Sociometrics in Software Engineering

Automated Social Recruiting through GitHub

by

Sagar Sachdeva, B.E.

Dissertation

Presented to the

University of Dublin, Trinity College

in fulfillment

of the requirements

for the Degree of

Master of Science in Computer Science

University of Dublin, Trinity College

September 2017

Declaration

I, the undersigned, declare that this work has not previously been submitted as an exercise for a degree at this, or any other University, and that unless otherwise stated, is my own work.

Sagar Sachdeva

August 27, 2017

Permission to Lend and/or Copy

I, the undersigned, agree that Trinity College Library may lend or copy this thesis upon request.

Sagar Sachdeva

August 27, 2017

Acknowledgments

Over the last one years, my journey as a masters student of Trinity College Dublin has been great. Firstly, I would like to express my sincere thanks to Prof. Stephen Barrett, who guided me in every phase of work and helped me in the successful completion of my thesis work. Without his support, the successful completion of this dissertation would be unmanageable.

I would particularly like to thank all the people in the software industry who responded to the questionnaire and helped in evaluation of my thesis. Last, but not least, I am grateful for the love and support given to me by my family all through my life to achieve my dreams.

Sagar Sachdeva

University of Dublin, Trinity College

September 2017

Sociometrics in Software Engineering

Automated Social Recruiting through GitHub

Sagar Sachdeva, M.Sc.

University of Dublin, Trinity College, 2017

Supervisor: Stephen Barrett

In order to find potential software engineers, companies search for candidates on various job portals and compare their competencies and experience with the job requirement. But this is a very cumbersome process as there is no way to validate the skills on a resume. Also, many times resume is not up to date and there is low availability of niche profiles. There are cases of fake projects and skills in resume as well.

The aim of this dissertation is to find possibility of existence of an automated system that provides recommendations of software engineers for a job. We have developed a system that identifies skills of software engineers from contributions done in open source platform, GitHub, and matches them against the skill requirements presented by a job description. The extraction of skill from the job description is done through an information extraction system built using natural language processing technique.

Our results indicate that existence of such an automated system is possible and is a scalable, efficient and effective solution for the recruitment problems mentioned above.

We have also outlined the limitations of our system in this dissertation.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

In this chapter, we give a brief overview of the entire dissertation. Section 1.1 discusses about the motivation behind this dissertation, where we describe the problems with current recruitment process and how social recruiting is becoming more and more popular. Section 1.2 defines research questions and scope for this dissertation. Section 1.3 defines the steps taken by us, as part of the workflow, in order to answer the defined research questions. And finally, section 1.4 presents the road-map for the dissertation.

1.1 Motivation

This section will describe the factor that motivated the topic of this dissertation and some background information.

1.1.1 Problems with Recruitment Process

Finding right candidates for a job is considered the most crucial step in a recruitment drive. But this is often the most challenging and time consuming step. According to a survey of talent acquisition leaders [50], 52% said that the most difficult part of recruitment was identifying the right candidates from a large applicant pool. Talent acquisition leaders are feeling increased pressure to find better methods of candidate sourcing, screening, and

shortlisting. Resumes have always played an integral part in shortlisting candidates in the recruitment process. According to a recent report [33], 93% of the recruiters still rely on resumes to find good candidates. But according to the industry stats [50], 75 to 88 percent of the resumes received for an open job requirement are unqualified or not strong enough to move forward with the interview process. Also, in an article [35], McCuller highlights that there is a lot of noise, in terms of inaccurate and irrelevant information, in the resumes of the candidates. Some candidates put fake experience and fake projects in their resumes. And validating such details from the resume as part of the initial screening process is a difficult and time consuming.

1.1.2 Growing Popularity of Social Networking Platforms

Social Media has brought a revolution in the way people collaborate and share information. Platforms such as Facebook, LinkedIN, Twitter etc enable millions of users across the globe to interact with each other and share content. Due to the abundance of information present on such platforms, there has been an increased attention over the last decade to the social aspects of software engineering, especially creating tools to improve software engineering practices. This increased influence of social media has triggered a new advancement in the field of candidate recruitment, called social recruitment. As per wikipedia [30], social recruiting or social hiring is recruiting candidates by using social platforms as talent databases. Social network platforms give a competitive edge in locating and engaging the best candidates available for an organization's talent requirement. According to a report [33], 92% of the recruiters are now using social media platform in their recruitment process, indicating the growing popularity of social recruiting.

Apart from the most popular social networking platforms mentioned above, information from social coding platforms can prove to be of great significance for the recruitment of software engineers. Such platforms enables developers to write and share codebases in

a collaborative environment. There are various social coding platforms such as GitHub, Bitbucket, CodePlex and Google Code, but GitHub [51] is claimed to be one of the largest code hosting sites. GitHub supports more than 22 million people to learn, share and work together to build softwares. There are more than 62 million projects hosted on the platform [51]. It also exposes information about user's activities as an API over web. This information can be used to generate important insights, thus helping recruiters to find the most suitable candidates from this pool of talented software engineers.

Even though GitHub exposes most of the information it has about a software engineer, it does not provide the insight about the skills of the software engineer directly. Therefore in this thesis, we leverage the information provided by the GitHub to find important parameters that help in determining the skills of the software engineers and rate those skills based on the contributions made, so as to address the shortcoming of recruitment of software engineers.

1.2 Research Question and Scope

During our initial research, we found that GitHub, as a source coding platform, provides traces of not only the technical capabilities of a software engineer, but also his behaviour while working in a team. It gives information on how he perform in a collaborative environment, giving deep insights about the character of the software engineer. These insights are very crucial for decision making, as the candidate who gets finally selected for the role has to perform his tasks being a team member. And such insights about the behaviour of the candidate cannot be evaluated from his resume, but can be gathered from his contributions across various projects in GitHub. Based on the research objective of this thesis, we have formulated following research questions:

- **RQ1** : What signals can be used from the GitHub profile of a developer to automate the process of identification and rating of his skills?

- **RQ2** : What technique would be suitable to automate the process of identification of required skill from a given job description and rate those skills based on their level of requirement?

- **RQ3** : What recommendation strategy would be best suited to match skills identified from developer's GitHub profile to the job specification presented by the recruiter?

Based on the research questions presented and with the aim of overcoming the recruiter's issues highlighted above and automating the initial screening stage of the recruitment process, we have restricted the scope of this dissertation to following:

- Develop an automated system to extract technical skills of a software engineer and rating those skills based on the contributions done by him over GitHub repositories.

- Develop an automated system that takes in a job description and using Natural Language Processing (NLP) techniques, extracts the required skills and rates those skills as per their level of requirement.

- Develop a recommendation system that takes in the output of both the above systems as input and generates a list of best suited software engineers for a given job description.

1.3 Workflow

Following are the steps of the workflow for our thesis:

1. Defined the research questions : based on the preliminary research we did in the areas of social recruiting and the present shortcoming of the interview process, we defined the research questions for the dissertation.

2. Further study to understand the domain : the next step was to understand the domain by reading through the findings of researchers in the areas of social coding platforms, natural language processing and recommendation strategies. This was done not only to get a better knowledge about the domain but to derive useful insights from the results of the researches already been done in these areas.

3. Implementation of the skill extracting system : designed and implemented the system that identifies and rates skills of software engineers from GitHub and generates profiles for the software engineers.

4. Implementation of job description parser : designed and implemented the system that takes in job description and extracts required skills from it using natural language processing techniques.

5. Implementation of the Match-Making system : designed and implemented a recommendation system that takes in skills of software engineers and required skills for a job, and generates a list of best suited candidates for that job.

6. Testing and generating Output : tested the entire application and generated the output for a selected set of GitHub users and a set of job descriptions as the input.

7. Evaluation : evaluated the entire approach of the system based on the results fetched from the previous phase .

1.4 Road-map

This dissertation is organised as follows:

- Chapter2 gives the background details about social coding platform : GitHub, existing research done on this platform. It also discusses briefly about Natural Language Processing technique and then describes the research done on various information

extraction systems. Finally it gives a brief overview about the various recommendation system techniques available and discusses the research done on job related recommendation systems

- Chapter 3 describes the approach the approach taken to identify and rate user skills based on his information available on GitHUb. It describes the information extraction technique used to extract skills from a job description. Finally it specifies the recommendation technique used to find software engineers which match closely to the job description.

- Chapter 4 discusses the implementation of all the approaches discussed in Chapter 3. It also outlines the limitations of these approaches taken.

- Chapter 5 discusses the validation process to evaluate the approach taken for accomplishing the research goal and shares the result for the dissertation.

- Chapter 6 provides the conclusion of the dissertation along with the contributions made and recommendations for the future work.

Chapter 2

State of the Art

In this chapter, we give an overview of existing research done around the research questions, which are defined in section 1.2. Section 2.1 gives a brief overview of the features provided by social coding platform, GitHub. Section 2.2 describes the existing research around the signals available on GitHub which can be used to find skills and competencies of a software engineer and also discusses the conclusions we derived from this research. Section 2.3 discusses briefly about Natural Language Processing technique and then describes the research done on various information extraction systems. The section also presents some of the conclusions that were derived from the research. Section 2.4 gives a brief overview about the recommendation system and its various available techniques. The section also presents the research done on job related recommendation systems and outlines the conclusions we derived from this research.

2.1 Open Source Coding Platform : GitHub

GitHub is a social network that has changed the way we work. It started as a collaborative platform for software engineers and now it is the largest online storage space of collaborative work that exists in the world. GitHub provides three main features : fork, pull request and merge.

Forking is the mostly commonly used way of collaborating on open source projects [36]. A fork is basically a copy of a repository on which the contribution is to be made, but with a link back to the original repository. This allows a GitHub user to freely experiment with the changes on the repository without affecting the original project. One of the most common example of forking is bug fixing which requires 3 basic steps : forking a repository to which fix is to be made, making the fix, submitting a pull request to the project owner of the original repository.

Pull request is another important feature on Github which helps developers collaborate efficiently [47]. In simple terms, It is mechanism for a developer to notify the collaborators of a repository, that the changes onto the repository are done. Once a pull request is opened, the developer can discuss and review the potential changes with the collaborators and follow-up comments before these changes are actually merged into the repository.

Merge is a mechanism provided to merge the changes done in the pull request to the upstream branch [48]. Any contributors with push access to the repository can merge the pull requests onto the repository in GitHub. If pull request has merge conflicts, the pull request can be checked out locally and merged using command line to resolve these conflicts. Then changes done to resolve these conflicts can be pushed back to the repository.

There are many other features in GitHub. One of them is Issues and Issues Tracker. Issues is a great way to keep a track of tasks, enhancements and bugs for your projects [49]. Githubs Issue Tracker has a special focus on collaboration. It provides ways to assign milestones, labels and assignees to issues and helps to filter out issues based on these parameters. It provides a way to assign labels to the issues which helps to organise different types of issues. There can be a chain of comments on an issue which aims to solve or resolve the issue. There is a Watch feature in GitHub where a user can subscribe

to receive notifications on events on a repository by watching the repository. A GitHub user can also follow another user, which will help the follower receive notifications about person being followed regarding repositories being created or forked or starred by him and check his public activity.

2.2 Finding Skills of Software Engineers

2.2.1 Research on Coding Platform : Github

GitHub is the most popular social coding platform, and based on the amount of data it holds, it is a popular domain for researches. Our initial exploration was quite diverse, mainly to understand GitHub as a platform and to know more about the previous and current research work going on over GitHub. A research done at University of Victoria [25] aimed at understanding the various characteristics of repositories in GitHub so that users can take advantage of GitHubs main features such as commits, pull requests and issues. Their research showed that majority of projects (71.6% to be precise) in GitHub are personal and inactive and are using this platform as free storage. They also inferred that if commits in pull-request are reworked, GitHub records only commits that are the result of peer-review, not the original commits. There was another research done in Singapore Management University [28], which aimed at identifying influential developers and projects on a sub-network of GitHub using PageRank. The research inferred that social coding improves collaboration among developers. Also, recommendation system to choose suitable developers to work together for particular projects in GitHub was identified as an area of future work in this research.

In terms of finding skills of software engineers and rating those skills, our research mainly aimed at finding out the key signals in a users GitHub profile which give a sense of how good a software engineer he is. So we went through some of the researches that

were done on similar areas. Marlow et al. [2] did research to analyse activity traces of GitHub users and identify signals which will aid in software developer recruitment and hiring. They conducted interviews with both employers and job seekers that pointed to specific cues provided on profiles that led them to make inferences about candidates technical skills, motivations and values. These cues were seen as more reliable indicators of technical abilities than information provided on resume. The research pointed out that in hiring domain, deception of qualification is a concern. From the interviews conducted, it was inferred that presence of code that was developed openly and shared with others, signalled strongly that developer valued openness, transparency and participation in a community. Active participation in other peoples projects was found to be the most reliable signal of commitment to the open source mindset. Publicly building on another persons work served as an assessment signal that the candidate is truly bought into the open source mindset. Also, they found that personal projects signalled a candidates love for programming and suggested a willingness to learn. An accepted commit to a high status project signalled the candidate was someone who produced quality code.

Badashian et al. [4] in their research set out to mine, analyze and correlate the members core contributions, editorial activities and influence in the two networks: GitHub and Stack Overflow. While studying the activities on GitHub, they defined three higher level indicators : development indicator which was the sum of commits and pull requests handled by the developer, management indicator which was the sum of issues reported, issue comments, issues closed or handled and projects watched by the developer, and lastly popularity indicator which was the number of followers for the developer. The research found strong correlation between the activities done in these three indicators which meant that active developers not only contribute to the main development activities but also engage in other managerial activities and gain more popularity.

Another research by Marlow et al. [3] did a qualitative investigation of impression

formation in an online distributed software development community with social media functionality, GitHub, in which they analyzed cues that drove users to form impression about other users. Their research catered around three main categories of impressions which were: general coding ability which included cues such as amount of activity, frequency of commits, number of projects owned vs forked etc, project-relevant skills which included cues like specific languages used etc, and finally personality and interaction style which included cues like past discussion posts and threads. They found that amount of commits done by a user had an impact on the impression formation about the coding ability of the user but can be incorrect in judging complete newcomers. They found that coding languages was an important factor in impression formation in terms of skills of a user. Also, they found that communication activity visible in a persons recent activity feed was good determinant of the personality of the user.

Vasilescu et al. [7] investigated the interplay between Stack Overflow activities and the development process, reflected by code changes committed to the largest social coding repository, GitHub. In their research, they observed that individuals who tend to answer many questions tend to have a high number of commits, and individuals that have a high number of commits tend to answer many questions. This was a very helpful observation because it tells us that people who have high number of commits are problem solvers and tend to help others by answering to their queries. That may be useful to through up light on the character of such developers.

Xavier et al. [16] did research to understand which factors can influence developers popularity and provide insights for individuals to enhance their own popularity. In their study they found that commit activity is an important factor for high popularity. But they also observed that some of the developers who had low activity in terms of commits, did have high number of followers and concluded that activities outside GitHub, like on blogs or social networks, might also be the reason for high popularity inside GitHub.

Tsay et al. [6] analyzed GitHub repositories to investigate what important implications do discussions have on project management in terms of new contributions and evolution of project. They observed that a project manager might weigh the cost of fixing a contribution against the benefit of recruiting new member to a project. So fixing issues was found to play an important role in project evolution. They also found that engaging in discussions and to socializing is a very important aspect of team building.

Bissyande et al. [20] investigated the issue trackers of around 100,000 repositories to find if any correlation existed between the amount of issues generated and various characteristics of a software project like age of project, size of the team, popularity of the project leader, lines of code written etc. In the research, they had identified issue reporting as one of the key features of collaborative environment as it helps in reporting existing bugs as well as suggesting feature enhancements that can be done as part of the project.

Badashian et al. [1] in their research got interested in the notion of influence in Soft- ware Social Networks and conducted an in-depth comparison of three influence metrics, number of followers, number of forked projects, and number of project watchers in GitHub. They analyzed the pairwise correlation of the three measures and then did an in-depth qualitative analysis of the similarities and differences of the top 30 influential GitHub users according to each of the three measures.The conclusion they drove from the research was that while number of followers is sign of popularity and fame in software engineering, number of forked projects is sign of code usability and content value.

Dabbish et al [15] in their research examined the value of transparency for large-scale distributed collaborations and communities of practice. In their study, they found the comments on a commit or issue as an interesting signal, where in case of an issue, it highlighted how significant the issue is and in case of commit, it highlighted how important a

commit is. They also noted that number of followers of a developer was interpreted as a signal of status in the community. Developers with lots of followers were treated as local celebrities.

Blincoea et al. [29] studied the motivation behind following (or not following) others and the influence of popular users on their followers. They used a mixed method approach including a survey of 800 GitHub users to uncover the reasons for following on GitHub and a complementary quantitative analysis of the activity of GitHub users to examine influence. Their quantitative analysis studied 199 popular (most followed) users and their followers. From the research, they found that as users popularity increases, so does their rate of influence, yet the same was not true for a popular users rate of contribution.

2.2.2 Conclusion from research on GitHub

Since past few years, GitHub has been a hot area of research and we saw, in the previous section, the kind of research done in this field. Keeping the research question RQ1 in mind, from our exploration we were able to reach the following conclusions :

1. There has not been much research done in the area of skill extraction or rating of skills of a software engineer from his GitHub contributions.

2. A lot of researches show that GitHub can not only be used as a platform for technical contributions but also as a platform for analyzing and enhancing soft-skills.

3. We did find some of the influential factors in determining the performance of the developer on GitHub such as amount of code written, the number and frequency of commits done, number of issues raised or solved, number of owned or watched repositories and contributions in those.

4. We also found that developers with more number of commits and other contributions have large number of followers and are very popular on this platform.

2.3 Requirement Extraction from Job Advertisement

Post our research on GitHub, the next part of our research dealt with extraction of skills from the job description. So our initial exploration in this area was to study the work that had been done in the field of text mining and information extraction. Since a job advertisement is generally written in human readable language, natural language processing was the most prominent technique that came up.

2.3.1 Natural Language Processing

As per Wikipedia [31], Natural Language Processing (NLP) is a field of computer science, artificial intelligence and computational linguistics concerned with interactions between computers and human (natural) languages. In particular, it is concerned with programming computers to fruitfully understand large natural language corpus. There are 2 components of NLP [53]:

1. Natural Language Understanding (NLU) : this deals with mapping the given input in natural language into useful representations and analyzing different aspects of the language.

2. Natural Language Generation (NLG) : it is a process of producing meaningful phrases and sentences in the form of natural language from some internal representation.

For our research goal, we are more interested in NLU than in NLG. As per Chopra et al. [27], 5 phases involved in NLP are:

1. Morphological and Lexical Analysis : The lexicon of a language is its vocabulary that includes its words and expressions. Morphology depicts analyzing, identifying and description of structure of words. Lexical analysis involves dividing a text into paragraphs, words and the sentences.

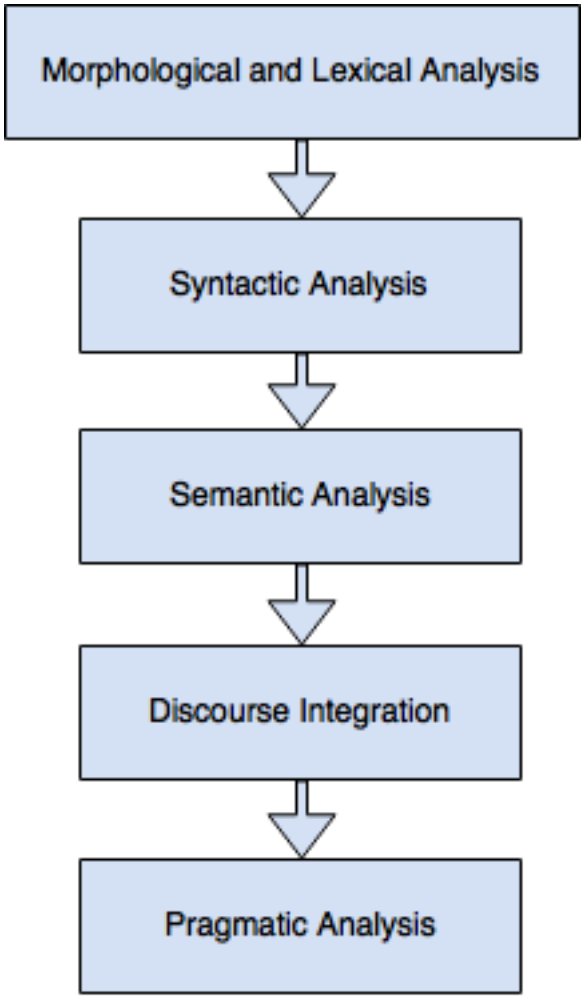


Figure 2.1: Phases of NLP

2. Syntactic Analysis : This involves analyzing words in a sentence to depict the grammatical structure of the sentence. The words are transformed into structure that shows how the words are related to each other. eg. the girl the go to the school. This would definitely be rejected by the English syntactic analyzer.

3. Semantic Analysis : This abstracts the dictionary meaning or the exact meaning from context. The structures which are created by the syntactic analyzer are assigned meaning. There is a mapping between the syntactic structures and the objects in task domain.

4. Discourse Integration : The meaning of any single sentence depends upon the sentences that precedes it and also invokes the meaning of the sentences that follow it.

5. Pragmatic Analysis : It deals with using and understanding sentences in different situations and how the interpretation of the sentence is affected.

2.3.2 Research on Text Mining and Information Extraction Systems

In past few years, Natural Language Processing has emerged as a hot area for researchers in the field of text mining and information extraction. Kumbhar et al. [22] researched that the reputation reporting system of e-commerce platform, which computes the reputation score for a seller, had issues. They proposed an approach called Comm Trust which evaluates the multidimensional trust for seller by analyzing buyers opinions on free text feedback comments. For building this solution, they used standard library named Stanford NLP Parser to pre-process and parse the comments, which was used to represent a sentence as a set of dependency relations between pairs of words in the form of (head, dependent) expressions. These grammatical relationships helped them do further analysis on the opinions.

Hauff et al. [5] in their research, proposed a pipeline that automated the process of matching job advertisement to developers. They used social coding platform, GitHub for extracting concepts (skills) of developers, and NER based model to extract concepts from job advertisements, weighing those concepts and then matching them. They used two techniques from natural language processing namely, NER in combination with named entity disambiguation (NED). Given an unstructured text, NER determined which words or phrases in the text referred to an entity, which can be any real-world entity. NED determined which concrete entity a particular word or phrase refers to. Also, to weigh these concepts, they used a scheme commonly used for information retrieval, TF-IDF. TF or Term Frequency is the frequency of a given term in a document whereas IDF or Inverse Document Frequency is an inverse function of the number of documents in which a term occurs. TF-IDF concept gives a low weight to concepts that appear in many documents while at the same time benefiting concepts which occur rarely across the entire corpus of documents. The research done and the solution developed by Hauff et al. closely aligns with what we intend to do. But the approach they had taken may have some drawbacks. For instance, the weighing of the concepts simply depends on the concept of TF-IDF. So if there are two job advertisements one asking for java developers with 5 years of experience and other asking for java developers freshly passed out from college, then their system will weigh the concept java same in both the cases. Also for extracting skills of developers, they only used README files with same TF-IDF approach as for job advertisements. Thus, they had not considered the contributions of the developer over GitHub while rating ones skills, which we have found to be crucial in our study on GitHub.

Zhang et al. [21] in their research proposed a rule based NLP approach for overcoming the existing challenges in automated information extraction from construction regulatory documents. They stated that in comparison to general non-technical text, domain-specific text is more suitable for automated NLP due to two main text characteristics : firstly,

construction text is likely to have less homonym conflicts than non-technical text and secondly, it is easier to develop an ontology (knowledge base) that captures domain knowledge as opposed to an ontology that captures general knowledge. This is important observation for our study as we aim at extracting domain specific information which revolves around software engineering. They also highlighted two main types of approaches taken in NLP: rule-based approach, and machine learning (ML)-based approach. Rule-based NLP uses manually-coded rules for text processing. ML-based NLP uses ML algorithms for training text processing models based on the text features of a given training text. Rule-based NLP tends to show better text processing performance but requires more human effort. They took rule-based approach, because of its expected higher performance. In their rule-based approach, they captured syntactic features, such as port-of-speech (POS) tags, of the text using various NLP techniques, including tokenization, sentence splitting, morphological analysis, POS tagging, and phrase structure analysis. They extracted semantic features of the text based on an ontology that represents the domain knowledge.

Kalva [26], in his report, proposes the approach taken by him in automating skill extraction from the resume using NLP concepts. For achieving this he used a skill finding algorithm which ranks the student resumes based on skills with respect to a job. He found that most of the skills found are proper nouns (named entities) which can be identified and extracted using the process of Named Entity Recognition (NER). For NER, he used NER APIs provided by apache OpenNLP [34]. These APIs have pre-trained models which can detect named entities like location, time, person, organization, money, percent, data. But to make an NER detect a new entity like skill, a model needs to be trained and the biggest difficulty in training a model is to create training data sufficient enough for statistical modelling. To train the model, Kalva downloaded more than 3000 jobs and 80 resumes from an online web API.

Apart from using NLP techniques for information retrieval, there are other method-

ologies being studied by researchers for extracting important information from plain text. Riloff [24] developed a system called AutoSlog that automatically built a domain-specific dictionary of concepts by extracting information from text. For developing this tool Riloff used a technique called selective concept extraction. Selective concept extraction is a form of text skimming that selectively processes relevant text while effectively ignoring surrounding text that is thought to be irrelevant to the domain. Riloff mentioned that knowledge-based natural language processing systems have good success with tasks that depend on a domain-specific dictionary.

2.3.3 Conclusion from research on Text Mining and Information Extraction Systems

Even though other techniques exist, Natural Language Processing is the most common area of research for overcome information retrieval challenges. Keeping our research question RQ2 in mind and after going through the existing research on text mining and information extraction systems, we reached following conclusions:

1. Automated Natural Language Processing techniques are more suited to domain-specific texts.

2. Rule-based Natural Language Processing approach requires more manual effort than Machine Learning base Natural Language Processing approach.

3. Rule-based Natural Language Processing approach shows better text processing performance than Machine Learning base Natural Language Processing approach.

4. Hauff et al. [5] derived a system which does similar function as what we intend to do in our dissertation, but their study had shortcomings which we have mentioned above and does not consider some parameters, that we have found important in our research.

2.4 Recommending Software Engineers for Job

The final part of our research dealt with providing recommendations of software engineers for the job advertisements. This research was done in two phases. Firstly we studied about recommendation system and various recommendation techniques available and then we explored about the research already done for design and development of systems which provide recommendations for a given job.

2.4.1 Recommendation Systems

According to Wikipedia [32], a recommendation system is a subclass of information filtering system that seeks to predict the "rating" or "preference" that a user would give to an item. According to Aggarwal [18], there are various ways in which the recommendation problem may be formulated, two of which are as follows:

1. Prediction version of problem: The first approach is to predict the rating value for a user-item combination. It is assumed that training data is available, indicating user preferences for items. The missing (or unobserved) values are predicted using a training model. This problem is also referred to as the matrix completion problem.

2. Ranking version of problem: In practice, it is not necessary to predict the ratings of users for specific items in order to make recommendations to users. Rather, a merchant may wish to recommend the top-k items for a particular user, or determine the top-k users to target for a particular item. This problem is also referred to as the top-k recommendation problem, and it is the ranking formulation of the recommendation problem.

Aggarwal [18] also stated in her book that some of the common operational and technical goals of recommender system are :

1. Relevance: The most obvious operational goal of a recommender system is to recommend items that are relevant to the user at hand.

2. Novelty: Recommender systems are truly helpful when the recommended item is something that the user has not seen in the past.

3. Serendipity: A related notion is that of serendipity, wherein the items recommended are somewhat unexpected, and therefore there is a modest element of lucky discovery, as opposed to obvious recommendations. Serendipity is different from novelty in that the recommendations are truly surprising to the user, rather than simply something they did not know about before.

4. Increasing recommendation diversity: Recommender systems typically suggest a list of top-k items. When all these recommended items are very similar, it increases the risk that the user might not like any of these items. On the other hand, when the recommended list contains items of different types, there is a greater chance that the user might like at least one of these items.

Isinkaye et al. [9] explored the different characteristics and potentials of different prediction techniques in recommendation systems in order to serve as a compass for research and practice in the field of recommendation systems. They mentioned three key phases of recommendation process:

1. Information collection phase : This collects relevant information of users to generate a user profile or model for the prediction tasks including users attribute, behaviors or content of the resources the user accesses. A recommendation agent cannot function accurately until the user profile/model has been well constructed. The system needs to know as much as possible from the user in order to provide reasonable recommendation right from the onset. Recommender systems rely on different types of input. Explicit feedback, which includes explicit input by users regarding their interest in item. Implicit feedback by inferring user preferences indirectly through observing user behavior. Hybrid feedback can also be obtained through the combination of both explicit and implicit feedback.

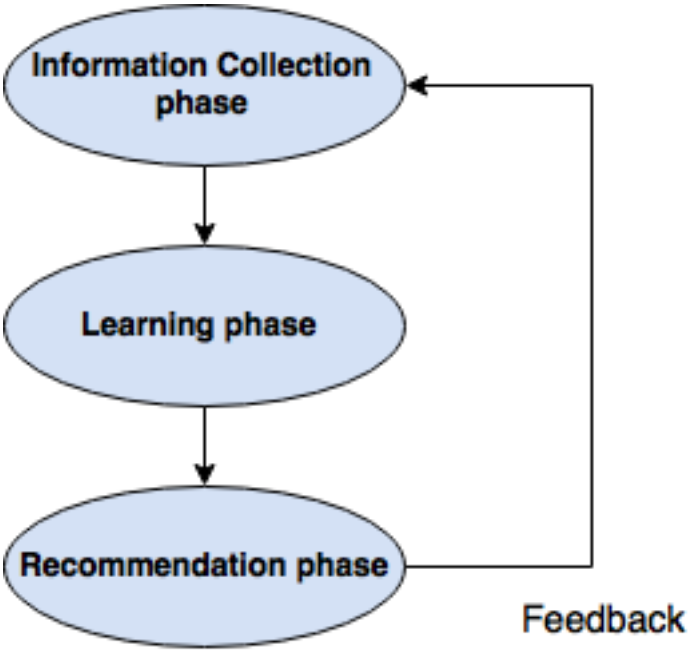


Figure 2.2: Recommendation phases

2. Learning phase : It applies a learning algorithm to filter and exploit the users features from the feedback gathered in information collection phase.

3. Recommendation phase : It recommends or predicts what kind of items the user may prefer. This can be made either directly based on the dataset collected in information collection phase which could be memory based or model based or through the systems observed activities of the user.

2.4.2 Recommendation Techniques

Recommendation Systems are categorized broadly into following three techniques:

1. Content-Based Filtering : According to the handbook [17], Content-based recommendation systems try to recommend items similar to those a given user has liked in the past. Systems implementing a content-based recommendation approach analyze a set of documents and/or descriptions of items previously rated by a user, and build a model or profile of user interests based on the features of the objects rated by that user.The recommendation process basically consists of matching up the at-

tributes of the user profile against the attributes of a content object. The result is a relevance judgment that represents the users level of interest in that object.

2. Collaborative filtering : According to Rafsanjan et al. [12], Collaborative filtering is very effective for forecasting customer precedence in choice of objects. It is an attempt to mechanize word-of-mouth recommendation procedure, that means the objects suggested to customer according to how customers having similar interests, categorized these objects. Collaborative filtering (CF) algorithms are usually categorized into:

    - Memory-based Collaborative Filtering : Another name of the algorithms of Memory-based is lazy recommendation algorithms. They postpone the calculative attempts for forecasting a customers precedence for an object to the time that customers ask for a collection of recommendations.
    - Model-based Collaborative Filtering : Another name of this model is Eager recommendation algorithms, Model-based Collaborative Filtering algorithms do majority of work that is hard in the training stage, where these algorithms build a forecasting model of problem in recommendations.

3. Hybrid Approach : In the research by Asanov [19], hybrid approach is defined as combination of collaborative approaches and content- based approaches.It states that using Hybrid approach, some limitations and problems of pure recommender systems, like the cold-start problem, can be avoided. The combination of approaches can proceed in different ways:

    - Separate implementation of algorithms and joining the results.
    - Utilize some rules of content-based filtering in collaborative approach.
    - Utilize some rules of collaborative filtering in content-based approach.
    - Create a unified recommender system, that brings together both approaches.

2.4.3 Research on Recommendation Systems for a Job

After exploring various recommendation techniques, we studied the work already done on job recommendation systems. Lu et al. [11] presented a hybrid recommendation system approach for job seeking and recruiting websites. Their system exploited job and user profiles and the actions undertaken by users in order to generate personalized recommendations of candidates and jobs. Apart from user profile available on websites, user activities, such as visiting, liking or applying a job post, were used to find the match.

Zarandi et al. [8] proposed an ontology-based hybrid approach to effectively match job seekers and job advertisements. They used job seeker's skills and level of competence in each skill, available on seeker's online application, to find one's suitability for the job. While concluding, they mentioned some very interesting points. They said that their approach relied on self declarations of competences and experiences which could be inaccurate or insufficient.Also, they mentioned that it would be interesting to use mechanisms to automatically discover up- to-date expertise information from secondary sources such as codes, documents, and forums.

Otaibi et al. [10] in their research on job recommendation systems, highlight that the traditional job recommendation systems suffer from inappropriateness because of the Boolean search methods. Such methods use queries containing combination of keywords that define skill requirements and use Boolean expressions (AND-OR) to find matches for job.

There were many other researches which proposed CV based job recommendation systems such as Malinowski et al. [13] proposed a bilateral recommendation approach to build a CV-recommender. YU et al. [14] proposed a Reciprocal Recommendation Algorithm for user-job match for which they used resume and online information available.

2.4.4 Conclusion from research on Recommendation Systems for a Job

Job Recommendation Systems have gained a lot of popularity a the research area in past few years. Keeping research question RQ3 in mind, following were some of the conclusions we reached after going through existing research on recommendation systems :

1. Since we are aiming to find software developers that suit the requirements of a job advertisement, which is analogous to suggesting an item to a user based on his requirements or past ratings, content-based filtering would be the best fit for our recommendation system.

2. Most of the recommendation systems available or popular in this area of research use CV or online information declared by the user to extract user characteristics. Implications of such approach have been mentioned in section 1.1 of this dissertation. The same concerns have been raised by Zarandi et al. [8].

3. Most of the recommendation techniques available cannot be leveraged to design recommendation strategy for recommending developer profile to suit recruiter's skill set requirements.

Chapter 3

Approach

In this chapter, we specify the approach taken by us in order to fulfill the scope of this dissertation. Section 3.1 describes the approach taken by us to identify and rate user skills with the aim of answering RQ1. Section 3.2 describes the information extraction technique used to extract skills from a job description in order to answer the RQ2.

3.1 Finding skills of software engineers

In this section, we first describe the approach taken by us to determine the skills of a software engineer using his GitHub profile. Then we describe the approach for identification of the signals useful for determining the proficiency of software engineer in those skills. Finally, we outline the limitations of our approach.

3.1.1 Approach to find skills

A GitHub user can publish his work over open source platform via repository. A 'repository' or 'repo' in GitHub is a digital directory or storage space where a user can access project, its files, and all the versions of its files. A user can publish his work over a repository which is created by him. He can also publish his contributions to repositories that are created by others, by directly being one of the contributors of the repository or

by issuing a pull request to the contributors of the repository for the changes done by him.

Based on the features provided and web APIs published by GitHub, we selected following two ways of finding out the skills of a software engineer :

1. **Repository Languages** : GitHub provides a feature that highlights language statistics on repository, which provides a way to to quickly see what languages a repository contains. It's a great way to get a general picture of a repository before one dives into the code. The files and directories within a repository determine the languages that make up the repository. GitHub uses the open source Linguist library [37] to determine file languages for syntax highlighting and repository statistics. GitHub provides the following web API [38] to list languages for the specified repository in JSON format. The API also provides the number of bytes of code written in that language.

   $$GET \quad /repos/ : owner/ : repo/languages$$

   As per the Linguist library [37], some files are hard to identify, and sometimes projects contain more library and vendor files than their primary code. For instance, one of the GitHub repository, with majority of code in java, contains some SQL scripts as well. But the API gives just java as the language. Also, with repositories containing XML configuration file, XML is not identified as a language. So to overcome such limitations, we found another way to identify languages for a repository.

2. **File Extensions** : This is the second way to extract languages used in the repository, by parsing through the file names and extracting the file extensions. So, repositories containing '.sql' or '.xml' files tells us that repository also contains files with languages like XML or SQL respectively. GitHub provides following web API

[46] to get the contents of a file or directory in a repository.

$$GET \quad /repos/ : owner/ : repo/contents/ : path$$

Based on the languages of the repositories, created or contributed to by the users, a list of skills, $Skills_{RL}$, is created. Also, based on the file extensions of the contents of the repositories created or contributed to, by the users, another list of skills, $Skills_{FE}$, is created. The union of both these lists is the final list of skills, *Skills*, for the user.

$$Skills = Skills_{RL} \cup Skills_{FE}$$

Using the above methods, skills, based on the repositories, created or contributed to by the users, are extracted for the software engineer.

3.1.2 Approach to find proficiency of skills

Based on the existing research, following important signals were found which might have been helpful to determine the quality of a software engineer :

1. **Content of Commits** : A commit to a repository records the changes to the that repository. These changes can be adding, deleting or updating a file or its contents. Each commit is assigned a unique 40 character string 'sha'. GitHub provides following web API [45] to get the contents of a file or directory in a repository.

   $$GET \quad /repos/ : owner/ : repo/commits/ : sha$$

   The API provides all the files changed in the commit with number of lines changed in each file.

2. **Frequency of Commits** : Frequency of commits is the number of commits done by the user per unit time. This unit can be a day, a week or a month. Frequency

of commits can be calculated from the recent commits done by the GitHub user.

3. **Number of Issues raised and closed** : GitHub provides following web API [40] to fetch list of all the events performed by a user.

$$GET/users/ : username/events$$

One of these events is 'IssuesEvent'. This event has two event types : opened and closed. Counting the number of events for the user with these event types gives us the number of issues raised and closed by the user.

4. **Number of comments made on the Issues** : In the information provided by the Github API for 'IssuesEvent', every issue has a field called 'comment', which is the number of comments for that issue.

5. **Average Length of Comments made on Issues** : Average length of comments made on the issues is average number of words used by a user while writing a comment for the issues. One of the events in GitHub is 'IssueComment'. The body field of the 'IssueComment' event gives the content of the comment made by the user.

6. **Number of repositories created, starred or watched** : GitHub provides following web API for getting list of repositories created, starred or watched by a user.

$$CreatedRepositories : GET/users/ : username/repos$$
$$StarredRepositories : GET/users/ : username/starred$$
$$WatchedRepositories : GET/users/ : username/subscriptions$$

7. **Number of User Followers** : GitHub provides following web API for getting the information about the user. One of the fields in information retrieved from the API

is 'number of followers' which gives the count of people following the user.

$$GET/users/ : username$$

To identify the useful signals from the above mentioned signals, a questionnaire was prepared and sent to the people working in the software industry. The details of the questionnaire is provided in Chapter 5 - Section 5.1. From the results of the questionnaire, 3 signals were identified for rating of the skills of a software engineer :

- **Content of Commits** : To rate the skills of a user based on the contents of commit, number of lines of file changed, in the commits done in every repository owned or starred by the user, is calculated. A score is assigned to the skill based on the number of lines changed in the file belonging to that skill.
- **Number of Issues closed** : To rate the skills of a user based on the issues closed, number of issues closed by a user for every repository owned or starred by the user is calculated. A score is assigned to the repository languages based on the number of issues fixed in that repository.
- **Number of User Followers** : For every user skill identified, a score is assigned based on the number of followers of the user.

The final list of skills contains the skills as well as scores assigned to them and this list represents the complete skill-set of the software engineer.

3.2 Requirement extraction from job advertisement

In this section, we first describe the approach taken by us to extract skills from a job advertisement. Then we describe the approach for for scoring of those extracted skills based on their level of requirement.

3.2.1 Skill extraction from job description

Extraction of skills from a job description requires identifying skills from the given text of requirements in human understandable language, which is a classic information extraction problem. From the research done, Named Entity Recognition (NER), came out as one of the ways of identifying real world entities, which in our case would be skills, from the text given.

While exploring for best possible approach to our information extraction problem, we use Stanford Named Entity Recognizer library, for which there is an online demo [43] also available. We also use spaCy Named Entity Recognizer library [44] for testing purposes. The library is tested with some of the job advertisements to see what percentage of skills are identified. The result for the tests with these libraries are as follows:

- For Stanford NER, more than 60% of the skills are not identified at all, including important skills like java, javascript etc.
- For spaCy NER, about 70% of the skills are identified. But some important skills like core java, are not identified and words like 'quick' are identified as entity, which introduces noise into the library output.

Based on the results received from the testing of these NER libraries, we decide not to go with NER approach for skill identification from given text. So we have built a knowledge-based system for this task. To build the knowledge base, we start looking for existing APIs which provide all the skills a software engineer may have. Unfortunately, to the best of our knowledge,there is no such open-source API available. Therefore, we create our own knowledge base for the skill extraction system. Therefore a file containing all relevant software engineers skill is used as the knowledge base.

Parsing the job description for identifying the available skills is the next task the engine does. This task is done in a 4 stage process :

- First stage is pre-processing of the text. It removes all stop words from the text. Stop words are words like articles, conjunctions, special characters, which we consider as noise in the text.

- Second stage is keyword based matching of the tokens with the knowledge-base of skills.

3.2.2 Scoring extracted skills

Extracted skills from the job description are to be scored based on their level of requirement. More the level of requirement for a skill, the better score it will get. Before finding the score to a skill, we process the text in following 2 steps :

- First step is stemming which is the process of reducing inflected (or sometimes derived) words to their word stem, base or root form. Foe eg. 'cutting' is stemmed to base form 'cut'.

- Second step is Part-of-Speech (POS) tagging of the individual tokens present in the text.

For scoring these required skills, following three criteria are used :

- **Line-Based Score** : while reviewing the requirement descriptions for many jobs, based on human cognitive ability, we find a trend that these descriptions are almost always presented in a set of lines with skills presented in each of these lines. Also, the level of requirement of skill in each line is different. The skills mentioned in the first line of the job description have higher level of requirement than skills present in the below lines. So our first criteria for rating the skills is based on the line on which they occur. The skills present at the top of requirement description are scored more than skills below them.

- **Quantifiable-Experience-Based Score** : the next criteria for scoring the skills is based on the years of experience the software engineer is required to have in those skills. Many job descriptions specify the number of years of experience required for the specified skill which helps us design a new scoring criteria. More the experience specified for a set of skills, better score they will get. To identify the amount of experience from a line in requirement description, we build a pattern-matching system, which looks for pre-defined pattern in the text to extract the years of experience specified. The pre-defined patterns are created keeping in mind all possible forms in which the text for years of experience can be written, even with minor grammatical errors. More details about the implementation are provided in sub-section 4.2.2.

- **Special-Keyword-Based Score** : apart from giving the amount of experience required for a skill, companies also specify the level of requirement of a skill with some special keywords, like highly desirable, substantial knowledge or experience etc. So our last criteria is based on identifying some special keywords present in the requirement description. We use the same technique of creating pre-defined patterns to identify these special keywords from the text. Along with these patterns, corresponding scores are also defined so that the score is assigned to the skills with which the pattern is found. More details about the implementation is provided in sub-section 4.2.2.

The final list of skills extracted from the job description along with the scores assigned to them represents the skill-set required for the job.

3.3 Recommending software engineers for job

In this section, we describe the approach taken by us to solve the recommendation problem of finding the software engineers best suited for the job advertisements.

3.3.1 Recommendation technique used

There are many recommendation techniques available to find closest match of items to a given data item. As discussed in subsection 2.4.4, the problem we have in hand is a content-based filtering problem. To get recommendations, we have a list of skills with scores for each software engineer and list of required skills with scores from the job description. To find top n closest recommendations for the job description, we use Euclidean Distance Approach.

Euclidean Distance is the "ordinary" straight-line distance between two points in Euclidean space. In a recommendation system, using Euclidean approach, we plot all the available data points in an n-dimensional space. These n-dimensions represents the parameters using which we represent the points. Now we plot the given test point in the same space and find Euclidean distance of each data point with the test point. The closest n-data points to the test point are the n closest matches to the given problem.

In our case data points are the list of scores assigned to skills for various software engineers, test point is the list of scores assigned to required skills extracted from job description and closest n-data points are the n-recommendations for the job. The list of n-recommendations is then sorted based on their distance from the test point to get the closest match at the top of the list.

Figure 3.1 represents a set of data point and a test point, in an 2-dimensional plane, with d1 and d2 being the 2 dimensions. The points represented inside the circle are the n-closest matching data-points to the test point.

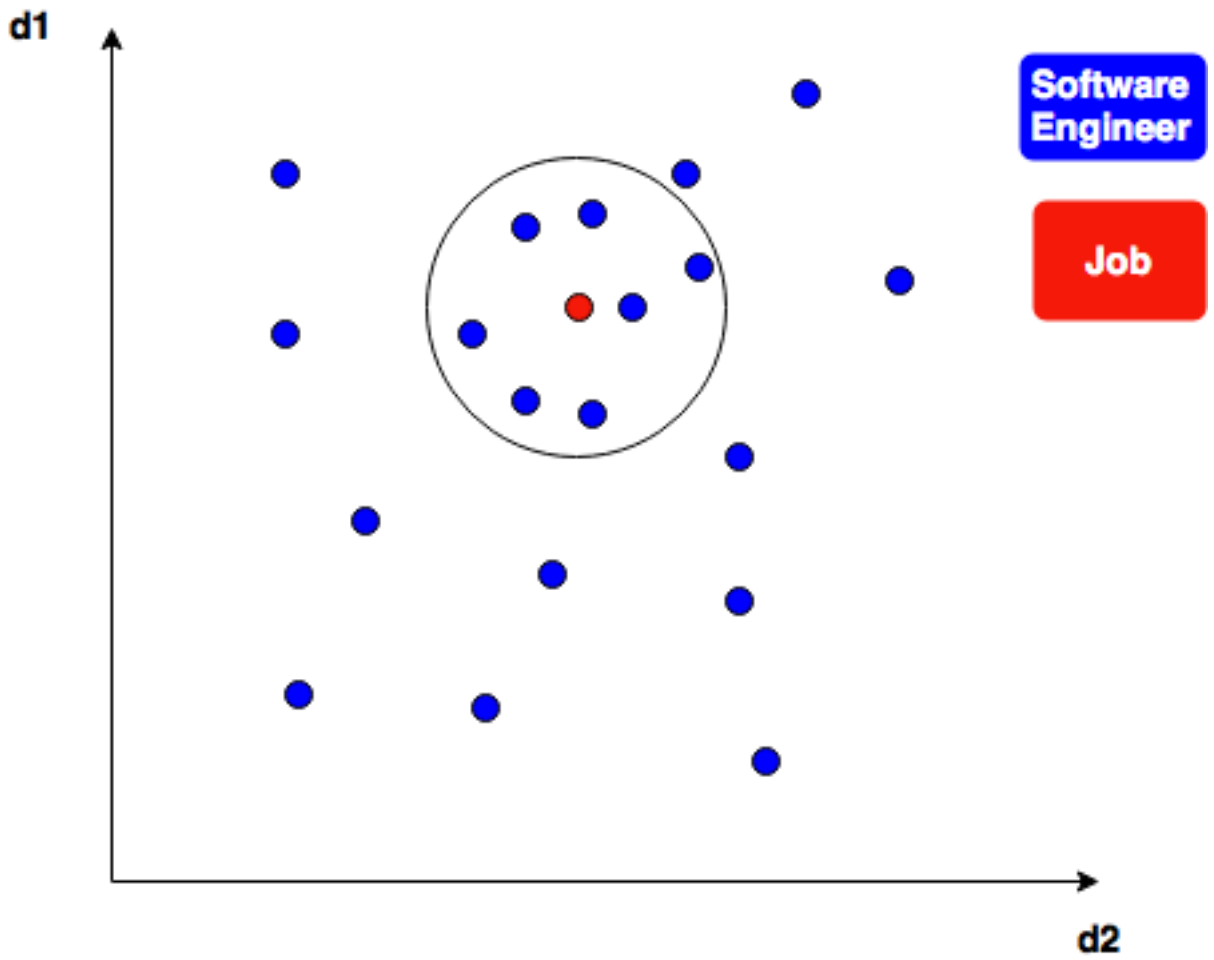


Figure 3.1: Euclidean Distance in 2-Dimensional space

Chapter 4

Implementation

In this chapter, we specify the implementation details to the approach defined in Chapter 3. Section 4.1 describes the components involved and their interactions such that the system generates developer profiles containing rated skills of the developer. Section 4.2 describes various components involved in the requirement extraction from a job description. Finally, section 4.3 describes the components in the recommendation system. At the end of each section we highlight provide the limitations in the approach taken.

4.1 Developer Profile Generator

4.1.1 Components Involved

Figure 4.1 shows the various components involved in generating a user profile and the flow of data through these components. This entire system is built using Java as the programming language with Hibernate framework as the 'Object-Relational Mapping' technology and MySQL as the relational database system to store the data for the users. The system exposes a RESTful Web Service which provides the list of user profiles over the web in JSON format.

Following are the components in detail :

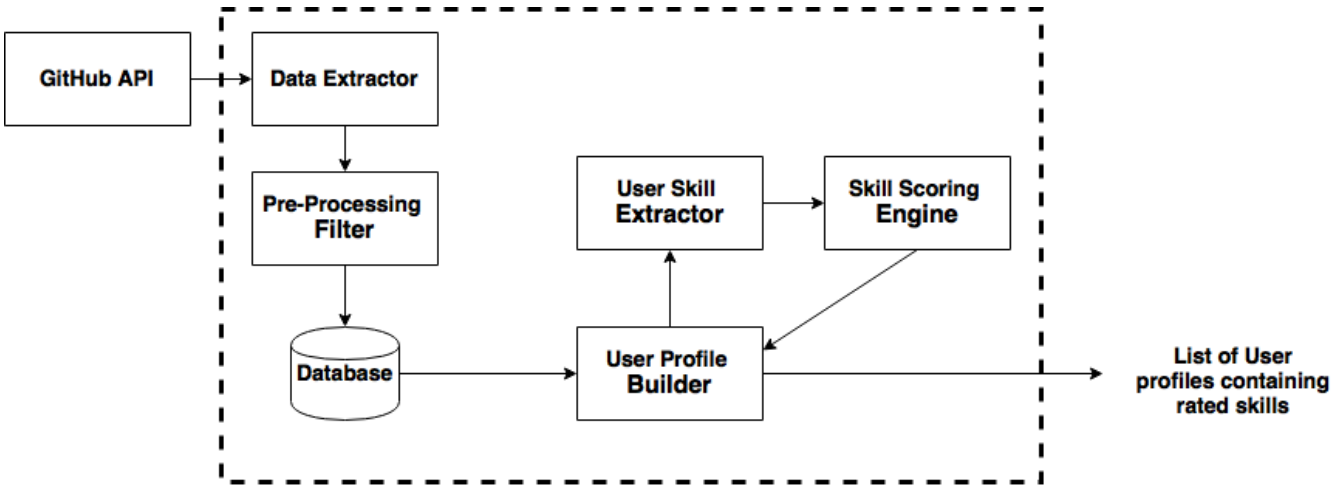


Figure 4.1: Components Interaction for Developer Profile Generator

- **GitHub API** : This component is an external component available over web. As discussed in section 3.1, GitHub provides a way for the developers to access the public data for development purpose. All requests for data from GitHub API are done through the URI 'https://api.github.com'. We use the 'Version 3' of the GitHub API available for developers. There are limits imposed by GitHub on the amount of data that can be fetched over a a period of time. The data over these APIs is available in JSON format. The output from this component if a JSON string of data provided by GitHub.

- **Data Extractor** : The job of extracting the data about the user from the GitHub API is done by the *Data Extractor* component. Since GitHub provides RESTful API over HTTP to get the data, java's 'HttpURLConnection' object is created using a URL, over which the data is available. After the setup of the connection, the JSON data is read as a stream using an 'InputStreamReader' object. The string of JSON data is then mapped to a 'JSONObject' available in 'json' library which provides useful APIs to parse a JSON string. The output from this component is an object of type 'JSONObject' containing data pulled form GitHub.

- **Pre-Processing Filter** : The JSON data pulled from GitHub contained a lot of information which is irrelevant to our problem statement like svn-url, merge or rebase

permissions etc for a repository or avatar-url, gists-url, html-url etc for a user. The information we require is the general data of user like username, company, email id and number of followers, plus the push events (which are commits), modifications done in the commits, issues events, issue-comment events for the user and information about repositories created or starred by the user. So, we need a component that filters out the unnecessary information pulled from GitHub and provides only the required information. And that is the functionality of the *Pre-Processing Filter* component. The output from this component is a series of java objects which contain relevant information to the problem in hand.

- **Database** : *Database* stores all the information required about the user to identify the skills ans well as rate those skills based on the contributions done in repositories. As pointed out earlier, we use relational database management system, MySQL, as the database technology. Hibernate framework is used to map the object received from the Pre-Processing Filter to the tables present in the database.

- **User Profile Builder** : After having all the necessary data, the next step is to process the data. The *User Profile Builder* component pulls the data from the database and builds the profile for the users. These profiles contain basic information about the users, plus the contributions done by the users in the open source platform. To identify the skills possessed by a user and the proficiency in those skills, *User Profile Builder* sends the user profiles to the *User Skill Extractor* component.

- **User Skill Extractor** : *User Skill Extractor* receives profiles for all the users from the *User Profile Builder*. As discussed in sub-section 3.1.1, the skills for the users are identified using a couple of steps : repository languages and file extensions. Languages used in repositories created or starred by the user are used to find user skills. But not all the languages are identified by the library used by GitHub. So we move to the next step of identifying skills, which is based on the file extensions of the files present in those repositories. The output from this component is all the

user profiles containing list of skills for each user.

- **Skill Scoring Engine** : To score the proficiency of every skill identified for a user by *User Skill Extractor* component, *Skill Scoring Engine* is used. As discussed in sub-section 3.1.2, three out of the six parameters are identified for rating the skills of the users : content of commits done, number of issues closed by the user and number of followers for the user. In the next section, we discuss the *Skill Scoring Engine* in detail.

4.1.2 User Skill Scoring Engine

Every skill identified for a user is assigned three types of scores. Figure 4.2 shows the three sub-components in the *Skill Scoring Engine* for finding out the proficiency of skills identified for a user.

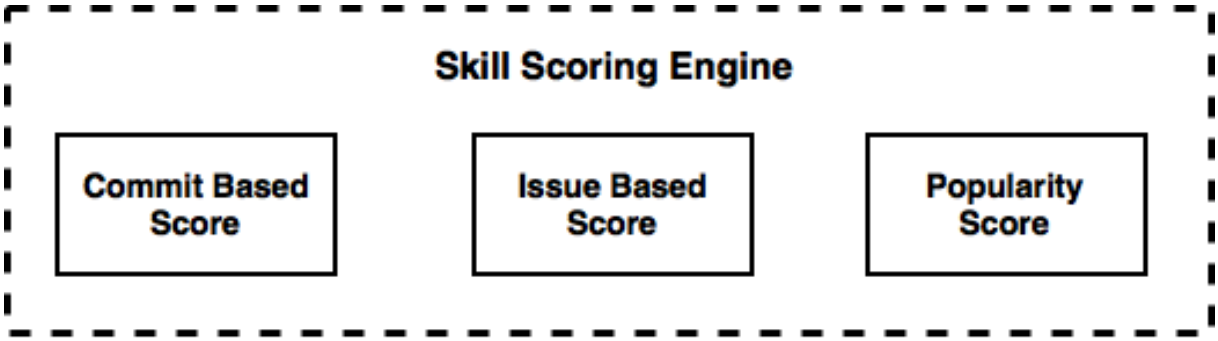


Figure 4.2: User Skill Scoring Engine

- **Commit-Based Score** : *Commit-Based Score* component scores a skill based on the content of the commit done by the user. For scoring based on the content, we consider the lines of code changed in each file in the commit and identify the skill to which the file belongs. The maximum score a skill can get is 10. Let S_{LOC} be the score assigned to the skill based on the lines of code changed by the user in his commits. If lines of code changed by the user for a skill is greater than average lines of code changed for that skill, then the user's skill is assigned more score. Following

is the Pseudo-code for evaluating S_{LOC} for a skill of a user.

Algorithm 1 Pseudo-code for assigning commit-based score

$U_{LOC} \leftarrow LOC\_of\_skill\_changed\_by\_user$
$T_{LOC} \leftarrow Total\_LOC\_of\_skill\_changed$
$U_{Count} \leftarrow Total\_No\_of\_users$

$AVG_{LOC} \leftarrow T_{LOC}/U_{Count}$

$score \leftarrow (U_{LOC}/T_{LOC}) * 10$

if $U_{LOC} < AVG_{LOC}$ **then**
  $S_{LOC} \leftarrow score + 2$
else $\{U_{LOC} >= AVG_{LOC}\}$
  $S_{LOC} \leftarrow score + 5$
end if

if $S_{LOC} >= 10$ **then**
  $S_{LOC} \leftarrow 10$
end if

- **Issue-Based Score** : *Issue-Based Score* component scores a skill based on the number of issues fixed by the user across the repositories in which the user has done contributions. So, if user solves an issue in a repository, then all the skills associated with that repository are assigned score. More the number of issues solved by the user, greater is the score assigned. The maximum score a skills can get is 10. Let S_{IC} be the score assigned to the skill based on the number of issues fixed by the user across repositories. Score, S_{IC}, is calculated for every repository contributed to by the user and is assigned to every skill associated with the repository. If there is a common skill in multiple repositories, then the sum of scores for that skill across all the repositories will be the final score assigned to the skill. If an issue closed by the user has more number of user comments on it, the issue based score assigned would be more. This is based on the understanding that more the number of comments on an issue, more severe is the issue. Following is the Pseudo-code for evaluating S_{IC} for a skill of a user.

Algorithm 2 Pseudo-code for assigning issue-based score

$I_{Count} \leftarrow No\_of\_Issues\_Closed$

for each i *in* I_{Count} **do**
  $S_{IC} \leftarrow S_{IC} + 1$
  $C_{Count} \leftarrow No\_of\_comments\_for\_closed\_issue$

  if $C_{Count} > 3$ **then**
    $S_{IC} \leftarrow S_{IC} + 1$
  end if
end for

if $S_{IC} >= 10$ **then**
  $S_{IC} \leftarrow 10$
end if

- **Popularity Score** : *Issue-Based Score* component does not consider the contributions made by the user but the popularity of the user across the social coding platform. This score is evaluated based on the number of followers of the user and is assigned to every skill of the user. Let $S_{Popularity}$ be the score assigned to the skill based on the popularity of the user. Following is the Pseudo-code for evaluating $S_{Popularity}$ for a skill of a user.

Algorithm 3 Pseudo-code for assigning popularity score

$U_{RANK} \leftarrow User\_rank\_in\_user\_list\_sorted\_by\_No\_of\_followers$
$U_{Count} \leftarrow Total\_No\_of\_users$

$S_{Popularity} \leftarrow [U_{RANK}/U_{Count}] * 10$

The Total Score assigned to a skill of the user,S_{Skill}, is calculated as :

$$S_{Skill} = [0.5 * S_{LOC}] + [0.3 * S_{IC}] + [0.2 * S_{Popularity}]$$

Thus, the final score of skill gets 50% of S_{LOC}, 30% of S_{IC} and 20% $S_{Popularity}$. These percentages are based on the results of the questionnaire to identify signals available on

GitHub for evaluating quality of software engineer, which we have discussed in Chapter 5 - Section 5.1.

4.1.3 Limitations of the approach

Following are some of the limitations of the approach taken for identification of user skills and his proficiency in those skills:

- In this approach, we are just calculating the number of lines as part of content of commit which might not be a very good parameter to measure the quality of changes done.
- We have not considered any soft skill as part the skill-set held by a software engineer. Skills like being a team-player, having leadership qualities might also be important and identifiable from GitHub activities.
- Assigning the score given by the property 'number of followers' to every skill might have introduced discrepancies in the results achieved.
- Currently we are assigning the score based on issues closed by a user in a repository to every language of that repository irrespective of the context of the issues that were solved. A person might have solved all the issues which concerned Java code but the score is assigned to every other language in the repository.

4.2 Skill-Based Requirement Extractor

4.2.1 Components Involved

Figure 4.3 shows various components involved in extracting skills from a job description and the flow of data through these components. This entire system is built using Java as the programming language. The system exposes a RESTful Web Service which provides the list of skills with rating based on level of requirement over the web in JSON format.

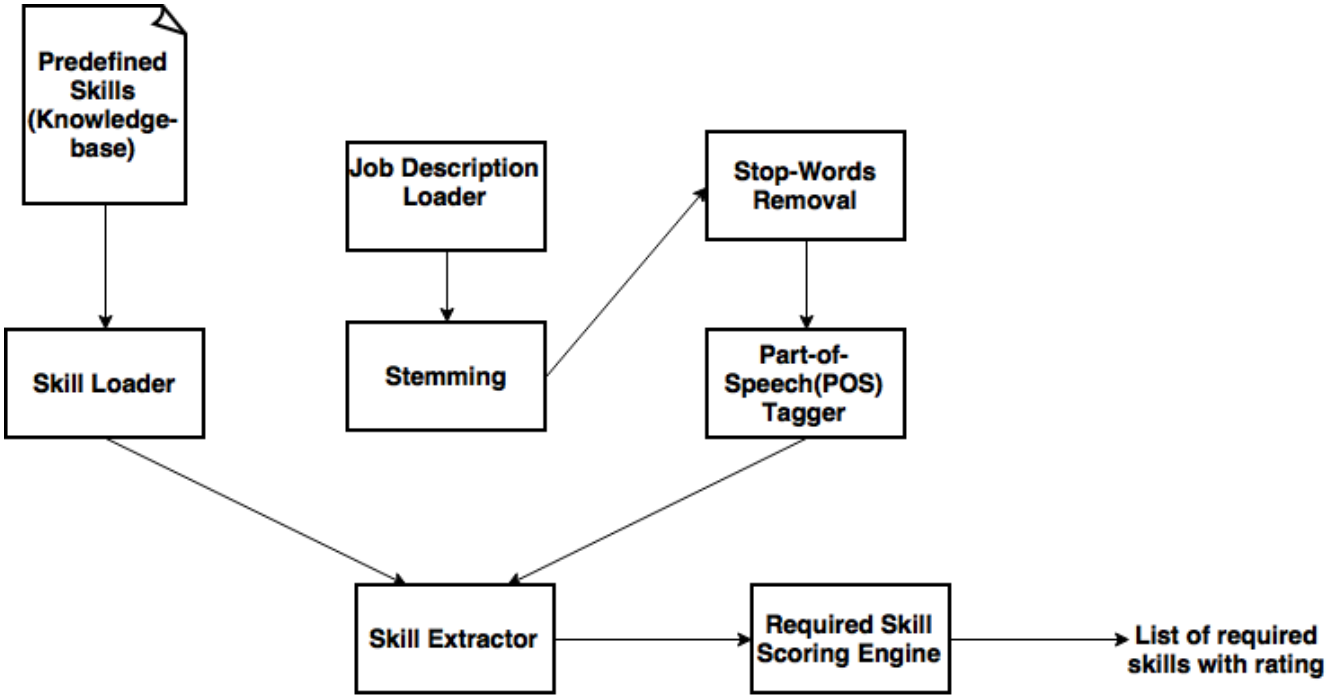


Figure 4.3: Components Interaction for Skill-Based Requirement Extractor

Following are the components in detail :

- **Predefined Skills Knowledge-Base** : This is a separate file which contains the knowledge base of the skills a software engineer may possess. The skills are used for skill extraction from a job description using keyword based matching.

- **Skill Loader** : Before the job description is loaded and processed, pre-defined skills from *Predefined Skills Knowledge-Base* file is loaded into the memory using *Skill Loader* component. The output from this component is a list of pre-defined skills.

- **Job Description Loader** : After loading the knowledge-base of skills, next step is to load the job description into the memory. This function is performed by *Job Description Loader*. The output from this component is a list of separate rows from the job description.

- **Stemming** : The first step after loading the job description into the memory is

stemming each row of job description. Stemming is done to bring every keyword i the text into its root or base form so that further processing of text becomes easier. For stemming, we have used the *Stemmer* class defined by Stanford-CoreNLP Library [42].

- **Stop-Words Removal** : The next step in processing of the job description is the removal of predefined stop-words which is done by *Stop-Words Removal* component. Stop-Words are considered to be noise in the input text. There is a separate file maintained which contains list of stop words which are to be removed from the stemmed text using simple keyword-based matching. Stop-words are removed after stemming is done because this allows us to reduce the amount of predefined stop-words maintained in the separate file. Foe eg. stemmed form of *am*, *is* and *are* is *be*, so we maintain only *be* in the pre-defined stop-words list.

- **Part-of-Speech (POS) Tagger** : The next step of processing is tagging each token in the text with part-of-speech tag. The POS Tagger Component reads the text available in English language and assigns part-of-speech tag to each word available in the text such as noun, verb, adjective etc. We use *Stanford Log-linear Part-Of-Speech Tagger* which is a java based implementation provided for POS Tagger [41]. Following is an example showing the input given and output received to a POS Tagger.

$$InputText : This\ is\ a\ sample\ text$$

$$OutputText : This/DT\ is/VBZ\ a/DT\ sample/NN\ sentence/NN$$

The text returned from the tagger is tagged with part of speech. *NN* represents noun, *VBZ* represents verb, *DT* is determiner. POS tagging of the text is done so that as to create patterns for scoring the identified skills.

- **Skill Extractor** : *Skill Extractor* component receives 2 forms of input. One is the

list of pre-defined skills from *Skill Loader* component, and other is the list of rows of job description with part-of-speech tag from POS Tagger. For identification of skills from the job description, *Skill Loader* does a simple keyword based search. So if *java* is a skill present in job description as well as list of predefined skills, it will be identified as a required skill. For scoring of the extracted required skills, *Skill Loader* sends the part-of-speech tagged list of rows of job description to the *Required Skill Scoring Engine*

- **Required Skill Scoring Engine** : This component assigns the score to each skill identified by the *Required Skill Scoring Engine* based on 3 criteria : the line on which the required skill is present in job description, the amount of experience specified in the job description for the skill and presence of some special keywords in the row in which skill is present. In the next section, we discuss the *Skill Scoring Engine* in detail.

4.2.2 Required Skill Scoring Engine

Figure 4.2 shows the three sub-components in the *Required Skill Scoring Engine* for finding out the level of requirement of the skills identified.

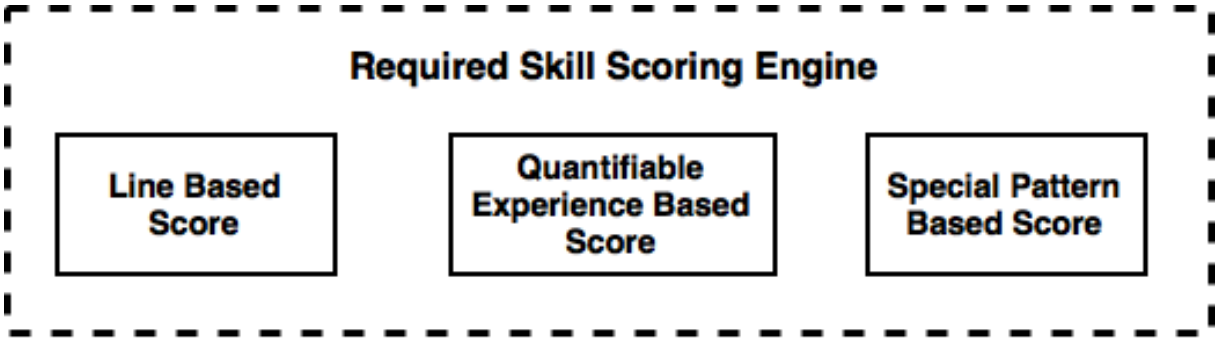


Figure 4.4: Requirement Skill Scoring Engine

- **Line-Based Score** : Line based score is assigned simply based on the line of job description on which the skill is present. Skills are rated more on the line above

than on the line below in text. The maximum line-based score that can be assigned to a skill is 5. Let S_L be the score assigned to a skill based on the line on which it is present in job description. Following is the pseudo-code to find S_L for every skill.

Algorithm 4 Pseudo-code for assigning line-based score

$L_{count} \leftarrow No\_of\_lines$

for each s *in* $skills$ **do**
  $S_{LineNo} \leftarrow Line\_Number\_containing\_Skill(s)$
  $score \leftarrow L_{count} - [S_{LineNo} - 1]$
  $S_L \leftarrow [score * L_{count}] * 5$
end for

- **Quantifiable Experience-Based Score**: To assign Quantifiable Experience based score to a skill, we use Part-Of-Speech tagged list of requirements received from *Skill Extractor*. We defined a list of patterns based on the expected part-of-speech tags from the text. A pattern is defined around a keyword to get better match results. Following is an example pattern defined around keyword 'experience' which matches with the different variations of the text :

  Defined Pattern : **/CD /NNS /IN /NN experience**

  Matched Text : $Having/VBG$ **5/CD years/NNS of/IN work/NN experience**$/VBP$ in/IN $object/RB$ $oriented/VBN$ $technology/NN$

  Matched Text : **5/CD years/NNS of/IN industry/NN experience**$/NN$ in/IN $java/NN$

  In the above example, we can see 2 different types of matched text which have same part-of-speech tag pattern '*/CD /NNS /IN /NN*' around the keyword '*experience*' and match with the defined pattern. This way, all the pre-defined patterns are looked for in the text. If match is found, then the value in the text with POS tag *CD* is extracted. This value is the amount of experience required for the skills. Let

S_{QE} be the score assigned to a skill based on the specified quantifiable experience. S_{QE} is assigned to all the skills present in the same line as the matched pattern. Maximum value of S_{QE} can be 5. Following is the pseudo-code to find S_{QE} for every skill for which predefined experience pattern has been identified.

Algorithm 5 Pseudo-code for assigning quantifiable experience-based score

$QE_{count} \leftarrow Amount\_of\_Experience\_Identified$

if $QE_{count} >= 7$ **then**
  $S_{QE} \leftarrow 5$
else if $QE_{count} >= 5$ **then**
  $S_{QE} \leftarrow 4$
else if $QE_{count} >= 3$ **then**
  $S_{QE} \leftarrow 3.5$
else if $QE_{count} >= 2$ **then**
  $S_{QE} \leftarrow 2.5$
else
  $S_{QE} \leftarrow 1$
end if

- **Special Pattern-Based Score** : Many times in the job description, amount of experience is not mentioned explicitly. Instead it is defined as special terms like substantial experience' or 'vastly experienced'. *Special Pattern-Based Score* assigns score to skills based on such special keywords. Again a predefined list of Part-Of-Speech Tag based patterns are used to identify such special keywords in the text. This predefined pattern list with the corresponding score is maintained as a separate file. This score is assigned to the skills only when no quantifiable experience pattern is found in the line. If S_{SK} is the special keyword based score, then

$$S_{SK} = 0, \quad iff \quad S_{QE} \neq 0$$

Total Score assigned to a required skill, S_{RS}, is calculated as

$$S_{RS} = S_L + S_{QE} + S_{SK}$$

4.2.3 Limitations of the approach

Following are some of the limitations of the approach taken for extraction of skills required for a job and scoring them based on the level of requirement :

- For identification of skills from the job description, we prepare a pre-defined list of skills which a software engineer might hold. This requires a lot of manual effort and since technologies are growing rapidly, the list needs to be updated regularly.
- Also, the patterns and keywords defined for rating the skills requires a lot of human effort and might need to be updated if new patterns are identified.
- Line-Based score, based on the trends seen, assumes that more important skills would be mentioned at the top of job description and less important skills at the bottom. If this assumption fails for some job description, then there would be some error in the scores assigned to the skills. This error is the deviation of the assigned score from the actual score a human would assign to the skill.
- The system scores the skills based on some of the identifiable features from the job description. But there might be some invisible information which the system would miss. For instance, the system identifies two skills S1 and S2 with same rating, but the actual recruiter wants a person with skills S1 more than a person with skill S2. Such cases will result in erroneous output.

4.3 Recommendation System

4.3.1 Components Involved

Figure 4.5 shows the various components involved in finding recommendations for software engineers based on the user profiles and the skills extracted from job description. This entire system is built using Java as the programming language. The system exposes a RESTful Web Service that provides a list of recommended user profiles for a job over the web in JSON format. As described in section 3.3, the recommendation task is performed using Euclidean Distance Approach.

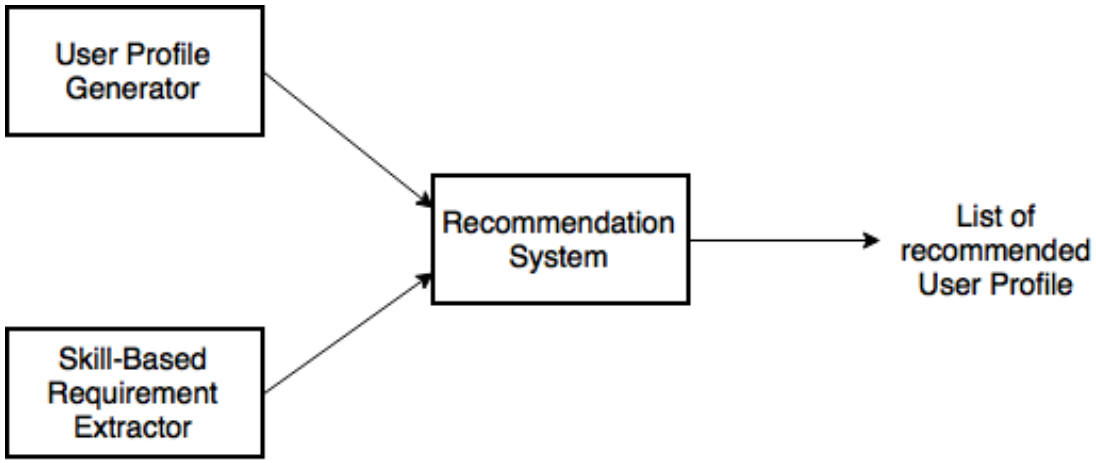


Figure 4.5: Components Interaction for Recommendation System

To provide recommendations, the *Recommendation System* takes as input the output of *User Profile Generator* and *Skill-Based Requirement Extractor*. The output from the *User Profile Generator* is a list of user profiles with each profile containing a list of rated skills for the user. The output from the *Skill-Based Requirement Extractor* is a list of required skills, each rated based on the level of requirement. If there are 'n' data points, which are lists of scores for skills possessed by n users, and there is one test point, which is the list of rated skills for the job, we need to find the Euclidean distance between test point and every data point.

But before finding this distance, the vector space representing the scores of user skills, U_{SV}, has to be in same format as vector space for the skills extracted from the job description, JD_{SV}. This is because the skills which user possess might be different, or in different order, from the skills which the job requires. So we define following algorithm to find a new vector space M_{SV} for every user which contains the same skills of user as the vector space containing skills of job and in the same order , JD_{SV}.

Algorithm 6 Pseudo-code for mapping user skill vector space to job skill vector space

```
JD_SV : List[skill, score]
U_SV : List[skill, score]

JD_SV ← Skill_Vector_for_Job

for each u in UserProfiles do
  U_SV ← Skill_Vector_for_User[u]
  Create New List M_SV : List[skill, score]
  for each j in JD_SV do
    M_SV.addItem(j.skill, 0)
    for each k in U_SV do
      if j.skill = k.skill then
        M_SV.assignScore(j.skill, k.score)
        break
      end if
    end for
  end for
  M_SV is the mapped Skill for User[u]
end for
```

After creating M_{SV} for every user, we calculate the Euclidean Distance between every M_{SV} and JD_{SV}. If m and j are two points in an k-dimensional plane and m = *(m1,m2,...,mk)* and j = *(j1,j2,...,jk)*, then Euclidean distance between m and j is defined as :

$$d(m,j) \;=\; \sqrt{\sum_{i=1}^{k}(\; m_i \;-\; j_i \;)^2}$$

Lastly, user profiles are sorted based on the corresponding Euclidean Distance between M_{SV} and M_{SV}, and the top n user profiles are the top n recommendations for the job.

4.3.2 Limitations of the approach

- Due to the requirement of finding closest matching software engineers to the job description, we have considered Euclidean distance approach. We have not tested the system with other approaches like Cosine-distance or Manhattan-distance which might give better results.

- As pointed out in the sub-section 4.2.3, The system scores the skills based on some of the identifiable features from the job description and some invisible information is not considered. If the invisible information is to be considered, Euclidean distance might not be the best approach.

Chapter 5

Evaluation & Result

In this chapter we describe the procedure we followed to evaluate the approach and present the result. Section 5.1 provides the details of the questionnaire presented to people working in the software industry to identify the signals helpful in determining the quality of a software engineer. The results are also provided in this section. Section 5.2 describes the experimental setup for getting the results. And finally, section 5.3 discusses the procedure and the outcome of the evaluation of the those results.

5.1 Evaluation of quality signals on GitHub

Based on the existing research on the social coding platform, GitHub, six signals are identified which might be helpful in determining the quality of software engineer. Now the task in hand is to shortlist the most important signals which we can use to identify the skills of a software engineer. So, we prepare a questionnaire asking relevant questions regarding these signals, and send it to the people working in the software industry, to understand what signals do people working in the industry identify as important for judging the quality of the software engineer.

Following are the questions asked and the responses received from the people. A

total of 96 people responded to the questionnaire. The distribution of responses is also presented in Appendix B.

- **Do you know the workflow of GitHub (social coding platform)?** : The first question of the questionnaire is to identify if the person answering the questionnaire understands the workflow of GitHub. This is done because most of the questions after this are related to working of GitHub and the intention is to remove noise from the responses received. Out of the 96 responses received, 7 said they were not familiar with the workflow of GitHub and 89 said that they were familiar with the workflow of GitHub.

Question	Yes	No
Do you know the workflow of GitHub (social coding platform)?	89	7

Table 5.1: Questionnaire Responses : Familiarity with GutHub workflow

Now for the rest of questionnaire, we consider only those 89 people who are familiar with the workflow of GitHub.

- **Have you even been a part of technical recruitment process playing the role of recruiter?** : This is asked to identify the kind of people answering the questionnaire. More than half of the people have no experience of being a recruiter in a recruitment process but some have the experience of being a recruiter.

Question	Yes	No
Have you even been a part of technical recruitment process playing the role of recruiter?	27	62

Table 5.2: Questionnaire Responses : Recruitment experience

- **How much would you rate 'content of commits' done by a GitHub user for determining how good a software engineer he is?** : This is the first

question in which we ask about a signal available for a user on GitHub. The signal is 'content of commits'. The options presented for a signal are 'highly relevant', 'relevant', 'not much relevant' and 'no relevance at all'.

Question	Highly relevant	Relevant	Not much	None at all
How much would you rate 'content of commits' done by a GitHub user for determining how good a software engineer he is?	36	42	7	2

Table 5.3: Questionnaire Responses : 'Content of Commits' signal

As can be seen from the data, almost 90% of the people think that this signal is highly relevant or relevant for determining the quality of a software engineer. Hence, this signal is chosen for scoring of the skills of a software engineer.

- **How much would you rate 'frequency of commits' done by a GitHub user for determining how good a software engineer he is?** : another signal identified from the research work done is how frequently does a GitHub user commits. Following are the results of the question.

Question	Highly relevant	Relevant	Not much	None at all
How much would you rate 'frequency of commits' done by a GitHub user for determining how good a software engineer he is?	4	8	51	26

Table 5.4: Questionnaire Responses : 'Frequency of Commits' signal

Results show that more than 80% of the people think that 'frequency of commits' is not a good criteria to judge the work of a software engineer. Hence, we reject this signal in our approach.

- **How much would you rate 'Number of Issues raised' by a GitHub user in a repository for determining how good a software engineer he is?** : Issues raised by a GitHub user is identified as a social characteristic for the user. Following are the results of the question.

Question	Highly relevant	Relevant	Not much	None at all
How much would you rate 'Number of Issues raised' by a GitHub user in a repository for determining how good a software engineer he is?	9	19	40	21

Table 5.5: Questionnaire Responses : 'Number of Issues raised' signal

Results show that more than 60% of the people think that 'number of issues raised' is not a good criteria to judge the work of a software engineer. Hence, we reject this signal in our approach.

- **How much would you rate 'Number of Issues Fixed' by a GitHub user in a repository for determining how good a software engineer he is?** : Issues fixed by a GitHub user is identified as a technical as well as social characteristic for the user. Following are the results of the question.

Question	Highly relevant	Relevant	Not much	None at all
How much would you rate 'Number of Issues Fixed' by a GitHub user in a repository for determining how good a software engineer he is?	22	30	20	15

Table 5.6: Questionnaire Responses : 'Number of Issues Fixed' signal

As can be seen from the data, almost 60% of the people think that this signal is highly relevant or relevant for determining the quality of a software engineer. Hence,

this signal is chosen for scoring of the skills of a software engineer.

- **How much would you rate 'Number of Comments made on Issues' by a GitHub user in a repository for determining how good a software engineer he is?** : another signal identified is the number of comments made on issues by a GitHub user. Following are the results of the question.

Question	Highly relevant	Relevant	Not much	None at all
How much would you rate 'Number of Comments made on Issues' by a GitHub user in a repository for determining how good a software engineer he is?	4	23	39	23

Table 5.7: Questionnaire Responses : 'Number of Comments made on Issues' signal

Results show that more than 60% of the people think that 'number of comments made on issues' is not a good criteria to judge the work of a software engineer. Hence, we reject this signal in our approach.

- **How much would you rate 'Average Length of Comments made on Issues' by a GitHub user in a repository for determining how good a software engineer he is?** : another signal identified is the length of comments made on issues by a GitHub user. Following are the results of the question.

Question	Highly relevant	Relevant	Not much	None at all
How much would you rate 'Average Length of Comments made on Issues' by a GitHub user in a repository for determining how good a software engineer he is?	7	19	42	21

Table 5.8: Questionnaire Responses : 'Average Length of Comments made on Issues' signal

Results show that more than 70% of the people think that 'length of comments made on issues' is not a good criteria to judge the work of a software engineer. Hence, we reject this signal in our approach.

- **How much would you rate 'Number of repositories starred or watched' by a GitHub user for determining how good a software engineer he is?** : another signal identified was the number of repositories starred or watched by a GitHub user. Following are the results of the question.

Question	Highly relevant	Relevant	Not much	None at all
How much would you rate 'Number of repositories starred or watched' by a GitHub user for determining how good a software engineer he is?	10	13	40	26

Table 5.9: Questionnaire Responses : 'Number of repositories starred or watched' signal

Results show that more than 70% of the people think that 'number of repositories starred or watched' is not a good criteria to judge the work of a software engineer. Hence, we reject this signal in our approach.

- **How much would you rate 'Number of repositories created' by a GitHub user for determining how good a software engineer he is?** : another signal identified is the number of repositories created by a GitHub user. Following are the results of the question.

Question	Highly relevant	Relevant	Not much	None at all
How much would you rate 'Number of repositories created' by a GitHub user for determining how good a software engineer he is?	8	18	54	9

Table 5.10: Questionnaire Responses : 'Number of repositories created' signal

Results show that more than 70% of the people think that 'number of repositories created' is not a good criteria to judge the work of a software engineer. Hence, we reject this signal in our approach.

- **How much would you rate 'Number of followers' a GitHub user has for determining how good a software engineer he is?** : In the research work done, number of followers of a user is identified as an influential signal. Following are the results of the question.

Question	Highly relevant	Relevant	Not much	None at all
How much would you rate 'Number of followers' a GitHub user has for determining how good a software engineer he is?	17	25	32	13

Table 5.11: Questionnaire Responses : 'Number of followers' signal

As can be seen from the data, almost 50% of the people think that this signal is highly relevant or relevant for determining the quality of a software engineer. Hence, this signal is chosen for scoring of the skills of a software engineer.

Based on the results from the questionnaire, three signals were identified as helpful in determining the quality of software engineer and were used for determining the proficiency of skills held by the software engineer : Content of commits, Number of issues fixed by

user, number of followers of the user.

5.2 Experimental Setup

As part of experimental setup, we crawl the profiles of 128 GitHub users. All these users are contributors to one of the popular repositories [39]. For these users, we collect basic information, information about the repositories they own or star, the languages used in these repositories, the recent commits done by these users and the content of those commits, the recent issues raised or fixed by these users and comments made on the issues. The amount of data collected is presented in the table below.

Information Type	**Count**
Users	128
Repositories	19370
Commits	2588
Issues	973
Comments	2770

Table 5.12: Experimental Setup

Also, to help evaluate the resulting recommendations of software engineers for a job, we creat a web based user interface, which takes job description as the input and generates the list of top 10 recommendations. For each recommendation, a user profile is displayed which contains all the information collected for the user. This information is provided to help evaluate the results. Based on the information of user, graphs are also displayed on his profile, showing the the commits done with the lines of code changed in those commit and issues fixed with the number of comments on those issues. The screen shots for the user profile can be seen in Appendix C.

5.3 Evaluation of Output

For evaluating the results given by the recommendation system following steps are followed :

- 15 random job descriptions available online are used for the evaluation purpose.
- 15 recruiters, out of the 27 recruiters who were also involved in the questionnaire, are consulted for the evaluation task.
- The recommendations generated by system are evaluated by these recruiters, with each recruiter evaluating results of 3 different job descriptions and each job description being evaluated by 3 different recruiters.
- The evaluation for recommendations generated is done by viewing the user profile over the web user interface of the software engineers and deciding if the recommendations are suitable for the job or not.
- The recruiters are told to assign each job description given to them with one of the following labels post evaluation :
  - Perfectly Matched, if all 10 recommendations match the job requirements.
  - Closely Matched, if 7 or more (but less that 10) recommendations match the job requirements.
  - Some Matched, if 4 or more (but less that 7) recommendations match the job requirements.
  - Less Matched, if less than 4 recommendations match the job requirements.

Following was the result of the evaluation done by the recruiters :

Out of the 45 responses received for 15 job descriptions, 9 are 'Perfectly Matched', 19 are 'Closely Matched', 13 are 'Some Matched' and 4 are 'Less Matched'

Job Description Number	Responses from Recruiters			
	Perfectly Matched (All 10 matched)	Closely Matched (>=7 Matched)	Some Matched (>=4 & <7 Matched)	Less Matched (< 4 Matched)
1	0	2	1	0
2	1	2	0	0
3	1	1	1	0
4	0	0	2	1
5	0	1	1	1
6	1	2	0	0
7	2	1	0	0
8	1	0	2	0
9	1	2	0	0
10	0	1	1	1
11	1	1	1	0
12	0	2	1	0
13	0	1	1	1
14	0	1	2	0
15	1	2	0	0

Figure 5.1: Evaluation of Output

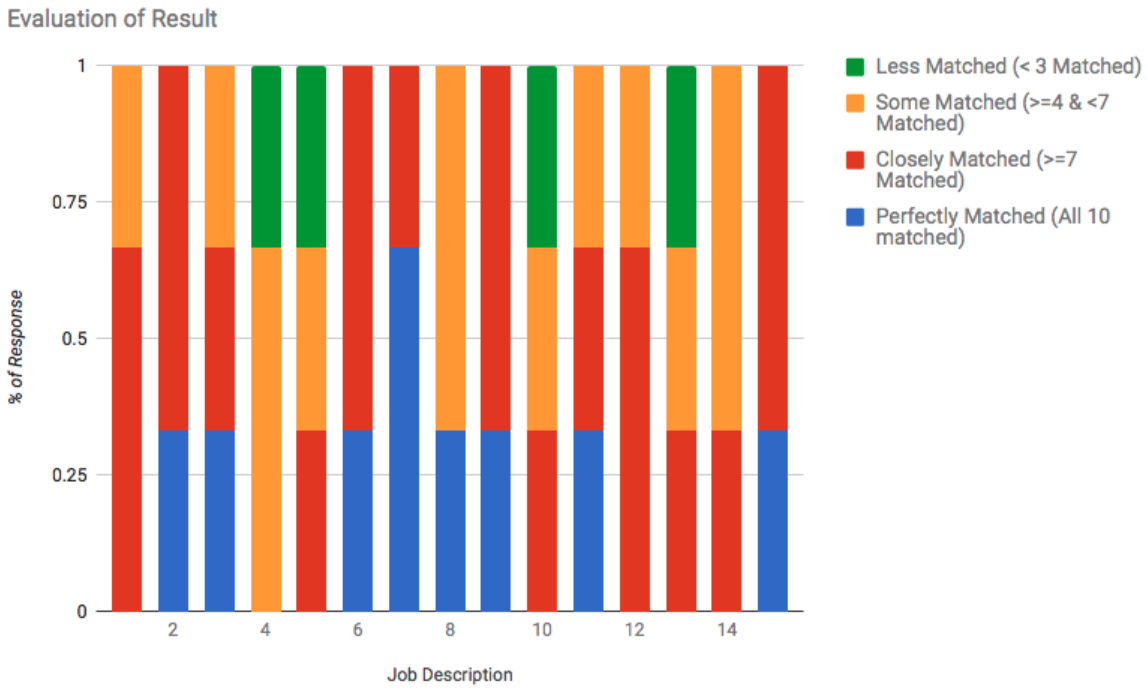


Figure 5.2: Evaluation of Output : Distribution

Chapter 6

Conclusion and Future Work

In this chapter, we provide the conclusions drawn from the research. then we present some ideas for the future work in this area.

6.1 Conclusions

In the first chapter of this dissertation 1.2 we presented three research question. Now we present the answers to those questions :

1. **What signals can be used from the GitHub profile of a developer to automate the process of identification and rating of his skills?** : To find out what signals are important for determining the proficiency of the skills held by a GitHub user, we created a list of properties a GitHub user has and did a questionnaire with industry people to identify which of these properties give indications about the quality of software engineer. Based on the results, we were able to identify three properties which we used in the system we built. But there were some limitations of our approach which we highlighted in the section 4.1.3.

2. **What technique would be suitable to automate the process of identification of required skill from a given job description and rate those skills**

based on their level of requirement? : To find a suitable technique, we investigated the existing research done in those areas, and identified some of the challenges in those approaches. Then we defined a keyword based skill identification approach from a job requirement text. To rate the identified skills based on their level of importance, we highlighted three criteria, used by us in out approach, in the section 3.2.2. Even though we did not do an explicit evaluation of the approach for skills extracted from the job description, the results of the evaluation of recommendation system presented in section 5.3 show that we were able to successfully achieve the research goal. But we did identify some limitations of the approach we had taken and presented it in section 4.2.3

3. **What recommendation strategy would be best suited to match skills identified from developer's GitHub profile to the job specification presented by the recruiter?** : Based on our initial investigation, we identified content-based filtering as the technique for our recommendation problem. For identifying suitable matches of software engineers for a job, we used Euclidean Distance Approach. To evaluate the performance of the recommendation system, we took help of 15 people who have been involved in the recruitment process in the industry. Based on the results of this evaluation presented in section 5.3, only in 2 out of the 15 job description, namely job description 4 and 8, no one identified the recommendations provided by the system as close matches (7 or more matched the job requirements). In 8 out of 15 job description, one or more people rated the recommendations provided by the system as perfect. Only in 4 out of 15 job descriptions, someone identified that less than 4 recommendations matched the job. Based on the results we can say that the approach taken to solve the recommendation problem does solve the problem to some extent. But some of the limitations identified in our recommendation approach are presented in section 4.3.2.

6.2 Future Work

In this section we present the ideas for the future work. As part of finding skills of software engineer, identification of soft skills from user profile of GitHub can be taken up as a further research. Also, we have considered lines of code changed in commit as a signal for rating skills of the user. Research can be done in analysis of the code committed by the user. For instance, complexity of the code committed can be used as a significant factor for identifying how good the developer is. In extraction of skills from a job description, the task was to intelligently identify the critical skills required for a job. Research can be done to check how well does 'Defeasible Reasoning' based approach performs for such systems. For the match-making problem in hand, further research can be done to check how well do techniques like Cosine-Distance and Manhattan-Distance perform.

Appendix A

Abbreviations

Short Term	**Expanded Term**
SHA	Secure Hash Algorithm
NLP	Natural Language Processing
NLG	Natural Language Generation
TF	Term Frequency
IDF	Inverse Document Frequency
ML	Machine Learning
NLU	Natural Language Understanding
API	Application Program Interface
POS	Part Of Speech
JSON	JavaScript Object Notation
NER	Named Entity Recognition

Appendix B

Evaluation of quality signals on GitHub

In this appendix we provide the screen shots related to the questionnaire used by us to identify signals which might be helpful in determining the quality of software engineer.

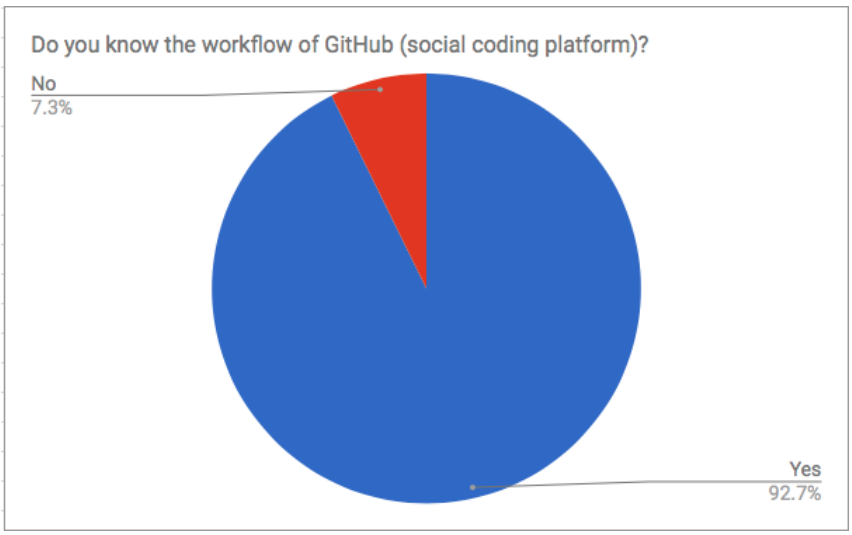


Figure B.1: Quality signals on GitHub : Question 1

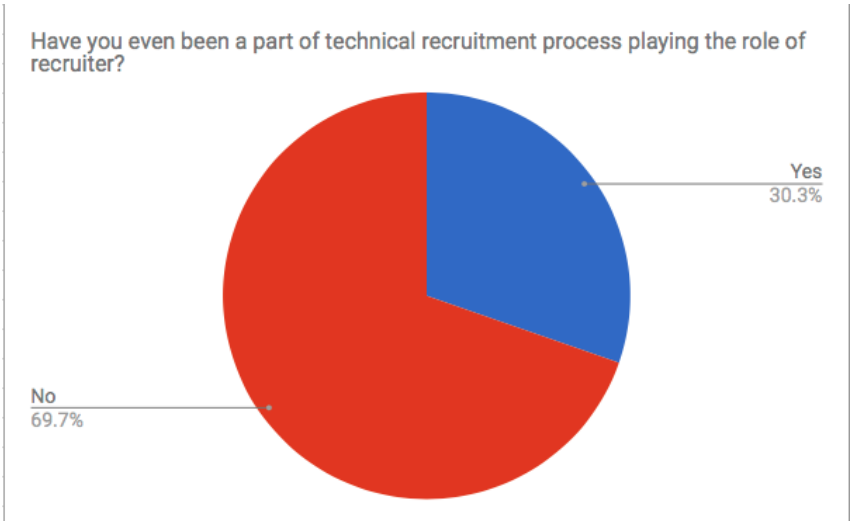


Figure B.2: Quality signals on GitHub : Question 2

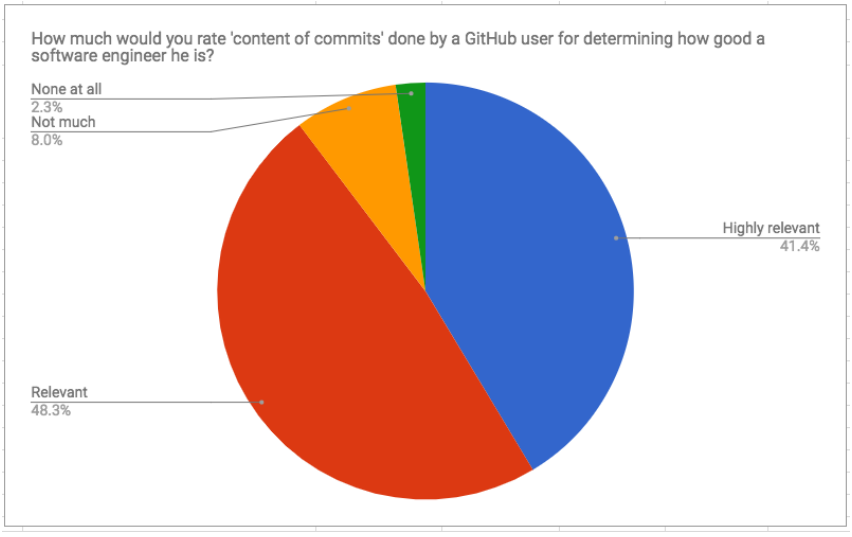


Figure B.3: Quality signals on GitHub : Question 3

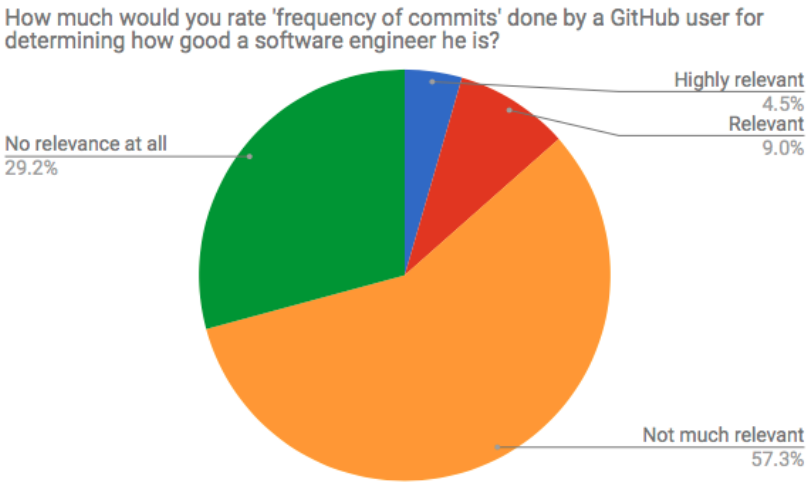


Figure B.4: Quality signals on GitHub : Question 4

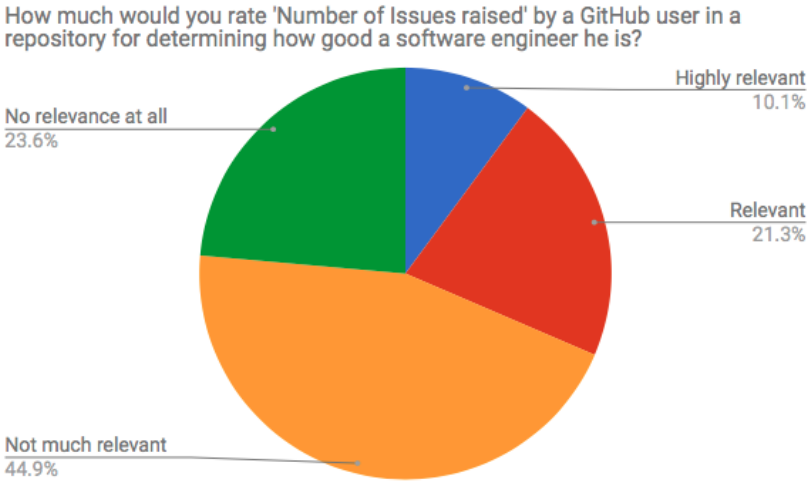


Figure B.5: Quality signals on GitHub : Question 5

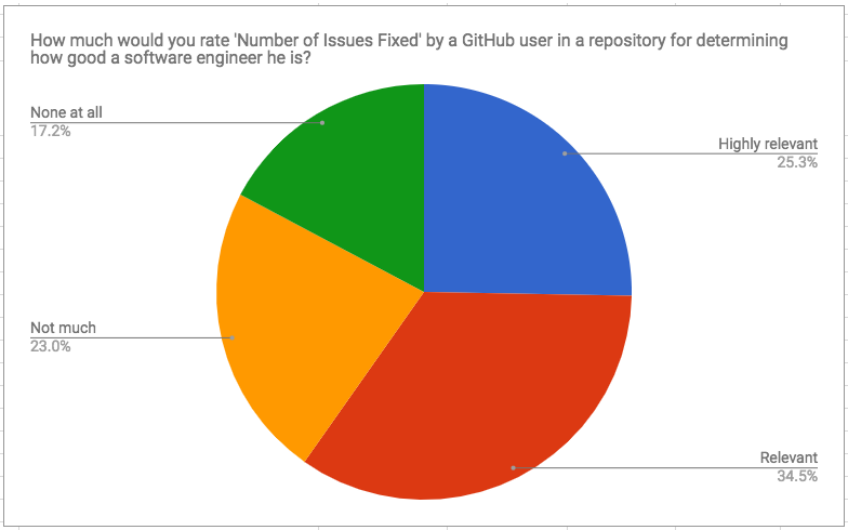


Figure B.6: Quality signals on GitHub : Question 6

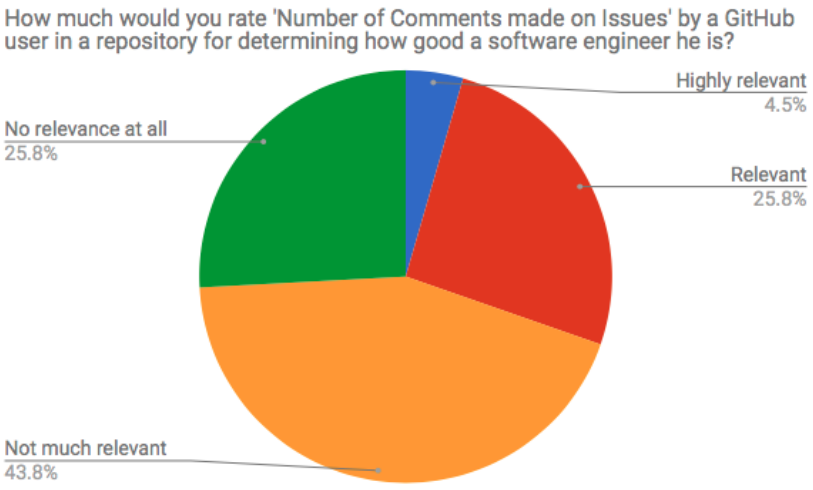


Figure B.7: Quality signals on GitHub : Question 7

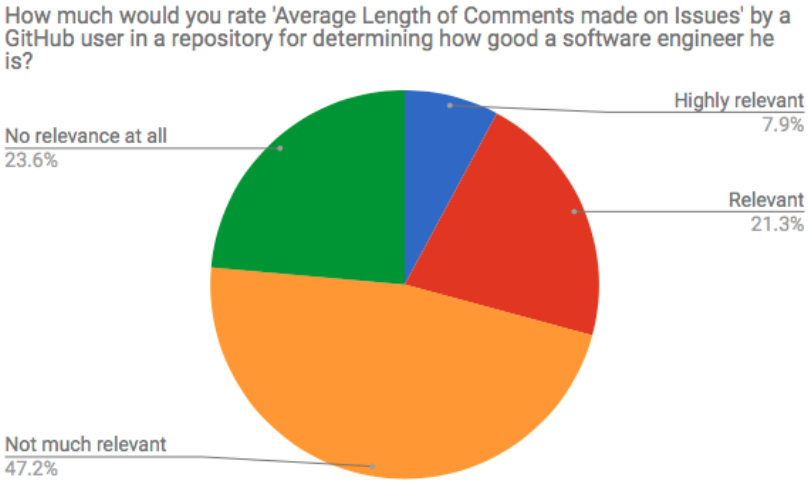


Figure B.8: Quality signals on GitHub : Question 8

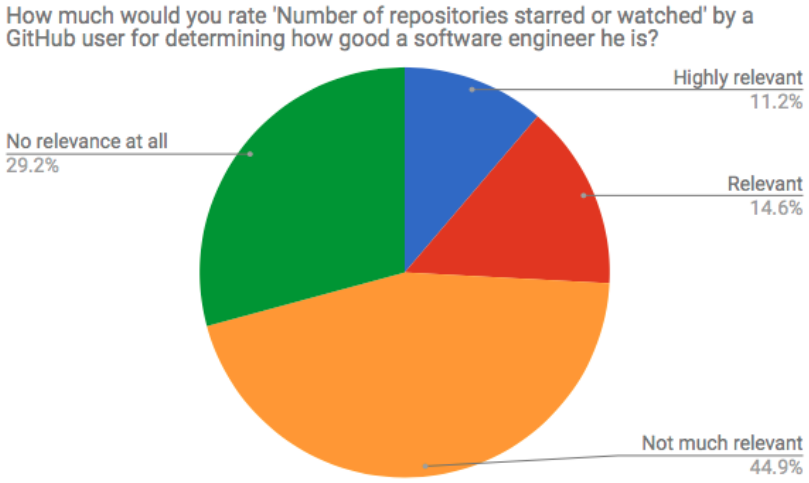


Figure B.9: Quality signals on GitHub : Question 9

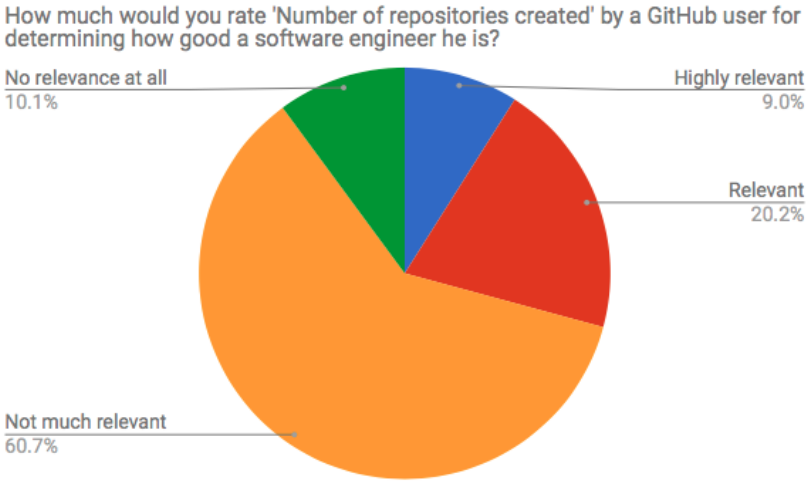


Figure B.10: Quality signals on GitHub : Question 10

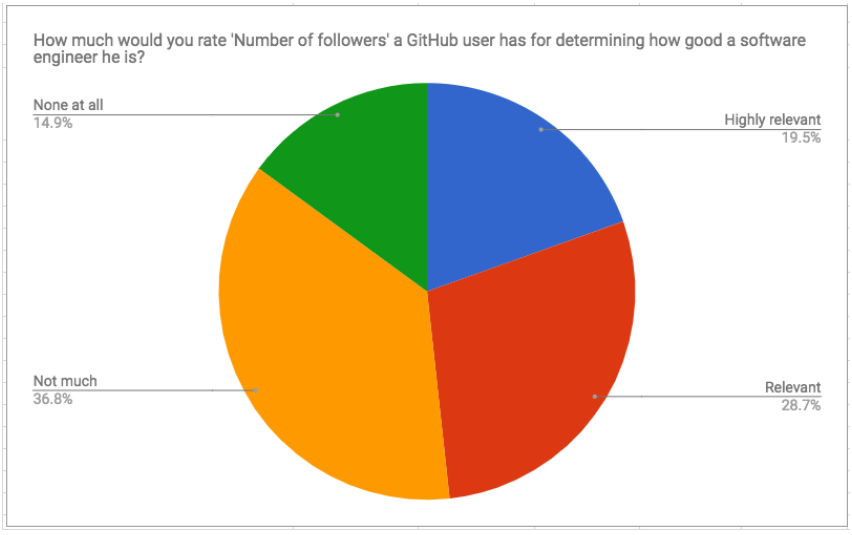


Figure B.11: Quality signals on GitHub : Question 11

Appendix C

User Profile

In this appendix we provide the screen shots related to the user profile presented for the evaluation of the recommendations provided by the system.

Figure C.1: User Interface : User Details

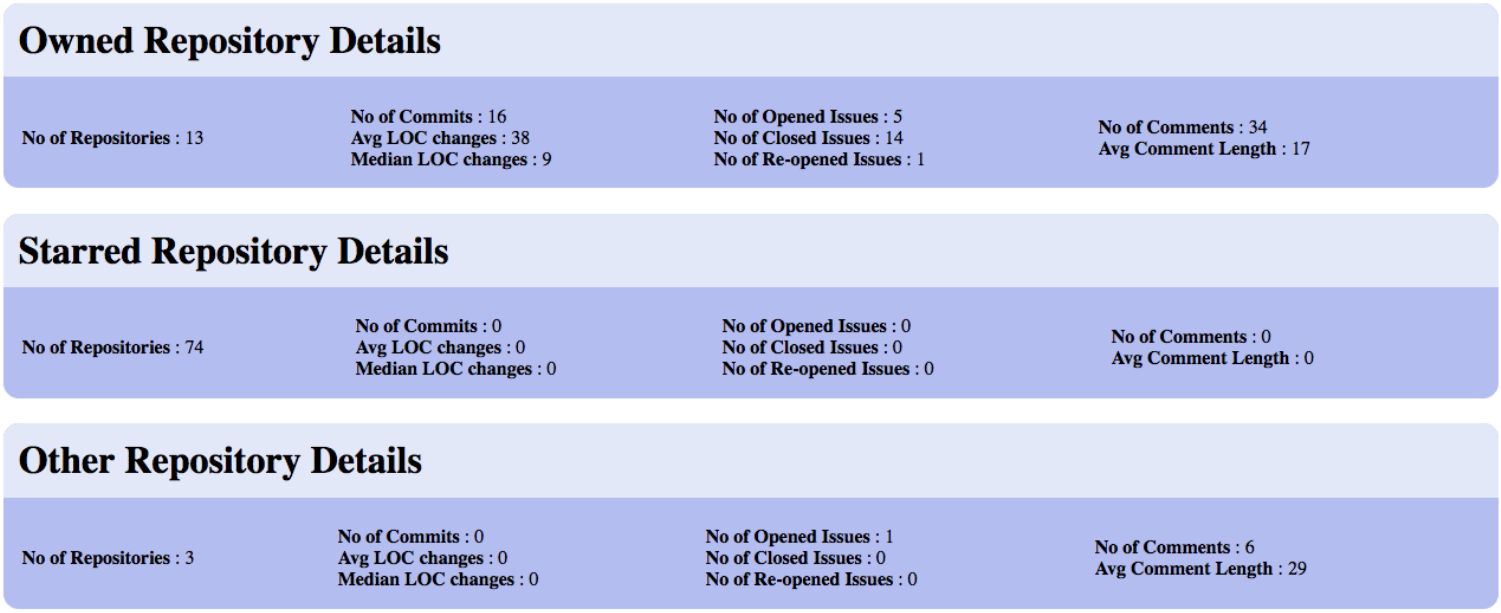


Figure C.2: User Interface : Contributions across Repositories

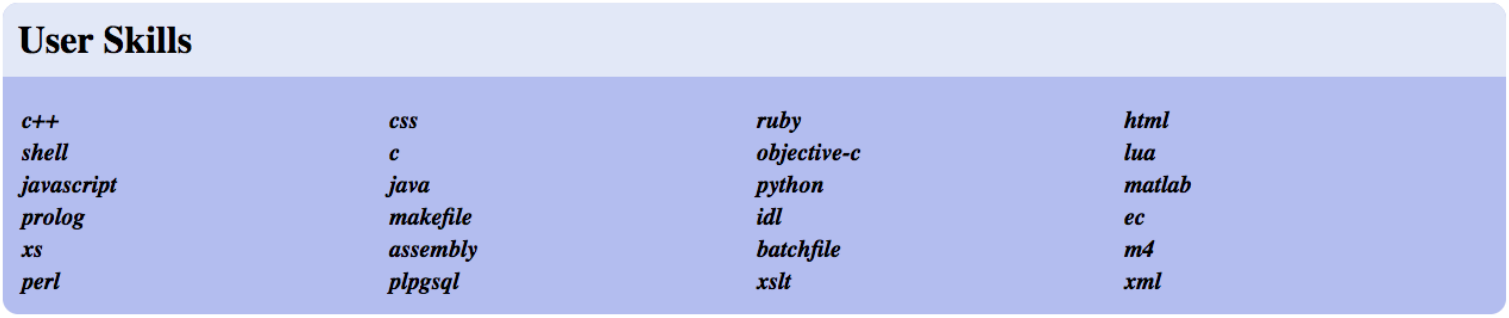


Figure C.3: User Interface : User Skills

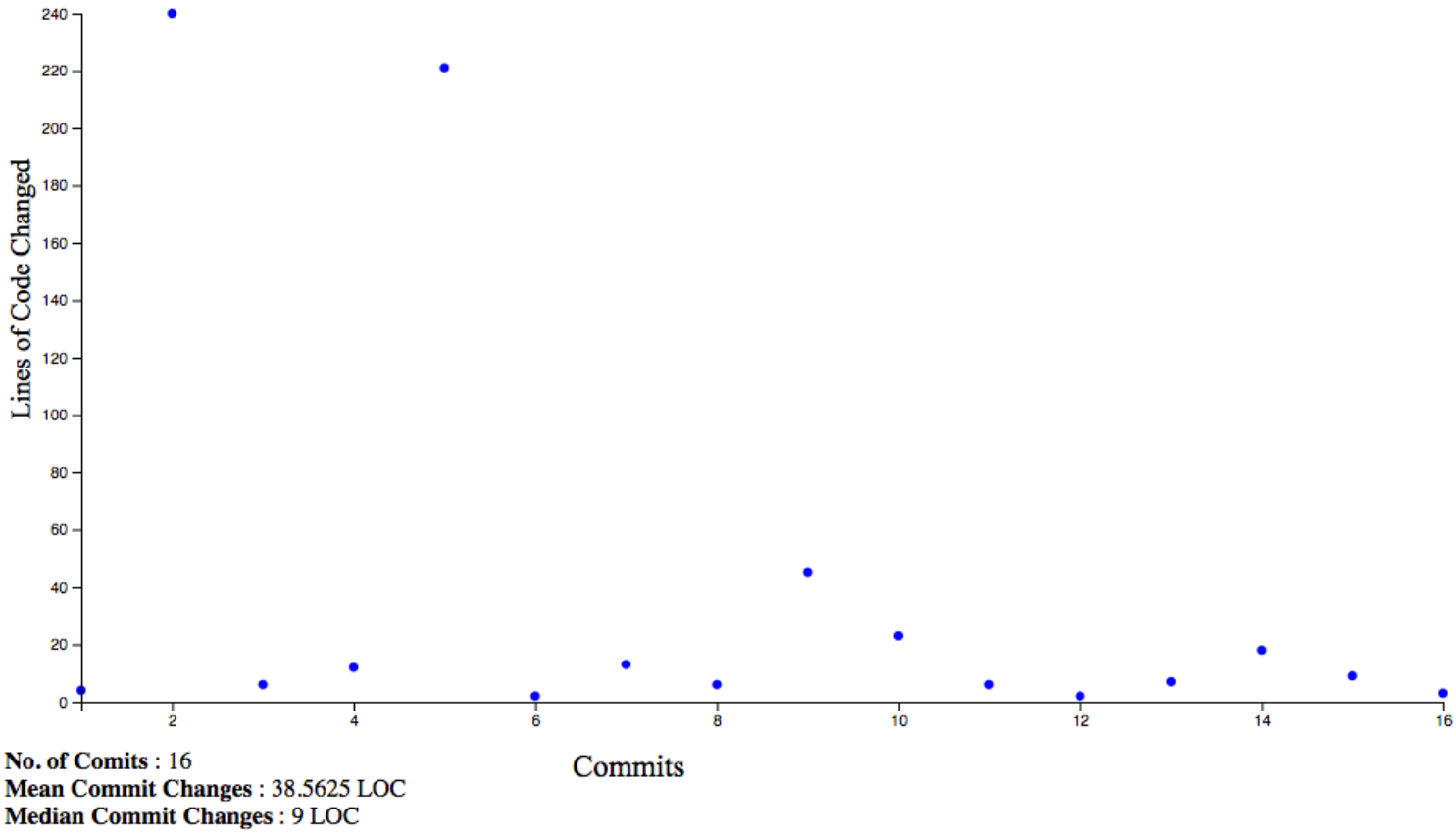
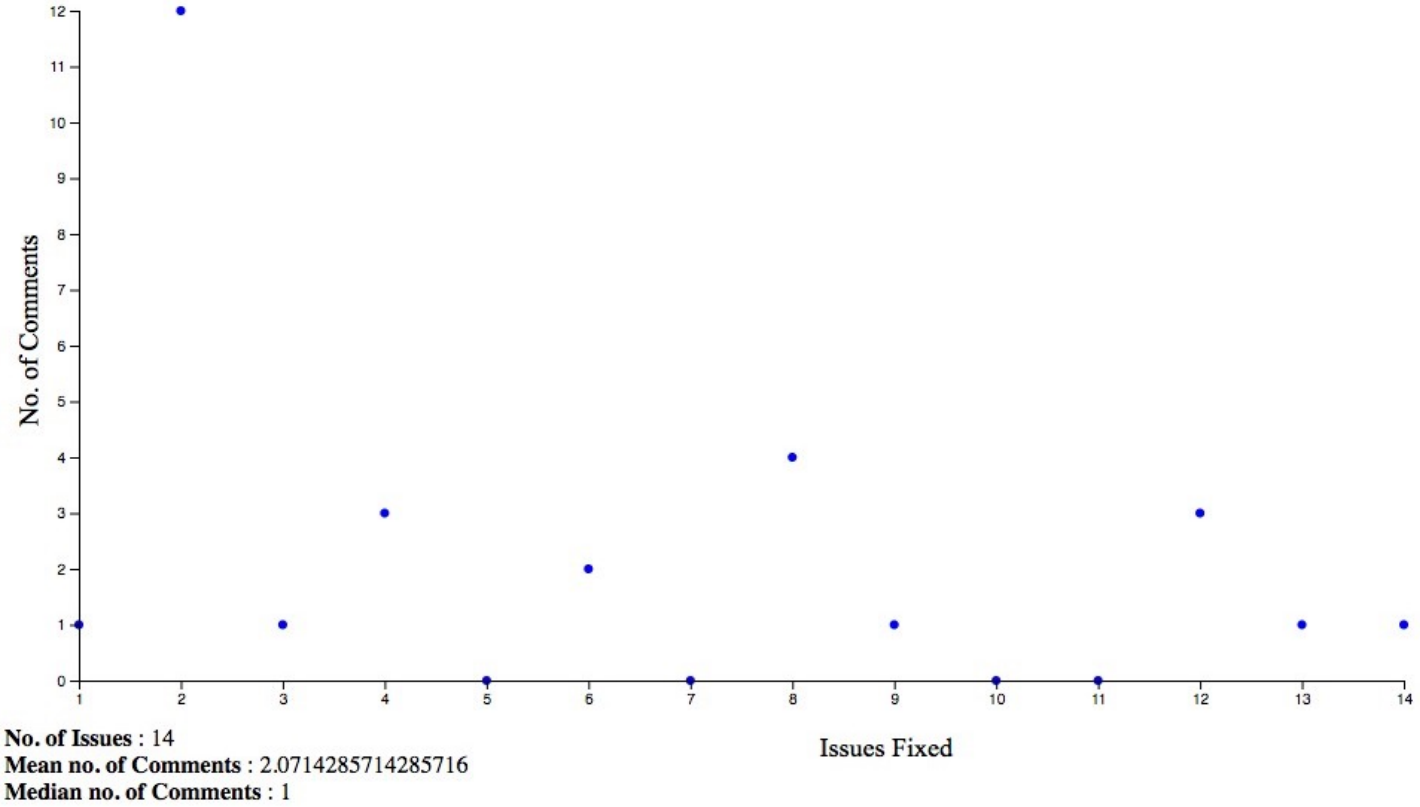


Figure C.4: User Interface : Lines of code changed vs Commits



No. of Issues : 14
Mean no. of Comments : 2.0714285714285716
Median no. of Comments : 1

Figure C.5: User Interface : Comments vs IssuesClosed

Bibliography

[1] Ali Sajedi Badashian, Eleni Stroulia. "Measuring User Influence in Github: The Million Follower Fallacy," in *International Conference of Software Engineering*, Austin, Texas, 2016, pp.15-21.

[2] Jennifer Marlow, Laura Dabbish. "Activity traces and signals in software developer recruitment and hiring, in *Computer Supported Cooperative Work*, San Antonio, Texas, 2013, pp.145-156.

[3] Jennifer Marlow, Laura Dabbish, Jim Herbsleb. "Impression formation in online peer production: activity traces and personal profiles in github, in *Computer Supported Cooperative Work*, San Antonio, Texas, 2013, pp.117-128.

[4] Ali Sajedi Badashian, Afsaneh Esteki, Ameneh Gholipour, Abram Hindle, Eleni Stroulia. "Involvement, Contribution and Influence in GitHub and Stack Overflow, in *Conference of the Center for Advanced Studies on Collaborative Research*, Markham, Ontario, Canada, 2014, pp.19-33.

[5] Claudia Hauff, Georgios Gousios. "Matching GitHub developer profiles to job advertisements, in *International Conference on Software Engineering*, Florence, Italy, 2015, pp.362-366.

[6] Jason Tsay, Laura Dabbish, James Herbsleb. "Let's talk about it: evaluating contributions through discussion in GitHub, in *Foundations of Software Engineering*, Hong Kong, China, 2014, pp.144-154.

[7] Bogdan Vasilescu, Vladimir Filkov, Alexander Serebrenik. "StackOverflow and GitHub: Associations between Software Development and Crowdsourced Knowledge, in *Social Computing Conference*, Alexandria, VA, USA, 2013, doi:10.1109/SocialCom.2013.35.

[8] Maryam Fazel-Zarandi, Mark S. Fox. "Semantic Matchmaking for Job Recruitment: An Ontology-Based Hybrid Approach, in *International Journal of Advanced Computer Science and Applications*, 2016.

[9] F.O. Isinkaye, Y.O. Folajimi, B.A. Ojokoh. "Recommendation systems: Principles, methods and evaluation, in *Egyptian Informatics Journal*, 2015.

[10] Shaha T. Al-Otaibi, Mourad Ykhlef. "Job Recommendation Systems for Enhancing E-recruitment Process," unpublished.

[11] Yao Lu, Sandy El Helou, Denis Gillet. "A Recommender System for Job Seeking and Recruiting Website, in *International World Wide Web Conference*, Rio de Janeiro, Brazil, 2013, pp.963-966

[12] Amir Hossein Nabizadeh Rafsanjani, Naomie Salim, Atae Rezaei Aghdam, Karamollah Bagheri Fard. "Recommendation Systems: a review, in *International Journal of Computational Engineering Research*, Vol 03, Issue 5, 2013.

[13] J. Malinowski, T. Keim, O. Wendt, T. Weitzel. "Matching People and Jobs: A Bilateral Recommendation Approach, in *39th Annual Hawaii International Conference*, Kauia, HI, USA, 2006, doi:10.1109/HICSS.2006.266

[14] Hongtao YU, Chaoran LIU, Fuzhi ZHANG. "Reciprocal Recommendation Algorithm for the Field of Recruitment, in *Journal of Information & Computational Science*, 2011.

[15] Laura Dabbish, Colleen Stuart, Jason Tsay, Jim Herbsleb. "Social Coding in GitHub: Transparency and Collaboration in an Open Software Repository, in

Computer Supported Cooperative Work, Seattle, Washington, USA, 2012, pp.1277-1286

[16] Joicymara Xavier, Autran Macedo, Marcelo de Almeida Maia. "Understanding the popularity of reporters andassignees in the Github, in *International Conference on Software Engineering & Knowledge Engineering*, Vancouver, Canada, 2014.

[17] Francesco Ricci, Lior Rokach, Bracha Shapira, Paul B. Kantor.(2011) "Recommender Systems Handbook," Springer International Publishing

[18] Charu C. Aggarwal.(2016) "Recommender Systems: The Textbook," Springer International Publishing

[19] Daniar Asanov. (2015) "Algorithms and Methods in Recommender Systems"

[20] Tegawende F. BISSYANDE, David LO, Lingxiao JIANG, Laurent REVEILLERE, Jacques KLEIN. "Got issues? Who cares about it? A large scale investigation of issue trackers from GitHub, in *Software Reliability Engineering (ISSRE), IEEE 24th International Symposium*, Pasadena, CA, USA, 2013, doi:10.1109/ISSRE.2013.6698918

[21] Jiansong Zhang, Nora M. El-Gohary. "Semantic NLP-based Information Extraction from Construction Regulatory Documents for Automated Compliance Checking,". Journal of Computing in Civil Engineering, July 2013

[22] Priyanka Kumbhar, Manjushree Mahajan. "Multi-Dimensional Trust Evaluation from Mining of E- Commerce Feedback Comments in *IJCA Proceedings on National Conference on Advances in Computing, Communication and Networking*, 2016.

[23] J.-D. Kim, T. Ohta, Y. Tateisi, J. Tsujii. "GENIA corpusa semantically annotated corpus for bio-textmining," *Bioinformatics*, Volume 19, Issue suppl_1,pp.i180i182.

[24] Ellen Riloff. "Automatically constructing a dictionary for information extraction tasks, in *AAAI'93 Proceedings of the eleventh national conference on Artificial intelligence*, Washington, D.C., 1993, pp.811-816

[25] Eirini Kalliamvakou, Georgios Gousios, Kelly Blincoe. "The Promises and Perils of Mining GitHub, in *International Conference on Software Engineering*, Hyderabad, India, 2014, pp.92-101

[26] Thimma Reddy Kalva. "Skill Finder: Automated Job-Resume Matching System," Utah State University, 2013

[27] Abhimanyu Chopra, Abhinav Prashar, Chandresh Sain. "Natural Language Processing, in *International Journal of Technology Enhancements and Emerging Engineering Research*, Vol 1, Issue 4, 2013.

[28] Ferdian Thung, Tegawend F. Bissyand, David Lo. "Network Structure of Social Coding in GitHub, in *Software Maintenance and Reengineering (CSMR), 17th European Conference on*, Genova, Italy, 2013, doi:10.1109/CSMR.2013.41.

[29] Kelly Blincoe, Jyoti Sheoran, Sean Goggins, Eva Petakovic, Daniela Damian. "Understanding the popular users: Following, affiliation influence and leadership on GitHub," Information and Software Technology Journal, 2015.

[30] "Social recruiting Wikipedia. Retrieved from https://en.wikipedia.org/wiki/Social_recruiting.

[31] "Natural Language Processing-Wikipedia. Retrieved from https://en.wikipedia.org/wiki/Natural_language_processing.

[32] "Recommender System-Wikipedia". Retrieved from https://en.wikipedia.org/wiki/Recommender_system.

[33] "The Jobvite Recruiter Survey - 2015". Retrieved from https://www.jobvite.com/wpcontent/uploads/2015/09/jobvite_recruiter_nation_2015.pdf.

[34] "Apache OpenNLP Developer Documentation". Retrieved from https://opennlp.apache.org/docs/1.6.0/manual/opennlp.html

[35] Patrick McCuller. "How to Recruit and Hire great Software Engineers," Article in Apress.

[36] Fork A Repo - User Documentation, https://help.github.com/articles/fork-a-repo/

[37] github/linguist: Language Savant, https://github.com/github/linguist

[38] Repositories — GitHub Developer Guide, https://developer.github.com/v3/repos

[39] ReactiveX/RxJava, https://github.com/ReactiveX/RxJava

[40] "Events — GitHub Developer Guide". https://developer.github.com/v3/activity/events-/#list-public-events-performed-by-a-user

[41] Stanford Log-linear Part-Of-Speech Tagger, https://nlp.stanford.edu/software-/tagger.shtml

[42] Stanford CoreNLP - Stemmer, https://github.com/stanfordnlp/CoreNLP/blob-/master/src/edu/stanford/nlp/process/Stemmer.java

[43] Stanford Named Entity Tagger, http://nlp.stanford.edu:8080/ner/

[44] spaCy Named Entity Tagger, http://textanalysisonline.com/spacy-named-entity-recognition-ner.

[45] Commits — GitHub Developer Guide, https://developer.github.com/v3/repos/commits/.

[46] Contents — GitHub Developer Guide, https://developer.github.com/v3/repos/contents/

[47] About pull requests - User Documentation, https://help.github.com/articles-/about-pull-requests/

[48] Merging a pull requests - User Documentation, https://help.github.com/articles-/merging-a-pull-request/

[49] Issues - Github Guides, https://guides.github.com/features/issues/

[50] Shortlisting Step-By-Step Guide For Candidate Recruitment,, https://ideal.com/-shortlisting/

[51] Github - About, https://github.com/about

[52] Importance of Soft Skills for Software Engineers, https://medium.com-/@anaida07/importance-of-soft-skills-for-software-engineers-7965be2074dc

[53] Artificial Intelligence Natural Language Processing, https://www.tutorialspoint-.com/artificial_intelligence/artificial_intelligence_natural_language_processing.htm