6-2017

An exploratory study of functionality and learning resources of web APIS on programmableweb

Yuan TIAN
Singapore Management University, ytian@smu.edu.sg

Pavneet Singh KOCHHAR
Singapore Management University, kochharps.2012@phdis.smu.edu.sg

David LO
Singapore Management University, davidlo@smu.edu.sg

An Exploratory Study of Functionality and Learning Resources of Web APIs on ProgrammableWeb

Yuan Tian, Pavneet Singh Kochhar, and David Lo
School of Information Systems, Singapore Management University, Singapore
{yuan.tian.2012,kochharps.2012,davidlo}@smu.edu.sg

ABSTRACT

Web APIs provide various functionalities that can be leveraged by developers in building their applications. ProgrammableWeb, which is the largest and most active web API and mashup collection, provides a record of thousands of web APIs and mashups. However, important properties about these large number of web APIs, such as their functionality and support/resources for learning, have never been studied by the existing research work.

In this study, we perform an exploratory analysis on functionality and learning resources of 9,883 web APIs and 4,315 mashups listed on ProgrammableWeb, and find that: (1) web APIs provide a wide range of functionalities related to business solution, text analysis, data source, etc.; many of them are substitutable; only a minority have been used with other APIs; (2) a majority of web APIs on ProgrammableWeb have provided resources to support developers in learning how to use the APIs.

1 INTRODUCTION

Web APIs are becoming more popular and important in recent years as more web service providers release APIs to allow developers access their services. However, as the number of web APIs increases, there comes a challenge for developers to search and find suitable APIs for their projects. To deal with this challenge, ProgrammableWeb[1], a web site that stores information about various web APIs and applications that are made from these APIs (i.e., mashups), was created in 2005. Now, ProgrammableWeb is the largest and most active site about web APIs and their mashups; it provides information about thousands of APIs and mashups.

Due to the importance of ProgrammableWeb, a number of researchers have analyzed it to gain general insights on web APIs and mashups. Previous studies analyzing ProgrammableWeb mainly focus on the construction and analysis of a network of APIs that are used together in mashups, and how this network evolves over time [3, 5, 15, 16, 19]. However, important properties about this large number of web APIs, such as their functionality and provided support/resources towards better usability, which are important pieces of information that developers want to know[2], have never been studied.

To gain insights into the functionality and usability of web APIs, this study investigates functionality and learning resources of web APIs on ProgrammableWeb by answering the following research questions:

RQ1: What are the functionalities provided by the thousands of web APIs? How many of them have complementary and substitutable web APIs?
RQ2: How many resources are available for developers to learn the usage of web APIs?

To answer the above research questions, we construct a large dataset by crawling information about ***all*** web APIs and mashups available on ProgrammableWeb in May 2014. For each web API and mashup, we collect its description from its profile page. After a pre-processing step, our data set contains a total of 9,883 API descriptions, and 4,315 mashup descriptions. We pick a statistically representative random sample (sampling with 95% confidence level and 5% margin of error) of web APIs that contains 370 APIs; for each API in this subset, we manually collect additional information available on the API provider home pages, as well as information from other corroborating sources, to get a complete view of these APIs.

2 BACKGROUND

ProgrammableWeb stores a list of web APIs along with a list of mashups, which are applications that are built on top of the APIs. ProgrammableWeb has a collection of over 11,000 APIs and over 7,000 mashup profiles.

A web API is a set of functionalities that are provided either for free or for a fee and they can be accessed typically as REST (REpresentational State Transfer)-based web services. ProgrammableWeb stores information about each API on its profile page. The most important pieces of information are the short and long descriptions that describe the functionalities of the API. Other information like the home page of the API provider is also provided.

[1] http://www.programmableweb.com/

[2] Our preliminary survey with 15 professional web developers found that almost all of them (14 out of 15) carefully consider functionalities and usability of web APIs before adopting them. They often spend much time (more than an hour to even a week) to find suitable web APIs.

EASE'17, June 15-16, 2017, Karlskrona, Sweden

ACM ISBN 978-1-4503-4804-1/17/06...$15.00
https://doi.org/http://dx.doi.org/10.1145/3084226.3084286

A mashup is an application that is built from one or more web APIs and integrates these APIs into a new service. Mashups often mediate among a set of web APIs provided by a heterogeneous set of providers. A mashup typically consists of three components: data mediation component, process mediation component, and user interface customization component [8].

3 FUNCTIONALITY OF WEB APIS

3.1 Methodology

3.1.1 Sub-Questions. To investigate functionalities of web APIs on ProgrammableWeb, we propose the following three sub-questions:

1.1) What are the main groups of functionalities provided by web APIs?
1.2) How many web APIs have complementary web APIs?
1.3) How many web APIs have substitutable web APIs?

Answer to the first question will show the main features that are provided by web APIs listed on ProgrammableWeb. The second and third questions deal with the relationships among web APIs. In this paper, we consider two relationship types: complementarity and substitutability. ***Complementary APIs*** are the APIs that can be combined together to implement some functionalities. In this study, we define that two APIs are complementary to each other if they have been used together in a mashup that are listed on ProgrammableWeb. ***Substitutable APIs*** are APIs that provide similar features, such as Trulia API[3] and Zillow API[4], which are alternative APIs that developers can use when they want to search for real estate information. Knowing alternative APIs help developers to compare and select the most suitable API for their projects.

3.1.2 Approach. To answer the three questions described in Section 3.1.1, we first employ a topic modeling algorithm to learn common topics from a corpus containing descriptions of 9,883 APIs (i.e., short description, long description, and keywords) extracted from the API profiles on ProgrammableWeb. ProgrammableWeb maintains a list of category names (e.g., Travel, USA, 3D) and allows its users to assign one primary category and one or more secondary category using category names from their list. However, there are more than 450 categories in the list and some of them are too fine-grained that only few web APIs belong to those categories. For instance, there is only one web API (i.e., Kassabok) under the category "Budget". By applying topic modeling, we are able to categorize web APIs into broad categories. For instance, using topic modeling, the web API Kassabok can be grouped with other web APIs that provide financial management functionalities. Sites such as ProgrammableWeb could benefit from this approach by combining both topics learned from API descriptions and user selected categories to provide a hierarchical view of the functionality of web APIs.

[3]http://developer.trulia.com/
[4]http://www.zillow.com/howto/api/APIOverview.htm

Next, we automatically analyze the 4,315 mashups to find complementary APIs that are used together in at least one mashup.

For RQ1.3, we randomly select a statistically representative sample of web APIs from the 9,883 APIs. The size of a statistical representative sample set is determined by two measures, i.e., margin of error and confidence level. In this study, to represent the 9,883 web APIs, we set a standard 95% confidence level and a 5% margin of error[5], which results a sample contain 370 APIs. For each of the sampled API, we find a list of top-10 APIs that are the most similar to it based on their descriptions. We then manually investigate these APIs to see if they are substitutable.

RQ1.1: Topic Discovery. In this work, we use a topic modeling algorithm, namely Latent Dirichlet Allocation (LDA) [2], to discover the topics contained in the descriptions of web APIs. LDA is an advanced natural language processing technique that can be directly applied on a corpus (i.e., a set) of textual documents to discover topics without any training data. LDA takes a parameter K that determines the number of topics that will be outputted. For each document, LDA assigns a set of probabilities for it to belong to each of the K topic. For each topic, LDA outputs a list of words that are the most relevant to the topic. LDA has been used by many past studies in software engineering [4, 6, 9, 10, 18]. One of the closest work to ours, by Barua et al., uses LDA to discover common topics in StackOverflow questions and answers [1].

LDA processes a corpus of textual documents, thus, we need to convert the web APIs into textual documents. Each web API has a profile page on ProgrammableWeb which contains descriptions of the functionality of the API. We merge these descriptions (i.e., short description, long description, keywords) into a text file and then apply four preprocessing steps to clean it. We first remove HTML tags (e.g, <a href="...") from this text. Then, we apply Part-of-Speech (POS) tagger to infer for each word its part of speech (e.g., noun, verb, etc.). To do this, we use a popular tool namely Stanford POS tagger [14]. We just keep nouns and verbs, which contain more information describing the functionalities of an API and drop other kinds of words (e.g., conjunctives, adjectives, etc.). Next, we remove common English stop-words and 30 other most common words as they might appear so frequently that they would appear in many different topics. As the fourth step, we reduce words to their base forms using the standard Porter stemming algorithm [11].

Next, we use a Java implementation of LDA referred to as the Stanford Topic Modeling Toolbox[6] to learn topics from the pre-processed texts. LDA takes a parameter K which is the number of topics. We automatically and systematically pick the best value of K by selecting a value of K that can achieve the lowest perplexity score. Perplexity score has been used as a standard metric in the natural language processing

[5]For instance, if 90% of the web APIs in sample set have substitable APIs, then in 95% of time between 85%-95% of the web APIs on ProgrammableWeb have substitable APIs.
[6]http://nlp.stanford.edu/software/tmt/tmt-0.4/

(NLP) community to evaluate the performance of an LDA setting [2]. To compute the perplexity score, we separate our dataset into a training data (80%) and a testing data (20%). Next, we vary the value of K from 5 to 50 with a step of 5, and train LDA models from the training data and compute the perplexity of the learned model on the testing data. We find that that $K = 35$ is the best topic number for our dataset.

Table 1: Top 10 Topics Discovered by LDA

Name	Top-10 Representative Words
Business Solution	busi compani softwar enterpris solut system product technolog enabl help
Text Analysis	text languag analysi extract gener return engin specifi semant term
Data Source	databas librari research refer collect queri resourc scienc metadata record
Geographical Service	map locat weather place geocod forecase latitud longitud return displai
Social Media	social share network commun group profil peopl friend post member
Online Payment	payment card account transact credit process merchant bill invoic onlin
E-commerce	product shop order store price retail ecommerc affili purchas item sale marketplac
Shipping & Delivery	address valid ship number code verif looup mail deliveri zip
System Admin	monitor test server internet control devic secur perform system host
Finance	trade exchang currenc bitcoin financi market stock rate price get

RQ1.2: Finding complementary APIs. We regard a pair of web APIs as complementary if they have been used together in a mashup listed on ProgrammableWeb. Therefore, we go through each mashup and mark any two APIs used in the mashup as a complementary pair. In this way, we could get all APIs that are complementary to an API.

RQ1.3: Finding substitutable APIs. To investigate API substitutability, we investigate a statistically representative random sample containing 370 web APIs. For each of these 370 APIs, we first compute its similarity with other APIs based on their preprocessed text information generated in Step 1. We represent each web API profile as a bag of words, and then convert it to a vector of weights following the vector space model (VSM) [7]. More specifically, for each API, we create its vector of weights, where every weight corresponds to a word that appears in the preprocessed description of the API. The weight of a word is computed using the term frequency - inverse document frequency (aka. tf-idf) weighting scheme. The tf-idf weight of word w in document d (in our case: preprocessed description of an API) given a set of documents D (in our cease: preprocessed descriptions of all APIs), denoted as $tf - idf(w, d, D)$, is computed as:

$$log(f(w,d)+1) \times log\frac{|D|}{|\{d_i \in D | w \in d_i\}|} \quad (1)$$

In the above equation, $f(w, d)$ is the number of times word w occurs in document d, and $\{d_i \in D | w \in d_i\}$ represents a set of documents that contain word w. After converting each API into a vector of weights, we compute the similarity between an API and another API as the cosine similarity [7] between their corresponding vectors of weights. Equation 2 shows how the cosine similarity between a vector V_q and another vector V_q is computed.

$$Similarity(q,d) = cos(q,d) = \frac{\overline{V_q} \bullet \overline{V_d}}{|\overline{V_q}||\overline{V_d}|} \quad (2)$$

In the above equation, $\overline{V_q} \bullet \overline{V_d}$ represents the inner product of the two vectors and $|\overline{V_q}|$ represents the size of vector V_q.

For each of the 370 APIs, after we compute its similarities with other APIs (i.e, 9,882 APIs) , we rank the other APIs based on their similarity scores. We then manually investigate the top-10 most similar APIs to see if any of them are substitutable with the target API. In total, we manually check 3,700 API pairs to study the substitutability of APIs.

3.2 Results

Functionalities of Web APIs. Using LDA, we learn 35 topics from the profiles of 9,883 APIs. Due to space limitation, we only present the top 10 topics which cover the most number of APIs in Table 1. For each topic, LDA outputs a ranked list of representative words that are the most related to the topic. Based on these words, we manually assign a name to each topic. In Table 1, we present the top-10 most representative words for each topic. Note that all the words are in their stemmed form.

Number of complementary APIs. Table 2 shows the number of APIs that have various numbers of complementary APIs and some sample APIs. On the one hand, we observe that out of the 9,883 APIs, most of them (93.1%) have not been used together with any other APIs in any mashup. On the other hand, there are few APIs that are complementary with 50 or even more than 100 web APIs.

Table 2: Complementary APIs

#Comp.	=0	=1	>1 & <50	≥50 & <100	≥100
#APIs	9,200	152	517	10	4
Sample API	Yahoo Messenger	Amplify	Bing	Amazon Product Advertising	Facebook

Number of substitute API pairs. Out of the 370 sampled APIs, most of them (84.6%) have substitutable APIs. Few APIs (15.4%) that do not have substitutable APIs are mostly related to governmental service, geographical-specific service, and high-tech service. Some APIs (33.5%) have more than

5 substitutable APIs, and they are mostly related to cloud storage service, online payment, etc.

4 LEARNING RESOURCES OF WEB APIS

4.1 Methodology

4.1.1 Metrics. Robillard et al. conducted a survey on API learning obstacles [12]. They find that documentation and other learning resources are important for developers to learn and use an API. Therefore, in this work, we consider the usability of an API mainly by looking into its documentation and other learning resources. In this work, we propose six metrics to measure the amount of supporting resources provided by a web API owner; each metric considers a different learning resource such as user manual, discussion platform, and StackOverflow. We describe these six metrics and mention in Table 3. In Table 3, we also give the reasons why these metrics are important.

For each API, we assign 6 binary scores (i.e., 0 or 1) to the six metrics (UM1-UM6) presented in Table3. While 1 means the API has a corresponding supporting resource, and 0 means the corresponding supporting resource is not provided by the API provider. For example, we assign to an API a UM3 score of 1 if the provider of the API gives code samples that can be used by developers to learn the usage of the API. We show some examples of APIs which are supported by learning resource measured by UM1, UM2, UM3, UM4, UM5, and UM6 in Figures 1, 2, 3, 4, 5, and 6 respectively.

Product Advertising API

The Product Advertising API provides programmatic access to Amazon's product selection and discovery functionality so that developers like you can advertise Amazon products to monetize your website.

The Product Advertising API helps you advertise Amazon products using product search and look up capability, product information and features such as Customer Reviews, Similar Products, Wish Lists and New and Used listings. You can make money using the Product Advertising API to advertise Amazon products in conjunction with the Amazon Associates program. Be sure to join the Amazon Associates program to earn up to 8.5% in referral fees when the users you refer to Amazon sites buy qualifying products.

Figure 1: Intent Documentation for Amazon Product Advertising API

You can start using the **Google Fonts API** in just two steps:

1. Add a stylesheet link to request the desired web font(s):
   <link rel="stylesheet" type="text/css"
   href="http://fonts.googleapis.com/css?family=Font+Name">
2. Style an element with the requested web font, either in a stylesheet:
   CSS selector { font-family: 'Font Name', serif; }

Figure 2: Step-by-Step Guide for Google Fonts API

Example: Java API Client

```
public static String WSQuery(String User, String Pass, String Lang, String ID, String Text,
String Detail, String OutFormat, String Normalized, String Theme) throws Exception
{
   URL Url = new URL("http://svc8.bitext.com/WS_NOps_Val/Service.aspx");
   String sResponse = "";
   String postData =
"User="+User+"&Pass="+Pass+"&Lang="+Lang+"&ID="+ID+"&Text="+URLEncoder.encode
(Text,"utf8")+"&Detail=" +
Detail+"&OutFormat="+OutFormat+"&Normalized="+Normalized+"&Theme="+Theme;
   byte[] textdata = postData.getBytes("UTF8");
   // Prepare web request
   ...
}
```

Figure 3: Code Sample for Bitext API

What does the "insufficient virtual circuits" error message mean?

The "Insufficient Virtual Circuits" message occurs when the number of concurrent transactions queued to process on your virtual terminal exceeds the timeout period allowed to complete any one transaction. Under normal circumstances it will only occur if you are processing lots of transactions at the same time. If this becomes a persistent problem, you will need to discuss increasing the number of virtual circuits available from us. This may involve additional costs.

Figure 4: Error Handling Instructions for PayPoint.net API

Support

Live Support Chat | Email Support

Google+ Community | Stack Overflow

Report an Issue | Frequently Asked(FAQ)

*This page is **open source**. Make it better for swag or submit an issue to let us know how to improve it!*

Figure 5: Technical Support for Iron.io IronMQ API

4.1.2 Approach. To assign metric scores to APIs, manual inspection is needed. This process is a tedious one since we need to manually inspect web pages that describe each API (created by the API provider) and search StackOverflow pages to find if a learning resource for an API is available. Thus, we can not investigate all of the 9,883 web APIs listed on ProgrammableWeb. Therefore, we select the same statistically representative sample of web APIs (i.e., 370 APIs) which is used in RQ1, and manually decide the values of the metrics for each of the 370 APIs. We describe the detailed steps in the following paragraphs.

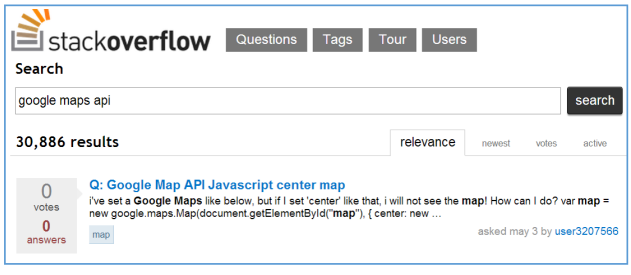

Figure 6: StackOverflow Post for Google Maps API

Table 3: Six Types of Learning Resources

Metric Name	Description	Reason to Consider
UM1: Intent Documentation	Description about the functionalities of an API and the purpose the API is intended to be used.	Intent documentation can help developers decide whether the API is suitable for his/her needs.
UM2: Step-By-Step Guide	Detailed description about the steps to get and use an API.	Step-by-step guide can help developers to quickly get started in using the API.
UM3: Code Sample	Code snippets or complete sample applications that are used to demonstrate special aspects or functionalities of an API.	A code sample can be reused and extended by developers for their needs. Developers will benefit from concrete examples that will help them better understand various functionalities of the API better.
UM4: Error Handling Instructions	Description about exceptions or common errors that can happen and how developers can deal with them.	Developers need to know how to handle unexpected situations, which often happen due to developers misconception on the functionalities and correct workings of an API.
UM5: Technical Support	Emails, chat services, and other means provided by API providers to help developers use their APIs.	Technical support makes it possible for developers to ask questions and receive help from experts.
UM6: StackOverflow Posts	Questions, answers, and discussions about an API on StackOverflow.	Relevant threads in StackOverflow allows developers to learn from questions that were asked by other developers and their answers.

Step 1: Manual Score Assignment. For each API in the sampled 370 web APIs, we assign scores of the six metrics presented in Table 3 to it. The scores of UM1 to UM5 are assigned based on information provided by the API provider on its home page, while the UM6's score depends on the StackOverflow search result. For each API, we thoroughly explore the API provider's home page by following relevant links in the home page to identify as much learning resources as possible. For UM6, we search StackOverflow using the API names as key words. We then assign the value of UM6 by browsing the returned questions and checking whether they are relevant to the API. This manual check is needed as sometimes a StackOverflow question can include words that appear in the API name but is not relevant to the API.
Step 2: Data Analysis. After step 2, each API in the sampled data set has 6 scores corresponding to the 6 proposed metrics. Next, we analyze how the scores vary for the 370 APIs.

4.2 Results

We observe that most of the sampled web APIs have intent documentation (92.7%), code sample (89.4%) and technical support (100%). Out of the 370 sampled API, 261 of them have step by step guides (i.e., 70.5%), 200 of them have instructions on how to deal with common errors (i.e., 54.0%). We find that only 206 of these APIs (i.e., 55.7%) have relevant posts in StackOverflow. We also calculated the number of learning resource types each API has, and find that that 62% of the sampled web APIs have at least 5 kinds of learning resources to help developers use them. 3% of the 370 APIs have only 1 learning resource to support developers.

5 THREATS TO VALIDITY

To mitigate the threat to external validity, we choose ProgrammableWeb as the target web API repository to conduct our exploratory study as it is the largest repository of web APIs. Still, web APIs on ProgrammableWeb might not present all web APIs. For RQ1 (except for the analysis of the substitutability of APIs), we consider all of the web APIs on ProgrammableWeb that are not out of service. However, since manual identifying substitble APIs/learning resources cost huge human efforts, for a part of RQ1 (substitutability of APIs) and RQ2, we sample a set of web APIs for investigation, results on which might not apply to the rest of web APIs on ProgrammableWeb. To mitigate such a threat on the generalizability of our findings, we randomly select a statistically representative sample of web APIs. In RQ2, we consider six metrics related to learning resources of web APIs. Although these six metrics cover various learning resources, they may not form an exhaustive set. In a future work, we plan to investigate possibility of adding more metrics.

6 RELATED WORK

6.1 Studies on ProgrammableWeb

Yu and Woodard investigated web APIs and mashups listed on ProgrammableWeb [19]. They applied network analysis on two types of networks on ProgrammableWeb. One network is the API-mashup network, where nodes represent APIs and mashups, and edges indicate which APIs are used by which mashups. The other network that they considered is the API network where APIs are connected if they have been used in one mashup. By analyzing the API-mashup network, they found that the distribution of number of APIs used in mashups follows power-law relationship and long-tail

property, which show that a large number of mashups tend to adopt a small set of popular APIs. They also found that diversity of mashups listed in ProgrammableWeb decreases over time indicating that mashup providers tend to reuse some design patterns when creating new mashups. Wang et al. studied the network and clustering properties of ProgrammableWeb and defined a new concept namely mashup entropy to measure the diversity of the mashup community [15]. Weiss et al. analyzed the evolution of API-mashup network of ProgrammableWeb and found that APIs that are frequently used in current mashups are likely to also be frequently used in future mashups [16]. Different from above works, our study focuses on functionality and learning resources of web APIs on ProgrammableWeb rather than properties of networks that are built from APIs and mashups.

6.2 Studies on Web Services

A numbert of studies have proposed approaches for web service discovery and matchmaking [13, 17]. These approaches help to match user needs against WSDL description files of web services in UDDI repositories. Wu and Wu proposed a system that takes in a web service specification in WSDL format and searches an UDDI repository for similar web services using four kinds of similarity measures, i.e., lexical similarity, attribute similarity, interface similarity, and QoS similarity [17]. Rong and Liu did a survey of web service discovery approaches and categorized them into keywords-based matchmaking, syntactics-based matchmaking, semantics-based matchmaking, and pragmatics-based matchmaking approaches [13]. Similar to these existing studies, we also investigate the functionality of web services. However, our goal is not to produce a better search technique, rather to investigate the range of functionalities that are provided by thousands of web APIs along with their complementarity and substitutability. Furthermore, we focus on web APIs, which are RESTful web services, rather than UDDI-based web services.

7 CONCLUSION & FUTURE WORK

In this study, we find that web APIs on ProgrammableWeb cover a variety of topics such as Business Solution, Data Source and many more. We also find that functionalities of many web APIs overlap, e.g., out of the 370 sampled APIs, 84.6% of them have substitutable APIs. Our study also finds that most of the web APIs on ProgrammableWeb have not been used together with any other APIs in any mashup. The wide variety of functional topics covered by web APIs, the large number of substitutable APIs, and the unexplored potentials of mixing-and-matching multiple APIs, highlight the need for a better tool to support web API search and recommendation. Such an advanced tool can help developers to better identify relevant web APIs satisfying a particular need in a shorter amount of time, weigh pros-and-cons of substitutable APIs before deciding on one, and recommend complementary APIs that can reduce wasted effort in reinventing the wheel. We plan to build such a tool in our future work.

Additionally, by checking six types of learning resources for 370 sampled web APIs, we observe that most of the web APIs have intent documentation (92.7%), code sample (89.4%) and technical support (100%). However, less than 80% of them have step by step guides (70.5%), instructions on how to deal with common errors (54.0%), and relevant posts in StackOverflow (55.7%). These observations highlight the need for additional tool support that can help developers in the creation of additional learning resources (especially step by step guides, instructions on how to deal with common errors, etc.) with ease. For example, it will be interesting to develop a tool that can mine multiple software forums to create a list of common errors and ways to deal with them. This is yet another direction of our future work.

ACKNOWLEDGMENTS

This research was supported by the Singapore Ministry of Education (MOE) Academic Research Fund (AcRF) Tier 1 grant.

REFERENCES

[1] Anton Barua, Stephen W Thomas, and Ahmed E Hassan. 2012. What are developers talking about? An analysis of topics and trends in Stack Overflow. *EMSE* (2012), 1–36.
[2] David M Blei, Andrew Y Ng, and Michael I Jordan. 2003. Latent dirichlet allocation. *JMLR* 3 (2003).
[3] Yuanbin Han, Shizhan Chen, and Zhiyong Feng. 2014. Mining Integration Patterns of Programmable Ecosystem with Social Tags. *Journal of Grid Computing* (2014).
[4] Abram Hindle, Michael W Godfrey, and Richard C Holt. 2009. What's hot and what's not: Windowed developer topic analysis. In *ICSM*. IEEE.
[5] Keman Huang, Yushun Fan, and Wei Tan. 2012. An empirical study of programmable web: a network analysis on a service-mashup system. In *ICWS*. IEEE, 552–559.
[6] Stacy K Lukins, Nicholas A Kraft, and Letha H Etzkorn. 2008. Source code retrieval for bug localization using latent dirichlet allocation. In *WCRE*. IEEE.
[7] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. 2008. *Introduction to Information Retrieval*. Cambridge University Press.
[8] E. Michael Maximilien, Ajith Ranabahu, and Karthik Gomadam. 2008. An Online Platform for Web APIs and Service Mashups. *IEEE Internet Computing* 12 (2008).
[9] Stephan Neuhaus and Thomas Zimmermann. 2010. Security trend analysis with cve topic models. In *ISSRE*. IEEE.
[10] Anh Tuan Nguyen, Tung Thanh Nguyen, Tien N Nguyen, David Lo, and Chengnian Sun. 2012. Duplicate bug report detection with a combination of information retrieval and topic modeling. In *ASE*. IEEE.
[11] Martin F Porter. 1980. An algorithm for suffix stripping. *Program: electronic library and information systems* (1980).
[12] Martin P Robillard and Robert Deline. 2011. A field study of API learning obstacles. *EMSE* 16 (2011).
[13] Wenge Rong and Kecheng Liu. 2010. A survey of context aware web service discovery: from user's perspective. In *SOSE*. IEEE.
[14] Kristina Toutanova and Christopher D Manning. 2000. Enriching the knowledge sources used in a maximum entropy part-of-speech tagger. In *EMNLP*. Association for Computational Linguistics.
[15] Junjian Wang, Huajun Chen, and Yu Zhang. 2009. Mining user behavior pattern in mashup community. In *IRI*. IEEE.
[16] Michael Weiss and GR Gangadharan. 2010. Modeling the mashup ecosystem: structure and growth. *R&d Management* 40 (2010).
[17] Jian Wu and Zhaohui Wu. 2005. Similarity-based web service matchmaking. In *SCC*. IEEE.
[18] Xin Xia, David Lo, Xinyu Wang, and Bo Zhou. 2013. Accurate developer recommendation for bug resolution. In *WCRE*. IEEE.
[19] Shuli Yu and C Jason Woodard. 2008. Innovation in the programmable web: Characterizing the mashup ecosystem. In *ICSOC*. LNCS 5472.