Understanding and Predicting Software Developer Expertise in Stack Overflow and GitHub

by

Sri Lakshmi Vadlamani

A Thesis submitted to the Faculty of Graduate and Postdoctoral Affairs in partial fulfilment of the requirements for the degree of Master of Computer Science with Specialization in Data Science

Carleton University
Ottawa, Ontario

©2021
Sri Lakshmi Vadlamani
The undersigned recommend to the Faculty of Graduate and Postdoctoral Affairs acceptance of the Thesis

Understanding and Predicting Software Developer Expertise in Stack Overflow and GitHub

Submitted by Sri Lakshmi Vadlamani
in partial fulfilment of the requirements for the degree of

Master of Computer Science

________________________
Dr. Olga Baysal, Thesis Supervisor

________________________
Dr. Alan Tsang, Internal Examiner

________________________
Dr. Omair Shafiq, External Examiner

________________________
Dr. Matthew Holden, Chair of Defence

________________________
Dr. Michel Barbeau, Department Chair

Carleton University

2021
Abstract

Knowledge and experience are touted as both the necessary and sufficient conditions to make a person an expert. This thesis attempts to investigate this issue in the context of software development by studying and predicting software developer’s expertise based on their activity and experience on GitHub and Stack Overflow platforms. The thesis studies how developers themselves define the notion of an “expert”, as well as why or why not developers contribute to online collaborative platforms. An exploratory survey was conducted with 73 software developers and a mixed methods approach was applied to analyze the survey results. The results provided deeper insights into how an expert in the field could be defined. The study provides a better understanding of the underlying factors that drive developers to contribute to GitHub and Stack Overflow, and the challenges they face when participating on either platform. Further, using machine learning techniques the thesis attempts to predict software developer expertise based on their participation on social coding platforms.

The quantitative analysis showed that JavaScript remains a popular language, while knowledge and experience are the key factors driving expertise. On the other hand, qualitative analysis showed that soft skills such as effective and clear communication, analytical thinking are key factors defining an expert. The thesis found that both knowledge and experience are only necessary but not sufficient conditions for a developer to become an expert, and an expert would necessarily have to possess adequate soft skills. Lastly, an expert’s contribution to GitHub seems to be
driven by personal factors, while contribution to Stack Overflow is motivated more by professional drivers (i.e., skills and expertise). Moreover, developers seem to prefer contributing to GitHub as they face greater challenges while contributing to Stack Overflow.

In the next part of this thesis, multi-label topic classification modeling, a supervised ML technique, was applied to predict a software developer’s expertise based on their usage of Stack OverFlow (SO) and GitHub (GH) platforms. This technique enables the classifiers to learn from a set of instances, where each instance has more than one label and thereby predict a set of class labels for each test instance. Furthermore, unlike topic modelling, an unsupervised ML technique, the topic classification problem requires the documents to be labeled before modelling. The initial labels for SO and GH user activity data are derived from the topic modelling technique carried out by Norbert Eke [1] on the same data set. In other words, this thesis attempted to: (i) determine the best predicting classifier-algorithm for contributors (experts) in SO and GH, (ii) understand the distribution of expertise and their characteristics among SO and GH user community, and (iii) compare the prediction capability of the model between SO and GH.

Initially, two Word vectorizers (TFIDF and Word2Vec) were applied on the labeled SO and GH datasets. Thereafter, with an aim to pick the best performing vectorizer, in terms of their predictive performance, ten (10) multi-label classifiers were applied on the outputs from these two word vectorization techniques. It was observed that Word2Vec performed better than TFIDF on these datasets, i.e., it had lower hamming loss and higher F1 score on each of the ten multi-label classifiers. Thereafter, an effective data augmentation technique, MLSMOTE was used to balance the labels. This was done in order to further improve the predictive performance and reduce the overfitting of the data. This was done with aim to identify the best performing classifier.
Moreover, from the experiment it was observed that the same set of people do not necessarily contribute on both the platforms. Also, the GH community have more concentrated expertise and this is reflected in the low levels of influence that one label has on the others within a network of labels. On the other hand, SO users have more diverse expertise and their label co-occurrence frequency is very high.

This research could be further expanded to include other professional networking social platforms to better understand and predict software expertise, with greater confidence.
I dedicate this thesis to my two amazing sons, Karthik & Krithik, and my husband Ganesh, who supported me throughout this journey.
Acknowledgments

I, Sri Lakshmi Vadlamani, would like to express my sincere gratitude to my amazing supervisor, Professor Olga Baysal, for her continuous guidance, advice, and friendly discussions. I was able to successfully complete this work solely because of her continuous efforts, her valuable feedback and positive reinforcements at every stage of this project and also throughout my MS journey.

I am very thankful to God for giving me two sons, Karthik and Krithik; my children thoroughly supported me during this journey by showing great situational awareness and for encouraging me and keeping me optimistic at every step of this journey and especially in some difficult moments. Also, I am thankful to my parents, my in-laws, my siblings and my husband for their support and motivation.

I am thankful to my peers Norbert Eke and Khadija Osman for their patience, optimism, support, and encouragement when I needed it the most.

Last, but not least, I would like to acknowledge the advice and encouragement of several professors: Prof. Omair Shafiq from Carleton University, Prof. Diana Inkpen and Prof. Herna Victor from Ottawa University.
# Table of Contents

**Abstract** iii

**Acknowledgments** vii

**Table of Contents** viii

**List of Tables** xi

**List of Figures** xii

**Nomenclature** xiv

## 1 Introduction 1

1.1 Software Developer Expertise ............................. 1

1.2 Motivation ................................................... 3

1.3 Research Questions .......................................... 6

1.4 Contributions ............................................... 8

1.5 Thesis Organization .......................................... 9

## 2 Background and Related Work 10

2.1 Background .................................................. 10

2.1.1 Grounded Theory ........................................ 11

2.1.2 Pre-processing of Text Data in Software Engineering . . . . . . . . 15
2.1.3 Vector Representation Techniques .................................. 17
2.1.4 Classification Problem ................................................. 18
2.1.5 Class Imbalance .......................................................... 20

2.2 Related Work ................................................................. 22
  2.2.1 Defining Software Developer Expertise .............................. 22
  2.2.2 Software Developer Expertise Recommendation and Prediction 24

3 Methodology ........................................................................ 34
  3.1 Understanding Software Developer Expertise ......................... 34
    3.1.1 Data Collection ......................................................... 36
    3.1.2 Survey Design ......................................................... 37
    3.1.3 CUREB Ethics Application ......................................... 38
    3.1.4 Survey Participation ................................................. 39
    3.1.5 Survey Analysis ....................................................... 39
      3.1.5.1 Characteristics of Survey Participants ..................... 40
      3.1.5.2 Data Analysis .................................................... 44
  3.2 Predicting Software Developer’s Expertise ............................. 46
    3.2.1 Data Collection ....................................................... 47
    3.2.2 Data Cleaning and Text Pre-processing ......................... 51
    3.2.3 Multi-label Binarization .......................................... 53
    3.2.4 Model Construction ................................................ 56
      3.2.4.1 Feature Extraction Models .................................. 56
      3.2.4.2 Multi-label Topic Classification Model ................... 58
      3.2.4.3 Balancing Multi-label Classification .................... 62
    3.2.5 Evaluation Metrics ................................................. 63

4 Results ............................................................................ 65
  4.1 Understanding Software Developer Expertise ........................... 65
4.1.1 Defining an Expert .................................................. 67
4.1.2 Driving Factors Behind Contribution to Social Coding Platforms ........................................... 73
4.1.3 Challenges Faced by Contributors ............................................. 79

4.2 Predicting Software Developer’s Expertise ............................................. 81
4.2.1 Best Predicting Vector Representation Technique ............................................. 82
4.2.2 Predictive Performance of Classifiers ............................................. 85
4.2.3 Effects of Balancing the Multi-label Datasets ............................................. 86
4.2.4 Factors Impacting Predictive Performance ............................................. 94

5 Discussion ................................................................. 99
5.1 Findings ................................................................. 99
5.2 Implications ............................................................ 101
5.2.1 Employers & Recruiters ............................................................ 101
5.2.2 Researchers ............................................................... 101
5.2.3 Software Developers ............................................................ 102
5.2.4 Suggestions to SO and GH Decision Makers ............................................. 102
5.3 Threats and Limitations ............................................................. 103

6 Conclusion ................................................................. 105
6.1 Summary of Contributions ............................................................. 105
6.2 Future Work ................................................................. 107

List of References ............................................................. 109

Appendix A Appendix ............................................................. 120
A.1 CUREB Acceptance Letter ............................................................. 120
A.2 Expertise Study Survey Questionnaire ............................................................. 123
A.3 Technical Report ................................................................. 134
## List of Tables

1. Stack Overflow user profile extraction .............................................. 48
2. GitHub user profile extraction ....................................................... 49
3. Labels given to topics of the best LDA model trained on Stack Overflow 50
4. Labels given to topics of the best LDA model trained on GitHub ........ 50
5. Overview of concept categories ....................................................... 67
6. Features of software developer expertise ......................................... 71
7. Contribution to GitHub ................................................................. 75
8. Contribution to Stack Overflow ....................................................... 78
9. Results of ML classifiers, when TFIDF is applied on SO dataset .......... 83
10. Results of ML classifiers, when Word2Vec is applied on SO dataset ... 83
11. Results of ML classifiers, when TFIDF is applied on GH dataset ...... 84
12. Results of ML classifiers, when Word2Vec is applied on GH dataset ... 84
13. Results after applying MLSMOTE: SO dataset ................................ 88
14. Results after applying MLSMOTE: GH dataset ................................ 91
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Research Design Framework</td>
<td>13</td>
</tr>
<tr>
<td>2</td>
<td>Mixed methods research approach</td>
<td>35</td>
</tr>
<tr>
<td>3</td>
<td>Role of survey participants</td>
<td>40</td>
</tr>
<tr>
<td>4</td>
<td>Age of survey participants</td>
<td>41</td>
</tr>
<tr>
<td>5</td>
<td>Gender of survey participants</td>
<td>42</td>
</tr>
<tr>
<td>6</td>
<td>Geography of survey participants</td>
<td>42</td>
</tr>
<tr>
<td>7</td>
<td>Education of survey participants</td>
<td>43</td>
</tr>
<tr>
<td>8</td>
<td>Number of years of programming experience of survey participants</td>
<td>43</td>
</tr>
<tr>
<td>9</td>
<td>Contribution to GH by survey participants</td>
<td>44</td>
</tr>
<tr>
<td>10</td>
<td>Usage of SO by survey participants</td>
<td>45</td>
</tr>
<tr>
<td>11</td>
<td>Predicting software developer expertise: process diagram</td>
<td>47</td>
</tr>
<tr>
<td>12</td>
<td>Top 100 frequently occurring words in SO corpus</td>
<td>54</td>
</tr>
<tr>
<td>13</td>
<td>Top 100 frequently occurring words in GH corpus</td>
<td>55</td>
</tr>
<tr>
<td>14</td>
<td>Multi-label classification</td>
<td>60</td>
</tr>
<tr>
<td>15</td>
<td>Expertise determinants</td>
<td>69</td>
</tr>
<tr>
<td>16</td>
<td>Contribution visibility on GitHub.</td>
<td>73</td>
</tr>
<tr>
<td>17</td>
<td>Purpose of using Stack Overflow</td>
<td>76</td>
</tr>
<tr>
<td>18</td>
<td>Best predicting classifier for Stack Overflow</td>
<td>86</td>
</tr>
<tr>
<td>19</td>
<td>Best predicting classifier for GitHub</td>
<td>87</td>
</tr>
<tr>
<td>20</td>
<td>Number of SO experts per topic</td>
<td>89</td>
</tr>
</tbody>
</table>
21 Number of GH experts per topic .................................................. 90
22 SO LPSVM: balanced vs. unbalanced ........................................... 92
23 GH MLKNN: balanced vs. unbalanced ......................................... 93
24 Tag distribution for SO ................................................................. 95
25 Tag distribution for GH ................................................................. 96
26 Label space for SO ........................................................................ 97
27 Label space for GH ........................................................................ 98
Nomenclature

Abbreviations

This thesis uses some common abbreviations existing in the domain of computer science. The following table lists the abbreviations and their meaning:

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>SO</td>
<td>Stack Overflow</td>
</tr>
<tr>
<td>GH</td>
<td>GitHub</td>
</tr>
<tr>
<td>NB</td>
<td>Naive Bayes</td>
</tr>
<tr>
<td>LR</td>
<td>Logistic Regression</td>
</tr>
<tr>
<td>SVM</td>
<td>Support Vector Machine</td>
</tr>
<tr>
<td>CSV</td>
<td>Comma-Separated Values</td>
</tr>
<tr>
<td>RQ</td>
<td>Research Question</td>
</tr>
<tr>
<td>CUREB</td>
<td>Carleton University Research Ethics Boards</td>
</tr>
<tr>
<td>MLMOTE</td>
<td>Multilabel Synthetic Minority Over-sampling Technique</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

The first chapter introduces expertise of developers (Section 1.1), with an aim to layout the aspects that: (i) define an expert in software development and (ii) help predict software developer expertise. The next section (Section 1.2) provides a brief motivation behind this work, followed by the research questions that this thesis seeks to answer (Section 1.3). This is followed by the contributions of this thesis (Section 1.4), and lastly conclude this chapter with a thesis structure (Section 1.5).

1.1 Software Developer Expertise

Experience and knowledge are generally considered as both necessary and sufficient to make a person an expert at anything. This thesis attempts to investigate this issue in the context of software development by studying software developer’s expertise based on their activity (knowledge) and experience on GitHub and Stack Overflow platforms. In other words, this thesis attempts to: (i) understand how developers themselves define the notion of an “expert”, as well as why or why not developers contribute to online collaborative platforms and (ii) predict expertise based on a software developer’s participation on social coding platforms. Towards this, an exploratory survey was conducted with 73 software developers and a mixed methods
approach was applied to analyze the survey results. The results provided: (i) an insight into defining an expert, (ii) a better understanding of the underlying factors that drive developers to contribute to GitHub and Stack Overflow, (iii) the challenges faced by contributors while participating on either platform, and (iv) the best multi-label classifier that predicts a software developer’s expertise.

An expert is “…someone who has a broad and deep competence in terms of knowledge, skill, and experience through practice and education in a particular field”, according to one definition provided by Wikipedia. In other words, an expert is someone who has honed their special skills or knowledge in solving problems to perfection through practice and knowledge. The question is, does a person considered as an “expert” have these abilities innate in them or could these skills be possibly developed over time and through practice. In other words, an expert is an individual who has amply demonstrated their skills and/or knowledge in solving problems by optimizing the process thereby minimizes cost and time and/or maximizes revenue. An expert could have these abilities innate in them or they could have possibly inculcated them over-time and through conscious deliberate practice. There are some who argue that there is only a weak correlation between expertise levels and amount of experience and this is tested as part of this thesis.

In some fields of knowledge, it is relative easy to identify experts, to measure their expertise levels and to rank them. This is because, objective measures are available to identify and measure expertise. For example, in sports, there are numerous objective measures that can be used to reliably rank experts (e.g., Corsi, Fenwick and xG are some of the measures used to measure, rank and predict performance of both a player and a team in ice hockey). However, in some fields or professions it is not so straightforward to measure and rank experts; software engineering is one such field where it is much more difficult to find reliable and objective metrics to measure expertise, and this has been previously reported in the software engineering literature.
1.2 Motivation

It is common knowledge that software development is organized around a collection of multiple tasks, that include but not limited to: designing, programming, testing, debugging, and maintaining. Notwithstanding, since the early 21st century, the field of software development has evolved quite rapidly, and this has created various avenues for specialization and expertise, such as expertise in programming languages (e.g., JavaScript, C/C++, Python, JavaScript, SQL), frameworks and libraries (e.g., Node.js, Angular.js, Spring, ASP, Django, JSF, Cocoa, React, Hadoop, Spark), technologies (e.g., IoT, deep learning, ML, computer vision, blockchain, quantum computing, cloud computing, augmented/VR) and databases (e.g., MySQL, PostgreSQL, Oracle, Redis, Cassandra). In essence, there could be multiple ways to identify experts and measure their expertise. In this context, the main motivation of this work is to identify an expert and more importantly to predict an expert, based on their presence and behavior on collaborative platforms.

This thesis focuses on two platforms to help identify, measure and predict expertise of a software developer: GitHub (GH) and Stack Overflow (SO). On the one hand, GH is the most popular social coding platform that allows massive worldwide collaboration for software developers on open source projects [5, 6]. On the other hand, SO is the most popular question answering website for software developers, providing a large number of code snippets and text on a wide variety of topics [7,8]. SO offers the largest online community for software developers to learn by sharing their knowledge, and also helps in advancing their careers. In other words, SO helps programmers and software developers to discuss their (technical) day-to-day problems with others within the software development community and help solve their problems. At a fundamental level, both platforms offer very different set of features. However, both transcend physical and language boundaries and serve as a valuable
resource to modern day software developers across the globe in their quest to become an expert software developer. This is evident from the current usage of these platforms worldwide by developers. In a most recent data dump from December 09, 2018[1] Stack Overflow lists over 42 million discussion posts from almost 10 million registered users. On the other hand, GitHub’s search results report over 52.0 million users as of February 09, 2021[2] and according to a study in 2018, there are reportedly more than a 100 million repositories in GH [9].

The first part of this thesis attempts to understand the characteristics of an expert with an aim to create a image of an expert. In other words, the first part of this thesis attempts to understand whether knowledge and experience are both necessary and sufficient conditions to become an expert. And, the thesis goes on to argue that later is the case and emphasizes that soft skills are more critical to both convert and refine knowledge & experience and move a person towards becoming an expert. Furthermore, this thesis also attempts to understand a developer behaviour and decisions on why they contribute to different online collaborative platforms and the problems they face while contributing on these platforms. While software research community offers a plethora of studies on how developers participate, contribute and interact with each other on various collaborative platforms [5,6,10–14], there is a lack of qualitative research studying developer motivation and behaviour that drives their contributions to these platforms. This thesis attempts to close this gap by undertaking an exploratory survey that was designed based on the state-of-the-art qualitative research [10,15,16] and also based on observations on how developers interact and participate on Stack Overflow and GitHub. Lastly, a mixed methods research approach involving both quantitative and qualitative analyses was applied in order to analyze the survey data. The first part of this thesis has resulted in a publication titled “Studying Software
Developer Expertise and Contributions in Stack Overflow and GitHub” by Vadlamani and Baysal [17]. This paper investigated expertise in software development based on a person’s activity and experience on GitHub and Stack Overflow platforms. Furthermore, it defined basic characteristics of an expert to understand the underlying factors that drive developers to contribute to GitHub and Stack Overflow, and the challenges they face when participating on either platform.

Now thereafter, the second part of the thesis attempts to predict software developer expertise using multi-label topic classification modelling, a “supervised” machine learning technique. This technique requires the documents to be labeled prior to undertaking the modeling exercise. In other words, by applying the multi-label topic classification modeling this thesis attempts to predict a software developer’s expertise based on their participation on Stack Overflow (SO) and GitHub (GH) platforms. This technique enables the classifiers to learn from a set of instances, where each instance has more than one label and thereby predict a set of class labels for each new instance. Furthermore, unlike topic modelling, an unsupervised ML technique, the topic classification problem requires the documents to be labeled before modelling. The initial labels for SO and GH user activity data are derived from the topic modelling technique carried out by Norbert Eke [1] on the same data set.

This research could benefit multiple stakeholders. Employers could benefit from knowing trends in specialization and understanding characteristics of an expert as it would help them hire the right expert at lower search-costs. Furthermore, learning the reasons that motivate developers to contribute to online collaborative platforms would help employers make efforts in retaining talent and attract more talent. Moreover, based on a software developer’s online contributions, the techniques developed in the second part of the thesis could help employers to more accurately predict experts using artificial intelligence and thus hire the right expert at lower search-costs. Employers could use the predictive power to chart out training and development requirements
of current employees; thus, helping corporations to better retain existing talent pool and to maximize their returns on investing in human resources. Researchers, on the other hand, could benefit by designing studies and developing theories related to multi-layered expertise and possibly expanding the number of platforms to include developer profiles and contributions from other social media platforms that include but not limited to Twitter and LinkedIn. Finally, software developers themselves could possibly benefit by learning the various factors that lead to becoming a software developer and the various clusters of specialization that they ought to focus on. This would save them time and cost by providing them the optimal combination of specialization. Knowing the trends in specialization would help developers formulate their goals and acquire the right set of tools to be successful at the job. Also, understanding the motivations that drive developers to contribute and participate to different online platforms can help the platform’s UX and increased participation and engagement by experts could help software developers to turn to the appropriate platform (GitHub, Stack Overflow) depending on their specific needs.

1.3 Research Questions

The following research questions are addressed in this thesis:

- **RQ1: How do developers define an expert?**

  This research question (Section 4.1.1) attempts to understand and define an expert based on their activity on social coding platforms: Stack Overflow (SO) and GitHub (GH). This research question is important, as it is the first step in identifying an expert and ranking their level of expertise in software engineering domain. Once expertise has been defined, the follow-up question would focus on understanding the factors that motivate developers to contribute on these platforms.
• **RQ2: What factors drive developers to contribute to SO and GH?**

This research question (Section 4.1.2) intended to understand the motivating factors that compel developers to participate and/or contribute on either of the two platforms. In other words, this RQ investigates whether personal factors (such as self needs or hobby or altruistic reasons) or professional factors (such as job, expertise or organization culture) drive developers to contributing on social coding platforms. The insights derived here could help expand the understanding of cross-platform developer expertise and help the decision makers of the respective online coding platforms to make appropriate amends as they deem appropriate.

• **RQ3: What challenges do developers face when contributing to each platform?**

This research question (Section 4.1.3) builds further on top of RQ2 and analyzes the difficulties faced by developers while contributing on the two platforms. This further would help decision makers of these platforms gain understanding of the issues faced by contributors and possibly work towards resolving the same in order to further increase participation on these platforms.

• **RQ4: What feature is best predicting the software developer expertise in SO and GH?**

In this research question (Section 4.2.1), the thesis moves on from understanding expertise and contribution to various coding platforms to predicting software developers expertise in various software fields, based on their contributions to these two social coding platforms. This RQ attempts to understand the best word vectorizer, in terms of predictive performance, on this specific dataset.
• **RQ5:** What is the predictive performance of classifiers in predicting software developer expertise in SO and GH?

Once the best word vectorizer on this specific dataset is identified from the above RQ, the onus shifts to comparing predictive performance across classifiers, with an aim to choose the best performing classifier. This research question (Section 4.2.2) attempts to identify the best multi-label classifier.

• **RQ6:** How effective is MLSMOTE, a multi-label balancing technique, in improving the predictive performance?

Mostly, all the real world data is unbalanced. This data is not an exception. Hence, this data is balanced with an aim to improve the predictive performance of the minority class labels. So this research question (Section 4.2.3) identifies the best predictive classifier after augmenting the multi-label classification data.

• **RQ7:** How SO and GH data characteristics impact the performance?

Based on the above three RQs (RQ4, RQ5 and RQ6), this final research question (Section 4.2.4) attempts to draw implications of contribution by experts on these two social coding platforms and identifies their idiosyncrasies with regards to their contribution to either platform.

### 1.4 Contributions

The below is a summary of the contributions of this thesis.

• Provide a qualitative study with the developers active on both SO and GH who participated in a survey on the topic of developer expertise and their motivation for contributing to each platform.
- Analyze driving factors behind contribution on both platforms, based on a survey analysis that offers insights into the developer perception of what constitutes to the notion of an expert developer.

- Determine the best predicting classifier-algorithm for contributors (experts) in SO and GH.

- Understand the distribution of expertise and their characteristics among SO and GH user community.

1.5 Thesis Organization

The structure for the rest of this thesis is as follows: Chapter 2 includes Background (Section 2.1) and Related Work (Section 2.2); while Chapter 3 describes our Methodology. Results are presented in Chapter 4, followed by Discussion in Chapter 5. The thesis concludes with Chapter 6 outlining Summary of Contributions in Section 6.1 and Future Work in Section 6.2.
Chapter 2

Background and Related Work

This chapter outlines the necessary background knowledge needed for this work (Section 2.1) and then presents all the related work (Section 2.2).

2.1 Background

This section is divided into two parts: (i) the first part provides an outline of the best practices, techniques and procedures used in understanding and measuring an expert, and (ii) the second part provides background information about the tools, techniques and methodology used in predicting an expert in the field of software development. More specifically, Section 2.1.1 provides a brief note about card sorting technique that was used to conduct qualitative analysis of the survey data. Thereafter, Section 2.1.2 introduces text processing of software engineering data. The Section 2.1.3 introduces vector representation techniques, such as TFIDF and Word2Vec; and, this is followed by a detailed discussion on multi-label class balancing in Section 2.1.4. Lastly, Section 2.1.5 provides a brief introduction to topic classification by explaining multi-label classification problem and the various types of classifiers used throughout this thesis.
GitHub (GH) and Stack Overflow (SO) are the two most popular online platforms that provide a social space for software developers, from across the world to discuss problems, share knowledge and collaborate on open source projects. These platforms have transcended the barriers of verbal language and hence are able to rapidly attract increasing members as well as their contributions. Moreover, rapid adoption of these platforms over the last decade meant that these platforms are not only important for the overall skill development of software developers, but they have also become important tools for potential employers and recruiters to understand the skills and interests of potential hires, thus reducing search costs of hiring new candidates [18].

Despite the overwhelming similarities in the manner in which these platforms serve software developers, researchers and employees, these two platforms have fundamental differences. On the one hand, GH is essentially a social coding platform where developers can collaborate on open source projects. Whereas, SO provides an online space that allows developers, irrespective of their expertise levels, to ask and/or answer queries of other developers, discuss the latest updates or bug fixes, provide documentation updates.

2.1.1 Grounded Theory

The Grounded Theory (GT), introduced by Glaser and Strauss [19], is a methodology to construct theory, about a social phenomena, entirely from data using both qualitative and quantitative data generation techniques [20], [3]. Thereafter, this methodology evolved into three different approaches of generating theory from data:

1. Glasser GT: Emphasis on inductive nature of GT (one end of the spectrum).

2. Strauss and Corbin GT: The emphasis here is on systematic strategies and verification (the other end of the spectrum).

3. Charmaz’s Constructivist GT: Takes the middle path between the above two
diverging schools; this school provides the flexibility of the original Glaser and Strauss paper and combines it with the robustness offered by constructivist epistemology.

Although these three might be different in their approaches, their underlying principle remain the same, as all three approaches rely on meticulous application of specific and systematic methods for collecting data and coding process for analysing data by designating words or phrases that capture the essence to sections of unstructured data. Thereafter, these codes are structured into higher levels of abstractions after a series of iterative and continuous comparison and aggregation, a process called as “constant comparison” in this literature.

In general, a Ground Theory includes the following series of iterative and continuous steps:

- **Data Sampling.** There are many available techniques for sampling; Tie et al. suggest starting with Purposive Sampling method.

- **Data Generation and/or Collecting Data.** This state involves collecting data using variety of methods that involve, but not limited to survey, interviews, focus groups and other data.

- **Various Stages of Coding.** Depending on the specific methodology, the stages of coding could be different. However, in general, it involves initial coding (coding; category identification); intermediate coding (selecting core category, data saturation); and advanced coding (developing a story and theoretical coding). These coding stages are undertaken along with “constant comparative analysis, theoretical sampling and memoing” of this literature. This theoretical sampling is recommended to be applied till a theoretical satiation is arrived.

The Figure taken from Tie et al. provides a comprehensive synopsis of the above enumerated process.
Figure 1: Research design framework: summary of the interplay between the essential grounded theory methods and processes. Credits to Tie et al. [20]
Moreover, the coding approach in each type of approach is slightly different. For example, the traditional Glasser coding approach would involve the following stages of sequential coding: (i) initial open coding, (ii) selective coding and (iii) theoretical coding. Whereas, the Strauss and Corbin’s evolved GT would involve: (i) open coding, (ii) axial coding and (iii) selective coding. Finally, Charmaz’s constructivist GT would have the following three (3) main phases:

1. *Initial coding.* In this first phase it is recommended that the researcher remains open in their approach and does not deviate from the data. In other words, the researcher should not be prejudiced by any pre-existing conceptions or notions of what the world could be or would be, i.e., it recommends that researcher remove any subjectivity and purely focus objectively on the story the data is presenting. Furthermore, Glaser takes this to an extreme by recommending that the researchers should not undertake any literature review prior to conducting GT research.

2. *Focused coding and categorizing.* Conduct literature review and integrate with results from Phase 1.

3. *Theory building.* This the last and final phase that brings everything together. The first two phases help provide abstract concepts from the data and the last phase draws connection within the abstract concepts that emerged from the first two phases to generate a single unifying theory.

The details about how the above process was implemented in this thesis is explained in Section 3.1.5.2.
2.1.2 Pre-processing of Text Data in Software Engineering

A software developer expertise could possibly be revealed by the outward signal the developer provides about their abilities. One way, a software developer signals their expertise is by participating and contributing to online social coding platforms. The participation on Stack Overflow and GitHub is filled with technical know-how that is intended to showcase, to others, their expertise. This is reflected in the text corpus on both the platforms, which contains a wide and diverse assortment of software engineering specific items and sometimes very esoteric terminology and the textual (natural language) data on these platforms is very often mixed with computer code.

The Stack Overflow platform focuses on questions and answers. On the one hand, software developers pose queries explaining the problem that they have run into; on the other hand, there is usually a discussion surrounding the question and there are invariably multiple answers/responses for most queries. The conversation might be between the initial author of the query and a respondent, or a discussion might emerge amongst various respondents. Irrespective, based on the queries posted and the detailed responses received on SO, the text corpus is made up of a variety of ingredients, that include but not limited to: source code, code snippets, text blocks explaining the problem and solution, hyperlinks providing reference to potential source code, journal article or API documentation. Similarly, on the GitHub platform, where developers collaborate on open source projects, share software code, discuss in-depth about a software project, its bugs and enhancements, the text corpus is equally varied and the discussion, among other things, is infused with technical jargon, generic text, code snippets, and error messages [14].

In essence, the text corpus on both platforms, is a complex mixture of natural language, technical terminology, computer source codes and hyperlinks, thus, the task
of pre-processing of this data in order to extract the appropriate data for analysis becomes very critical. The software engineering literature provides multiple techniques to pre-process such data, so as to ensure that the right text corpus is extracted, whilst maintaining the contextual sanctity of the software artefacts. However, each technique has its own pros and cons; thus, each technique has to be researched carefully to understand how it was applied for analyzing data from SO and GH.

Tokenization, stop-word & HTML symbol removal, normalizing strings by converting to lower case and applying stemming are some of the techniques that have been used for text pre-processing [21]. Tian et al. [22] used these techniques to clean and extract data from the two platforms; moreover, in this paper, during the text processing phase, the function names were split from the keywords extracted from the cleaned up source code.

In addition to removing stop words and conducting stemming, Campbell et al. [23] suggested performing lexical analysis, especially while applying LDA models, prior to building a vocabulary. Furthermore, additional (optional) measures like, filtering out uncommon words and/or high frequency words, and creating a bag-of-words representation of text, were also suggested in this work.

On Stack Overflow, other pre-processing methods involved: (i) converting text to lowercase, (ii) removing code snippets, HTML tags, punctuation whilst ensuring some important software terms and symbols are maintained [24].

On Stack Overflow, another strategy is to: (i) remove line breaks, code blocks, HTML tags, (ii) replace special characters including HTML symbols with a normal character, and (iii) replace sequence of multiple spaces with a single space. On the other hand, the strategy on GitHub included all the above, plus removing: (i) code comments, (ii) characters denoting section headers, (iii) vertical and horizontal lines, (iv) characters that represent links or formatting. This was the solution proposed and performed by Treude and Wagner [25].
Finally, a recent study by Liao et al. [14] employed the following techniques on GitHub: (i) removed code-specific items from within the HTML tags, (ii) removed short posts that contain five or less tokens and (iii) took into consideration the commit history of a software developer.

2.1.3 Vector Representation Techniques

Since their inception, the two platforms, SO and GH, have historically collected huge amounts of data; moreover, at the rate at which new members are joining these platforms, the textual data inputted daily into these platforms is exponentially increasing. However, computers are inept at contextualizing and classifying textual data; therefore, a meaningful processing of such a text corpus is required in order for a machine to make sense of this data. This involves performing tasks like clustering, classification and machine translation on the textual data. In other words, it involves creating a representation of words that captures its essence, understanding the semantic relationships and appropriately contextualizing the data, based on their usage. This is done by converting words into numbers or by using word embeddings of textual data, so it becomes possible for computers to handle the text corpus from these platforms.

There are different types of word embeddings, each with their own pros and cons; broadly, though, the word embeddings could be classified based on: (i) frequency and (ii) prediction.

The frequency based deterministic word embeddings, generally has three types of word vectorization techniques: (i) Count Vector, (ii) TF-IDF Vector, and (iii) Co-occurrence Vector.

However, the deterministic embedding models are limited in their scope. This was cured by the introduction of Word2Vec, a prediction model that assigns probabilities

---

1A simple explanation of word embeddings: [https://www.analyticsvidhya.com/blog/2017/06/word-embeddings-count-word2veec/](https://www.analyticsvidhya.com/blog/2017/06/word-embeddings-count-word2veec/)
to the words within a corpus. This methodology, introduced by Mikolov et al. in 2013, worked especially well with word similarities and word analogies. More specifically, Word2Vec methodology is a combination of two shallow neural networks that learn weights and thus act as word vector representations: (i) Continuous Bag of Words (CBOW) and (ii) Skip-gram model.

The Word2Vec methodology, a feed-forward neural network model with one hidden layer, learns and gives a high dimensional word embeddings as output from an input of textual data in the form of one-hot encoding. On the one hand, the CBOW model predicts the current word based on its context (surrounding words); on the other hand, using the current word, Skip-gram learns by predicting the surrounding words. In essence, both these models do the inverse of one another and hence complement each other. In addition to Word2Vec, the deep learning literature has other word vector representation models, that include but not limited to, GloVe and FastText.

2.1.4 Classification Problem

Classification involves the use of machine learning algorithms that are trained to map input samples to target pre-determined class labels. Basically, the data is split into a ‘train’ and ‘test’ datasets. After ensuring the ‘train’ dataset is adequately representative of the overall data, the model uses this dataset to map examples of input data to specific pre-determined class labels. In other words, the class labels are the categories into which the input labels can be mapped into, prior to undertaking the modeling exercise; moreover, each class labels are assigned unique integer value. In general, there are two types of classifications:

1. **Single label classification.** This involves mapping an input sample to an *unique* pre-determined target label. This classification can be further divided into:
(a) **Binary classification.** This classification type has two unique disjoint data labels and all the input data is mapped into either one of these two labels, based on a specific pre-determined criteria. For example, this method could be used in areas that include, but not limited to, spam detection \[30\] and disease diagnosis \[31\]. Although a popular method, this method has its limitations, as most real life problems do not have binary labels, but multiple target labels. This limitation is resolved by using multi-class classification method.

(b) **Multi-class classification.** This classification type has more than two unique disjoint data classes. Some real life problems that use this type of classification include, but not limited to, character recognition \[32\] and bio-metric identification \[33\].

2. **Multi-label classification.** The single label classification, however, breaks down in instances when the input samples can be potentially mapped to more than one target label, i.e., the unique target label association no longer holds. This is resolved by using multi-label classification of the input data, wherein each input sample can be mapped into multiple categories. For example, a software engineer can have expertise in multiple domains such as Java, C, C++, Python, Databases, etc., and this individual cannot be associated with one label. Thus, in this sense, classifying experts is a multi-label classification problem. One more example is its use in classifying movies; the same movie could be classified both as ‘action’ and a ‘comedy’ at the same time \[34\]. Furthermore, this classification is emerging rapidly as its range of real world applications is increasing \[35\]. Several machine learning techniques have been developed for this classification problem and these can be broadly grouped under three categories: 1) Algorithm Adaption Methods, 2) Problem Transformation Methods,
and 3) Ensemble Methods. These methods are further described in Sec. 3.2.

2.1.5 Class Imbalance

One major issue encountered in machine learning classification, while dealing with real world problems is that most often there are disproportionately higher number of observations in one specific class or in a group of classes, whilst other classes are under-represented. In other words, the more important class is usually over-represented and the minority classes tend to be ignored in the ‘train’ dataset. These skewed observations, also known as class imbalance, leads to inaccurate prediction of the classes in the ‘test’ dataset. Furthermore, the machine learning algorithms are designed to have a greater prediction capability when the number of samples in each class are balanced, i.e., they are about equal.

In the literature, there are several methods to deal with the class imbalance problem and the following are the most popular ones:

- **Changing the Performance Metric.** In the presence of class imbalance, accuracy, a metric used to evaluate the performance of a classification model would become misleading. In such cases, it is recommended to use additional metrics to evaluate the performance of the model. Some of the best evaluation metrics are: (i) Confusion matrix, (ii) Precision, (iii) Recall, (iv) F1 score and (v) Hamming loss.

- **Change the Classifier.** It is recommended to evaluate the model using different classifiers. One frequently applied classifier, especially on an imbalanced dataset is decision tree and its variants. This because of the splitting rule that governs the class variables forces both classes to be addressed.

- **Resampling Techniques** are used to balance the data. This involves either under-sampling by randomly removing the majority class instances or randomly adding
the minority class instances.

1. **Under-Sampling** the majority class, that leads to random elimination of majority class examples. The major disadvantage with this method is that it could discard some potential useful data, which could be useful for prediction.

2. **Over-Sampling** the minority class instances are added to the dataset, thereby balancing it. This however has two disadvantages: (i) it increases the probability of over-fitting the data and (ii) it increases the run-time, especially if the dataset is already large.

- **Generate Synthetic Samples.** This is also an over-sampling technique wherein minority class instances are generated synthetically by interpolating instead of replicating. The advantage of this method is that it avoids over-fitting.

- **Ensemble Learning Methods.** This methodology helps improve the performance of a single classifier. These method involves two techniques: (i) **Bagging**, an abbreviation for Bootstrap Aggregating, generates \( n \) independent bootstrap training samples with replacement; also, all these predictions are summed up at the end. The main advantage of this technique is that reduces over-fitting and improves accuracy; and (ii) **Boosting**, on the other hand, combines weak learners to create one complex strong learner that improves the model prediction. The main advantage is that the implementation process is simple and it overcomes the over-fitting problem.
2.2 Related Work

This section provides a literature review of the related work in two main areas: *defining software developer expertise* is presented in Section 2.2.1 and *software developer expertise recommendation and prediction* in Section 2.2.2.

2.2.1 Defining Software Developer Expertise

Sindhgatta [38] attempted to identify a software developer’s domain expertise from the source code written by the developers. More specifically, in order to detect and extract the domain expertise of a software engineer, the author applied information retrieval and document clustering techniques. These techniques filtered out all software programming keywords from a corpus of text documents; thereafter, these documents were clustered and grouped in order to extract relevant concepts. Lastly, multiple case studies were applied to evaluate the technique and to confirm that this technique can extract the relevant concepts from the source code. In fact, Montandon et al. [39] followed a very similar approach to that of the work by Sindhgatta.

Software development is divided into a collection of tasks, of which the important ones are: programming/coding, testing and debugging. Bergernsen et al. [40] proposed an instrument to measure the first aspect (programming), based on a fixed set of tasks pertaining to it. However, this measure might not help in identifying expertise in software development, as modern-day software developers have diversified their expertise across multi-specialties and the importance of soft skills has increased over time, i.e., this measure has no objective manner to measure their expertise in these multiple facets of software development.

Constantinou and Kapitsaki [41] attempted to extract software developer expertise in different programming languages based on commit activity of developers on
GitHub, considering both the number of commits, as well as the contribution consistency. Moreover, the results are then validated against the same experts’ activity on Stack Overflow using the linked dataset of active users on both the platforms (Vasilescu et al. [13]). The paper concludes by noting that their approach can: (i) identify software engineers with similar expertise and (ii) recommend experts for specific tasks.

Baltes and Diehl [3] have presented the first conceptual theory of software development expertise that is grounded in data from mixed methods survey of 335 software developers, literature review on expertise and focused surveys to come up with a comprehensive conceptual theory of expertise. They note that experts’ performance may be inversely proportional to time and that experience and expertise may not necessarily be correlated. However, their study only focused on expertise in Java programming language.

Montandon et al. [39] have studied the question of identifying experts in software development by focusing their efforts on identifying experts in usage of libraries and frameworks among GH users. Supervised and unsupervised ML classifiers were used to identify experts in the three most popular JavaScript libraries (facebook/react; mongodb/nobe-mongodb; & socketio/socket.io). A ground truth of expertise was built on a self-reported survey of 575 developers on these libraries. The paper tackles two research questions: 1) identifying the accuracy of the ML classifiers in identifying experts in libraries, and 2) the features that distinguish an expert from a novice in the studied libraries. The ML classifiers F-measure turned out to be low and the authors interpreted this performance might be due to usage of GitHub as a full proxy for expertise and they note that experts rarely contribute to public GitHub projects. On the second question, they rely on clustering to identify experts with similar feature values. Their results suggest that GitHub could be a partial proxy for expertise in frameworks and libraries. This paper differs from Montandon et al. [39] by allowing
the respondents to self-select themselves and by choosing two platforms (GH and SO) to help cross-validate. Xiong et al. [11] studied developer behavior across GH and SO by mining accounts that were linked, using a CART decision tree approach. Their results found high levels of correlation between activity on the two platforms.

Li et al. [42] attempted to understand the factors that separate a ‘great’ software engineer from an ‘ordinary’ one. The authors used a mixed-method approach to understand the same. In the first part, the authors surveyed about 2,000 software engineering experts to understand their subjective opinion about the attributes of great software engineers. Thereafter, the authors conducted interview over emails to interpret their findings and to understand the influence that contextual factors might have on subjective ratings of experts. The paper concludes that great software engineers: (1) learn continuously, (2) have the ability to write good code, (3) practice informed decision making, (4) avoid making others’ jobs difficult and (5) ability to optimize their current behavior based on future payout and risks.

GH and SO have also been empirically studied for a variety of topics. For example, some research focused on analyzing topics that developers asked about in SO [43–48]. Similarly, an extensive research has analyzed programming languages that developers used in GH and their relationships to GH contributions [10,12,49–54]. There are also studies characterizing social network properties of GH and SO [47,55–58].

2.2.2 Software Developer Expertise Recommendation and Prediction

Many researchers have studied expertise in GH and SO including (not limited to) Chen et al. [59] recommended experts based on an analysis of correlations between search keywords in Google and SO. Silvestri et al. [60] conducted a comparative study across SO, GH and Twitter to investigate correlations between interactions.
Yan et al. [61] used heterogeneous information network analysis across various online communities to draw a comprehensive picture of a software developer and their expertise. Yu et al. [62] used feature matching developed a cross-domain developer recommendation. Venkataramani et al. [63] built a recommendation system on SO based on data mined from GH. Constantinou et al. [41] measured developers’ commit activity on GH and validated that by checking on their activity on SO. Huang et al. [64] provided skill scoring by correlating developer activity on GH and SO. Zhang et al. [65] proposed DevRec, a recommendation system on GH and SO, based on association matrix is proposed. However, the system does not consider user attributes and weighting mechanism. Some of the other papers in this area are discussed below in a little more detail.

Alonso et al. [66] is probably one of the earliest paper that dealt with the issue of expertise identification. The authors applied a ‘rule-based’ classification of source code trees, on open source projects, in order to find the expertise of contributors. The authors made a distinction between the contributors depending on their respective commit privileges. In other words, there are two type of contributors: (i) project owners/developers, these people are smaller in number but with greater commit privileges, and (ii) project contributors, there are many people who contribute to the project via bug fixes, recommendations/suggestions and patches but have very low level of commit privileges. The authors went on to use the (version controlled) log files and source code trees to automatically detect and predict the expertise of developers.

Using graph theory, Surian et al. [67] extracted and recommended the top developers based on computation of similarity measures that are dependent on random walks of the graph with restarts. The paper aimed to predict experts for a specific project based on the individual’s historic work experience and their stated skills; this method has predicted the experts with 83.3% accuracy on a ground truth data set. Moreover, this paper created a graph based model to understand and represent
developers collaborating on projects.

Nguyen et al. [68] demonstrated how the automated developer expertise ranking could become a valuable tool to assess software development and a software developer expertise. The authors analyzed the ability of a software developer to solve problems or defects in a software program or source code in order to infer about the individual’s expertise in software development. The paper categorized each rectified defect by subject/topic and ranked software developers based on the difficulty level of the problem solved and the time to solve the problem. Moreover, the paper cross-validated their findings by interviewing many project managers in the field of software engineering; these project manager were tasked to compare their subjective ranking with that of the ranking of the study.

Venkataramani et al. [63] mined software developer activities on open source code repositories to rank their technical expertise. The authors built a recommendation system to: (i) rank and recommend software developers based on their expertise that is gauged by their contribution to open source projects and (ii) provide question recommendation on Stack Overflow based on data mined from GitHub. Constantinou and Kapitsaki [41] constructed a similar cross-platform recommendation system.

Whilst some studies focused on building a cross-platform recommendation systems, the other focused entirely on either GitHub or Stack Overflow. Teyton et al. [69] designed a ‘search engine’, using a novel query language, to identify and recommend relevant experts of common libraries among software developers on GitHub. According to this work, an experts on library is defined as software developer who has committed changes to the underlying base source code of the library. Although, this search engine can help query libraries on open platforms and hence is useful in identifying experts on social coding platforms, the authors note that this system lacks proper validation and hence might be the reason why this is underused. In a similar work, Hammad et al. [70] ranked software developers based on their knowledge.
and provides a recommendation system that automatically suggests a list of potential candidates who are experts, i.e., are best suited to work on a specific software design change task. The methodology does two things: (i) on one hand, it takes into account the ‘knowledge’ required to successfully complete a task and (ii) on the other hand, it detects the type of software knowledge possessed by a so called ‘software’ expert. Lastly, the authors opined that this recommendation system might be apt for large scale projects with many software developers. In a similar work, Kagdi et al. [71] ranked software developers based on implementation of change requests on software. Moreover, they used a ‘concept location’ methodology along with software repository mining, to recommend expert software developers. Their work reports a very high variance in accuracy values, ranging between 47% and 96%, with regards to recommending developers to fix specific bugs. Whereas, the variance in accuracy values is much lower, ranging between 43% and 60%, for recommending developers to implement specific features.

Tian et al. [22] leveraged on topic modeling and used a collaborative voting technique to predict and recommend the expert that can best answer a new question on Stack Overflow. Using LDA models, the authors modeled an user’s interests and expertise on a specific topic. They built a user profile of software developers based on their previous queries and/or their responses to queries; also the paper assumed: (1) an individual user with high interest and expertise on the specific subject is considered the best person to respond to a query, (2) the questions on SO came from a variety or a bag of topics and (3) a particular user’s interests and expertise on a related topics drove their interest and expertise in questions they chose to answer on SO. Lastly, the paper compared their topic modeling approach with TFIDF, a frequency based deterministic baseline approach and concluded that the topic modeling approach outperformed the baseline TFIDF approach.
In an exploratory study on GitHub, Zhang et al. [72] developed a recommendation mechanism that would match software developer, based on their user activity and behavior on the platform, with compatible projects on GitHub. In other words, the authors attempted to understand the types of user behavior data and their relation with project recommendation. They found that user activities and behaviours that are most relevant for recommending projects to software developers are: ‘fork’, ‘watch’, ‘pull-request’ and ‘member’. Lastly, the authors noted that the majority of recommended projects are either inter-dependent, or co-used, or they are just very similar projects.

In a study similar to Surian et al. [67], Hauff and Gousios [18], attempted to match developer profiles to job adverts and provided automatic recommendation to software developers based on their activity on GitHub. The study’s methodology could be divided into 3 broad steps: (i) extracted concepts from job description and from software developer profiles, (ii) created a hierarchy of these concepts by appropriately weighting them and (iii) matched developer profile with that of the job description.

González et al. [73] proposed a 5-way multi-class classifier system to predict the tags assigned to questions. Their model had 76% F1-Score for the best performing classifier. Sood et al. [74] developed a tag suggestion system for blog posts for finding similar blog posts and suggesting some of their tags by using information retrieval methods. Stanley et al. [75] proposed a Bayesian probabilistic model to predict tags for Stack Overflow posts. For each tag they computed an activation score given the words in the title and the body that mainly depends on co-occurrence statistics. Kuo [76] has worked on Stack Overflow tag prediction. The author used a co-occurrence model to predict tags. Their co-occurrence model had a 47% classification accuracy in predicting one tag per post.

In a work spanning across three platforms: Stack Overflow, GitHub and Twitter,
Silvestri et al. [60] linked together user accounts across these platforms and conducted a comparative analysis of user interaction on the these three platforms. The aim of this study was to understand the correlation of cross-platform usage. The authors found that: (i) Twitter users are more interconnected than SO and GH, (ii) in comparison to both Twitter and GH, SO has the less skewed distribution of user popularity.

In another study on GitHub, Yu et al. [77] attempted to build a recommendation system for reviewers of pull requests in GitHub, based on previous approaches used in code review and bug fix assignments. For this recommendation system, they proposed a new method that involved mining social relationships between the individual submitting and the potential reviewer of the corresponding pull request; moreover, the authors also used traditional ML adapted methods. Their results showed that the traditional ML adapted methods outperformed the baseline; on the other hand, there is no statistically significant difference in performance between the new method, suggested by the authors, and traditional ML methods. Lastly, the authors experimented with a hybrid methodology and found significant improvement in results, i.e., significant increase in precision and recall, and a more balanced overall accuracy than other approaches. Similar to this work, Rahman et al. [78] proposed another novel technique to recommend ‘code reviewers’ taking into account the software developer’s technology experience and considering their cross-project. Moreover, the authors carried out their experiments on ten (10) commercial projects and an equal number of external libraries that are dependent on commercial projects. Their results indicated a high technical accuracy in code reviewer recommendation, between 85% and 92%.

Greene and Fischer [79] created ‘CVExplorer’, a tool to help employers receive recommendations about potential employees for open positions. This tool extracts, explores and visualizes technical skills of developers by mining and exploring their
activity on GitHub. Thereafter, this tool creates an ‘tag cloud’ of a potential employees and allows the recruiters to quickly scan an employees skill set. In addition, this tool made these tags interactive, so if a recruiter wanted to explore further about a specific skill of a potential employee they could click on a tagged word to get further details about the activity done by a potential employee. The authors noted that this tool was evaluated at two companies and it was showed that this tool proved to be very useful for ‘non-technical’ recruiters to hire ‘technical’ employees. In a similar vein, Huang et al. [64] modeled and evaluated a developer’s programming ability by mining and evaluating a developer’s activity across both Stack Overflow and GitHub. In other words, the authors developed a tool called ‘CPDscroer’, that assigns each developer an expertise score for each computer language upon mining and analyzing software developer’s GitHub projects and Stack Overflow posts. The authors noted that their approach is both practical and robust and it achieved a high precision score of about 80% on both SO and GH data.

Badashian et al. [80] created a developer recommendation system that ranks and lists developers who are most capable of resolving software bugs. They used a novel approach to determine a developer’s expertise in resolving bugs, based on the individual’s contributions in Stack Overflow; moreover, they used SO repository to created this ranking, as they believed that it is a rich source of developer expertise related data. Lastly, the authors claimed a high level of accuracy of this proposed novel approach in recommending/assigning developers to fix software bugs. Another recommendation system, on open source repositories, was proposed by Zhang et al. [65] by combining cross-platform user activities on both Stack Overflow and GitHub. The results showed that this system was successful in proposing candidates for the cold start problem, whilst some of their hybrid approaches performed well at recommending experts for open-source projects.
In another work similar to Teyton et al. [69], Liao et al. [81] ranked influential developers on GitHub. A developer’s influence is calculated as the number of followers a developer gains over a period of time. Thereafter, the developers are ranked based both on their influence and their commits on GitHub, using a novel network-based algorithm, *DevRank*. Further work in these lines was been carried out by Tian et al. [82]. In their work Tian et al. [82] proposed another methodology to extract information about the top experts in various areas of software engineering. Their methodology identified and ranked experts (or ‘geeks’) in various software fields based on their activity on GitHub, Stack Overflow and over both the platforms, by considering relationship between developers on various projects. Wang et al. [83] designed and implemented an online tool to rank and characterize an expert and allow for real-time searching and profiling of a software development expert. This profiling analysis is based on a software developer’s commits of source code to GitLab repositories. A developer is profiled based on the four (4) broad perspectives: (i) quantity of code submitted, (ii) quality of code, (iii) contribution to check on the time committed and understand the overall skill and (iv) the score of the projects they participated.

There are different types of recommendation platforms that are available to rank and identify experts in the field of software engineering. Wang et al. [84] surveyed the various recommendation systems available for the Stack Overflow platform. They provided an overview of the latest research techniques and classified the existing recommendation methods into eight (8) categories: (i) simple, (ii) language model-based, (iii) topic model-based, (iv) network based, (v) classification-based, (vi) probabilistic, (vii) collaborative filtering and (viii) hybrid methods. Moreover, the authors used various metrics to evaluate each of these expertise recommendation methods, these include but not limited to, precision, recall, F1-Score, matching set count, Pearson correlation coefficient, area under ROC curve, accuracy by rank. Lastly, the authors suggested that going forward more external data ought to be used to develop more
robust and realistic user models.

In essence, the field of software development in terms of expertise has two broad areas: (1) identify and define an expert and (2) measure, rank and predict an expert. Research in these areas are focused on online social coding platforms, such as GitHub and Stack Overflow. Researchers prefer these social coding platforms, as the data is visible and (relatively) easily available to carry out meaningful research.

On the subject of identifying and defining an expert, the research addressed the detection of developer expertise based on a software developer’s contribution on social coding platforms. Research in this area has been limited and/or addressed in a dissipated manner, i.e., based on one specific issue of expertise. For example, Sindhgatta’s work focused on identifying expertise based on source code and Montandon et al. used a similar approach but only looked at expertise in three JavaScript libraries and frameworks among GitHub users. Bergersen et al. for example only focused on programming, whereas modern day software expert has a more diverse expertise domains. Constantinou and Kapitsaki’s work only considered commit activity on GitHub, i.e, only one source of data, but validate the results from experts’ activity on Stack Overflow. Unlike other research in this area, Baltes and Diehl provided a more comprehensive conceptual theory of software development; also, Li et al. used a mixed-method survey to identify and separate the ‘great’ software developers from the ‘ordinary’ ones.

The second part of the literature review specifically dealt with measuring, ranking and predicting software developer expertise. Most papers reviewed only measured and ranked experts using various techniques, however, there were no papers on predicting software developer expertise. Moreover, the papers that produced ranked software developer expertise were only based on a partial measure or based on one specific aspect of a software developer; a notable exception being, Tian et al. who used a generalized expertise formulas to calculate expertise scores using novel expertise
formulas. Alonso et al. [66] was one of the first papers to measure expertise using rule-based classification of source code on open source projects. There are many tools created by researchers to rank experts, Greene and Fischer [79] created a tool called ‘CVExplorer’ that gives recommendation based on a developer’s activity on GitHub; Huang et al. [64] developed a tool called ‘CPDscorer’ based on developer’s activity on both GitHub and Stack Overflow; thereafter, Badashian et al. [85] created a software developer recommendation system that ranks developers based on their ability to solve bugs in a code, in Stack Overflow, while Zhang et al. [65] combined both SO and GH platforms. In essence, there are different types of recommendation platforms that are available to rank and identify software developers and their expertise; Wang et al. [84] provided a detailed survey of the methods used and their prediction accuracy.

All the above papers were considered in this thesis for: (i) defining an expert and (ii) predicting an expert. The first part of the thesis uses a grounded theory approach to identify and define an expert [3]. The second part learns from the numerous suggestions ([84, 22, 69, 67, 79, 81, 80]) and limitations of other papers ([39, 41, 38, 82]) when developing and proposing a novel technique to predict software developer expertise. This thesis, however, is unique that it uses a multi-label classification, a supervised learning technique, to predict software developer expertise in both GitHub and Stack Overflow.
Chapter 3

Methodology

This chapter describes the methodology followed to understand and predict software developer expertise with specific reference to GitHub and Stack Overflow. In order to reflect this understanding, this chapter is divided into two broad sections: Section 3.1 describes in detail the overall methodology of understanding expertise, and Section 3.2 provides methodology for predicting expertise based on multi-label topic classification technique.

3.1 Understanding Software Developer Expertise

Section 3.1.1 provides a detailed process used to collect data of software developers that are active and common between Stack Overflow and GitHub. Thereafter, these experts that are active and common between the two platforms are surveyed to understand the characteristics and perceptions about experts in the field of software development. The details of this survey design are presented in Section 3.1.2. Moreover, before this survey was sent out to the developers, the survey had to be approved by the Carleton University Research Ethics Board (CUREB); this process is described in Section 3.1.3. Lastly, Section 3.1.5 provides a brief overview about the survey participation, its participants’ characteristics (Section 3.1.5.1) and methodology used to
Figure 2: Mixed methods research approach.

In other words, the research was designed in three phases: In **Phase 1**, an online survey was conducted by sending a questionnaire to developers that were active on both GitHub and Stack Overflow. In **Phase 2**, quantitative analysis was performed on the survey data, while in **Phase 3**, an open coding approach was applied to draw insights and to answer the research questions. The goal of this thesis, is to better understand how developers define, build and maintain their expertise via contributions to two different collaborative platforms. Figure 2 summarizes the overall mixed methods research approach.
3.1.1 Data Collection

For the purposes of this thesis, the Stack Overflow\footnote{https://stackoverflow.com/} and GitHub\footnote{https://github.com/} data were obtained from SOTorrent \footnote{https://empirical-software.engineering/projects/sotorrent/} and GHTorrent \footnote{http://ghtorrent.org/}, respectively. SOTorrent\footnote{https://www.win.tue.nl/mdse/stackoverflow/} is an open source dataset that provides access to official Stack Overflow data dumps. This dataset is built to mine and analyze Stack Overflow posts. Similarly, GHTorrent\footnote{https://www.win.tue.nl/mdse/stackoverflow/} is also an open dataset that collects data from all publicly available GitHub projects. For the purpose of this thesis, the SOTorrent dataset version from December 9th 2018 was used and the GHTorrent data set version from March 1st 2019 was used. Since, the aim of this thesis was to understand cross-platform expertise, the GHTorrent and SOTorrent datasets were linked in order to identify and extract the common users among the two platforms. This thesis used the replication package which has been made publicly available by Vasilescu et al. \cite{13}.

However, after March 2016 GHTorrent stopped storing email addresses of the platform users in order to comply with the EU’s General Data Protection Regulations (GDPR). Norbert Eke \cite{1} has proposed a solution to circumvent this problem; this involved manually checking for matching of user IDs and log-in names between March 2019 and February 2016 data versions. This resulted in 83,350 user accounts that are linked between SO and GH platforms.

Moreover, this thesis is interested in identifying currently active users that are common between the two platforms. Therefore, following Norbert Eke \cite{1}, the dataset was split by timelines of contribution activity by users on both platforms. More specifically, the user profiles were split by those that contributed before January 1st 2016 and labeled them as past users; and, users that were active between 2016 and
2019 were labels as active users. This resulted in 30,774 active user profiles on GitHub and 675 active user profiles on Stack Overflow. Thereafter, from the list of active users on the two platforms, the thesis extracted 423 common users that were active on both platforms.

Furthermore, an “active user” for the purpose of this study was defined as the person who made 10 or more commits in GH and similarly 3 or more comments on Stack Overflow over the last three years (“Active Users”).

### 3.1.2 Survey Design

The thesis started with no preconceived notions about the possible results (i.e., open-minded), and therefore the survey mainly included open-ended questions for the purpose of data collection. Furthermore, since this thesis aims at understanding expertise from a multi-specialty point of view, a specific selection criteria was not defined (i.e., programming language), instead it offered multiple choice to the respondents for providing their input. The full survey can be found in Appendix A.2.

The survey questionnaire was divided into four sections:

1. **Background information** — Ten closed-ended questions (main role, age, gender, education level, years of development experience, etc.) were included in this section. In particular, the thesis was interested in asking the following questions:

   (a) purpose of using GH,

   (b) purpose of using SO.

2. **Expertise levels** — Five closed-ended questions were asked in this section (participants to rate on a 1 to 5 scale, with 5 being an expert):

   (a) expertise in programming languages,

   (b) expertise in frameworks and libraries,
(c) expertise in technologies,
(d) expertise in databases, and
(e) consideration(s) given by respondents when rating their expertise.

3. Expertise discussion — Two open-ended questions were posed to respondents:
   (a) list 3 to 5 important skills of an expert software developer,
   (b) asked if the respondents regularly monitored their skills, if so how?

4. Contribution to GH and SO — Three open-ended questions were posed to respondents in this section:
   (a) how they decide to contribute on GH,
   (b) how they decide to participate on SO,
   (c) top 5 tags on SO they participated and whether these tags reflected their expertise.

3.1.3 CUREB Ethics Application

Prior to sending out the survey to the 423 identified active common respondents, the entire experiment (including the survey questions) have been submitted to the Carleton University Ethics Review Board (CUREB) for their review and approval. This approval process required filling out a detailed protocol form and a consent form. Basically, the CUREB forms were designed to ensure that the participants are not put under any risk and if there is any risk, they wanted to ensure how the researcher would minimize the same. For example, as part of the submission detailed information was provided to CUREB, that included but not limited to the details of the research work, participants, recruitment methods, compensation, data collection, data storage, analysis and data disposal. The CUREB wanted to ensure that there
were no unwarranted risks and that there are no conflicts of interest. The CUREB approval for this project was received in November 2019, after multiple iterations and revisions of the survey questionnaire. A copy of CUREB clearance letter is provided in Appendix A.1.

The next steps, after the survey questionnaire was approved was to send the online survey to the random sample of 423 Active Users. The study utilized Qualtrics, an online survey platform, for the purpose of conducting the survey and collecting data. The Qualtrics, which is a paid service, was a CUREB approved survey platform and in addition it was found that Qualtrics provided real-time analytics of the survey progress.

### 3.1.4 Survey Participation

The first round of survey questionnaire was sent in November 2019 and was available to the respondents for two weeks. A total of 423 emails were sent out inviting the participants; of these, 30 respondents opted out of this survey and 8 emails bounced back. Thus, leaving a final sample of 385 respondents. Of these, 73 respondents started working on the survey (a response rate of 18.9%). Finally, 41 of the 73 respondents completed and submitted the survey, i.e., a submission rate of 56%. It ought to be noted that this thesis only considered fully completed survey responses, even though many of the incomplete ones provided input to many of the questions.

### 3.1.5 Survey Analysis

At the outset, to get a better understand the survey respondents, the participant characteristics and background are reviewed in greater detail. Thereafter, a mixed method approach was applied to analyze the final and fully completed survey results.

[https://www.qualtrics.com//](https://www.qualtrics.com//)
This involved both quantitative and qualitative analysis of the responses to the closed-ended and open-ended included in the survey.

3.1.5.1 Characteristics of Survey Participants

The beginning of the survey consisted of background-related questions. The characteristics of the population of participants are summarized as follows:

1. **Main role**: The majority of the respondents have identified themselves as software developers (33 out of 41, i.e., 80% of the respondents). The other categories of respondents include: software architects (2), team lead (2), consultant (1), researchers (1) and others (2). Figure 3 summarizes the role of survey participants.

2. **Age of respondents**: Most of the respondents are in the age group of 30 to 40 (23, 56.1%), followed by 40+ (14, 34.1%) and the remaining are between the
Figure 4: Age of survey participants.

Figure 4 summarizes the age of survey participants. The age is between 20 and 30 (4, 9.8%).

3. **Gender**: 39 of the 41 respondents are male and the remaining 2 have self-identified themselves as “other”. Figure 5 summarizes the gender of survey participants.

4. **Geographic location**: 38 of the 41 respondents have provided their geographic location; 44.7% (17) of the respondents who answered are from Europe (including UK), 36.8% (14) respondents are from North America, 13.2% (5) are from Australia and 5.3% (2) are from Asia. Figure 6 summarizes the geography of survey participants.

5. **Education**: Of the 41 respondents, 22 (53.7%) have a bachelor degree or equivalent, 9 (22.0%) have a masters degree or equivalent, 7 (17.0%) have a PhD or equivalent and 3 (7.3%) have self-classified themselves under the ‘other’ category. Figure 7 summarizes the education levels of survey participants.
Figure 5: Gender of survey participants.

Figure 6: Geography of survey participants.

6. **Number of Years of Programming Experience**: 90.2% (37 out of the 41 respondents) have noted that they have 10+ years of software development experience; the remaining 4 respondents have noted that they have between 7 and 9 years of software development experience. Figure 8 summarizes the years
of programming experience of survey participants.

7. **Contributions to GH**: Out of the 41 respondents, 28 have noted that they host both private projects and also contribute to public projects; whereas, 12
Figure 9: Contribution to GH by survey participants.

have exclusively noted that they only contribute to public projects and 1 respondent has noted that they only host private projects on GitHub. Figure 9 summarizes the contribution of survey participants on GH.

8. **Usage of SO**: Out of the 41 respondents, 28 have noted that they host both private projects and also contribute to public projects; whereas, 12 have exclusively noted that they only contribute to public projects and 1 respondent has noted that they only host private projects on GitHub. Figure 10 summarizes the usage of SO by survey participants.

### 3.1.5.2 Data Analysis

Quantitative analysis was applied on the multiple-choice responses of the survey (Phase 2), while an open coding approach was used to analyze the the open-ended survey responses (Phase 3). Since there were no predefined groups or categories, an open coding approach [89] was used to build the theory. Upon analyzing the the quotes, themes and categories emerged and evolved during the open coding process.
Initially, “cards” were created, thereby splitting 41 survey responses into 324 individual quotes; these generally corresponded to individual cohesive statements. Thereafter, the cards were grouped into themes and merging themes into categories. For each open-ended question, the analysis proceeded in three steps:

1. Card sorting was performed on the 20% of the cards extracted from the survey responses to identify initial card groups. Independently, the author’s research supervisor, Prof. Olga Baysal also conducted card sorting. The author and her research supervisor then met to compare and discuss their respective identified groups.

2. The two coders performed another independent round, sorting another 20% of the quotes into the groups that were agreed-upon in the previous step. Thereafter, the methodology involved calculating and reporting the inter-coder reliability to ensure the integrity of the card sort process. Furthermore, two of
the most popular reliability coefficients for nominal data were selected: percent agreement and Cohen’s Kappa. Coder reliability is a measure of agreement among multiple coders for how they apply codes to text data. To calculate agreement, the number of cards were counted for each emerged group for both coders and used ReCal2 [90] for calculations. The coders achieved a substantial degree of agreement; on average two coders agreed on the coding of the content in 97% of the time (the average percent agreement varies across the questions and is within the range of 86–100%; while the average Cohen’s Kappa score is 0.6835).

3. The rest of the card sort for each open-ended question (i.e., 60% of the quotes) was performed by both coders together.

A document providing the complete coding and its results are provided in Appendix A.3.

### 3.2 Predicting Software Developer’s Expertise

Section 3.2.1 provides a detailed process used to collect data about software developer’s user profiles and their expertise labels, on both Stack Overflow and GitHub. More specifically, this data is gathered from the LDA modeling exercises by Norbert Eke [1]. Thereafter, Section 3.2.2 highlights the data wrangling and text preprocessing used on this dataset to make it ready for further analysis. Section 3.2.3 provides the process of binarizing the multi-label expertise datasets of SO and GH. Thereafter, Section 3.2.4 provides details about how the model is constructed; this includes feature extraction techniques, multi-label classification models, and the classifiers used in this thesis. Furthermore, this section also provides details about balancing multi-label classification datasets. Lastly, Section 3.2.5 provides the metrics
used to evaluate the various models with an objective to chose the best performing/predicting model.

### 3.2.1 Data Collection

The data collection process for this section is very similar to that for the data collection in Section 3.1, albeit with a few changes. To start with, like in the above section, the SO and GH data were collected from SOTorrent and GHTorrent, respectively. Moreover, the GHTorrent and SOTorrent datasets were linked in order to identify and extract the common users among the two platforms [13] and used the modification suggested by Eke [1]. This resulted in a total of 83,350 users that had accounts on both platforms.

Furthermore, for the purpose of conducting topic modeling experiment, Eke [1] has extracted user profile data on both platforms. This resulted in user profile datasets, one each for SO and GH, that described the complete activity of the 83,350 users on these platforms. More specifically, the user profile dataset for the 83,350 SO users,
included the following text attributes: *badge names* received by a user, the *about me* description from the respective user profiles, *questions* submitted by a user, *answers* given by a user, *titles* and *tags* of posts that a user participated in, and *comments* made by a user to any post. Table 1 provides the attribute name, attributes and their corresponding description of SO user profile dataset.

Similarly, Eke [1] has extracted the user profile dataset for the 83,350 users on GH. The GH profile dataset, that documented the user activity on GH, included the following text attributes: *names*, *labels*, *languages* used, *description of the repository* that a user owns, as well as their *commit* and *pull request comments* posted on GitHub.

---

### Table 1: Stack Overflow user profile extraction.

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Attribute(s)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Badges</td>
<td>Badges.Name</td>
<td>Concatenation of list of badges obtained by the user</td>
</tr>
<tr>
<td>About Me</td>
<td>Users.AboutMe</td>
<td>Stack Overflow user profile’s about me description</td>
</tr>
<tr>
<td>Post Answer</td>
<td>Posts.Body, AcceptedAnswerId</td>
<td>The user’s each individual answer concatenated with the question it is related to</td>
</tr>
<tr>
<td>Post Question</td>
<td>Posts.Body</td>
<td>The user’s each individual question concatenated with the accepted answer it is related to</td>
</tr>
<tr>
<td>Title and Tags for Questions</td>
<td>Posts.Tags, Posts.Title</td>
<td>Concatenation of post tags and title for each question that the user asked</td>
</tr>
<tr>
<td>Title and Tags for Answers</td>
<td>Posts.Title, Posts.Tags</td>
<td>Concatenation of post tags and title for each answer that the user provided</td>
</tr>
<tr>
<td>Comments</td>
<td>Comments.Text, Posts.Body, Posts.Title</td>
<td>Concatenation of the user’s each individual comment and the post (question or answer) it is related to</td>
</tr>
</tbody>
</table>

---

7Extracted verbatim from Eke’s thesis. Table credit to [1]
Table 2 provides the attribute name, attributes and their corresponding description of GH user profile dataset.

Table 2: GitHub user profile extraction.

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Attribute(s)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project Name, Description and Metadata</td>
<td>Projects.name, description, language, Repo-Labels.name</td>
<td>Description of each user’s project together with the repository’s name, description, languages used, repository labels it contains.</td>
</tr>
<tr>
<td>Commit-Comments</td>
<td>Commit-Comments.name</td>
<td>List of user’s commit comments.</td>
</tr>
<tr>
<td>Code-review Comments</td>
<td>Pull-Request-Comments.name</td>
<td>List of user’s code review (pull request) comments.</td>
</tr>
</tbody>
</table>

Thereafter, based on the above user profile datasets, Eke has created a complex attribute that concatenated all the text attributes of each of the 83,350 users, on both GH and SO platforms. Thus, he created a *full activity* attribute for each user on both platforms. Moreover, Eke has used Latent Dirichlet Allocation (LDA) modelling to create expertise labels for each of the 83,350 users on both the platforms. Table 3 and Table 4 provide a list of topics obtained from the SO and GH datasets, using the LDA model, respectively.

In essence, from Eke’s datasets, for each of the two platforms, the following were obtained: (i) the *user profile* datasets along with the full-activity attribute and (ii) the *expertise multi-label* datasets. These four (4) datasets became the ‘raw input’ data for this thesis. And, in order to proceed with the prediction of expertise using topic classification modeling, the two user profile datasets were further cleaned and pre-processed; also, the remaining two (2) expertise multi-label datasets were binarized.

---

8Extracted verbatim from Eke’s thesis. Table credit to [1](#).
9Table 3 and Table 4 are credited to [1](#).
Table 3: Labels given to topics of the best LDA model trained on Stack Overflow.

<table>
<thead>
<tr>
<th>Topic</th>
<th>Given Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Data Visualization</td>
</tr>
<tr>
<td>1</td>
<td>Front-end Web Development</td>
</tr>
<tr>
<td>2</td>
<td>Python</td>
</tr>
<tr>
<td>3</td>
<td>Algorithms</td>
</tr>
<tr>
<td>4</td>
<td>HTTP</td>
</tr>
<tr>
<td>5</td>
<td>OOP</td>
</tr>
<tr>
<td>6</td>
<td>Server-client</td>
</tr>
<tr>
<td>7</td>
<td>Java</td>
</tr>
<tr>
<td>8</td>
<td>Data Types</td>
</tr>
<tr>
<td>9</td>
<td>C/C++</td>
</tr>
<tr>
<td>10</td>
<td>Version Control</td>
</tr>
<tr>
<td>11</td>
<td>Data Management</td>
</tr>
<tr>
<td>12</td>
<td>Java Build Tools</td>
</tr>
<tr>
<td>13</td>
<td>Ruby</td>
</tr>
<tr>
<td>14</td>
<td>iOS Development</td>
</tr>
<tr>
<td>15</td>
<td>Parallel Programming</td>
</tr>
<tr>
<td>16</td>
<td>Android</td>
</tr>
<tr>
<td>17</td>
<td>VIM</td>
</tr>
<tr>
<td>18</td>
<td>PHP</td>
</tr>
<tr>
<td>19</td>
<td>Data Searching</td>
</tr>
<tr>
<td>20</td>
<td>SSL + Game Development</td>
</tr>
<tr>
<td>21</td>
<td>Encryption</td>
</tr>
<tr>
<td>22</td>
<td>JavaScript Programming</td>
</tr>
<tr>
<td>23</td>
<td>File Management</td>
</tr>
<tr>
<td>24</td>
<td>UI</td>
</tr>
<tr>
<td>25</td>
<td>Web Layout</td>
</tr>
<tr>
<td>26</td>
<td>Database</td>
</tr>
<tr>
<td>27</td>
<td>Angular</td>
</tr>
<tr>
<td>28</td>
<td>Unix</td>
</tr>
<tr>
<td>29</td>
<td>Distance Calculation</td>
</tr>
<tr>
<td>30</td>
<td>Web Graphics</td>
</tr>
</tbody>
</table>

Table 4: Labels given to topics of the best LDA model trained on GitHub.

<table>
<thead>
<tr>
<th>Topic</th>
<th>Given Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Web Development</td>
</tr>
<tr>
<td>1</td>
<td>Back-end Development</td>
</tr>
<tr>
<td>2</td>
<td>C/C++</td>
</tr>
<tr>
<td>3</td>
<td>Web layout</td>
</tr>
<tr>
<td>4</td>
<td>Ruby</td>
</tr>
<tr>
<td>5</td>
<td>Data Science</td>
</tr>
<tr>
<td>6</td>
<td>Real time App Development</td>
</tr>
<tr>
<td>7</td>
<td>Unix</td>
</tr>
<tr>
<td>8</td>
<td>Rust</td>
</tr>
<tr>
<td>9</td>
<td>PHP</td>
</tr>
<tr>
<td>10</td>
<td>Scala/Haskell</td>
</tr>
<tr>
<td>11</td>
<td>JVM</td>
</tr>
<tr>
<td>12</td>
<td>Front-end Web Development</td>
</tr>
<tr>
<td>13</td>
<td>Mobile App Development</td>
</tr>
<tr>
<td>14</td>
<td>EMACS</td>
</tr>
</tbody>
</table>


3.2.2 Data Cleaning and Text Pre-processing

To start with, the two user profile datasets on SO and GH were cleaned and pre-processed. In this regards, the SO dataset was loaded into a dataframe for easy manipulation and exploration using the pandas package. Then, the duplicate instances were identified and dropped for further processing. Next, the numeric features like ‘GH_UserId’, ‘active’, ‘unifiedId’, ‘SO_UserId’ were dropped as they are not needed for the analysis. Since all the individual text features like, ‘aboutMe’, ‘commentData’, ‘postQuestions’, ‘postAnswers’, were concatenated and appear as a single attribute called ‘fullactivity’, the individual text attributes are also dropped from the dataset. Finally, all the null values in the data set are removed. This resulted in the sample frame of 12,898 SO users. This cleaned SO dataset has only two attributes: (i) unified ID and (ii) full activity of each user.

Thereafter, the GH dataset was loaded into a dataframe for easy manipulation and exploration using the pandas package. Then, the duplicate instances were identified and dropped for the further procedure. Next, the numeric features like ‘SO_UserId’, ‘active’, ‘unifiedId’, ‘GH_UserId’ were dropped as they are not needed for the analysis. Since all the individual text features like, ‘repoNames’, ‘repoLabels’, ‘repoDescriptions’, ‘repoLanguages’, ‘codeReviewComments’, ‘commitComments’ were already concatenated and taken as a single feature called ‘fullactivity’, they are also dropped from the dataset. Finally, all the null values in the data set are removed. This resulted in the sample frame of 13,381 GH users. This cleaned GH dataset has only two attributes: (i) unified ID and (ii) full activity of each user.

Thereafter, the next step involved ‘text pre-processing’; an important step of this project, as the characters, words, and sentences identified at this stage become the vital parameters that are passed to all further processing stages. This ‘text pre-processing’ is carried out on both SO and GH datasets that are obtained after cleanup.
in the above step. More specifically, the following steps are carried out in the process of text cleaning on the “fullactivity” column.

1. **Decode HTML entities to Text**: BeautifulSoup, a Python library for HTML parser, is used to decode the entities and extract the meaningful text for further analysis.

2. **Convert the text to lower case**: It is important to convert the whole text to lower case to avoid duplicate. The process of text analysis is usually case-sensitive.

3. **Replace special characters with space**: Python regular expressions package is used to recognize special characters; a list of individual characters that are each considered a special character, include: ‘[/(){}\|@.;:”0-9a-z#+]’; and, these are replaced with a space. These special characters are like garbage which will affect the accuracy measures while modelling.

4. **Stop word removal**: Many words in documents that occur very frequently are meaningless as they are used to join words together in a sentence. It is commonly understood that stop words do not contribute to the context or content of textual documents. Due to their high frequency of occurrence, their presence in text mining presents an obstacle in understanding the content of the documents. Stop words accounts 25-30% of total word counts in a SO users document. To improve the efficiency of the whole model, the stop words are removed from the corpus using **nltk** package.

5. **Stemming**: Stemming is the process of coalescing the variant forms of a word into a common representation, the stem. This is a widely used procedure in text processing for information retrieval (IR) based on the assumption that posing a query with the term presenting implies an interest in documents containing
the words presentation and presented. In this project stemming to the whole corpus is applied using nltk SnowBallStemmer package.

6. **Tokenization**: This is the process of breaking a stream of text into words, phrases, symbols, or other meaningful elements called tokens. The aim of the tokenization is the exploration of the words in a sentence. The list of tokens becomes input for further processing such as parsing or text mining. In this project each instance is tokenized to words using nltk textblob package.

Figure 12 and Figure 13 provide a snapshot of the top 100 frequently occurring words in SO and GH datasets, respectively. This frequency distribution of words shows that the text pre-processing was successful as mostly only words related to software engineering domain were retained in the SO and GH corpus. This also provides assurance that the topic classification task on either platform would yield appropriate results.

### 3.2.3 Multi-label Binarization

Next, the two (2) expertise multi-label datasets of SO and GH were binarized, using the multi-label binarizer, a scikit-learn class that returns an Numpy array from an input of categorical data. The multi-label binarizer encodes the data into binary variables; giving it a value of ‘1’ if the particular label exists and a value of ‘0’ if a label does not exist.

Finally, after binarizing, the cleaned and pre-processed SO dataset was concatenated with the multi-label binary values, thus forming the final SO input data set for the purpose of this thesis. Similarly, a final GH input dataset was created for the purpose of this thesis. In essence, this thesis has one (1) final input dataset corresponding to each platform.
Figure 12: Top 100 frequently occurring words in SO corpus.
Figure 13: Top 100 frequently occurring words in GH corpus.
3.2.4 Model Construction

The final datasets obtained for SO and GH, are divided into training data and test data with the intention of using the train data to find the parameters of the particular model being used (fitting the model on the training data) and then applying this to the test data to determine the model’s performance and to draw conclusions about its predictive capability. For the sake of convenience, from this point on till the end of this chapter, the term ‘dataset’ would be used as a generic term to include the final datasets of both SO and GH.

3.2.4.1 Feature Extraction Models

Now after splitting data, the ‘fullactivity’ column text is converted to vectors in both the ‘test’ and ‘train’ datasets using word vectorization methodology. The test and train split vectorized matrices are compressed using sparse matrix which is a memory efficient data structure that enables to store large matrices with very few non-zero elements. In this thesis, two word vectorization methodologies are employed for feature extraction: (i) TFIDF and (ii) Word2Vec. The aim of using these techniques is to compare the vectorization that performs better in terms of its predictive ability. Moreover, applying word vectorizers increases the dimensionality of the data which leads to choosing of linear models for predictive analysis.

**TFIDF:** Tf-idf means “term frequency–inverse document frequency”, is a numerical/statistical measure that is intended to reflect and evaluate the relevance and importance of a word to a document, within a collection of documents. TFIDF achieves this by multiplying: (i) the number of times a word appears in a document and (ii) the inverse document frequency of the word across set of documents. Moreover, TFIDF is mostly important in automated text analysis.
However, this word vectorization methodology has some limitations:

1. It computes document similarities directly in the word-count space, thus making it slow for documents with large corpus of text.

2. It does not consider the semantic similarities between words, thus possibly reducing the predictive performance.

In this thesis, scikit-learn implementation of TFIDF vectorizer is used. Furthermore, the below parameters are empirically validated and considered:

1. min_df: The object type is either float or int with a default of ‘1’. When building the vocabulary, it ignores terms that have a document frequency strictly lower than the given threshold. For the purpose of this thesis, a threshold of ‘0.00009’ was considered.

2. max_features: The object type is int and it has a default of ‘None’. If not ‘None’, then the word vectorizer builds a vocabulary that only consider the top max_features that are ordered by term frequency across the corpus. Herewith, owing to computational complexity, 40,000 top ranked features ordered by term frequency were considered.

3. ngram_range: This is another important feature that is considered in word vectorizing. For example, the ngram_range of (1, 3) means unigram, bigram and trigram. And, for the purpose of this thesis, an n-gram range of (1,3) is considered.

Word2Vec: Since the aim of this thesis is to achieve a better predictive performance, Word2Vec, another word vectorizer methodology was applied on the same dataset [26]. This word vectorizer methodology uses neural network models to learn word associations from a large corpus of text. Once the corpus is trained, the
Word2Vec model can detect synonyms and it can even recommend additional words for a partial sentence. This is an advantage that Word2Vec has over TFIDF. Moreover, as the name suggests, Word2Vec represents each unique word with a list of numbers called vectors. One of the advantages of this model is that a simple mathematical function (e.g., the cosine similarity between the vectors) could be used to indicate the level of semantic similarity between the words represented by those vectors. Furthermore, Word2Vec is a group of related models that are used to produce word embeddings. For the purpose of this thesis a pre-trained word embeddings from software engineering domain were used to convert the text to word vectors based on the semantic similarity [24]. And these word embeddings are loaded into the model using Gensim\textsuperscript{10}, an open source library used for natural language processing.

3.2.4.2 Multi-label Topic Classification Model

After vectorizing the dataset, the next step involves applying topic classification models on the multi-label dataset. ‘Topic Classification’, a supervised learning technique, categorizes a data object into one of the several pre-determined classes. In machine learning, there are three (3) types of classification problems:

1. **Binary classification**: If the total number of disjoint classes are two, then the problem is identified as binary classification;

2. **Multi-class classification**: If the disjoint labels are more than two, then the problem is a multi-class classification; and

3. **Multi-label classification**: The goal here is to learn from a set of instances, where each instance belongs to one or more classes from a set of disjoint class labels.

The focus of the second part of this thesis is to predict the ‘expertise’ of an individual based on their participation on online coding platforms. Moreover, it is very

\textsuperscript{10}https://pypi.org/project/gensim/
much likely that each individual participant could be an expert in multiple fields or domains of software engineering. Therefore, in this context, each individual would have multiple labels associated with them, i.e., each instance has many labels. Therefore, applying the multi-label classification models to predict the labels of experts is the best and possibly the only solution available.

In the machine learning literature, multi-label classification problems are broadly divided into three (3) methods [91] (summarized in Figure 14):

1. **Problem Transformation Method.** There are two models in this method:

   (a) **Binary Relevance:** In this method each target variable \((y_1, y_2, \ldots, y_n)\) is treated independently and the model is reduced to \(n\) classification problems. However, this method ignores the correlation between labels by treating each label independently which is the main drawback.

   (b) **Label Powerset:** In this the problem is transformed into multi-class classification problem; the target variables \((y_1, y_2, \ldots, y_n)\) are combined and each combination is treated as a unique class. This method will produce many classes. Though this method takes label correlation into account, it leads to imbalanced dataset with multiple classes associated. In other words, the main drawback of this model is the number of classes or subsets could be exponential if the number of labels increase. This would lead to sparse and imbalanced dataset [92].

2. **Algorithm Adaptation (ML\(k\)NN):** Multi-label K Nearest Neighbours uses k-Nearest Neighbors to find nearest examples to a test class and uses Bayesian inference to predict labels. This is a distance-based method and works well when there is a relationship between distance and labels.
3. **Ensemble Method (Classifier Chain)**: Classifier chains are almost similar to binary relevance; however, it overcomes the disadvantage of binary relevance by taking label dependencies into account. The features \((x_1, x_2, \ldots, x_m)\) are initially used to predict \(y_1\). Next \((x_1, x_2, \ldots, x_m, y_1)\) is used to predict \(y_2\). At the nth step, \((x_1, x_2, \ldots, x_m, y_1, \ldots, y_{n-1})\) predicts \(y_n\). Furthermore, an ensemble learning method called ‘bagging’ is applied wherein several independent models are fit and their predictions are averaged in order to obtain a model with low variance. The other possible ensemble techniques, though not considered in this thesis, include: boosting and stacking. Also, the ordering in which the labels are predicted can be determined by the researcher and can greatly influence the results.

![Diagram of multi-label classification](image)

Figure 14: Multi-label classification.

Thereafter, the thesis used the following three base algorithms for each of the models, sans MLkNN:

1. **Logistic Regression** (**LR**): A machine learning classification algorithm that is used to predict the probability of a categorical dependent variable. In logistic regression, the dependent variable is a binary variable that contains data coded as 1 (yes, success, etc.) or 0 (no, failure, etc.). In other words, the logistic
regression model predicts $P(Y=1)$ as a function of $X$.

The followings assumptions should be considered before applying the Logistic Regression:

(a) There should be no missing values in the dataset.

(b) The target feature or the variable must be binary (only two values) or the ordinal (categorical variable with the ordered values).

(c) All the other data variables should not have any relationship. It means they are independent and have no correlation between them.

(d) The data shall not contain values less than 50 observations for reliable results.

2. **Gaussian Naïve Bayes (GNB)**: This is a probabilistic machine learning model that is based on Bayes Theorem and is used for document classification problems. The features/predictors used by the classifier indicate the frequency of the words within the document. Moreover, when the predictors take up a continuous value (and not discrete values), it is assumed that these values are sampled from a Gaussian distribution.

3. **Linear Support Vector Machine (linear-SVM)**: This algorithm classifies the data points by finding a hyperplane in an $N$-dimensional space that maximizes the separation between the two classes. The vectors that define the hyperplane are the support vectors. Also, maximizing the separation between hyperplanes provides greater confidence in classifying the datapoints on the 'test' dataset and hence improves 'prediction' power of the model.

In summary, the thesis has essentially used a total of ten (10) classifiers on the datasets: binary relevance with three base algorithms (3); label power set with three base algorithms (3); classifier chain with three base algorithms (3) and MLkNN (1).
3.2.4.3 Balancing Multi-label Classification

In real world classification problems the number of instances associated with one class are way lesser than the other class thus leading to the problem of data imbalance. This in turn greatly affects the machine learning algorithm performance. This is the same problem that has occurred in the case of expertise multi-label classification where the labels are unevenly distributed. Thus, the thesis attempts to apply MLSMOTE \cite{94}, a multi-label balancing technique to the best performing feature model.

MLSMOTE, a most popular and effective data augmentation techniques, in order to handle data imbalance in multi-label classification. This technique, as the name suggest, is an extension or variant of the classic SMOTE (Synthetic Minority Over-sampling Technique) \cite{95}. The challenge with applying SMOTE to a multi-label classification model is that it augments the majority class labels alongside the minority class labels, as each instance of the data is associated with various labels. MLSMOTE, a variant of SMOTE, attempts to overcome this problem by selecting the appropriate data points, especially the ones which primarily have minority labels. Then, it also generates new data for the feature vectors corresponding to those label data. In other words, MLSMOTE could improve the performance by avoiding the over-fitting problem and/or it could improve the predictive performance of the minority classes. The following steps briefly describe the algorithm behind this technique:

- Selects data to over-sample, that is data with minority class labels;
- Chooses an instance of the above data;
- Finds the k-nearest neighbors of the above data instance;
- Chooses a random data point which is the k-nearest neighbors of the selected data point and makes a synthetic data point anywhere on the line joining both these points;
Repeats the process until data is balanced.

### 3.2.5 Evaluation Metrics

In traditional multi-class problems, accuracy is the most common evaluation criteria. Additionally, there exists a set of standard evaluation metrics that includes precision, recall, F1-Score, and ROC area defined for single label multi-class classification problems. However, in multi-label classification, predictions for an instance is a set of labels and, therefore, the prediction can be fully correct, partially correct or fully incorrect. Therefore, none of these existing evaluation metrics have the ability to capture such notion in its entirety. This makes evaluation of a multi-label classifier more challenging than evaluation of a single label classifier and these metrics must be appropriately adapted to perform model evaluation \[96\]. Notwithstanding, the most important evaluation metric for multi-label models is *Hamming Loss*.

**Accuracy (A):** Accuracy for each instance is defined as the proportion of the predicted correct labels to the total number (predicted and actual) of labels for that instance. Overall accuracy is the average across all instances.

\[
A = \frac{1}{n} \sum_{i=1}^{n} \frac{|Y_i \cap Z_i|}{|Y_i \cup Z_i|} \quad (1)
\]

**Precision (P):** Precision is the proportion of predicted correct labels to the total number of actual labels, averaged over all instances.

\[
P = \frac{1}{n} \sum_{i=1}^{n} \frac{|Y_i \cap Z_i|}{|Z_i|} \quad (2)
\]
Recall \((R)\): Recall is the proportion of predicted correct labels to the total number of predicted labels, averaged over all instances.

\[
P = \frac{1}{n} \sum_{i=1}^{n} \frac{|Y_i \cap Z_i|}{|Y_i|}
\]

\((3)\)

**F1-Measure \((F)\):** F1-measure is the harmonic mean of precision and recall.

\[
F_1 = \frac{1}{n} \sum_{i=1}^{n} \frac{2|Y_i \cap Z_i|}{|Y_i| + |Z_i|}
\]

\((4)\)

As in single label multi-class classification, the higher the value of accuracy, precision, recall and F1-Score, the better the performance of the learning algorithm.

**Hamming Loss \((HL)\):** Hamming Loss is the fraction of incorrectly predicted class labels to the total number of actual labels. In case of all the correctly classified tags, Hamming Loss will be a perfect 0.

\[
HL = \frac{1}{kn} \sum_{i=1}^{n} \sum_{l=1}^{k} [I(l \in Z_i \land l \notin Y_i) + I(l \notin Z_i \land l \in Y_i)]
\]

\((5)\)

where, ‘\(I\)’ is the indicator function and ‘\(T\)’ is a multi-label dataset consisting \(n\) multi-label examples \((x_i; Y_i), 1 \leq i \leq n\), \(x_i \in x'\ y_i \in y = 0, 1^k\); with a label-set \(l\); \(|l| = k\). And, \(h\) is a multi-label classifier and \(z_i = h(x_i) = 0, 1^k\) is the set of label memberships predicted by \(h\) for the sample \(x_i\).
Chapter 4

Results

In this chapter, the results of experiments conducted are presented with an aim to provide answers to all research questions from Section 1.3. This chapter is broadly divided into two sections, Section 4.1 addresses results achieved on the task of understanding expertise and Section 4.2 provides insights into predicting software developer expertise. More specifically, the first section presents and analyzes the results of the first three RQs: (RQ1) defining an expert, (RQ2) the driving factors behind contribution to social coding platforms, and (RQ3) challenges faced by contributors. And, the second section addresses the remaining four RQs: (RQ4) best predicting vector representation technique, (RQ5) predictive performance of classifiers, (RQ6) effects of balancing the multi-label datasets, and (RQ7) factors impacting predictive performance.

4.1 Understanding Software Developer Expertise

This section presents outcomes of the open coding approach and the corresponding concept categories that emerged therefrom; thereafter the results and findings for each of the three (3) research questions are presented. In other words, an open coding technique was applied on the 324 individual statements. During this process, 46 main
categories emerged. Each identified category consisted of between one and forty-four comments or quotes. Further, these categories were broadly grouped into four concept categories: 1) skills of an expert, 2) contribution to GH, 3) contribution to SO, and 4) challenges faced by developers in contributing to collaborative platforms. Table 5 presents these categories in detail reporting the number of quotes, the number of respondents, and the totals. Each of these concept categories consists of a minimum of five to a maximum twenty three categories.

The four concept categories that have emerged during the card sort do not necessarily directly correspond to the tasks performed by experts on the online collaborative platforms. Instead, these categories are a combination of actions (i.e., behavior) and mental activities (e.g., motivation) performed by experts contributing to the online collaborative platforms under consideration (i.e., GH and SO).

1. Defining features of an expert. This concept category identifies the essential skills and competencies of an expert, as well as the characteristics and behavior that an expert might possess.

2. Factors driving developers to contribute to GH. This category offers insights into why developers decide to contribute to GitHub and what factors facilitate their activity on GitHub.

3. Factors driving experts to contribute to SO. This category highlights the details on the developer motivation behind their contribution to Stack Overflow.

4. Challenges faced by experts in contributing to online collaborative platforms. This category addresses the challenges developers face when contributing to each platform. Such challenges can be addressed by improving current platforms to increase developer motivation and engagement on these platforms. While all 41 participants answered a “yes/no” question related to challenges, only 20 of them
Table 5: Overview of concept categories.

<table>
<thead>
<tr>
<th>Concept Category</th>
<th>Participants</th>
<th>Quotes</th>
<th>Categories</th>
</tr>
</thead>
<tbody>
<tr>
<td>Experts’ skills</td>
<td>41</td>
<td>176</td>
<td>23</td>
</tr>
<tr>
<td>Contribution in GH</td>
<td>41</td>
<td>75</td>
<td>9</td>
</tr>
<tr>
<td>Contribution in SO</td>
<td>41</td>
<td>50</td>
<td>9</td>
</tr>
<tr>
<td>Challenges</td>
<td>20</td>
<td>23</td>
<td>5</td>
</tr>
</tbody>
</table>

provided a rationale for their answer (as reflected in Table 5 for “Challenges” category).

4.1.1 Defining an Expert

RQ1: How do developers define an expert?

Since, both quantitative and qualitative analysis methods were used to analyze the survey data, below are the results for each of these methods, respectively.

The software development experts were asked four closed-ended questions that asked them to rank their expertise in various areas of software development (programming language, frameworks & libraries, technologies and databases). The respondents were given a list of options in each of these modules and were asked to self-select their expertise level on a scale of 1 to 5 (with 1 being “no knowledge” and 5 being an “expert”). Also, there was one closed-ended question that asked the considerations given by the respondents in rating their expertise in the areas of software development. The respondents were given five options and were also given a freedom to respond outside of these choices. These questions aimed at uncovering whether the respondents would consider their professional experience, their depth of knowledge, the size of the projects for measuring their performance against other developers. The respondents could choose more than one option on this question.

The first question probed on expertise in programming languages. Out of all
the programming languages, JavaScript turned out to be most popular among respondents and more than 70% have self-identified themselves as expert (level 5) or proficient (level 4). This was followed by SQL (50% combined for both level 5 and level 4, due to low response numbers), C/C++ (49%), and then Python (44%). It was noted that C/C++ were on the top of this table; this could possibly be because these were popular languages a few years ago and given that about 90% of the respondents are above 30 years, it is likely that these respondents might have previously acquired expertise in these languages [97].

The second question pertained to expertise in frameworks and libraries. Surprisingly, the response rate was low and also, they have mainly marked as beginner (level 2) or intermediate (level 3). When the responses for level 4 (proficient) and level 5 (expert) were combined, it was found that about 29% of the respondents have stated they are experts in React (Facebook) and NodeJS.

The third question was about expertise in technologies. The response was even lower here. Only two respondents noted that they were experts in blockchain. Thereafter, about six respondents have noted that they are proficient in IoT. Moreover, two of the 41 respondents claimed that they were proficient in deep learning, 4 claimed proficiency in machine learning and 3 in computer vision. This is indeed a surprising result, especially given that ML, deep learning and computer vision have been recently gaining a momentum by becoming the hot trend in expertise demand.

The fourth question was on database expertise. It was observed that about 29% of the respondents have claimed expertise in PostgreSQL, followed by 22% claiming expertise in MySQL and SQLite, each.

The last question (multiple choice) asked the respondents to answer the factors they considered to be important to them when they were answering the above questions about expertise. Depth of knowledge (as supported by 26% of the responses) is the top factor that developers believe to influence their answers to the above four
Figure 15: Expertise determinants.

questions on expertise. The other factors influencing their opinions are: breadth of general programming (22%), professional experience (21%), performance compared to others (15%) and project size (14%). Figure 15 presents the overall summary of these results.

Thereafter, the survey probes the developers themselves to find the necessary and sufficient conditions that define an “expert”. More specifically, the aim is to better understand if knowledge and experience alone are sufficient or if a requisite amount of soft skills are needed for a software developer to become an expert. Moreover, the order of importance of these three factors was further investigated to understand if there is any normative difference between the technical and social skills.

Technical skills (number of total participants (P): 21; number of total quotes (Q): 44 as shown in Table 6), broad knowledge (P:7; Q:8) and domain knowledge (P:2; Q:2) are perceived to be the features that characterize expertise. On the other
hand, experience could be captured by features such as planning & organization (P:9; Q:11), experience (P:2; Q:4), and vision (P:1; Q:1). Lastly, communication skills (P:15; Q:15), analytical thinking (P:24; Q:26), behavior which includes tenacity and perseverance (P:9; Q:17) are some of the features that characterize soft skills of an expert. Effective and clear communication is a key skill for an expert software developer and was emphasized by the respondents. One respondent added the need for clarity and “clear communication” (P3Q21P4, where P3Q21P4 is the “code” we assign to each individual statement meaning Participant 3, Question 21, Phrase 4), while another noted the need for “soft communication skills” (P31Q21P4). The other key consideration is analytical thinking, which is a soft skill that could be honed with experience. Analytical thinking not only requires an expert to “think beyond technology” (P11Q21P3), but also requires that they think abstractly about complex problems (P38Q21P2) and also have an ability of “…breaking down complex problems into simpler problems” (P40Q21P1). Behavior or attitude has been identified as another critical factor in identifying an expert. Some of the features identified are, being stubborn (P5Q21P3), patience (P13Q21P3, P27Q21P3, P38Q21P4), tenacity (P13Q21P4, P27Q21P4), curiosity (P27Q21P3) and “being empathetic about the needs of other people” (P41Q21P1). Furthermore, based on the detailed responses, Table 6 reports the features, emerged categories and their response rate (i.e., number of participants and total number of quotes) for each of these three factors.

Moreover, the survey asked respondents to list the important factors in an order of importance. It is noted that even in that ordering mostly the soft skills appeared quite prominently in the beginning of a response, i.e., they were more important features of an expert, as per the respondents. Furthermore, from Table 6 it is evident that the most important skills that a software development expert ought to possess are perceived to be soft skills, skills that are both necessary and sufficient conditions to become an expert.
Table 6: Features of software developer expertise.

<table>
<thead>
<tr>
<th>Expertise</th>
<th>Participants</th>
<th>Quotes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Technical</td>
<td>21</td>
<td>44</td>
</tr>
<tr>
<td>Domain knowledge</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Broad knowledge</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>30</strong></td>
<td><strong>54</strong></td>
</tr>
<tr>
<td>Experience</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Experience</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>Critical thinking</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Vision</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Requirements</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>Teamwork</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Teaching skills</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>17</strong></td>
<td><strong>20</strong></td>
</tr>
<tr>
<td>Soft Skills</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Planning &amp; organization</td>
<td>9</td>
<td>11</td>
</tr>
<tr>
<td>Analytical thinking</td>
<td>24</td>
<td>26</td>
</tr>
<tr>
<td>Creativity</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Understanding</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Behavior</td>
<td>9</td>
<td>17</td>
</tr>
<tr>
<td>Communication skills</td>
<td>15</td>
<td>15</td>
</tr>
<tr>
<td>Searching &amp; seeking help</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>66</strong></td>
<td><strong>78</strong></td>
</tr>
<tr>
<td>Monitoring Skills</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Self tracking</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Dashboards</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>Testing tools</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>Online tools</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>Time tracking tools</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>20</strong></td>
<td><strong>24</strong></td>
</tr>
</tbody>
</table>
Within this same broad area, the respondents were asked about the ways they *monitor their own activities* on a regular basis (using software metrics or other statistics) and the specific mechanism that they use to measure their software development activities. The majority of the respondents (61%) have noted that they do not monitor their activities and only 34% have responded affirmatively on this question. Three main features that emerged from this category are self tracking (P:3; Q:3), testing (P:7; Q:6), online collaborative tools (P:4; Q:5), time tracking tools (P:3; Q:3), and dashboards (P:5; Q:7).

In summary, the results for this research question indicate that:

- JavaScript is perceived to be the leading programming language within a software development expertise, in line with the findings of Georgiou et al. 98;

- Surprisingly, the technical expertise diversification is lower than what one would expect;

- Knowledge (48% of the responses, calculated as the sum of depth of knowledge (26%) and breadth of general programming (22%)) and experience (50%) are considered to be the key “expert” characteristics. This finding is in line with the high-level concept (relationships) of the expertise theory by Baltes and Diehl 3.

- Lastly, based on the open-ended questions posed, it was evident that a software developer’s behavior traits, soft skills and their communication skills are key factors that an expert posses.

**Answer to RQ1**: Soft skills together with adequate knowledge and experience are the key characteristics that make a software developer an expert. In other words, soft skills are as important as knowledge and experience for software developers to have and develop to be considered an expert in the field.
4.1.2 Driving Factors Behind Contribution to Social Coding Platforms

RQ2: What factors drive developers to contribute to GH and SO?

GitHub allows developers to contribute to projects in multiple ways: submitting a bug report, a pull request, engaging in the discussion on project’s features, etc. This research question aims to understand how developers participate in projects (private vs. public contributions), as well as what motivates developers to contribute to GitHub. In other words, this question seeks to provide insights into the possible drivers behind developers’ contributions to GitHub.

To start with, the closed-ended question enquired how the respondents used GH, i.e., either contributing to public or private projects. The majority of respondents (28 out of 41 or 68% of the participants) noted that they use it for both private and public projects; while 29.7% contributed only to public projects, and only 2.4% used GitHub for hosting private projects (as shown in Figure 16).
Upon establishing how developers use GH, the drivers that motivated the developers to contribute on GitHub were identified (Table 7). Thereafter, the drivers were classified into: (1) personal drivers and (2) professional drivers. The personal drivers included contributing for hobby/fun, catering to self needs, and providing inputs due to altruistic tendencies of developers. On the other hand, the professional drivers were more related to job-related contributions to GitHub, including organizational culture, job, expertise and contributions to open source community.

Furthermore, it was observed that experts contribute to GitHub more due to personal rather than professional motivation. Among the personal drivers, self needs is determined to be the key driver behind developers’ contributions to GitHub. And, most of the participants mention that they started contributing because they were trying to solve a problem or a challenge they were going to resolve; some responses on contribution included, “if there’s a bug I ran into” (P1Q25P2), “contribute bug fixes to software I use” (P3Q25P2), “I need to fix something broken that’s blocking me” (P10Q25P2) and “I scratch my own itches” (P22Q25P1). Self needs, however, is not a prime driver for contribution to Stack Overflow, because of fundamental difference between the two collaborative platforms, GitHub allows for active participation, while Stack Overflow is more reactive contribution based on specific questions and answers.

The other important personal drivers that motivate developers to contribute to GitHub, include, hobby/fun and helping. Some of the quotes of the former include, “...some of it is hobby though” (P4Q25P2) and “personal interest, if I spend my time for free, I want to enjoy it” (P11Q25P1). And, on helping others, the responses were as follows: “I contribute to new projects I think would be useful to others” (P6Q25P2), “I contribute because I care about what’s being made” (P10Q25P1), while another developer noted that “they look like they might be useful” (P18Q25P1).

On the other hand, among the professional drivers it was found that “expertise” is the most critical driver. One developer mentions that “mostly contribute to libraries
Table 7: Contribution to GitHub.

<table>
<thead>
<tr>
<th>Personal Drivers</th>
<th>Participants</th>
<th>Quotes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hobby/fun</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>Self needs</td>
<td>27</td>
<td>34</td>
</tr>
<tr>
<td>Helping</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>Total</td>
<td>39</td>
<td>47</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Professional Drivers</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Organizational culture</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Job</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Expertise</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>Open source community</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Total</td>
<td>13</td>
<td>14</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Challenges</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Issue complexity</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>Lack of time</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>Total</td>
<td>14</td>
<td>14</td>
</tr>
</tbody>
</table>

used in work-related projects” (P36Q25P2) as a response about their nature of contribution in GitHub. Others contribute because it is their job (P19Q25P1, P4Q25P1) or to rectify existing bugs (P34Q25P3, P32Q25P1, P10Q25P3) or it is compatible with their skills (P7Q25P1). The other professional drivers include organizational culture (P7Q25P4, P7Q25P3) and experts working on B2B solutions; for example one respondent noted, “I work on B2B solutions or dev oriented solutions, so the core of these solutions can be open-sourced so other devs can contribute to it.” (P2Q25P2).

On the other hand, Stack Overflow, an online Q&A community for developers,
allows software developers to freely ask and answer questions on any aspect of coding and hence, share their knowledge and advance their careers. Moreover, Stack Overflow also helps programmers and software developers by providing a platform to discuss technical problems with others within the software development community. The search functionality also enables others with similar coding problems to benefit from the posted solutions. Participants within this community usually participate for three specific reasons, i.e., participation trends: (i) view discussions to get help, (ii) post questions on a particular problem to seek help and (iii) offer answers and participate in their discussions. This research question explores developers’ participation trends in Stack Overflow and then investigates the factors that motivate developers to contribute to Stack Overflow. In summary, this research question aims at understanding the possible drivers behind developers’ contributions in Stack Overflow.

Thereafter, the closed-ended question was analyzed to understand the participation trends within Stack Overflow. The majority of respondents (37 out of 41 or
90%) noted that they use SO for viewing discussions to get help; while 80% (33 out of the 41) of the respondents said that they post questions on a particular problem to seek help; and, 85% (35 out of 41) of respondents noted that they also participate in answer discussions. Figure 17 presents these results.

It was found that unlike for GitHub results (RQ2), skills & expertise are the key drivers behind contribution in Stack Overflow (Table 8). The respondents mention that they contribute only in areas that they are confident in and which they believe they are qualified. More specifically, professional factors are the primary drivers behind contribution in Stack Overflow, while the personal drivers seem to take the back seat. Moreover, an “expert” bias can be observed, especially when it comes to contributing via posting questions as one respondent mentions that “…the SO answerer community can be quite cruel to askers; this has made me less inclined to participate recently” (P25Q25P4). And another respondent noted “I participate on discussion I can contribute. I avoid discussion in which I can’t” (P37Q26P1). This is in stark contrast with participation in GitHub, and this may be because a certain level of expertise is expected by the SO community while answering or asking questions, whereas GitHub community is more open contributions from a greater community, i.e., any developer is free to contribute and the project owners can either accept or decline a contribution (e.g., a pull request).

Notwithstanding the “expert bias” in Stack Overflow, it is ironic to note that some of the experts who responded report that they do not participate in Stack Overflow anymore because of lack of interesting questions (P10Q26P1) and that the reward system is skewed towards easier questions, as noted by one expert, “Questions that I can answer, particularly if technically challenging. Unfortunately SO rewards contributing to easy topics over hard.” (P34Q26P1).

There is however, one interesting case for using Stack Overflow for the purpose of documenting expertise (or knowledge base) of a software developer, as one response
Table 8: Contribution to Stack Overflow.

<table>
<thead>
<tr>
<th>Personal Drivers</th>
<th>Participants</th>
<th>Quotes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Helping</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Hobby/fun</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>Self-learning</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Own needs</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>Total</td>
<td>17</td>
<td>17</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Professional Drivers</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Skills &amp; expertise</td>
<td>19</td>
<td>20</td>
</tr>
<tr>
<td>Better answer</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Alternative solutions</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Total</td>
<td>23</td>
<td>24</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Challenges</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Rewarding system</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Tagging system</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>Total</td>
<td>8</td>
<td>9</td>
</tr>
</tbody>
</table>

highlights: “…due to time constraints: I recently use Stack Overflow primarily as a way of documenting my hard-won findings on a topic (e.g., posting and answering a question, then linking to it in the code), but am happy to get other answers and opinions added” (P16Q26P1). Another respondent discusses this further saying that it “…needs to be a niche area where there isn’t much expert competition. I can’t compete with ppl writing a comprehensive answer…” (P20Q26P1).

Some of the similarities between the factors driving contributions across both platforms are related to participants being motivated by “doing it for fun” (for GH, P:7; Q:8; while in SO, P:5; Q:5) and out of personal interest rather than as a chore;
some responses include “personal interest. If I spend time for free, I want to enjoy it” (P11Q25P1), “has to be interesting” (P37Q25P1), and personal interest (P39Q25P2, P21Q26P2, P9Q26P1).

**Answer to RQ2:** Majority of experts participate in both hosting private projects, as well as contributing to public projects. The findings of this thesis show that developers believe that personal drivers are more critical than professional factors in motivating them to contribute to GitHub. Among the personal drivers, developers are mainly driven by self needs when contributing to GitHub. On the other hand, contributions to SO are motivated more by professional drivers (i.e., skills and expertise). There seems to be a bias towards experts, yet developers do not seem to be keen to participate on Stack Overflow as they believe the questions are either no longer interesting or they are too easy.

### 4.1.3 Challenges Faced by Contributors

**RQ3: What challenges do developers face when contributing to each platform?**

This research question identifies key challenges developers face when contributing to Stack Overflow or GitHub. In a way, this question provides insights into why developers do not contribute to the platform or contribute in a limited capacity. Thereafter, two categories of challenges were identified for GitHub: one related to *issue complexity* and one due to *lack of time*. While for Stack Overflow, challenges are related to its *tagging system* and *rewarding system*.

For the open-ended question on the survey as to why developers do not contribute to GitHub, majority of the respondents mention lack of time as the reason, including “I do not contribute more to other people’s projects due to time constraints” (P16Q25P3), “I rarely contribute because I don’t have enough spare time”
It seems that developers are already contributing to their open source projects in GH and simply have no time to contribute to other projects as explained by P13Q25P2: “I have my own projects to work on on the side, so I don’t really have the time or interest to contribute to other open source projects”. Keeping GitHub projects small enough so others can understand and build them, making issues small and interesting are the key suggestions for GitHub project owners for encouraging better contributions and engaging more contributors on the projects. Another reason of why developers may decide to contribute or not to the project is the project’s or issue’s complexity. One developer (P8Q25P1) mentions that “it mostly depends on the complexity of the project”, while another respondent says that “when I try to contribute, I generally choose small intriguing issues” (P23Q25P2). It was also observed that developers’ contributions are driven based on whether they can build the project locally on their machines as supported by the following quote “whether I can build the project locally on my Macbook” (P26Q25P3).

Based on the responses, it was found that developers prefer to contribute to GH rather than participate on SO. Some of the issues cited were related to the difficulty finding interesting questions to answer: “I mostly don’t participate in Stack Overflow any more. It’s hard to find interesting questions to answer, and my time to create answers has gone down, too” (P10Q26P1). Another respondent (P34Q26P1) pointed out that SO is rewarding answers to easy topics rather than more challenging ones: “Unfortunately, SO rewards contributing to easy topics over hard”. While some developers felt that SO platform is less relevant to more specialized topics and thus they do not feel it’s rewarding to contribute to the discussions, “I do a lot more than I write on SO. SO is a waste of time outside specialized niches” (P33Q29P1).

Furthermore, about 50% of the participants mentioned that their poor contribution to SO is related to the current tagging system. Since tags are automatically determined and assigned by the gamification mechanisms within SO, participants found
that their “collected” tags (tags assigned to a developer’s profile) may no longer reflect their current expertise or interests. This response was consistent and reflects the fact that individual interests and expertise are dynamic in nature and they do evolve over time. As one respondent notes that “my SO profile [is my] expertise during PhD” (P34Q29P2) and thus does not reflect his/her current skills.

**Answer to RQ3**: In GitHub, developers mainly face technical challenges such as project/issue complexity, as well as personal challenges such as lack of time. While on Stack Overflow, developers’ lack of contribution and participation is perceived to be related to its tagging system (mismatch of SO tags with the developers’ expertise) and outdated/demotivating reward system.

### 4.2 Predicting Software Developer’s Expertise

This section presents outcomes of the prediction of multi-label classification, in other words, it attempts to predict the software developer expertise. In this section, the findings for each of the four research questions are presented. More specifically, Section 4.2.1 presents the best predicting vector representation technique on both SO and GH datasets is presented. In this section, the two word vectorization methodologies, TFIDF and Word2Vec, are compared against one another in both SO and GH, independently. Based on the results from this section, a specific word vectorizer is selected. Section 4.1.1 then goes on to address the predictive performance of various classifiers, based on the chosen word vectorizer, on both SO and GH datasets. This is done prior to balancing these datasets. Thereafter, the dataset is synthetically augmented using MLSMOTE technique. In Section 4.2.3 the classifier performance post-balancing is reviewed. The last section, Section 4.2.4 attempts to tie-up the information and understand how the dataset characteristics of SO and GH impact the overall performance in predicting the software developers.
4.2.1 Best Predicting Vector Representation Technique

RQ4: What feature is best predicting the software developer expertise in SO and GH?

Two word vectorization methods were applied on both the SO and GH datasets: (i) TFIDF, a deterministic method, and (ii) Word2Vec, a probabilistic method. This research question aims to find out which of these two feature extraction methods performs best on both the SO and GH datasets, respectively.

In order to find the best feature extraction method, ten ML classifiers were applied and their performance metrics were evaluated. In general, accuracy measure is the most common criteria to evaluate model performance, in most traditional classification problems, such as binary-class and multi-class problems. Furthermore, precision, recall, F1-Score, area under ROC are some of the standard evaluation metrics for such classification problems. However, in multi-label classification problems, accuracy might not be the appropriate measure to evaluate the performance of the model, as predictions of an instance is a set of labels. Therefore, the prediction could vary with different levels of correctness, i.e., it could be fully correct, partially correct or fully incorrect. Hence, this thesis uses micro F1-Score and Hamming Loss as the key metrics to evaluate the performance [91], [96].

Table 9 and Table 10 provide the results of all the ten ML classifiers along with the set of evaluation metrics on the SO Dataset. It was observed that both TFIDF and Word2Vec did not have any significant difference in performance, in terms of the evaluation metrics. However, Word2Vec was chosen as the best performing vectorizer, because of the following reasons: (1) Word2Vec used the word embeddings that were trained on software engineering domain corpus [24], (2) the Word2Vec word embeddings used are multi-dimensional vectors that capture a word’s relationship with other words, whereas TFIDF is only based on frequency and does not take into
Table 9: Results of ML classifiers, when **TFIDF** is applied on SO dataset.

<table>
<thead>
<tr>
<th>ML Classifiers</th>
<th>Precision</th>
<th>Recall</th>
<th>F1-Score</th>
<th>Hamming Loss</th>
<th>Accuracy</th>
<th>Execute Time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Binary Relevance - NB</td>
<td>0.917</td>
<td>0.961</td>
<td>0.938</td>
<td>0.086</td>
<td>0.136</td>
<td>314</td>
</tr>
<tr>
<td>Binary Relevance - LR</td>
<td>0.948</td>
<td>0.964</td>
<td>0.956</td>
<td>0.060</td>
<td>0.258</td>
<td>334</td>
</tr>
<tr>
<td><strong>Binary Relevance - SVM</strong></td>
<td><strong>0.960</strong></td>
<td><strong>0.974</strong></td>
<td><strong>0.967</strong></td>
<td><strong>0.045</strong></td>
<td><strong>0.367</strong></td>
<td><strong>141</strong></td>
</tr>
<tr>
<td>Label Powerset - NB</td>
<td>0.920</td>
<td>0.960</td>
<td>0.939</td>
<td>0.084</td>
<td>0.134</td>
<td>34</td>
</tr>
<tr>
<td>Label Powerset - LR</td>
<td>0.934</td>
<td>0.962</td>
<td>0.947</td>
<td>0.072</td>
<td>0.217</td>
<td>8,498</td>
</tr>
<tr>
<td>Label Powerset - SVM</td>
<td>0.940</td>
<td>0.961</td>
<td>0.951</td>
<td>0.068</td>
<td>0.249</td>
<td>103</td>
</tr>
<tr>
<td>Classifier Chain - NB</td>
<td>0.930</td>
<td>0.945</td>
<td>0.938</td>
<td>0.085</td>
<td>0.166</td>
<td>139</td>
</tr>
<tr>
<td>Classifier Chain - LR</td>
<td>0.945</td>
<td>0.968</td>
<td>0.956</td>
<td>0.060</td>
<td>0.278</td>
<td>1,423</td>
</tr>
<tr>
<td>Classifier Chain - SVM</td>
<td>0.956</td>
<td>0.976</td>
<td>0.966</td>
<td>0.047</td>
<td>0.379</td>
<td>424</td>
</tr>
<tr>
<td>MLKNN</td>
<td>0.936</td>
<td>0.943</td>
<td>0.940</td>
<td>0.082</td>
<td>0.151</td>
<td>8,296</td>
</tr>
</tbody>
</table>

Table 10: Results of ML classifiers, when **Word2Vec** is applied on SO dataset.

<table>
<thead>
<tr>
<th>ML Classifiers</th>
<th>Precision</th>
<th>Recall</th>
<th>F1-Score</th>
<th>Hamming Loss</th>
<th>Accuracy</th>
<th>Execute Time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Binary Relevance - NB</td>
<td>0.881</td>
<td>0.791</td>
<td>0.834</td>
<td>0.215</td>
<td>0.024</td>
<td>6</td>
</tr>
<tr>
<td>Binary Relevance - LR</td>
<td>0.943</td>
<td>0.961</td>
<td>0.952</td>
<td>0.066</td>
<td>0.210</td>
<td>16</td>
</tr>
<tr>
<td>Binary Relevance - SVM</td>
<td>0.945</td>
<td>0.963</td>
<td>0.953</td>
<td>0.064</td>
<td>0.220</td>
<td>25</td>
</tr>
<tr>
<td>Label Powerset - NB</td>
<td>0.920</td>
<td>0.945</td>
<td>0.933</td>
<td>0.093</td>
<td>0.078</td>
<td>35</td>
</tr>
<tr>
<td>Label Powerset - LR</td>
<td>0.932</td>
<td>0.963</td>
<td>0.947</td>
<td>0.073</td>
<td>0.203</td>
<td>157</td>
</tr>
<tr>
<td>Label Powerset - SVM</td>
<td>0.936</td>
<td>0.962</td>
<td>0.949</td>
<td>0.070</td>
<td>0.220</td>
<td>226</td>
</tr>
<tr>
<td>Classifier Chain - NB</td>
<td>0.822</td>
<td>0.756</td>
<td>0.787</td>
<td>0.277</td>
<td>0.002</td>
<td>7</td>
</tr>
<tr>
<td>Classifier Chain - LR</td>
<td>0.939</td>
<td>0.965</td>
<td>0.952</td>
<td>0.066</td>
<td>0.226</td>
<td>37</td>
</tr>
<tr>
<td>** Classifier Chain - SVM**</td>
<td><strong>0.942</strong></td>
<td><strong>0.965</strong></td>
<td><strong>0.953</strong></td>
<td><strong>0.064</strong></td>
<td><strong>0.234</strong></td>
<td><strong>57</strong></td>
</tr>
<tr>
<td>MLKNN</td>
<td>0.938</td>
<td>0.929</td>
<td>0.933</td>
<td>0.090</td>
<td>0.121</td>
<td>168</td>
</tr>
</tbody>
</table>

Account semantic similarities, and (3) the execution time for Word2Vec vectorizer was significantly lower, possibly due to the dense nature of the data matrix; whereas, TFIDF is a sparse matrix.

Table [11] and Table [12] show the results of the ten ML classifiers along with the set of evaluation metrics on the GH dataset. The results indicate that Word2Vec is performing better than TFIDF, in terms of all the evaluation metrics. Possibly, this was due to implementing the ML model with word embeddings from software engineering domain. Moreover, even on the GH dataset, executing the model using Word2Vec was significantly faster compared to TFIDF vectorizer.
Table 11: Results of ML classifiers, when **TFIDF** is applied on GH dataset.

<table>
<thead>
<tr>
<th>ML Classifiers</th>
<th>Precision</th>
<th>Recall</th>
<th>F1-Score</th>
<th>Hamming Loss</th>
<th>Accuracy</th>
<th>Execute Time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Binary Relevance - NB</td>
<td>0.407</td>
<td>0.499</td>
<td>0.448</td>
<td>0.340</td>
<td>0.036</td>
<td>227</td>
</tr>
<tr>
<td><strong>Binary Relevance - LR</strong></td>
<td><strong>0.678</strong></td>
<td><strong>0.458</strong></td>
<td><strong>0.547</strong></td>
<td><strong>0.210</strong></td>
<td><strong>0.098</strong></td>
<td><strong>298</strong></td>
</tr>
<tr>
<td>Binary Relevance - SVM</td>
<td>0.615</td>
<td>0.476</td>
<td>0.537</td>
<td>0.228</td>
<td>0.074</td>
<td>109</td>
</tr>
<tr>
<td>Label Powerset - NB</td>
<td>0.588</td>
<td>0.329</td>
<td>0.422</td>
<td>0.250</td>
<td>0.092</td>
<td>2,125</td>
</tr>
<tr>
<td>Label Powerset - LR</td>
<td>0.712</td>
<td>0.252</td>
<td>0.373</td>
<td>0.235</td>
<td>0.150</td>
<td>4,328</td>
</tr>
<tr>
<td>Label Powerset - SVM</td>
<td>0.573</td>
<td>0.416</td>
<td>0.482</td>
<td>0.247</td>
<td>0.118</td>
<td>121</td>
</tr>
<tr>
<td>Classifier Chain - NB</td>
<td>0.412</td>
<td>0.497</td>
<td>0.450</td>
<td>0.336</td>
<td>0.037</td>
<td>382</td>
</tr>
<tr>
<td>Classifier Chain - LR</td>
<td>0.670</td>
<td>0.407</td>
<td>0.506</td>
<td>0.220</td>
<td>0.117</td>
<td>553</td>
</tr>
<tr>
<td>Classifier Chain - SVM</td>
<td>0.599</td>
<td>0.449</td>
<td>0.514</td>
<td>0.236</td>
<td>0.097</td>
<td>203</td>
</tr>
<tr>
<td>MLKNN</td>
<td>0.539</td>
<td>0.447</td>
<td>0.489</td>
<td>0.259</td>
<td>0.070</td>
<td>7,424</td>
</tr>
</tbody>
</table>

Table 12: Results of ML classifiers, when **Word2Vec** is applied on GH dataset.

<table>
<thead>
<tr>
<th>ML Classifiers</th>
<th>Precision</th>
<th>Recall</th>
<th>F1-Score</th>
<th>Hamming Loss</th>
<th>Accuracy</th>
<th>Execute Time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Binary Relevance - NB</td>
<td>0.454</td>
<td>0.598</td>
<td>0.516</td>
<td>0.311</td>
<td>0.065</td>
<td>2</td>
</tr>
<tr>
<td><strong>Binary Relevance - LR</strong></td>
<td><strong>0.672</strong></td>
<td><strong>0.490</strong></td>
<td><strong>0.567</strong></td>
<td><strong>0.207</strong></td>
<td><strong>0.092</strong></td>
<td><strong>5</strong></td>
</tr>
<tr>
<td>Binary Relevance - SVM</td>
<td>0.669</td>
<td>0.482</td>
<td>0.560</td>
<td>0.209</td>
<td>0.088</td>
<td>16</td>
</tr>
<tr>
<td>Label Powerset - NB</td>
<td>0.505</td>
<td>0.607</td>
<td>0.551</td>
<td>0.273</td>
<td>0.082</td>
<td>22</td>
</tr>
<tr>
<td>Label Powerset - LR</td>
<td>0.723</td>
<td>0.253</td>
<td>0.375</td>
<td>0.234</td>
<td>0.158</td>
<td>101</td>
</tr>
<tr>
<td>Label Powerset - SVM</td>
<td>0.650</td>
<td>0.343</td>
<td>0.449</td>
<td>0.233</td>
<td>0.155</td>
<td>192</td>
</tr>
<tr>
<td>Classifier Chain - NB</td>
<td>0.435</td>
<td>0.597</td>
<td>0.503</td>
<td>0.327</td>
<td>0.077</td>
<td>3</td>
</tr>
<tr>
<td>Classifier Chain - LR</td>
<td>0.647</td>
<td>0.446</td>
<td>0.528</td>
<td>0.221</td>
<td>0.124</td>
<td>8</td>
</tr>
<tr>
<td>Classifier Chain - SVM</td>
<td>0.645</td>
<td>0.441</td>
<td>0.524</td>
<td>0.222</td>
<td>0.114</td>
<td>34</td>
</tr>
<tr>
<td>MLKNN</td>
<td>0.544</td>
<td>0.570</td>
<td>0.557</td>
<td>0.241</td>
<td>0.075</td>
<td>214</td>
</tr>
</tbody>
</table>
Answer to RQ4: In summary, the Word2Vec vectorizer seemed to be best predicting the software developer expertise than TFIDF for both Stack Overflow and GitHub social coding platforms. Going forward, the thesis only concentrates on Word2Vec.

4.2.2 Predictive Performance of Classifiers

RQ5: What is the predictive performance of classifiers in predicting software developer expertise in SO and GH?

In the previous RQ, Word2Vec has been selected as the preferred vectorizer on both SO and GH datasets. In this RQ, the objective is to identify the best classifier algorithm amongst the four multi-label classification models with Naive Bayes, Support Vector Machine and Logistic Regression as the base classifiers for each model.

Figure 18 shows the micro F1-Score and Hamming Loss of the ten classifiers that were applied on the SO dataset. There appears to be no significant difference in scores between all the classifiers. However, among the classifiers, it can be observed that Support Vector Machine with Binary Relevance and Classifier Chain models have the best relative predictive performance, both in terms of micro F-1 score and Hamming Loss; this classifier has the highest micro F1-Score of 95.3% and lowest Hamming Loss of 0.064, among all the classifiers applied on this dataset.

Moreover, in a real world setting, a software developer usually develops expertise in multiple fields (e.g., a software developer might be an expert in both Python programming as well as in databases). This means: (i) each instance has multiple labels and (ii) these multiple labels are correlated. Also, since classifier chain SVM (CCSVM) preserves the label correlations, it is expected that this classifier would perform better on the SO dataset.

On the other hand, for the GH dataset, Figure 19 shows the micro F1-Score and Hamming Loss of the ten classifiers. Based on the metrics, it appears that Binary
Figure 18: Best predicting classifier for Stack Overflow.

Relevance with Logistic Regression and Support Vector Machine as base classifiers and MLKNN have performed better in predicting the software developer expertise. Among these, Binary Relevance with Logistic Regression classifier has the highest micro F1-Score (56.7%) and the lowest Hamming Loss (0.207).

**Answer to RQ5:** In summary, before balancing the datasets, the following classifiers performed best on SO and GH datasets, respectively:

- **Classifier Chain SVM (CCSVM)** for the SO dataset;
- **Binary Relevance with Logistic Regression (BRLR)** for the GH dataset.

### 4.2.3 Effects of Balancing the Multi-label Datasets

RQ6: How effective is MLSMOTE, a multi-label balancing technique, in improving the predictive performance?
As discussed in Section 3.2.1, after pre-processing the data, exploratory data analysis was performed to analyze the distribution of labels across the two datasets. From the initial exploration, it was observed that there are 31 unique labels identified as expertise labels in SO corpus. They are: Data-Visualization, Front-end-Web-development, Python, Algorithms, HTTP, OOP, Server-client, Java, Data-Types, C/C++, version-control, Data-Management, java-build-tools, Ruby, iOS-Development, Parallel-Programming, Android, VIM, PHP, Data-Searching, ssl+-game-dev, Encryption, Javascript-Programming, File-Management, UI, Web-Layout, Database, Angular, Unix, Distance-Calculation, and Web-Graphics.

The count plot (Figure 20) for each label is plotted to check the distribution of labels across the SO corpus. This plot shows that the labels data management, data types, OOP, Parallel Programming, VIM occurred more than 12,600 times, each. And the labels such as Andriod, C/C++ and Java build tools each occurred more than 12,500
Table 13: Results after applying MLSMOTE: SO dataset.

<table>
<thead>
<tr>
<th>ML Classifiers</th>
<th>Precision</th>
<th>Recall</th>
<th>F1-Score</th>
<th>Hamming Loss</th>
<th>Accuracy</th>
<th>Execute Time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Binary Relevance - NB</td>
<td>0.878</td>
<td>0.785</td>
<td>0.829</td>
<td>0.221</td>
<td>0.020</td>
<td>4</td>
</tr>
<tr>
<td>Binary Relevance - LR</td>
<td>0.943</td>
<td>0.957</td>
<td>0.950</td>
<td>0.069</td>
<td>0.173</td>
<td>10</td>
</tr>
<tr>
<td>Binary Relevance - SVM</td>
<td>0.943</td>
<td>0.959</td>
<td>0.951</td>
<td>0.067</td>
<td>0.180</td>
<td>15</td>
</tr>
<tr>
<td>Label Powerset - NB</td>
<td>0.934</td>
<td>0.952</td>
<td>0.943</td>
<td>0.079</td>
<td>0.184</td>
<td>20</td>
</tr>
<tr>
<td>Label Powerset - LR</td>
<td>0.929</td>
<td>0.959</td>
<td>0.944</td>
<td>0.078</td>
<td>0.166</td>
<td>124</td>
</tr>
<tr>
<td>Label Powerset - SVM</td>
<td>0.946</td>
<td>0.963</td>
<td>0.954</td>
<td>0.063</td>
<td>0.269</td>
<td>212</td>
</tr>
<tr>
<td>Classifier Chain - NB</td>
<td>0.825</td>
<td>0.763</td>
<td>0.792</td>
<td>0.273</td>
<td>0.002</td>
<td>7</td>
</tr>
<tr>
<td>Classifier Chain - LR</td>
<td>0.938</td>
<td>0.960</td>
<td>0.949</td>
<td>0.070</td>
<td>0.186</td>
<td>16</td>
</tr>
<tr>
<td>Classifier Chain - SVM</td>
<td>0.940</td>
<td>0.961</td>
<td>0.951</td>
<td>0.068</td>
<td>0.193</td>
<td>58</td>
</tr>
<tr>
<td>MLKNN</td>
<td>0.940</td>
<td>0.940</td>
<td>0.940</td>
<td>0.082</td>
<td>0.140</td>
<td>257</td>
</tr>
</tbody>
</table>

times. This shows that the SO dataset is imbalanced.

On the other hand, there were 15 unique labels in the GH corpus; they are: web-development, back-end development, C/C++, web layout, Ruby, data science, real time app development, UNIX, Rust, PHP, Scala/Haskell, JVM, front-end web development, mobile app development, and EMACS. The count plot (Figure 21) for each label is plotted to check the distribution of labels across the GH corpus. This plot shows that the label backenddev has the highest number of experts (9,253), followed by PHP (8,256) and Unix (6,693). Again, the GH dataset is also highly imbalanced.

In order to overcome this problem, the SO and GH datasets are balanced using the MLSMOTE technique, as discussed in Section 3.2.4. Table 13 and Table 14 display the the results after the SO and GH datasets were augmented, respectively. In general, an overall model performance is attained post balancing of the datasets.

More specifically, on the SO dataset, Label Powerset with Support Vector Machine as the base classifier seem to be performing better after balancing, with a F1-Score of 95.4% and a Hamming Loss of 0.063. In essence, the MLSMOTE technique that was applied has improved the predictive performance of the minority class instances and avoided the over-fitting problem. This is evident from Figure 22.
Figure 20: Number of SO experts per topic.
Figure 21: Number of GH experts per topic.
Table 14: Results after applying MLSMOTE: GH dataset.

<table>
<thead>
<tr>
<th>ML Classifiers</th>
<th>Precision</th>
<th>Recall</th>
<th>F1-Score</th>
<th>Hamming Loss</th>
<th>Accuracy</th>
<th>Execute Time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Binary Relevance - NB</td>
<td>0.559</td>
<td>0.634</td>
<td>0.594</td>
<td>0.323</td>
<td>0.046</td>
<td>5</td>
</tr>
<tr>
<td>Binary Relevance - LR</td>
<td>0.688</td>
<td>0.627</td>
<td>0.656</td>
<td>0.245</td>
<td>0.053</td>
<td>20</td>
</tr>
<tr>
<td>Binary Relevance - SVM</td>
<td>0.689</td>
<td>0.621</td>
<td>0.653</td>
<td>0.246</td>
<td>0.051</td>
<td>49</td>
</tr>
<tr>
<td>Label Powerset - NB</td>
<td>0.656</td>
<td>0.697</td>
<td>0.676</td>
<td>0.249</td>
<td>0.205</td>
<td>82</td>
</tr>
<tr>
<td>Label Powerset - LR</td>
<td>0.696</td>
<td>0.449</td>
<td>0.546</td>
<td>0.279</td>
<td>0.152</td>
<td>379</td>
</tr>
<tr>
<td>Label Powerset - SVM</td>
<td>0.701</td>
<td>0.648</td>
<td>0.674</td>
<td>0.234</td>
<td>0.249</td>
<td>590</td>
</tr>
<tr>
<td>Classifier Chain - NB</td>
<td>0.542</td>
<td>0.627</td>
<td>0.581</td>
<td>0.337</td>
<td>0.050</td>
<td>7</td>
</tr>
<tr>
<td>Classifier Chain - LR</td>
<td>0.637</td>
<td>0.614</td>
<td>0.627</td>
<td>0.273</td>
<td>0.077</td>
<td>20</td>
</tr>
<tr>
<td>Classifier Chain - SVM</td>
<td>0.636</td>
<td>0.627</td>
<td>0.632</td>
<td>0.272</td>
<td>0.073</td>
<td>93</td>
</tr>
<tr>
<td>MLKNN</td>
<td>0.696</td>
<td>0.766</td>
<td>0.729</td>
<td>0.212</td>
<td>0.212</td>
<td>744</td>
</tr>
</tbody>
</table>

which provides individual label scores of the best performing classifier, that is Label Powerset - SVM; moreover, this figure provides comparison between balanced and unbalanced values for each label. From Figure 22 we can observe that after balancing the F1-Score has improved for the following minority instance labels: Angular, Database, Distance-Calculation, Server-Client, UI, Unix, Web-Graphics, and Web-Layout.

For the GH dataset, on the other hand, MLKNN, an algorithm adaption multi-label classification model does seem to be performing better after balancing, with a F1-Score of 72.9% and a Hamming Loss of 0.212. Similar to results in the SO dataset, applying the MLSMOTE technique has improved the predictive performance of the minority class instances. This is evident from Figure 23 which provides individual label scores of the best performing classifier, that is MLKNN; moreover, this figure provides comparison between balanced and unbalanced values for each label. From Figure 23 we can observe that after balancing the F1-Score has improved for the following minority instance labels: JVM, C/C++ Dev, Data Science, EMACS, Front-end Web Development, Mobile App Development, Real Time App Development, Ruby, Rust, Scala and haskell, Web Development, and Web Layout.
<table>
<thead>
<tr>
<th>Labels</th>
<th>UNBALANCED</th>
<th></th>
<th></th>
<th></th>
<th>BALANCED</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Precision</td>
<td>Recall</td>
<td>F1-score</td>
<td>Support</td>
<td>Precision</td>
<td>Recall</td>
<td>F1-score</td>
<td>Support</td>
</tr>
<tr>
<td>Algorithms</td>
<td>0.815</td>
<td>0.9084</td>
<td>0.8595</td>
<td>2,794</td>
<td>0.8315</td>
<td>0.9073</td>
<td>0.8677</td>
<td>3,850</td>
</tr>
<tr>
<td>Android</td>
<td>0.9762</td>
<td>0.9989</td>
<td>0.9874</td>
<td>3,772</td>
<td>0.9838</td>
<td>0.9973</td>
<td>0.9905</td>
<td>5,249</td>
</tr>
<tr>
<td>Angular</td>
<td>0.6947</td>
<td>0.2018</td>
<td>0.3128</td>
<td>327</td>
<td>0.7564</td>
<td>0.5123</td>
<td>0.6109</td>
<td>691</td>
</tr>
<tr>
<td>Data-Management</td>
<td>0.9839</td>
<td>0.9989</td>
<td>0.9914</td>
<td>3,801</td>
<td>0.9899</td>
<td>0.9983</td>
<td>0.9941</td>
<td>5,295</td>
</tr>
<tr>
<td>Data-Searching</td>
<td>0.9712</td>
<td>0.9979</td>
<td>0.9844</td>
<td>3,751</td>
<td>0.9773</td>
<td>0.9982</td>
<td>0.9867</td>
<td>5,197</td>
</tr>
<tr>
<td>Data-Types</td>
<td>0.9788</td>
<td>0.9987</td>
<td>0.9886</td>
<td>3,782</td>
<td>0.9867</td>
<td>0.9983</td>
<td>0.9924</td>
<td>5,265</td>
</tr>
<tr>
<td>Data-Visualization</td>
<td>0.9655</td>
<td>0.9981</td>
<td>0.9815</td>
<td>3,727</td>
<td>0.9734</td>
<td>0.9963</td>
<td>0.9847</td>
<td>5,176</td>
</tr>
<tr>
<td>Database</td>
<td>0.6870</td>
<td>0.4280</td>
<td>0.5274</td>
<td>764</td>
<td>0.7136</td>
<td>0.4857</td>
<td>0.5780</td>
<td>980</td>
</tr>
<tr>
<td>Distance-Calculation</td>
<td>0.3684</td>
<td>0.1250</td>
<td>0.1867</td>
<td>56</td>
<td>0.7468</td>
<td>0.4836</td>
<td>0.5871</td>
<td>122</td>
</tr>
<tr>
<td>Encryption</td>
<td>0.9663</td>
<td>0.9989</td>
<td>0.9823</td>
<td>3,733</td>
<td>0.9751</td>
<td>0.9967</td>
<td>0.9858</td>
<td>5,186</td>
</tr>
<tr>
<td>File-Management</td>
<td>0.6829</td>
<td>0.4623</td>
<td>0.5513</td>
<td>941</td>
<td>0.6953</td>
<td>0.4605</td>
<td>0.5440</td>
<td>1,214</td>
</tr>
<tr>
<td>Front-end-Web-development</td>
<td>0.7893</td>
<td>0.9617</td>
<td>0.8670</td>
<td>2,921</td>
<td>0.8264</td>
<td>0.9609</td>
<td>0.9886</td>
<td>4,066</td>
</tr>
<tr>
<td>Http</td>
<td>0.9582</td>
<td>0.9981</td>
<td>0.9777</td>
<td>3,694</td>
<td>0.9714</td>
<td>0.9951</td>
<td>0.9831</td>
<td>5,124</td>
</tr>
<tr>
<td>Java</td>
<td>0.9670</td>
<td>0.9979</td>
<td>0.9822</td>
<td>3,733</td>
<td>0.9800</td>
<td>0.9967</td>
<td>0.9883</td>
<td>5,204</td>
</tr>
<tr>
<td>Javascript-Programming</td>
<td>0.7086</td>
<td>0.8304</td>
<td>0.7617</td>
<td>1,975</td>
<td>0.7754</td>
<td>0.8206</td>
<td>0.7701</td>
<td>2,637</td>
</tr>
<tr>
<td>OOP</td>
<td>0.9842</td>
<td>1.0000</td>
<td>0.9920</td>
<td>3,802</td>
<td>0.9837</td>
<td>0.9979</td>
<td>0.9938</td>
<td>5,295</td>
</tr>
<tr>
<td>Parallel-Programming</td>
<td>0.9808</td>
<td>0.9987</td>
<td>0.9897</td>
<td>3,790</td>
<td>0.9888</td>
<td>0.9979</td>
<td>0.9933</td>
<td>5,285</td>
</tr>
<tr>
<td>Php</td>
<td>0.9516</td>
<td>0.9967</td>
<td>0.9737</td>
<td>3,671</td>
<td>0.9672</td>
<td>0.9945</td>
<td>0.9806</td>
<td>5,093</td>
</tr>
<tr>
<td>Python</td>
<td>0.9655</td>
<td>0.9973</td>
<td>0.9811</td>
<td>3,728</td>
<td>0.9745</td>
<td>0.9965</td>
<td>0.9854</td>
<td>5,179</td>
</tr>
<tr>
<td>Ruby</td>
<td>0.8917</td>
<td>0.9868</td>
<td>0.9368</td>
<td>3,412</td>
<td>0.9187</td>
<td>0.9837</td>
<td>0.9301</td>
<td>4,664</td>
</tr>
<tr>
<td>Server-client</td>
<td>0.4425</td>
<td>0.2219</td>
<td>0.2956</td>
<td>347</td>
<td>0.5632</td>
<td>0.5656</td>
<td>0.5644</td>
<td>709</td>
</tr>
<tr>
<td>UI</td>
<td>0.6848</td>
<td>0.1921</td>
<td>0.3000</td>
<td>328</td>
<td>0.7566</td>
<td>0.4441</td>
<td>0.5997</td>
<td>707</td>
</tr>
<tr>
<td>Unix</td>
<td>0.4000</td>
<td>0.1359</td>
<td>0.2029</td>
<td>206</td>
<td>0.5198</td>
<td>0.3958</td>
<td>0.4494</td>
<td>432</td>
</tr>
<tr>
<td>VIM</td>
<td>0.9793</td>
<td>0.9992</td>
<td>0.9891</td>
<td>3,781</td>
<td>0.9882</td>
<td>0.9968</td>
<td>0.9925</td>
<td>5,280</td>
</tr>
<tr>
<td>Web-Graphics</td>
<td>0.3333</td>
<td>0.0635</td>
<td>0.1067</td>
<td>63</td>
<td>0.7011</td>
<td>0.4519</td>
<td>0.5495</td>
<td>135</td>
</tr>
<tr>
<td>Web-Layout</td>
<td>0.3696</td>
<td>0.0821</td>
<td>0.1344</td>
<td>207</td>
<td>0.5743</td>
<td>0.4383</td>
<td>0.4971</td>
<td>397</td>
</tr>
<tr>
<td>c/c++</td>
<td>0.9767</td>
<td>0.9981</td>
<td>0.9873</td>
<td>3,774</td>
<td>0.9863</td>
<td>0.9972</td>
<td>0.9917</td>
<td>5,267</td>
</tr>
<tr>
<td>iOS-Development</td>
<td>0.9048</td>
<td>0.9911</td>
<td>0.9460</td>
<td>3,471</td>
<td>0.9282</td>
<td>0.9900</td>
<td>0.9881</td>
<td>4,008</td>
</tr>
<tr>
<td>java-build-tools</td>
<td>0.9762</td>
<td>0.9989</td>
<td>0.9674</td>
<td>3,771</td>
<td>0.9838</td>
<td>0.9952</td>
<td>0.9900</td>
<td>5,245</td>
</tr>
<tr>
<td>ssl-game-dev</td>
<td>0.9570</td>
<td>0.9992</td>
<td>0.9776</td>
<td>3,694</td>
<td>0.9733</td>
<td>0.9957</td>
<td>0.9849</td>
<td>5,154</td>
</tr>
<tr>
<td>version-control</td>
<td>0.9517</td>
<td>0.9965</td>
<td>0.9736</td>
<td>3,662</td>
<td>0.9636</td>
<td>0.9953</td>
<td>0.9792</td>
<td>5,108</td>
</tr>
<tr>
<td>Micro Avg.</td>
<td>0.9361</td>
<td>0.9620</td>
<td>0.9489</td>
<td>81,478</td>
<td>0.9461</td>
<td>0.9630</td>
<td>0.9545</td>
<td>114,014</td>
</tr>
<tr>
<td>Macro Avg.</td>
<td>0.8148</td>
<td>0.7601</td>
<td>0.7650</td>
<td>81,478</td>
<td>0.8681</td>
<td>0.8337</td>
<td>0.8464</td>
<td>114,014</td>
</tr>
<tr>
<td>Weighted Avg.</td>
<td>0.9315</td>
<td>0.9620</td>
<td>0.9435</td>
<td>81,478</td>
<td>0.9443</td>
<td>0.9630</td>
<td>0.9522</td>
<td>114,014</td>
</tr>
<tr>
<td>Samples Avg.</td>
<td>0.9358</td>
<td>0.9605</td>
<td>0.9415</td>
<td>81,478</td>
<td>0.9460</td>
<td>0.9626</td>
<td>0.9493</td>
<td>114,014</td>
</tr>
</tbody>
</table>

Figure 22: SO LPSVM: balanced vs. unbalanced.
Answer to RQ6: In summary, after balancing the datasets, we can conclude that the following classifiers performed best on SO and GH datasets, respectively:

- **For the SO dataset:** Label Powerset SVM performed best after balancing compared to Classifier Chain SVM (CCSVM) prior to balancing;

- **For the GH dataset:** MLKNN performed best after balancing compared to Binary Relevance with Logistic Regression (BRLR) prior to balancing.

Furthermore, the prediction of minority classes has improved the F-1 score on average by 29.12% and 26.69% for SO and GH, respectively.
In addition, a non-parametric Wilcoxon test is conducted to test the null hypothesis that there is no statistically significant difference between the best performing classifier, post balancing, and the remaining nine (9) classifiers. For the SO dataset, all the null hypothesis are rejected. This means that only Label Powerset SVM is performing best after data augmentation. For the GH dataset, it was observed that except for Label Powerset - SVM (LP-SVM) and Label Powerset - NB (LP-NB), the null hypothesis is rejected for all other classifiers. In other words, only LP-SVM (p-value of 0.5750) and LP-NB (p-value of 0.7988) are performing very similar to the chosen MLKNN.

4.2.4 Factors Impacting Predictive Performance

RQ7: How do SO and GH data characteristics impact the performance?

Expertise is not an uni-dimensional facet of an individual, it is indeed multi-dimensional. In other words, individuals de-risk themselves by acquiring and developing expertise in multiple areas/fields. The same is the case with experts in the field of software development. There are many areas this is reflected. This thesis, however, focused its attention on the social coding platforms, Stack Overflow and GitHub. Needless to state, software developers on both these platforms have consistently contributed on various aspects of software engineering and displayed their multi-faceted expertise. In other words, the SO and GH datasets are both multi-label datasets. However, the models applied on both datasets did not perform equally well. On the one hand, the models on the SO dataset performed really well; whereas, on GH dataset the same models only had a moderate performance. This could be due to a plethora of reasons and the main one among them could be because of the different nature of the user profiles on SO and GH.

Figure 24 demonstrates that about 76% of total SO users are experts in multiple fields. For example, in the SO dataset, there are about 3,800 users with 22 tags,
about 3,300 users with 21 tags and 2,400 users that have 23 tags. The maximum number of expertise tags that a software developer on Stack Overflow have are 26. On further checking the count plot, it is observed that the distribution of expertise tags per user is highly skewed towards the right. There are high number of users with 20 or more tags and a very low number of users with less than 19 tags. This implies that the Stack Overflow community has highly diversified expertise.

On the other hand, the GH dataset provides a completely contrasting picture. Figure 25 shows that about 76% of total GH users are experts in only five(5) or less fields. For example, 5,742 GH users (accounting to about 43% of total GH users) have less than or equal to 2 expertise tags; while 1,420 users (10.6% of total users) have 3 expertise tags. The maximum number of tags present per user are 15. On checking the count plot of the distribution of tags per user, it is noted that the distribution is highly skewed towards the left. There are a large number of users with 5 or less tags; whereas, there are very few users with more than 6 tags. This implies that the GH
Finally, a label graph is plotted on both the platforms to explore the label space with an aim to understand the correlation among the topics. This data characteristics is one of the features that could affect the predictive performance of the expertise. Each label is a node in the graph and an edge exists when labels co-occur, weighted by the frequency of co-occurrence. Community detection methods such as the Louvain algorithm is used to cluster the label graph. This is implemented in the NetworkXLabelGraphCluster with the parameter method like Louvain.

Furthermore, as shown in Figure 26 for SO dataset there are 31 labels and 465 edges and two (2) clusters. This shows the influence one label has on other labels. From this network graph, it is visible that yellow colored labels are one cluster and the purple colored labels belong to another cluster. More specifically, in the yellow colored cluster, there are four expertise labels that have a low probability of occurrence with other expertise labels; the four labels in the yellow cluster are: Distance Calculation (node # 8), Unix (node # 22), Web-graphics (node # 24) and Web layout (node...
Moreover, this implies that the expertise within the SO community is more densely connected.

Whereas, as shown in Figure 27 for the GH dataset, there are 15 labels and 105 edges and two (2) clusters. From this network graph, it is visible that yellow colored labels are one cluster and the purple colored labels belong to another cluster. More specifically, in the yellow colored cluster, there are seven expertise labels that have a low probability of occurrence with other expertise labels; the seven labels in the yellow cluster are: Back-end Development (node # 1)), C/C++ (node # 2), data science (node # 3), PHP (node # 7), real-time app development (node # 8), RUST (node # 10), and UNIX (node # 12). Moreover, this implies that the expertise within the GH community is more sparsely connected.
Answer to RQ7: In summary, we can draw the following observations about the characteristics of SO and GH datasets that have an impact on the predictive performance of the overall model:

- **Concentration of Expertise**: On the SO dataset, it is observed that expertise is more diversified. That means, the software developers contributing on this platform, that has a ‘questions and answers’ style, have a wide variety of knowledge on various software fields. Whereas, on the other hand, the GH community, that is a project repository based one, it is observed that expertise is more concentrated. In other words, the software developers contributing to this platform are having very focused expertise.

- **Dependency/Co-occurrence of Expertise**: It is observed that both SO and GH datasets have clusters of expertise. However, the SO dataset is observed to have highly dense network among the labels, while the GH dataset has a sparse network.
Chapter 5

Discussion

This chapter presents the overall findings in Section 5.1, discusses the implications of this study in Section 5.2, and addresses potential threats to validity in Section 5.3.

5.1 Findings

The first part of the thesis dealt with understanding software developer expertise. The findings indicate that knowledge and experience are the two factors that drive expertise. The quantitative analysis showed that JavaScript remains a popular language, while knowledge and experience are the key factors driving expertise. On the other hand, qualitative analysis showed that soft skills such as effective and clear communication, analytical thinking are key factors defining an expert. Also, it could be inferred that an expert’s contribution to GitHub seems to be driven by personal factors, while contribution to Stack Overflow is motivated more by professional drivers (i.e., skills and expertise). Moreover, developers seem to prefer contributing to GitHub as they face greater challenges while contributing to Stack Overflow.

The second part of the thesis focuses on predicting software developer expertise. We found that there are about 83,350 individuals that have accounts in both Stack Overflow and GitHub. Moreover, after cleaning the data and pre-processing the text,
it resulted in 12,898 SO user profiles and 13,381 GH users profiles, along with their topic labels. Also, it was noticed that though these users have accounts in both Stack Overflow and GitHub, they did not, however, contribute to both the platforms; this was observed by linking the two datasets to check on their active contribution on the two platforms. More precisely, only about 3 to 4% of the users had full activity on both platforms. Furthermore, the findings show that Word2Vec, a probabilistic word embedding, is the most suitable vectorizer for both SO and GH datasets. And, upon using MLSMOTE to augment the data, it was found that Label Powerset - SVM classifier and and MLKNN performed best for predicting the expertise on SO and GH dataset, respectively. Moreover, the GH community have more concentrated expertise and this is reflected in the low levels of influence that one label has on the others within a network of labels. On the other hand, SO users have more diverse expertise and their label co-occurrence frequency is very high.

In essence, the first part of the thesis has qualitatively defined that knowledge and experience along with sufficient soft skills are key in forming software developer expertise. Moreover, in order to quantify this understanding, the second part of the thesis has explored prediction of software developer expertise. However, our prediction model has only leveraged knowledge and experience and not considered the soft-skills of a software developer. This limitation can be addressed in a future work by extracting and identifying developers’ soft skills from other platforms such as their LinkedIn profiles (e.g., self declared soft skills, skill endorsements, recommendations) or by applying the tone/emotion detection and sentiment analysis techniques to the developers’ participation and engagement in online discussion forums such as Reddit.
5.2 Implications

This research has wide ranging implications and is expected to have an impact on four main target audiences: 1) employers, recruiters and recruitment oriented services, 2) researchers, 3) software developers, project managers and software companies, and 4) Stack Overflow and GitHub decision makers.

5.2.1 Employers & Recruiters

The findings of this work could benefit employers and recruiters, as well. For example, knowing the trends in specialization and characteristics of an expert could help employers to hire the right expert at a minimal search-cost. Also, learning the reasons that motivate developers and thus make efforts to retain talent and attract more talent.

In addition, the second exercise, could help the recruiters and employers to predict expertise of a software developer more accurately based on the individual’s activity and participation on social development platforms. This would again help reduce the search-costs and time involved in hiring the appropriate talent.

5.2.2 Researchers

The findings of this thesis can form the basis for further comprehensive multi-layered research by expanding the number of platforms and social media sources to draw a complete picture of on “expert” and “expertise”. One of the observations from this study is that each developer brings their own opinion on the definition of the expertise. And therefore, the targeted population of the developers who participate in such expertise-related surveys should be diverse and inclusive. Furthermore, researchers could possibly explore a variety of other endogenous and exogenous aspects. Also, it would be possible to expand this work to include profiles of software developers on a
a number of social platforms such as Twitter and LinkedIn.

5.2.3 Software Developers

The insights from this work can help developers to better understand the platforms and increase their awareness on some of the critical aspects that may discourage developers from gaining expertise (e.g., Stack Overflow’s tagging system and reward mechanisms) or, on the contrary, further support their learning and expertise development (e.g., contributing to open source community in GH or sharing knowledge on SO). Furthermore, developers often monitor their activities and progress. However, modern tools are too complex and create an informational overload on software developers. Thus, in order to reduce burden of information overload it is critical to offer personalized dashboards [99], [100], [101], as some participants have mentioned, that improve developer situational awareness [102,103] and support self-monitoring activities.

5.2.4 Suggestions to SO and GH Decision Makers

The results of this study highlight the challenges developers face when participating on Stack Overflow. Based on the developers responses, it is understood that they feel the current SO tagging system needs major upgrade as tags assigned to the developers’ profiles do not accurately reflect their current level of expertise. We recommend Stack Overflow, as a platform, to add a tag updating mechanism so developers can decide which tags are still relevant to their expertise. Additionally, in the interest of nurturing future talent Stack Overflow needs to be more welcoming to new and novice contributors having their moderators be aware of and better manage subtle and overt “bullying” that these developers might be facing.
Our results also highlight the challenges developers face when participating on GitHub. It was observed that developers face project and/or issue complexity on GitHub. Moreover, some developers lack adequate computing resources when accessing and working on GitHub projects. The recommendation to GitHub would be to consider integrating cloud computing resources to alleviate this challenge.

### 5.3 Threats and Limitations

This thesis is subject to several threats and limitations.

First, it has to be noted that the randomly selected sample of active users on SO and GH lacks gender diversity. While challenging, inclusion and equity are becoming more critical parameters when recruiting participants for qualitative studies like this one. The dataset does not provide gender information on users (which can be classified as personal information and thus violate privacy protection laws such as GDPR), while it is possible to have inferred the gender by applying gender detection techniques \[104, 105\] in order to strive for a gender balanced sample. The thesis could be extended to include such techniques and approaches for targeting a more diverse population of participants.

Second, while the study and the corresponding survey were carefully designed and the survey questions were drafted to ensure clarity, as with any exploratory study, there is a chance that some degree of researcher bias might have been introduced inadvertently. Moreover, when applying the open coding approach it is quite possible that this might have introduced a researcher bias into the analysis. Although, the coders have tried to minimize this bias by coding the 20% of the card sorts extracted from each question independently, measuring and reporting the coder reliability (agreement) on the next 20% and reporting these values in the study.
Third, as with any survey method, to control for sampling bias can be challenging. This thesis targeted users of Stack Overflow and GitHub who actively participate on both platforms by contributing to Stack Overflow discussions and being active developers and contributors on various GitHub projects. While the findings might not generalize outside of the selected population of users, it is believed that developers who contribute to both platforms may share and experience similar behaviour and patterns.

Fourth, it is understood that the definition of “active users” is somewhat ambiguous and arbitrary. While the developer emails were obtained from the 2013 GHTorrent dataset (due to the GDPR being adopted in 2016, email addresses are no longer published), the “active users” are determined to be active on these platforms based on their activity and contributions as of December 2019. However, the data certainly lacks active users who have joined SO and GH after 2013.

Finally, further research studies are needed to be able to provide greater insight into developer contributions to various online social and collaborative platforms, beyond Stack Overflow and GitHub, to be able to develop an empirical body of knowledge on this topic. To encourage replication of this study, the survey used has been documented and open coding results in a technical report which together with anonymized survey responses is made available online [106].
Chapter 6

Conclusion

Lastly, the thesis concludes the work with a summary of the contributions in Section 6.1 and future research paths that are outlined in Section 6.2.

6.1 Summary of Contributions

Understanding software developer expertise. This thesis presented a qualitative study of software developer expertise with an attempt to understand the reasons why or why not developers contribute to GitHub and Stack Overflow. Also, the thesis tried to understand the ways in which experts monitor their performance and the challenges they face when contributing to either collaborative platform. To understand these aspects better, a survey was designed with both open-ended and closed-ended questions. The survey was sent to a sample of experts that were active on both Stack Overflow and GitHub. An open coding was conducted on the 324 individual quotes obtained from the open-ended questions. As a result of the open coding, 46 main categories together with four concept categories have emerged. Through the research questions, the thesis was able to gain insights into developers’ opinions and perception on key characteristics of an expert, factors that drive developers to contribute to GitHub and Stack Overflow, as well as the challenges developers face in the process.
The findings suggested that JavaScript and C/C++ are the leading programming language skills and surprisingly the extent of skill diversification among experts was lower than expected. Moreover, a software development expert would have both technical and soft skills.

Furthermore, users who contribute to both platforms seem to favour GitHub more often as it’s less biased towards experts. Also, developers use GitHub for personal reasons, while Stack Overflow is more catered to professional purposes.

Lastly, it was found that developers on GitHub face technical challenges such as project complexity, as well as personal challenges such as lack of time. On the other hand, developers on Stack Overflow faced a more negative reinforcements demotivating their contributions.

**Predicting software developer expertise.** This thesis built a model to predict a software developer’s expertise based on their activity and participation on Stack Overflow and GitHub platforms. The thesis used a multi-label classification analysis, a supervised machine learning (ML) technique to predict expertise. The model has accurately predicted the most popular tags. The findings suggest that, after balancing, *Label Powerset - SVM* for SO and *MLKNN* for GH datasets, using Word2Vec vectorizer as a feature extraction model, are the best performing classifiers.

Furthermore, it was found that the software developers contribution on Stack Overflow are not necessarily the same as those contribution on GitHub, and vice versa.

Lastly, the model found that software developers contribution on GitHub have a more concentrated expertise areas; whereas, the software developers contribution on Stack Overflow had expertise in greater range of topics.
6.2 Future Work

Understanding software developer expertise. The first part of this thesis has a few limitations, and these could be cured as part of future works.

Theory of software developer expertise. Baltes and Diehl \[3\] have recently proposed a conceptual theory of software development expertise. While their theory is grounded in the online survey responses of 355 software developers, the targeted population was focused primarily on Java experts. This thesis further investigates the concepts of “expert” and “expertise” within the field of software development and discovers additional findings on why developers may contribute to some social platform but not the other. While the results of this thesis are grounded in the survey responses of fewer developers, the thesis targeted a different population of developers who are active on both platforms in order to better understand how they build and maintain their expertise across multiple platforms. The next steps would be to extend the theory developed by Baltes and Diehl by considering broader scope of “expertise” definition, in particular it would be interesting to build a theory of cross-platform expertise that focuses on the specific properties and characteristics of the experts who are contributing to multiple platforms. In other words, it would be interesting to investigate the range of skills and behaviour that are supportive of developers’ success in becoming better experts.

Knowledge transfer across multiple platforms. The findings demonstrate that developers do share their knowledge on multiple platforms. One of the interesting directions would be to conduct studies to further understand what technical knowledge and skills are more transferable across different platforms. Such insights can help developers better plan how they contribute to different platforms. If some platforms are more favourable for specific knowledge transfer, developers can, for example, increase their participation on that platform.
Predicting software developer expertise. The second part of this thesis could also be further extended in a few ways.

One possible extension could be to predict the software developer’s expertise on other popular social platform like LinkedIn and check if there are any differences between the inferences drawn from the currently chosen platforms. Furthermore, the prediction of expertise could be made more robust by using datasets from multi-platforms [107].

Another possible extension could be to model and predict the SO and GH user profile using deep learning and neural network models [108].

One other possible extension could be to investigate evolution of expertise over time and the factors that impact these changes in expertise. In other words, it would be interesting to track developer expertise over time.

Lastly, another possible extension could involve multi-task, multi-channel and multi-input learning to better predict expertise among software developers. This is a fast emerging methodology that provides a more holistic understanding of expertise and possibly a greater predictive capability [109].
List of References


[88] Qualtrics, “Qualtrics software.”


Appendix A

Appendix

A.1 CUREB Acceptance Letter
CERTIFICATION OF INSTITUTIONAL ETHICS CLEARANCE

The Carleton University Research Ethics Board-B (CUREB-B) has granted ethics clearance for the research project described below and research may now proceed. CUREB-B is constituted and operates in compliance with the Tri-Council Policy Statement: Ethical Conduct for Research Involving Humans (TCPS2).

Ethics Protocol Clearance ID: Project # 111751

Research Team: Mrs. Sri Vadlamani (Primary Investigator)
Dr. Olga Baysal (Research Supervisor)

Project Title: Building a Theory of Software Developer Cross Platform Expertise

Funding Source (If applicable):


Please ensure the study clearance number is prominently placed in all recruitment and consent materials: CUREB-B Clearance # 111751.

Restrictions:

This certification is subject to the following conditions:

1. Clearance is granted only for the research and purposes described in the application.
2. Any modification to the approved research must be submitted to CUREB-B via a Change to Protocol Form. All changes must be cleared prior to the continuance of the research.
3. An Annual Status Report for the renewal of ethics clearance must be submitted and cleared by the renewal date listed above. Failure to submit the Annual Status Report will result in the closure of the file. If funding is associated, funds will be frozen.
4. A closure request must be sent to CUREB-B when the research is complete or terminated.
5. During the course of the study, if you encounter an adverse event, material incidental finding, protocol deviation or other unanticipated problem, you must complete and submit a Report of
Adverse Events and Unanticipated Problems Form, found here: https://carleton.ca/researchethics/forms-and-templates/

Failure to conduct the research in accordance with the principles of the Tri-Council Policy Statement: Ethical Conduct for Research Involving Humans 2nd edition and the Carleton University Policies and Procedures for the Ethical Conduct of Research may result in the suspension or termination of the research project.

Upon reasonable request, it is the policy of CUREB, for cleared protocols, to release the name of the PI, the title of the project, and the date of clearance and any renewal(s).

Please contact the Research Compliance Coordinators, at ethics@carleton.ca, if you have any questions.

CLEARED BY: Natasha Artemeva, PhD, Chair, CUREB-B

Date: November 27, 2019

Janet Mantler, PhD, Vice-Chair, CUREB-B
A.2 Expertise Study Survey Questionnaire
Consent Form

Towards Understanding Software Developers' Cross Platform Expertise - Consent Form

Dear Developer,

Thank you for taking part in this survey! This survey aims at understanding how developers build and maintain good reputations and profiles across multiple online development forums such as Stack Overflow and GitHub. It is expected that this survey would only take 10-15 minutes for you to complete.

This survey data would be used for research purposes and would not be used for any commercial endeavors. Your input would help tremendously and be much appreciated!

Please note that there are no risks associated with this study, and the participation in this study is voluntarily and if you feel uncomfortable while answering any questions, you can withdraw from the survey within three (3) weeks from the time of submission. Your data will be stored and protected by Qualtrics on Toronto-based servers but may be disclosed via a court order or data breach. Furthermore, the identity data that is collected in this survey will be coded to maintain the anonymity. And, please be advised that this coded data would be deleted one year after the submission of the thesis.

The results of this research may also be published in professional research journals or presented at scientific conferences, but any such presentations will report only aggregated findings, which in some instances may be illustrated by short, anonymous quotes carefully selected so as not to breach individual confidentiality. This research has been cleared by Carleton University Research Ethics Board (CUREB-B Clearance #111751).
Should you have any ethical concerns with the study, please contact the REB Chair, Carleton University Research Ethics Board-B, by phone: 613-520-2600 ext. 4085 or by email: ethics@carleton.ca. For all other questions about the study, please contact Sri Lakshmi Vadlamani or the thesis supervisor Prof. Olga Baysal at carletonu.thesis21@gmail.com.

I voluntarily agree to participate in this study.

- Yes
- No

Background Information

What is your main role during software development? If "Other" Please specify.

- Software developer/engineer
- Software tester/QA
- Software Architect/Designer
- Administrator
- Manager
- Team lead
- Consultant
- Researcher
- Other
What is your age? (optional)

- < 20
- 20 - 30
- 30 - 40
- 40+

What is your gender? (optional)

- Male
- Female
- Other

What is your country/continent of residence (e.g., Europe, North America)? (optional)

What is your highest degree earned? If 'Other' Please specify?

- Bachelor or equivalent
- Master or equivalent
- PhD. or equivalent
- Other

For what purpose do you use GitHub?

- [ ] Host private projects
- [ ] Contribute to public projects

What is your GitHub User ID? (The information you provide will be kept confidential)
For what purpose do you use Stack Overflow?

☐ View discussions to get help
☐ Post questions on a particular problem to seek help
☐ Participating in discussions (answering, solving or commenting in the threads)

What is your Stack Overflow User ID? (The information you provide will be kept confidential)

How many years of software development experience do you have?

☐ <1
☐ 1 - 3
☐ 4 - 6
☐ 7 - 9
☐ 10+

Expertise levels in different area

Expertise levels in different area

Please rate your programming language expertise on the following scale of 1 (None) to 5 (Expert). If 'Other' please specify your answer clearly otherwise click "None".

(1) None  (Beginner)  (Intermediate)  (Proficient)  (Expert)
Please rate your Frameworks & Libraries expertise on the following scale of 1 (None) to 5 (Expert). If 'Other' please specify your answer clearly otherwise click "None".

<table>
<thead>
<tr>
<th>Language</th>
<th>(None)</th>
<th>(Beginner)</th>
<th>(Intermediate)</th>
<th>(Proficient)</th>
<th>(Expert)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Java</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C/C++</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Python</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C#</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VisualBasic (including Net)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>JavaScript</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SQL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PHP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Objective-C</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Groovy</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Swift</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ruby</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Assembly Language</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Go</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pearl</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MATLAB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Other</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Node.js</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Please rate your Technologies expertise on the following scale of 1 (None) to 5 (Expert). If 'Other' please specify your answer clearly otherwise click "None".

Angular.JS
Spring
ExpressJS
ASP
Django
.Net Core
Vue.js
Struts
JSF
Backbone.js
Cocoa
React
Hadoop
Spark
Other

Firebase
Internet of Things
Deep Learning
Machine Learning
Please rate your Databases expertise on the following scale of 1 (None) to 5 (Expert). If 'Other' please specify your answer clearly otherwise click "None".

<table>
<thead>
<tr>
<th>Database</th>
<th>(None)</th>
<th>(Beginner)</th>
<th>(Intermediate)</th>
<th>(Proficient)</th>
<th>(Expert)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MySQL</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
<tr>
<td>SQL Server</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
<tr>
<td>SQLite</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
<tr>
<td>PostgreSQL</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
<tr>
<td>MongoDB</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
<tr>
<td>Oracle</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
<tr>
<td>Redis</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
<tr>
<td>Cassandra</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
<tr>
<td>Other</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
</tbody>
</table>

When rating your own expertise in the above skills, what did you consider? If 'Other' please specify clearly.

☐ My professional experience in software development (as opposed to private projects)
Expertise Discussion

Please name 3-5 most important skills that a software development expert should be good at:
(Please start with the most important skills)

Do you regularly monitor your own software development activities (e.g., using software metrics or other statistics)?

- Yes
- No
- I don't want to answer

If yes, how do you monitor your software development activities?
Cross Platform Expertise

How do you decide which project to contribute to in GitHub? If you don't contribute, why?

How do you decide which topics to participate (for example answering, solving or commenting in the threads) in Stack Overflow? If you don't participate, why?

What are the top 5 tags that you are participating in Stack Overflow?

Do the top 5 tags mentioned above reflect your key expertise?

☐ Yes

☐ No

If 'No', please justify
Powered by Qualtrics
A.3 Technical Report
1. INFORMAL OVERVIEW
The paper seeks to understand whether experts in software development are made based on their knowledge and experience alone or there are other ingredients that make them “experts”. It was found out that both knowledge and experience are only necessary but not sufficient and that soft-skills are the key in making them an “expert” in the field of software development.

An exploratory survey of software developers was conducted and a combination of quantitative and qualitative methods were used to analyze the survey results. Based on our criteria of “active users” a total of 423 software developers were identified as being active on both GitHub and Stack Overflow. A survey questionnaire was sent to these users and of this 73 started working on the survey and 41 finally completed the survey. The study aimed to understand, classify and analyze the underlying themes and concepts that could be derived from these survey responses.

The completed responses were split into 324 quotations or individual statements and an open card sort was performed. The objective of this was gain insight into high-level themes and/or categories about expertise. Five (5) open ended questions were posed and during the qualitative analysis phase, 4 high-level ‘concept-categories’ emerged from these:

1. Expert’s skills (participants – 41; quotes – 176; categories – 23)
2. Contribution in GitHub (participants – 41; quotes -75; categories – 9)
3. Contribution in Stack Overflow (participants – 41; quotes – 50; categories – 9)
4. Tags in SO match the expertise (participants – 41; quotes – 23; categories – 5)

An open coding technique was applied on the 324 comments to classify them into natural evolving ‘concept-categories’. Moreover, the high-level ‘concept-categories’ were further sub-divided into 46 categories or themes representing various aspects of expertise and each category consisted of between one (1) and forty-four (44) comments or quotations. An overview of the concept categories and their themes are elaborated in the next section with details of the participant and the number of quotations (statements) in each ‘concept-category’ and theme. Moreover, for each theme a synthetic quote (SQ), an overall summary of the thought on that specific topic, is derived from the overall quotations or statements made on the specific theme. For a complete analysis, please refer to Appendix A.

2. CONCEPT CATEGORIES
The four (4) ‘concept-categories’ that have emerged during the card sorting exercise do not necessarily have direct correspondence to the tasks performed by experts on the social coding platforms. Instead,
these categories are a combination of actions and mental activities performed by experts contributing to the social coding platforms (GH and SO). The four (4) concept categories are:

1. **Defining features of an expert** *(Software developer skills; Monitoring skills)*. This area pertained to identifying the essential skills and competencies of experts who contributed to software development, in general (specifically within social coding platforms) and the features that an expert would have to possess and how this could be monitored.

2. **Factors driving experts to contribute in GitHub** *(contribution in GitHub)*. This concept category offers insights into the reason’s experts contribute to GitHub and the associated inhibiting factors.

3. **Factors driving experts to contribute to Stack Overflow** *(contributions in Stack Overflow)*. This concept category offers insights into the reason’s experts contribute to Stack Overflow and the associated inhibiting factors.

4. **Challenges faced by experts in contributing to online collaborative platforms** *(tags in Stack Overflow match the expertise)*. This is very specific to Stack Overflow, as it is the only platform amongst the two that allow for tags. This ‘concept-category’ checked on how the tags in Stack Overflow match the expertise of a software developer.

Technical skills are critical for a programmer to possess; for example, under the software developers’ skills question, based on the card sorting exercise, there were 44 quotes that were classified as ‘technical’ and this was from comments obtained from 21 out of the total respondents who believed that this is a key skill. It must be noted that some participants emphasized by providing more than one quote or comment related to this specific theme. However, soft skills like communication skills (P – 15; Q – 15), analytical thinking (P – 24; Q – 26), planning & organization (P – 9; Q – 11) and behavior that includes tenacity, patience and perseverance (P – 9; Q – 17) play a crucial in forming expertise.

Contribution to GH and SO does have very similar themes, in terms of the contribution of experts. However, from the qualitative analysis, some subtle difference can be observed. For example, “self-needs” is the most crucial driving factor for expert’s contribution to GH (P – 27; Q – 34); whereas, this factor is not very important for contribution to SO (P – 3; Q - 3). On the other hand, “skills/expertise” are crucial for contribution to SO (P – 19; Q -20). Finally, tags within SO are automatically selected and do not often reflect the current state or expertise of a contributor (P – 7; Q – 7).

<table>
<thead>
<tr>
<th>Open Ended Questions in the Survey</th>
<th># of Participants</th>
<th># of Quotes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Software Developers’ Skills</td>
<td>41</td>
<td>153</td>
</tr>
<tr>
<td>▪ Communication skills</td>
<td>15</td>
<td>15</td>
</tr>
<tr>
<td>▪ Technical</td>
<td>21</td>
<td>44</td>
</tr>
<tr>
<td>▪ Analytical thinking</td>
<td>24</td>
<td>26</td>
</tr>
<tr>
<td>▪ Planning &amp; organizing</td>
<td>9</td>
<td>11</td>
</tr>
<tr>
<td>▪ Creativity</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>▪ Experience</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>▪ Understanding</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>▪ Critical thinking</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>▪ Behavior</td>
<td>9</td>
<td>17</td>
</tr>
<tr>
<td>▪ Vision</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>▪ Domain knowledge</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>▪ Broad knowledge</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>▪ Requirements</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>▪ Teamwork</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>▪ Teaching skills</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>▪ Searching/seeking help</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>▪ Miscellaneous</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
2. Monitoring Skills

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Self-tracking issues</strong></td>
<td>3</td>
</tr>
<tr>
<td><strong>Dashboard</strong></td>
<td>5</td>
</tr>
<tr>
<td><strong>Online collaborative platforms</strong></td>
<td>3</td>
</tr>
<tr>
<td><strong>Online documentation</strong></td>
<td>1</td>
</tr>
<tr>
<td><strong>Testing tools</strong></td>
<td>5</td>
</tr>
<tr>
<td><strong>Time tracking tools</strong></td>
<td>3</td>
</tr>
</tbody>
</table>

3. Contribution to GitHub

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Hobby/fun</strong></td>
<td>7</td>
</tr>
<tr>
<td><strong>Issue complexity</strong></td>
<td>6</td>
</tr>
<tr>
<td><strong>Organization culture</strong></td>
<td>1</td>
</tr>
<tr>
<td><strong>Self needs</strong></td>
<td>27</td>
</tr>
<tr>
<td><strong>Job</strong></td>
<td>1</td>
</tr>
<tr>
<td><strong>Helping</strong></td>
<td>5</td>
</tr>
<tr>
<td><strong>Expertise</strong></td>
<td>10</td>
</tr>
<tr>
<td><strong>Open source project</strong></td>
<td>1</td>
</tr>
<tr>
<td><strong>Lack of time</strong></td>
<td>8</td>
</tr>
</tbody>
</table>

4. Contribution in Stack Overflow

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Better answer</strong></td>
<td>3</td>
</tr>
<tr>
<td><strong>Helping</strong></td>
<td>3</td>
</tr>
<tr>
<td><strong>Hobby/fun</strong></td>
<td>5</td>
</tr>
<tr>
<td><strong>Skill/expertise</strong></td>
<td>19</td>
</tr>
<tr>
<td><strong>Self-learning</strong></td>
<td>3</td>
</tr>
<tr>
<td><strong>Own needs</strong></td>
<td>6</td>
</tr>
<tr>
<td><strong>Alternative solutions</strong></td>
<td>1</td>
</tr>
<tr>
<td><strong>Tagging system</strong></td>
<td>7</td>
</tr>
<tr>
<td><strong>Rewarding system</strong></td>
<td>1</td>
</tr>
</tbody>
</table>

5. Tags in SO match the expertise

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Limited participation</strong></td>
<td>1</td>
</tr>
<tr>
<td><strong>Limitation of platform</strong></td>
<td>1</td>
</tr>
<tr>
<td><strong>Gamification configuration</strong></td>
<td>7</td>
</tr>
<tr>
<td><strong>Mismatch if tags with expertise</strong></td>
<td>4</td>
</tr>
<tr>
<td><strong>Miscellaneous</strong></td>
<td>8</td>
</tr>
</tbody>
</table>

An appendix to this document, provides a comprehensive overview of the categories and themes, and a selection of participants’ quotes. Furthermore, a synthetic quote (SQ) synthesizing participants’ comments/ statements for each theme is provided in the appendix.

3. ABOUT THE AUTHORS

**Sri Lakshmi Vadlamani** is a master student in computer science with data science specialization at the School of Computer Science, Carleton University. Her research interests include mining software repositories, topic classification and topic modeling using natural language processing. She has a MS in Electrical Engineering from California State University, Northridge.

**Olga Baysal** is an Associate Professor at the School of Computer Science, Carleton University. Her research interests span a wide range of software engineering, including empirical software engineering, mining software repositories, data science, data analytics, software analytics, software maintenance and evolution, and human aspects of software engineering. She obtained her PhD from the University of Waterloo, Canada.
Appendix A

Software Developers Skills
[Participants (P): 41; Quotes (Q): 153]

- **[P: 15, Q: 15] Communication skills**
  - **Includes**: Effective & clear communication is a key skill for a software developer.
  - **Summary Quotient (SQ)**: Good communication skills defines software developer’s expertise and are important for the below reasons:
    - Discuss technical options with team members and colleagues, ensuring that point is conveyed appropriately.
    - Teach a colleague or friend about the great technology/practice which are being used, i.e., effective knowledge transfer.
    - Defend the development or technical strategy that one believes is best while conveying to manager/customer that it has more pros than cons... And describing accurately those pros and their advantages as well as the consequences of not doing it right.

- **[P: 21, Q: 44] Technical**
  - **Includes**: Software developer’s technical skills like programing languages, version control, testing, debugging, designing, documentation.
  - **SQ**: It is important to have strong understanding on the programing languages, data structures, debugging or algorithms and to be able to interpret ideas clearly into code, to be an expert in software development.
    - P1Q21P1. Proficiency in the particular programming language.
    - P6Q21P5. Writing clear idiomatic programs in at least one language.
    - P25Q21P1. "Thinking in code" (fluently translating ideas into code).

- **[P: 24, Q: 26] Analytical thinking**
  - **Includes**: Problem solving skills and ability to think abstractly.
  - **SQ**: Being able to think logically and breaking down of complex problems to simpler problems would add to the definition of software developer’s expertise.
    - P16Q21P2. Reducing problems to minimal test cases
    - P32Q21P2. Big-picture/small-details analytics
    - P6Q21P4. Breaking complex problems into simpler tasks

- **[P: 9, Q: 11] Planning & organizing**
  - **Includes**: Organizing, focusing and planning.
  - **SQ**: Being able to balance the workload and achieve overall goals of time and space performance are defined to be important skills
    - P16Q21P4. Balancing time/space trade-offs
    - P41Q21P2. Being organised
• **[P: 3, Q: 3] Creativity**  
  o **Includes**: Improvising and having aesthetic sense.  
  o **SQ**: Creativity is the source to improvise solutions to problems for dominating complex systems such as software development.  
    ▪ P2Q2P2. Improvisation  
    ▪ P5Q2P2. Aesthetic sense

• **[P: 2, Q: 4] Experience**  
  o **Includes**: Breadth of knowledge, experience.  
  o **SQ**: Having the breadth and depth of knowledge and experience to back it up is very important. In other words, clear representation of the code comes with experience, which is key for an expert software developer.  
    ▪ P16Q2P1. writing code in a way that can be understood by others  
    ▪ P16Q2P6. (or preferably working with methods that are "correct-by-construction")

• **[P: 4, Q: 4] Understanding**  
  o **Includes**: Common sense.  
  o **SQ**: Having the capacity to comprehend dynamic ideas and express them in simple/clear terms is an important skill.  
    ▪ P14Q2P3. ability to understand abstract concepts  
    ▪ P39Q2P1. expressing intent clearly

• **[P: 3, Q: 3] Critical thinking**  
  o **Includes**: Flexibility and pragmatism.  
  o **SQ**: It is very important to actively & skillfully conceptualize, apply and analyze the problem and solve it.  
    ▪ P24Q2P4. Quickly determining what is relevant  
    ▪ P8Q2P1. pragmatism

• **[P: 9, Q: 17] Behavior**  
  o **Includes**: Tenacity, patience, continuous learning, curiosity & flexibility.  
  o **SQ**: Behavior and attitude towards problem solving is a very crucial skill and this requires a programmer to be tenacious, patient, curious and flexible in order for them to reach their goals. This also requires them to show ample of empathy towards others both at their work place and outside.  
    ▪ P41Q2P1. Being empathetic about the needs of other people  
    ▪ P20Q2P5. Workplace behavior  
    ▪ P13Q2P4. Tenacity

• **[P: 1, Q: 1] Vision**  
  o **Includes**: Attention to the end goal.  
  o **SQ**: Having a clear and a well-defined end goal is very important in completing a project.  
    ▪ P8Q2P3. Pay attention to the goal
• [P: 2, Q: 2] Domain knowledge
  o Includes: Domain knowledge.
  o SQ: Key to solving a problem is to both understand the problem and to have a good understanding of the domain where the software is being used and/or the domain of the programming.
    ▪ P1Q21P3. understanding of the domain that is being worked in (not programming itself, but the domain that the software is being used in)
    ▪ P20Q21P3. Domain knowledge

• [P: 7, Q: 8] Broad knowledge
  o Includes: Comprehensiveness, knowledge ecosystem, self-learning, fundamentals.
  o SQ: Having a wider knowledge base that is outside of programming itself would help; it would help the programmer in gaining solid fundamental understanding of the issues at hand. This however, requires substantial self-learning, researching on the subject and having an ability to tap into the knowledge ecosystem. This also includes knowledge and good grasp of other languages: Mathematics and English.
    ▪ P7Q21P1. Comprehensiveness
    ▪ P16Q21P3. researching prior art
    ▪ P20Q21P4. Ecosystem knowledge
    ▪ P21Q21P3. self-learning
    ▪ P33Q21P2. fundamentals

• [P: 6, Q: 7] Requirements
  o Includes: Requirements gathering, defining specification and translate requirements to design solution.
  o SQ: The first step of any project is requirement gathering of the end-user; clearly defining the specifications of the inputs and most importantly, it requires an ability to translate the requirements into solutions.
    ▪ P10Q21P1. requirements elicitation
    ▪ P11Q21P2. Translate requirements to design solutions
    ▪ P25Q21P3. attention to detail
    ▪ P33Q21P1. knowing what needs to be done

• [P: 3, Q: 3] Teamwork
  o Includes: Teamwork and ability to work together.
  o SQ: Key to any project (including software development) success is the ability to work in a team and to understand and appreciate the contribution of others. This comes from a basic recognition that all projects (of some value and size) need contribution of other stakeholders.
    ▪ P22Q21P3. ability to work together
    ▪ P20Q21P2. teamwork

• [P: 2, Q: 2] Teaching skills
  o Includes: Teaching and mentoring.
One of the key skills in software development is to teach and mentor others. This would create an effective eco-system of helpful users and builds a support network, which in turn would make the project more efficient and effective.

- **P24Q21P2. Mentoring others**
- **P18Q21P4. Teaching users**

### [P: 2, Q: 2] Searching/seeking help
- **Includes**: Google error messages.
- **SQ**: Self-help sometimes is the best help as it might be difficult to communicate pain points of a developer to others, in that case, it is recommended to rely upon internet for possible solutions and guidance to solve the problem.
  - **P24Q21P3. Googling error messages**
  - **P28Q21P4. Searching solution on Internet**

### [P: 1, Q: 1] Miscellaneous
- **Includes**: Cutting and pasting.
- **SQ**: There is one irrelevant comment and hence is classified as “garbage”.
  - **P26Q21P4. cutting-and-pasting**

### Monitoring Skills
**[Participants (P): 14, Quotes (Q): 23]**

- **[P: 3, Q: 3] Self tracking Issues**
  - **Includes**: Monitoring resolved issues, having a database of public facing issues.
  - **SQ**: After execution of a project (e.g., software development), the monitoring and control phase is an important one within the overall project. Hence, creating an issue-register and a comprehensive database of all the issues and their lessons learned is a key to overall project success.
    - **P32Q23P1. I mostly reflect on what I am doing, why, and if it helps me solve problems faster and maintain the code in the long run.**
    - **P16Q23P3. public-facing issue databases**
    - **P35Q23P1. Monitoring issues resolved**

- **[P: 5, Q: 7] Dashboard**
  - **Includes**: GitHub dashboard with commit graph, tracking metrics, heat charts, GitHub insights.
  - **SQ**: Providing summary information is essential to understand and easy to communicate with other stakeholders. Social platforms, like GitHub, provide dashboards to summarize the contribution and progress of an individual developer. These dashboards provide valuable information both to the developer as well as other relevant stakeholders
    - **P5Q23P1. via github insight**
    - **P20Q23P1. GitHub dashboard with commit graph**
- P38Q23P1. tracking metrics I'm interested in
- P30Q23P2. GitHub/GitLab heat charts

- [P: 3, Q: 4] Online collaborative platforms
  - Includes: GitHub, OhLoh, Git Commits.
  - SQ: Monitoring tools are provided within the online collaborative platforms to help track progress.
    - P7Q23P1. git commits
    - P30Q23P4. OhLoh, etc.

- [P: 1, Q: 1] Online documentation
  - Includes: Publicly available progress reports.
  - SQ: As part of project monitoring and audit it is critical to have publicly available progress reports.
    - P16Q23P2. public-facing progress reports

- [P: 5, Q: 6] Testing tools
  - Includes: Forensics, continuous integration and testing, linters, static analysis.
  - SQ: Testing is a key part of the monitoring and control process of a project. Testing could be a continuous quality check and/or it could be done at the end of the project before handing over the project (software program) to the end user.
    - P39Q23P1. Automated code metrics, such as cyclometric complexity or code coverage percentages.
    - P16Q23P1. continuous integration and testing
    - P14Q23P2. static analysis (like Sonar cloud)
    - P10Q23P3. Forensics
    - P10Q23P1. Linters
    - P29Q23P1. Tests

- [P: 3, Q: 3] Time tracking tools
  - Includes: Time tracking with detailed activity breakdown, burn-down charts.
  - SQ: Time is always of the essence. It is essential to keep a good/detailed track of time on each activity and sub-activity. This would help know/predict well in advance any time overruns. Hence, time tracking tools is key within the monitoring and control process of the project. Also included in here is an altruistic theme of helping others and thus a more well-rounded growth of the development team.
    - P3Q23P1. Time tracking with detailed activity type breakdown
    - P10Q23P4. burn-down charts
    - P35Q23P2. time spent helping others
Contribution in GITHUB
[Participants (P): 41, Quotes (Q): 75]

- **[P: 7, Q: 8] Hobby/fun**
  - **Includes**: Hobby, personal interest, fun, enjoy coding.
  - **SQ**: Contribution on GitHub is not always driven by professional requirements and choices. In fact, there is ample evidence that notes that contribution to social coding platforms is often driven by software developers’ personal interest and/or hobby. They contribute because they enjoy working on these challenges created by projects.
    - P4Q25P2. some of it is hobby though
    - P11Q25P1. Personal interest. If I spend my time for free, I want to enjoy it
    - P25Q25P3. I prefer to contribute to projects that use languages I enjoy working in
    - P37Q25P1. Has to be interesting

- **[P: 6, Q: 6] Issue complexity**
  - **Includes**: Complexity of the project, size of the issue and mentally challenging.
  - **SQ**: Contribution to GitHub is often related to the complexity of the project, size of the issues and intrigue and challenge involved. Contribution is generally positively correlated with the level of challenge.
    - P23Q25P2. When I try to contribute, I generally choose small intriguing issues.
    - P37Q25P2. mentally challenging.
    - P8Q25P1. It mostly depends on the complexity of the project.
    - P7Q25P2. size of the issues.
    - P30Q25P1. As a Linux distribution contributor most of my contributions are based on issues packaging software.

- **[P: 1, Q: 2] Organization culture**
  - **Includes**: Project organization and process of building.
  - **SQ**: Contribution to GitHub requires a culture of project management that involves detailed project organization that involves planning, execution and monitoring during the process of building a software development.
    - P7Q25P3. build process
    - P7Q25P4. organization of the project

- **[P: 27, Q: 34] Self needs**
  - **Includes**: Driven by own needs and scratch my own itches.
  - **SQ**: Contribution to social coding platforms often begins with a problem or challenge the one is faced with and do not have ready available answers. In such instances, developers often rely on the social platforms for ready answers; however, if they do not find one
they tend to work on developing the same and thus end up contributing to the overall knowledge base.

- P1Q25P3. a feature I need I will try and contribute
- P1Q25P1. Based off my own needs.
- P3Q25P2. contribute bug fixes to software I use
- P10Q25P2. I need to fix something broken that's blocking me
- P13Q25P1. I generally only contribute bug fixes if I run into something in a third-party library.
- P17Q25P1. If I encounter a bug and I want it fixed badly enough, I will attempt to and if successful contribute my fix
- P21Q25P1. What I'm currently using that needs changes
- P27Q25P1. I use the project and want it to keep being maintained
- P34Q25P1. I contribute to things that I use and
- P41Q25P1. Ones that I use and that have something that's causing me problems which I'm able to fix fairly easily.
- P35Q25P1. I generally contribute to a project that I will be using in some way or another

- **[P: 1, Q: 1] Job**
  - **Includes:** Paid work.
  - **SQ:** Only a small percent of respondents noted that they contribute to social coding platforms as part of their work.
    - P3Q25P3. paid work.

- **[P: 5, Q: 5] Helping**
  - **Includes:** Useful to others, good quality help.
  - **SQ:** Contribution to social coding platforms often is due to altruistic reasons. Some developers want to help others with a part of their code or challenge that they are stuck on; or the contributors help others to try and make others’ code more efficient.
    - P6Q25P2. I contribute to new projects I think would be useful to others
    - P8Q25P2. how much time I have to invest so I can contribute good quality help.
    - P10Q25P1. I contribute because I care about what's being made,
    - P18Q25P1. They look like they might be useful.

- **[P: 10, Q: 10] Expertise**
  - **Includes:** Contribute to libraries to fix bugs, provides patches.
  - **SQ:** Contributions GitHub often involve experts who are currently working on similar fields in their current job (or in a professional setting) do contribute to certain libraries to fix bugs or provide important patches.
    - P4Q25P1. What I end up using for work guides what I contribute to mostly
- P16Q25P2. but I will patch anything if I am able.
- P29Q25P2. anything where I see a problem that I can find a decent solution to in an hour or so
- P36Q25P2. Mostly contribute to libraries used in work-related projects
- P32Q25P1. I contribute to projects on GitHub if I can fix bugs

- **[P: 1, Q: 1] Open source project**
  - **Includes:** B2B solutions or development-oriented solutions.
  - **SQ:** There is one comment on the availability of the project and its relation to contribution. Contribution to solutions depend on the open-ended nature of source of the project; as the contributors believe in making a base contribution to the solution that would allow others to build upon.
    - P2Q25P2. I work on B2B solutions or dev oriented solutions, so the core of these solutions can be open-sourced so other devs can contribute to it.

- **[P: 8, Q: 8] Lack of time**
  - **Includes:** Do not contribute due to time constraints, SO answerer community can be cruel to answer seekers.
  - **SQ:** There are a few irrelevant comments on why people do not contribute to GitHub. Some of them also declined to state the reasons and others noted that they do not contribute because of lack of time and even others noted that they feel intimidated by the answerers and hence are hesitant to ask questions in the community.
    - P13Q25P2. I have my own projects to work on on the side, so I don’t really have the time or interest to contribute to other open source projects.
    - P23Q25P1. I rarely contribute because I don’t have enough spare time.
    - P25Q25P4. The SO answerer community can be quite cruel to askers; this has made me less inclined to participate recently.

**Contribution in Stack Overflow**

[Participants (P): 41, Quote (Q): 50]

- **[P: 3, Q: 3] Better answer**
  - **Includes:** Quest for better (optimal or efficient) solution to a problem.
  - **SQ:** Contributions in StackOverflow are sometimes because the developer was unable to find a suitable or efficient answer to their problem (question). Hence, they usually tend to go back, once they find a solution, and contribute a solution that they believe is better fit.
    - P17Q26P1 If I find someone who has asked the same question but the answers are not good, I like to return later to add my own answer once I have found it
- P6Q26P2. answer questions I've had for which there wasn't a complete answer

- [P: 3, Q: 3] Helping
  - Includes: Helpful, useful and save time.
  - SQ: Contributors to SO usually contribute when they believe that their contribution would be help other users and it would save someone else’s time.
    - P7Q26P1. Whenever I think my contribution would be useful to other users
    - P38Q26P3.it still will save someone’s time

- [P: 5, Q: 5] Hobby/fun
  - Includes: Enjoy it as a past-time activity; interests the person and reflects a positive mental state.
  - SQ: Contributors often contribute on SO because they enjoy doing so and because it interests them and it reflects their mind-set of helping others. Usually, they also do it as a past-time activity.
    - P31Q26P1. depends on my mental state
    - P9Q26P1. It interests me
    - P3Q26P1. All over the board, as a pastime

- [P: 19, Q: 20] Skill/expertise
  - Includes: Familiar programming languages and knowledge area where one feels qualified to answer.
  - SQ: Contributors on SO often contribute on programming languages and libraries that they know are qualified and have a certain level of expertise to answer the questions. Furthermore, contributors often contribute in areas that they believe are niche and there is not much work or contribution already done in such area.
    - P1Q26P1. Based on programming languages or libraries that I know and
    - P4Q26P1. I usually try to answer some questions in an area I know something about
    - P12Q26P1. the one I feel qualified
    - P16Q26P1. Again due to time constraints: I recently use Stack Overflow primarily as a way of documenting my own hard-won findings on a topic (e.g. posting and answering a question, then linking to it in the code), but am happy to get other answers and opinions added.
    - P20Q26P1 Needs to be a niche area where there isn't much expert competition. I can't compete with ppl writing a comprehensive answer in &lt;10s ;)
    - P25Q26P1. I participate in topics I know well, especially where I have niche expertise.
    - P37Q26P1. I participate on discussions I can contribute. I avoid discussions in which I can't.
• **[P: 3, Q: 3] Self learning**
  o **Includes:** Testing abilities, learning about technologies that one is interested and effectively sharing knowledge.
  o **SQ:** Contributors on SO are motivated by self learning and desire to share their knowledge with others and this drives their level and extent of contribution. Also, they usually tend to participate in technologies that they would like to learn and currently they do not have the bandwidth or chance to work on in their regular jobs.
    ▪ P16Q26P2. When I had more time I participated in C++ to learn about the C++11 standard and test my ability to share that knowledge effectively.
    ▪ P32Q26P2. I might also ask questions and immediately answer a question if I've just found a solution to a tricky question.
    ▪ P8Q26P1. I mostly participate in questions related to technologies that I want to learn but I don't have currently chance to work on at work so I can try to help solving real life problems with that technology.

• **[P: 6, Q: 6] Own needs**
  o **Includes:** Work based on personal needs and relevance to personal work.
  o **SQ:** Contributors often monitor SO to get answers for questions pertaining to their own projects. In essence, their activity on SO depends on the relevance to their current work or if their search on internet is not providing them necessary answers.
    ▪ P35Q26P1. I monitor stack overflow for questions related to my open source project. I seldom answer other questions.
    ▪ P14Q26P1. Based on relevance to current work
    ▪ P19Q26P1 If I land on the question during a search
    ▪ P22Q26P1 Missing key information on the internet
    ▪ P36Q26P1. Only participate in topics that relate to my own work

• **[P: 1, Q: 1] Alternative solutions**
  o **Includes:** New perspective.
  o **SQ:** There is one comment on this aspect, wherein the respondent has noted that their contribution to SO is to add “new perspective” or a new approach to the problem that they are attempting to solve.
    ▪ P11Q26P2. that my opinion adds a new perspective

• **[P: 7, Q: 8] Tagging system**
  o **Includes:** Some irrelevant observations, viz., management changes, lack of time.
  o **SQ:** Some responses were not appropriate and hence been classified as ‘miscellaneous’. Some of the responses are below:
    ▪ P23Q26P1. I don’t partecipate, mainly because I don’t have enough time.
    ▪ P30Q26P1. I no longer contribute to SO due to their management changes this year.
P13Q26P1. I used to answer more questions when I had more free time at work. Now I'm busy enough that I don't search for more questions to help out with.

P10Q26P1. I mostly don't participate in Stack Overflow any more. It's hard to find interesting questions to answer,

P32Q26P1. I ask questions on Stack Overflow if I think a problem is well suited and clearly defined.

- **[P: 1, Q: 1] Rewarding system**
  - **Includes**: contribute when technically challenging; however, SO rewards contribution to easy topics.
  - **SQ**: There is one response with regards to complexity of contribution and the reward mechanism on SO. The respondent noted that they contribute when the problem is challenging and mentally stimulating. However, the person has noted that SO rewards contribution to easy topics, which need to be further investigated.
  - P34Q26P1. Questions that I can answer, particularly if technically challenging. Unfortunately SO rewards contributing to easy topics over hard.

**Tags in Stack Overflow match the expertise**

[Participants (P): 20, Quotes (Q): 23]

- **[P: 1, Q: 1] Limited participation**
  - **Includes**: Respond only to popular questions.
  - **SQ**: Some participants only respond to popular queries.
    - P6Q29P1. I answered one very popular question with a very popular answer

- **[P: 1, Q: 1] Limitation of platform**
  - **Includes**: Only limited to specialized niche areas.
  - **SQ**: Some participants are hesitant to respond as they believe that SO only deals with specialized niche areas; anything outside of that is not well dealt with and hence they believe is a waste of time.
    - P33Q29P1. I do a lot more than I write on SO. SO is a waste of time outside specialized niches.

- **[P: 7, Q: 7] Gamification configuration**
  - **Includes**: Automatically selecting tags
  - **SQ**: Tags within SO are automatically selected and do not often reflect the current state or expertise of a contributor. Some examples:
    - P34Q29P2. My SO profile reflects expertise during PhD.
- P26Q29P1. I have two very highly-voted answers that seem to outweigh all the other stuff I've done on SO: one on VLAs in C++ and one on strings/Unicode in C#.
- P3Q29P1. Maybe historic skew, definitely skewed towards mainstream, popular tags
- P17Q29P1. I am not so interested in answering tagged questions as it tends to be a race and often the site regulars will beat you to it, so you put in a lot of effort answering for little gain. By focusing on questions that do not have good answers yet, I feel I get more value out of contributing.
- P18Q29P1. They reflect the tags that have the most questions asked.
- P20Q29P1. As I said, my key expertise is in areas where there are so many experts that all questions are answered very quickly by others. My top rated answers are on niche areas where I'm one of the only experts.
- P21Q29P1. Top tags metric doesn't work very well if you answered 2-3 questions many years ago that happened to have those tags on

- **[P: 4, Q: 6] Mismatch if tags with expertise**
  - **Includes**: Mismatch between tags assigned and the actual level of expertise.
  - **SQ**: Based on feedback, it is understood that sometimes there is a mismatch between tags assigned and the actual level of expertise of contributors.
    - P1Q29P1. I don't work professionally with Elm nor Haskell, and I'm at the beginner level for both.
    - P1Q29P2. Numpy and Django on the other hand, I do work professionally with and I am at the expert level for those.
    - P9Q29P2. my current expertise is mainly Go
    - P34Q29P1. In my day job I work with backend web tech (flask) and databases.
    - P35Q29P1. These are mostly related to the open source project I manage, not the daily work I have been doing for the past several years.

- **[P: 8, Q: 8] Miscellaneous**
  - **Includes**: Some miscellaneous observations: I don’t use SO; Don’t know my tags
  - **SQ**: There are some irrelevant comments and hence are classified as “miscellaneous”.
    - P2Q29P1. I don't use SO
    - P19Q29P1. I don't know what my tags are.
  - P27Q29P1. I found answers to questions that became popular because they concern a lot of people. Usually it's because I faced the problem, found a corresponding question on StackOverflow but the answers didn't help. After find the solution on my own, I would go back and add my answer.