Knowledge Discovery from Software Engineering Ecosystem

An Internship Report

Submitted in Partial Fulfillment for the Award of

M.Tech in Information Technology

By

Rahul Venkataramani (MT2011115)

To

International Institute of Information Technology, Bangalore.

June 2013

Certificate

This is to certify that the internship report titled **Knowledge Discovery from Software Engineering Ecosystem** submitted by **Rahul Venkataramani (MT2010115)** is a bona fide work carried out under my supervision at **Infosys Limited** from **the 1st of January, 2013** to **the 30th of June, 2013 (six months)**, in partial fulfillment of the **Master of Technology** course at **International Institute of Information Technology, Bangalore**. His performance & conduct during the internship was satisfactory.

[illegible] 55550 (On behalf of Dr. Srini)

Dr. Srinivas Padmanabhuni

Principal Research Scientist & Associate Vice President,

Infosys Labs,

Infosys Limited, Bangalore 560 100.

INFOSYS LIMITED BANGALORE

Place:

The 15th of June, 2013.

Acknowledgements

My last visit to Eclipse workspace[1] showed a staggering 42 projects. Ofcourse, I was not the author of each of them. But for a student who did not evince much interest in programming before coming to Infosys Labs, I owe my programming skills to Mr. Allahbaksh Asadullah. In the initial days, he taught me programming like an infant is taught walking. Sharing a similar penchant for jokes which have not always made others laugh, He provided me extremely interesting company and made my stay in Infosys memorable. I also owe my gratitude to Mr. Basavaraju M for the innumerable reviews which helped me find a name for myself in Google Scholar. It was a pleasure to work with Mr. Vasudev Bhat who never failed to add his pragmatic signature in every project discussion we had. I would also like to take this opportunity to thank Mrs. Sheetal and Ms. Nikita for their support in switching the fan off. Any acknowledgement would be incomplete without the encouraging words of my parents, hours of philosophy with Prof. Srinath and Open Systems lab-mates and all my friends who have put up with my poor jokes. I also thank Hatti Kappi[2] for keeping me awake during many an afternoon.

[1] `www.eclipse.com`

[2] `www.hattikaapi.com`

Contents

List of Figures

Abstract

The notion of coding being a lonely man's profession does not hold true in today's age. Use of technical forums for posting doubts and getting them resolved, source code repositories to version source code and interact with peer developers, community created *wikis* have made the process of software development a social activity than ever before. This has led to the creation of software engineering ecosystem which includes open source code repositories, technical question and answer sites, code *wikis* and customized social networking sites. In the light of these recent developments, research needs to be conducted to study these seemingly disparate elements in cogent fashion. As an initial step towards research in this area, we have looked into two problems

- Discovery of technical expertise from Open source code repositories
- Building a semantic search for large source code repositories.

Both these projects involve augmenting knowledge from one information network into another.

Chapter 1

The Software Engineering Ecosystem

In this chapter, we will introduce the basic philosophy in the research work undertaken and provide a foundation for the applications which have been described in the following chapters.

1.1 Introduction

Developers are increasingly turning towards online tools during the software development phase. Such tools have a social element built into them for interaction with peers.

Developers use online source code repositories for storing, versioning projects and collaborate with peer developers.The source code repositories store the various artifacts generated during the process of software creation. Some of these artifacts include bug reports generated during the various stages of software development, signatures of software developers involved, the number of commits made etc. This information is generated by developers because of the various interactions between them, the knowledge,belief that every individual possesses etc. This information can be used to derive greater insights into the software development process as a whole and solve a number of software engineering problems in specific. When a number of individuals collaborate and compete towards a certain goal, the intelligent behaviour exhibited by the group as a whole is termed as *collective intelligence*[6]. Collective intelligence is studied by practitioners and researchers from a number of disciplines including sociology, cognitive sciences, biologists, computer scientists,network theorists etc.. It is widely debated that the collective intelligence of a *group* is greater than the sum of individuals put together. Collective intelligence has been used to solve a number of problems which has proved tough to solve using only computers. In this paper, we propose to explore the paradigm of solving software engineering research problems using the collective intelligence of human beings. A number of sporadic attempts have been made in this direction without providing a unified theory to approach problems from this perspective. Each software artifact created by humans involve cognitive efforts of a group of individuals involved in the project. To develop a peice of software code involves coordination among developers, explicit or otherwise, accessing various related documents ,posting and answering questions on various question and answer sites, *wikis* etc.. Mining this metadata which is generated along with the actual code can potentially give a lot of insights about the development process and a number of problems which are solved by humans implicitly. We argue that the complex human brain solves a number of problems during the course of software creation.Solving these problems again using engineering methods is an overkill in most cases. As an application of this concept we have built a semantic search engine for large source code repositories which captures the intrinsic human notion of *domain*. As mentioned, the problem of collective intelligence can be looked into from various perspectives. In this paper, we attempt to look at it from the lenses of network theory. Network theory is an important tool to capture elements of collective human intelligence since it captures the nature and frequency of interaction among different agents in the network. A large number of sociological, economic and information theoretic problems have been solved by using social network analysis or SNA as it is popularly called. Although, a few problems in the software engineering domain too have been solved using network analysis, a convincing argument to show that these attempts can be concisely categorized into the domain of application of human intelligence has not been made. Most of the attempts have been ad hoc and lack the umbrella theory needed to study the subject. In the next section, we have provided an overview of the various attempts to use the collective intelligence of humans to solve engineering problems.

1.2 Related Work

Bruch *et al.*[3] were the first ones to argue that the power of collective intelligence is not captured during the software development process because of lack of tools. According to the authors, current day tools are optimized for a single user working on a code and does not use the knowledge of other developers. In a bid to fill the vacunae, the authors have developed a prototype of an editor called IDE 2.0 to utilize the elements of collective intelligence to provide newer features and better the existing editor functions like code completion, code snippet recommedation etc. The network of developers connected to each other through explict follower followee relationship or through implicit connections like using the same code package etc. have been studied in various contexts of software engineering. These efforts can also be seen as a manifestion of the use of collective intelligence to solve complex software engineering research problems. Pathapati *et al.*[13] first construct an implicit social network where the graph G is an undirected, weighted,bipartate graphs with one partition as set of project nodes P and other partition as a set of contributor nodes C. These two sets of nodes are connected to each other by a set of edges E. In this work, the authors use such a graph to retrieve similar project or people given a seed project(s).However, such an approach can be used only for local computations in a graph and fails for applications like project search where the entire knowledge of the graph is required.Network analysis has been used to solve a number of problems related to finding the impact of a bug on a system, classifying bugs according to different metrics etc. In a work on classifying bugs according to its severity[7], Huusing *et al.* constructed a socio-technical network which captured the dependency strength (coupling degree) information.In this model, the authors extend LDA over the source code project and measure the dependency of a module on other modules using topic model analysis. Using the approach, the authors have classified the dependency between software modules into either strong or weak relations. Applying social network network metrics like degree centrality, betweenness centrality, information centrality etc. helps to isolate the bugs which are most critical to the system. In a work on predicting failures[1], the authors argue that program dependency information and social factors in development when studied in isolation do not give sufficient insidght into the software development process. The authors construct a socio-technical network and then analyze failures. Using this approach, with a fair degree of success, the authors could predict failures in the next release of Eclipse[1].

A number of works in this way have harnessed the power of collective intelligence to solve software engineering problems without presenting the umbrella theory of how some of these problems are actually solved by humans during the software development process. In this paper, we attempt to provide a general approach to solve problems of such kind by

[1]www.eclipse.com

harnessing the collective intelligence of the human brain. This approach we hope provides a paradigm shift in the way problems in software engineering research are solved.

1.3 A human centric approach in software engineering research

A large number of tough problems are solved by cognitive processes in the mind during the software development process. In the steps outlined below, we will list the points that should be considered to harness this power of collective intelligence.

1.3.1 Is it completely solved by an individual developer?

During the development phase, a developer solves many problems some of which are not recorded explicitly in the platforms they work on. Sometimes, the developers do not have sufficient information about the problem that is being solved. In such cases, they seek help from fellow developers to collate information and overcome the information deficit. Infact, collating information from distributed sources is one of the hallmarks of collective intelligence. Sometimes, the thinking process of a developer may be altered or biased by a number of external factors including the projects that he contributes to, the works of his peer developers etc. Such biases also need to be accounted for in the model before solving it.

1.3.2 Signals necessary to capture the information

While a number of problems might be indeed solved by a developer, not all the signals may be adequately represented in the online space due to a number of reasons like cumbersome work, lack of time, privacy constraints, etc. Since these factors are subjective in nature, different developers leave different different traces of the development process. The underlying platform on which the developer is developing software also plays a vital role in understanding the development process. A software code repository system tracks a number of aspects of the software development process like recording the number of commits, recording of messages along with commit, timestamps of various events, the contribution of each developer etc.. The most challenging part of solving a problem using collective human intelligence is to map the signals generated by the developers to the problem at hand. Sometimes, it may not be so obvious and requires a bit of prose to explain the mapping.

1.3.3 Representing the symbols formally

In problems involving multiple developers where interactions among developers are very important or the thought process of a developer is biased by his peer developers, graph theory is a popular way of represent the influence of the network and derive insights into the problem. Some of the other popular models involve the use of topic models, discrete mathematics etc.

In this chapter, we have briefly introduced the notion of utilizing human intelligence in solving software engineering problem and in the following chapters we show applications to demonstrate the same.

Chapter 2

Discovery of Technical Expertise in Large Source Code Repositories

2.1 Introduction

During the software development process, developers post questions and answer doubts in various Question and Answer(Q&A) websites. The ‘usefulness’ of an answer is determined by the number of “upvotes” the answer receives from peer programmers. In such a model, the reputation of the user answering the question is ignored. Hence, it is difficult for an amateur developer to ascertain if the user answering the question has any experience of working on a project related to the domain of the question. Another drawback in most of the present day Q&A websites is the unavailability of a recommendation system to help a user posting a question identify the potential developers who can answer that question. Also, a user willing to answer a question has the additional task of finding questions which match his competence[4].

One of the ways of quantifying the expertise of a developer in a domain is by measuring his contributions in the said domain. Sometimes, the dataset in a domain is insufficient to make accurate recommendations. Winoto et al.[15] provide insights into how such a dataset can be augmented with an auxiliary dataset to overcome the information deficit. One of the key observations they make is that the two datasets must be semantically close to each other to make better recommendations. For e.g., movies and music are closer to each other than movies and textbooks. During the software development process, developers use GitHub to host and share code among peer programmers. They also use collaboratively edited Q&A websites like StackOverflow to discuss the problems that arise during this process. Since the content in these websites are semantically close to each other, they make a good case for cross domain recommendations. Dabbish et al.[5] have pointed out that humans, while browsing a large source code repository, make a number of non trivial semantic inferences about the quality of commit, the proficiency of the author etc. In this paper, we propose a model to capture the technical expertise of developers by mining their activity from the open source code repositories and its application in a target domain. We validated this model by building a recommendation system for online Q&A websites like StackOverflow by mining user expertise from GitHub.

2.2 Definitions

- **Technical Terms:** The various programming language constructs like method names, class names etc. constitute the technical terms. These terms T are mined from the source code repositories.

- **Tags:** In Q&A websites, users annotate the questions with words that describe the question. These are referred to as “tags”. Let *tags* represent the set of tags. e.g. javaio, IOException could be the tags for a question pertaining to a problem in Java I/O.

2.3 Proposed Model

The proposed model demonstrates the capture of expertise of a developer in a domain by mining open source code repositories and its application in a target domain like providing recommendations in an online Q&A website. The model is constructed in three stages as described in the following sections:

2.3.1 Capturing the Expertise of a Developer

The familiarity of a developer with a technical term is measured by his frequency of use of the term. The familiarity of a set of developers D with terms T can be represented by the relation $\gamma : D \times T \rightarrow n$, where increasing values of n denote better familiarity of the developer with the particular term. A value of $n = 0$ denotes that the developer has never used the term. In our model, a project P is modelled as $P = (F, D, \alpha)$, where F represents the set of source code files and $\alpha : F \times D \rightarrow \{0, 1\}$. A value of $\alpha(F_i, D_i) = 1$ denotes the file F_i is modified by a developer D_i. From the set of files F, we extract a set of technical terms used in each file. The set of terms and their frequency of occurrence is denoted by $\beta : (F \times T) \rightarrow n$ where n denotes the frequency of occurrence of a term Ti occurring in file Fi. Using the relations α and β, we arrive at the value for γ.

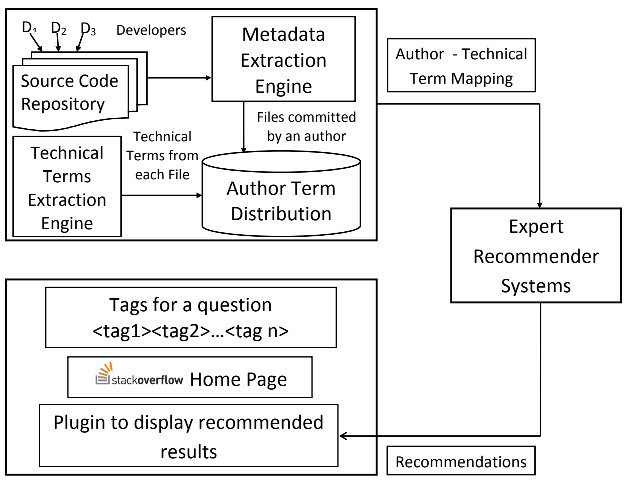
D1 D2 D3 Developers
Source Code Repository
Metadata Extraction Engine
Files committed by an author
Technical Terms Extraction Engine
Technical Terms from each File
Author Term Distribution
Author - Technical Term Mapping
Expert Recommender Systems
Tags for a question <tag1><tag2>...<tag n>
stackoverflow Home Page
Plugin to display recommended results
Recommendations

Figure 2.1: Block Diagram of Proposed Model

2.3.2 Mapping Technical Terms to Tags

The output from the first stage is a relation between a developer and a term. However, the tags used in the Q&A websites do not capture information at this level of granularity. Hence, there needs to be a mapping between the terms obtained from mining source code repositories and the technical tags used in the target domain. The mapping function M is defined as: $M : T \times tags \rightarrow n$. M is a one to many non-injective function i.e. every technical term can be a part of multiple tags.

2.3.3 Mapping Developer Expertise in the Target Domain

Using the relations γ and M, we model the expertise of a developer in the target domain by the relation $\delta : Tags \times D \rightarrow n$, where n denotes the relative expertise of a developer in a topic denoted by the given tag. Thus, we have modelled the expertise of a developer in a target domain by using a vocabulary of *tags*.

2.4 Model Evaluation & Future Work

To test the validity of the model, we built a recommendation system for StackOverflow. In particular, the two types of recommendations we make are:

- **Developer Recommendation:** *Given a question q, recommend one or more developers who possess the necessary expertise to answer the question.*
- **Question Recommendation:** *Given a developer d, recommend one or more questions for which he/she possesses the necessary expertise to answer.*

For building the prototype, we considered 5000 questions of Java domain from StackOverflow and 20 Java projects from GitHub. This data enabled us to build a database which maps the proficiency of every author w.r.t all the tags which co-occur with tag *Java* on StackOverflow using the proposed model. For lack of an API which allows us access of common users in GitHub and StackOverflow, we manually checked the tags of the questions to which the authors in GitHub, that we mined, have answered. For a sample of 15 authors, we found that around 7 authors answered questions with tags which our model has discovered he/she is proficient in. The rest of the authors answered more number of questions pertaining to their area of interest rather than the programming language concepts that they were proficient in.

In the current work, the expertise of a developer is modelled only on the basis of his proficiency in programming language concepts and not necessarily in his area of interest. In the future, we plan to extend the current model to reflect the user's area of interest along with his programming language expertise. This work is only a prototype in the larger vision of harnessing information from different information networks and leveraging the same to derive richer semantics. These semantics lend themselves to a number of applications like skilled recruitment, recommendation systems for Q&A websites, finding related content, etc.

Chapter 3

A Semantic Search for Large Source Code Repositories

3.1 Introduction

Increasing size and complexity of software systems has necesiated the need for source code repositories with version control system like Github to be used prominently in the development proecess. This has resulted in thousands of projects stored in popular open source repositories like Github and in private repositories of large software companies.

Since the number of projects in code repositories like Github [1] have crossed millions and encompass a wide range of domains, it is desirable for users to browse through projects in a domain. However, conventional source code repositories do not allow semantics like domain of a project. For example, given a query “distributed systems”, the search engine understands the term as a string rather than a domain. We define *domain* as a collective notion that exists among a statistically significant population about certain projects related to-

[1]`http://github.com`

gether by a common theme. The domain is not an implementation technology but a concept to which a number of projects and technologies belong. An average technical professional with relevant experience would immediately point out "Hadoop" among other projects as one of the top projects belonging to the domain "distributed systems". However, Github does not list Hadoop among its top results for the given query. This anomaly is because of the search engine's lack of semantic understanding of the term "distributed systems". Developers might not explicitly mention the name of the domain in the description page of the project to which it belongs to, due to a number of reasons ranging from carelessness to the domain being obvious given the context.

The notion of searching for projects in a given domain is closely related to classifying projects in a source code repository into different categories. In the classification paradigm, the projects are classified into a finite set of categories, unlike search engines where users browse through projects according to a keyword(s) entered according to his discretion. A few systems like SourceForge [2] have partially overcome this problem by forcing the users to annotate the projects with tags before uploading them. However, since the categories are defined statically and by humans, updating them according to a community's interests requires another round of human effort. Classification based on similiarity measures calculated from source code artifacts [14] cannot be always used because of restricted access to source code in many corporate repositories. In this work, we attempt to build a system to retrieve projects which belong to a given domain, the domain being input through the search query. This work is of high relevance to the developer community as it will help the developers quickly browse through the most popular projects in a given domain.

The model that we propose can also be used for other allied applications including catego-

[2] `http://sourceforge.net`

rization of projects, measuring popularity of projects among the developer community etc.. In the proposed model, we use an auxillary knowledge source to expand our domain into a bigger set of semantically related terms. We further use social network analysis techniques on the implicit network arising between different projects because of latent co-development relations. We have built DISCOVER, a prototype to demonstrate semantic search engine over large source code repositories using the model proposed in this paper.

3.2 Related Work

While most of the conventional scientific literature focus on code search engines, a distinction has to be made between a code search engine and our work. In our work, we attempt to build a system to retrieve a list of projects in a given domain while the primary function of a code search engine is to retrieve snippets of relevant code from a source code repository. A number of attempts [8], [11],[10] have been made to extracts topics from projects through the use of source code artifacts. Latent Dirichlet Analysis on source code artifacts has been popularly used in the software engineering community to extract topics from projects. However, in the current work we hold the view that a project name and a brief discription of the same contains sufficient semantic information for developers to understand the domain to which the project belongs to. We are not aware of any other work to build a semantic search engine using the developer project contribution network and augmenting it with an auxillary folksonomy dataset. In section 3.3, we describe the model for semantic search. Section 3.4 gives a brief overview of the experimental setup and the results obtained. Section 3.5 outlines future work.

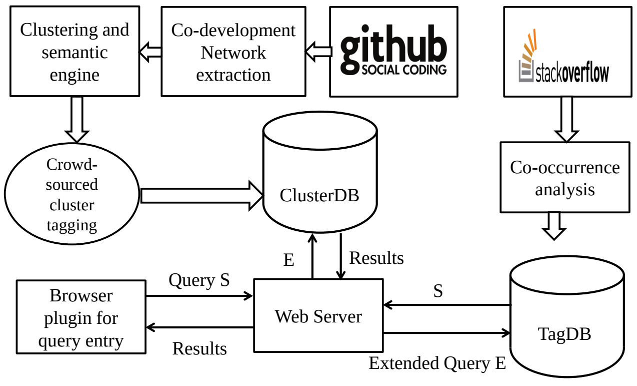
Clustering and semantic engine
Co-development Network extraction
github SOCIAL CODING
stackoverflow
Crowd-sourced cluster tagging
ClusterDB
Co-occurrence analysis
E
Results
Query S
S
Browser plugin for query entry
Web Server
TagDB
Results
Extended Query E

Figure 3.1: System architecture diagram

3.3 The model

Given a source code repository consisting of many projects the problem we address is the following: *Given a query representing a conceptual domain, find the projects in the corpus which belong to this domain.*

The proposed solution to the given problem can be better described in the following stages:

- Detecting clusters of projects in source code repository
- Tagging clusters of projects by crowdsourcing
- Expanding a given query to an extended list of keywords.
- Ranking clusters of projects.

3.3.1 Detecting clusters of projects in source code repository

Tagging[3] is a crowdsourced process to annotate a data item with non-hierarchial keyword or term. This kind of metadata helps describe an item and allows it to be found again by browsing or searching. Large public source code repositories like Github contains millions of projects and even the private respositories of large software development companies contains thousands of projects and is increasingly rapidly. It is an extremely tedious task to tag each project in such manmoth repositories indivdually.

The rationale behind clustering of projects is to reduce the human effort in tagging projects. While, it is practically infeasible to tag each project in any large source code repository, it is easier to tag the clusters generated from such a corpus. Thus, it is desirable to obtain clusters of related projects. Bird *et al.* [1] argued the need for dependency relations and contribution history to be used together in measuring the similarity between software artifacts. They claimed that the software components within a project that were developed by the same developer were implicitly related. We extend this claim to argue that software projects in a source code repository which had a number of common developers were related to each other.

Formally, *If projects P_1 and P_2 contain d developers who have contributed to both the projects and $d > k$, where k is a minimum threshold then the projects P_1 and P_2 are implicitly related to each other.*

An affiliation network [9] between developers and projects captures the latent co-development relations in a source code repository. An affiliation network is an undirected bi-partate graph with two types of nodes - developers D and projects P and can be formally de-

[3] `http://en.wikipedia.org/wiki/Tag_(metadata)`

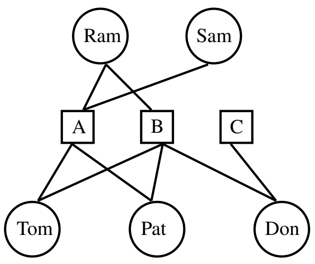

Figure 3.2: A developer contribution network. Circles denote developers and rectangles denote projects. An edge indicates the contribution of a developer to a project.

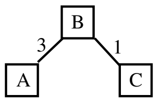

Figure 3.3: Project similarity graph. This graph is constructed from the the developer contribution network from Fig. 3.3.1. The edge weights indicate the number of common developers between the two projects.

fined as $A = (D,P,T)$. D and P are the two sets of vertices that represent developers and projects, respectively. T is the set of edges that represent the contribution of developers towards projects $T \subseteq \{(p,d)|p \in P\&d \in D\}$. Using this representation of a source code repository, we construct a project similarity graph $G = (V,E,)$ to capture the similarity between projects in a source code repository. V is the set of vertices that represents all the projects in a code repository. E is the set of edges representing the common developers between projects in P. The function $w : E \to \mathbb{N}$ is the corresponding count of common developers between projects. A higher value of w indicates a higher degree of similarity between projects. The weight of a edge E_{ij} between two projects P_i and P_j is calulated by $w_{ij} = \{|k \in D||T_{ik} \in T\&T_{jk} \in T\}$.

In the current work, communities of projects from the above project similarity graph

G are extracted using the algorithm proposed by Vincent *et al.* [2]. Given a group of communities, $C = \{C_1, C_2, ..., C_n\}$ and a set of projects $P = \{P_1, P_2, ..., P_n\}$, we describe community membership of a project by the boolean function $GroupMember(P_i, C_j)$. A $true$ value for $GroupMember(P_i, C_j)$ indicates that project P_i belongs to community C_j and is $false$ otherwise.

3.3.2 Tagging clusters of projects by crowdsourcing

The clusters obtained by performing the steps outlined in the require to be tagged with relevant keywords such that the tags collectively represent the domain that the projects in the cluster belong to. In the current work, we delegated the task of tagging the obtained clusters to a set of volunteers. Thus, each cluster is annotated with a set of tags which represent the domain along which most of the projects have clustered. Formally, the process of tagging can be represented by the relation, $\gamma : C \times T$, where C represents the clusters of projects and T represents the set of tags used to tag the obtained clusters.

3.3.3 Expanding a given query to an extended list of keywords.

A query S is a word(s) W input by the user of the semantic search engine. This query represents the domain in which the user desires to find relevant projects. These domain names might not be mentioned explicitly in the *wiki* page of the projects to facilitate easy retrieval. Also, projects in related domains or sub domains cannot be retrieved by naive string match searches. Hence, there exists a need for an ontology to encompass the different domains, their sub-domains and the relevant projects. However, constructing such an ontology by hand is an extremely tedious task and involves a lot of ambiguity because of divergent opinions in the community. The availability of a taxonomy to include the various domains, skills involved, technologies used would prove to be of immense importance to the technical community at large and finds a number of applications in recommendation systems, team formation strategies etc.. To this end, we attempted to construct a taxonomy using the folksonomy tagset data available from one of the largest question and answer sites, StackOverflow[4]. The question of constructing a taxonomy from folksonomy has been studied from the perspective of knowledge management [12]. However, each crowdsourced dataset has its peculiar features and the techniques used in the given domain could not be directly used in our work. In this work, we describe a method to discover semantically related tags to a given tag without providing richer semantics like subsumption, inheritance etc.. Thus, in this section our goal is, *given a query S with $|S| = m$, find the set of related words $|E| = n$ where $n > m$.* For example, given a query *no-sql* , we attempt to find out semantically related terms like *MongoDB*,*Cassandra*,*Neo4J*,*Hadoop* etc..

The StackOverflow dataset contains a community generated corpus of questions Q and answers A. Each question is annotated with a set of tags to facilitate for easy retrieval. The StackOverflow is a dataset of the form, $\mathbb{F} := (Q, T, \alpha)$, where Q represents the set of questions, T the set of tags used to annotate the questions and the binary relation between the tags and questions is represented by $\alpha \subseteq Q \times T$.

With the backdrop of the given dataset, we constructed a co-occurence graph between tags to reflect the community's perception about the relationship between different tags. These tags represent the different technical concepts commonly used in software engineering.The constructed co-occurence graph can be represented by the relation $G = (T, E, w)$. The vertices T represent the tags in the graph. Edges E between tags represent the co-occurence of

[4]`http://stackoverflow.com`

tags in a question. $E_{ij} = \{(T_i, T_j) | \forall q \in Q \& (q, T_i) \in \alpha \& (q, T_j) \in \alpha\}$. The weight function $w : E \rightarrow \mathbb{N}$ represents the co-occurence count between tags in the dataset.

This graph G is used to find the tags most related to a given query Q. From the tag which represents the given query S, all the neighboring tags are ordered by the edge weights with the given query tag. The top k neighboring tags are considered to be most relevant to the given query tags and helps us to expand the given query term into more relevant meaningful terms helping us achieve better results. Each term in the neighboring tags are not equally important and hence are weighted by a factor w_{ij}, representing the weight of the edge between the given query tag T_i and the neighbor T_j . The stopwords are removed from the list of neighboring tags which form a part of the expanded query list. If the query term contains more than a single word, thereby representing two or more tags in the graph G, the intersection of neighbors of the terms are given higher precendence by weighting them appropriately.

3.3.4 Ranking clusters of projects

The crowdsourcing technique helped us in tagging all the clusters with related keywords. The tags are used to collectively describe a cluster of projects. However, certain tags like *Java* might not add any meaning specific to the cluster since it might be used to tag all the projects where the language *Java* is used. To alleviate this problem and to increase the relevance of the tags to a cluster of projects, we apply term frequency inverse document frequency technique (TFIDF) to normalize the score of each tag for a cluster. This problem is akin to a document being asked to tag by a group of English speaking volunteers. If the document is written in English, there is a very low probability of *English* being one

of the tags. However, if it is written in French and the volunteers are only aware that the language used is French without understanding what the document is *about*, then there is a high probability of *French* being one of the tags. Thus, tags reflect a community's depth of knowledge about a topic. A dataset like StackOverflow is not bound to suffer from such bias since it enjoys a wide community support. However, in our experiments in tagging clusters of projects absence of bias cannot be ensured and hence we need TFIDF. The score s of a cluster C_i for a given query term S_j is calculated by multiplying the weights of the query terms with the weight of the matching term in the cluster.
$s(C_i) = w(S_j) \times w(T_i)$. The clusters are ordered according to the calculated scores. The projects within a given cluster are ordered according to the user controlled parameter like best string match, last updated project, most forks etc.. If the users knows the exact name of a project that he wishes to search for, the native search engine is a better tool. The use case for the current work is when a user wishes to browse through different projects from a given broad level domain.

3.4 Experiments

In this section, we will first describe the dataset and the experimental setup used in the experiments, and then present the evaluation results, in brief.

3.4.1 Dataset and experimental setup

We used two datsets for conducting experiments: StackOverflow dataset for query expansion and a partial repository of Github to test our hypothesis. StackOverflow is a collaboratively edited question and answer site for programmers and is one of the most used sites for posting technical questions. We used the latest data dump for StackOverflow at the time of writing the paper which was updated till March,2013. The raw dataset consists of 4,189,241 questions and 32,051 unique tags to annotate the questions. This dataset is used to construct the co-occurence graph G. The co-occurence graph is stored in a RDBMS instance. We used the Github REST API to retrieve a partial list of projects to form our experimental source code repository. Our partial dataset consists of 30,906 Java projects contributed to by 22,322 developers. Using this dataset, we construct the project similarity graph which was provided as input to Gephi [5] for obtaining communities of projects. Gephi inplements the algorithm proposed by Vincent *etc. al.*[2]. We ignore those communities which have a support of less than 5 repositories, since such communities are typically formed because of a single developer and is mostly written for personal or academic use. These clusters were manually tagged by a number of developers working in Infosys [6] and stored in a database. When a query is fired by the user, it is first expanded to a bigger set by the query expansion engine using the StackOverflow co-occurence graph. Through emperical analysis, we observed that doubts concerning programming languages being ubiquitous in nature, the tags representing programming languages were inevitably strong neighbors of most queries. However, since our objective is to build a system which retrieves projects belonging to a given domain and is agnostic of the programming language used, we have eliminated the names of language from the expanded query list by listing them as stop words. This expanded query set is then used to retrieve the most relevant clusters from the given source code repository.The setup for expanding the query list and ranking the clusters is hosted on an internal Apache Tomcat server. The results are displayed to the user

[5]`https://gephi.org`

[6]`www.infosys.com`

through a Google Chrome extension which modifies the default Github Search engine to use our implementation instead.

3.4.2 Evaluation results

The evaluation of the results of the semantic code search engine can only be evaluated quantitatively because of a lack of a gold standard in such a discipline. The evaluation of our current work was carried out in a couple of stages. In the first stage, we tested our hypothesis of *Developers contribute to projects not randomly but only according to their interest areas.* Since, our clusters are formed on the basis of above mentioned underlying hypothesis, we tested the veracity of our hypothesis by first tagging each cluster independently by two developers. We selected 15 developers working across various teams at Infosys to tag 10 clusters of projects each in the first round. In the next round, the clusters were exchanged between the developers without their knowledge and they were requested to perform the same exercise again. If, for a given cluster the tags obtained from the first and second round exhibit a high degree of similarity, it means that the clusters are formed by an underlying theme which is evident to developers across a community. To measure the agreement between two developers over the tagging of a given cluster, we used a measure similar to Cohen's Kappa since it cannot be directly used where 1 item is tagged by N evaluators.

Agreement measure $\gamma = (\mathbb{P}_{max} - \mathbb{P}_{actual})/(\mathbb{P}_{max} - \mathbb{P}_{min})$ where $\mathbb{P}_{max}$ denotes the maximum number of tags for a given cluster that is possible for a given cluster from k developers. Here, we have assumed that a developer is forced to tag only n tags for a cluster. Thus, k developers, if in disagreement with each other can tag at the worst case, a maximum of $k * n$ uniquely different tags. This is represented by $\mathbb{P}_{max}$. If all the developers are in total agreement over each other, the total number of unique tags would be n. $\mathbb{P}_{min}$ represents the theoretical minimum. Thus, in the best case scenario, when every developer has the same idea about the cluster,i.e. $\mathbb{P}_{actual} = \mathbb{P}_{min}$ and $\gamma = 1$. In the worst case scenario, $\mathbb{P}_{actual} = \mathbb{P}_{max}$ and $\gamma = 0$. The results are still under evaluation.

3.5 Future Work

Through this work, we have explored the area of extracting semantics from source code repositories through techniques used in social network analysis. We have also provided an example of augmenting knowledge from one technical forum into another to provide better insights. This work is a part of a larger vision to model the entire software engineering ecosystem comprising of source code repositories, developers, question and answer sites, technical blog posts, wikis and use the information to arrive at improved semantics. This work will be useful to large corporations with burgeoning source code repositories, online public source code repositories like Github, SourceForge etc.. In the future, we plan to look at the semantics involved in folksonomy dataset and make further attempts towards generating a taxonomy with formal relationship between different tags. This work would greatly benefit the technical community. We also plan to look at tagging clusters of projects automatically without human intevention as this would make the system truly automatic.

Chapter 4

Conclusion

Through the work, we have investigated methods to model the software engineering ecosystem. As an initial step towards research in this area, we have explored augmenting knowledge from question and answer sites with source code repositories and thereby deriving richer semantics. We envision that in the future, knowledge from disparate information networks would be used cohesively with each other in the creation of interesting applications. Our investigations lead us to believe that many complex software engineering problems could be solved relatively easily by using the collective intelligence of developers. We anticipate a number of tools and platforms built to harness the collective intelligence of developers, for which the current research provides a number of clues.

Bibliography

[1] C. Bird, N. Nagappan, H. Gall, B. Murphy, and P. Devanbu. Putting it all together: Using socio-technical networks to predict failures. In *Software Reliability Engineering, 2009. ISSRE'09. 20th International Symposium on*, pages 109–119. IEEE, 2009.

[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(10):P10008, 2008.

[3] M. Bruch, E. Bodden, M. Monperrus, and M. Mezini. Ide 2.0: collective intelligence in software development. In *Proceedings of the FSE/SDP workshop on Future of software engineering research*, pages 53–58. ACM, 2010.

[4] D. Cosley, D. Frankowski, L. Terveen, and J. Riedl. Suggestbot: using intelligent task routing to help people find work in wikipedia. In *Proceedings of the 12th international conference on Intelligent user interfaces*, pages 32–41. ACM, 2007.

[5] L. Dabbish, C. Stuart, J. Tsay, and J. Herbsleb. Social coding in github: transparency and collaboration in an open software repository. In *Proceedings of the ACM 2012 conference on Computer Supported Cooperative Work*, pages 1277–1286. ACM, 2012.

[6] W. Dong et al. *Modeling the structure of collective intelligence*. PhD thesis, Massachusetts Institute of Technology, 2010.

[7] W. Hu and K. Wong. Using citation influence to predict software defects.

[8] A. Kuhn, S. Ducasse, and T. Grba. Semantic clustering: Identifying topics in source code. *Information and Software Technology*, 49(3):230 – 243, 2007. 12th Working Conference on Reverse Engineering.

[9] S. Lattanzi and D. Sivakumar. Affiliation networks. In *Proceedings of the 41st annual ACM symposium on Theory of computing*, pages 427–434. ACM, 2009.

[10] E. Linstead, S. Bajracharya, T. Ngo, P. Rigor, C. Lopes, and P. Baldi. Sourcerer: mining and searching internet-scale software repositories. *Data Mining and Knowledge Discovery*, 18(2):300–336, 2009.

[11] E. Linstead, P. Rigor, S. Bajracharya, C. Lopes, and P. Baldi. Mining concepts from code with probabilistic topic models. In *Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering*, ASE '07, pages 461–464, New York, NY, USA, 2007. ACM.

[12] K. Liu, B. Fang, and W. Zhang. Ontology emergence from folksonomies. In *Proceedings of the 19th ACM international conference on Information and knowledge management*, pages 1109–1118. ACM, 2010.

[13] S. K. Pathapati, S. Venugopalan, A. P. Kumar, and A. Bhamidipaty. People and entity retrieval in implicit social networks. In *Internet Multimedia Systems Architecture and*

Application (IMSAA), 2011 IEEE 5th International Conference on, pages 1–8. IEEE, 2011.

[14] T. Wang, G. Yin, X. Li, and H. Wang. Labeled topic detection of open source software from mining mass textual project profiles. In *Proceedings of the First International Workshop on Software Mining*, pages 17–24. ACM, 2012.

[15] P. Winoto and T. Tang. If you like the devil wears prada the book, will you also enjoy the devil wears prada the movie? a study of cross-domain recommendations. *New Generation Computing*, 26(3):209–225, 2008.