Studying the Change History of Code Snippets on Stack Overflow and GitHub

by

Saraj Singh Manes

A Thesis submitted to
the Faculty of Graduate Studies and Research
in partial fulfilment of
the requirements for the degree of

Master of Computer Science

with

Specialization in Data Science

Department of Computer Science
Carleton University
Ottawa, Ontario, Canada
May 2020

Copyright ©
2020 - Saraj Singh Manes
The undersigned recommend to
the Faculty of Graduate Studies and Research
acceptance of the Thesis

Studying the Change History of Code Snippets on Stack Overflow and GitHub

Submitted by Saraj Singh Manes
in partial fulfilment of the requirements for the degree of

Master of Computer Science

Dr. Olga Baysal, Thesis Supervisor

Dr. Tony White, Internal Examiner

Dr. Michael Weiss, External Examiner

Dr. Majid Komeili, Chair of Defence

Carleton University
2020
Abstract

Stack Overflow (SO) is a popular Q&A forum for software developers, providing a large number of copyable code snippets. While GitHub is an independent code collaboration platform, developers often reuse SO code in their GitHub projects. These code snippets get revised and edited on both platforms after their creation and adoption. In this work, we investigate such evolution of SO posts and their adapted code snippets on GitHub. To accomplish our goal, we mine the SOTorrent dataset that provides connectivity between code snippets on the SO posts with software projects hosted on GitHub. We then create an evolutionary history of adapted code snippets by mining 26K GitHub projects to study their evolution, by creating a new dataset \textit{GHCodeSnippetHistory}.

Along with studying characteristics of GitHub projects that reference SO posts and what famous SO discussions can be found in GitHub projects. Our results demonstrate that, on average, developers make 45 references to SO posts in their projects, with the highest number of references being made within the JavaScript code. We also found that 79\% of the SO posts with code snippets that are referenced in GitHub code do change over time (at least ones), raising code maintainability and reliability concerns. The evolution of snippets on both platforms is driven by the original author of posts (SO) and adapted code snippets (GH). Finally, our results show the adapted snippets on GH and corresponding posts on SO evolve independently of one another.
To my parents and teachers for supporting and believing in me.
Acknowledgments

I, Saraj Singh Manes, would like to express my sincere gratitude to my amazing supervisor, Professor Olga Baysal, for her continuous guidance, advice, and friendly discussions throughout my thesis. Our joint vision for this project, her continuous efforts in providing me with valuable feedback and support made this work successful.

I am very grateful to my family - my parents, siblings and my friends. They gave me a tremendous amount of support by motivating and encouraging me.

I also want to thank all the thesis defence committee members for their valuable time.
# Table of Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abstract</td>
<td>iii</td>
</tr>
<tr>
<td>Acknowledgments</td>
<td>v</td>
</tr>
<tr>
<td>Table of Contents</td>
<td>vi</td>
</tr>
<tr>
<td>List of Tables</td>
<td>xi</td>
</tr>
<tr>
<td>List of Figures</td>
<td>xii</td>
</tr>
<tr>
<td><strong>1 Introduction</strong></td>
<td></td>
</tr>
<tr>
<td>1.1 Motivation</td>
<td>2</td>
</tr>
<tr>
<td>1.2 Motivational Examples</td>
<td>4</td>
</tr>
<tr>
<td>1.2.1 Example 1</td>
<td>4</td>
</tr>
<tr>
<td>1.2.2 Example 2</td>
<td>6</td>
</tr>
<tr>
<td>1.3 Research Roadmap</td>
<td>8</td>
</tr>
<tr>
<td>1.4 Research Questions</td>
<td>10</td>
</tr>
<tr>
<td>1.5 Thesis Contributions</td>
<td>11</td>
</tr>
<tr>
<td>1.6 Thesis Organization</td>
<td>12</td>
</tr>
<tr>
<td><strong>2 Background and Related Work</strong></td>
<td></td>
</tr>
<tr>
<td>2.1 Mining Software Repositories</td>
<td>13</td>
</tr>
<tr>
<td>2.2 Related Work</td>
<td>14</td>
</tr>
</tbody>
</table>
2.2.1 Code Evolution ........................................... 15
2.2.2 Code reuse .............................................. 15
2.2.3 Co-analysis of Stack Overflow and GitHub ............. 17
2.2.4 Mining Stack Overflow .................................. 19
2.3 Related Datasets ........................................... 20
2.3.1 SOTorrent .............................................. 20
2.3.2 GHTorrent .............................................. 22

3 Pilot Study ...................................................... 25
3.1 Mining SOTorrent .......................................... 26
3.2 Methodology ................................................ 27
3.2.1 Mapping SOTorrent with GHTorrent .................... 28
3.3 Data Selection .............................................. 29
3.3.1 Data Pre-processing ..................................... 30
3.3.2 Tags ...................................................... 34
3.3.2.1 Language Tags ....................................... 34
3.3.2.2 Concepts ............................................ 34
3.3.3 Sentiment Valence Analysis .............................. 35
3.4 Results ...................................................... 36
3.4.1 RQ1: How often do GitHub developers reference SO posts in their code? .............................................. 36
3.4.2 RQ2: What are the characteristics of GitHub projects referring to the SO discussions? .............................................. 37
3.4.2.1 Team Size ............................................ 37
3.4.2.2 Language ............................................ 38
3.4.3 RQ3: What types of SO discussions are most popular in the GitHub projects? .............................................. 39
3.4.3.1 Language Tags ........................................... 40
3.4.3.2 Concepts ................................................. 41
3.4.4 RQ4: Do SO discussions with code snippets evolve over time? 41
3.4.5 RQ5: How is the sentiment of SO discussions related to SO post evolution? ................................................. 42
3.5 Threats to Validity ............................................. 45
3.6 Future Work .................................................... 47
3.7 Conclusions ..................................................... 48

4 Methodology ..................................................... 49

4.1 Creation of Dataset ............................................ 50
  4.1.1 Code Context ............................................. 51
  4.1.2 Code Miner ............................................... 55
  4.1.3 Project Sampling ......................................... 59
  4.1.4 Details and Schema of New Dataset ......................... 61

4.2 Computation and Analysis ..................................... 62
  4.2.1 Code Miner Metrics ..................................... 62
  4.2.2 Gini Index ................................................. 64
  4.2.3 Shapiro-Wilk Test ....................................... 65
  4.2.4 T-test ..................................................... 65
  4.2.5 Time Series Analysis ..................................... 66

4.3 Feature Extraction ............................................ 68
  4.3.1 Commit message sentiment ................................ 68
  4.3.2 Derived features ......................................... 69
  4.3.3 Affinity .................................................. 69

4.4 Modelling ..................................................... 70
  4.4.1 Logistic Regression ....................................... 71
4.4.2 Random Forest ........................................... 72
4.4.3 AdaBoost .................................................. 72
4.4.4 Linear Discriminant Analysis .......................... 73
4.4.5 Support Vector Machines .............................. 74
4.4.6 Multinomial Naive Bayes Classification ......... 75
4.5 Model Evaluation ........................................... 75
4.5.1 Confusion Matrix ........................................ 76
4.5.2 ROC Curve ................................................ 77

5 Results ............................................. 79
5.1 RQ1: How much do code snippets evolve on GitHub and Stack Overflow? 80
5.2 RQ2: How often do reused code snippets evolve in GitHub and Stack Overflow? 84
5.3 RQ3: Who is driving the evolution of code snippets on both platforms? 88
5.4 RQ4: To what extent do reused code snippets co-change on both platforms? 91
5.5 RQ5: How accurately can we predict the likelihood of a change in reused code snippets? 94

6 Discussion ........................................ 97
6.1 Findings .................................................. 97
6.2 Implications ............................................... 98
   6.2.1 Correctness and stability of SO knowledge .......... 98
   6.2.2 Different acceptance criteria for SO answers ...... 99
   6.2.3 A dataset for code clone detection .................. 100
   6.2.4 Recommendations to software developers .......... 100
6.3 Threats to Validity ....................................... 101
**List of Tables**

<table>
<thead>
<tr>
<th>Table</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Example 1 – Revisions of a code snippet on Stack Overflow.</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>Example 1 – Reused code snippet in the GitHub project, no revisions.</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>Example 2 – a SO code snippet.</td>
<td>7</td>
</tr>
<tr>
<td>4</td>
<td>Example 2 – Reused code snippet in the GitHub project with two revisions.</td>
<td>8</td>
</tr>
<tr>
<td>5</td>
<td>Deriving GitHub project information.</td>
<td>56</td>
</tr>
<tr>
<td>6</td>
<td>Features of a training point $f_i$.</td>
<td>70</td>
</tr>
<tr>
<td>7</td>
<td>Data set summary.</td>
<td>79</td>
</tr>
<tr>
<td>8</td>
<td>Number of revisions per reused code snippet in GH.</td>
<td>80</td>
</tr>
<tr>
<td>9</td>
<td>Number of edits per SO code snippet.</td>
<td>82</td>
</tr>
<tr>
<td>10</td>
<td>Revision characteristics for reused GH code snippets.</td>
<td>85</td>
</tr>
<tr>
<td>11</td>
<td>Revision characteristics for Stack Overflow code snippets.</td>
<td>85</td>
</tr>
<tr>
<td>12</td>
<td>Gini coefficients for edit time distributions on both platforms.</td>
<td>88</td>
</tr>
<tr>
<td>13</td>
<td>T-test results for GitHub vs. Stack Overflow activities.</td>
<td>93</td>
</tr>
<tr>
<td>14</td>
<td>Evaluation metrics for different classification algorithms.</td>
<td>94</td>
</tr>
<tr>
<td>15</td>
<td>Logistic regression coefficients and p-values.</td>
<td>96</td>
</tr>
<tr>
<td>16</td>
<td>T-test results for GitHub vs. Stack Overflow activities.</td>
<td>112</td>
</tr>
</tbody>
</table>
List of Figures

1. Extent of edits in SO Posts. Adapted from [1]. ..................................  3
2. Research roadmap to study the usage and evolution of code snippets on the GH and SO platforms. ..................................................  9
3. Example of blocks in SO post. ...................................................... 21
4. SOTorrent’s dataset schema. ......................................................  22
5. GHTorrent’s dataset schema. ......................................................  23
6. SOTorrent and GHTorrent linkage. .............................................  29
7. Number of SO references in GitHub projects. .................................  31
8. GitHub team size distribution. ...................................................  37
9. How often do developers refer to SO? [Note: the # of SO references are normalized by team size]. .............................................  38
10. Language distribution in GitHub projects. ..................................... 39
11. Posts with matching language with projects. .................................. 40
12. Distribution of language tags in SO posts and GitHub projects. .......... 41
13. Most popular SO concepts being borrowed by GitHub developers. ........ 43
14. Discussion on the SO post with postID = 1152321. .......................... 44
15. Sentiment timeline for the SO post discussions with postID = 1152321. 45
16. The overall trend across selected SO discussions. ............................ 46
17. Evolution of code snippets in SO posts. ....................................... 47
18. Creation of code context from commits. ....................................... 52
19  Real example, origin commit, code in Code Context.  . . . . . . . . . . 54
20  Real example, update commit, Code Context version 1.  . . . . . . . . . . 55
21  Component Diagram of Mining process  . . . . . . . . . . . . . . . . . . 58
22  Distribution of a sampled dataset.  . . . . . . . . . . . . . . . . . . . . . 61
23  Schema of the created database.  . . . . . . . . . . . . . . . . . . . . . . 63
24  Gini Index from Lorenz Curve  . . . . . . . . . . . . . . . . . . . . . . . 64
25  The steps for generating a simulated time-series of GitHub commit
    activities.  . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 67
26  Confusion matrix showing four possible outcomes.  . . . . . . . . . . . . 77
27  ROC curve plot.  . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 77
28  Revisions of reused code snippets on GitHub.  . . . . . . . . . . . . . . 81
29  Revisions in changed code snippets (2+ revisions) on GitHub.  . . . . . 83
30  Revisions in posts on Stack Overflow.  . . . . . . . . . . . . . . . . . . . 83
31  Density plot of revision distributions over time on GH (in blue) and
    SO (in orange).  . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 87
32  Contributors on Stack Overflow.  . . . . . . . . . . . . . . . . . . . . . . 89
33  Contributors on GitHub.  . . . . . . . . . . . . . . . . . . . . . . . . . . 90
34  Co-change time-series of reused code snippets.  . . . . . . . . . . . . . 92
35  ROC curves for each classification algorithm.  . . . . . . . . . . . . . . 95
Chapter 1

Introduction

Stack Overflow (SO) consists of a large corpus of software development knowledge in the form of Question Answer (Q & A) posts and extensive commentary on posted knowledge in the form of comments. This volume of information on all subjects is available in the form of posts that include text and code snippets. A code snippet is a small block of code in a particular choice of programming language. The knowledge as form of text, accompanied by optional code snippets, serves as a description for a question or answer, asked by a registered user of the Stack Overflow platform. Stack Overflow consists of about 40 million such question-answer posts and the number keeps on increasing daily.

This vast information attracts a broad audience of software developers and engineers in the form of contributors or active consumers of the present knowledge. According to the official statistics of Stack Overflow [2], around 50 million developers visit Stack Overflow monthly to learn and share their knowledge. This large community of software experts with different levels of experience, participate not only to create the knowledge in the form of questions and answer posts but also help to validate created knowledge with upvotes and comments. They continuously monitor newly created threads and help authors of new posts in identifying weaknesses in
their answer scripts by submitting their commentary in the comment section of the post and edits to their answers.

Dependence on Stack Overflow is so severe that even new learners such as students prefer to learn new programming languages and frameworks from SO instead of diving into textbooks and learning them in the traditional way [3]. Ready-to-use code snippets provide an easy way to find solutions for daily programming problems that developers face all around the world. Naturally, this knowledge is transferred to other software artifacts. This work studies one such migration of knowledge and its relation to a particular type of software artifacts, i.e., open-source projects hosted on GitHub. GitHub is a git-based software repository hosting service that allow developers to collaborate on their project development. Providing a reference to the knowledge source which is well received by the broader community of experts, can increase chances of acceptance of a pull request in an open-source project. Hence, a large number of references exist in GitHub projects to Stack Overflow posts (we discuss this in detail in later chapters).

### 1.1 Motivation

Recently, Baltes et al. have released the SOTorrent dataset [4]. SOTorrent contains the version history of all SO posts created from the official data dump. The authors of SOTorrent have performed an initial analysis [1] on Stack Overflow posts investigating the frequency of edits and distribution of edits with respect to the creation time. One of the key results of their study was the evidence of the edits as 35.9% posts were edited at least once. These results of edits are summarized in Figure 1.

As shown in Figure 1, Stack Overflow posts undergo a large number of edits. We then may ask how these edits are affecting the code snippets included in those posts. To answer this question, we performed our initial study [5] as part of the MSR
Challenge 2019, where we wanted to characterize the GitHub projects that borrow the code snippets from SO posts (the study is described in Chapter 3). This analysis sparked our interest in studying the evolution of such code snippets in open source projects (described in Chapters 4 and Chapter 5) and whether the evolution of code snippets exhibits similar trends on SO and GH. In different research fields, term “evolution” holds quite different meaning, but in our work, we define evolution as the change history of code snippets. We use term change and evolution interchangeably in this work.

Table 3 and Table 4 outline one such example of a borrowed content in a GitHub project and how it evolves with the corresponding SO content. An important feature to note is that code comments in GH has an explicit reference to the original source of information in SO. The presence of such references in open source projects has
allowed Baltes et al. [1] to build a dataset that links the SO content history with the GH projects. Our work extends this linkage with the history of reused code snippets of referred GitHub projects. For brevity, we show only one revision of the snippet and its adapted version in Table 4. Please note that the code difference (i.e., diff of the code snippet) shown is only a snippet of the original diff.

1.2 Motivational Examples

To demonstrate evidence of the evolution of code snippets on Stack Overflow and GitHub, we provide two examples of code reuse from Stack Overflow posts to real GitHub project. In both examples, the original and adapted content receives a different number of revisions on both platforms, i.e., Stack Overflow and GitHub.

1.2.1 Example 1

Our first example shows that the original code snippet gets revised twice on Stack Overflow (see Table 1), while the adapted code snippet in GH undergoes no revision once being reused (see Table 2).
Revision # | Content of a SO code snippet
---|---
1 | //TEXT
class Number:
def __init__(self, number):
    self.number = number
def __eq__(self, other):
    """Overrides the default implementation"""
    if isinstance(other, Number):
        return other.number == self.number
    return NotImplemented

//TEXT
2 | class Number:
def __init__(self, number):
    self.number = number
def __eq__(self, other):
    """Overrides the default implementation"""
    if isinstance(other, Number):
        return self.number == other.number
    return NotImplemented

Table 1: Example 1 – Revisions of a code snippet on Stack Overflow.

[1](http://stackoverflow.com/questions/390250/elegant-ways-to-support-equivalence-equality-in-python)
<table>
<thead>
<tr>
<th>Revision #</th>
<th>Content of a GH code snippet</th>
</tr>
</thead>
</table>
| 1          | `def __eq__(self, other):
             
             
             
             """ Test this instance for equality with another. Note: This code
             was taken from http://stackoverflow.com/questions/390250/
             elegant-ways-to-support-equivalence-equality-in-python-
             classes.
             """
             
             if isinstance(other, self.__class__):
                
                return self.__dict__ == other.__dict__
             
             else:
                
                return False |

Table 2: Example 1 – Reused code snippet in the GitHub project, no revisions.

1.2.2 Example 2

While the original code snippet has no revisions after the creation of the answer post on Stack Overflow, the reused code gets two updates in GitHub.

[https://github.com/josenavas/qiime/blame/master/qiime/util.py](https://github.com/josenavas/qiime/blame/master/qiime/util.py)
Here is some code you can use to query the worksheets from my copy of your spreadsheet.

```python
#!/usr/bin/python
from gdata.spreadsheet.service import SpreadsheetsService
key = '0Aip8Kl9b7wdidFBzRGpEhoUVPaEg2X0F2YWtwYkE'
client = SpreadsheetsService()
feed = client.GetWorksheetsFeed(key, visibility='public',
    projection='basic')
for sheet in feed.entry:
    print sheet.title.text
```

**Tips** I find it really helpful when working with t ...

Table 3: Example 2 – a SO code snippet.

Table 4 shows the original version of the adapted code snippet with the link to the SO post in its comments.

<table>
<thead>
<tr>
<th>Revision #</th>
<th>Content of a GH code snippet</th>
</tr>
</thead>
</table>
| 1          | `def load.google.spreadsheet.mapping_file(spreadsheet_key, worksheet_name=None):
    
    """Some of this code is based on the following websites, as well as the gdata.spreadsheet.text_db module: http://stackoverflow.com/a/12031835"
    
    # TODO test if no connection
    gd_client = SpreadsheetsService()
    worksheets_feed = gd_client.GetWorksheetsFeed(spreadsheet_key, visibility='public', projection='basic')
    if len(worksheets_feed.entry) < 1:
        raise RemoteMappingFileError("The Google Spreadsheet with key "'{}'" does not have any worksheets associated with it." % spreadsheet_key)
    
    try:
        worksheets_feed = gd_client.GetWorksheetsFeed(spreadsheet_key, visibility='public', projection='basic')
    except gaierror:
        raise RemoteMappingFileError("Could not establish connection with server. Do you have an active Internet connection?")` |
| 2          |                                                                   |

Table 4: Example 2 – Reused code snippet in the GitHub project with two revisions.

### 1.3 Research Roadmap

The overall goal of our research aims at analyzing the evolution of code snippets on Stack Overflow and their adapted versions in GitHub. By usage, we mean copying (and possibly adapting) the code snippet from an answer post on SO and then pasting...
it into a public GH project \cite{6}. Figure 2 presents our research roadmap. We start our research with a pilot study \cite{7} to gain first insights into whether and how many SO code snippets developers refer to in their GitHub projects. By mining GHTorrent \cite{8} and SOTorrent \cite{1}, our initial results demonstrate that contributors refer to 45 posts, on average, in their GH projects. For this stage of the analysis, we linked the GHTorrent dataset with SOTorrent as described in the methodology of the pilot study (refer to Section 3.2.1).

To facilitate an effective quantitative analysis of the evolution of code snippets in SO and GitHub, we developed a tool named \textit{Code Miner}. \textit{Code Miner} extracts data from SOTorrent and GitHub to create a history of reused code snippet changes (described in Section 4.1.2). The dataset created with \textit{Code Miner} consists of both the adapted code snippets and their revisions along with the metadata of their commits.

Figure 2: Research roadmap to study the usage and evolution of code snippets on the GH and SO platforms.
1.4 Research Questions

Our research focuses on studying the evolution of code snippets on both Stack Overflow and GitHub to support a better understanding of code re-use across two collaborative development platforms, as well as developer behaviour on these platforms. The aim of this work is not detecting clones of any kinds but identifying the patterns in change and revision history of code snippets on both platforms. We thus answer a number of research questions through our work that is divided into two parts: 1) a pilot study, and 2) a large-scale code snippet evolution analysis. Our pilot study [7] is the preliminary analysis performed on SOTorrent and GHTorrent and is described in detail in Chapter 3.

The key highlight from the pilot study was that reused code snippets from Stack Overflow evolve a lot in their lifetimes. Therefore, we are interested in conducting a large-scale study on investigating evolution (and perhaps, co-change) of code snippets on SO and GH. To further extend our research, we address the following research questions:

- **RQ1: How much do reused code snippets evolve in GitHub and Stack Overflow?**
  
  While from pilot study we found that reused code snippets evolve in SO, RQ1 is related to gaining insights into the extent of such changes on both platforms, i.e., GH and SO.

- **RQ2: How often do reused code snippets evolve in GitHub and Stack Overflow?**
  
  The answer to this question would quantify the code evolution in terms of the revision size, the time it takes for code to be changed, etc.
• **RQ3**: Who is driving the evolution of code snippets on both platforms?

Results of this RQ would inform us about the ownership of the content on both platforms and the role of the original author in the evolution of code snippets.

• **RQ4**: To what extent do code snippets co-change on both platforms?

The answer to this question would inform us whether developers need to be on the lookout for changing SO content for respective platforms to mitigate the effects of inaccurate information migration.

• **RQ5**: How accurately can we predict the likelihood of a change in reused code snippets?

This answer investigates that given the revision history of a reused code snippet from a GitHub project, how accurately we can predict its future revision. Such change prediction tools can help to determine what code files are more susceptible to changes in future, thus providing data-driven evidence for improved maintenance of open source projects.

### 1.5 Thesis Contributions

The contributions of this research work are as follows:

1. This research topic supports better understanding of the change history of code snippets on two different collaborative platforms.

2. Development of novel techniques to extract a change history of reused code snippets in GitHub, by building *Code Miner* tool (Section 4.1).

3. Development of the revision history dataset for reused code snippets that can be further mined for conducting code clone detection and analysis (Section 4.1.2).
4. Analysis and comparison of the code snippet evolution on SO and GH (Section 5.2).

5. Empirical evidence of the co-change of code snippets across two different platforms (Section 5.4).

6. Use of machine learning to develop models that predict the evolution of reused code snippets in GH (Section 5.5).

7. A set of recommendations to software developers on maintaining their awareness on the implications of adapting code snippets from Stack Overflow into their GitHub code repositories (Section 6.2.4).

1.6 Thesis Organization

The rest of this thesis is organized as follows. Chapter 2 includes Background (Section 2.1) and Related Work (Section 2.2). Chapter 3 describes our pilot study, while Chapter 4 describes our Methodology. Results are presented in Chapter 5 followed by Discussion in Chapter 6. The thesis concludes with Chapter 7 outlining the summary of contributions (Section 7.1) and future directions (Section 7.2).
Chapter 2

Background and Related Work

This chapter presents the background of existing qualitative and quantitative research in various areas such as code evolution, code reuse from online platforms, migration of content from Stack Overflow and GitHub and vice-versa, as well as the related work summarizing different approaches followed by researchers and practitioners to address these topics.

2.1 Mining Software Repositories

According to Han et al. [9], data mining is the process of discovering interesting patterns and knowledge from large amounts of data. The source of data can include data dumps, large databases, data warehouses, the Internet, other information repositories, or the data that is streamed to data stores/systems dynamically. Another popular term used for data mining is knowledge discovery from data or KDD. The main components of the data mining process are: (1) data cleaning, (2) data integration, (3) data selection, (4) data transformation, (5) data extraction, (6) pattern evaluation, and (7) knowledge presentation.

Data mining is a major component of this research work as we mine both GitHub and Stack Overflow datasets to answer our research questions. Our mining process
involves all of the above mentioned components of the data mining process and is described in detail in Chapter 4.

Mining Software Repositories (MSR) is a field of software engineering research that analyzes the rich data available in software repositories with the goal of uncovering compelling and actionable information about software systems and projects. MSR applies the techniques of data mining on various data sources of software artifacts such as source code, test suits, version control systems (e.g., git), collaborative software development platforms (e.g., GitHub), bug tracking systems (e.g., Bugzilla), developer discussion forums (e.g., Stack Overflow) and many more.

This work is related to MSR as we mine large datasets of GitHub (i.e., GHTorrent) and Stack Overflow (i.e., SOTorrent), extract specific data, create new datasets for analysis, analyze the extracted data and present the results of this analysis.

2.2 Related Work

Frequently visited problems in mining software repositories is how developer make the most use of legacy code, finding bugs, and prediction of bugs based on the identification of bug inducing changes. Moritz et al. have discussed the mining of software repositories to identify the most used APIs. Similarly, Hsu and Lin have mined software repositories to find code snippets for code reuse. Using topic modeling, Chang and Blei used this information for debugging and traditional bug finding. Similarly, such information can be used for searching code histories for previous bug fixes, which can be supplemented with bug repositories, like Bugzilla.
2.2.1 Code Evolution

Source code encapsulates the behaviour description of the software system, and a change of source code naturally reflects the evolution of software behaviours, hence software evolution. Causation of software evolution perhaps cannot be extracted from software change history alone, as it needs to be supplemented with external knowledge sources like bug/feature databases, requirement documentation, and change in the source of information like Stack Overflow discussion forums. For example, Yan et al. [16] have tried to identify software evolution from VCS alone. The unit of analysis in their work was software contracts: API and interfaces, which expose software artifacts in use. As software contracts are a tool for specifying the functional behaviours of the product, the study of their evolution signifies how the behaviour of software as a unit changes over time. The contract also comprises of precludes and post conditions at the time of invoking contracts. Authors justify the use of API as a unit of analysis as if contract/API is modified as part of a fix; it represents new features and enhancements to software, while any other change which does not alter the contract substantially is considered as refactoring. Such assumptions, though, make analysis easier as they do not comment on the evolution patterns of code. Further, as paper outlines, their analysis of GitHub projects is limited by the presence of unit tests in the repository; hence, they are not an indicator of the general population. In this work, we do not argue about the causation of revisions in the source code of GitHub posts and the SO code snippets but rather try to identify common patterns of the evolution of code snippets in each platform.

2.2.2 Code reuse

As a software artifact evolves, structural/behavioural changes become more convoluted and time-consuming. As such, code reuse from the previous version or third
party vendors become an appealing problem \[17\]. Code reuse is formally defined as
"the use of existing software or software knowledge to construct new software" \[18\]. Various researchers have tried to develop different techniques on how to reuse existing code. Wang et al. \[19\] outline general principles that can be used to leverage an existing code base while developing new features yet reducing the code redundancy.

A similar parallel between reuse of code from an existing code base is to reuse code from online forums such as Stack Overflow. Several researchers have studied the reuse of code from Stack Overflow posts to GitHub projects \[18\] \[6\] \[20\]. Lotter et al. \[18\] have investigated the reuse of SO content in popular open-source Java projects. The authors have analyzed around 150K of Stack Overflow posts and detected their clones and reuse in 12 GitHub Java projects. Further, the central unit of analysis in this research work was clone detection. The authors have also tried to find clones among the projects along with clones from SO to GitHub projects. Clone results were 3.3% and 77.2%, respectively. While first value emphasizes that code reuse from Stack Overflow to GitHub, the later one indicates that the reuse of code among projects is much more significant. While the results of their analysis were substantial, the number of projects under consideration was low.

Kamiya et al. \[21\] argue that code-reuse leads to maintainability issues that could introduce multiple points of failure if code snippets used are “buggy” and constitute to almost half of the entire code snippets in their dataset.

Code reuse becomes more of a challenge if code is taken and reused from forums such as Stack Overflow due to potential code incompatibility and non-efficient solutions that can be tied to the lack of author’s understanding of the context. Further, such code is more explained on online forums with a textual description of logic to provide a better understanding, thus leaving things for the adaptor to figure out causing potential bugs to be introduced in the reused code. Ragkhitwetsagul et al. \[22\] argue that more and more engineers and developers tend to refer to Stack Overflow while
they do not understand certain aspects of code failure. Such lack of understanding
when adapting the content from online forums affects the quality of the reused code.

Several studies have been conducted about code reuse and Stack Overflow. For
example, Abdalkareem et al. [23] analyzed code reuse from Stack Overflow in mobile
applications and found that around 1.3% of the apps in their dataset were composed
of code snippets from Stack Overflow. Furthermore, the code borrowed and adapted
from SO was introduced to the established apps later in their lifetimes.

2.2.3 Co-analysis of Stack Overflow and GitHub

GitHub and Stack Overflow are equally important collaborative platforms for soft-
ware developers, yet they differ in context and use. GitHub, like SO, has an active
social component, but its primary purpose of service is to host and maintain software
projects and artifacts. Users hosting their software repositories on GitHub, often have
troubles with their software and its troubleshooting, thus they seek help for specific
problems on SO.

SO has been the focus of a lot of independent research studies, like using topic
modeling on SO discussions to categorize questions and answers [24] [25]. Empiri-
cally, there is little evidence on how information is being adapted from Stack Overflow
to GitHub. Some studies have tried to investigate the relationship between SO and
GH. Vasilescu et. al [26] studied the relationship between questions askers on Stack
Overflow vs. commit contributors on GitHub. They answered questions whether
participation on Stack Overflow relates to productivity on GitHub. The primary
significance of this work was the evidence of an overlap of the knowledge sharing eco-
systems. Less experienced developers on GitHub can ask questions on Stack Overflow,
and experienced ones can answer those. Others can be engaged in discussion, partic-
ularly comment sections, until the solution can satisfy the requirement of information
for the less experienced user who contributes to GitHub.
Yang et al. [27] attempted to investigate the impact of one platform over the other. To operationalize the problem as code snippets that exist on both platforms, they used cloning and measure of similarity as a measure of equality of information present on both sides. The authors focused their work on Python GitHub projects. They found that 86% of code snippets on GitHub were reused copies of the rest of the 14%, indicating a large amount of code duplication, while such duplication on Stack Overflow was only 1.3%. Further, only 1% of code snippets were the same on both platforms. This result indicates that clone studies do not provide a better indication of the adaptation of code snippets or snippets in general.

Like Yang et al. [27], several researchers state that clones lead to bug propagation and software maintenance issues [21], while others believe that clones are not harmful at all, rather beneficial [28], [29]. Chaiyong et al. [30] argue that there two directions of snippet migration: 1) snippet is cloned from a software project to a Q&A website as an example; or 2) code is cloned from a Q&A website to a software project to perform some task. Yet, clones may lead to licensing violations [6]. The authors attempted to investigate the toxic effects of such clones. Unlike other research on this topic, they try to trace the origin of code snippets and perform quantitative as well as qualitative analysis on code clone subjects. Their key results indicate that 19% of the participants on SO know their snippets are out of date, while 66% of SO visitors reported they faced troubles after referring SO snippets. This means that once code snippet is adapted, it evolves within the GitHub space with or without any regard to the corresponding Stack Overflow code snippet. Further, the authors state that 0.5% of code snippets from SO have been buggy, to begin with, when they were copied to GitHub projects.
2.2.4 Mining Stack Overflow

Many researchers have already mined Stack Overflow for various purposes. For instance, Greco et al. [31] have mined Stack Overflow along with IDE tools to predict developer behaviour and developed a recommendation system to recommend useful content from the platform. Topic modeling is a standard tool of choice when it comes to analyzing SO content. Zou et al. [32] use topic modeling to analyze non-functional requirements (NFRs) on Stack Overflow and how they evolve. Vásquez et al. [33] used topic modeling techniques to extract hot topics from mobile development related questions, while Rahman et al. [34] use it for analyzing unresolved questions on the platform and built a prediction model of whether a question is more likely to be unanswered.

Soni et al. [35] mine the comment section to determine triggers for edits of SO posts. We described a similar analysis in the discussion section but performed a sentiment analysis to analyze the attitude of the community towards the quality of content. Shao et al. [36] mine SO data dump to propose a recommendation service for learning programming study with the help of Stack Overflow customized for the learner’s point of view. Yin et al. [37] offer another recommendation system that can suggest source code based on queries from Stack Overflow code snippet collection.

Stack Overflow is a hot topic among the MSR research community, because of its sheer size and large interaction base. Any analysis performed on Stack Overflow has far-reaching consequences because of the significant dependence of software developers and engineers on the information that this platform has to offer. Yet, we found there is a lack of work on the topic of the evolution of code snippets within the GitHub space. Our novel approach aims at identifying evolution trends and goes beyond the step of clone detection.
2.3 Related Datasets

To understand the evolution of code snippets on two different collaborative platforms, we mined two different datasets named SOTorrent and GHTorrent, linked them together and built a new dataset $\textit{GHCodeSnippetHistory}$. Details regarding SOTorrent and GHTorrent are described in the following section. Linkage details of both datasets are described in the $\textit{Pilot Study}$ chapter 3, while the construction of $\textit{GHCodeSnippetHistory}$ is described in the $\textit{Methodology}$ chapter 4.

2.3.1 SOTorrent

SOTorrent \cite{5} is an open dataset based on the official Stack Overflow data dump. SOTorrent provides access to the version history of Stack Overflow content at the level of a whole post and an individual post block \cite{1}. A post in SOTorrent is defined as one entity in the discussion thread such as a question, comment, or answer to the question. Thus, a discussion thread on Stack Overflow is a set of posts in SOTorrent. Further, a post is divided into blocks. A post block can be one of the two types: 1) code block or 2) text block. A post block typically includes code snippets (code blocks) and the textual content (text blocks) and is dependent on the author’s formatting style of the post. Figure 3 shows an example of code blocks and text blocks in a SO post.

The version history of a Stack Overflow post in SOTorrent is defined in terms of the version history of blocks. Each edit, whether an existing post block is edited, created or deleted, creates a new version to the entire post and that particular post block. After segregating a post into blocks, Baltes et al. \cite{5} used matrix similarities to identify the predecessor of a block and thus creating a version graph of post blocks.
The schema of the SOTorrent dataset is presented in Figure 4. Apart from the version history of posts, SOTorrent provides some related metadata on posts, users who drive the evolution of SO posts and status of their contributions on the platform.

Along with the version history of content, SOTorrent links SO posts to external resources in two ways: 1) by extracting external URLs from post content and (2) by providing a table with links to SO posts found in the source code of GitHub projects, called PostReferenceGH in SOTorrent’s schema. This linkage of SO posts to GitHub projects is a significant starting point for us, as it allows us to map SO posts to GitHub projects. As described in later chapters, we use these links to build linkage between two significant datasets, (1) GHTorrent, a dataset of GitHub projects, as described in Section 2.3.2 and (2) GHCodeSnippetHistory dataset, a dataset we built as part of this work, containing GitHub’s code snippet version history and changes.
2.3.2 GHTorrent

GHTorrent \cite{38} is an online queryable mirror of the GitHub's events. The GHTorrent project aims to create a scalable offline mirror of the GitHub's live event streams and store it as persistent data. GitHub has recently become a significant software repository hosting site for open source projects. Being an open-source platform, GitHub provides a REST API for users to collect various git events. Git events include meta-information about various interactions happening on the platform, for example, commit to a project, the opening of a pull/issue request. These events are used by the GHTorrent project to create the GHTorrent dataset. The project uses the aforementioned GitHub API to collect raw data and extract, archive and share queryable research data. As part of this project, the authors have collected more than 900GB of data and 10 GB of metadata.
In Figure 5, the schema of the GHTorrent database is presented. This schema follows GitHub’s API structure; most entities belong to projects and contain entities corresponding to actions initiated by the user. The schema also identifies the user relation to commits and pull requests. GHTorrent uses a MongoDB database, which offers the added benefit of enabling querying its raw data. For each retrievable item, dependencies according to the schema are specified, thus enabling all data tables to be filled in sync. For example, if the project is retrieved, it is essential to retrieve its users and contributors, similarly, to retrieve a pull request data it is essential to retrieve the project data.
Many researchers have already exploited the GHTorrent dataset. Some of the most interesting research efforts have been focused on the analysis of drive-by-commits, analysis of test incentives and pull development analysis [8] [39] [7] [40] [41].
Chapter 3

Pilot Study

According to the SOTorrent \cite{5} data (as of December 2018), Stack Overflow hosts over 42 million posts. These posts may consist of code snippets, text descriptions and links to external sources for further references, thus, providing a good pool of ready-to-use code snippets, informal documentation and discussions on various code-related concepts.

Based on our initial mining and exploration of SOTorrent, we found approximately 6.5 million references to Stack Overflow posts in the GitHub projects, with some projects even referencing more than 6,000 SO posts. This scale of interdependence between open source projects and Stack Overflow discussions motivates us to conduct an empirical study investigating the nature of SO references in the GitHub code repositories. In this pilot study, we investigate the characteristics of Stack Overflow posts that are referenced in the GitHub code repositories, as well as the characteristics of the GitHub projects that refer to the SO posts. To conduct our study, we linked SOTorrent, an official data dump of Stack Overflow, to the publicly available GitHub data dump, called GHTorrent \cite{8}. Such mapping of SO to the GitHub projects can allow us to gain insights into rich information about developers and their projects such as projects’ commit history, team size, etc.
SO Torrent dataset \[5\], as explained in Section 2.2, was developed from the SO official data dump \[1\]. It provides block level information of Stack Overflow posts and their version histories.

In this pilot study, we address the following research questions:

- RQ1: How often do GitHub developers reference SO posts in their code?
- RQ2: What are the characteristics of GitHub projects referring to the SO discussions?
- RQ3: What types of SO discussions are most popular in the GitHub projects?
- RQ4: Do SO discussions with code snippets evolve over time?
- RQ5: How is the sentiment of SO discussions related to SO post evolution?

The first two research questions can offer insights into whether GitHub developers borrow code or concepts from SO, and if so, how often they do this and what GitHub projects are more likely to include references to SO posts in their codebase while RQ3 can help us to understand what concepts developers reference most. RQ4 addresses the possibility of bug migration from one platform to another and thus provides insights into the problem of code snippet maintainability. Finally, RQ5 explores possible triggers for the edits of Stack Overflow posts by focusing on the comment sections of a post.

### 3.1 Mining SO Torrent

The release 2018-02-16 of the dataset contains the version history of all 38.4M questions and answers in the official SO data dump. It contains 60.2M post versions and 186.9M post block versions, ranging from the creation of the first post on July 31, 2008 until the last edit on December 1, 2017. They extracted links to 11.1M distinct
URLs from 19.4M different post block versions and further identified 5.8M links to SO posts in 430.5K public GitHub repositories.

SOTorrent contains all tables from the official Stack Overflow data dump, published December 1, 2017, as schema diagram in Figure 4. In the SO dump, one version of a post corresponds to one row in the table PostHistory. However, that table does not only document changes to the content of a post, but also changes to metadata such as tags or title. Authors of SOTorrent selected edits that changed the content of a SO post, identified by their PostHistoryTypeId. Authors split the content of each version into text and code blocks and extracted the URLs from all text blocks using a regular expression. One row in table PostReferenceGH represents one link from a file in a public GitHub repository to a post on SO. To extract those references, authors utilized Google BigQuery, which allows to execute SQL queries on various public datasets, including a dataset with all files in the default branch of GitHub projects. To find references to SO, they applied the following regular expression to each line of each non-binary file in the dataset, as mentioned in their paper [1]:

```regex
(?i: https?:// stackoverflow\.com /[^\s\(\)\.,\"\*])
```

Because there are different ways of referring to questions and answers on SO, i.e., using full URLs or short URLs, they mapped all extracted URLs to their corresponding sharing link (ending with /q/ for questions and /a/ for answers) and stored that link together with information about the file and the repository in which the link was found in table PostReferenceGH.

### 3.2 Methodology

In this pilot study, both mentioned datasets SOTorrent and GHTorrent have been explored in conjunction. First, looking at SOTorrent, we calculated a rough estimate...
of SO posts references in open source projects. This distribution is shown in Figure 7.

To investigate our research questions, a mapping was established between SOTorrent and GHTorrent.

We now describe these datasets, provide details on how we map them, and how we extract various characteristics of the GitHub projects such as team size, language tags, as well as attributes of the SO posts including tags.

### 3.2.1 Mapping SOTorrent with GHTorrent

SOTorrent and GHTorrent are created and being maintained independently. There is no foreign key that can be used to run a join operation for mapping both datasets. While SOTorrent mentions a “path URL” to the GitHub’s raw dump file, unfortunately, it can not be used as a foreign key. Yet, we can try to extract such key from this “path URL”. Below is an example of the GHUrl entry in PostReferenceGH table:

```
https://raw.githubusercontent.com/RationalAsh/freeIMU/master/debug/decode_float.py
```

From this URL, we can identify a pattern such as “https://raw.githubusercontent.com/⟨user-name⟩/⟨project-name⟩/⟨branch-type⟩/⟨relative-path-to-file⟩”. We then can extract “⟨user-name⟩/⟨project-name⟩” as the user and project names from this pattern. This “derived” project name can then be used to compute a join operation on the PostReferenceGH table in SOTorrent with the Projects table in GHTorrent with the join condition specifying the “derived” name in PostReferenceGH to be the same as the name of the project in GHTorrent. Details of result of this step are stated in Section 3.3.1.

Having joined both datasets provides great insights to attributes of projects in question. Next step is to compute contributor information for each project. This is again derived from commits table in GHTorrent dataset. For each project, unique ID of an author of a commit to that project, is counted as one contributor/developer.
Then, all such contributors are added to count the contributing team size of project. Analysis and statistics of this step are again mentioned in Section 3.4.2.1.

### 3.3 Data Selection

As stated earlier, by mining SOTorrent, we extracted 6,039,434 references to external sources (mainly GitHub) with a total of 439,646 different GitHub repositories having references to Stack Overflow. Figure 7 shows the number of GitHub projects that reference SO posts. To minimize potential noise in the data, we tried to eliminate outliers by removing the lower 5% and upper 5% of the data points, i.e., projects that have too few references to SO (e.g., 189,349 GitHub projects include only one reference to SO) or too many references (e.g., over 600). After filtering out these outliers, our final dataset contains 556 GitHub projects with the total unique references to 21,537 SO posts. On average, a GitHub project contains 176 references to SO posts, while the minimum and maximum numbers are 15 and 565 references, respectively. Prepossessing steps taken while mapping both datasets and selecting the final set of 556 projects and 21,537 posts are outlined in the following Section 3.3.1.
3.3.1 Data Pre-processing

A very first raw analysis of PostReferenceGH shows the scale and size of data being considered. The total number of references in this table was in the millions while the number of projects or repositories was in the thousands. This is a considerable amount of data to analyze. Further, Figure 7 depicts the inverse relationship between the number of repositories vs. the number of Stack Overflow references in those repositories. The plot in Figure 7 shows that a very large number of projects refer to a minimal number of posts. For example, the number of projects referring to only one Stack Overflow post is 189,349, and those who refer to two Stack Overflow posts are 83,193, with three Stack Overflow references we have 38,055 projects.

Increasing reference count by one leads to a decrease in number of projects by almost 43%, indicating that majority of projects have one or two references to Stack Overflow posts. A similar but reverse trend can be observed on the other end of the graph where a very small number of projects has a very large number of SO Post references. It seems natural to use this number of references per project to use a criteria to filter potential outliers.

**Step 1:** The average of number of references is around 250. To proceed with final data selection, we sorted the project count vs. SO count table by SO count. Then in this dataset we removed first and last 5% entries, thus moving forward with 5-95% of data. In this step we reduced the number of projects from 439,646 to 12,746 projects. Let’s call this as Set A. The average number of SO references in this subset is 178 per project. The minimum number of references is 15 and the maximum is 567. Thus, a very large number of projects have SO references less or equal to 14. Studying evolution of code snippets in these omitted projects provides a small sample space if the reference density is compared with the project size. Further, these projects do
Figure 7: Number of SO references in GitHub projects.

not seem to rely on Stack Overflow for concepts and ideas, thus making study of SO posts and their use less interesting.

So far, filtration of references and repositories is performed on the SOTorrent’s PostReferenceGH table alone; now, we explain further filtration from the GHTorrent’s side after linking both datasets together using techniques explained in Section 3.2.1.

Having selected this data only from SOTorrent, the next step is to analyze it in combination with GHTorrent. This implies a join operation between the PostReferenceGH table in SOTorrent and the Project table in GHTorrent. This imposes another challenge due to the scale of data in GHTorrent. The size of this table in its CVS file format is 18GB. Tools on the Linux VM environment fail to load and then process this table as cache size increases multiple folds. To achieve better data processing, we use custom-built scripts and programs to process an offline copy of GHTorrent’s dataset. Join key is derived in the form of $<\text{user-name}>/\langle\text{project-name}\rangle$, as explained in Section 3.2.1.
We were able to map 12.5 K projects to 11.9K projects in the GHTorrent dataset. Yet we have lost the data for 835 projects. A possible explanation for this loss could be that both datasets had been created independently and have different timestamps, thus some projects (i.e., 835 projects) might have moved from a public to a private space as we found no entry for these in the public dataset of GitHub, i.e., GHTorrent.

To consider further filtration, we choose team size as a criterion, which is derived from GHTorrent’s commit meta-data information as follows.

**Determining the Project’s Team Size.** To extract the team size of each project from GHTorrent, we considered the number of commits (i.e., contributions) of each project. We identified all unique authors of all commits and considered their count as our “team size” metric. Another possible way to determine the contributing authors is to consider pull-request information, but not every commit gets merged to the master branch with a pull request, as it highly depends on team culture and practices.

Mining the Commits table again imposes the same challenge as we faced when mining the Project table. The size of the commit table is 29GB in CSV format. Again, using custom scripts, we were able to reduce the size of this table to 1.1 GB. In GHTorrent, each user is assigned an ID, although anonymized. While calculating team size, each unique author ID is counted once.

By completing this calculation of team size, we were able to retrieve contributor information for 4,552 projects. This is only the 38.8 % of total projects. This is a considerable reduction and an unexpected one. To double-check the consistency of data at this stage, we went back to the original the Commits table in GHTorrent and calculated the total number of projects for which commit information is available in this table. The results obtained for the entire dataset of GHTorrent were consistent to the ones we obtained earlier. In GHTorrent, there are around 83M projects with some meta-data and only 47M of these (52%) have commit information in Commits table.
Although both percentages do not match exactly, it does indicate that the Commits table in GHTorrent does not have all the information about all of the projects. The reason for this missing information is not documented in the GHTorrent project. From the documentation of the project, we found the “#issue32” incident. In this, some mischievous people/organizations extracted committers’ email addresses from the initial GHTorrent dataset and started spamming/targeting those contributors. This was deemed to be a violation of General Data Protection Regulation (GDPR) and online privacy, and as a result GitHub has disabled APIs retrieval of “identity” information about its users.

The distribution of our team size of all projects follows the exponential pattern, as shown in Figure 8. We have a large number of projects for which the number of developers or contributors is one, and similarly, there are some projects for which this number is 2,969. Both extremes tell us a lot about the nature of the GH projects. Projects that have a single or very few developers represent some sort of side projects, these are not good indicators of collaboration with others. Thus, the intent of a single developer to refer to a SO post has no or little impact on software development in teams. Yet such practice shows a good habit of addressing the code ownership.

**Step 2:** We sorted all projects with Stack Overflow references by their team size, derived from GHTorrent, and dropped 5% projects on both ends to proceed further with our analysis (thus eliminating obvious outliers from the data). Our final subset of GitHub data contains 547 projects, with team size varying from 5 to 726 developers.

At this stage, we have all commit and developer information of projects which have references to Stack Overflow. The total number of unique Stack Overflow posts referred to in these 547 projects is 21,576. We wanted to analyze posts from their tags point of view, as well as code snippet evolution point of view.
3.3.2 Tags

To extract post tags for our analysis, we mined the Post table (about 93GB of uncompressed XML data) of SOTorrent. Our subset data from SOTorrent contains references not only to the “question” posts but also to the “answer” posts. We have 26% of the total referred posts as answers; the rest of the references are question posts. However, tags for the posts in SOTorrent are associated with the “question” posts only. Thus, we extracted tags for the “answer” posts by gathering the tags from the corresponding parent “question” post. We then classify post tags into two categories: language tags and concept tags. Such classification would allow us to better understand what type of SO posts GitHub developers reference in their code.

3.3.2.1 Language Tags

While mining tags, we are more interested in language and concept tags. For each post, we classify the tags as being a concept or language tag. While doing so, we needed as a reference to define what a language tag would be. To eliminate any ambiguity, we used Wikipedia [42] as a reference for identifying all programming languages and their abbreviations. Any tag matching a language name or its abbreviation is classified as a language tag. Some cases were resolved manually, e.g., <C++> and <C++11> tags are both identified as C++ language.

3.3.2.2 Concepts

Any tag that is not a language tag is considered to be a concept tag. However, we soon noticed that such classification is not quite accurate as we found a large number of concepts related to frameworks or platforms, e.g., <android> being the most frequent one in our dataset. Since we are more interested in the concepts related
to code development, we manually filtered out tags that were relevant to platforms, frameworks or tools.

The analysis of this subset is discussed next, in Section 3.4.3.2. Although this final subset of projects and posts is small, its selection was at random without introducing bias. Results and trends observed from the analysis of this subset should be applicable to a data set at large.

### 3.3.3 Sentiment Valence Analysis

To explore sentiment for edits of SO posts, we have performed a sentiment analysis on comments on corresponding post. Performing sentiment analysis on comments and edits of an SO posts’ discussion had to be different than just looking at positive and negative sentences. The goal was to identify the times when there was a change in sentiment within the discussion on a post, therefore performing traditional sentiment analysis by classifying each sentence’s sentiment polarity was not a desirable solution. We explored different kinds of sentiment analyzers with the main goal of finding out how sentiment changes over time and creating a sentiment time line for the discussion within a SO post. The R package named Syuzhet allows this type of analysis by looking at the “latent structure of narrative by means of sentiment analysis” [43]. The author of the package describes that “it reveals the sentiment shifts within a sequence of sentences [or SO post discussion thread] by extracting sentiments and creating sentiment-based plot arcs from text” [43]. The Syuzhet package contains four sentiment dictionaries and provides a method for sentiment extraction developed by the NLP group at Stanford. Most methods within this library depend on the coreNLP package. For our research purposes we used the default “Syuzhet” lexicon, which was developed by the Nebraska Literary Lab.

1[https://cran.r-project.org/web/packages/syuzhet/vignettes/syuzhet-vignette.html](https://cran.r-project.org/web/packages/syuzhet/vignettes/syuzhet-vignette.html)
2[https://stanfordnlp.github.io/CoreNLP/](https://stanfordnlp.github.io/CoreNLP/)
Using this R package, we created a sentiment time line for the discussion in each analyzed SO post. The Syuzhet package extracts sentiments and calculates a sentiment valence score at each point on the time line generating a sentiment valence plot. Using these sentiment valence plots all the sentiment changes over time are visually observable, while the individual sentiment valence scores are analyzable.

3.4 Results

Mapping and combining two very large data sources such as SOTorrent and GHTorrent provide a wide research landscape for data mining. For the scope of this pilot study, we limited our data mining and analysis to four research questions. The results for each research question are presented in this section.

3.4.1 RQ1: How often do GitHub developers reference SO posts in their code?

Count of projects vs. count of SO references is inversely proportional to each other in nature. Figure 7 shows the distribution of SO references per developer across GitHub projects. We can see that most of the projects have around 30 SO references per developer (on average, each developer has 45 references to SO in their code), while some developers reference SO discussions more than 250 time in their code. Moreover, we determine that the average number of SO references per project is 176 (median is 144 references, while first and third quartiles are at 76 and 241 references, respectively).

More than 10% of projects refer to 15 SO posts, with first, third quartile are 76 and 241 references respectively. Median stands at 144. This shows that open source projects in GitHub frequently reference SO discussion posts. One point to be noted here that term reference to a post means that in project’s code there is mentioned
link to SO post. Without this link, it is not possible to establish a link to Stack Overflow. Thus, the actual frequency is only higher than the derived distribution.

3.4.2 RQ2: What are the characteristics of GitHub projects referring to the SO discussions?

For this RQ2, we are interested in determining what type of GH projects are more likely to reuse code from SO.

3.4.2.1 Team Size

Figure 8 shows the team size distribution on GitHub. The team size varies between 5 to 726 developers. Please note that only actual contributors to code are counted as developers. Median team size is around 45 developers, while first and third quartiles are at 24 and 84 developers, respectively. The larger the team, the more SO references are present in code. We speculate that as the team size increases more developers are involved in discussing the referenced SO posts. This can also affect the stability
of the code snippet or concept borrowed and reused from Stack Overflow. Figure 9 shows the number of references per developer in all the projects in our dataset.

### 3.4.2.2 Language

The language of a project is extracted from the file extension that includes a reference to a SO post. Results are also mapped to the GHTorrent’s Projects table with the language field. Some GitHub projects have code written in multiple languages. Thus, we believe that looking at the language of the file that has a SO reference gives a more accurate picture of what code (in terms of a programming language) is more likely to reuse code snippets or concepts from Stack Overflow.

Figure 10 shows the language distribution in GitHub projects with SO references. The majority of the GitHub code repositories that reference SO discussions are related to languages like JavaScript, PHP, Python, Java, CSS, C++, HTML, C, Ruby, C#, etc.
The unknown portion of pie chart in Figure 10 contributes to the percentage of files present in our dataset that have a reference to SO post, yet their extension does not match any known file extensions, hence we are unable to conclude the programming language of such files. A decent proportion of these files have a .md extension.

3.4.3 RQ3: What types of SO discussions are most popular in the GitHub projects?

We answer this research question by considering the attributes of SO posts — what makes a post a popular post in GitHub projects? This analysis is based on the tags of SO posts that we classified as language or concept tags. As mentioned earlier, tags related to frameworks or platforms are excluded from the analysis.
We first observed that the majority of the SO posts in our dataset refer to the projects with the matching language tag. 79.5% (1,344,623) of all SO references in our dataset have the same language tag as the language of the project’s code. This number constitutes to 79.45% of total number of references. Figure 11 shows this distribution. The rest of 20.5% of the SO references we categorized as concepts or conceptual references, i.e., when code snippets were adapted to another language domain.

This motivates us to look at most famous concept tags of posts. We also noticed that the language distribution in projects is similar to the language tag distribution of the SO posts, as shown in Figure 12 with the exception of *typescript* and *json*. Here, we notice an interesting trend that the language tag distribution in projects vs. the language tag distribution in posts are quite similar. Both scales are not normalized in Figure 12 magnitudes represent the number of projects and posts, respectively.
3.4.3.2 Concepts

To better understand what concepts are being borrowed from Stack Overflow, we look at top 30 most popular posts with non-programming language tags. We have selected top 30 most frequently occurring tags as most frequent concepts. In first run, we had some frameworks related tags like “android”. But those tags were manually filtered out and only generic tags were considered in final result as concepts. Figure 13 shows most frequent concepts that are reused by GitHub developers with top popular concepts being related to Linux OS, string and regular expressions.

3.4.4 RQ4: Do SO discussions with code snippets evolve over time?

Investigating the possibility of bug migration from Stack Overflow to GitHub projects can offer some insights about maintainability of copied or reused code snippets. To answer this, we consider all SO posts and their evolution which can be tracked by
analyzing the versions of code blocks of those posts. In our dataset, around 27% of the SO posts which are referenced in GitHub are of an answer type, while the rest 73% of the posts are questions. Developers are more likely to refer to the question posts as a proxy for referencing the overall discussion on a specific SO topic. However, to determine the evolution of code snippets, we consider accepted answers regardless of whether a developer referenced a question or an answer post in their projects. We believe that it is the answers to those questions which developers refer to, thus to look at code snippet we decided to focus on the accepted answers to those question posts. Then in these answer posts, we focus on the number of code versions.

We found that 13,821 SO posts in our dataset (i.e., 64% of all SO posts) contain code snippets. Figure 17 demonstrates how many times code snippets change over time. Out of all posts, the code snippets have changed at least once in 10,858 posts (79%) and more than 20 times in 1,193 (9%) posts. The most striking finding in our work is the extend of the code snippets that evolve, i.e., 79% of code snippets in Stack Overflow are changed over time. This finding sheds some light into the evolution of code snippets. One of the implications of this work is to improve developer awareness of the code snippet maintainability. We advise developers who reuse SO code snippets to keep an eye on those snippets and discussions in Stack Overflow since they do evolve over time.

3.4.5 RQ5: How is the sentiment of SO discussions related to SO post evolution?

When a Stack Overflow post evolves, the whole discussion related to this post, which often happens within the comment section, grows over time. Baltes et al. [1] mentioned that most of the comments happen around the edit time of a post. This
Figure 13: Most popular SO concepts being borrowed by GitHub developers.
suggests that there is a relationship between the post edits and comments. To attribute that comments cause edits, we performed a *Sentiment Valence Analysis* on comments. Performing a sentiment analysis on comments and edits of a Stack Overflow posts’ discussion had to be different than just looking at positive and negative sentences. The goal is to identify the times when there was a change in sentiment within the discussion on a post, therefore performing a traditional sentiment analysis by classifying each sentence’s sentiment polarity is not sufficient. To study how sentiment changes over time, we created a sentiment timeline for the discussion within a SO post. The R package named Syuzhet was used to reveal the sentiment shifts within a sequence of sentences (or discussion thread on SO posts) by extracting sentiment and creating sentiment-based plot arcs from the text.

![Figure 14](image-url)

At a particular timestamp, the value of emotional or sentiment valence is commutative of the entire thread at that time. As the discussion thread grows, attitude varies, each point on plot corresponding to a new comment on thread. The red lines
Figure 15: Sentiment timeline for the SO post discussions with postID = 1152321. In Figure 15 show the edit timestamps of the example post. Further, the points in Figure 15 mark timestamps of comments in corresponding discussion of the related post. We see that the sentiment of the discussion varies a lot. Initially, the sentiment is going into both positive and negative directions, while at the end the sentiment settles at a positive value. Figure 16 presents a histogram of the first 20 edits for all SO posts with accepted answers in this dataset, plotted with their overall sentiment polarity of the discussion thread at that time. As the volume of edits increases, the overall sentiment of the post discussion thread of becomes more positive.

## 3.5 Threats to Validity

The main threat is related to the lack of the updated data on the project contributors in GHTorrent. Instead, we extracted the team size from the Commits table of
Figure 16: The overall trend across selected SO discussions.

GHTorrent that we believe is accurate in identifying the actual code contributors. Yet, for 55% of GitHub projects we found no commit information. We do not know why commit data is missing for these projects.

We could consider other means to determine the team size. For example, we could mine Project_Commits and Pull_Request tables. However, our brief exploration and analysis showed that even these sources do not contain all the information for all of the projects. For example, Project_Commits had information about 64,597,126 from a total of 83,624,114 projects. Further, not every commit happens via pull request, thus neither Pull_Request nor Project_Commits can be reliable for determining the team size.

Another threat is related to the way we treat GitHub projects and their forked repositories, i.e., each project is considered and analyzed independently without considering its origin. If a project has been forked from another project, they carry over
the same code base. Therefore, our data set may have some duplicated SO references from the forked children.

### 3.6 Future Work

Answering five listed questions in this pilot study provides a path to investigate a number of in-depth research questions. One very important question we are interested in is maintainability of reused SO code snippets in GitHub projects. We will be expanding our work to answer questions related to code maintainability. Although RQ4 provides 50% reassurance that a very small number of bugs are migrating from Stack Overflow to GitHub, if we only take edit as criteria of bug identification in a post. There is still 50% chance of bug introduction to a code base if the code is copied from Stack Overflow. Thus, the code needs to be adapted before it is incorporated in a software project. Any edits to the reused code snippet would be considered as contributing to the evolution of that snippet in GH. Further, an adapted code snippet can evolve independently or it can also be affected by the potential SO post edits.
Again if the latter is true, it would mean that the reused code evolves along with post and thus developer needs to keep an eye on post edits.

The attitude of a developer towards a referred code snippet can also be assessed with sentiment analysis of comments around SO post reference.

All of this quantitative analysis can provide more insights into usability and implications of SO post reference. We also wish to expand on quantitative analysis along with qualitative analysis in our future work.

### 3.7 Conclusions

This pilot study investigates whether and how Stack Overflow posts are referenced by GitHub developers in their open source projects. We have conducted our analysis of SO references in GitHub projects by mapping two very large data sets from SOTorrent and GHTorrent. Our preliminary findings demonstrate that GitHub developers do reference SO discussions in their code and thus allowing us to study such phenomenon. We found that developers are more likely to reference programming language-related discussions that match the language of their code. We also observed that 79% of posts with code snippets evolve over time. Our further work will focus on better understanding the evolution of SO code snippets as well as adapted code snippets on GitHub.
Chapter 4

Methodology

In our pilot study, we have explored whether and how Stack Overflows posts are referenced in GitHub projects. However, we did not study the actual projects that reference SO discussions nor the actual reused code snippets. To understand the evolution of code in both platforms, we have to analyze two types of evolution patterns and data sets, one representing Stack Overflow (via SOTorrent) and the other one reflecting GitHub. In the pilot study, the preliminary analysis was performed using two datasets: GHTorrent and SOTorrent. Both data sets were also combined to infer the team size of GitHub projects that refer to more than 24 Stack Overflow posts in their code. Although GHTorrent is suitable for characterizing the projects that are reusing Stack Overflow snippets, it is not sufficient for further in-depth analysis for the following reason. GHTorrent consists of the events from GitHub and metadata related to those events (e.g., commit), but code changes (as code diffs) related to those events are not present in the data set. For the analysis of reused code snippet evolution within the GitHub space, events related to code snippets need to be identified using code diff. Therefore, our main effort is related to extracting the history of changes from the commit history of the code snippets that are adapted from Stack Overflow.

The lack of the dataset that can help to study the evolution history of code snippets in GitHub inspires us to mine GitHub projects and create a new dataset
that we call \textit{GHCodeSnippetHistory}. As described in Section 2.2, the majority of the previous research work aimed at analyzing SO and GH in combination and considered only hand-picked data points to make certain observations [45]. Our aim is to collect a generic sample dataset that is representative of the entire population of GitHub projects. We describe the approach of collecting a representative sample of the GitHub projects and how the history of code snippet evolution was created next.

Chapter 4 is organised as follows. Section 4.1.1 describes the important concept used to built data set for further analysis by mining GitHub. Section 4.1.2 describes the core functionality and components of a custom built tool named \textit{Code Miner} used for the data mining process. Section 4.1.3 describes the data sampling algorithm used which randomly samples the GitHub projects which are then mined with \textit{Code Miner} tool. Section 4.2 describes different statistical tests used in data analysis. Section 4.3 describes the feature extraction process. These features are used in different classification algorithms, described in Section 4.4 that aim at predicting future revisions of the adapted code snippets.

### 4.1 Creation of Dataset

To understand how GH projects are mined and converted to data points for our dataset, as well as what a code snippet data point consists of, we need to introduce two main components: the concept of \textit{Code Context} and the \textit{Code Miner} tool. \textit{Code Context} describes how an evolution of a code snippet is extracted from a given GitHub project, while \textit{Code Miner} is a tool we built as a part of this work that collects data from around 40,000 GitHub projects.
4.1.1 Code Context

To emphasize on principle component of the target data set, which we would like to build, we introduce a new term called Code Context. This term can help with understanding the narrative of the algorithms and processes described in the next section.

Since we want to focus on the evolution of code, the unit of the analysis and focus is code. When we look at the code that contains a reference to a Stack Overflow post, that we first need to find the original commit in which that reference was introduced. We call it the origin commit. Since GitHub stores the version history of all files, it is easy to determine all the commits that touched a given file. Given we have all the commits for a file and the changes they introduced, in chronological order, we can find the origin commit. To identify the origin commit, we can make use of code changes of each candidate commit. The earliest commit which has introduced SO reference is labelled as the origin commit \( C_0 \). The changes introduced in the origin commit become the code snippet of interest, and we would like to explore how this code evolves overtime. Again this code snippet is not claimed to be a clone of the corresponding SO code snippet (if exits) but assumed to be the adapted (i.e., reused) code snippet.

Suppose that a changeset \( \delta_0 \), which is known to be part of the origin commit \( C_0 \), and touching file \( f \), i.e., \( \text{Foo.py} \), which is known to have a reference to SO post, transforms the revision \( r_1 = 1.17 \) of \( \text{Foo.py} \) into \( r_2 = 1.18 \) (see Figure [18]), i.e., \( \delta_0 \) introduces new lines to \( r_2 \) or changes and removes lines of \( r_1 \). First, we detect these lines \( L \in \delta_0 = \delta_a \) that have been added by \( \delta_0 \) in \( r_2 \). These lines comprise the first version of the reused code snippet. To locate them, we use the \textit{git show origin-commit-hash} command. In our example, we assume that lines 7, 15–22 have been added, and line 10 has been deleted. We ignore deleted lines in the origin commit, as they are of...
Figure 18: Creation of code context from commits.
no interest and not related to the reused snippet. This subset $\delta_a$, comprises a code
context ($\Delta$), i.e., set of lines added to the origin commit $\Delta = \delta_a$. From this point
onwards, we want to identify all commits that change the code lines part of this code
context.

Now consider a revision $r_j$ to $f$, i.e., Foo.py. Let $C_j$ be the commit that upgrades
file version from $r_{j-1}$ to $r_j$. Let $\delta_j$ be the changeset of $C_j$, and $\delta_d \subseteq \delta_j$ is set of deleted
lines, and $\delta_a$ be added lines in $C_j$. If, a code line, $L \in \delta_d$, i.e., line is part of changeset,
and $L \in \Delta$, i.e., line is part of code context as well, then $C_j$ is modifying the current
context and added to the dataset, else revision $r_j$ is not of interest to us as it is not
modifying the reused code snippet. If $C$ modifies the context, we update the context
as $\Delta = \Delta \setminus \delta_d \cup \delta_a'$, where $\delta_a' \subseteq \delta_a$, and is collection of all lines in $\delta_a$, which are in
close proximity to the already present lines in $\Delta$, i.e., code context.

Figure 18 demonstrates this concept via an example from a GH project (file is
renamed for the sake of simplicity).

Once we have all commits from the file history, we identify $C_0$ by
looking for the commit whose changeset has introduced a reference to SO
post, i.e., the following URL: [http://stackoverflow.com/questions/27000/

All lines added in this origin commit are marked green in Figure and form *Code
Context*. Then, we identify another commit $C_1$, which happens later than $C_0$ but
consists of the changeset, which touches some of the lines added to the origin and
is present in code context. This commit is considered to be part of the snippet
evolution. Code context is now updated by removing deleted lines and adding lines
that were introduced close to already existing lines. To be specific, any line which is
newly added but is at max two lines away from code lines which are already part of
the context. The reason for choosing this magic distance (i.e., two lines) is because
each modified line in *git diff* is represented as deletion of old lines and the addition
of a new line, i.e., one line is added, and one line is deleted. Thus, to identify the boundary modification and appending of lines, we use the distance criteria of two lines. We continue to iterate through all the commits of the file that are present in the commit-graph of the file that creates different revisions of that file. From these commits, we select a subset of commits that modify the code context, that is created with the origin commit and evolves with each modifying commit, thus capturing the evolution of a code snippet.

Figure 19: Real example, origin commit, code in Code Context.
4.1.2 Code Miner

Code Miner is a tool developed to create the desired database of GHCodeSnippetHistory, using the concept of Code Context (described in Section 4.1). This tool takes PostReferenceGH as input seed data from SOTorrent. The entire workflow of this tool is as follows.

Code Miner operates on seed data points, which can be a subset or the entire table PostReferenceGH from SOTorrent. Essentially, Code Miner clones a git project and then mines its file history. However, it involves two main challenges: 1) determining which project to clone and deriving its clone URL, 2) performing the clone operation efficiently; as explained in Section 4.1.3, we aim to mine over 40,000 SO references in GitHub projects, which essentially means cloning around 23,000 GitHub projects. Since we have a large number of projects to mine, the tool is optimized and engineered to perform mining more efficiently. The URLs to GitHub projects that reference SO posts are stored in PostReferenceGH table and look similar to the following: https://raw.githubusercontent.com/RationalAsh/freeIMU/master/debug/decode_float.py
To derive the project information from a URL, it can be split as shown in Table 5:

<table>
<thead>
<tr>
<th>Element</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>HEADER</td>
<td><a href="https://raw.githubusercontent.com">https://raw.githubusercontent.com</a></td>
</tr>
<tr>
<td>USER/REPO-NAME</td>
<td>RationalAsh/freeIMU/</td>
</tr>
<tr>
<td>BRANCH</td>
<td>master</td>
</tr>
<tr>
<td>FILE-PATH</td>
<td>debug/decode_float.py</td>
</tr>
</tbody>
</table>

Table 5: Deriving GitHub project information.

For a given project, i.e., USER/REPO-NAME, our tool aggregates all SO references to avoid multiple project clones. Then, it sends a GitHub’s REST API request to gather general information on the project like the repository size, number of files, clone URL. If the request is a success, it implies that the project is public and can be cloned without infringement of copyright. The URL of such a request has the following signature: `https://api.github.com/repos/USER/REPO-NAME`.

Making such a network call to a GitHub server allows us to determine the visibility of a repository that Code Miner would attempt to clone. Initially, we wanted to avoid cloning repositories that were above a specific size (i.e. 4GB). However, it appears that GitHub provides only the repository size that is an aggregation of all file sizes, excluding the size of `.git` sub-directory. This means that even though the repository size can be very small, cloning this repo may take time due to the large size of its `.git` directory.

One of the challenges which we have experienced while running Code Miner is that the size of the repository was not truly reflective of the actual disk size of the project repository. The actual project size, in some cases, was ten times the size of the repository we got through the API call. Since `.git` repo contains the entire history of all files, it is crucial for our mining efforts. A large size of this file is both a blessing and a curse. It creates a significant time latency while cloning the project, thus slowing down the creation of the database. On the other hand, it indicates a rich
development activity on the given repository or project. High activity on the project is desirable as we are interested in studying reused snippets that are edited over several commits and thus provide more insightful observations about code evolution.

After cloning the repository, the next step is to locate the file in the codebase with the reused code snippet. This is the file that has a URL to Stack Overflow post. The relative path to file is the same as FILE-PATH. If the file exists, the tool proceeds to mine the commit history of the file. We had around 15% failures during the cloning step and finding the file within a cloned repository. These failures were related to the repository or file being deleted from GitHub.

The tool runs \texttt{git blame} on the identified file and creates a \texttt{blame} file. \texttt{Git blame} gives the entire commit-graph of a given source file. The next step is to sort this graph of commits in chronological order; mostly, it means reversing the order of the commits as they are already sorted by increasing commit time. Using the concept of Code Context as explained in Section 4.1, a subset of commits from the commit-graph is selected since these contribute to the evolution of the reused code snippet. While parsing these commits, it is essential to consider what features and attributes of a commit should be extracted and stored in the database. We decided to include the entire metadata of the commit along with the changes in the file of interest (i.e., changes to the reused snippet). Changes in co-edited files are ignored, while the names of those files are preserved for the analysis of the co-dependency of changes.

Code Miner works in sequential order as it is a single threaded application, and the key bottleneck of the entire process was related to the network bandwidth. To speed up the process and make use of parallel processing, we created a helper tool that managed multiple instances of Code Miner. This tool helped to spawn and track multiple instances of Code Miner, as well as to troubleshoot the run time problems for specific failures.
Figure 21: Component Diagram of Mining process
4.1.3 Project Sampling

PostReferenceGH table in SOTorrent dataset has a catalogue of 6.5 million (from SOTorrent version 2019/06/21) references used in public GitHub projects that refer to 150,000 Stack Overflow posts. These references span over 440,000 GitHub projects with 1,678 different file formats identified from the file extensions. Since this table is the basis for further data mining of GitHub projects, it means that we have potentially 440,000 repositories to mine. It can be very costly for Code Miner to mine all 440,000 repositories and create a database of these repositories with the corresponding changes to the files containing the reused code snippets.

Further, mining at such a scale also requires access to ample disk storage. The current version of tools developed for GitHub mining does not scale well with the mentioned amount of projects. To mine GitHub repositories at a large scale, we would need a distributed computing infrastructure and access to dedicated storage and computing power. Such an effort would be similar to the work of Bogart et al. [46]. The design of tools for mining GitHub at a massive scale is a topic of research on its own and is out of the scope of our work.

Therefore, we need to find a way of scaling down the dataset in terms of the number of projects. However, we want to preserve the generality of our results and observations derived from the reduced (i.e., sampled) GitHub data, as well as minimize selection bias. Thus, we develop a random sampling technique, i.e., Data Sampling Algorithm. While being simple, it successfully captures the generic snapshot of available data in PostReferenceGH table.

Our technique of sampling GitHub projects is inspired by Treude et al. [47]. Similar to Treude et al., we have set the magic number to around 40,000, which is the desired number of references we want to mine.
Algorithm 1 Data Sampling Algorithm

1: procedure SampleData
2: \hspace{1em} DESIRED DATASET SIZE ← 40,000
3: \hspace{1em} while loop until data-size < DESIRED DATASET SIZE do
4: \hspace{2em} loop
5: \hspace{3em} postId ← Select a random number in range(0, 56412356)
6: \hspace{3em} if postId exits in PostReferenceGHTable then
7: \hspace{4em} count ← Count allpostId references in PostReferenceGHTable
8: \hspace{4em} totalCount += count
9: \hspace{4em} posts.append( postId )
10: \hspace{2em} for a postId in posts do
11: \hspace{3em} count ← Count all references of postId in PostReferenceGHTable
12: \hspace{3em} sampleCount ← count / counter * DESIRED DATASET SIZE
13: \hspace{3em} indexes ← Sample sampleCount unique numbers in (0, count)
14: \hspace{3em} Add all references at indexes for postId in resultant dataSet
15: \hspace{2em} Save dataSet

The details of the algorithm are presented in Algorithm 1. First, a number is sampled within a range of 0 to 56,412,356. The sampled number is taken as a postId. The explanation of the above magic number for the range of sampling is that it represents the post ID range in SOTorrent. The selected SO post is then looked up in the reference table to determine whether open source projects refer to this selected post. If the answer is yes, then the post is added to a list for further analysis.

After randomly sampling all SO posts for further analysis, the total number of references is computed. This count is to be normalized to the desired project count. For each SO post, a normalized reference count is computed from the total count, then a series of unique numbers are sampled within range of 0 and the total reference count of the post. These unique numbers are indexes of all references which are selected for mining the code history from PostReferenceGH. Finally, all such indexed references are aggregated to create a sampled version of the PostReferenceGH table.
4.1.4 Details and Schema of New Dataset

Figure 22 shows the schema of the newly created dataset. This dataset consists of two main components. The first significant part of this database is the table Entity. This table includes all metadata related to commits that reflect the evolution of code snippets and concepts borrowed from StackOverflow. Each commit is identified within a project with Commit Hash, but this is not a unique ID across all commits in this database. Since this commit hash is the same as in git history of the parent project, different projects can have commits that can have the same commit hash ID. To distinguish between such commits, FileId and RepoName are used. Along with this information, the table includes information about the author and the time of the commit. The number of lines added and deleted, which are specific to the code snippet of interest, are also computed from Context as defined in Section 4.1.
Further, to be able to trace back to the original Stack Overflow post, PostId from SOTorrent is included along with each commit information. The rest of the fields are the same as for entries in the PostReferenceGH table in SOTorrent.

PostId also acts as a foreign key to connect this derived database back to SOTorrent. Similarly, RepoName can be used as a foreign key to map back to GHTorrent to extract more information about a GitHub project.

The second main component of the dataset is the code diff document database. Both the code diff and the original code-snippet are saved in file format document. Each diff can be mapped back to Entity table using CommitHash and FileId. This code diff is not necessarily the same as code diff of the related commit, as we are only interested in diff lines that change the original code snippet as described in how Code Miner tool works (see Section 4.1).

4.2 Computation and Analysis

4.2.1 Code Miner Metrics

Algorithm 1 resulted in 39,705 Stack Overflow–GitHub references were sampled from the PostReferenceGH table of SOTorrent. These projects were mined with the Code Miner tool. The Workflow Manager tool was used to spawn and manage multiple copies of the Code Miner tool. Both tools were deployed on Google Compute Cloud for about one week, on five VMs, with one instance of Workflow Manager on each VM. Further, each Workflow Manager spawned five instances of Code Miner tool resulting in a total of 25 instances of Code Miner tool. Out of 39,705 sampled references, the tool failed on 5,940 references. These failures included a failure to clone the repository and a failure caused by the fact that the source file containing SO post reference was deleted from the project. Out of 25,820 GH projects, the tool failed to clone 2,920
projects (the repository was no longer available or deleted) and failed to locate 1,679
source files (i.e., the deleted or renamed files) in the cloned projects, both amounting
to 5,940 failures. Clone failures were due to the inability of the tool to find the clone
URL. The reason for this is either the changed visibility of the project or the deletion
of the repository. At the end of this mining process, the dataset included merely
173,000 commits and their code changes about a total of 33,765 SO references in
22,900 GH projects.
4.2.2 Gini Index

The Gini index \cite{48} or Gini coefficient is a statistical measure of distribution developed by the Italian statistician Corrado Gini in 1912. It is commonly used in the field of Economics to gauge wealth distribution, mostly income distribution among a population. Gini coefficient ranges from 0 to 1, with 0 representing perfect equality, while 1 represents perfect inequality. In terms of wealth distribution, the Gini coefficient close to 1 means that all wealth is more concentrated around a few individuals. Gini index is easy to understand with the aid of graphical representation of the Lorenz curve. Lorenz curve shows income (or wealth) distribution by plotting the population percentile by income on the x-axis and cumulative income on the y-axis. The Gini coefficient is equal to the area below the line of perfect equality (0.5 by definition) minus the area below the Lorenz curve, divided by the area below the line of perfect equality. In other words, it is double the area between the Lorenz curve and the line of perfect equality.
4.2.3 Shapiro-Wilk Test

The Shapiro-Wilk test \[49\], proposed by S. S. Sharpio and M. B. Wilk in 1965 computes a W statistic that tests whether a given sample $x_1, x_2, \ldots, x_n$ comes from a normal distribution. Small values of $W$ are evidence of departure from normality. This test is typically used in comparison studies with other goodness of fit tests.

The $W$ statistic is calculated as follows:

$$W = \frac{\left( \sum_{i=1}^{n} a_i x_{(i)} \right)^2}{\sum_{i=1}^{n} (x_i - \bar{x})^2}$$  \hspace{1cm} (1)$$

, where the $x_{(i)}$ are the ordered sample values ($x_{(1)}$ is the smallest) and the $a_i$ are constants generated from the means, variances and covariances of the order statistics of a sample of size $n$ from a normal distribution.

4.2.4 T-test

The t-test \[50\] is a parametric statistic that helps to determine how significant is the difference between two groups. In other words, it helps to determine if the differences, measured in means/averages, could have happened by chance between groups. The t-test computes a t-statistic and p-value. A p-value is a probability that the results from the sample data occurred by chance, ranging from 0 to 1. A low p-value indicates that data does not occur by chance. For example, a p-value of 0.01 means there is only a 1% probability that the results from an experiment happened by chance. In most cases, a p-value of 0.05 (5%) is accepted to mean the data is valid. A t-test assumes that the data follows a normal distribution, hence called parametric data.

The normality assumption can be checked with the Shapiro-Wilk test. In this work, we use the t-test to determine the relationship between edit/change events on Stack Overflow and GitHub platforms.
4.2.5 Time Series Analysis

Our RQ 5.1 and RQ 5.2 represent the macroscopic view of code evolution and revision activities on Stack Overflow and GitHub. The microscopic view takes a temporal aspect into account by considering code revisions on GitHub through committing and, on Stack Overflow, through edit activities, as time series. These microscopic views can answer the dependency of revisions on both platforms. Thus, to study the interaction between revision activities on both platforms, we follow the approach suggested by Xuan et al. [51]. As per their approach, using the original time series of revision activities as the control group, several simulated activities are produced, and then simulated series is compared with the original control group, and the statistical difference is determined. The entire process to generate simulated series from the original control group is described below, and this methodology is used with RQ 5.4 to determine the dependence between revisions on both platforms.

Consider the timeline of GitHub and Stack Overflow edit activities of a particular pair of a post and its reused code snippet on GitHub as shown in Figure 25 (a) and (b). Let $\mathcal{A}$ and $\mathcal{B}$ be two activities we would like to compare (e.g., $E$ for edit of Stack Overflow post and $R$ for revision of the reused code snippet). For every event $E_i$ of $\mathcal{A}$ we measure the impact latency $\epsilon_i$ as the difference between the earliest event of $R \in \mathcal{B}$ following $E_i$ (see Figure 25 (a)). The sequence $\epsilon$ characterises the relationship between $\mathcal{A}$ and $\mathcal{B}$. Next, to study whether the sequence of $\mathcal{B}$ events, i.e., revisions on GitHub ($R_i$), for a particular edit on Stack Overflow $E_i$ could have occurred by chance, we create $m$ random permutations of $\mathcal{B}$ events ($B_1, \ldots, B_m$). While reshuffling, we make sure that the time difference or “idling period” between two consecutive revisions ($R_i, R_{i+1}$) of the reused code snippets remains the same, while the order of the “idling periods” is randomised (as shown in Figure 25 (c)).
Let $\epsilon_1, \ldots, \epsilon_m$ be series of impact latencies corresponding to $B_1, \ldots, B_m$ (as shown in Figure 25 (d)).

![Diagram showing steps for generating a simulated time-series of GitHub commit activities.](image)

Figure 25: The steps for generating a simulated time-series of GitHub commit activities.

Finally, we aggregate all the impact sequences such as $\epsilon$ for all such pairs in dataset into $E^B$. Then, we compare $E_B$ with each one of $E^{B_1}, E^{B_2}, \ldots, E^{B_m}$, where $E_B$ is referred as control sample while others, $E^{B_i}, i \in [1, 10]$ as simulated sample. Finally we compare each simulated sample with control sample using t-tests. The results should be as follows:

1. If $A$ and $B$ are independent from each other, $E^B$ will be statistically indistinguishable from its simulated counterparts.

2. If $A$ and $B$ are correlated in some manner, $E^B$ will be statistically longer or shorter than its simulated counterparts, indicating acceleration and deceleration.
We use this method for answering our RQ:[5.4] related to the dependence of GitHub revisions on Stack Overflow post edits.

4.3 Feature Extraction

Feature extraction is a process of preprocessing data and building derived values, i.e., features that summarize the data well, are non-redundant, informative and facilitate in model construction by reducing training times, and thus simplifying the models with better generalization with no overfitting. Now we describe various features extracted from the evolution history of reused code snippets.

4.3.1 Commit message sentiment

Each commit includes some sort of message from the author of the commit. This message can be empty in some cases, but typically involves some textual information about the nature of the commit. Further, if the data point in consideration is a merge commit, we may have a compound commit message which includes messages from multiple commits that are being merged as part of one merge commit. Our observation from RQ5 of the preliminary study that the sentiment of a comment is correlated with post edits, inspires us to also consider the sentiment of the commit message and determine whether it is correlated with the evolution of code snippet. Thus, we use vaderSentiment library to compute the sentiment score of each text message and use it as part of the feature space. Vader Sentiment library is a rule-based model for sentiment analysis of social media text by Hutto et al. [52]. It computes four scores, i.e., 'pos', 'compound', 'neu' and 'neg', representing positive, compound, neutral and negative sentiment in text.
4.3.2 Derived features

Another important feature that can be derived from a commit message is that whether the commit is coming from a pull request or not. For example, consider the following commit message:

```
Add proxy-bypass option to prevent using proxy depending on rule (#2187).
```

In the above commit message, \#2187 represents a pull request ID. One can parse the commit message text and determine whether it has a reference to a pull request.

Further, we can consider how many different developers have contributed so far to one code snippet by identifying the unique commit authors who have touched that code. Also, we can check whether the current code snippet is being changed by the developer who was the original author of the code. We can do so by comparing the email ID of the current author to the email ID of the original contributor who introduced the reused code snippet into the GitHub project. Another metric can be the time since the last commit.

4.3.3 Affinity

Consider the feature space $f_i$ of one commit $C_i$ after all the feature transformations. Then, the feature space of this commit is further extended by concatenating features of its previous commit $C_{i-1}$. This idea is based on affinity used in anomaly detection algorithms, where all the features of a data point are transformed into relative features with respect to every other data point in the training set $\mathcal{D}$. But instead of comparing features of a commit with respect to every commit, we just append the last commit with current commit and let model learn the relative relationship between features. In mathematical terms, this would mean the following:

$$F_i = f_i \cup f_{i-1}.$$  \hspace{1cm} (2)
If there exists no prior commit, we use a vector of $O$’s in place of $f_{i-1}$.

All data points are transformed this way, and finally we use $F_i$ where $i \in |D|$, for input to classification algorithm. Table 6 lists all the features in a data point $f_i$.

<table>
<thead>
<tr>
<th>Feature</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Lines added</td>
<td>Numeric</td>
<td>Lines of code added to a commit</td>
</tr>
<tr>
<td>Number of Lines deleted</td>
<td>Numeric</td>
<td>Lines of code deleted from a commit</td>
</tr>
<tr>
<td>Revision number</td>
<td>Numeric</td>
<td>Number of code revisions</td>
</tr>
<tr>
<td>Is original author</td>
<td>Boolean</td>
<td>Revision author is same as original author</td>
</tr>
<tr>
<td>Number of contributors</td>
<td>Numeric</td>
<td>Number of developers involved in code changes</td>
</tr>
<tr>
<td>Time since last revision</td>
<td>Time (in minutes)</td>
<td>Time difference between revisions</td>
</tr>
<tr>
<td>Source post type</td>
<td>Categorical</td>
<td>Referred SO post type (question or answer)</td>
</tr>
<tr>
<td>Change type</td>
<td>Categorical</td>
<td>Addition, Deletion, Modification or Rename</td>
</tr>
<tr>
<td>Language</td>
<td>Categorical</td>
<td>Java, Python, JS or PHP</td>
</tr>
<tr>
<td>Positive score</td>
<td>Numeric</td>
<td>Positive sentiment score of commit message</td>
</tr>
<tr>
<td>Negative score</td>
<td>Numeric</td>
<td>Negative sentiment score of commit message</td>
</tr>
<tr>
<td>Neutral score</td>
<td>Numeric</td>
<td>Neutral sentiment score of commit message</td>
</tr>
<tr>
<td>PR commit</td>
<td>Boolean</td>
<td>Is commit a part of a PR request</td>
</tr>
</tbody>
</table>

Table 6: Features of a training point $f_i$.

4.4 Modelling

To predict the further evolution of reused code snippets on GitHub projects, we used five different classification algorithms. We take all code snippets and flat map the data points at each revision time, i.e., if the code snippets have one or more revisions in the future, we mark its label as 1, else 0. For example, if a code snippet has three commits in its change history, i.e., $\{C_0, C_1, C_2\}$, then we have three data points, where the first two commits have label 1, i.e., $\{C_0, C_1\}$, and the last commit has label 0, i.e., $C_2$. The reason for this labelling is that revisions $C_0$ and $C_1$ have at least one
more revision that follows them, indicating the code snippet at the state defined by these two revisions still evolves in future, thus has a positive label.

Using this flat map process, we converted our dataset of about 34,000 references and corresponding code snippets into around 71,000 data points for classification problem (on average, each code snippet’s evolution history has two commits). Then, we use the 75-25% split for train and test data split. We trained classifier algorithms/models on normalized datasets. We performed the normalization using a min-max scalar.

Classification algorithms used in this work are the following:

1. Logistic Regression,
2. Random Forest,
3. Linear Discriminant Analysis,
4. Ada Boost Classifier, and
5. Support Vector Classifier, and

For all these models, we have used the scikit-learn \cite{scikit-learn} package for Python. We now describe each classification method in detail.

### 4.4.1 Logistic Regression

Logistic regression \cite{logistic_regression} one of the most common supervised learning techniques. Logistic regression is a classifier of an observation in one of the two or more classes. In theory, logistic regression can be binomial or multinomial. In our setting, we use binomial logistic regression. The logistic function is at the core of logistic regression.
The logistic function is an \( S \) shaped sigmoid curve. The equation of logistic function is as, with \( x \) as a real value number between \(-\infty \) to \( \infty \).

\[
f(x) = \frac{e^x}{1 + e^x}
\]

(3)

The logistic regression model provides log-odds of the probability of an event in the linear combination of independent predictors. Binary logistic regression is an expression of the likelihood of an event \( Y \), which in our case is whatever given code snippet will get another revision in future, occurring against a set of \( X = (X_1, X_2, ..., X_k) \) explanatory variables which can be discrete, continuous, or a combination.

\[
\text{logit}(Pr(Y_i = 1|X_i = x_i)) = \text{logit}(\pi_i) = \beta_0 + \beta_1 x_{i1} + ... + \beta_k x_{ik}
\]

(4)

### 4.4.2 Random Forest

Random Forest \([55]\) is a supervised learning technique used for classification and regression. Random Forest is an ensemble of many numbers of decision trees created from a randomly selected subset of training data. Random Trees are best at avoiding the overfitting of data. While splitting the node during the construction of trees, the split that is picked is the best split among a random subset of the features. Random Forest outputs the class that is the mode of classes (classification) or mean prediction (regression) of the individual trees. Random Forest follows the divide and conquer approach to increase performance.

### 4.4.3 AdaBoost

Boosting \([56]\) is a technique in which simpler classifiers are combined to create a learning classifier. The main idea of boosting is to take a simple or weak classifier, i.e., any classifier that will do at least slightly better than chance and uses it to build
a much better classifier, thereby boosting the performance of the weak classification algorithm. This boosting is done by averaging the outputs of a collection of weak classifiers. The most popular boosting algorithm is AdaBoost. Boosting is a specific example of a general class of learning algorithms called ensemble methods, which attempt to build better learning algorithms by combining multiple simpler algorithms.

The basic idea of AdaBoost is to iteratively re-weight the training examples based on the outputs of some weak learners. Misclassified training examples are given in consecutive iterations, while the weightage of correctly classified examples is reduced, thus forcing classifiers to focus more on incorrectly classified examples. Final resultant classifier thus assigns weightage to each component ("weak classifiers") according to how that component performed during each iteration. Lower the training error, larger the weightage for that component classifier. AdaBoost is known for performing better than the state-of-the-art classification algorithms.

4.4.4 Linear Discriminant Analysis

Logistic regression involves finding the likelihood of a response given predictors, using a logistic function. Mathematically, logistic regression finds $Pr(Y = k|X = x)$, which is conditional distribution of response given predictors. LDA or linear discriminant analysis \cite{54}, on the other hand, uses a less direct approach and uses Bayes theorem to find this conditional probability by flipping distributions of predictors modelled separately in each response class. For this LDA assumes that predictors $X$ are approximately normally distributed in each class. If $\pi_k$ is the prior probability of response class $k$, and $f_k(x) = Pr(X = x|Y = k)$ denotes the density function of $X$ in $k$th class, then using Bayes theorem:

$$Pr(Y = k|X = x) = \frac{\pi_k f_k(x)}{\sum_{l=1}^{K} \pi_l f_l(x)}$$  (5)
Pr(Y = k|X = x) is called posterior probability, which is the basis for classification in LDA classifier.

### 4.4.5 Support Vector Machines

Support Vector Classifier (SVC), an implementation of Support Vector Machines [57] in sklearn [53], is a more flexible version of the Maximal Margin classifier, which can only work if there exists a hyperplane separating two classes in a classification problem. Maximal Margin Classifier thus has zero training error. If such hyperplane does not exist, the Maximal Margin Classifier is overly sensitive to individual data points in training data and will not be able to find acceptable separating boundary. To avoid this problem, SVC or sometimes called soft margin classifier sacrifices some training examples and achieves a largest possible margin of a hyperplane separating both classes from the nearest data points.

SVC classifies a test observation based on which side of hyperplane it lies. Mathematically, SVC classification problem can be formulated as optimization problem as follows:

$$\max_{(\beta_0, \beta_1, ..., \beta_p, \epsilon_1, ..., \epsilon_n, M)} M,$$

subject to $\sum_{j=1}^{p} \beta_j^2 = 1$ and,

$$y_i(\beta_0 + \beta_1 x_{i1} + ... + \beta_p x_{ip}) \geq M (1 - \epsilon_i),$$

$$\epsilon_i \geq 0, \sum_{i=1}^{n} \epsilon_i \leq C,$$

where $C$ is non-negative tuning parameter, $M$ is width of margin, the quantity which classifier want to maximize, and $\epsilon_1, ... , \epsilon_n$ are slack variables which allows individual training examples to be wrong side of separating hyperplane.
Multinomial Naive Bayes Classification is a supervised probabilistic learning method. It is one of the two classic naive Bayes variants used in text classification problems. The distribution is characterized by a vector of parameters $\theta_c = (\theta_{c1}, \ldots, \theta_{cp})$ for each class $c$, where $p$ is the number of features and $\theta_{ci}$ is the probability $P(x_i|c)$ of feature $i$ appearing in a sample belonging to class $c$.

Parameter $\theta_c$ is estimated with MLE (Maximum Likelihood) but with smoothed version, i.e., relative frequency counting.

$$\hat{\theta}_{ci} = \frac{N_{ci} + \alpha}{N_c + \alpha n},$$  \hspace{1cm} (9)

where $N_{ci} = \sum_{x_i \in D} x_i$ is the number of times feature $i$ appears in a sample of class $c$ in the training set $D$, and $N_c = \sum_{i=1}^{n} N_{ci}$ is the total count of all features for class $c$.

The smoothing priors $\alpha \geq 0$ accounts for features not present in the learning samples and prevents zero probabilities in further computations. Setting $\alpha = 1$ is called Laplace smoothing, while $\alpha < 1$ is called Lidstone smoothing.

Model Evaluation

After mining evolution histories of posts on SO and their adapted code snippets on GitHub, we create models to predict future revisions of reused code snippets. The modelling component of data mining includes model development, feature extraction, and model evaluation. In this section, we describe all major components of the modelling process used in this research work. We now describe various metrics that we use for evaluating various models.
4.5.1 Confusion Matrix

A domain expert or source of truth helps to label the data points for a supervised machine learning algorithm. Similarly, model classifier labels data by predicting the label. Domain expert and classifier may not always agree with each other. Comparing the classification output (positive, negative) with labels (true, false) is necessary for the evaluation of the predictive power of a classification model. Figure 26 shows a confusion matrix. The confusion matrix shows four possible outcomes when we compare the label of the real-world observation given by the expert or source of truth to the classification of the corresponding data point given by the classification algorithm. The possible outcomes (and the four corresponding sets) are (1) hit \((TP)\), (2) false-alarm \((FP)\), (3) miss \((FN)\), and (4) correct-reject \((TN)\).

If the expert/source of truth labels an observation as “True” and the algorithm classifies the corresponding data point as a “Positive”, then the outcome is a “hit”. In this case, the expert and the algorithm agree with each other. The hit square in Figure 26 is shown in green colour to illustrate the agreement between the truth and the algorithm. Similarly, when truth and algorithm agree on a data point as “False” and “Negative” respectively, it is called correct reject or “True Negative”. The corresponding square is coloured blue in the Figure 26. The squares corresponding to “false alarm” and “miss” are located in the top right and bottom left. The two squares are coloured orange and red to indicate disagreement.

A confusion matrix helps to compute two types of errors such as \textit{Type-I error} or \textit{True Positive Rate} and \textit{Type-II error} or \textit{False Positive Rate}. Both errors are computed as follows:

\begin{align*}
\text{True Positive rate} &= \frac{TP}{T} \quad (10) \\
\text{False Positive rate} &= \frac{FP}{F} \quad (11)
\end{align*}
Figure 26: Confusion matrix showing four possible outcomes.

where $T$ is total number of true labeled data points in test data, and $F$ is number of data points marked by domain expert.

4.5.2 ROC Curve

Figure 27: ROC curve plot.
The ROC curve is a popular graphic for simultaneously displaying the Type-I and Type-II types of errors for all possible classification thresholds. The overall performance of a classifier, summarized over all possible thresholds, is given by the area under the curve (AUC). An ideal ROC would be hugging the top left corner so that the larger the AUC, the better the classifier. ROC curves are useful for comparing different classification models. On the ROC curve plot, a linear line is drawn, implying a ROC curve where false positive rates are equal to that of true positive rate. This linear line is to identify an AUC with a value of 0.5. In binary classification, this value or ROC line indicates a random chance classifier, i.e., classifier that randomly selects one from two possible outputs for the given data point. Figure 27 illustrates the ROC curve plot.
Chapter 5

Results

In this chapter, we present the results of our analysis and provide answers to the research questions related to the code snippet evolution analysis, as stated in Section 1.4. Sections 5.1, Sections 5.2 and Sections 5.3 present the results about the evolution of code snippets on Stack Overflow and GitHub, in particular, on how much, how and how often code snippets change over time, as well as who is driving their evolution. Section 5.4 describes the findings of the co-change of code snippets across two platforms. And finally, Section 5.5 reports the results of applying machine learning modelling to predict the likelihood of change in the reused code snippets.

We used Code Miner to extract the data from Stack Overflow and GitHub and build our dataset that captures the evolution of reused code snippets. The tool mines code snippets and creates a code snippet evolution history from project history. The history consists of the commit meta and the code diff. Summary of the dataset is reported in Table 7.

<table>
<thead>
<tr>
<th># Projects</th>
<th># References</th>
<th># Commits</th>
</tr>
</thead>
<tbody>
<tr>
<td>22,900</td>
<td>4,634</td>
<td>73,322</td>
</tr>
</tbody>
</table>

Table 7: Data set summary.
The number of projects listed here is less than the original target set in the sampling algorithm, i.e., 40,000. The reason for this discrepancy is that either those sampled projects were deleted or the file containing a SO link was deleted, for more details refer to Section 4.1.3.

In this chapter, code snippets analyzed as a whole, independent of their corresponding SO posts, as a parallel to Chapter 3. The co-analysis of evolution on both platforms is performed in Section 5.4.

5.1 RQ1: How much do code snippets evolve on GitHub and Stack Overflow?

To visualize how much code snippets evolve, we concentrated on the number of revisions per snippet on both platforms, i.e., Stack Overflow and GitHub. Each change in a reused code snippet on GitHub is referred to as a revision and as a edit on Stack Overflow.

The number of revisions on GitHub can be translated to the number of commits per snippet. Table 8 shows the number of commits per reused code snippet in GitHub projects.

<table>
<thead>
<tr>
<th>Metric (# revisions)</th>
<th>All Code Snippets</th>
<th>Changed Code Snippets</th>
</tr>
</thead>
<tbody>
<tr>
<td>Min.</td>
<td>1.00</td>
<td>2.00</td>
</tr>
<tr>
<td>1st Qu.</td>
<td>1.00</td>
<td>2.00</td>
</tr>
<tr>
<td>Median</td>
<td>1.00</td>
<td>3.00</td>
</tr>
<tr>
<td>Mean</td>
<td>2.2</td>
<td>6.54</td>
</tr>
<tr>
<td>3rd Qu.</td>
<td>1.00</td>
<td>7.0</td>
</tr>
<tr>
<td>Max.</td>
<td>507.00</td>
<td>187.0</td>
</tr>
</tbody>
</table>

Table 8: Number of revisions per reused code snippet in GH.
As one can see from both Figure 28 and Table 8 (Column All Code Snippets), the majority of reused code snippets does not get revised over time. We call them \textit{unchanged code snippets}. That is, these code snippets had only one commit in the
project history, the commit in which they were introduced to the GitHub project. These *unchanged code snippets* represent 78.8% (26,330 in total) of the total reused code snippets. While 79% of SO posts had at least one edit (when considering the change percentage of SO posts which have code snippets as discussed in Chapter 3), 78.8% of the reused code snippets in GH have no revision, thus indicating that SO post code snippets undergo more revisions than their GH counterparts.

To better understand the evolution of reused code snippets, it makes sense to segregate these GitHub code snippets into two categories:

1. **Unchanged code snippets**: Code snippets that are never updated after being introduced to a GH project.

2. **Changed code snippets**: Code snippets that are revised at least once after their inception.

Figure 29 shows the distribution of *changed code snippets* and their SO references (5% outliers are being removed). Table 8 (Column *Changed Code Snippets*) presents the summary of Figure 29.

<table>
<thead>
<tr>
<th>Metric</th>
<th>Number of edits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Min.</td>
<td>1.00</td>
</tr>
<tr>
<td>1st Qu.</td>
<td>2.00</td>
</tr>
<tr>
<td>Median</td>
<td>3.00</td>
</tr>
<tr>
<td>Mean</td>
<td>4.61</td>
</tr>
<tr>
<td>3rd Qu.</td>
<td>5.00</td>
</tr>
<tr>
<td>Max.</td>
<td>753.00</td>
</tr>
</tbody>
</table>

Table 9: Number of edits per SO code snippet.

Figure 30 shows the distribution of edits of code snippets on Stack Overflow, while Table 9 summarizes the distribution. Unlike adopted GitHub code snippets, the percentage of unedited posts is less than corresponding adapted code snippets on
Figure 29: Revisions in *changed code snippets* (2+ revisions) on GitHub.

Figure 30: Revisions in posts on Stack Overflow.
GitHub. On Stack Overflow, 23.72% of code snippets undergo no edits after their creation, while on GitHub, 78.8% of code snippets remain unchanged once introduced to the codebase.

| Answer to RQ1: On average, reused code snippets in GitHub undergo fewer revisions compared to their counterparts on Stack Overflow, i.e., 78.8% of the reused code snippets remain unchanged. |

5.2 RQ2: How often do reused code snippets evolve in GitHub and Stack Overflow?

To define how code snippets evolve, we look at the extent of the change in each revision and the change type of the reused code snippets and SO code snippets. By the extent, we mean the size of the change in LOCs. Table 10 and Table 11 show summaries of multiple revisions for GitHub code snippets and Stack Overflow code snippets, respectively, showing the median time difference between revisions and the change size. In these tables, code snippets are categorized by their revision/edit count represented by the column “Revision #” and “Edit #” for GH and SO, respectively. The 0th edit/revision type means the creation/introduction of the code snippet on the corresponding platform. Any further revision after 7th revision is merged in 7+ category, for both platforms. Further, we report the change type as well. This change type is specific to each platform. For GitHub, the change type is either ADD, DELETE, MODIFY, or RENAME depending on whether the code lines were added, deleted or modified, or file being renamed for the adapted code snippet in the corresponding revision. Similarly, for Stack Overflow, change type is either TEXT or CODE, indicating whether the code snippet or the text block of the post was revised during the edit. Column “Major Change Type” indicates the change type category for the majority of code snippets at a particular revision number, while “Secondary
<table>
<thead>
<tr>
<th>Rev. #</th>
<th>Major Change Type</th>
<th>Second Major Change Type</th>
<th>Time (in H,D,Y)</th>
<th>Change Size [LOC]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>ADD</td>
<td>MODIFY</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>MODIFIED</td>
<td>DELETE</td>
<td>&lt; 1H</td>
<td>5Y</td>
</tr>
<tr>
<td>2</td>
<td>MODIFIED</td>
<td>RENAME</td>
<td>&lt; 1H</td>
<td>~ 4.7Y</td>
</tr>
<tr>
<td>3</td>
<td>MODIFIED</td>
<td>DELETE</td>
<td>&lt; 1H</td>
<td>~ 5.8Y</td>
</tr>
<tr>
<td>4</td>
<td>MODIFIED</td>
<td>DELETE</td>
<td>&lt; 1H</td>
<td>~ 5.6Y</td>
</tr>
<tr>
<td>5</td>
<td>MODIFIED</td>
<td>DELETE</td>
<td>&lt; 1H</td>
<td>~ 4.3Y</td>
</tr>
<tr>
<td>6</td>
<td>MODIFIED</td>
<td>RENAME</td>
<td>&lt; 1H</td>
<td>~ 3.7Y</td>
</tr>
<tr>
<td>7+</td>
<td>MODIFIED</td>
<td>DELETE</td>
<td>&lt; 1H</td>
<td>~ 4.9Y</td>
</tr>
</tbody>
</table>

Table 10: Revision characteristics for reused GH code snippets.

<table>
<thead>
<tr>
<th>Edit #</th>
<th>Major Change Type</th>
<th>Second Major Change Type</th>
<th>Time (in H,D,Y)</th>
<th>Change Size</th>
<th>Text</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>TEXT &amp; CODE</td>
<td>TEXT</td>
<td>-</td>
<td>-</td>
<td>3</td>
<td>8</td>
</tr>
<tr>
<td>1</td>
<td>TEXT</td>
<td>TEXT &amp; CODE</td>
<td>&lt; 1H</td>
<td>9.8Y</td>
<td>3.5D</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>TEXT</td>
<td>CODE</td>
<td>&lt; 1H</td>
<td>~ 10Y</td>
<td>143D</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>TEXT</td>
<td>TEXT &amp; CODE</td>
<td>&lt; 1H</td>
<td>~ 9.1Y</td>
<td>~ 149D</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>TEXT</td>
<td>CODE</td>
<td>&lt; 1H</td>
<td>~ 10.3Y</td>
<td>111D</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>TEXT</td>
<td>TEXT &amp; CODE</td>
<td>&lt; 1H</td>
<td>~ 8.3Y</td>
<td>~ 144D</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>TEXT</td>
<td>CODE</td>
<td>&lt; 1H</td>
<td>~ 7.3Y</td>
<td>~ 110D</td>
<td>2</td>
</tr>
<tr>
<td>7+</td>
<td>TEXT</td>
<td>CODE</td>
<td>&lt; 1H</td>
<td>~ 6.6Y</td>
<td>~ 6.2D</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 11: Revision characteristics for Stack Overflow code snippets.

*Change Type* denotes the runner up change type for all code snippets. Table 10 shows that the vast majority of code snippets are of MODIFIED type over multiple revision, with the change size of 24 LOCs, on average. Similarly, from Table 11, we can observe that the majority of all edits to Stack Overflow posts have the change size of around 2 lines for the text block and around 1 line for the code block. As a note, SO posts are defined as either Text and Code blocks in SOTorrent, as described in Section 2.3.1.
To investigate how often code snippets are changed on each platform, we look at the time difference between two consecutive revisions/edits. To show the timeline range over which the consecutive revisions can happen, we report the minimum (in Hours $H$) and maximum time (in Years $Y$) difference along with the median time (in Days $D$). Figure 31 shows the distribution of time difference between subsequent revisions on both platforms, for all snippets. These distributions are again classified by their revision numbers. For example, in Figure 31, 2\textsuperscript{nd} revision, indicated by number 2 on the y-axis and its corresponding density curve, shows the distribution of time difference between second and first revisions for all code snippets.

As shown in Figure 31, the revision time difference distribution follows a unique pattern on a platform across multiple revisions, i.e., time difference distribution curves representing each of the seven revisions on GitHub are of a similar pattern while the nature of these revision curves is very different across the platforms. We observe that GitHub distributions resemble bimodal distributions, while further analysis is needed to confirm this.

To make sense of how the majority of snippets are distributed within a curve, in terms of their time difference, we use Gini Index. Table 12 reports Gini coefficients for each of the curves shown in Figure 31. A Gini coefficient of value close to 0 means that the edit time difference for all the posts is uniformly distributed. Gini coefficient’s value close to one means that the edit time difference is very high for a few posts. For example, the curve 1 for GitHub with a Gini coefficient of 0.68 indicates that the revision time difference is more uniformly distributed compared to the curve 7+.

Combining the results from Table 12 and Figure 31, we notice that distribution of time between consecutive edits is relatively more uniform for Stack Overflow than for GitHub revisions. For GitHub, initial revisions happen closer to their creation,
Figure 31: Density plot of revision distributions over time on GH (in blue) and SO (in orange).
<table>
<thead>
<tr>
<th>Revision Number</th>
<th>GitHub</th>
<th>Stack Overflow</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.684</td>
<td>0.745</td>
</tr>
<tr>
<td>2</td>
<td>0.779</td>
<td>0.684</td>
</tr>
<tr>
<td>3</td>
<td>0.794</td>
<td>0.663</td>
</tr>
<tr>
<td>4</td>
<td>0.799</td>
<td>0.675</td>
</tr>
<tr>
<td>5</td>
<td>0.802</td>
<td>0.657</td>
</tr>
<tr>
<td>6</td>
<td>0.795</td>
<td>0.660</td>
</tr>
<tr>
<td>7+</td>
<td>0.812</td>
<td>0.787</td>
</tr>
</tbody>
</table>

Table 12: Gini coefficients for edit time distributions on both platforms.

i.e., within one hour, while the more code snippet is being changed, the more time it takes (likely years) between the revisions.

Answer to RQ2: The time difference between edits is more uniformly distributed for SO code snippets compared to the revision time for reused code snippets on GitHub. SO posts and code snippets undergo minor changes of 2 lines to the textual description of the answers, while the change size of GH code snippets is 24 LOCs, on average.

5.3 RQ3: Who is driving the evolution of code snippets on both platforms?

GitHub and Stack Overflow are social software development environments, i.e., a large amount of social interaction happens around code and software development. One of the dominant trends, especially on Stack Overflow, is that posts and code snippets can become the product of collective ownership as multiple authors change the content over time. To visualize who is contributing to the content changes, i.e., posts on Stack Overflow and the code snippets that are reused on GitHub, we try to categorize authors as original vs. non-original. Original authors are the users...
who created the content in the first place, e.g., created a post on Stack Overflow or committed the reused code snippet in a GitHub project. In contrast, *non-original author* is anyone but the original author.

Figure 32 shows the percentage of SO post edits by original vs. non-original author, while Figure 33 shows the portion of commits to reused code snippets in a
Figure 33: Contributors on GitHub.

particular revision by original vs. non-original author. We notice that the original authors take up most of the ownership of the content as they remain the primary drivers of the content (post or code) maintenance on both Stack Overflow and GitHub platforms.
Answer to RQ3: The author of the original code snippet remains the leading contributor to its maintenance and evolution on both platforms.

5.4 RQ4: To what extent do reused code snippets co-change on both platforms?

Having established that code snippets evolve on both platforms, i.e., Stack Overflow and GitHub, next, we wanted to know to what extent revisions might be happening in parallel, i.e., co-changing. Figure 34 shows the co-change of code snippets on both platforms, as a function of time. In this figure, the x-axis timeline of edits represents the number of days since 2008-08-05, till present. As the timeline increases, more and more posts are created on Stack Overflow, similarly more and more code snippets are reused from SO and introduced in GitHub projects. Further, we observe a lag between the GitHub curve and Stack Overflow curve representing the time differences, on average, between creation and reuse of code snippets. It takes at least 1,000 days since any SO discussion has taken place, for the code snippet to be adapted and reused in a GitHub project. Eventually, the number of SO posts created on SO becomes stable (i.e., the posts in our dataset), and their content also gets more stable as GitHub projects start reusing the content from SO. We also notice that the majority of adapted code snippets do not evolve at all.

Looking at the evolution history of all reused code snippets and their SO counterparts as time-series data, as shown in Figure 34, we observe different peaks at different timestamps. For GitHub, the peak of the green curve is at 41.2% of the height of the yellow curve at that time, denoting that 41.2% of reused code snippets have changed at that timestamp. Similarly, for Stack Overflow, the green curve is at 53.3% of the height of the yellow curve, implying that nearly half of the referred code snippets have evolved at that timestamp. Finally, blue curves on the graphs
of both platforms indicate the percentage of code snippet pairs that co-change at a particular timestamp. Since the scales of the two graphs are different, the height of the peaks seems different. However, the blue curve is the same on both graphs. The highest peak of the blue curve is at 10% of the reused code snippets. This means that at most 10% of the adapted code snippets and their SO counterparts were evolving together across two platforms at any moment of time.

Based on the results, we can say that the adapted code snippets evolve while their counterpart posts evolve on Stack Overflow because the timelines overlap. But it’s not clear whether the evolution on Stack Overflow is triggering revisions of the
Table 13: T-test results for GitHub vs. Stack Overflow activities.

<table>
<thead>
<tr>
<th>Sample</th>
<th>Q1 μ</th>
<th>Q1 p</th>
<th>Q2 μ</th>
<th>Q2 p</th>
<th>Q3 μ</th>
<th>Q3 p</th>
<th>Q4 μ</th>
<th>Q4 p</th>
</tr>
</thead>
<tbody>
<tr>
<td>Control</td>
<td>779.94</td>
<td>-</td>
<td>602.01</td>
<td>-</td>
<td>490.40</td>
<td>-</td>
<td>379.43</td>
<td>-</td>
</tr>
<tr>
<td>Simulated 1</td>
<td>779.94</td>
<td>0.999</td>
<td>602.09</td>
<td>0.997</td>
<td>496.67</td>
<td>0.718</td>
<td>381.34</td>
<td>0.878</td>
</tr>
<tr>
<td>Simulated 2</td>
<td>779.94</td>
<td>0.999</td>
<td>600.98</td>
<td>0.964</td>
<td>501.13</td>
<td>0.537</td>
<td>379.80</td>
<td>0.977</td>
</tr>
</tbody>
</table>

adapted GitHub code snippets. To determine whether there is a dependency between Stack Overflow evolution and GitHub evolution, i.e., there is a relation between the edit activities on Stack Overflow and the commit activities on GitHub, we follow the approach proposed by Xuan et al. [51] and Vasilescu et al. [26]. This approach is described in Section 4.2.5. In this approach, we compute impact latencies $\epsilon$ for each pair of SO post and adapted code snippet. This latency denotes the time difference between edit activity on Stack Overflow and revision activity on GitHub. For more details, refer to Section 4.2.5.

Further, following Vasilescu et al. [26], we split computed impact latencies $\epsilon_i$’s of each pair into quarters by the number of GitHub revisions. Final t-test results are presented in Table 13 for two simulated samples. Results for all ten simulations are presented in Appendix A.

Based on the t-test results, we conclude that reused code snippets in GitHub evolve independently of their corresponding Stack overflow code snippets. This means that Stack Overflow edits do not impact the evolution of the reused code snippets in GitHub and vice versa. This finding, while surprising, is based on the data-driven evidence.

Answer to RQ4: While having overlapping evolution timelines, reused code snippets and their corresponding SO post code snippets evolve independently of one another.
<table>
<thead>
<tr>
<th>Classifier</th>
<th>Accuracy</th>
<th>Precision</th>
<th>Recall</th>
<th>F1 Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logistic Regression</td>
<td>0.808</td>
<td>0.819</td>
<td>0.817</td>
<td>0.818</td>
</tr>
<tr>
<td>Linear Discriminant Analysis</td>
<td>0.808</td>
<td>0.817</td>
<td>0.820</td>
<td>0.818</td>
</tr>
<tr>
<td>Bagging Model</td>
<td>0.864</td>
<td>0.853</td>
<td>0.897</td>
<td>0.874</td>
</tr>
<tr>
<td><strong>Random Forest</strong></td>
<td><strong>0.867</strong></td>
<td><strong>0.860</strong></td>
<td><strong>0.891</strong></td>
<td><strong>0.876</strong></td>
</tr>
<tr>
<td>AdaBoost</td>
<td>0.846</td>
<td>0.835</td>
<td>0.882</td>
<td>0.858</td>
</tr>
<tr>
<td>SVC</td>
<td>0.814</td>
<td>0.816</td>
<td>0.836</td>
<td>0.826</td>
</tr>
</tbody>
</table>

Table 14: Evaluation metrics for different classification algorithms.

5.5 RQ5: How accurately can we predict the likelihood of a change in reused code snippets?

After gathering all the revision history, we wanted to find out how accurately one can predict future edits of reused code snippets. In RQ4, we established that there is no conclusive effect of SO post edits on GitHub code snippets. Thus the version history and characteristics of SO posts (except for the category of referred post) were not considered as features in classification algorithms to predict whether the reused code snippet will be changed or not. Construction of the feature space is described in Section 4.3 and the classification algorithms we used are described in Section 4.4.

Table 14 reports the performance metrics for each classifier on same train and test datasets. Figure 35 illustrates ROC curves for each classification algorithm.

Our results suggest that Random Forest classifier achieves the highest accuracy and recall. The simplest model, i.e., Logistic Regression, also achieves a decent accuracy and precision. To interpret which predictors are the best in predicting the likelihood of evolution, based on p-values, we use logistic regression model. Some of the most powerful predictors are **Relative change size in revision** and **Author of revision**, followed by **Change Type** and **Programming language**. Coefficients of these predictors, along with their p-values, are reported in Table 15.
Figure 35: ROC curves for each classification algorithm.
<table>
<thead>
<tr>
<th>Predictor</th>
<th>Coefficient</th>
<th>p-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Author of Revision</td>
<td>-0.054</td>
<td>&lt; 0.001</td>
</tr>
<tr>
<td>Last Author</td>
<td>-1.031</td>
<td>&lt; 0.001</td>
</tr>
<tr>
<td>Relative Change Size</td>
<td>-0.365</td>
<td>&lt; 0.001</td>
</tr>
<tr>
<td>Change Type ADD</td>
<td>-3.84</td>
<td>&lt; 0.001</td>
</tr>
<tr>
<td>Change Type DELETE</td>
<td>-2.74</td>
<td>&lt; 0.001</td>
</tr>
<tr>
<td>Change Type MODIFY</td>
<td>5.95</td>
<td>&gt; 0.1</td>
</tr>
<tr>
<td>Change Type RENAME</td>
<td>-1.68</td>
<td>&lt; 0.001</td>
</tr>
<tr>
<td>PL Java</td>
<td>1.13</td>
<td>&lt; 0.001</td>
</tr>
<tr>
<td>PL JavaScript</td>
<td>-3.43</td>
<td>&lt; 0.001</td>
</tr>
<tr>
<td>PL Python</td>
<td>1.20</td>
<td>&lt; 0.001</td>
</tr>
<tr>
<td>PL PHP</td>
<td>-1.37</td>
<td>&lt; 0.001</td>
</tr>
<tr>
<td>Post Type Q</td>
<td>-0.14</td>
<td>&lt; 0.001</td>
</tr>
<tr>
<td>Post Type A</td>
<td>-2.43</td>
<td>&lt; 0.001</td>
</tr>
<tr>
<td>Sentiment Score</td>
<td>(\sim 0.0)</td>
<td>&gt; 0.1</td>
</tr>
<tr>
<td>Time difference</td>
<td>(\sim 0.0)</td>
<td>&gt; 0.1</td>
</tr>
</tbody>
</table>

Table 15: Logistic regression coefficients and p-values.

One of the surprising observations is that the time difference between revisions, as well as the sentiment scores of commit messages, have little predictive power, unlike previously reported results [58]. Since the sentiment of the majority of commits messages is neutral, it has a little variance, unlike the sentiment of SO post comments.

Answer to RQ5: *Using features extracted from the GitHub commit meta-data, Random Forest classifier achieves the highest accuracy of 86.7%, while simple Logistic Regression model achieves 80.8% accuracy in predicting the likelihood of GitHub code snippet revisions.*
Chapter 6

Discussion

In this chapter, we present the overall findings in Section 6.1, offer several implications of this study in Section 6.2, and discuss potential threats to validity in Section 6.3.

6.1 Findings

The overarching theme of this work was to understand the evolution of code snippets in Stack Overflow and how these code snippets are reused and changed on GitHub. Thus, we tried to establish whether there is any relationship between the edits to SO post code snippets and the revisions to the adapted/reused code snippets in the GitHub project repositories. We also wanted to determine whether edit activities on SO code snippets impact the commit activities to those adapted GH code snippets.

We found that code snippets evolve on both platforms, i.e., GitHub and Stack Overflow, simultaneously and independently. The pace of evolution is different on both platforms. Initial edits and revisions on both platforms happen quite fast. However, the average time difference between later edits on Stack Overflow is higher compared to the average time between revisions on GitHub. On the contrary, the change of content in GitHub revisions is much higher than the change size of Stack Overflow edits. On average, a GitHub revision involves 24 lines of code, while on
Stack Overflow, the average change is 2 lines. Further, the majority of such changes in edits of SO posts are of TEXT type than CODE type, indicating that edits on Stack Overflow are more likely to affect the description of the answers rather than the code snippets.

Since the content on both platforms evolves independently, predictive models of future revisions on GitHub should be built using GitHub’s context and characteristics only. Features extracted from the meta-data of code snippets and their revisions are found to be strong predictors of future revisions.

Though we found that the evolution of code snippets happens on both platforms, we have not investigated why code snippets undergo multiple corrections either on Stack Overflow or GitHub. That is, what are the drivers for these revisions. In this chapter, we discuss the factor that may influence the changes on the Stack Overflow posts. Later, we also discuss potential relationships between changes on both platforms that can be further explored in future.

6.2 Implications

Our research may have a variety of implications and can further drive the progress in other research areas. We now discuss some of the implications that are targeted to the greater scientific community working with Stack Overflow or GitHub data and beyond.

6.2.1 Correctness and stability of SO knowledge

Given we have the evolution history and the usage history of a Stack Overflow discussion, we can predict further improvements of the knowledge resource. Such prediction can be instrumental in deciding whatever information can be used in other software
artifacts without any unintended consequences or not. One example would be predicting the migration of bugs from Stack Overflow to GitHub projects. As we noted earlier, comments are triggers that further improve Stack Overflow answers and discussions. Since we can detect the sentiment polarity of the comment as a disagreement or encouragement, comments are one of the key indicators of the SO post/discussion stability. Together with the usage history of posts in GitHub, SO comments can help to determine whether code snippets would undergo any changes prior to their adaption in the projects’ codebase.

Further, a similar usability score can be one of the features of the question-answer ranking algorithm for a SO discussion thread. A lot of question-answer ranking approaches, e.g., by Xu et al. [59], have focused only on the text and metadata of the answer post to determine its score. However, we believe that two essential features that are often missing in these ranking algorithms are as follows:

1. **Comment sentiment**: Comments and their sentiment and tone may indicate approval or disapproval of the given answer and can be quite insightful to the greater SO community.

2. **Edit history of code blocks**: The change history of SO code blocks can help determining whether the code is ready to be used and adapted by the users in their own code repositories.

### 6.2.2 Different acceptance criteria for SO answers

The current practice on Stack Overflow is as follows: the user posting a question is the one who selects one of the answers as the accepted answer. Most of the time, the user is looking for an answer to his/her problem as soon as possible. Thus, if the current solution to the problem meets certain satisfactory criteria, the user marks it as accepted. Therefore, acceptance of the solution is often based on its timely
delivery and availability rather than its correctness and robustness. In our analysis of GitHub projects, we have found that 54% of the references to SO posts have no accepted answer. This finding suggests that Stack Overflow should perhaps re-define the definition of the accepted answer. Perhaps, “best for now” option is good enough for users in need of immediate solution. But as the community discussion takes place and the more comments are added to the post, the answer would certainly gain better stability over time and reach better quality. Therefore, we suggest Stack Overflow to add some indicator for the answer “maturity” that reflects the evolution and the quality of the provided solution. Additionally, the author’s reputation and ratings of upvoters should also be considered while accepting answers.

### 6.2.3 A dataset for code clone detection

As a part of Code Miner, we mined GitHub projects to create a code history that stores code diffs along with the related metadata. Code, generated as a part of data mining, is mostly unexplored, yet it provides ready-to-be-used dataset for clone detection across a large number of projects. Origin commits along with the corresponding code diff become the basis for clone detection of relevant SO code snippets. Later, one can study the subsequent commits (with/without code diffs) to analyze changes of the cloned/borrowed code.

### 6.2.4 Recommendations to software developers

After analyzing the evolution of code snippets on Stack Overflow and GitHub (based on results in Table 10 and Table 11), we conclude that evolution on Stack Overflow happens at a slower speed compared to GitHub. Thus, we recommend software developers and engineers to keep an eye on the SO discussions as these posts and code snippets that are borrowed to GitHub projects do change over time. Based on the
average change time from Table 11, we suggest developers to re-visit SO discussions that they refer in their code every three months. Such improved practice would allow developers to keep their awareness of any critical changes to the code they adapt from SO.

Further, while adapting a code snippet, we advise developers to study the edit history of the post, as higher activity on the answer post may be an indicator of a better content quality. Also, developers may need to consider the entire thread of the discussions rather than just the accepted answer. Code snippets adapted from the overall SO discussion thread are less likely to undergo further revisions. The number of comments on a post is another indicator to watch for. Our sentiment analysis results demonstrate that SO posts with higher number of comments carry higher degree of agreement on the content.

### 6.3 Threats to Validity

Our study is subject to several threats to validity which we address next.

1. To recreate a history of commits of the reused code snippets, Code Miner heavily relies on the `git history` of the project. Often the commit history is modified via `git --rebase` option. In such scenarios, it is challenging to recreate original commit history. Such altered history significantly reduces the number of commits per code snippet while increasing the number of changed lines in the merged commit.

2. Similarly, if code from third party libraries and modules is borrowed in the project and the reference to SO post is present in such code, Code Miner is not able to recreate the history of such code. Currently, Code Miner does not segregate such references and code snippets from the dataset. The reason for
avoiding such separation is due to the fact that a number of files are coming from external imports, and it would require a project-specific context or even manual labelling of the data (which is subject to introducing bias) to analyze such files.

3. One particular project named cdnjs has been removed from the final list of projects from our dataset. The decision for eliminating this project was driven by its size (100GB) which was causing technical problems with disk space and speed of a cloning task.

4. Our entire analysis is based on the continuous comparison of each GitHub code snippet with its corresponding SO code snippet. If the reference points to a question post of the SO discussion thread, we have mapped the reference to the accepted answer or highly voted answer (if no accepted answer is available). This mapping is based on the assumption that the selected answer provides the best explanation of the solution that the author of the GitHub code snippet is looking for.

5. While analyzing revisions of the reused code snippets, we found some revisions when “Committer ID” is not the same as “Author ID”. The author is the original developer who wrote the code, while the committer is the person who committed code on behalf of the author [60]. This is significant in GitHub as it allows one to rewrite the history or apply patches on author’s behalf, thus encouraging true collaboration. In this work, while looking at the contributors to the evolution of code snippets, we focused on the commit authors only excluding the committers from the analysis. Therefore, if the commit history experiences a loss of the original sequence of revisions or commits, then this error propagates to our data and analysis.
Chapter 7

Conclusions

Lastly, we conclude our work with a summary of our work and contributions in Section 7.1 and then discuss future directions in Section 7.2.

7.1 Summary of Contributions

This work studies whether and how GitHub developers reference Stack Overflow posts in their open source projects and how much those reused code snippets evolve on both platforms. We have studied SO references in GitHub projects by mapping SOTorrent dataset to GHTorrent dataset and built a new dataset that provides a mapping of these references to the revision history of the reused code snippet along with code diffs.

Our preliminary findings demonstrate that GitHub developers do reference SO discussions in their code and thus allowing us to study such a phenomenon. We found that developers are more likely to reference programming language-related discussions that match the language of their code. We also observed that 79% of posts with code snippets evolve over time. Further, 21.2% of reused snippets do evolve on GitHub space as well, some as many as 507 times. Characteristics of such evolution show that time difference between two consecutive revisions can vary from a few minutes to few
years. Finally, we have observed the ownership of the content on both platforms and found that the creator of the content plays a critical role in the evolution of code snippets.

A small percentage of referenced code snippets evolve on both platforms simultaneously, indicating a possibility of cross-platform change triggers such as edits of information on Stack Overflow may cause revisions in referred code snippets on GitHub and vice versa. This percentage of co-changing code snippets, while varying with time, is very small on the overall timeline. Nevertheless, during the earlier age (initial days of the creation and adaption) of code snippets, the number of co-changed code snippets can be significant. Our future work will focus on a better understanding of the cause of revisions on both platforms and cross-platform change triggers. Investigating triggers involves looking at discussions and other activities likes upvotes/downvotes happening around the time of an edit. Similarly, for cross-platform issues, we need to aggregate information about open/closed Issues on a project, as well as participation of the author of a commit on SO discussions. Since we have the email address of the author of the revisions on GitHub, this email can be used to map a SOTorrent user with GitHub user, using display name on Stack Overflow.

7.2 Future Work

After extracting and studying the evolution of code snippets on Stack Overflow and the evolution of reused snippets on GitHub, the next step would be to study their correlation. One of the directions is to perform code clone detection and analysis. Our SnippetGHHistory dataset, a contribution of this work, provides the change history of the reused code snippets, and SOTorrent can provide corresponding code snippets from Stack Overflow. Clone detection can help to determine how code snippets are adapted to GitHub projects and whether the change of the content on one platform
leads to the changes on another platform. As we observed, the co-change happens on both platforms, but we have not explored the dependence of evolution patterns on each other. This dependence analysis can also shed light on the behaviour of developers and their code adaption styles.

Our RQ 5.4 findings suggest that code snippets evolve independently on each platforms (e.g., SO has no impact on GH) based on the randomized simulation datasets and t-tests. But these findings are quite different from our intuition. In our future work, we would like to explore this dependence using more complex models, e.g., Markov Chain Models.

Figure 31 demonstrates that unlike Stack Overflow GitHub revision time forms a bimodal distribution. In future, we would like to explore this observation by performing a further analysis of the difference of time distributions on these platforms.

Another extension of this work would be to predict the time of the next change. Given the history of code snippet usage in GitHub projects and their revisions, a machine learning technique can predict the revision times that can be used as indicators of the code snippet stability. These indicators can be used to increase developer awareness on possible migration of software errors and bugs from Stack Overflow to GitHub.
List of References


[Online; accessed 01-March-2020].

M. Gethers, “Export: Detecting and visualizing api usages in large source code 
repositories,” in 2013 28th IEEE/ACM International Conference on Automated 


[Online; accessed 27-January-2020].

[16] Y. Yan, M. Menarini, and W. Griswold, “Mining software contracts for software 
evolution,” in 2014 IEEE International Conference on Software Maintenance 

[17] H. Gonzalez, N. Stakhanova, and A. A. Ghorbani, “Measuring code reuse in 

stack overflow and popular open source java projects,” in 2018 25th Australasian 

software evolution: A case study,” in 2010 10th International Conference on 

snippets there?,” in 2017 IEEE/ACM 14th International Conference on Mining 

code clone detection system for large scale source code,” IEEE Transactions on 


Appendix A

T-test Results

As part of RQ4, we simulated 10 samples of impact time on GitHub activities as shown in Figure 25. Table 16 lists out all results from 10 simulated samples.

<table>
<thead>
<tr>
<th>Sample</th>
<th>Q1</th>
<th>Q2</th>
<th>Q3</th>
<th>Q4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>μ</td>
<td>p</td>
<td>μ</td>
<td>p</td>
</tr>
<tr>
<td>Control</td>
<td>779.9385</td>
<td>-</td>
<td>602.01</td>
<td>-</td>
</tr>
<tr>
<td>Simulated 1</td>
<td>779.9386</td>
<td>0.99999</td>
<td>602.09</td>
<td>0.997</td>
</tr>
<tr>
<td>Simulated 2</td>
<td>779.9386</td>
<td>0.99999</td>
<td>600.98</td>
<td>0.964</td>
</tr>
<tr>
<td>Simulated 3</td>
<td>779.9386</td>
<td>0.99999</td>
<td>602.98</td>
<td>0.997</td>
</tr>
<tr>
<td>Simulated 4</td>
<td>779.9386</td>
<td>0.99999</td>
<td>600.08</td>
<td>0.964</td>
</tr>
<tr>
<td>Simulated 5</td>
<td>779.9386</td>
<td>0.99999</td>
<td>602.08</td>
<td>0.997</td>
</tr>
<tr>
<td>Simulated 6</td>
<td>779.9386</td>
<td>0.99999</td>
<td>600.98</td>
<td>0.964</td>
</tr>
<tr>
<td>Simulated 7</td>
<td>779.9386</td>
<td>0.99999</td>
<td>602.08</td>
<td>0.997</td>
</tr>
<tr>
<td>Simulated 8</td>
<td>779.9386</td>
<td>0.99999</td>
<td>600.98</td>
<td>0.967</td>
</tr>
<tr>
<td>Simulated 9</td>
<td>779.9386</td>
<td>0.99999</td>
<td>602.08</td>
<td>0.997</td>
</tr>
<tr>
<td>Simulated 10</td>
<td>779.9386</td>
<td>0.99999</td>
<td>600.98</td>
<td>0.964</td>
</tr>
</tbody>
</table>

Table 16: T-test results for GitHub vs. Stack Overflow activities.