requests header analyzer transforms the $Authorization$ header's value as $Authorization : BasicFILTERED$.
- **API body analyzer**: The body analyzer captures the request and response bodies and infers the structure of the body as an object with auto-detected field names and data types. The request body analyzers produces the following information from the given example for API documentation:

Field	Data Type	Example
title	String	My New Blog post
content	String	This is a new blog post

- **API response header analyzer**: The response header analyzer maintains a list of commonly found response headers (e.g. host, pragma, server, etc.) that add little value to API documentation and automatically removes them from the documentation.
- **Custom content analyzer**: The version, resource, and action analyzers detect commonly seen patterns and extracts the relevant information. The result is also used to automatically infer a human readable description for each API action. For the given example, the auto detected description is as follows: $CreateaPost$ by combining the HTTP verb ($POST$) with the API resource ($post$). When unfamiliar formats are used by an API or customization is required, the custom content analyzer allows API developers to override each of the auto-detected attributes such as the API version, resource, action and description by using a set of SpyREST specific request headers (e.g. $x - spy - rest - version, x - spy - rest - resource$, etc.).

As shown before, SpyREST uses the aforementioned analyzers to transform the raw HTTP request and response information from example REST API calls into a structured representation of the API as follows: A REST API has one or more versions, each version has one or more API resources, each resource has one of more actions, and each action is defined by its URL, query parameters, request and response headers and bodies, and one or more usage examples.

SpyREST has a web based UI to render a hierarchical representation of the aforementioned structured data as the API documentation. The web UI also features a wiki-like editor on each page so that human written detailed descriptions can be added with rich content to explain business concepts. In addition to displaying the structured API documentation, the web UI also features an executable code using cURL [6], a commonly used tool for accessing resources over HTTP, for each captured API example so that the API client developers can try the API examples without having to write custom code.

To generate usable API documentation, REST API developers need to run appropriate example API calls via SpyREST proxy server. Once a set of example API calls are defined, the documentation can be auto-updated by replaying the example API calls. The replay only updates the auto-generated part of the documentation leaving the human written descriptions unchanged.

To summarize, SpyREST provides tool support to improve the process of REST API documentation with usage examples, automatic updates, executable examples and a consistent hierarchical representation of the API.

IV. Conclusion

The conclusion goes here. this is more of the conclusion

Acknowledgment

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

[6]https://curl.haxx.se/

REFERENCES

[1] M. Robillard and R. DeLine, "A field study of API learning obstacles," *Empirical Software Engineering*, vol. 16, no. 6, pp. 703–732, 2011.

[2] M. Robillard, "What makes APIs hard to learn? the answers of developers," *Software, IEEE*, vol. PP, no. 99, pp. 1–1, 2011.

[3] R. T. Fielding, "Architectural styles and the design of network-based software architectures," Ph.D. dissertation, University of California, Irvine, 2000.

[4] B. A. Myers, S. Y. Jeong, Y. Xie, J. Beaton, J. Stylos, R. Ehret, J. Karstens, A. Efeoglu, and D. K. Busse, "Studying the documentation of an api for enterprise service-oriented architecture," *J. Organ. End User Comput.*, vol. 22, no. 1, pp. 23–51, 2010.

[5] M. F. Zibran, F. Z. Eishita, and C. K. Roy, "Useful, but usable? factors affecting the usability of APIs," in *Proc. of 2011 Working Conference on Reverse Engineering (WCRE)*. IEEE, 2011, pp. 151–155.

[6] T. Scheller and E. Kühn, "Automated measurement of api usability: The API concepts framework," *Information and Software Technology*, vol. 61, pp. 145–162, 2015.

[7] A. Kuhn and R. DeLine, "On designing better tools for learning APIs," in *Search-Driven Development - Users, Infrastructure, Tools and Evaluation (SUITE), 2012 ICSE Workshop on*, 2012, pp. 27–30.

[8] L. Shi, H. Zhong, T. Xie, and M. Li, "An empirical study on evolution of API documentation." in *Proc. of Conference on Fundamental Approaches to Software Engineering (FASE)*, vol. 6603. Springer, 2011, pp. 416–431.

[9] A. J. Ko and Y. Riche, "The role of conceptual knowledge in api usability," in *Proc. of 2011 Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. IEEE, 2011, pp. 173–176.

[10] D. Hoffman and P. Strooper, "API documentation with executable examples," *Journal of Systems and Software*, vol. 66, no. 2, pp. 143 – 156, 2003.

[11] J. E. Montandon, H. Borges, D. Felix, and M. T. Valente, "Documenting APIs with examples: Lessons learned with the APIMiner platform," in *Proc. of 2013 Working Conference on Reverse Engineering (WCRE)*. IEEE, 2013, pp. 401–408.

[12] Z. Zhu, Y. Zou, B. Xie, Y. Jin, Z. Lin, and L. Zhang, "Mining api usage examples from test code," in *Proc. of 2014 IEEE International Conference on Software Maintenance and Evolution (ICSME),*. IEEE, 2014, pp. 301–310.

[13] J. Stylos, B. A. Myers, and Z. Yang, "Jadeite: Improving API documentation using usage information," in *'09 Extended Abstracts on Human Factors in Computing Systems*, ser. (CHI EA). ACM, 2009, pp. 4429–4434.

[14] S. M. Nasehi, J. Sillito, F. Maurer, and C. Burns, "What makes a good code example?: A study of programming Q&A in StackOverflow," in *Proc of. IEEE International Conference on Software Maintenance*, 2012, pp. 25–34.

[15] C. Parnin and C. Treude, "Measuring API documentation on the web," in *Proceedings of the International Workshop on Web 2.0 for Software Engineering*, ser. (Web2SE). ACM, 2011, pp. 25–30.

[16] Y.-C. Wu, L. W. Mar, and H. C. Jiau, "Codocent: Support api usage with code example and api documentation," in *Proc. of 2010 International Conference on Software Engineering Advances (ICSEA)*. IEEE, 2010, pp. 135–140.

[17] C. Chen and K. Zhang, "Who asked what: Integrating crowdsourced faqs into api documentation," in *Companion Proceedings of the International Conference on Software Engineering*, ser. ICSE Companion 2014. ACM, 2014, pp. 456–459.

[18] B. Dagenais and M. P. Robillard, "Recovering traceability links between an api and its learning resources," in *Proc. of 2012 International Conference on Software Engineering (ICSE)*. IEEE, 2012, pp. 47–57.

[19] C. Treude and M. P. Robillard, "Augmenting api documentation with insights from stack overflow," in *Proceedings of the 38th International Conference on Software Engineering*, ser. ICSE '16. New York, NY, USA: ACM, 2016, pp. 392–403. [Online]. Available: http://doi.acm.org/10.1145/2884781.2884800

[20] M. Maleshkova, C. Pedrinaci, and J. Domingue, "Investigating web APIs on the world wide web," in *Proc. of European Conference on Web Services (ECOWS)*. IEEE, 2010, pp. 107–114.

[21] S. Sohan, C. Anslow, and F. Maurer, "A case study of web api evolution," in *Proc. of 2015 IEEE World Congress on Services (SERVICES)*. IEEE, 2015, pp. 245–252.

[22] M. J. Hadley, "Web application description language (wadl)," 2006.

[23] J. Mangler, P. P. Beran, and E. Schikuta, "On the origin of services using RIDDL for description, evolution and composition of RESTful services," in *Proc. of International Conference on Cluster, Cloud and Grid Computing (CCGrid)*. IEEE/ACM, 2010, pp. 505–508.

[24] J. Kopecky, K. Gomadam, and T. Vitvar, "hRESTS: An HTML microformat for describing RESTful web services," in *Proc. of International Conference on Web Intelligence and Intelligent Agent Technology (WI-IAT)*, vol. 1. IEEE, 2008, pp. 619–625.

[25] R. Verborgh, T. Steiner, D. Van Deursen, J. De Roo, R. d. Walle, and J. GabarrÃş VallÃľs, "Capturing the functionality of web services with functional descriptions," *Multimedia Tools and Applications*, vol. 64, no. 2, pp. 365–387, 2013.

[26] P. Danielsen and A. Jeffrey, "Validation and interactivity of web API documentation," in *International Conference on Web Services*. IEEE, 2013, pp. 523–530.

[27] N. Li, C. Pedrinaci, M. Maleshkova, J. Kopecky, and J. Domingue, "OmniVoke: A framework for automating the invocation of web APIs," in *Proc. of IEEE International Conference on Semantic Computing*, 2011, pp. 39–46.

[28] M. Polák and I. Holubová, "Rest api management and evolution using mda," in *Proceedings of the Eighth International C* Conference on Computer Science & Software Engineering*, ser. C3S2E '15. New York, NY, USA: ACM, 2008, pp. 102–109. [Online]. Available: http://doi.acm.org/10.1145/2790798.2790820

[29] S. M. Sohan, C. Anslow, and F. Maurer, "Spyrest: Automated restful API documentation using an HTTP proxy server (N)," in *30th IEEE/ACM International Conference on Automated Software Engineering, ASE 2015, Lincoln, NE, USA, November 9-13, 2015*, 2015, pp. 271–276. [Online]. Available: http://dx.doi.org/10.1109/ASE.2015.52

[30] ——, "Spyrest in action: An automated restful API documentation tool," in *30th IEEE/ACM International Conference on Automated Software Engineering, ASE 2015, Lincoln, NE, USA, November 9-13, 2015*, 2015, pp. 813–818. [Online]. Available: http://dx.doi.org/10.1109/ASE.2015.92