

API as a Social Glue

Rohan Padhye
IBM Research India
ropadhye@in.ibm.com

Debdoot Mukherjee
IBM Research India
debdomuk@in.ibm.com

Vibha Singhal Sinha
IBM Research India
vibha.sinha@in.ibm.com

ABSTRACT

The rapid growth of social platforms such as Facebook, Twitter and LinkedIn underscores the need for people to connect to existing and new contacts for recreational and professional purposes. A parallel of this phenomenon exists in the software development arena as well. Open-source code sharing platforms such as GitHub provide the ability to follow people and projects of interest. However, users are manually required to identify projects or other users whom they might be interested in following. We observe that most software projects use third-party libraries and that developers who contribute to multiple projects often use the same library APIs across projects. Thus, the library APIs seem to be a good fingerprint of their skill set. Hence, we argue that library APIs can form the social glue to connect people and projects having similar interests. We propose APINet, a system that mines API usage profiles from source code version management systems and create a social network of people, projects and libraries. We describe our initial implementation that uses data from 568 open-source projects hosted on GitHub. Our system recommends to a user new projects and people that they may be interested in, suggests communities of people who use related libraries and finds experts for a given topic who are closest in a user's social graph.

Categories and Subject Descriptors

K.6 [Management of Computing and Information Systems]: Project and People Management; D.2.8 [Software Engineering]: Reusable Software—*Reusable libraries*

General Terms

Human Factors

Keywords

Mining software repositories, usage expertise, social networks, recommender systems

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

ICSE '14, May 31 – June 7, 2014, Hyderabad, India

Copyright 2014 ACM 978-1-4503-2768-8/14/06 ...\$15.00.

1. INTRODUCTION

In recent years, there has been a rapid growth in the use of social platforms such as Facebook, Twitter and LinkedIn that enable people to discover and maintain relations for recreational or professional purposes. For example, Twitter suggests to users other people having similar interests whose tweets they may like to follow, while LinkedIn helps professionals find people with similar skill sets for recruiting or consulting. A similar parallel exists in the software development world as well. Developers collaborate with each other on projects using source control management systems. Platforms such as GitHub.com allow users to follow developers and projects of interest to keep themselves updated of new activity. However, subscribing to a person or project is still a predominantly manual activity in all such platforms.

We propose APINet, a system to automatically discover a developer's interest and/or expertise by mining software repositories and using this information to recommend new connections and communities. We first observe that most software projects use third-party libraries. We also observe that developers who contribute to multiple projects often use the same API in more than one project, indicating that the API forms a skill set for such developers. An analysis of 568 GitHub projects reveals that of the 1,355 developers who contributed to at least 2 projects, 395 of them used the same package in at least 2 projects. Of the 4,030 packages that were used across multiple projects, 2,996 were used by at least two developers. The APIs used in a project help estimate to some level what functionality a project is using or building upon. For example, a developer who has used the `org.apache.lucene` package in one project may be interested in connecting to other developers using Apache Lucene, or projects that require a search engine component, or even people who use Apache Solr, a library that is commonly found along with Lucene. APINet extracts developers' API usage profiles from code version archives, builds a social network of people, projects and libraries and uses statistical techniques to generate recommendations of other people, projects and library communities.

The main contributions of this paper include:

- A method to create a cross-project social network for developers by mining API usage from version management systems.
- A description of our implementation of this system for 568 Java projects hosted on GitHub.
- A discussion on potential applications of this system for finding experts, recommending people and projects to a user and discovering topic-specific communities.

2. SYSTEM OVERVIEW

In this section, we present a brief overview of APINet and discuss its initial prototype, which is built with data from GitHub. Figure 1 outlines the different stages in APINet. First, we mine a network of people, projects and API packages from data that is crawled from code versioning systems. We link API packages to the projects that have imported them and to developers who may be using them. Next, we resolve pairwise relationships between different entities which are connected in such a network with a Personalized PageRank (PPR) algorithm [4]. Such relationships spell out the association of an entity with respect to another entity, even if the two entities are not directly linked. PPR relationships become the basis for driving interesting applications such as recommending social connections, creating communities and so on. In the following subsections, we describe each of these stages in more detail and illustrate them with examples from the APINet prototype with GitHub data.

2.1 Mining API Usage

There are three types of entities in APINet—*project*, *person* and *package*. To bootstrap an APINet, we choose a set of *projects* and pull their source files from their respective code versioning systems. We extract the list of contributors from the change logs of the source files to set up the *person* nodes in the network. Also, we crawl the profile pages (if available) for these contributors to associate meta-data (e.g., email, affiliation) with the person nodes. *Packages* are resolved by parsing `import` statements in the source files. A key step is to filter the *packages* to keep only those that represent APIs. In our current implementation, we assume that packages that have been imported in at least two distinct projects belong to re-usable libraries.

We set up the initial APINet prototype with 568 top-starred Java projects in GitHub. We found 7,384 contributors for this set of projects, of which 5,015 had GitHub profiles; we only considered the latter set for recommendations. We parsed 23,919 unique packages. Out of these, 4,030 packages were imported in 2 or more projects.

In the network graph, there is an edge from a project k to every package p that is imported by some file in k . Similarly, a developer d is linked to a package p if we believe that d is familiar with the API of p . There are several ways of extracting this association, such as analyzing individual method invocations present in code contributed by the developer [10]. However, such techniques require knowledge of a project’s dependencies in order to resolve types from program fragments. Our prototype implementation trades off this accuracy for the ability to quickly analyze hundreds or thousands of projects using arbitrary build systems. We use the following heuristic: let $contrib(f, d)$ be the relative contribution made by developer d to the file f , computed as the fraction of total commits involving f that are authored by d , and let $imports(f)$ be the set of packages imported in file f , then our *confidence* that developer d is familiar with package p is computed as:

$$confidence(d, p) = \max_{\forall f: p \in imports(f)} [contrib(f, d)]$$

If a file has only one contributor, we can say with 100% confidence that this person is familiar with the APIs used in that file. We prune our graph by removing edges where the confidence is less than 50%. Now, since we are more interested in distinctive packages (e.g. `soot.jimple`) than

common packages (e.g. `org.junit`), we set the edge weights as the product of the *confidence* and the *inverse document frequency* (*idf*) of the package p , calculated as:

$$idf(p) = \log_{|K|} \frac{|K|}{|\{k \in K : p \in imports(k)\}|}$$

where, K is the set of projects analyzed and $imports(k)$ is the set of packages imported by project k .

A network created as above can help answer different kinds of queries on expertise of people and the projects they participate in. *Who is familiar with package A and has worked on project B? What APIs are common to project X and project Y?* One can import the network to a graph database (e.g. Neo4j, Titan) and use a graph query language (e.g. Ciper, Gremlin) to process such queries. Also, Codebook [1] provides a framework for running queries on rich graphs that connect a variety of software engineering artifacts (e.g. source files, methods, work-items). The focus of this paper is how to go beyond running exact queries toward generating ranked recommendations via statistical inference on such networks. Next, we describe computation of Personalized PageRank, the basis for recommendations.

2.2 Computing Personalized PageRank

Imagine a random surfer who walks over the APINet graph. Say, it starts at a *person* node, s and walks to any of its neighbor, v , package nodes with a probability, $C(s, v)$. From the package, v , it may again walk to any of v ’s neighbors—projects, packages or people. At it any point, it can choose not to walk to the neighbors but jump (called *teleport* in PageRank terminology) back to the source node s . In the steady state for such a random walk, the probability of the surfer landing at any node in our network reflects the strength of its association with the source node, s . Such a random surfer differs from the one described in PageRank [9] because here the *teleports* happen to the root and not to any random node in the network. This helps to personalize the ranks with the respect to the chosen root [4]. A similar method has been described by Chakrabarti et. al. [3] for searching on Entity-Relationship (E-R) networks. Formally, the $|V| \times 1$ Personalized PageRank vector for a node u in the graph $G = (V, E)$ is written as p_r , and is a solution to:

$$p_r = \alpha C p_r + (1 - \alpha) r$$

where, r is a $|V| \times 1$ teleport vector where $r(u) = 1$ and $r(v) = 0, v \neq u$; C is the $|V| \times |V|$ matrix with the probabilities of walking the different edges in E and α gives the probability of walking as against teleporting.

A Personalized PageRank (PPR) vector for any entity in our network helps easily determine the set of top- k entities that have a strong association with it. We calculate this vector for all entities to arrive at the pairwise PPR values. Personalized PageRank is an attractive approach for inferencing on the APINet network because it takes into account for the transitive relationships that impact association between any two entities. For instance, two people may be deemed similar in terms of their expertise if they have worked on similar but not exactly same APIs. Similarity between two APIs can in turn depend on the similarity of projects that they share, similarity of people who use them and so on.

2.3 APINet Applications

Now, we describe a set of applications around APINet, which are aided by the Personalized PageRank computation.

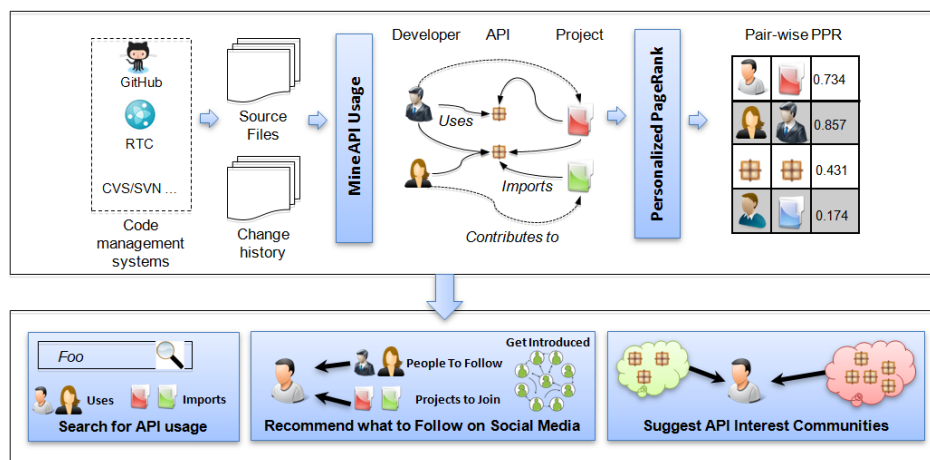


Figure 1: APINet: System Diagram

2.3.1 Search for API usage

A common query is to search for people who are skilled in a particular API (e.g. Apache Lucene) and projects which use it. To address such a query, first we select a set of packages, where there is a keyword match (e.g. find package names that contain `lucene`). Next, we would like to have a Personalized PageRank vector with respect to the nodes in our network corresponding to the selected packages. Such a vector is computed as a linear combination of the pre-computed PPR vectors for the individual matched packages [6] and presented as a ranked list of results.

2.3.2 Recommend what to Follow in Social Media

The Personalized PageRank vector for any person X leads us to ranked recommendations for what X can potentially follow in social fora:

1. *People to Follow*: People with top PPR scores are expected to be holistically similar to X ; in terms of API expertise, participation in projects and connections to people. Thus, they may be good candidates who X can follow on social networks, forums etc. Manual inspection of sample people recommendations obtained from the APINet-GitHub prototype suggests that they appear to be relevant if we set a PPR threshold of at least 0.0005. For such a threshold, we observe that there are 63 people recommendations on average per person (max=213, min=6).
2. *Projects to Follow*: X may be interested to keep abreast of developments in projects that have high PPR scores with respect to X . For our GitHub implementation, we performed a sanity test to check whether projects where people have contributed attain high PPR scores even when we remove all *person* \rightarrow *project* edges from our network. Out of the 4443 people for whom we could generate project recommendations, 3525 users were recommended at least one of their own projects within top 10 project recommendations.

2.3.3 Suggest API Interest Communities

An efficient way to group together related packages is to cluster them based on their pair-wise scores. We create a graph of *package* nodes that are connected by edges weighed with their PPR scores. Next, we partition this graph into communities such that the communities are densely connected and nodes from different communities are sparsely

connected. We apply the Louvain community detection algorithm [2] that maximizes modularity of communities. Once we obtain communities of packages, we observe how people are mapped to these communities by virtue of the packages used by them. A person's membership to different API communities can give a good picture of his/her API expertise. For the GitHub dataset, we find a total of 97 communities. Of these, 64 communities have upto 20 packages, 13 communities have 20-40 packages, and the rest have > 40 packages.

2.3.4 Illustration

Figure 2 is a screenshot of APINet populated with data from GitHub. It illustrates the profile of Drew Walters, a developer who uses libraries such as Apache Cordova, Google Zebra Crossing and the RIM API to develop a BlackBerry plug-in for the PhoneGap project in GitHub. First of all, we are able to recommend new social connections for Drew—people who share similar expertise in mobile platform development but have not collaborated with Drew directly on any project. Next, we suggest new projects that he can follow. We verified that all of the recommended projects can actually make use of Drew's expertise because they have dependencies on libraries such as the BlackBerry API. Further, we show that Drew is a member of a community that groups together different packages used in mobile development SDKs. Note that, in general, a person can be associated with different communities.


3. RELATED WORK

Mining developer expertise from source code change history is a well studied topic e.g. in Expertise Browser [8]. Here the focus is to suggest experts for specific modules in a given project, given how frequently and recently different developers have modified different code pieces. This development expertise is useful for intra-project tasks such as bug assignment and knowledge transfer but would be difficult to use in our scenario where the goal is to find inter-project connections. Instead we focus on usage expertise of library APIs which is a project-independent skill. Our current implementation mines API profiles based on file-level changes by a developer. However, we could further refine this by tracking API usage at a method level [10].

The Codebook framework [1] is a social-networking-inspired approach to create connections between artifacts and people in software repositories. Various relationships between

APINet: The Developer Social Network


Find experts on a topic Search!




Drew Walters
IBM
Austin, TX

Works on: [purplecabbage/phonegap-plugins](#)
Works with: [Alejandro Etchegovien](#), [Antonino Caccamo](#), [Libby Baldwin](#), [Brian Antonelli](#), [Chris Do...](#)
Uses: [net.rim.device.api](#), [net.rim.blackberry.api](#), [org.apache.cordova](#), [com.google.zxing](#)


Should Check Out These People



Don Coleman
Chariot Solutions
Philadelphia, PA
Uses: [net.rim.device.api](#), [org.apache.cordo...](#)




Eceenux
Poland, 3cities
Uses: [com.google.zxing](#), [android](#), [org.json](#)




Ryan Willoughby
Nitobi
Vancouver, BC
Uses: [org.apache.cordova](#), [com.google.zxin...](#)

[See more...](#)


... And These Projects



alunny/ChildBrowser
Hosted at: [Github.com](#)
Uses: [net.rim.blackberry.api](#), [net.rim.device...](#)



chariotolutions/phonegap-nfc
Hosted at: [Github.com](#)
Uses: [net.rim.device.api](#), [org.apache.cordova](#)



wildabeast/BarcodeScanner
Hosted at: [Github.com](#)
Uses: [org.apache.cordova](#), [com.google.zxing](#)

[See more...](#)

... Or These Communities

Libraries	People
android	scolivernet
com.google.zxing	dodola
com.phonegap	Tom Morris
net.rim.blackberry.api	Alex Kuijper
net.rim.device.api	Tad Fisher

Figure 2: Screenshot of a sample profile on APINet (<http://code.comprehend.in:8080/apinet>)

people and artifacts (such as a person fixing a bug or a work-item mentioning a method name) are modelled in the Codebook graph. Applications in this framework are built by defining regular language reachability problems on this graph. We are similar to their approach in modeling connections between software artifacts, but our method is different as we look for statistical similarity between entities and use it to find connections between people and projects based on similar APIs used. The cross-project re-use of software libraries has been explored by CodeWeb [7], which focuses on the code that uses these libraries. Our study revolves around exploring relationships between such libraries and the developers that make use of them.

Suggesting connections and identifying communities has also been studied in the context of friend-based social networks such as Facebook, Twitter, LinkedIn. For example, in [11] authors visualize communities in friend-based social networks using graph-based community detection algorithms, while in [5] the authors try to identify changing community structure over time in social networks. In this paper we explored use of such community detection algorithms on a network created from API usage information.

4. CONCLUSION AND FUTURE WORK

We have mined API usage profiles for a set of open-source projects and observed that the use of third-party library APIs can be considered a project-independent form of expertise. We have presented an approach for creating a social network of developers using their API usage as the connecting glue. Our initial implementation suggests to developers other projects and people they might be interested in based on similarity of their API profiles. We also detect communities by clustering together similar libraries and recommend these to people having related experience. Several potential applications can arise out of such a social network. For example, people and project recommendations could be useful for project recruiting or team formation. An API-based expertise search tool is useful for cross-project consulting and our social network can be used to determine common colleagues who can generate an introduction as well. Topic-specific communities that are detected from the social network can serve as platforms for information exchange around a particular domain.

We plan to extend this work in two directions. Firstly, the accuracy of API expertise calculation can be greatly im-

proved if we analyzed individual program fragments contributed by developers instead of relying on heuristics based on file-level package imports. One way of resolving type information is to rely on build systems such as Apache Maven, from which accurate dependency information can be extracted. The use of central repositories for libraries will also help in our second extension, in which we plan to extract high-level descriptions of libraries from sources such as API documentation, source code comments, etc. and use these to improve our community algorithms to group together competing offerings in the same domain. This can be used to build additional applications to support, for example, a developer who needs to choose one of many libraries offering similar functionality.

5. REFERENCES

- [1] A. Begel, Y. P. Khoo, and T. Zimmermann. Codebook: Discovering and Exploiting Relationships in Software Repositories. ICSE, 2010.
- [2] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre. Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2008.
- [3] S. Chakrabarti. Dynamic personalized pagerank in entity-relation graphs. pages 571–580. ACM, 2007.
- [4] T. H. Haveliwala. Topic-sensitive PageRank. WWW, 2002.
- [5] J. Heer and D. Boyd. Vizster: Visualizing online social networks. INFOVIS, 2005.
- [6] G. Jeh and J. Widom. Scaling personalized web search. WWW, 2003.
- [7] A. Michail. Code web: Data mining library reuse patterns. ICSE '01, pages 827–828, 2001.
- [8] A. Mockus and J. D. Herbsleb. Expertise Browser: A Quantitative Approach to Identifying Expertise. ICSE, 2002.
- [9] L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: bringing order to the web. Technical Report. Stanford InfoLab, 1999.
- [10] D. Schuler and T. Zimmermann. Mining Usage Expertise from Version Archives. MSR, 2008.
- [11] C. Tantipathananandh, T. Berger-Wolf, and D. Kempe. A framework for community identification in dynamic social networks. KDD, 2007.